SCHEDULING REPUTATION MAINTENANCE IN AGENT-BASED COMMUNITIES USING GAME THEORY

Mohamed Amine M'HAMDI

A thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science (Software Engineering)

Concordia University

Montréal, Québec, Canada

December 2011

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: Mohamed Amine M'Hamdi

Entitled: Scheduling Reputation Maintenance in Agent-based Communities Using Game Theory

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Software Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. V. Haarslev Chair

Dr. B. Fung Examiner

Dr. P. Grogono Examiner

Dr. J. Bentahar Supervisor

Approved by _____
Chair of Department or Graduate Program Director

_____
Dean of Faculty

Date December 20, 2011

Abstract

Scheduling Reputation Maintenance in Agent-based Communities Using Game Theory

Mohamed Amine M'HAMDI

In agent-based systems, agents can be organized within groups, called communities, where members are providing similar or complementary services. An example of such systems is agent-based communities of web services, where web services are abstracted as rational agents and empowered with decision making capabilities and can interact with each other. Managing reputation of each agent and of the whole community is a key issue towards securing this type of systems, where a controller agent is designed to observe and check the behavior of each member to update and maintain the system's reputation. Scheduling the check (i.e. maintenance) by deciding about the moments where the check has to be done is still an open problem. Because it is highly expensive, maintenance cannot be done every moment or based on small history of agents' behaviors. We propose in this thesis a scheduling algorithm that helps the controller agent improve the quality of the reputation mechanism, which increases the trust value of users toward the community. The proposed algorithm is based on a class of games called Bayesian Stackelberg. Our Bayesian Stackelberg game is designed between the controller agent and community members. We simulate and compare the efficiency of our algorithm with other stochastic techniques, namely uniform, normal and Poisson distributions. This research draws the lines for future work in the subject of optimizing reputation mechanisms through maintenance in different time intervals.

Acknowledgments

I would like to express my deepest gratitude to my supervisor Dr. Jamal Bentahar. His guidance and continuous feedback made my research work an awarding and fruitful experience.

I would also like to thank my research mates: Babak, Giti, Mahsa, Maziar, Mohamed, Rim, Sina, and Wei. The success of this research and this thesis would not have been possible without them. My thanks go to my professors from whom I learned research spirit and professionalism in the subjects taken.

Special gratitude goes to my parents for their continuous support through my studies far away from home. My gratitude goes to my grandmother, my sister, and my brother for their moral support and encouragement.

I would also like to thank my friend Oualid for the living experience in Canada. I also would like to thank my friends that I met at Concordia that space is not enough to mention all of them for the time we spend together.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces the context, motivation, problem definition, research questions that we aim to answer, and summary of contributions of this thesis. We conclude this chapter with an overview of the thesis organization.

1.1 Context of Research

Trust and reputation have gained tremendous research interest in the area of multi-agent systems. In such open systems, social aspect implies that tasks accomplishment depends on cooperation and delegation of those tasks to other agents. Multi-agent system by definition means that agents need intra and inter communities cooperation from other agents to achieve and pursue their designed objectives. However, selection process means that agents need reliable and credible information about other agents, specifically with respect to reputation.

Agents are selfish and rational. Hence, they have incentives to get higher profit with lower expense and energy when accomplishing their tasks. Thus, it is possible for such agents to act maliciously whenever they see the opportunity to do so; whenever they are invited to give feedback about other agents for instance. This potentially decreases the reputation of a community of agents since its feedback file is a fundamental reference to other agents. In other terms, if these agents are

making decisions based on previous unreliable feedback, then how can we make communities more trustworthy?

We define an agent-based community as a virtual organization of autonomous rational agents having incentives to interact with each other, share information and expertise, and collaborate [2]. In agent-based service communities, customers interact with providers (community members) based on reputation. Reputation values represent customers' aggregate views and perceptions about these agent members in terms of satisfaction. Controller agent is a special agent in the community, which is responsible for observing and monitoring the behavior of each agent in the same community [3]. This particular agent has also the responsibility to attract agents to this community and dismiss bad and malicious members [4], [5]. In such settings, autonomous agent providers have incentives to join and leave these communities based on their reputation since joining good reputation communities will lead to more profit and increase in payoffs. Therefore, it is up to the controller agent to manage its reputation mechanism by performing follow up of the community members, through feedback file for instance.

1.2 Motivations

As argued in [6], trust can be computed using different gathered parameters such as previous feedback from historic interaction and referral agents impressions and satisfactions. Many of the trust frameworks differ in defining the trust parameters as well as the coefficient of each parameter. They also rely on updating agents' beliefs on a regular basis based on the agents' behaviors to keep reputation and trust of each agent, and so the whole system updated.

These Trust frameworks rely heavily on frequent updates in order to balance agents' beliefs between direct and indirect trust assessments[1] on one hand [7], and dynamic update of these beliefs [6] on the other hand. As belief update is crucial in any trustful and reliable system, it should be implemented properly and efficiently. In fact, as argued in [8], predictability is a major and complex

[1]Direct assessment means agents know each other so they can evaluate each other based on previous direct interaction. Indirect assessment uses information provided by third parties such as referees.

problem in belief update. A plausible explanation is that agents are autonomous and intelligent entities so they can learn over time the patterns used to perform the update and act maliciously so that their trust values get increased after each belief update. The problem we are facing is scheduling maintenance updates so that the maintenance moments are hard to be predicted. In this context, malicious agents that propagate fake feedback have the chance to observe the scheduling process in order to determine when to counterattack, or collude. This means, if update scheduling is deterministic, it is very likely that agents know when to act maliciously and benefit from such vulnerability scheme. In this thesis, the term maintenance update refers to the update activity performed by the controller agent to adjust the aggregated reputation value of each autonomous agent in a community. This definition is different from the meaning of the maintenance phase in the software life cycle.

1.3 Problem Definition

It has been shown in [9] that under periodic maintenance, controller agent is able to alleviate collusion and discourage agents to act maliciously where this controller updates its trust belief about each agent in the community. This maintenance allows increasing the reliability of the reputation mechanism. Nevertheless, it is likely that autonomous agents can take advantage of predictable maintenance phases of the controller agent (i.e. predictable check moments). Thus, the problem is how to help the controller agent choose moments in time during a certain interval in order to perform the check by monitoring the agents' behavior, and so the maintenance update to reduce malicious acts as much as possible. The key problem is then making the maintenance update phases hard to predict by the agents.

In fact, scheduling is a critical and open issue in open multi-agent systems. The controller's main goal and responsibility would be to perform a scheduling process following an algorithm to decide about its maintenance updates.

Agents can decide when it is in their best interest to act maliciously, or collude, in order to

mislead the controller and stay in the community. In other words, driven by their motivations, malicious and bad members aim at staying in the community and generating profits for services they provide by deciding to deceive the controller agent with fake feedback. To address this problem, two constraints need to be emphasized in designing our scheduling algorithm. The first one is that the controller agent is not willing to perform the monitoring and check at every single moment. This will require many resources and cause overhead in the system and lack of efficiency in terms of overall performance as each agent should be monitored all the time. The second constraint is related to the number of feedback needed to perform the update; it is inefficient to perform maintenance and reputation adjustment based on very few feedback. Furthermore, it is not recommended for the controller agent to perform one single maintenance check in a large time interval. This will motivate agents to collude in that time interval and will not help the controller agent minimize the collusion scenarios in the community.

1.4 Research Questions

In an attempt to optimize reputation mechanism in order to allow for enhanced trust between agents, our aim is to answer the two following research questions:

- What would be a better scheduling algorithm for the controller agent to follow in an open multi-agent system in both static and dynamic environments in order to minimize collusion and malicious acts that damage reputation and trust mechanisms?
- As far as predictability is concerned, how can we establish this scheduling algorithm that should perform better than stochastic randomization techniques?

1.5 Summary of Contributions

Our first contribution is providing game theoretic foundation for agent communities and associated communication between the controller and community members. This foundation has a key advantage; it models the complexity of selfish interactions with respect to payoff function, which simplifies the motivation function for malicious agents. The second contribution is providing a scheduling strategy based on the game theoretic foundation and an associated model. Typically, our controller agent will be performing the task of determining when to process the scheduling during different time intervals. We present a scheduling algorithm that applies optimization techniques to determine the next moments in time to perform the maintenance update. This algorithm uses and extends a technique proposed originally in [8].

We conducted many simulations that compare our combined game theoretic model with the proposed strategy against common and stochastic scheduling schemes. We analyze the performance of the controller agent under different scheduling algorithms in terms of fake feedback propagation, controller's payoff, and predictability output.

1.6 Thesis Overview

The rest of this thesis is organized as follows. Chapter 2 presents the background as well as related work with respect to the improvement of trust and reputation in multi-agent systems. Chapter 3 discusses our proposed game theoretic scheduling algorithm that the controller agent will use in order to perform maintenance update. Chapter 4 presents simulation results comparing our solution with other periodic scheduling alternatives and shows how our algorithm performs with respect to the fake feedback rate, controller's payoff, and predictability output. Chapter 5 concludes this thesis by outlining main research's output and setting up the path for future work.

Chapter 2

Background and Related Work

This chapter demonstrates an understanding of agents, multi-agent systems, and an understanding of both trust and reputation in the context of distributed artificial intelligence. After that, we present research work that has been performed in order to enhance trust frameworks and reputation mechanisms in multi-agent systems.

