Appl Intell
DOI 10.1007/s10489-011-0280-5

Service selection in stochastic environments: a learning-automaton based solution

Anis Yazidi · Ole-Christoffer Granmo · B. John Oommen

Abstract In this paper, we propose a novel solution to the problem of identifying services of high quality. The reported solutions to this problem have, in one way or the other, resorted to using so-called "Reputation Systems" (RSs). Although these systems can offer generic recommendations by aggregating user-provided opinions about the quality of the services under consideration, they are, understandably, prone to "ballot stuffing" and "badmouthing" in a competitive marketplace. In general, unfair ratings may degrade the trustworthiness of RSs, and additionally, changes in the quality of service, over time, can render previous ratings unreliable. As opposed to the reported solutions, in this paper, we propose to solve the problem using tools provided by Learning Automata (LA), which have proven properties capable of learning the optimal action when operating in unknown stochastic environments. Furthermore, they combine rapid and accurate convergence with low computational complexity. In addition to its computational simplicity, unlike most reported approaches, our scheme does not require prior knowledge of the *degree* of any of the above mentioned problems associated with RSs. Instead, it gradually learns the identity and characteristics of the users which provide fair ratings, and of those who provide unfair ratings, even when these are a consequence of them making unintentional mistakes.

Comprehensive empirical results show that our LA-based scheme efficiently handles any degree of unfair ratings (as long as these ratings are binary—the extension to non-binary ratings is "trivial", if we use the S-model of LA computations instead of the P-model). Furthermore, if the quality of services and/or the trustworthiness of the users change, our scheme is able to robustly track such changes over time. Finally, the scheme is ideal for decentralized processing. Accordingly, we believe that our LA-based scheme forms a promising basis for improving the performance of RSs in general.

Keywords Reputation systems · Learning automata · Stochastic optimization

The first author gratefully acknowledges the financial support of the *Ericsson Research*, Aachen, Germany, and the third author is grateful for the partial support provided by NSERC, the Natural Sciences and Engineering Research Council of Canada. A preliminary version of this paper was presented at IEA/AIE'10, the 2010 International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Cordoba, Spain, in June 2010.

A. Yazidi · O.-C. Granmo · B.J. Oommen (✉)
Department of ICT, University of Agder, Grimstad, Norway
e-mail: oommen@scs.carleton.ca

A. Yazidi
e-mail: anis.yazidi@uia.no

O.-C. Granmo
e-mail: ole.granmo@uia.no

B.J. Oommen
School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6

Published online: 17 February 2011

1 Introduction

1.1 Problem formulation

With the abundance of services available in today's world, identifying those of high quality is becoming increasingly difficult. Since this, typically, involves the comprehensive marketplace, an entire body of research has recently come to the forefront, namely, the study of so-called "Reputation Systems" (RSs). Such systems have attracted a lot of attention during the last decade in academia as well as in

the industry, because they present a, hopefully, transparent method by which the user community, within a social network, can rank the quality of the services in question. RSs have also emerged as an efficient approach to handle trust in online services, and can be used to collect information about the performance of services in the absence of direct experience.

In this paper we intend to model and study how experiences can be shared between users in a social network, where the medium of collaboration is a RS. The basic premise, of course, is that it is possible for users to expediently obtain knowledge about the nature, quality and drawbacks of specific services by considering the experiences of other users. Indeed, the information that people used to share with their friends on campus or over a cup of coffee is now being broadcasted online, and one can thus, easily and electronically, take advantage of the comments of thousands of people participating in the social network. For instance, in a social network of tourists, sharing the experiences concerning restaurants can certainly prove advantageous.

The above premise is true if the basis of the decision is accurate, up-to-date and fair. Unless a person is naive, he must accept the fact that every user may not communicate his experiences truthfully. In fact, the social network and the system itself might contain misinformed/deceptive users who provide either unfair positive ratings about a subject or service, or who unfairly submit negative ratings. Such "deceptive" agents, who may even submit their inaccurate ratings innocently, have the effect that they mislead a RS that is based on blindly aggregating users' experiences. Furthermore, when the quality of services and the nature of users change over time, the challenge is further aggravated.

From this perspective, we can state a fundamental paradox[1] about using RSs by virtue of the fact that they are prone to "ballot stuffing" and "badmouthing" in a competitive marketplace. Users who want to promote a particular product or service can flood the domain (i.e., the social network) with sympathetic votes, while those who want to get a competitive edge over a specific product or service can "badmouth" it unfairly. Thus, although these systems can offer generic recommendations by aggregating user-provided opinions, unfair ratings may degrade the trustworthiness of such systems. Additionally, changes in the quality of service, over time, can render previous ratings unreliable. In general, unfair ratings may degrade the trustworthiness of RSs, and changes in the quality of service, over time, can render previous ratings unreliable.

[1] Instead of relying purely on traditional information sources, a user can opt to take advantage of social networks in the form of RSs to get more reliable recommendations. But instead, he risks ending up with even worse reliability than what was offered with traditional information sources because misinformed/deceptive users may "contaminate" the RSs. Hence the paradox!

This problem, of separating "fair" and "unfair" agents for a specific service, is called the Agent-Type Partitioning Problem ($ATPP$). Put in a nutshell, in this paper, we propose to solve the above mentioned paradoxical $ATPP$ using tools provided by Learning Automata (LA), which have powerful potential in efficiently and quickly learning the optimal action when operating in unknown stochastic environments. It adaptively, and in an on-line manner, gradually learns the identity and characteristics of the users who provide fair ratings, and of those which provide unfair ratings, even when these are a consequence of them making unintentional mistakes.

The solutions provided here have been subjected to rigorous experimental tests, and the results presented are, in our opinion, both novel and conclusive.

1.2 Reputation systems: state of the art

Finding ways to solve the $ATPP$ and thus counter the detrimental influence of unfair ratings on a RS has been a focal concern of a number of studies [3, 5, 13, 22, 28, 30]. Dellarocas [5] used elements from collaborative filtering to determine the nearest neighbors of an agent that exhibited similar ratings on commonly-rated subjects. He then applied a cluster filtering approach to filter out the most likely unfairly positive ratings. Sen and Sajja [22] proposed an algorithm to select a service provider to process a task by querying other user agents about their ratings of the available service providers. The main idea motivating their work is to select a subset of agents, who when queried, gives a minimum probabilistic guarantee that the majority of the queried agents provide correct reputation estimates. However, comprehensive experimental tests show that their approach is prone to the variation of the ratio of deceptive agents. As opposed to these, Zacharia [30] proposed a game-theoretic model to solve the trust problem in online markets. In [29], the authors present a heuristic methodology to reduce the computational complexity of ratings prediction in a trust network topology while maintaining accuracy. Their approach relies on experimentally verifying that the trust network exhibits the so called "small-worldness" network property. The authors verified the validity of their approach through experimental data extracted from online trust sites. Apart from online markets applications, the concept of trust was shown to be useful to avoid access to fraudulent and malicious web sites [15]. In [15], the authors presented a proxy-based approach that makes use of safety ratings provided by McAfee SiteAdvisor in order to prevent access to untrustworthy web sites.

It is worth noting that combining reports from different witnesses is akin to the problem of fusioning possibly conflicting sources of information [2, 7, 10]. Buchegger and Le Boudec [3] tackled the latter issue as follows: They proposed

a Bayesian reputation mechanism in which each node isolates malicious nodes by applying a deviation test methodology. Their approach requires the agent to have enough *direct* experience with the services so that he can evaluate the trustworthiness of the reports of the witnesses. While this is a desirable option, unfortunately, in real life, such an assumption does not always hold, specially when the number of possible services is large. In [4], Chen and Singh evaluated the quality of feedbacks assuming that a feedback is credible if it is consistent with the majority of feedbacks for a given user. Their approach, though promising, unfortunately, suffers from a deterioration in the performance when the ratio of deceptive agents is high. In [28], Yu and Singh devised a modified weighted majority algorithm to combine reports from several witnesses to determine the ratings of another agent. The main shortcoming of the work reported in [28] is its relatively slow rate of convergence. In contrast, in [27], Witby and Jøsang presented a Bayesian approach to filter out dishonest feedback based on an iterated filtering approach. In their approach, the authors extended the so-called "Beta" reputation system presented by Jøsang and Ismail [9]. The authors of [6] proposed a probabilistic model to assess peer trustworthiness in P2P networks. Their model, which, in one sense, is similar to ours, differs from the present work because the authors of [6] assume that a peer can deduce the trustworthiness of other peers by comparing its own performance with reports of other peers about itself. Though such an assumption permits a feedback-evaluating mechanism, it is based on the fact that peers provide services to one another, thus permitting every party the right to play the role of a service provider and the service consumer (a reporting agent). Our approach, which we briefly describe in the next section and then explain in greater detail subsequently, makes a clear distinction between these parties—the service provider and the reporting agent.

1.3 Overview of our solution

In this paper, we provide a novel solution to the above problems, and in particular to the $ATPP$, based on Learning Automata (LA), which can learn the optimal action when operating in unknown stochastic environments. Furthermore, they combine rapid and accurate convergence with low computational complexity. In addition to its computational simplicity, unlike most reported approaches, our scheme does not require prior knowledge of the *degree* of any of the above mentioned problems with RSs. Rather, it adaptively, and in an on-line manner, gradually learns the identity and characteristics of the users who provide fair ratings, and of those which provide unfair ratings, even when these are a consequence of them making unintentional mistakes.

Learning is achieved by interacting with a so-called "Environment", and by processing its responses to the actions that are chosen. Such automata have various applications such as parameter optimization, statistical decision making and telephone routing [14, 18–20]. Narendra and Thathachar [14] have dedicated a book that reviews the families and applications of LA, and a brief survey of this field is included here in the interest of completeness.

By suitably modeling reports about direct experiences involving a specific service as responses from the corresponding "Environment", our scheme intelligently groups agents according to the rating that they give to the same service. To formalize these responses, we define an agent to be "fair" (or "trustworthy") if it reports the service performance correctly with a probability $p > 0.5$. Similarly, an agent is said to be "deceptive" if it reports the inverted service performance with a probability $q > 0.5$. The beauty of our scheme is that although the identity of the reporting agents is unknown, "fair" agents will end up in the same group, while "deceptive" agents will converge to another group.

Unlike most existing reported approaches that only consider the feedback from "fair" agents as being informative, and which simultaneously discard the feedback from "unfair" agents, in our work we attempt to intelligently combine (or fuse) the feedback from fair and deceptive agents when evaluating the performance of a service. Moreover, we do not impose the constraint that we need a priori knowledge about the ratio of deceptive agents. Consequently, unlike most of existing work, that suffer from a decline in the performance when the ratio of deceptive agents increases, our scheme is robust to the variation of this ratio. This characteristic phenomenon of our scheme is unique.

1.4 Contributions of this paper

The novel contributions of this paper, when it concerns RSs, are the following:

- We demonstrate the applicability of LA to RSs—thus providing a promising real-time solution to this paradoxical problem. To our knowledge, the paper presents the first reported LA-based solution for any problem within the field of RSs.
- Since our solution is based on LA, it is both computationally simple and memory efficient.
- With regard to the field of LA itself, our scheme presents twofold contributions to traditional LA-based partitioning algorithms [17, 19, 20]. First of all, we do not impose the constraint that an equal number of agents must reside in the same partition. Secondly, we experimentally demonstrate that the partitioning still yields accurate results when the environment is stochastic. In this sense, the parameter used here to decide whether two agents are reckoned "similar" is stochastic, while the latter parameter was assumed *constant* in [17, 19, 20].

- Most importantly, however, our scheme maximizes the likelihood of selecting high quality services in the presence of an *unknown* ratio of deceptive agents. Indeed, not only does the scheme not require the a priori knowledge of the ratio of the deceptive agents, but it is also very robust to extremely high ratios of such deceptive or even malicious agents!

We conclude this section by mentioning that our results probably represent the state-of-the-art!

1.5 Paper Organization

Earlier, in Sect. 1.2 we presented a brief survey of the available solutions for dealing with fair and unfair agents in RSs. The rest of the paper is organized as follows. First of all, in Sect. 2, we present a formal statement of the problem. Then, in Sect. 3 we present a brief overview of the field of LA. Thereafter, in Sect. 4 we present our solution, which is the LA-based scheme for selecting services in stochastic environments. Experimental results obtained by rigorously testing our solution for a variety of scenarios and for agents with different characteristics, are presented in Sect. 5. Section 6 concludes the paper.

2 Modeling the problem

Let us consider a population of L services (or service providers), $\mathcal{S} = \{S_1, S_2, \ldots, S_L\}$. We also assume that the social network (or pool of users) consists of N parties (synonymously called "agents") $\mathcal{U} = \{u_1, u_2, \ldots, u_N\}$. Each service S_l has an associated quality, which, in our work is represented by an "innate" probability of the service provider performing exceptionally well whenever its service is requested by an agent. This probability is specified by the quantity θ_l, assumed to be unknown to the users/agents. For a given interaction instance between user agent u_i and service S_l, let x_{il} denote the performance value, which, for the sake of formalism, is assumed to be generated from a distribution referred to as the *Performance Distribution* of S_l. After the service has been provided, the user/agent u_i observes the performance x_{il}, where $x_{il} \in \{0, 1\}$. Since we intend to reduce our problem to a *maximization* problem, we assume that '0' denotes the lowest performance of the service, while '1' denotes its highest performance.[2]

At this juncture, after the agent has experienced the quality of the service, he communicates his experience to the rest of the network. Let y_{il} be the report that he transmits to other agents after he experiences x_{il}, where,[3] $y_{il} \in \{0, 1\}$. It is here that we have to model the genuineness of an agent communicating his evaluation accurately. To do this, we assume that agent u_i communicates his experience, x_{il}, truthfully to other agents in the population, with a probability p_i. In other words, p_i denotes the probability that agent u_i is not misreporting his experience. For ease of notation, we let $q_i = 1 - p_i$, which represents the probability that agent u_i does, in fact, misreport his experience. The intention for this symbolism should be obvious, because clearly, $p_i = Prob(x_{il} = y_{il})$.

Observe that as a result of this communication model, a "deceptive" agent will probabilistically tend to report low performance experience values for high performance services and vice versa. Our aim, then, is to incrementally partition the agents as being true/fair or deceptive, concurrently with their experiences being communicated to us. Furthermore, at the same time as the agents are being partitioned, our aim is to use the present state of the ongoing partitioning as a basis for decision making when selecting services. Thus, our scheme can be divided into two interacting phases, namely, an agent partitioning phase and a service selection phase. The input to the agent partitioning phase is the reports communicated by the other agents, while the input to the service selection phase is the current agent partitioning. Decisions about whether to access a service or not are the output of the overall procedure, with the goal of making the decisions that maximize the service performance experienced by the agent that acts upon those decisions.

Note that we do not assume that an agent can access any service, any time he wants. Rather, we assume that service access is spatially and temporally restricted. Thus, the true nature of services and agents are only gradually revealed, i.e., as the overall system of agents and services are observed over time. Also, note that we assume that some of the agents spend some time on exploration, and not all of their time purely on exploiting the experiences communicated by other agents. This is necessary in order for the overall system to discover the nature of new services when they are introduced, as well as detecting the new nature of an old service that changes nature.

Formally, the Agent-Type Partitioning Problem ($ATPP$), can be stated as follows: A social network consists of N agents, $\mathcal{U} = \{u_1, u_2, \ldots, u_N\}$, where each agent u_i is characterized by a fixed but unknown probability p_i of him reporting his experience truthfully. The ATPP involves partitioning $\mathcal{U}$ into 2 (mutually exclusive and exhaustive) groups so as to obtain a 2-partition $\mathbb{G}_k = \{G_i \mid i = 1, 2\}$, such that each group, G_i, of size, N_i, exclusively contains only the

[2] The extension to non-binary ratings (for example when x_{il} is a real number in the unit interval) is "trivial", if we use the S-model of LA computations instead of the P-model.

[3] We mention, in passing, that other researchers, have used the notation y_{il} to signify the rating of the service.

agents of its own type, i.e., which either communicate truthfully or deceptively.

Since the set of all possible solutions is isomorphic to the set of all possible subsets of $\mathcal{U}$, we conjecture that the problem of determining the optimal 2-partition is NP-hard.

To simplify the problem, we assume that every p_i can assume one of two[4] possible values from the set $\{p_d, p_f\}$, where $p_d < 0.5$ and $p_f > 0.5$. Then, agent u_i is said to be fair if $p_i = p_f$, and is said be deceptive if $p_i = p_d$.

Based on the above, the set of fair agents is $\mathcal{U}_f = \{u_i | p_i = p_f\}$, and the set of deceptive agents is $\mathcal{U}_d = \{u_i | p_i = p_d\}$.

Let y_{il} be a random variable defined as below:

$$y_{il} = \begin{cases} 1 & \text{w.p } p_i.\theta_l + (1-p_i).(1-\theta_l) \\ 0 & \text{w.p } p_i.(1-\theta_l) + (1-p_i).\theta_l. \end{cases} \tag{1}$$

Consider the scenario when two agents u_i and u_j utilize the same service S_l and report on it. Then, based on the above notation, their reports relative to the service S_l are y_{il} and y_{jl} respectively, where:

$$\begin{aligned} &Prob(y_{il} = y_{jl}) \\ &= Prob[(y_{il} = 0 \wedge y_{jl} = 0) \vee (y_{il} = 1 \wedge y_{jl} = 1)] \\ &= Prob[(y_{il} = 0 \wedge y_{jl} = 0)] \\ &\quad + Prob[(y_{il} = 1 \wedge y_{jl} = 1)] \\ &= Prob(y_{il} = 0) \cdot Prob(y_{jl} = 0) \\ &\quad + Prob(y_{il} = 1) \cdot Prob(y_{jl} = 1). \end{aligned}$$

Throughout this paper, we shall denote $Prob(y_{il} = y_{jl})$ to be the probability that the agents u_i and u_j will agree in their appraisal. This quantity has the following property.

Theorem 1 *Let u_i and u_j two agents. If both u_i and u_j are of the same nature (either both deceptive agents or both fair), then $Prob(y_{il} = y_{jl}) > 0.5$. Similarly, if u_i and u_j are of different nature, then $Prob(y_{il} = y_{jl}) < 0.5$.*

Proof The proof is straightforward. □

We shall now proceed to present a brief overview of LA, the toolkit to solve the $ATPP$.

[4]Generalizing this so that each p_i can be an element of a set $\{p_{d_1}, p_{d_2}, \ldots p_{d_J}, p_{f_1}, p_{f_2}, \ldots p_{f_M}\}$, where every $p_{d_i} < 0.5$ and every $p_{f_j} > 0.5$ is rather trivial. It merely involves extending the arguments presented here for all possible pairs $\langle p_{d_i}, p_{f_j} \rangle$. Notice that in the same vein, agent u_i would be considered fair if $p_i \in \{p_{f_1}, p_{f_2}, \ldots p_{f_M}\}$, and he would be deceptive if $p_i \in \{p_{d_1}, p_{d_2}, \ldots p_{d_J}\}$.

3 Stochastic learning automata

Learning Automata[5] (LA) have been used in systems that have incomplete knowledge about the Environment in which they operate [1, 14, 21, 25]. The learning mechanism attempts to learn from a *stochastic Teacher* which models the Environment. In his pioneering work, Tsetlin [26] attempted to use LA to model biological learning. In general, a random action is selected based on a probability vector, and these action probabilities are updated based on the observation of the Environment's response, after which the procedure is repeated.

The term "Learning Automata" was first publicized by Narendra and Thathachar [14]. The goal of LA is to "determine the optimal action out of a set of allowable actions" [1]. The distinguishing characteristic of automata-based learning is that the search for the optimizing parameter vector is conducted in the space of probability *distributions* defined over the parameter space, rather than in the parameter space itself [24].

In the first LA designs, the transition and the output functions were time invariant, and for this reason these LA were considered "Fixed Structure Stochastic Automata" (FSSA). Tsetlin, Krylov, and Krinsky [26] presented notable examples of this type of automata. The solution we present here, essentially falls within this family and so we shall explain this family in greater detail in Sect. 3.1.

Later, Vorontsova and Varshavskii [14] introduced a class of stochastic automata known in the literature as Variable Structure Stochastic Automata (VSSA). In the definition of a VSSA, the LA is completely defined by a set of actions (one of which is the output of the automaton), a set of inputs (which is usually the response of the Environment) and a learning algorithm, T. The learning algorithm [14] operates on a vector (called *the Action Probability vector*)

$$P(t) = [p_1(t), \ldots, p_R(t)]^{\mathrm{T}},$$

where $p_i(t)$ $(i = 1, \ldots, R)$ is the probability that the automaton will select the action α_i at time 't', $p_i(t) = Pr[\alpha(t) = \alpha_i]$, $i = 1, \ldots, R$, and it satisfies

$$\sum_{i=1}^{R} p_i(t) = 1 \quad \forall t.$$

Note that the algorithm $T : [0,1]^R \times A \times B \to [0,1]^R$ is an updating scheme where $A = \{\alpha_1, \alpha_2, \ldots, \alpha_R\}$, $2 \leq R < \infty$, is the set of output actions of the automaton,

[5]In the interest of completeness, we have included a brief review of the field of LA here. The review found in the earlier version of the paper has been abridged as per the desire of the referees. The list of applications of LA is also extensive, but omitted here in the interest of brevity and the advice of the referees.

and B is the set of responses from the Environment. Thus, the updating is such that

$$P(t+1) = T(P(t), \alpha(t), \beta(t)),$$

where $P(t)$ is the action probability vector, $\alpha(t)$ is the action chosen at time t, and $\beta(t)$ is the response it has obtained.

3.1 Fundamentals of FSSA

Since the solution to the $ATPP$ which we present here essentially falls within the family of FSSA, we explain them now in greater detail. A $FSSA$ is a quintuple ($\underline{\alpha}$, $\underline{\Phi}$, $\underline{\beta}$, F, G) where:

- $\underline{\alpha} = \{\alpha_1, \ldots, \alpha_R\}$ is the set of actions that it must choose from.
- $\underline{\Phi} = \{\phi_1, \ldots, \phi_S\}$ is a set of states.
- $\underline{\beta} = \{0, 1\}$ is its set of inputs. The '1' represents a penalty, while the '0' represents a reward.
- F is a map from $\Phi \times \beta$ to Φ. It defines the transition of the internal state of the automaton on receiving an input. F may be stochastic.
- G is a map from Φ to α, and it determines the action taken by the automaton if it is in a given state. With no loss of generality, G is deterministic.