2.1 Agent

As defined in [10] (page 15), an agent is a computational entity (i.e. a program) that runs on computing devices, has autonomous control over its behavior, and can act without the intervention of humans and other systems. It has its own attributes, believes, and behaviors to allow it to act individually and socially.

It is very important to compare and contrast an intelligent autonomous agent with an object in object-oriented paradigm. In terms of similarities, they both encapsulate attributes and operations on those attributes; and they both communicate via message passing. Differences emerge in three dimensions: autonomous, flexibility and control. Unlike an object where methods can be invoked upon one another, an agent decides whether or not to perform an action requested from another agent or initiated by itself. In general, objects do it because they are asked to; agents do it because

they want to. Objects need to be invoked to act; agents however are known to be reactive, proactive, and social entities having autonomous behaviors. As far as control is concerned, it is possible for an object to create a thread of another object. It is not however possible to do that among agents because agents are considered to have their own thread of control.

Differences among autonomous agents are not limited to design objectives; they are also concerned by agents architectures and their logic abstractions. According to [11] (page 42), there are four types of concrete architectures for intelligent agents:

- Logic-based architecture: this architecture places logical reasoning and deduction in the center or the heart of the architectural aspect of agents. For example, a vacuum cleaner that considers a certain space such as room as grid-like and operates based on sensing a dirt in a given grid and performs the cleaning is an logic based implementation of such agent.
- Reactive architecture: this architecture maps situations to actions in order to determine what to do next.
- Belief-desire-intention architecture: this agent architecture encapsulates internal states and characteristics that represent beliefs, desires, and intentions of agents in order to decide which actions to perform; and finally
- Layered architecture: agents of this type are meant to ease and overcome the problems of previous architectures. This solution consists of building agents in multi-layer models in order to deal with different types of behaviors such as cooperation and planning.

Let us consider a simple example of an agent application, heating to be specific. Figure 1 shows an agent's behavioral cycle that goes through three steps. The first one is perception. In this stage, agent receives information through its sensors. Once information is received, agent makes perception about what the environment is like. For example, the room temperature is under 75 degrees Fahrenheit. The next stage is decision making, or what to do. The agent maps the corresponding decision to perform under the specified conditions determined in the perception stage, which is based on the if-then rule. In the heating case, "turn the heater on if the temperature is

under 75 degrees" is considered as the rule to follow. The next and final stage is the action stage in which this agent performs the deduced strategy to follow through actuators, such as turning on or off the heater.

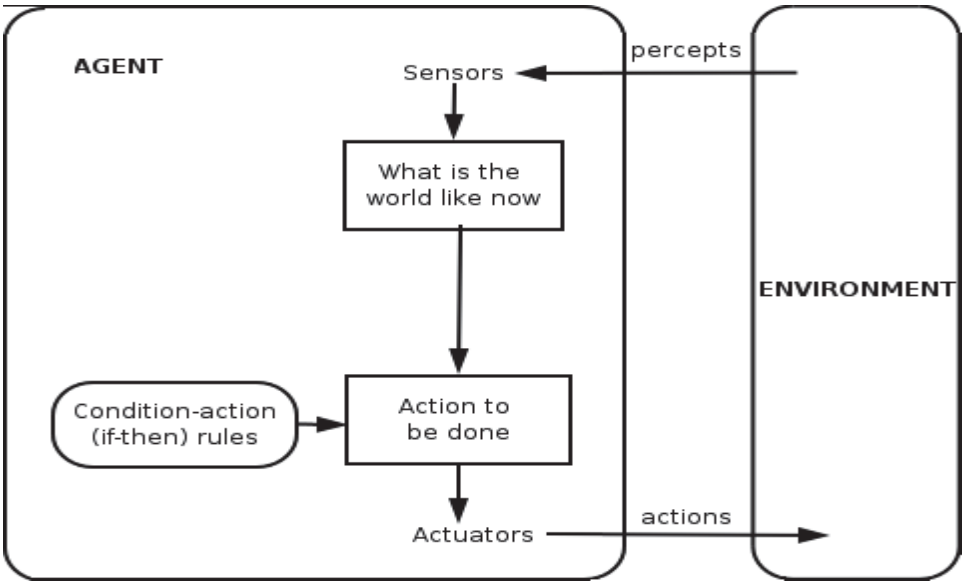

Figure 1: Simple reflex agent (from [1])

The basis of our agent's action or reaction is the information received from the temperature sensor in one area space. It assumes that information about this environment is to a great extent close to complete. Reality of things can get complicated. For example, if we want to have optimality of temperature across different independent rooms equipped with independent thermostats on each of them, we will be faced with at least three challenges. The first one is that we need information on each room separately. The second challenge is that this intelligent system needs to optimize the energy consumption of electricity towards reducing costs. The third challenge is to have independencies of decisions in case of system failure; in other words, we do not want a centralized system that if failed, all the thermostats of the rooms will not be able to function properly. It is logical to think that one agent implementation is not a feasible solution to the problem because of third challenges. A new approach is needed to answer and satisfy this intelligent technological need.

In this section, we presented in short the agent's design problem that can be summarized as building computational entities that can independently and autonomously act in order to successfully

carry out tasks that are being delegated. We also introduced that one agent design solution is not always an optimal and satisfying solution. We need to consider the implementation of many agents interacting with each other. The next section introduces the viewpoint of multi-agent systems that shows the role played by agents.

2.2 Multi-Agent Systems

This section presents a general viewpoint of multi-agent systems and how one agent can act partially on its environment. In other words, agents act autonomously based on dedicated goals and objectives. This brings the problem of building agents to cooperate, coordinate, and negotiate to carry out tasks successfully when agents do not have the same interest and/or goals [10] (page 109).

In short, multi-agent systems can be defined as systems mainly composed of intelligent autonomous agents that are massive-scale, openly distributed, capable of effectively and autonomously deploying and redeploying computational resources to solve large computational problems such as huge data sets and complex processing requirements [11]. For example, consider a communication network that is automated using autonomous agents. One use of the agents is to achieve automatic and dynamic load balancing, high scalability, and self-healing networks to overcome congestion and optimize network resources in different dynamic circumstances such as node failures.

One very useful application of multi-agent systems is to help predict coming issues and problems of applying certain algorithm in a particular situation. In short, multi-agent systems to a great extent can be considered as artificial virtual societies that abstract either individuals' or groups' behavior in a social context. This approach has the purpose to simulate interactions given application and deployment of mechanisms and rules to see what problems could raise by such application to better serve a given community with a better solution.

Multi-agent systems are not only a collection of agents to perform actions towards achieving one designed goal. E-commerce is an example of collection of agents where interests and goals are divergent and in many cases contradictory. It is up to each agent acting as consumer to decide

which other agent as provider to choose to satisfy its needs. E-bay for instance is a major online service interaction where agents request services from other agents. However, service selection is heavily based on previous historical interaction between these agents and/or other trusted agents principally because agents cannot certainly predict the future of such interactions.

To maintain a sound reputation mechanism among agents, it has been proposed to conceptualize agents and gather them within communities. In this case, it is assumed that agents do not receive requests unless they are part of a community. Figure 2 explains the community conceptual architecture with special emphasis on communities of web services CWS [12]:

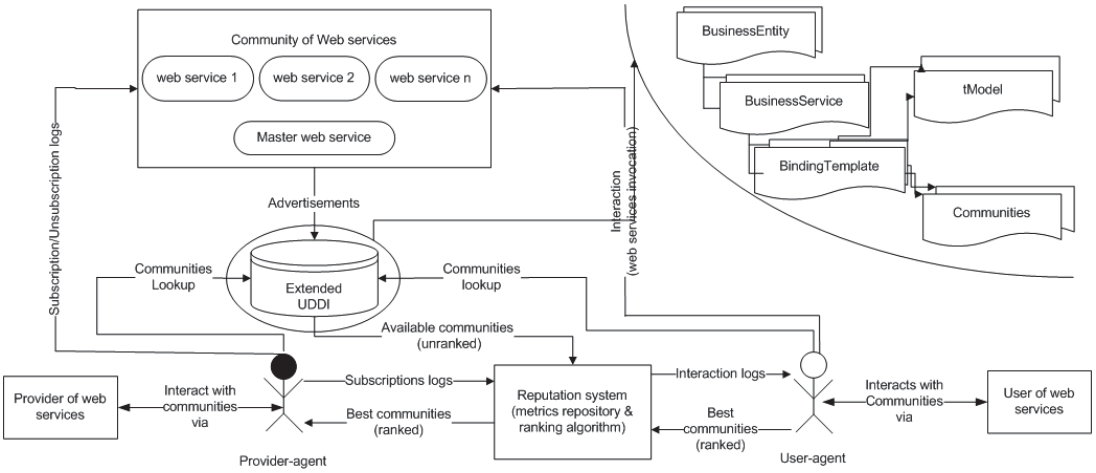

Figure 2: Architecture of reputation-based CWSs

The components together with their performance represented in Figure 2 are explained below:

- **User agent**: it is a proxy between user web service, extended UDDI, CWS and reputation system.
- **Provider agent**: like user agent, provider agent is a proxy between provider web service, extended UDDI, CWS and reputation system.
- **Master agent**: it is a community's representative in the sense that it hires, fires, and manages incoming requests to community members or web services. Besides, this agent has the objective to maximize the community's profit by trying to increase incoming requests. Hence, community's reputation improvement is very important. In this context, the controller agent

can be the master agent.