As discussed above, the automaton is offered a set of actions, and it is constrained to choose one of them. When an action is chosen, the Environment gives out a response $\beta(t)$ at a time 't'. The automaton is either penalized or rewarded with an unknown probability c_i or $1 - c_i$, respectively. On the basis of the response $\beta(t)$, the state of the automaton $\phi(t)$ is updated and a new action is chosen at $(t+1)$. The penalty probability c_i satisfies:

$$c_i = Pr[\beta(t) = 1 | \alpha(t) = \alpha_i] \quad (i = 1, 2, \ldots, R).$$

The basic idea used to solve the $ATPP$ is based on a sub-class of LA solutions that has been used to solve the object partitioning problem [8, 16]. As documented in the literature, the object partitioning problem involves partitioning a set of $|\mathbb{P}|$ objects into $|\mathbb{N}|$ groups or classes, where the main aim is to partition the objects into groups that mimic an underlying unknown grouping. In other words, the objects which are accessed together must reside in the same group [16]. In the special case when all the groups are required to contain the same number of objects, the problem is also referred to as the Equi-Partitioning Problem (EPP). Many solutions involving LA have been proposed to solve the EPP, but the most efficient algorithm is the *Object Migrating Automaton* (OMA) [16]. The latter was first proposed by Oommen and Ma [16], and some modifications were added by Gale et al. [8] to create the *Adaptive Clustering Algorithm* (ACA). Since the OMA is, in one sense, the prior art on which our present solution is built, and since we have compared our scheme with the OMA, we briefly describe its design here.

3.2 Object migrating automaton (OMA)

The Object Migrating Automaton (OMA) is an ergodic automaton that has R actions $\{\alpha_1, \ldots, \alpha_R\}$ representing the possible underlying classes. Each action α_i has its own set of states $\{\phi_{i1}, \phi_{i2}, \ldots, \phi_{iM}\}$, where M is the depth of memory, and $1 \leq i \leq R$ represents the number of classes. ϕ_{i1} is called the most internal state and ϕ_{iM} is the boundary (or most external) state.

A set of W physical objects $\{A_1, A_2, \ldots, A_W\}$ is accessed by a random stream of queries, and the objects are to be partitioned into groups so that the frequently *jointly*-accessed objects are clustered together. The OMA utilizes W abstract objects $\{O_1, O_2, \ldots, O_W\}$ instead of migrating the physical objects. Each abstract object is assigned to a state belonging to an initial random group but in its boundary state. The objects within the automaton move from one action to another, and so, in this case, all the W abstract objects move around in the automaton. If the abstract objects O_i and O_j are in the action α_h, and the request accesses $\langle A_i, A_j \rangle$, then the OMA will be rewarded by moving them towards the most internal state ϕ_{h1}. But a penalty arises if the abstract objects O_i and O_j are in different classes, say α_h and α_g, respectively. Assuming O_i is in $\zeta_i \; \varepsilon \; \{\phi_{h1}, \phi_{h2}, \ldots, \phi_{hM}\}$ and O_j is in $\zeta_j \; \varepsilon \; \{\phi_{g1}, \phi_{g2}, \ldots, \phi_{gM}\}$, they will be moved as follows:

- If $\zeta_i \neq \phi_{hM}$ and $\zeta_j \neq \phi_{gM}$, O_i and O_j are moved one state toward ϕ_{hM} and ϕ_{gM}, respectively.
- If exactly one of them is in the boundary state, the object which is not in the boundary state is moved towards *its* boundary state.
- If both of them are in their boundary states, one of them, say O_i is moved to the boundary state of the other object ϕ_{gM}. In addition, the closest object to them is moved to the boundary state ϕ_{hM}, so as to preserve an equal number of objects in each group.

It is important to point out that the random stream of queries contains information about an optimal partition, and the OMA attempts to converge to it. The automaton is said to have converged when all the objects in a class are in the deepest (or second deepest) most-internal state.

The OMA can be improved by the following: Assume that a pair of objects $\langle A_i, A_j \rangle$ is accessed, where O_i is in the boundary state, while O_j is in a non-boundary state. In this case, a general check should be made to locate another object in the boundary state of the partition containing O_j. If there is an object, then swapping is done between this object and O_i in order to bring the two accessed objects into the

same partition. In turn, instead of waiting for a long time to have these accessed objects in the same partition, the convergence speed can be increased by swapping the objects into the right partitions.

The formal algorithm for the OMA is found in [8, 16], and omitted here in the interest of space.

3.3 Similarities between the $ATPP$ and the EPP

The idea behind using LA as a tool to solve the $ATPP$ comes from the elegance of using them to solve the EPP. The similarity between the EPP and the $ATPP$ render LA as one of the promising candidate tools to solve the latter. This is because:

- As in the case of the EPP, the $ATPP$ is (possibly) NP-hard, primarily due to the exponential growth in the number of partitions of objects/agents.
- The EPP dictates that each partition must have the same number of objects. It is easy to see that an analogous condition can be imposed with the $ATPP$ if the number of fair and deceptive agents are equal. Relaxing this constraint will be one of our major challenges.
- The EPP and $ATPP$ seek to partition the objects/agents into groups that mimic the underlying unknown groups of objects and agents respectively. In the case of the EPP, the objects which are accessed together more frequently by a random sequence of queries are said to be in the same partition. As opposed to this, in the $ATPP$, the agents which are similar to each other (by being either fair or deceptive), are required to be in the same group so as to maximize the "within-group" and to minimize the "between-group" similarities.

We now highlight the *differences* between the two problems.

3.4 Limitations of the OMA in the $ATPP$ context

The reported instances of the OMA are not directly applicable for the $ATPP$. To develop our solution, we highlight the main restrictions, and the necessary enhancements which must be added to the OMA in order for it to be useful in our present application domain.

- First of all, unlike the EPP, the $ATPP$ does not require the number of agents in each group to be the same.
- In case of the $ATPP$, the user does not have access to the stream of random queries. Rather, the only available data is the set of instances when the appraisal of one agent concurs with that of another. It is thus apparent that we have to artificially "generate" a sequence of "queries" (or pairs) which can be used to operate on a machine similar to the OMA. The above restriction has a "two-edged" implication. First of all, in the EPP, the user usually requests the system to obtain a query pair of the form $\langle O_i, O_j \rangle$. However, in the $ATPP$, it is our responsibility, while designing the algorithm, to determine which agents should be deemed similar or dissimilar, and the reader will observe that this determination is a problem to be solved in its own right. Secondly, in the OMA, the placement of the objects in the automaton and the stream of random queries, together, serve to either reward or penalize the automaton. However, in the case of the $ATPP$, the question of obtaining a reward/penalty response is not provided by the user, but it has to be *inferred*. This again has to be solved.
- Unlike the EPP, which has no way of penalizing "non-accessed elements", a solution to the $ATPP$ must develop a strategy for penalizing such agents by considering how similar the agents within the same groups are. Clearly, this is superfluous for the EPP because, in that problem, the automaton is absolutely dependent on the user's queries. In the present problem, it is crucial that an automaton can quantify how fitting an agent is for any given group.
- The optimal partition for the EPP yields crucial information in the stream of random queries. As opposed to this, in the context of the $ATPP$, the system has no notion of how to characterize the optimal partition. This renders the problem of adapting the OMA to solve the $ATPP$ more difficult.
- In the same vein, the definition of the optimal partition for the EPP is quite different from that of the corresponding solution for the $ATPP$. In the case of the EPP, all objects which are accessed together more frequently should be in the same partition, while in the $ATPP$ all agents which *respond* in a similar way should be in the same group.
- The criteria which are used to reward and penalize the automaton in the EPP is quite unlike the one used for the $ATPP$. In the EPP, the automaton is rewarded or penalized based on the (unknown) probability of any two objects being jointly accessed. But in the context of the $ATPP$, the automaton is reward or penalized by "conducting a comprehensive study" of the relation between the individual agents, which, furthermore, may or may not participate in a communication within the social network.
- Although the EPP and $ATPP$ utilize analogous rules for a reward phenomenon, as we shall see, they differ in performing the penalty rules. In case of the EPP, the automaton enforces the rule that the pertinent object migrates, if and only if at least one of the accessed objects is at the boundary state of the different partitions. As opposed to this, in the $ATPP$, the automaton enforces the rule that the agents are migrated to the alternate group whenever the task of migrating is dictated by their *joint* appraisal probabilities.

- The automaton used to solve the EPP is said to have converged, when all the objects are found in the most (or the last two) internal states of each partition. However, we propose that the convergence in the $ATPP$ occurs when the measure of their clusters is satisfactory. For example, if all the agents in both class are in their most internal states, the within-cluster distance of each cluster would be zero, and the between-cluster distance would be $2M$. We can then say that convergence has occurred if the weighted sum of the within-cluster distance and their between-cluster distance is greater than a given threshold.

4 A LA-based solution to the $ATPP$

This section describes, in fair detail, all the aspects and algorithmic issues associated with our LA-based solution to the $ATPP$. To initiate this, in Sect. 4.1 we first explicitly state the inputs, outputs and goals of the entire exercise. Then, in Sect. 4.2, we present a formal overview of the solution, including the components of its 7-tuple formulation. Each of the elements of this automaton are formally explained here. Since the formulation of the LA has to explicitly declare the responses it makes to Rewards and Penalties, Sect. 4.3 explicitly defines these both diagrammatically and algorithmically. This is followed, in Sect. 4.4, by a description of how the user concentrates his observation on a sliding window of interactions with other agents in the social network, and a simple example (in Sect. 4.6) concludes this section.

4.1 Inputs, outputs and goals

In order to develop our LA-based solution to the $ATPP$, it is appropriate that we re-iterate what the corresponding inputs and outputs are. The input to our automaton, is the set of user agents $\mathcal{U} = \{u_1, u_2, \ldots, u_N\}$ and the reports that are communicated within the social network. With regard to the output, we intend to partition $\mathcal{U}$ into two sets, namely, the set of fair agents $\mathcal{U}_f = \{u_i | p_i = p_f\}$, and the set of deceptive agents $\mathcal{U}_d = \{u_i | p_i = p_d\}$. The intuitive principle that we use is that the agents that have the same nature (fair or deceptive) will report similar experiences about the same service, and we shall attempt to infer this phenomenon to, hopefully, migrate them to the same partition. Observe that since agents of different nature report dissimilar experiences about the same service, we hope to also infer *this*, and, hopefully, force them to converge into different partitions.

4.2 Formal definition of the LA-based $ATPP$ solution

We define the Agent Migrating Partitioning Automaton ($AMPA$) as a 7-tuple as below: ($\mathcal{U}$, $\underline{\Phi}$, $\underline{\alpha}$, $\underline{\beta}$, $\mathbb{Q}$, $\mathbb{G}$, $\mathbb{W}$), where

- $\mathcal{U} = \{u_1, u_2, \ldots, u_N\}$ is the set of agents.
- $\underline{\Phi} = \{\phi_1, \phi_2, \ldots, \phi_{2M}\}$ is the set of states.
- $\underline{\alpha} = \{\alpha_1, \alpha_2\}$ is the set of actions, each representing a group into which the elements of $\mathcal{U}$ fall.
- $\underline{\beta} = \{$'0', '1'$\}$ is the set of responses, where '0' represents a *Reward*, and '1' represents a *Penalty*.
- $\mathbb{Q}$ is the transition function, which specifies how the agents should move between the various states. This function is quite involved and will be explained in detail presently.
- The function $\mathbb{G}$ partitions the set of states for the groups. For each group, α_j, there is a set of states $\{\phi_{(j-1)M+1}, \ldots, \phi_{jM}\}$, where M[6] is the depth of memory. Thus,

$$G(\phi_i) = \alpha_j \quad (j-1)M + 1 \leq i \leq jM. \tag{2}$$

  This means that the agent in the automaton chooses α_1 if it is in any of the first M states, and that it chooses α_2 if it is in any of the states from ϕ_{M+1} to ϕ_{2M}. We assume that $\phi_{(j-1)M+1}$ is the most internal state of group α_j, and that ϕ_{jM} is the boundary state. These are called the states of "*Maximum Certainty*" and "*Minimum Certainty*", respectively.
- $\mathbb{W} = \{W_l^D(t)\}$, where, $W_l^D(t)$ = {Last D records prior to instant t relative to service S_l}.

  Our aim is to infer from $\mathbb{W}$ a similarity list of agents deemed to be collectively similar. From it we can, based on the window of recent events, obtain a list of pairs of the form $\langle u_i, u_j \rangle$ deemed to be similar. The question of how $\mathbb{W}$ is obtained will be discussed later.

To see how all these components flow together, we shall now explain how the learning cycle is performed—which is the central kernel where the $ATPP$ is solved.

The learning phase is the core of the clustering. The $AMPA$ model is initialized by placing all the agents at the boundary state of their initially randomly-chosen groups. This indicates that the $AMPA$ is initially uncertain of the placement of the agents, because the different states within a given group quantify the measure of certainty that the scheme has for a given agent belonging to that group. As the learning cycle proceeds, similar agents will be rewarded for their being together in the same group, and they will be penalized by either moving toward their boundary state, or to another group, as will be clarified presently.

[6] Generally speaking, the depth of the memory, M, in the LA could play an important role in determining the accuracy of the LA, while the eigenvalues of the underlying chain would determine the machine's rate of convergence. To the best of our knowledge, we are not aware of any method used to determine the best value for M except the trial-and-error approach. The effect of varying M will be explained in Sect. 5.

4.3 Reward and penalty transitions

Philosophically, we mention that since we require that all the elements of $\mathcal{U}$ move among the states of the machine, it is distinct from traditional FSSA, which, being a traditional finite state machine, always finds itself in only one of a finite number of states. Also, if agent u_i is in action α_j, it signifies that it is in the sub-partition whose index is j. Moreover, if the states occupied by the nodes are given, the sub-partitions can be trivially obtained by invoking (2).

Let $\zeta_i(t)$ be the index of the state occupied by agent u_i at the t^{th} time instant. Based on $\{\zeta_i(t)\}$ and (2), let us suppose that the automaton decides a current partition of $\mathcal{U}$ into sub-partitions. Using this notation we shall later describe the transition map of the automaton. Since the intention of the learning process is to collect "similar" agents into the same sub-partition, the question of "inter-agent similarity" (i.e., inferring which two agents should be grouped together) is rather crucial. In the spirit of Theorem 1, we shall reckon that two agents are similar if they are of the same nature, implying that the corresponding probability of them agreeing is greater than 0.5.

We now consider the reward and penalty scenarios separately.

4.3.1 Transitions for rewards

(a) Whenever two agents u_i and u_j test the same service, if their corresponding reports are identical (either both '0' or both '1'), and they currently belong to the same partition, the automaton (and, in particular, u_i and u_j) is rewarded. This mode of rewarding is called the *RewardAgreeing* mode depicted in Fig. 1, and the algorithm is formally given in Algorithm 1 titled "Reward_Agreeing_Nodes".

(b) As opposed to this, if u_i and u_j are dissimilar and they currently belong to distinct partitions, the automaton (and again, in particular, u_i and u_j) is rewarded. This mode of rewarding is called the *RewardDisagreeingMode*. The algorithm is identical to the formal algorithm given in Algorithm 1, and is thus omitted to avoid repetition.

By way of explanation, more specifically, on being rewarded, since the agents u_i and u_j are in the "correct" group, say α_p, with regard to the agent it is being compared with, both of them are moved towards the most internal state of that group, one step at a time. Observe that it does not matter whether either (or both) of them is in a boundary state. The overall scheme is given in Algorithm 2 titled "Reward_Agent".

4.3.2 Transitions for penalties

Here we encounter three cases as listed below.

Algorithm 1 Procedure Reward_Agreeing_Nodes

Input: ζ_i and ζ_j: the indices of the states where the agents R_i and R_j are located in the LA.
Output: The updated values of ζ_i and ζ_j.
Method:

1: Reward_Agent(ζ_i)
2: Reward_Agent(ζ_j)
3: **return** ζ_i and ζ_j

End Procedure Reward_Agreeing_Nodes

Algorithm 2 Procedure Reward_Agent

Input: ζ_i, which represents the index of the state where the agent R_i is located in the LA.
Output: The updated value of ζ_i.
Method:

1: **if** (ζ_i mod M $\neq 1$) **then**
2: $\quad \zeta_i = \zeta_i - 1$
3: **end if**
4: **return** ζ_i

End Procedure Reward_Agent

(a) Whenever two agents u_i and u_j test the same service, if their corresponding reports are identical (either both '0' or both '1'), and they currently belong to distinct partitions, the automaton (and, in particular, u_i and u_j) is penalized. More specifically, this case is encountered when two similar agents, u_i and u_j, are allocated in distinct groups say, α_a and α_b respectively (i.e., R_i is in state ζ_i, where $\zeta_i \in \{\phi_{(a-1)M+1}, \ldots, \phi_{aM}\}$, and R_j is in state ζ_j, where $\zeta_j \in \{\phi_{(b-1)M+1}, \ldots, \phi_{bM}\}$). This mode of rewarding is called the *PenalizeAgreeing* mode depicted in Fig. 2, and the algorithm is formally given in Algorithm 3 titled "PenalizeAgreeingNodes".

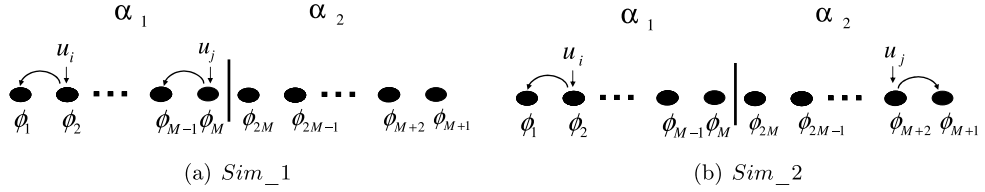
Algorithm 3 Procedure Penalize_Agreeing_Nodes

Input: ζ_i and ζ_j: the indices of the states where the agents R_i and R_j are located in the LA.
Output: The updated values of ζ_i and ζ_j.
Method:

1: Penalize_Agent(ζ_i)
2: Penalize_Agent(ζ_j)
3: **return** ζ_i and ζ_j

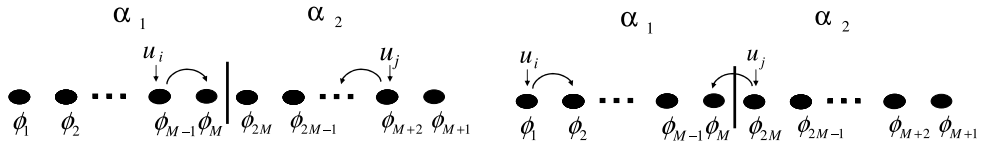
End Procedure Penalize_Agreeing_Nodes

(b) However, if u_i and u_j are both assigned to the same sub-partition, but they should rather be assigned to distinct groups, the automaton is penalized. Analogous to the above, this mode of penalizing is called the *PenalizeDisagreeing* mode, because, in this mode, agents which are



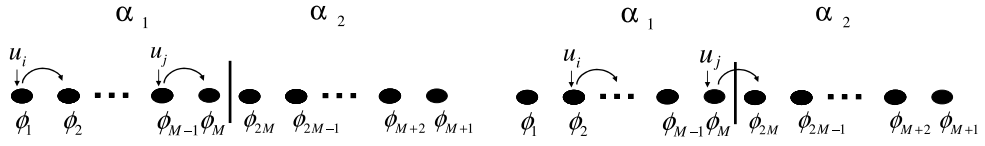
(a) Sim_1 (b) Sim_2

Fig. 1 (**a**) RewardAgreeing Mode: This is the case when both agents u_i and u_j belong to the same partition. Observe that it does not matter whether either (or both) of them is in a boundary state. (**b**) Reward Disagreeing Mode: This is the case when both agents u_i and u_j belong to different partitions. Again, observe that it does not matter whether either (or both) of them is in a boundary state



(a) Sim_1 (b) Sim_2

Fig. 2 PenalizeAgreeing Mode: This is the case when both agents u_i and u_j belong to different partitions. In (**a**), neither of them is in a boundary state. As opposed to this, in (**b**), the figure depicts the case when one of them, u_j, is in a boundary state



(a) Sim_1 (b) Sim_2

Fig. 3 PenalizingDisagreeing mode: This is the case when both agents u_i and u_j belong to the same partition, when, in actuality, they should belong to distinct partitions. In (**a**), neither of them is in a boundary state. As opposed to this, in (**b**), the figure depicts the case when one of them, u_j, is in a boundary state

Algorithm 4 Procedure Penalize_Agent

Input: ζ_i, which represents the index of the state where the agent R_i is located in the LA.
Output: The updated value of ζ_i.
Method:

```
if (ζ_i mod M ≠ 0) then
    ζ_i = ζ_i + 1
else
    if (ζ_i = M) then
        ζ_i = 2M
    else
        ζ_i = M
    end if
end if
return ζ_i
```

End Procedure Penalize_Agent

actually dissimilar are assigned to the same subpartition and they are therefore penalized. This is depicted in Fig. 3, and the algorithm is identical to the formal algorithm given in Algorithm 3. It is again omitted to avoid repetition.

(c) In both these cases, if the agent in question is in a boundary state, it is subsequently moved to the boundary state of the alternate choice.

Again, by way of explanation, on being penalized, if the agents u_i and u_j are in the same group, say α_a, both of them are moved *away* from the most internal state of that group, one step at a time. See Figs. 2 and 3 and Algorithm 4 titled "Penalize_Agent". If either of them is in a boundary state, it switches actions to go to the boundary state of the alternate action.

4.4 The window of observations

Since we adopt a simple interaction model, at each time instant a random agent chooses a random service to interact with, and can report his experience to the rest of agents.

We now address the question of recording the reports associated with the various agents, which in turn, involves the set $\mathbb{W}$, defined above. In our work, we adopt a "tuple-based" window to store the reports for a given service [12], where

the size of the window is quantified in terms of the number of tuples. We have opted to do this because it is easier to deal with tuple-based windows, since the size of each window in terms of the number of tuples is fixed. As opposed to this, a time-based window would be specified in terms of time units, where the size of each window instance may vary based on the arrival process. Observe that our approach is consistent with the work of Shapiro [23] where it was proven that in an environment in which peers can change their behavior over time, the efficiency of a reputation mechanism is maximized by giving higher weights on recent ratings and where older (stale) ratings are discounted. Clearly, this is equivalent to enforcing a sliding window.

In the partitioning phase, the agent in question observes the reports of other agents. Based on the similarity of the reports relative to the same service, the agent in question partitions the reporters into two sets. Obviously, the reporters are either deceptive or fair. Despite the ability of our LA-based clustering algorithm to separate between the two groups $\mathcal{U}_f$ and $\mathcal{U}_d$, the agent can not determine which of the two groups is the fair one, ($\mathcal{U}_f$), and which is the deceptive one, ($\mathcal{U}_d$), unless he tries the services himself. Intuitively, if the agents knows this information, he can just take the inverse of the reports of the "liars" as being trustworthy, while he considers the rest of the reports, obtained from the fair agents, as also being trustworthy.