- **Extended UDDI**: In this registry, we restricted the access of user and provider agents to the list of master agents, whereas master agents have access to the UDDI registry of web services members.
- **Reputation system**: Because web services in communities offer similar and competing services, they need to be evaluated. Therefore, both user and provider agents need to gather operational data that reflect performance metrics.
- **Controller agent**: This agent is assigned in order to take under surveillance the logging file, which consists of reputation values such as response time log and success log, and updates the assigned reputation of communities. In addition, this controller agent's main responsibility is to remove the cheated or poorly performed agent that supports particular community.

One question that raises at this point is why one agent need to interact with another agent in such context? The answer is task delegation. As we have seen in the previous section, agents differ not only in their objectives but also in their design architecture, which implies that for different needs, agents have to consult with other agents, which are believed to have the expertise to accomplish their tasks. Therefore, agents need to rely on other agents to achieve their respective goals. The problem is which agent to choose and what are the characteristics in general context that drives these autonomous, flexible, and self-controlled entities to choose particular agents from others. This brings the question of trust and reputation.

2.3 Trust and Reputation

In this section, we define trust and reputation and their significance in multi-agent systems concerning the social aspect of modeling these systems. We also mention the relationship between the two concepts.

2.3.1 Reputation

Reputation can be defined as the social opinion or assessment of a group towards a person, a group, an organization, or an agent using different evaluation criteria. It is a significant factor in many fields, such as education, business, and online communities. For the sake of simplicity but without losing generality, according to Merriam Webster [13] dictionary, it is "overall quality or character as seen or judged by people or in general a recognition by other people of some characteristic or ability".

It is extensively used in online trading such as eBay by customers to get information about other agents. Since consumers in general behave based on their perceptions of services that are determined by factors as needs, desires, believes, and images, they need to know if it is very likely to perform deals with these services or autonomous agents that deliver those services. This is why it is heavily emphasized by mechanisms of social control to decide whether certain agents should stay or leave a community.

2.3.2 Trust

Trust can be defined as the trustfulness of a trustor or the extent to which the trustor is willing to take the risk of trust being abused by the trustee. This understanding is shared by Merriam Webster [14] dictionary that explicitly defines trust as "assured reliance on the character, ability, strength, or truth of someone or something". In other words, when agents decide to invest in other agents concerning services they cannot perform, they need to know up to which extent they can be satisfied. As pointed out in [6], in the context of open multi-agent systems with no central control and where agents are known to be autonomous, trust is essential to initiate interaction between agents.

In general, there are two types of trust to be conceptualized [15]:

- Individual-level trust: as the name implies, this type of trust models agents beliefs about the honesty and reciprocity of other agents' interactions

- System-level trust: as the name indicates, this type suggests that autonomous agents in a certain environment are bound by protocols, rules, and mechanisms to respect. These rules regulate that specific environment, which is generally known as community.

According to [15], using the correlation between system-level trust and user-level trust, it is logical to deduce that agents can analyze the strategies to be used. Such analysis can help them gather information through various means about potential agents; therefore, they can decide for themselves which agents are trustworthy and which are untrustworthy. In parallel, these agents are being imposed conditions that would cause them to lose utility if they do not comply and use the systems capabilities to propagate their reputation to promote future interactions with other agents in the community or jeopardize these interactions that can be translated as penalties. The two types of trusts are illustrated in Figure 3.

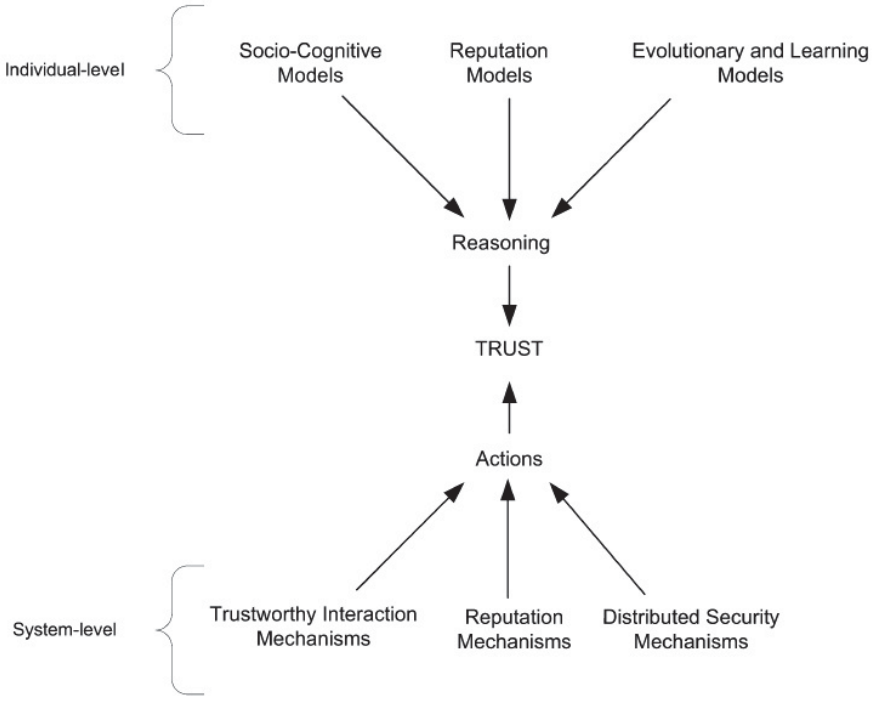

Figure 3: A classification of approaches to trust in multi-agent systems

2.4 Game Theory and Optimization

In this section, we define the notions and theories we will use in our framework, namely game theory and linear programming.

2.4.1 Game Theory

Game theory can be defined as a compact model of interaction between two or more players. As explicitly mentioned in [16] (page 337): "By a game we mean roughly a situation of conflict between two or more people, in which each contestant, player, or participant has some, but not total, control over the outcome of the conflict". Mentioning the word outcome leads us to the notion of the utility function, which is used to define the payoffs of the game [10] (page 109):

Definition 2.4.1 *A utility function U_i for agent i is a function $U_i : O_i \to \mathbb{R}$ mapping to $\mathbb{R}$ a set of outcomes (or states) O_i ($O_i = \{o_1, o_2, ..., o_n\}$) that i has preference over. Preference theory suggests that an agent i prefers the outcome o_x over o_y if $U_i(o_x) > U_i(o_y)$.*

Since players can change their actions or strategies, we need to define the meaning of both pure strategy and mixed strategy. In addition, we differentiate between the two as presented in [17] and [16] (page 354):

Definition 2.4.2 *A strategy is a complete contingent plan that defines the action an agent will select in every distinguishable state of the world.*

Definition 2.4.3 *A mixed strategy for a player P_1 is a vector $X = (x_1, x_2, ..., x_m)$ of nonnegative real numbers satisfying the condition $x_1 + x_2 + ... + x_m = 1$, with the interpretation that P_1 plays strategy s_i with probability x_i, $1 \leq i \leq m$.*

Players' behaviors are generally analyzed under both one shot games and consecutive round games. One of the important notions of game theory towards this analysis is Nash equilibrium that gives us a good idea in many games about the steady state in those repetitive games. The following

defines Nash equilibrium given two players i and j playing strategies s_1 and s_2, respectively [10] (page 113):

Definition 2.4.4 *In general, we will say that two strategies s_1 and s_2 are in Nash equilibrium if:*

1. *under the assumption that agent i plays s_1, agent j can do no better than play s_2; and*
2. *under the assumption that agent j plays s_2, agent i can do no better than play s_1.*

In many cases, achieving better solution concepts means identifying Pareto optimal situation in a game. Pareto efficiency is defined as follows:

Definition 2.4.5 *An outcome is Pareto efficient if there is no other outcome that improves one player's utility without making somebody else worse off.*

2.4.2 Linear Programming

Linear programming can be defined as a "mathematical technique to maximize or minimize given function of variables that are defined under constraint conditions" [16] (page 3). For example, consider the famous diet problem in which we have to determine an adequate diet for a person to sustain himself/herself while paying minimum cost. Under concrete perspective, let's consider the case where the available food in local store are meat steak, eggs, and potatoes with the following nutritional information:

	Per unit of meat steak	Per unit of egg	Per unit of potatoes	Requirements
Unit of carbohydrates	4	2	1	10
Unit of vitamins	8	9	8	21
Unit of proteins	2	3	1	9
Unit cost	45	25	33	

Table 1: Diet optimization problem

Our objective behind this modeling is simple; we need to find out how many units we need to buy of meat steak, eggs and potatoes that meets our minimum requirements with minimum cost. Let x_1, x_2, and x_3 be the number of units of meat steak, eggs, and potatoes respectively. This implies that we need to formulate our problem in the following way:

$$
\begin{aligned}
min \quad & 45x_1 + 25x_2 + 33x_3 \\
s.t. \quad & 4x_1 + 2x_2 + 1x_3 \geq 10 \\
& 8x_1 + 9x_2 + 8x_3 \geq 21 \\
& 2x_1 + 3x_2 + 1x_3 \geq 9 \\
& x_1, x_2, x_3 \geq 0
\end{aligned}
\tag{1}
$$

This gives us the solution value of 117.5 with 1.5 units of meat steak, 2 units of eggs, and no unit of potatoes.