With regard to the set $\mathbb{W}$, the decision making procedure that maximizes the likelihood of choosing high performance services, works as follows. Every agent stores the last D reports seen so far. Thus, the agent in question maintains, for every service, a sliding window over the last D reported experiences, which guarantees gathering the most recent reports. Let

$$W_l^D(t) = \{\text{Last } D \text{ records prior to instant } t \text{ relative to service } S_l\}$$

At time instant t, $W_l^D(t)$ contains the D tuples with the largest time stamps (where, if the total number d of reports seen so far is smaller than the length of the window D, the vector contains these d elements). Clearly, $W_l^D(t)$ stores the most recent D tuples.[7] Also, let $W_l^D[k]$ denote the record of index k in the vector, or the $(D-k)^{th}$ last record.

4.5 The decision making phase

In the spirit of what we have developed so far, we assume that the services belong to two categories: high performance services and low performance services. A high performance service is a service with a high[8] value of θ, and similarly, a low performance service is a service with a low value of θ. We suppose that agent u aspires to interact with high performance services. Therefore, every time u desires to access a service, u creates a list L of the recommended services by applying a majority voting method, as explained below. Based on this, u chooses a random service among the elements of this created list. In order to create a list of the high performance services, for every service S_l, agent u evaluates the feedback from agents that may have directly interacted with service S_l during the last D interactions. We adopt the terminology of a "witness" to denote an agent solicited for providing its feedback. In this sense, at instant t, agent u examines the service history vector $W_l^D(t)$ that contains the last reports of the witnesses regarding the performance of service S_l. For every report in the vector $W_l^D(t)$, agent u should take the reverse of the report as true if he believes that the witness is a "liar", and consider the rest of the reports as being trustworthy. Following such a reasoning, given D trustworthy reports about a given service, we can apply a deterministic majority voting to determine if the service is of high performance or of low performance. Obviously, if the majority of the agents assign the service a ranking of '1', the service is assumed to be of a high performance, and consequently, it is added to the list L.

However, a potential question is that of determining which partition is the deceptive one, and which involves the fair agents. In order to differentiate between the partitions we design a LA that learns which of the partitions is deceptive and which is fair—based on the result of the interaction between agent u with the selected service S_l. The automaton is rewarded whenever agent u selects a recommended service from the list L and the result of the interaction is a high performance (meaning '1'). Similarly, the automaton is penalized whenever agent u selects a recommended service from the list L and the result of the interaction is a low performance (meaning '0'). Again, we suppose that agent u in question is initially assigned to the boundary state. We observe the following:

- If agent u is in class α_j then u supposes that all the agents in α_j are fair, and the agents in the alternate class are deceptive.
- Whenever agent u decides to interact with a high performance service, he creates the list L of recommended services, and proceeds to choose a random service from L.
- If the result of the interaction is '1', a reward is generated, and the agent u goes one step towards the most internal state of class α_j.

[7] In future, unless there is ambiguity, for ease of notation, we shall omit the time index t.

[8] In the section which describes the simulations performed, a typical value that we choose for high performance services is $\theta = 0.8$, and for low performance services is $\theta = 0.2$.

- If the result of the interaction is '0', a penalty is generated and agent u goes one step towards the boundary state of class α_j.
- If agent u is already in the boundary state, he switches to the alternate class.

The formal algorithm which puts all the pieces of this puzzle together, follows in Algorithm 5. Observe that the agent whom we are interested in (say, u, who is in state ξ) invokes this. He periodically,[9] with a periodicity T, invokes the *Service Selection* module given formally in Algorithm 6. Since, as we alluded to earlier, we are working with a simple interaction model, our solution assumes that at every time instant, a random agent u_i is allowed to experience the quality of a random service, and that he communicates his experience to the rest of the network. This report will serve as an input to the *OMA-Based-Partitioning*, given formally in Algorithm 7. Consequently, the agent u will be able to continuously invoke an "intelligent" partitioning strategy between his consecutive accesses to the services, and also incrementally partition the set of reporting agents. The automaton associated with the agent u will converge to the action which yields the minimum penalty response in an expected sense. In our case, the automaton will converge to the class containing the fair agents, while the deceptive agents will converge to the alternate class.

Algorithm 5 Main_Algorithm

Input: $\mathcal{U} = \{u_1, \ldots, u_N\}$, the set of agents, and T, a parameter which specifies the access periodicity of the system.
Output: Choice of an accessed service at every T^{th} time instant.
Method:

```
1: for Every time instant n do
2:    if ((n mod T = 0)) then
3:       Service_Selection(U)
4:    else
5:       OMA_Based_Partitioning(U)
6:    end if
7: end for
End Main_Algorithm
```

4.6 Example

To show how the proposed decision making works we provide an example (see Fig. 4). In this example, we suppose

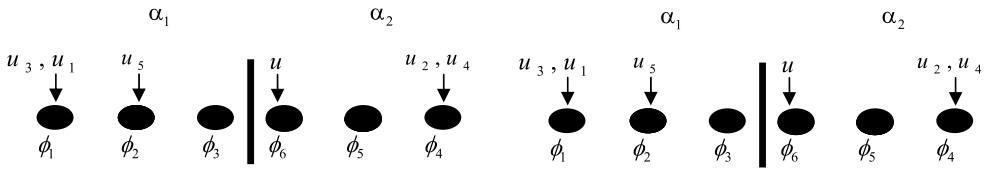
[9]Here, we suppose that the agent u aspires to access a high performance service with a pre-defined fixed frequency, for example, after every T time instances. Observe that we could just as easily have resorted to a Poisson distribution.

Algorithm 6 Procedure Service_Selection

Input: A current partition of $\mathcal{U}\{u_1, \ldots, u_N\}$ into sub-partitions. Note that ζ, which represents the state occupied by agent u.
Output: The updated value of ζ.
Method:

```
1: for every service S_l in the pool of available services do
2:    Initialize vote ← 0
3:    Initialize L ← ∅
4:    W_l^D: report vector relative to S_l
5:    for k ← 1 to D − 1 do
6:       j ← Index of the agent associated to record W_l^D[k]
7:       if ((u and u_j are in same group)∧(W_l^D[k] = 1))
         then
8:          vote := vote + 1
9:       else
10:         if ((u and u_j are in different groups)∧(W_l^D[k] =
            0)) then
11:            vote := vote + 1
12:         end if
13:      end if
14:   end for
15:   if (vote ≥ D/2) then
16:      L ← L ∪ S_l
17:   end if
18: end for
19: S_r: Service chosen at random from the list L
20: x_r: Result of the interaction as u accesses S_r
21: ζ: Index of the state occupied by agent u
22: if (x_r = 1) then
23:    Reward_Agent(ζ)
24: else
25:    Penalize_Agent(ζ)
26: end if
End Procedure Service_Selection
```

that the current partition is depicted by Fig. 4(a). Agents u_1, u_3 and u_5 belong to a different partition than agents u_2 and u_4. We suppose that the agent in question, u, desires to interact with a high performance service. u examines the report vector $W_l^D(t)$. Moreover, we suppose that $W_l^D(t)$ is composed of $\{y_{1l} = 0, y_{3l} = 0, y_{5l} = 0, y_{2l} = 1, y_{4l} = 0\}$. Since u is in state ϕ_6, it belongs to partition α_2, and therefore assumes that agents u_2 and u_4 are fair agents, since they belong to the same partition as he. Similarly, u assumes that agents u_1, u_3 and u_5 are deceptive. Therefore, u inverts the reports received from the "assumed deceptive agents", namely $(y_{1l} = 0, y_{3l} = 0, y_{5l} = 0)$ and trusts the reports from the "assumed fair agents", namely $(y_{2l} = 1, y_{4l} = 0)$. Applying the majority voting method, four 1's and one 0, leads to a majority of 1, meaning that the service S_l is assumed a priori to be a high performance service. We suppose that u



(a) Sim_1 (b) Sim_2

Fig. 4 An example of how the decision making process works. In (**a**) we encounter the so-called *PenalizeAgreeing* mode. Here the agents u_i and u_j belong to different partitions, and neither of them is in a boundary state. As opposed to this, in case (**b**), the LA encounters the *PenalizeAgreeing* mode, when both agents u_i and u_j belong to different partitions, and one of them, u_j, is in a boundary state

Algorithm 7 Procedure OMA_Based_Partitioning

Input: $U = \{u_1, \ldots, u_N\}$ is the set of agents to be partitioned.
Output: Partitioning agents into two sub-partitions.
Method:

```
A random agent i chooses a random service S_l
x_il: Result of the interaction
y_il: Reported experience
Update the vector W_l^D
for k ← 1 to D − 1 do
    j ← Index of the agent associated to record W_l^D[k]
    if (y_il = y_jl) then
        if (u_i and u_j are in same group) then
            Reward_Agreeing_Nodes(i, j)
        else
            Penalize_Agreeing_Nodes(i, j)
        end if
    else
        if (u_i and u_j are in same group) then
            Penalize_Disagreeing_Nodes(i, j)
        else
            Reward_Disagreeing_Nodes(i, j)
        end if
    end if
end for
```

End Procedure OMA_Based_Partitioning

selects S_l after creating the list L. In addition, suppose that the result of the interaction is ranked as being a "low" performance. Therefore, the automaton depicted in Fig. 4(b) is penalized. Since u is at the boundary state, ϕ_6, u switches from its current partition, and is assigned to the state ϕ_3. Being in state ϕ_3 (and consequently in partition α_1), u assumes that agents u_2 and u_4 are deceptive agents since they now belong to a different partition than he. Similarly, u now assumes that agents u_1, u_3 and u_5 are fair, and the system is ready for the next interaction!

5 Experimental results

The performance[10] of our scheme for RSs has been tested by simulation in a variety of parameter settings, and the results that we have obtained are truly conclusive. To quantify the quality of the scheme, we measured the average performance of the selected services over all interactions, and this was used as the performance index. All the results reported have been obtained after averaging across 1,000 runs, where every run consisted of 40,000 steps. In other words, each time i the agent in question, guided by our learning scheme, selects a service, the resulting performance x_i is either 0 or 1. Thus, assuming that the agent selects a total of n services across all runs and time steps, we define the average performance to be $\sum_{i=1}^{n} x_i$.

The interactions between the agents and the services were generated at random, and at every time instant, a random agent was made to select a random service. In our current experimental setting, the number of agents was 20 and the number of services in the pool of available services was 100. The services belonged to two categories, namely, high performance services with $\theta \geq 0.5$, and low performance services with $\theta \leq 0.5$. The agent in question (i.e., the one which we are interested in) periodically accessed a service every 1,000 runs—as per the above-mentioned procedure.

We report now the results obtained by testing the scheme in a variety of settings.[11]

5.1 Performance in static environments

We first report the results for environments which are static. Figure 5 shows the average performance of the service selection scheme when the testing was done over 40,000 runs. In this particular setting, 10% of the services were high

[10] We are grateful to the feedback from the anonymous Referees, whose comments helped to improve this section significantly.

[11] We have done experiments for numerous settings and scenarios. In the interest of brevity, we merely report a few representative (and typical) experimental results, so that the power of our proposed methodology can be justified.