Duality theorem

In mathematics, there are frequently relationships between problems that are not obvious but once understood, they reap many dividends. If we consider the relationship between integral and derivative function expressed in fundamental theorem of calculus, it can provide a unified and coherent view of this calculus problem. As far as linear programming is concerned, the duality theorem as mentioned in [16] (pages 125-126) satisfies this coherent view. We define it as follows:

Theorem 2.4.1 Duality theorem *A linear programming problem stated in the following form is said to be in max form, which can be rewritten as:*

$$
\begin{aligned}
max \quad & c_1x_1 + c_2x_2 + c_3x_3 + ... + c_nx_n \\
s.t. \quad & a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + ... + a_{1n}x_n \leq b_1 \\
& a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + ... + a_{2n}x_n \leq b_2 \\
& . \\
& a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + ... + a_{mn}x_n \leq b_m \\
& x_1, x_2, x_3, ..., x_n \geq 0
\end{aligned} \tag{2}
$$

has its dual as this min problem

$$
\begin{aligned}
min \quad & b_1y_1 + b_2y_2 + b_3y_3 + ... + b_my_m \\
s.t. \quad & a_{11}y_1 + a_{21}y_2 + a_{31}y_3 + ... + a_{m1}y_m \geq c_1 \\
& a_{12}y_1 + a_{22}y_2 + a_{32}y_3 + ... + a_{m2}y_m \geq c_2 \\
& . \\
& a_{1n}y_1 + a_{2n}y_2 + a_{3n}y_3 + ... + a_{mn}y_m \geq c_n \\
& y_1, y_2, y_3, ..., y_m \geq 0
\end{aligned} \tag{3}
$$

Hence, if we have to apply theorem 2.4.1 to the diet problem 1, we will have the following dual problem:

$$
\begin{aligned}
max \quad & 10y_1 + 21y_2 + 9y_3 \\
s.t. \quad & 4y_1 + 8y_2 + 2y_3 \leq 45 \\
& 2y_1 + 9y_2 + 3y_3 \leq 25 \\
& 1y_1 + 8y_2 + 1y_3 \leq 33 \\
& y_1, y_2, y_3 \geq 0
\end{aligned}
\tag{4}
$$

By definition, this problem has the same optimal solution to the problem mentioned in the previous section. Our values of y_1, y_2, and y_3 are 10.625, 0, and 1.25, respectively.

Complementary slackness

According to [16] (pages 155-156), the complementary slackness theorem relates solution points of a linear programming problem and its dual as mentioned in the following theorem:

Theorem 2.4.2 Complementary slackness *Suppose* $X^* = (x_1^*, ..., x_n^*)$ *is a feasible solution to the problem of*

$$\textit{Maximizing } c.X \quad \textit{subject to} \quad AX \leq b,\ X \geq 0 \ \textit{(problem 1)}$$

and $Y^* = (y_1^*, ..., y_n^*)$ *is a feasible solution to the dual problem of*

$$\textit{Minimizing } b.Y \quad \textit{subject to} \quad A^tY \geq c,\ Y \geq 0 \ \textit{(problem 2)}$$

Then X^* *and* Y^* *are optimal solution points to their respective problems if and only if, for each i,* $1 \leq i \leq m$, *either*

$$\textit{(slack in the } i^th \textit{ constraint of problem 1 evaluated at } X^*) = b_i - \textstyle\sum_j a_{ij}x_j^* = 0$$

or

$$y_i^* = 0$$

and, for each j, $1 \leq j \leq n$*, either*

$$\textit{(slack in the } j^th \textit{ constraint of problem 2 evaluated at } Y^*) = \sum_i a_{ij} y_i^* - c_j = 0$$

or

$$x_j^* = 0$$

2.5 Related Work

We start this section by drawing the link between trust and reputation in the context of autonomous agents. In [18], Hazard and Singh focused on combining trust and reputation in an architectural model. In this work, a life cycle model of reputation and trust aims to connect them functionally and architecturally. The authors also stressed that trust is the forward view in time with respect to strategy; whereas reputation is looking backward in time with respect to signaling and determining agents' types. This viewpoint determines the coalition between trust and reputation in an open multi-agent system where selection of agents providing services is heavily based on overall reputation that affects the trust value. Figure 4 demonstrates this coalition. This figure points out that trust is heavily based on reputation. Therefore, if reputation parameters are not reliable, it is very likely that agents will be mistaken in their decisions and trust agents with fake reputation. This could lead to disappointment and dissatisfaction in terms of overall interaction and not fulfilling the systems objectives.

In [7], Bentahar *et al.* focused on dealing with collusion in "social network-based trust for agent-based services" that discusses service selection based on trust values collected either directly or indirectly. Its main challenge is to find out how to design a mechanism so that best strategy for customer agents will be revealing exactly what they believe about provider agents. This work proved that when truth telling strategy is encouraged by providing incentives, the proposed trust model outperforms competitive models namely BRS [19], TRAVOS [20], and FIRE [21] in terms of

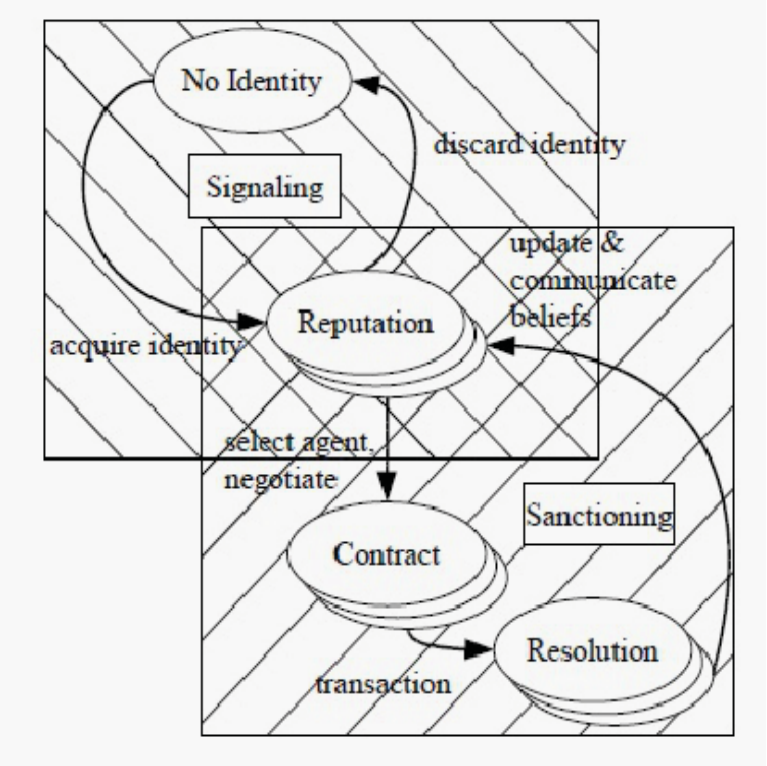

Figure 4: Trust and reputation life cycle from an agent's perspective

cumulative utility gained and good selection percentage. This is mainly due to the fact that the proposed model assesses the credibility using other services suggestions depending on how much they know the provider services. In terms of detecting fickle behavior, this proposed model performs better than the other models since it updates agents beliefs regularly and allows customer agents to be flexible in their decision selection. This trust framework relies on feedback, which are reputation parameters and it assumes these reputation parameters to be reliable.

Another important work in the literature that involves computation of an optimal value of incentive to encourage truth telling is [22]. This work showed that under specific circumstances, it is possible to compute payment mechanism in order to provide the right amount to reviewers whenever they are asked to rate a given service provider in both cooperative and non-cooperative settings. In this context, cooperative and non-cooperative settings refer to the degree of coordination in terms of reporting strategies among agents. The main result of the study suggested the existence

of the possibility of achieving one Nash Equilibrium[1] if certain conditions are being met such as the number of feedback and payment amount. Nevertheless, this work does not consider mixed strategies as defined in 2.4.3. This means agents can still collude if they adopt an efficient mixed strategy. In addition, this incentive compatible computational model proved that the relative cost is exponentially increasing as colluding fractions increase in partial coordination scenarios.

Investigation of the emergence of trust in social networks as introduced in [24] has received a great deal of attention. It analyzes the issue of edge creation from different agents over the social correlation using a combination of declarative and numerical techniques. These techniques determine the likelihood of edge creation that conceptualizes trust. As results of this research, it has been shown that the proposed framework, thanks to edge creation to maintain interactions, performs better in terms of good selection of providers. This trust framework is based on edge creation, which is driven by the reputation network parameters that are essentially based on historic feedback.

On the same basis, the authors in [9] proposed a maintenance based trust for multi-agent systems. They investigated how a retrospect trust adjustment can help an agent (e.g. Ag_a) assess other agents (denoted here by Ag_b) based on three dimensions: direct trust assessment, indirect trust assessment based on trustworthy agents, and indirect trust assessment based on referee agents as shown in Figure 5. The main contribution of this research is summarized as follows: mutual interactions between agents and update of their trust belief based on the final results in order to assess the credibility of the trustee agent in the so called maintenance phase.

Although the work investigated the optimization part of the maintenance phase in order to make it more adaptable to different situations, the problem of determining the suitable moment to perform the update has not been addressed. In [25], the authors suggested that user perception is not enough to compute service reputation, but how trustworthy the provider has been satisfying the service level agreement should be accounted for, which has been measured through a metric called verity. Nonetheless, monitoring and updating this verity has not been investigated.

[1] "a solution concept of a game involving two or more players, in which each player is assumed to know the equilibrium strategies of the other players, and no player has anything to gain by changing only its own strategy unilaterally" [23].

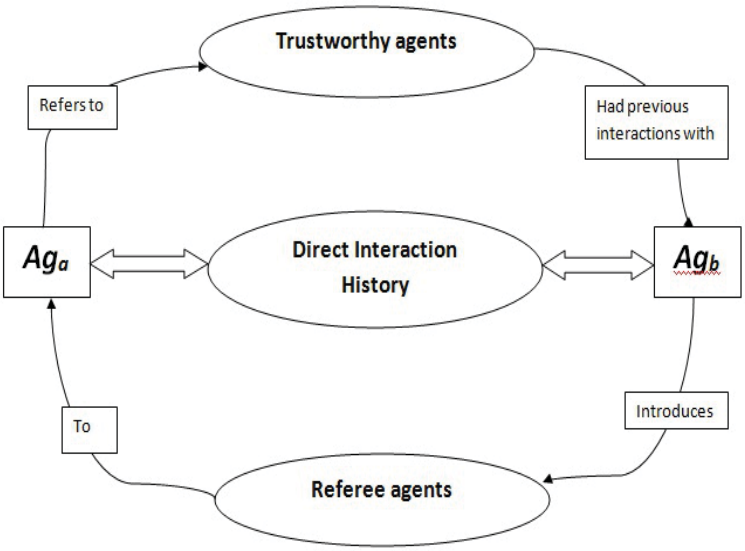

Figure 5: Maintenance based trust mechanism

As far as real time monitoring of services is concerned, several proposals in the literature have shown the possibility of improving the quality of these services in communities. These proposals aimed at implementing management block toward monitoring services. In [26], multi and cooperative broker[2] architecture for service selection has been proposed. Each broker manages services in its domain and shares information about these services with other brokers. The monitoring policy proposed in [27], which separated monitoring from management activities by dedicating a specific community to host monitors, has allowed online detection of violations (i.e. web services are not operating as expected). This has also reduced the overhead of other individual communities's managers. A tuning of this separation strategy for QoWS (Quality of Web Service) improvement is presented in [28] where managerial implementation has been extended to handle selection, communication, monitoring, adaptation, and load balancing on a periodical basis or upon request. The authors showed that QoWS attributes such as response time and availability are getting improved with this managerial community implementation. QoS management has been analyzed in [29] where a new architecture providing advanced management functionalities has been developed. These functionalities include extending the service description with QoS-centered annotation, including a validation

[2]a broker can be seen as third party that mediates between clients and providers

	Stays silent (cooperates)	Confesses (defects)
Stays silent (cooperates)	(-1, -1)	(-12, 0)
Confesses (defects)	(0, -12)	(-3, -3)

Table 2: Example of prisoner's dilemma game

process to test the service interface and the level of QoS that can be provided, supporting QoS negotiation, and monitoring the provided QoS.

The previously mentioned research aimed at creating more trusty networks that rely on reputation parameters either collected from other agents or more likely from history and log files that reflect historical behaviors of agents. One important question is how can we provide a meaningful and sound reputation mechanism in order to have history files display reliable and credible information? Researchers have provided many suggestions to answer this question by modeling the problem as a prisoner's dilemma game. For the sake of clarity, we define prisoner's dilemma game in the following paragraph.

Considering two players i and j being detained together in prison. For the detectives to make a case, they need a confession from one of them or both. They separate them and propose to each one of them the following deal made by the attorney general: if a prisoner confess and the other does not, this prisoner will go free and the other will be locked away for 1 year; if both prisoner's confesses, each serves 3 months in prison; if both prisoner's do not confess, they each serve 1 month in prison. Table 2 models this game.

Given that player i is the row player and j is the column player, it is obvious that a better solution strategy profile for both of them would be to cooperate, or to stay silent. However, pure strategy Nash Equilibrium tells us that the steady state of this game is that both players would defect. This means that in a multi-agent system environment where service providers are permanently competing in an open environment, agents probably would not report truthfully to each other; thus, an aggregation of this behavior would make an open multi-agent system less trustworthy in terms of total reputation parameters reflected on the feedback file. Therefore, how can we converge agents to cooperate, given that it is improbable for both agents to communicate about their choices during

the game?

In the context of achieving better solution concepts for prisoner's dilemma, Banerjee and Sen explored this possibility in their research entitled "Reaching Pareto optimality in prisoner's dilemma using conditional joint action learning" [30]. They considered that independent learning and joint action learning shows less promising results due to the fact that this type of learner assumes that actions of different agents are uncorrelated, which is not true in general. They present a new action learner called CJAL that understands that its own actions affect the action of other agents. Implementing this type of learner over time will allow agents to converge to a Pareto optimal state where both of them cooperate and act truthfully. The limitation with this type of learner is that players converge to a Pareto optimal under specific condition related to the payoff structure of the game. That means in other cases, players may not converge to Pareto optimal and they converge instead to a Nash Equilibrium state.

Which strategy profile to adopt in a repeated prisoner's dilemma game to get higher payoff? One important experiment to mention in the context of strategy profile selection is Axelrod's prisoner's dilemma tour [11, 31] in which five different strategies play again each other a repeated prisoner's dilemma game. These five strategies are:

- Random strategy: randomly cooperating or defecting with equal probability distribution;
- All-D: this strategy involves defecting all the time in all rounds;
- Tit-for-Tat: cooperating in the first round and subsequently doing what the opponent did in the previous round;
- Tester: testing on the first round the opponent by defecting, if the opponent ever retaliates with defecting, then subsequently playing Tit-for-Tat. Otherwise, playing a repeated sequence of cooperating for two rounds, and then defecting; and
- Joss: Like tester, this strategy is intended to exploit "weak" opponents. It is essentially Tit-for-Tat, but 10% of the time the strategy suggests defecting instead of cooperating.

The setup of this experiment gave the opportunity to each strategy to play against other strategies

for a number of consecutive rounds. The final results showed that Tit-for-Tat is the overall winner since it had the opportunity to play against other programs and strategies that were also inclined to cooperate. This draws the conclusion on the overall rules to take into consideration when designing a strategy profile to succeed in prisoner's dilemma. They can be summarized in four quality attributes [11, 32]:

- Forgiveness: do not be envious, which means not necessary to beat your opponent in order for you to do well;
- Niceness: do not be the first to defect;
- Reciprocate cooperation and defection; and
- Clarity: make the actions the agents should play under a given strategy clear.

Control and investigation turned out to be necessary to promote the quality of reputation parameters, which are quality and market share. Game theoretic analysis in [3] aimed at investigating the likelihood of circumstances that pushes agents to act truthfully in order to provide accurate reputation assessment and avoid providing fake feedback. Contributions of this work also lied specifically in what incentives to be provided to act truthfully while these agents are aware of penalties assigned by a special agent called controller agent, Cg. It has been shown that when Cg's accuracy level increases, agent's tendency to fake decreases over time. This conclusion was aligned with the conditions that lead agent to act truthfully using game theoretic structure [33]. Considering a network of providers and consumers that are able to maximize their profits, this research showed situations under which rational agents can act on a best response basis as well as a game theoretic analysis that computed the controller's detection threshold such that trust mechanism would be collusion-resistant. In the same work [33], the authors determined the percentage of the fixed size window that the controller agent needs to analyze in order to minimize fake feedback. It has been experimentally shown that under window percentage of 40% or 60%, collusion are minimized across the runs. However, elaboration on periodic maintenance and its impact on the quality of the reputation mechanism are missing in this work.

This work has proved that trust framework can be achieved if previous recommendations are followed. There is one problem with this strategy, especially the maintenance phase, which is related to learning. If agents have the chance to learn ahead of time the strategy profile of the controller agent, maintenance schedule for instance, these malicious agents will be ready far ahead of time to know when to act maliciously in a community of agents. This situation is even risky if for instance this controller agent is following predictable time moments. This is similar to the work reported in [8], which proposed a better solution concept to patrolling and monitoring activities that may be vulnerable to attacks, especially if adversaries are aware of such patrolling pattern. This solution lies in deploying ARMOR software assistant that casts this patrolling problem as Bayesian Stackelberg game to allow the software agent appropriately weight the different action in randomization as well as uncertainty over adversary types.