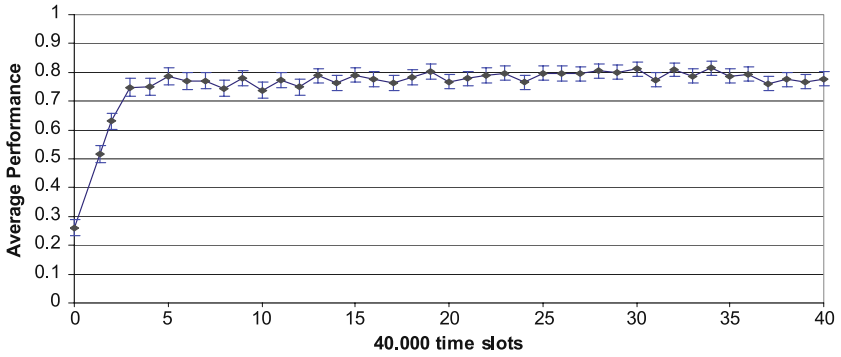


Fig. 5 The behavior of the $AMPA$, measured in terms of the average performance, in an environment when the behavior of the agents is static

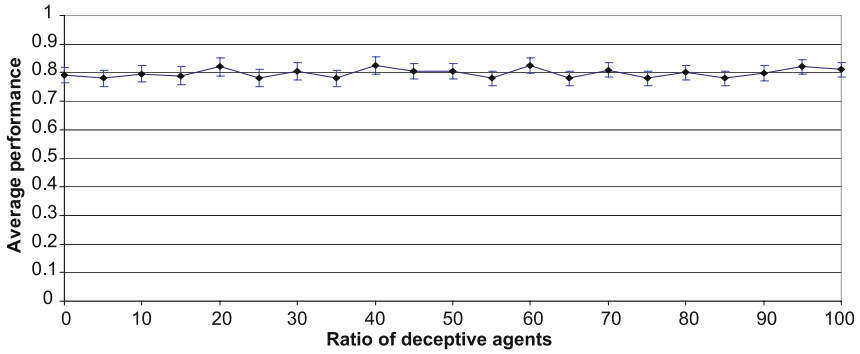


Fig. 6 The behavior of the $AMPA$ in an environment when the ratio of fair/deceptive agents is varied. The figure shows the variation of the average performance under different ratios of deceptive agents

performance services with $\theta = 0.8$, and 90% were low performance services with $\theta = 0.2$. Further, 15 of the reporting agents were deceptive with $p_d = 0.2$, while 5 were fair agents with $p_f = 0.8$. The depth of memory used for the LA was $M = 10$, and the length of the sliding window was 100. The results that we have obtained are shown in Fig. 5, which demonstrates the ability of the approach to accurately infer correct decisions in the presence of the deceptive agents. In Fig. 5, 95% confidence intervals for the averages performance are also plotted. Observe that the scheme achieves a near-*optimal* index that asymptotically approaches the performance of the high performance services, i.e., $\theta = 0.8$.

5.2 Immunity to the proportion of deceptive agents

We now consider the problem of investigating how "immune" our system is to the percentage of deceptive agents. Figure 6 presents the average performance of the system (over all interactions) when the ratio of deceptive agents is varied. Observe that Fig. 6 are also reports the 95% confidence intervals for these averages performance indices. The reader will agree that the simulations results demonstrate that the scheme is truly "immune" to varying the proportions of fair and deceptive agents. In fact, even if all agents are deceptive (i.e., this is equivalent to a ratio of 100%), the average performance is stable and again achieves near-optimal values that approach the index of the high performance services, $\theta = 0.8$. In our opinion, this is quite remarkable!

5.3 Varying the spread between deceptive and fair agents

To further demonstrate the power of the scheme, we have considered the effect of varying the spread between deceptive and fair agents. To analyze this, in this experiment, 50% of the services provided were set to be "high performance" services. Figure 7 displays the average performance when p_f and p_d were set to the following pairs: (0.8, 0.2), (0.6,0.4), and (0.5,0.5). The reader should observe that as the spread between the fair and deceptive agents is decreased, the environment becomes increasingly more "difficult", rendering the task of differentiating between them to be more exacting. In the particular case where $p_f = 0.5$ and $p_d = 0.5$, the process of choosing the services is totally random, which results in a theoretical performance of 0.5, because,

$$P(\theta = 0.8) \times 0.8 + P(\theta = 0.2) \times 0.5$$
$$= 0.5 \times 0.8 + 0.5 \times 0.2 = 0.5.$$

However, in every case, the $AMPA$ seems to asymptotically attain near-optimal solutions.

5.4 Periodically changing service performance

To investigate the behavior of the $AMPA$ with performances which changed with time, we first considered the scenario when these changes were made periodically. Indeed, we achieved this by changing all the service performances periodically every 5,000 runs. Further, the changes

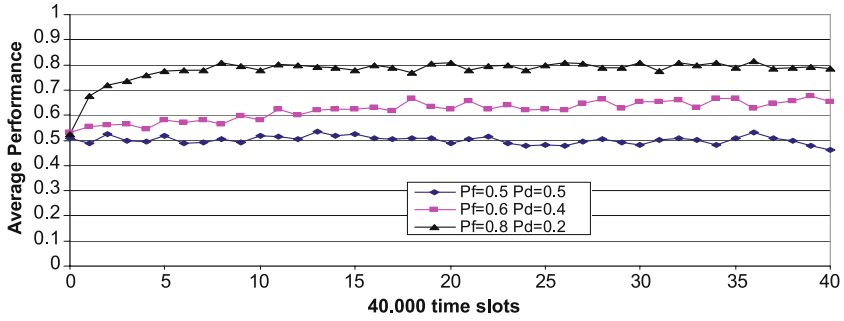


Fig. 7 The variation of the performance of the $AMPA$ with increasing spreads between the trustworthiness of deceptive and fair agents

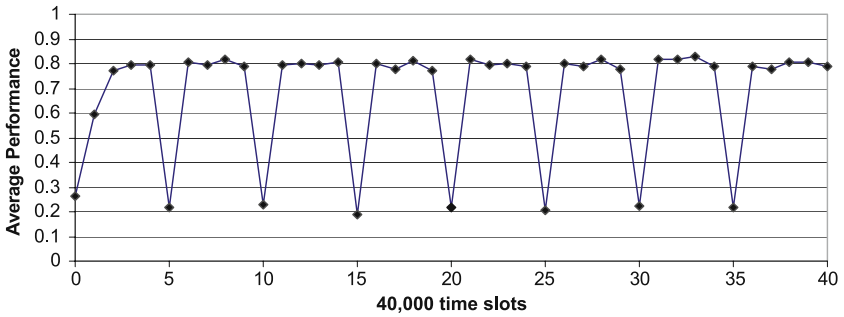


Fig. 8 The performance of the $AMPA$ with periodically changing service performances

were made "drastic", i.e., by inverting them from their prior values as per:

$$\theta_{l,new} = 1 - \theta_{l,old}.$$

In the simulation settings, we used the following parameters: There were 10% high performance services with $\theta = 0.8$, and 90% low performance services with $\theta = 0.1$. Further, we assumed that 15 of the reporting agents were deceptive, with $p_d = 0.2$, while 5 were fair agents with $p_f = 0.8$. The memory depth used for the LA was $M = 10$, and the length of the sliding window length was 100. From the results shown in Fig. 8, the reader will observe that the scheme is able to adapt favorably to such changes. Indeed, from Fig. 8, we notice that as the behavior of the services changed (i.e., at every 5,000th step), the subsequent access by agent u resulted in choosing a low performance service. However, the scheme was well able to adapt to that change, and that, rather rapidly, because of the state changes of the $AMPA$ and the sliding window approach. In fact, as the window slides, the reports related to older (i.e., "stale") service performances were discounted, and replaced by more recent reports, which, in turn, better reflected the current performance of the services. Again, we believe that the way by which the $AMPA$ tracks this change is quite remarkable.

5.5 Immunity to the ratio of low performance services

The next scenario we report is the $AMPA$'s immunity to the ratio of low performance services. Simulations results demonstrate that the scheme is immune to varying this ratio. In this case, we observe that for the system to achieve near-optimal performance, the time window should be chosen to be relatively large. In fact, it turns out that the agent in question still opts to choose high performance services even if their ratio is as low as 10%. Figure 9 depicts the average performance (over time) with variations in the ratio of low performance services. In our experiments, we utilized the following ratios: 10%, 30%, 50%, 70%, and 90%, and in the simulation, 15 of the reporting agents were deceptive with $p_d = 0.2$, while 5 of the agents were fair with $p_f = 0.8$. The reader will observe that the $AMPA$ converges to near-optimal performance in *every* single setting!

5.6 Varying the spread between high/low performance services

To further demonstrate the property of the $AMPA$ to be insensitive to the spread between high and low performance services, we conducted a set of experiments in which these parameters were varied. Figure 10 depicts the average performance when the indices of the high and low performance services were set to the following pairs: $(0.8, 0.2)$, $(0.7, 0.3)$, $(0.6, 0.4)$ and $(0.5, 0.5)$ respectively. In this experiment, 50% of the services provided were set to be "high performance" services. The reader will observe that as the spread decreased, it was (understandably!!) increasingly difficult for the scheme to accurately select only high performance services. Of course, in the limit when we encounter the particular case of $(0.5, 0.5)$, the process of choosing the services turns out to be totally random, which resulted in a theoretical performance of 0.5.

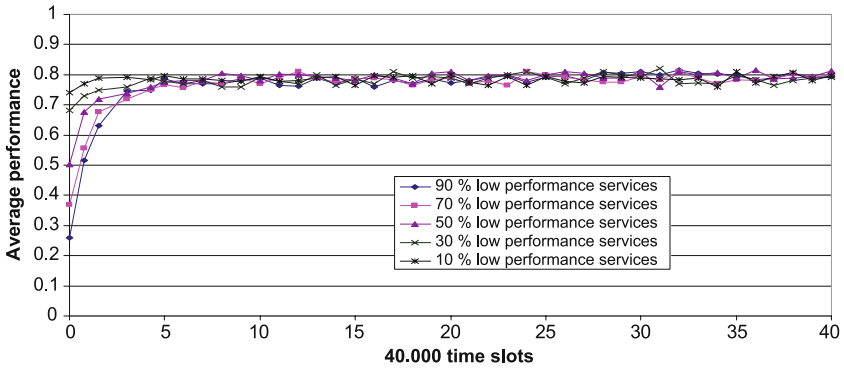


Fig. 9 The performance of the $AMPA$ with variations in the ratio of low performance services

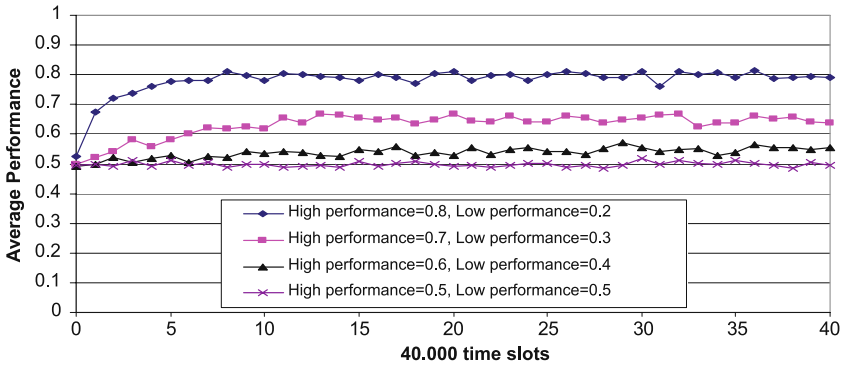


Fig. 10 The performance of the $AMPA$ as the spread between the high and low performance services is decreased

5.7 Effect of changing the memory size

The final result that we report confirms the basic theory of LA, which asserts that the performance of the machine increases with the size of the memory. To test the effect of the machine's memory, numerous experiments were conducted in a number of environmental settings. In this experiment, 50% of the services provided were set to be "high performance" services. The results obtained in every case was identical, namely, that the performance increased with the memory. Indeed, Fig. 11 illustrates that decreasing the memory depth from 5 to 2 results in a lower performance. Again, as anticipated by the theoretical results, a smaller memory undermines the quality of the partitioning process, making it difficult for the $AMPA$ to accurately differentiate between the deceptive and fair agents.