Chapter 3

Game Theory Analysis[1]

3.1 Problem

As stated in Chapter 1, our controller agent faces the challenge of scheduling the periodic maintenance update. This maintenance aims at updating the reputation value of each agent in the community that constitutes the controller's belief set. To motivate the problem before we present our solution idea, we recall the two main assumptions we base our framework on: rational behavior of community members and autonomous learning of these members. However, there are two constraints that need to be considered. The first one is that the controller agent considers expensive in terms of resources and time to perform check at every single moment in time. The second one highlights the inefficiency of performing maintenance and reputation adjustment based on very limited amount of information. On the same basis, it is not recommended for the controller agent to perform one single maintenance check in one large interval. It is an incentive for autonomous agents to collude more in that large interval, which would cause an increase in fake information percentage. The main objective of our scheduling algorithm is to reduce fake feedback in favor of truthful reporting in a community of agents. Hence, our problem is to design a scheduling algorithm to allow the controller agent to select moments in time from a time interval given the two aforementioned constraints. We

[1] the content of this chapter has been published in [34]

also recall that our controller agent is the specialized and delegated agent to investigate the logging file, which consists of aggregation of feedback values on each provider agent in the community.

In this chapter, we firstly present the general overview of the solution through the flow chart of the controller's algorithm to schedule the maintenance activity. Next, we explain in details the game theoretic framework between the controller agent and community members. In fact, the controller agent is facing two challenges. First, this controller needs to commit to a schedule before the community members do. Because the strategy the controller commits to can be observed by the community members, the game we are modeling is a Stackelberg game where the controller is the leader and community members are the followers [35,36]. Second, the controller agent is performing maintenance of different community members having different types (i.e. different probabilities of acting maliciously). This implies that we are modeling a Bayesian Stackelberg game [35–38].

3.2 Flow Chart of the Scheduling Problem

Figure 6 depicts the scheduling problem flow chart, which is composed of five main steps. The first step is to observe during specific moments the history of the logging file (history file) to get for instance the number of requests made for each agent as well as the obtained quality of service (arrow 1). The second step is to construct the Bayesian Stackelberg game from remarks and investigation of the history file (arrow 2). This step includes determining the payoff matrices R and C of the two players according to their possible strategies. The third step is formalizing the scheduling problem as an optimization problem using the payoff matrices defined in the second step and applying a multiple-integer linear programming technique called DOBSS, Decomposed Optimal Bayesian Stackelberg Solver [8,35,39]. This technique will solve the constructed Bayesian Stackelberg game by providing probability distributions of the actions of this controller agent (arrow 3). The fourth step is selecting among the probability distributions of the controller's pure strategies those that are different than zero, which correspond to an optimal schedule (arrow 4). Then the controller follows these moments to perform its maintenance (arrow 5). The details of the second, third, and fourth steps are provided

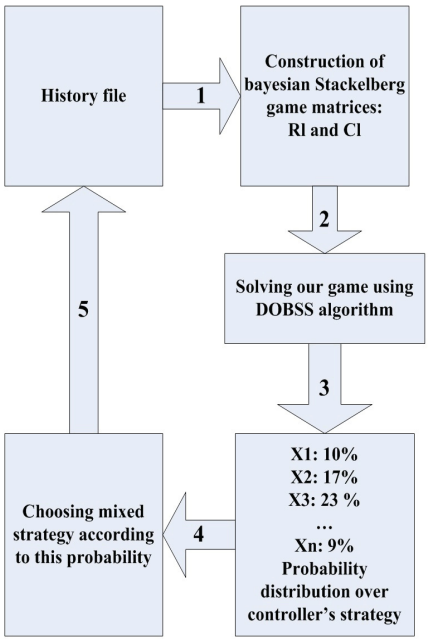

Figure 6: Flow chart diagram of the controller agent's algorithm

in subsequent sections.

3.3 Bayesian Stackelberg Game Definition

Two players are considered in our community-based multi-agent system: the controller agent and community members. The strategy profile of the controller agent is either to perform a maintenance check in a specific moment during a given time interval or not. The strategy profile of each member is to act truthfully or maliciously, which corresponds to colluding, in a specific moment of time.

Our objective is to allow the controller agent to schedule moments in time for maintenance update. It is crucial to mention three factors that determine the constraints of our solution.

- The first one is that the controller agent, which is an agent that takes the feedback file under surveillance as defined in [3], is not willing to investigate this file at every moment in time because of limited resources and capacities. For example, during 100 minutes of activities during which feedback submissions are performed at every second, the controller agent does not have the capacity to perform the investigation and maintenance at every single minute.

On the one hand, there are cases where communities might not have activities for a period of time. On the other hand, in a busy environment, it is not wise to schedule very few periodic maintenances since the chances of collusion are very high.

- The second factor is the type of community members. In open multi-agent systems, agents differ in there strategies in terms of when to perform the collusion as well as how damaging the fake feedback is compared to the real reputation parameter. For example, we may have agents that collude 2% of the time and others may be willing to collude for 90% of a time interval. Even if we have two colluding agents with the same percentage, say 20%, one agent may be willing to collude at moments 1 and 2 while the other at moments 8 and 9 in an interval of 10 moments. Also, we may have agents that collude with service consumers, as defined in [3], which are agents that continuously seek for services provided by some other agents, by populating very high feedback corresponding to dramatic change of behavior like 10/10 or a feedback that meets the average like 7/10, which is hard to detect. Community members are also assumed to be rational in their behaviors [40]; they act towards maximizing their own payoffs.
- The third one is that the controller agent needs to avoid choosing the same moment of check during subsequent time intervals. The strategy the controller chooses should be hard to predict, so it should not be an easy task to forgive malicious agents learn in advance the strategy of the controller agent overtime. Otherwise, those malicious agents will have better opportunity to schedule their collusion with maximum benefit and causing tremendous damage to the reputation of community.

To be focussed on the schedule problem, we assume that when the check is performed, the controller has the capacity to detect malicious acts if performed by the checked member with full accuracy. The game between the controller and community members can then be formalized as follows, which corresponds to the phase of constructing the Bayesian Stackelberg game matrices R and C in Figure 6:

- **Malicious act penalized**: this is the case where the controller agent performs the maintenance (the check) and penalizes the malicious agent by dismissing it from the community. The controller agent gets a positive payoff of $+\pi$ as we assume that the users are rewarding the controller for making the community secure. The community member gets a major penalty of $-(N+L)P$, where N is the average number of possible requests the agent gets if not fired, L is the expected increase in the number of requests the agent can get if not penalized, and P is the reward of that agent for serving each request. $N+L$ is then the expected number of requests the member can get if not fired and $-(N+L)P$ represents the loss undergone by the member because of being fired from the community.
- **Malicious act ignored**: this is the worst case of the controller agent in which a community member is colluding but the controller did not prevent it by performing the maintenance check. This corresponds to the controller payoff of $-\pi$ (as users will not pay the controller since the system is not secure) and a reward of the community member of the value $+(N+L)P$ (L corresponds to the increase resulting from collusion to promote the member's reputation).
- **Good performance ignored**: this is the case where the controller agent is not performing maintenance and community member is not colluding, but performing well in terms of its reputation increase. This corresponds to the payoff of the controller of a value of $+\lambda$, which is less than $+\pi$ (as the community is secure and users are happy, but no update has been made to justify a higher payoff), and a payoff of $+(N+L)P$ for the member as the member is benefiting from the increase L.
- **Good performance recognized**: this is the case where the controller agent is performing maintenance and community member is not colluding, but performing well in terms of its reputation increase. This corresponds to the payoff of the controller of $+\pi$ (as the community is secure and updated and users are happy), and a payoff of $+(N+L)P$ for the member.
- **Bad performance penalized**: this is the case where the controller agent is penalizing the member by dismissing it from the community not for collusion, but for bad performance. The

controller payoff is $+\pi$ since an effort has been done to make the community more reputable, which increases the quality of service for the users. The corresponding payoff of the member is $-NP$ because of being fired, which means requests are lost.

- **Bad performance ignored**: this is the case where the controller agent is not doing the maintenance and the member is not colluding, but doing bad in terms of reputation. The controller payoff is $-\pi$ and the corresponding payoff of the member is $+NP$.

We show in Table 3 the payoff structure for our game theoretic framework at a given moment in time with the column player as the controller agent and the row player as community member.

	Perform maintenance	Ignore maintenance
Collude	$(-(N+L)P, +\pi)$	$(+(N+L)P, -\pi)$
Not collude with good performance	$(+(N+L)P, +\pi)$	$(+(N+L)P, +\lambda)$
Not collude with bad performance	$(-NP, +\pi)$	$(+NP, -\pi)$

Table 3: Strategy profiles and payoff structure of our game

In order to concretely present the game and makes it clear, we present the payoff of a scenario in a time interval of five moments. One possible scenario is that the controller is performing maintenance update in moments 1 and 4 as shows in Table 4 and the community member under consideration is assumed to be performing bad.

	Collude	Not collude with bad performance
Moment 1	$(+\pi, -(N+L)P)$	$(+\pi, -NP)$
Moment 2	$(-\pi, +(N+L)P)$	$(-\pi, +NP)$
Moment 3	$(-\pi, +(N+L)P)$	$(-\pi, +NP)$
Moment 4	$(+\pi, -(N+L)P)$	$(+\pi, -NP)$
Moment 5	$(-\pi, +(N+L)P)$	$(-\pi, +NP)$

Table 4: One possible scenario of payoff structure between the controller agent and a community member

The controller's actions correspond to the row part of the table while the community member's actions correspond to the column part of the table. This model shows that it is straightforward for this community member to avoid collusion at moments 1 and 4 and act maliciously in other moments in order to maximize its payoff. This would mean that we have a dramatic increase of fake feedback

in this time interval, of up to 60% of the corresponding total feedback or reputation parameters.

3.4 Optimal Schedule

Our objective is scheduling the maintenance update, which means deciding about the moments to perform the check in a time interval of M moments. As explained in the previous section, if the scheduling algorithm provides predictable time line, then it increases the vulnerability of the reputation mechanism and malicious web services in that community may benefit from it. We inspire by and extend the technique proposed in [8] that aims to decide which points to check in an airport. However, our work is different from [8] in two perspectives. The first one is about the game model: the structure of our game is different from the one presented in [8] in both the payoff structure and actions available to the players, which makes the strategy profiles of the two games completely different. The second difference concerns the problem to be solved. In our work, a controller agent will have to come up with a scheduling algorithm with respect to time; while in [8] the software assistant has to come up with a solution for a resource allocation problem according to the available checkpoints in site (i.e. the airport). We use DOBSS, Decomposed Optimized Bayesian Stackelberg Game [8, 35, 39] in our framework for the following reasons:

- It solves a Bayesian Stackelberg game given the leader's payoff, follower's payoff, and probability distribution of each follower type.
- It provides as output an optimal mixed strategy that takes into consideration the rationality theory of the follower.
- It saves exponential time overhead by working on the compact version of the game rather than applying complex transformation techniques [41]

In order to understand our controller's algorithm, it is very important to explain the logic behind optimal mixed strategy for a Bayesian Stackelberg game that was introduced in [8, 35, 39]. Let us consider Table 5, adapted from [35], bearing in mind that the leader is the row player and follower is the column player. Nash equilibrium, as defined in Definition 2.4.4 (Chapter 2), tells us that the

	c	d
a	2,1	4,0
b	1,0	3,2

Table 5: Payoff table for an example of a normal form game

only pure strategy is that the leader plays a and follower plays c. In fact, playing b for the leader is strictly dominated since it ensures higher payoff over the follower. Let us consider the following scenario: given that the leader commits to a mixed strategy of playing a and b with equal probability (0.5), the follower will choose to play d because $0.5 * 1 + 0.5 * 0 < 0.5 * 0 + 0.5 * 2$. The payoff of the leader would be $0.5 * 4 + 0.5 + 3 = 3.5$.

Back to our problem, the question that arises is how to determine the probability distributions over the vector of pure strategies of our controller, which consists of moments in a time interval. To answer this question, let us start by considering a problem of one leader against one follower. First, we present the mathematical reasoning with one community member type and then generalize the solution to many types. Let us now introduce the problem parameters. During a given interval of time, there is a given number of discrete moments where the controller has to play. In each moment, the controller has two strategies: perform the check (strategy 1) or not (strategy 0). Similarly, there is a given number of discrete moments where the member has to play by choosing between acting truthfully (strategy 1) and acting maliciously (strategy 0). Let x and y be the pure strategy vectors of the controller and community member respectively. x_i is the strategy chosen by the controller at moment i and y_j is the strategy chosen by the member at moment j. Thus, $x_i \in \{0, 1\}$ where $x_i = 1$ (resp. $x_i = 0$) means strategy 1 (resp. 0) is played by the controller at moment i. $y_j \in \{0, 1\}$ has the same meaning for the member. Let I be the index set of acting moments of the controller agent; J be the index set of acting moments of the member; and R and C be the payoff matrices such that R_{ij} is the controller agent's reward and C_{ij} is the member's reward such that the controller agent is playing strategy x_i and the member is playing strategy y_j. $|I|$ and $|J|$ are the cardinalities of I and J respectively, where $|J|$ is function of $|I|$ and α represents the size of the time window to be investigated relative to the log file where the feedback are recorded. In this thesis, we use the

following linear function:

$$|J| = (\alpha + 1)|I| \tag{5}$$

By fixing the strategy vector of the controller (i.e fixing x), the community member has to solve the optimization problem (6) to find its optimal response to x.

$$\begin{aligned} \arg\max_{y} \quad & \sum_{j \in J} \sum_{i \in I} C_{ij} x_i y_j \\ s.t. \quad & y_j \in \{0,1\} \qquad j \in J \end{aligned} \tag{6}$$

The objective function is maximizing the member's payoff and the constraint makes feasible any member's mixed strategy. When solving this problem, it is obvious that for a given j, if $\sum_{i \in I} C_{ij} > 0$ then $y_j = 1$. Thus, we obtain:

$$\begin{aligned} \sum_{i \in I} C_{ij} > 0 \quad &\Rightarrow \quad y_j = 1 \\ \sum_{i \in I} C_{ij} \leq 0 \quad &\Rightarrow \quad y_j = 0 \end{aligned}$$

Consequently, we obtain the following equation:

$$\sum_{i \in I} C_{ij} y_j = \max(0, \sum_{i \in I} C_{ij}) \quad j \in J \tag{7}$$

Given a fixed strategy vector y of the community member, The controller must choose its own strategy x that is best response to y. Associated with the strategy of performing check (strategy 1) at moment i is a cost t_i. Thus, the optimization problem the controller has to solve is formalized as follows:

$$\begin{aligned} \arg\max_{x} \quad & \sum_{i\in I}\sum_{j\in J} R_{ij}(1-t_i)x_i y_j \\ s.t. \quad & \sum_{i\in I} x_i > 0 \\ & x_i \in \{0,1\} \qquad\qquad i \in I \end{aligned} \tag{8}$$

The objective function maximizes the controller's payoff by maximizing the reword R_{ij} and minimizing the cost t_i, which is the proportion of the payoff of our controller. The second constraint forces the controller to have at least one check moment. From this problem, we obtain the general problem the controller has to solve (i.e. without fixing the member's strategy y) as follows:

$$\begin{aligned} \arg\max_{x,y} \quad & \sum_{i\in I}\sum_{j\in J} R_{ij}(1-t_i)x_i y_j \\ s.t. \quad & \sum_{i\in I} x_i > 0 \\ & x_i \in \{0,1\} \qquad\qquad i \in I \\ & y_j \in \{0,1\} \qquad\qquad j \in J \\ & 0 \le t_i \le 1 \end{aligned} \tag{9}$$

The problem with the optimization problem (9) is that it does not consider the optimal solution y of the member. This means, we need to add a constraint forcing the obtained solution for the controller's strategy vector x to be associated with the optimal solution for the member's strategy vector y. This is archived by integrating the Equation 7 as a constraint. Therefore, the controller's

problem becomes:

$$
\begin{aligned}
\arg\max_{x,y} \quad & \sum_{i\in I}\sum_{j\in J} R_{ij}(1-t_i)x_i y_j \\
s.t. \quad & \sum_{i\in I} x_i > 0 \\
& x_i \in \{0,1\} && i \in I \\
& \sum_{i\in I} C_{ij} y_j = \max(0, \sum_{i\in I} C_{ij}) && j \in J \\
& y_j \in \{0,1\} && j \in J \\
& 0 \le t_i \le 1
\end{aligned}
\tag{10}
$$

The first and second constraints enforce a feasible mixed strategy for the controller, and the third and fourth constraints enforce a feasible mixed strategy for the community member.

3.5 Controller's Scheduling Algorithm

We showed in the previous section how one leader can optimize its payoff against one follower using multiple integer quadratic programming. In order to solve the case of many follower types, we need to consider the following additional parameters, which makes the game Bayesian:

- p^l: the probability of encountering a member of type l;
- R^l and C^l: the payoff matrices such that R^l_{ij} and C^l_{ij} are the rewards associated with the strategies x_i and y_j of the controller agent and community member of type l respectively;
- y^l_j: the pure strategy of the member of type l at moment j;
- L: the set of members types.

Hence, the problem (10) becomes:

$$
\begin{aligned}
\underset{x,y}{\arg\max} \quad & \sum_{i\in I}\sum_{l\in L}\sum_{j\in J} p^l R^l_{ij}(1-t_i)x_i y^l_j \\
s.t. \quad & \sum_{i\in I} x_i > 0 \\
& x_i \in \{0,1\} && i \in I \\
& \sum_{i\in I} C^l_{ij} y^l_j = \max(0, \sum_{i\in I} C^l_{ij}) && j \in J \\
& y^l_j \in \{0,1\} && j \in J \\
& 0 \le t_i \le 1
\end{aligned}
\tag{11}
$$

Notice that this optimization problem is a quadratic programming problem since we have two unknown integers to solve $x_i y^l_j$. We can make the program linear using the following change of variables: $x_i y^l_j = z^l_{ij}$. Consequently, we obtain the following equivalent problem:

$$
\begin{aligned}
\underset{x,y}{\arg\max} \quad & \sum_{i\in I}\sum_{l\in L}\sum_{j\in J} p^l R^l_{ij}(1-t_i) z^l_{ij} \\
s.t. \quad & 0 \le \sum_{i\in I}\sum_{j\in J} z^l_{ij} \le |I||J| \\
& 0 < \sum_{i\in I} z^l_{ij} \le y^l_j |I| && j \in J \\
& 0 \le \sum_{j\in J} z^l_{ij} \le |J| && i \in I \\
& z^l_{ij} \in \{0,1\} && (i,j) \in I \times J \\
& \sum_{i\in I} C^l_{ij} y^l_j = \max(0, \sum_{i\in I} C^l_{ij}) && j \in J \\
& y^l_j \in \{0,1\} && j \in J \\
& 0 \le t_i \le 1
\end{aligned}
\tag{12}
$$