5.8 Experimental comparison

In this section, we compare[12] our proposed approach with two popular algorithms for dealing with deceptive agents, namely Yu and Singh's weighted majority method [28], and Sen and Sajja's approach [22]. The main idea of the algorithm in [28] is to assign a weight to every witness that reflects how credible he is. Before accessing a service, the agent in question, u, requests the predictions of the individual witnesses concerning the service performance. The witnesses convey their predictions to u in the form of belief functions [28]. After accessing the service, the agent in question updates the weight of every witness based on the result of interacting with the service. Deceptive agents will tend to submit inaccurate predictions and thus their relative weights will decrease over time. Similarly, the weights of fair agents will increase over time. An aggregated prediction, denoted as λ in [28], is computed by the agent in question as a weighted combination of the witnesses' predictions. In order to be able to compare our algorithm with the scheme proposed in [28], we were forced to add a *Service Selection Procedure* to Yu and Singh's weighted majority method, i.e., one that is analogous to the service selection procedure presented in our Algorithm 6. In the service selection procedure, whenever agent u desires to access a service, u creates a list L of the recommended services that have an aggregated prediction λ greater[13] than $\frac{1}{2}$.

In Fig. 12, we report the evolution of the average performance for our approach, and compare it with Yu and Singh's weighted majority algorithm. In the simulation settings, we used the following parameters: There were 10% high performance services with $\theta = 0.8$, and 90% low performance services with $\theta = 0.2$. Moreover, we assumed that 15 of the re-

[12] We are grateful to the anonymous referees who suggested such a comparison.

[13] Using the value $\frac{1}{2}$ as a decision threshold is commonly used in Weighted Majority Algorithms [11].

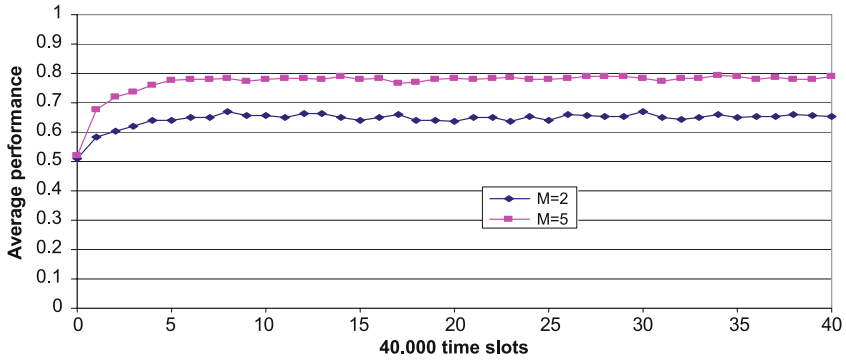


Fig. 11 The variation of the performance of the $AMPA$ by reducing the length of the memory

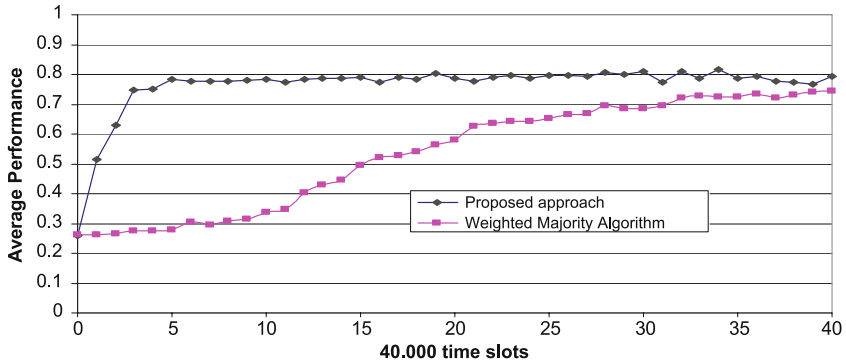


Fig. 12 The comparison of the performance of the $AMPA$ with the Weighted Majority Algorithm

porting agents were deceptive, with $p_d = 0.2$, while 5 were fair agents with $p_f = 0.8$. For the weighted majority algorithm, we adopted the number of episodes to be $H = 10$, as described in [28]. From Fig. 12, we remark that our proposed approach exhibits a faster convergence speed than the weighted majority algorithm. The slow convergence speed of the weighted majority algorithm can be explained by the gradual reduction of the weight of deceptive agents. This gradual modification of weights leads to the weighted average being "polluted" by the deceptive agents for a considerable amount of time, before their detrimental effect is filtered out.

Table 1 summarizes the average performance at time instant 20,000, for different ratios of deceptive agents, where the total number of agents was set to be 20. Table 1 shows that the average performance for the weighted majority algorithm monotonically decreases as the ratio of deceptive agents increases. This also demonstrates that the speed of convergence of the weighted majority algorithm decreases monotonically, as we increase the ratio of deceptive agents. On the other hand, Table 1 also reports that the learning speed of our proposed approach is not affected at all by the increased ratio of deceptive agents, thus demonstrating the superiority of our approach.

In addition to comparing our algorithm to the weighted majority algorithm, we have also evaluated another popular approach, namely the approach proposed by Sen and Sajja in [22]. In [22], the authors made use of a reinforcement learning technique to locate high performance service providers. A fundamental assumption in the latter approach is that the majority of agents are fair. When this assumption is true, the probability guarantee defined in [22] is obtained by querying all witnesses.[14] In Fig. 13, we report a comparison of the average performance under varying ratios of deceptive agents. In the experimental settings, 50% of the services were assumed to provide high performance. We observe that Sen and Sajja's algorithm suffers from a performance decline as the number of deceptive agents increases. As opposed to this, from Fig. 13, we observe that the average performance of our scheme remains near-optimal in every setting. It is worth noting that the robustness of our approach (when dealing with high ratios of deceptive agents) is not due to inverting the ratings of deceptive agents. In fact, we could exclusively base our predictions on the ratings of fair agents and discard the ratings of deceptive ones. However, intelligently combining the ratings from both fair and deceptive agents when evaluating the performance of a service, reduces the variance of the resulting aggregate.

5.9 Evaluation of computational efficiency

The issue of addressing the computational complexity of our solution is not easily answered. The real issue is that the problem itself is NP-Hard because it can be reduced to the

[14] In the interest of brevity, the full details of this approach are omitted. We refer the reader to [22] for further details about the scheme.

Table 1 A comparison of the performances of the $AMPA$ and the method due to Yu and Singh [28] after 20,000 time steps

Deceptive agents ratio	Weighted majority		Proposed approach	
	Average	Standard deviation	Average	Standard deviation
10%	0.786	0.013	0.796	0.013
20%	0.752	0.014	0.795	0.013
30 %	0.718	0.014	0.792	0.013
40%	0.701	0.014	0.790	0.013
50%	0.674	0.015	0.794	0.013
60%	0.635	0.015	0.793	0.013
70%	0.611	0.015	0.798	0.013
80%	0.581	0.016	0.797	0.013
90%	0.547	0.016	0.796	0.013

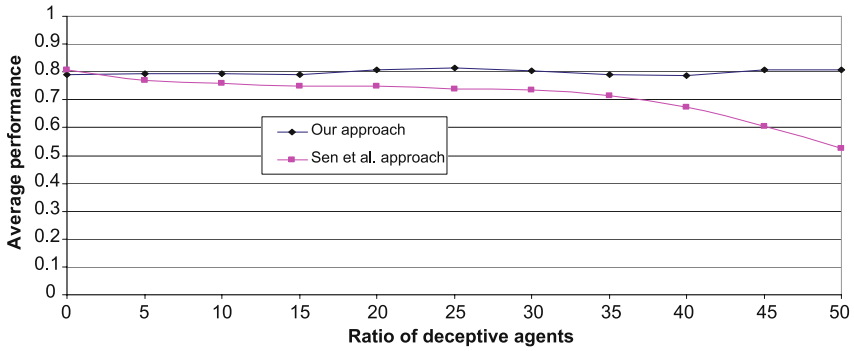


Fig. 13 The variation of the average performance of the $AMPA$ and the method of Sen and Sajja [22] when the ratio of fair agents is decreased

partitioning problem, and so, if we merely stated that at each step we required a *linear* number of operations, we would be presenting an unfair picture to the reader. Indeed, generally speaking, the time complexity of a fixed-structure LA depends on the depth of the memory, M, which plays an important role in determining the accuracy of the LA, while the consequent eigenvalues of the underlying chain (also dependent on M) would determine the rate of convergence. In practice, the LA is assumed to have converged when all objects are in (or close to) the most internal states in each partition. Even though one needs but a linear number of moves per iteration, to the best of our knowledge, we are not aware of any method that can be used to pre-determine the number of iterations required for a solution to reach these most internal states. Hence we present below, what we believe is the most appropriate (fair and scientific) method of reporting the complexity of the solution.

In order to assess the computational efficiency of our approach, we also measured the execution times on a laptop PC containing a 2.1 GHz Intel Core Duo CPU with 2 GB of RAM, running Windows XP 2002. All the algorithms were implemented using Java, and the code was compiled using the Sun Java compiler (javac). Observe that our main algorithm consists of two main procedures, namely the Service_Selection Procedure and the OMA_Based_Partitioning Procedure, whose detailed descriptions are respectively found in Algorithm 6 and Algorithm 7. In this perspective, we analyzed the computational efficiency of both these procedures separately.

Note that in all of the experiments, the execution time is expressed in milliseconds. Figure 14 depicts the evolution of the execution time of the Procedure OMA_Based_Partitioning when the size of the sliding window was varied from 10 to 100, and when we fixed the number of services to 100. From Fig. 14, we observe that the required execution time for the OMA_Based_Partitioning increases linearly as the size of the sliding window increases. Note that we achieve high accuracy with relatively small window sizes, and it is not unreasonable to reckon that this linear increase in computation time is rather insignificant in real-world applications.

In the rest of this section, we evaluate the performance of the Procedure Service_Selection. Here, we measured the execution times by varying the number of services and for varying sliding window sizes. In Fig. 15, we fixed the window size to be 100 and varied the number of services from 10 to 100. From the figure, we note that the execution time of Procedure Service_Selection increases almost linearly as we increase the number of services, which is also quite commendable!

Similarly, in Fig. 16, we display the results when the number of services was fixed to be 100, and we varied the

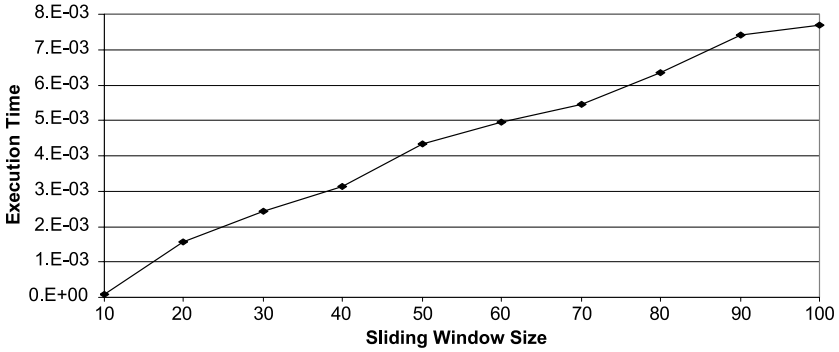


Fig. 14 Execution time of the Procedure OMA_Based_Partitioning with varying Sliding Window sizes

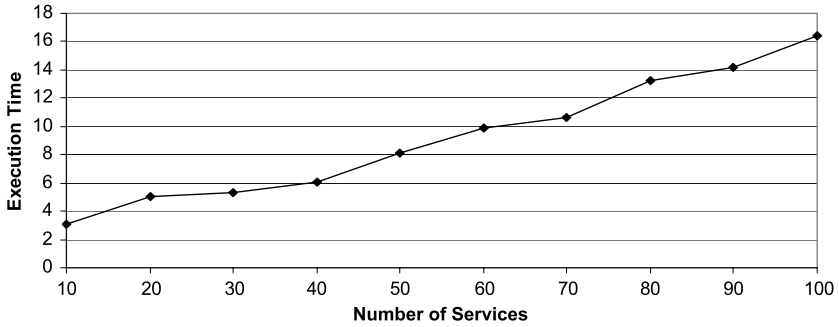


Fig. 15 Execution time of the Procedure Service_Selection with varying number of services

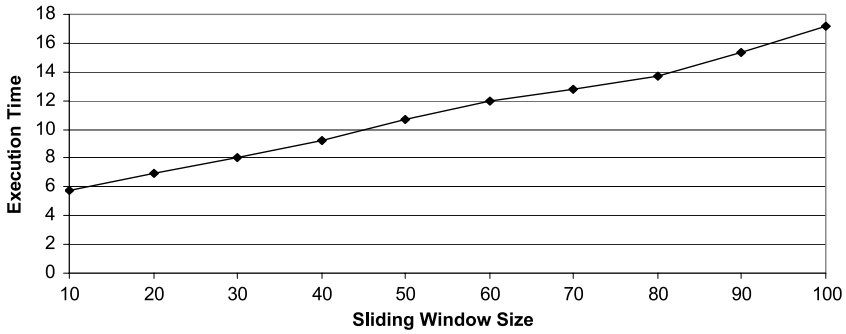


Fig. 16 Execution time of the Procedure Service_Selection with varying Sliding Window sizes

window size from 10 to 100. From this figure, we note that by increasing the window size, the execution time increases as well, again, almost linearly!

5.10 Utilizing the ATPP to real-world applications

The ultimate intention of this paper is to have the algorithm functional in a real-world device. The problems with testing and incorporating it on real-world applications are many and listed below:

1. First of all, it is extremely hard, if not impossible to get real-life data on which we can test the algorithm. Indeed, we would like to perform a real-world study with "real" good and bad services, and with "real" people taking the role of truth-tellers and liars. However, the question, really, is one of finding service providers and users who will willingly participate in such an experiment, and the logistics of this exercise is unsurmountable.
2. Secondly, even if are able to "recruit" service providers for this task, it will be impossible to find real-life users who will serve as "ballot-stuffers" or "badmouthers".
3. Even if people serve in these capacities, the results of the tests should, in actuality, be verified using psychological and cognitive criteria. Unfortunately, such a study would have to be carefully designed in order to provide reliable results, and would require human resources and knowledge in the latter domains that remain outside the scope of the present project. We respectfully submit that his is a multi-disciplinary project in its own right.
4. From an optimistic perspective, we remark that by introducing a large degree of noise in our simulations, we are able to "stress" the schemes quite severely, thus mimicking the "nuisances" of the real world. This is what we have done in this paper.
5. In this regard, we have included a comparative evaluation with other popular approaches for dealing with deceptive agents ratings, namely, Yu and Singh's weighted majority

method [28] and Sen and Sajja's reinforcement learning approach [22] (see Sect. 5.8). From these evaluations, we believe that it is clear that our scheme provides the same kind of functionality as what is required from practical cases. One should, however, note that our scheme outperforms the latter schemes in the experiments.

Finally, we conclude this section by remarking that from a more realistic viewpoint, now that the problem setup has been clarified, we believe that it is, presently, much clearer how the scheme fits into a practical case. Thus, a more restricted real-world study is planned as a next step in cooperation with *Ericsson Research*, where we intend to build a mobile phone-based prototype of the scheme presented in this paper, with a focus on learning from a real-life social network. Briefly stated, in this prototype each mobile phone will be equipped with an instance of ATPP. Furthermore, the ratings users submit about services must be accessible by each mobile phone, preferably based on lazy propagation of ratings on a per need basis. Thus, as ratings of other users are observed by a given instance of ATPP, and the respective user obtains direct experience from his own interaction with services, ATPP updates its state accordingly, as demonstrated in the simulations of this paper. Another possible practical application of our algorithm is stated in the paper by Sen and Sajja [22] where user agents need to select processor agents to achieve processor tasks. This avenue of research is also currently being investigated.

6 Conclusions

In this paper, we have considered an extremely pertinent problem in the area of "Reputation Systems" (RSs), namely the one of identifying services of high quality. Although these RSs offer generic recommendations by aggregating user-provided opinions about the quality of the services under consideration, they are prone to "ballot stuffing" and "badmouthing" in a competitive marketplace. Clearly, such unfair ratings may degrade the trustworthiness of RSs, and additionally, changes in the quality of service, over time, can render previous ratings unreliable. In this paper, we have presented a novel solution for the problem using tools provided by the family of Learning Automata (LA). Unlike most reported approaches, our scheme does not require prior knowledge of the *degree* of any of the above mentioned problems associated with RSs. Instead, it gradually learns the identity and characteristics of the users which provide fair ratings, and of those who provide unfair ratings, even when these are a consequence of them making unintentional mistakes.

Comprehensive empirical results show that our LA-based scheme efficiently handles any degree of unfair binary ratings. Furthermore, if the quality of services and/or the trustworthiness of the users change, our scheme is able to robustly track such changes over time. The paper also contains a detailed comparison of the method with the state-of-the-art. Finally, the strategy is ideal for decentralized processing, and so, as such, we believe that our LA-based scheme forms a promising basis for improving the performance of RSs.

A possible extension of our work is to develop the analogous methodology for continuous reports instead of boolean. Also, in this work, every agent in a social network can communicate with all other agents. In practice, though, most of the time, one agent may resort to his friends for partial information of available services. The question of how we can devise a solution for the service reputation effectively and efficiently (in this setting) is a research task in itself, and is an interesting problem for future work.

References

1. Agache M, Oommen BJ (2002) Generalized pursuit learning schemes: new families of continuous and discretized learning automata. IEEE Trans Syst Man Cybern, Part B, Cybern 32(6):738–749
2. Altincay H (2006) On the independence requirement in Dempster-Shafer theory for combining classifiers providing statistical evidence. Appl Intell 25:73–90. doi:10.1007/s10489-006-8867-y
3. Buchegger S, Le Boudec J (2004) A robust reputation system for P2P and mobile ad-hoc networks. In: Proceedings of the second workshop on the economics of peer-to-peer systems
4. Chen M, Singh JP (2001) Computing and using reputations for internet ratings. In: Proceedings of the 3rd ACM conference on electronic commerce, Tampa, FL, USA. ACM, New York, pp 154–162
5. Dellarocas C (2000) Immunizing online reputation reporting systems against unfair ratings and discriminatory behavior. In: Proceedings of the 2nd ACM conference on electronic commerce, Minneapolis, Minnesota, USA. ACM, New York, pp 150–157
6. Despotovic Z, Aberer K (2004) A probabilistic approach to predict peers performance in P2P networks. In: Cooperative information agents VIII, pp 62–76
7. Faceli K, de Carvalho A, Rezende S (2004) Combining intelligent techniques for sensor fusion. Appl Intell 20:199–213. doi:10.1023/B:APIN.0000021413.05467.20
8. Gale W, Das S, Yu C (1990) Improvements to an algorithm for equipartitioning. IEEE Trans Comput 39(5):706–710
9. Jøsang A, Ismail R, Boyd C (2007) A survey of trust and reputation systems for online service provision. Decis Support Syst 43(2):618–644
10. Li X, Dai X, Dezert J, Smarandache F (2010) Fusion of imprecise qualitative information. Appl Intell 33:340–351. doi:10.1007/s10489-009-0170-2
11. Littlestone N, Warmuth MK (1994) The weighted majority algorithm. Inf Comput 108(2):212–261
12. Mayur Datar M, Gionis A, Indyk P, Motwani R (2002) Maintaining stream statistics over sliding windows. SIAM J Comput 31(6):1794–1813
13. Mundinger J, Le Boudec J-Y (2008) Analysis of a reputation system for mobile ad-hoc networks with liars. Perform Eval 65(3–4):212–226

14. Narendra KS, Thathachar MAL (1989) Learning automata: an introduction. Prentice-Hall, New Jersey
15. Obied A, Alhajj R (2009) Fraudulent and malicious sites on the web. Appl Intell 30:112–120. doi:10.1007/s10489-007-0102-y
16. Oommen B, Ma D (1988) Deterministic learning automata solutions to the equipartitioning problem. IEEE Trans Comput 37(1):2–13
17. Oommen BJ, de St Croix EV (1996) Graph partitioning using learning automata. IEEE Trans Comput 45(2):195–208
18. Oommen BJ, Fothergill C (1993) Fast learning automaton-based image examination and retrieval. Comput J 36(6):542–553
19. Oommen BJ, Ma DCY (1988) Deterministic learning automata solutions to the equipartitioning problem. IEEE Trans Comput 37(1):2–13
20. Oommen BJ, Ma DCY (1992) Stochastic automata solutions to the object partitioning problem. Comput J 34(6):A105–A120
21. Poznyak AS, Najim K (1997) Learning automata and stochastic optimization. Springer, Berlin
22. Sen S, Sajja N (2002) Robustness of reputation-based trust: boolean case. In: Proceedings of the first international joint conference on autonomous agents and multiagent systems: part 1, Bologna, Italy. ACM, New York, pp 288–293
23. Shapiro C (1982) Consumer information, product quality, and seller reputation. Bell J Econ 13(1):20–35
24. Thathachar MAL, Sastry PS (2002) Varieties of learning automata: an overview. IEEE Trans Syst Man Cybern, Part B, Cybern 32(6):711–722
25. Thathachar MAL, Sastry PS (2003) Networks of learning automata: techniques for online stochastic optimization. Kluwer Academic, Boston
26. Tsetlin ML (1973) Automaton theory and the modeling of biological systems. Academic Press, New York
27. Whitby A, Jøsang A, Indulska J (2005) Filtering out unfair ratings in bayesian reputation systems. J Manag Res 4(2):48–64
28. Yu B, Singh MP (2003) Detecting deception in reputation management. In: Proceedings of the second international joint conference on autonomous agents and multiagent systems, Melbourne, Australia. ACM, New York, pp 73–80
29. Yuan W, Guan D, Lee Y-K, Lee S (2010) The small-world trust network. Appl Intell 1–12. doi:10.1007/s10489-010-0230-7
30. Zacharia G, Maes P (2000) Trust management through reputation mechanisms. Appl Artif Intell 14(9):881–907