It is worth mentioning that the parameter p^l (the probability of encountering a member of type l) encapsulates the probability of receiving requests from that member. Because we are solving multiple-integer linear programming (with multiple members), the overall requests distribution is considered in the problem 12. Furthermore, the equivalence of the problems (11) and (12) is straightforward by construction as the objective function of (12) is a direct transformation of the objective function of (11) and each constraint in (12) is directly obtained from a constraint in (11) using the change of variables: $x_i y_j^l = z_{ij}^l$. The objective function maximizes the payoff of the controller agent against different community members having different types. The output of this program is the mixed strategy for the controller, which dictates the strategy to choose at each moment (i.e. performing the check or not) given that the community members are playing the best strategies (i.e. best responses). Thus, the controller's strategy vector we obtain is the optimal schedule we are seeking, which corresponds to stage four in the controller's flow chart (Figure 6) explained earlier. To show how the dynamism is captured in this optimization problem, let us consider the following example with $|I| = |J| = 5$ (i.e. acting over 5 moments). Assuming that the solution the algorithm provides for x is (1; 0; 0; 1; 1); this means the controller should perform the check at moments t_1; t_4, and t_5. This will correspond to the best strategies of the member. Consequently, most likely the member will play honest all the time or collude during times where the check is performed (as the controller is playing assuming that the member is maximizing its payoff). In fact, the following theorem holds:

Theorem 3.5.1 *The optimal solution given by solving the problem 12 is Pareto optimal.*

Proof: Solving problem 12 results in an optimal solution for the controller in terms of payoff given that the community member is maximizing its payoff. Consequently, any change in the controllers strategy will not make this controller better off without making the member worse off (the controller can gain more by detecting a collusion only if the member gains less by being detected as malicious). Conversely, any change in the member strategy can make this member better off (malicious action not detected) only if it makes the controller worse off. Thus, there is no Pareto improvement that

can be made, so we are done.

Chapter 4

Simulation Results

This chapter presents the settings of our simulations in terms of implementation context and technological details. We show our results in both static and dynamic environments concerning reducing the number of fake feedback. We conclude this chapter with a qualitative analysis of our algorithm in terms of variance of output and controller's overall payoff.

4.1 Simulation Settings

Our simulation consists of interactions between users and web services. According to a certain schedule, the controller agent performs a maintenance update. Every agent is being checked for its performance and this agent is being penalized by leaving the community in case the average reputation calculated is less than an average reputation defined by the controller agent. Therefore, the web service is being penalized for performance reasons only.

At every moment, certain agents may collude once at most. The number of requests received by each web service is directly proportional to the maintenance results. Thus, each none penalized web service is being rewarded with an increase in terms of requests that can be received during that time interval.

At the beginning of every round, the controller agent follows an algorithm to decide when to

perform the update in that round, taking into consideration the strategies of collusion of the web services for that round. Possible collusion can be defined as the fake feedback provided by the user towards the web service that does not demonstrate its true performance, which can be made at every moment.

For example, consider web services A, B, and C as part of one community that conditions for each web service to perform on average 7 out of 10 with a variance of 0.2. In other words, agents are allowed to stay in the community if their reputation average falls between 6.8 and 7.2 inclusively. At the start of the simulation, they can receive up to 10 requests during a time interval of ten moments. At moment 3, our controller performs its maintenance phase during which agent A's reputation average is 6.9, agent B's reputation average is 7.15, and agent C's reputation average is 6.5. According to our rule, agent C gets major penalty by leaving the community; which means it will not get any more income for services. Logically speaking, agent A's reputation value is closer to 7 than agent B's reputation value. Therefore, our controller will devote more requests to agent A than agent B, say 11 requests to agent A and 9 requests to agent B for the following interval starting from the moment 3.

The simulation of the controller behavior is implemented in five forms. The first form is the fixed time check: the controller agent chooses fixed moments in time at every round and it gets repeated for many rounds. The second form is a uniform random time check: the controller agent chooses moments in time using a uniform random distribution at the beginning of every round. The third form is a normal random time check: the controller agent chooses moments in time using a normal random distribution at the beginning of every round. The fourth form is a poisson random time check: the controller agent chooses moments in time using a poisson random distribution at the beginning of every round. The fifth form is using a mixed strategy that is based on a game theoretic implementation consisting of a Bayesian Stackelberg game: the controller agent chooses moments in time based on the solution provided by the DOBSS.

Concerning the technologies, we list their uses in our two simulation settings that will be discussed

in the next section. We implemented our simulator in Java 6 as compiler and interpreter using Netbeans 6.9.1. The simulation results (data) supply a relational database under MySQL 5.5. At the end of each run, statistical data are registered in the database and is being imported to an Excel file for analysis. As for the four scheduling algorithms, we have proceeded as follows:

- Uniform random scheduling: we have used the `nextInt()` function from java.util.Random library in order to allow our controller agent to select uniform moments in time.
- Normal random scheduling: we have used the `nextGaussian()` function from java.util.Random library in order to allow our controller agent to select moments in time following the Gaussian distribution.
- Poisson random scheduling: we have used the `com.softnetConsult.utils.random.Poisson` and the average feedback for a given time line as the parameter λ in order to allow our controller agent to select moments following a Poisson distribution in time.
- DOBSS scheduling: we have used the external library `LPSolver 5.1` that takes as input the game theory model we implemented and gives as output moments in time for the controller to perform the check.

The runs were performed on a Windows machine with Intel Core2 CPU, 1.86GHz for each processor and 5GB of RAM. In summary, our java project consists of the following classes:

- Agent: this abstract class is our superclass with Id and name as attributes.
- ControllerAgent: this abstract class is a subclass of the agent that demonstrates our controller agent with attributes such as mixedStrategy for saving next schedule, average for the average performance, and threshold for the precision degree of the average.
- BayesianStackelbergControllerAgent: this subclass of ControllerAgent performs its scheduling based on the Bayesian Stackelberg game with attributes such as Rl for payoff for our controller, Cl for payoff for provider agents regardless of their types, and p as probability distribution of the types of these agents.
- FixedControllerAgent: this subclass of ControllerAgent performs its scheduling based on a

fixed, pre-determined schedule for every time interval.

- RandomControllerAgent: this abstract class is also a subclass of ControllerAgent. It is meant to provide software reuse features among the three random controller classes that are defined next.
- UniformRandomControllerAgent: this subclass of RandomControllerAgent performs its scheduling algorithm based on random generation of variables using java.util.Random as explained previously.
- PoissonControllerAgent: this subclass of RandomControllerAgent performs its scheduling algorithm based on the Poisson random generation of variables.
- NormalRandomControllerAgent: this subclass of RandomControllerAgent performs its scheduling algorithm based on `nextGaussian()` generator of Gaussian variables.
- WebService: this subclass of Agent simulates our provider agent. Among its parameters are balance, collusion degree, response time as the time the agent takes to reply to a certain request, waiting queue for different user agents, when to collude for scheduling collusion scenarios with selected user agents, and charging price.
- User: this subclass of Agent is our consumer agent with properties of balance, preferred time response, which is compared against the actual waiting time to rate the agent's satisfaction, and collusion price that corresponds to the price service provider needs to pay in order for this user agent to provide fake feedback to alter the reputation mechanism.
- ReputationValue: this class encapsulates our reputation parameter with parameters of web service id, user id, reputation, and time at which this satisfaction has been registered into a log file, which is called feedback file.
- FeedbackFile: this class represents our history file for every single web service in the community. It is organized in the form of an adjacency linked list storing reputation values in each cell.

4.2 Result Discussion

4.2.1 Collusion Percentage

Our static simulation is characterized by the following data:

- Number of users = 100.
- Number of web services = 50.
- Number of runs = 10.

The distribution of colluding web services in the experiment is presented in Table 6. Our results consist of analyzing over 10 different runs on each strategy the percentage of fake feedback in the community. Figure 7 shows collusion percentage at each moment in time across the five scheduling algorithms. Two conclusions can be drawn from this figure. The first one is that the DOBSS algorithm manages to reduce the number of collusion earlier than the other algorithm for less than 5%. As the vertical lines on each graph of the figure shows, precisely at moment 40 DOBSS manages to reduce the number of collusion to 5 percent where the other algorithms reach the same result at moments no earlier than 45. The second remark is that the DOBSS manages to have the smallest number of collusion at the last moment compared to the other algorithms. Table 7 exhibits this fact, where the percentage has been rounded to two decimal numbers.

Distribution interval of the collusion degree	Distribution percentage
0	20
]0, 20[	12
[20, 50[	34
[50, 70[	18
[70, 100]	16

Table 6: Distribution of the collusion degree in our static simulation

	DOBSS	Fixed time	Normal random	Poisson random	Uniform random
Moment 95	1.51	2.47	2.01	2.02	2.31
Moment 96	1.22	2.61	2.44	2.74	2.27
Moment 97	1.77	2.32	2.59	2.83	1.92
Moment 98	1.80	2.59	2.30	2.97	1.95
Moment 99	1.52	2.44	2.50	2.64	2.03

Table 7: Percentage of the collusion degree at the last five moments

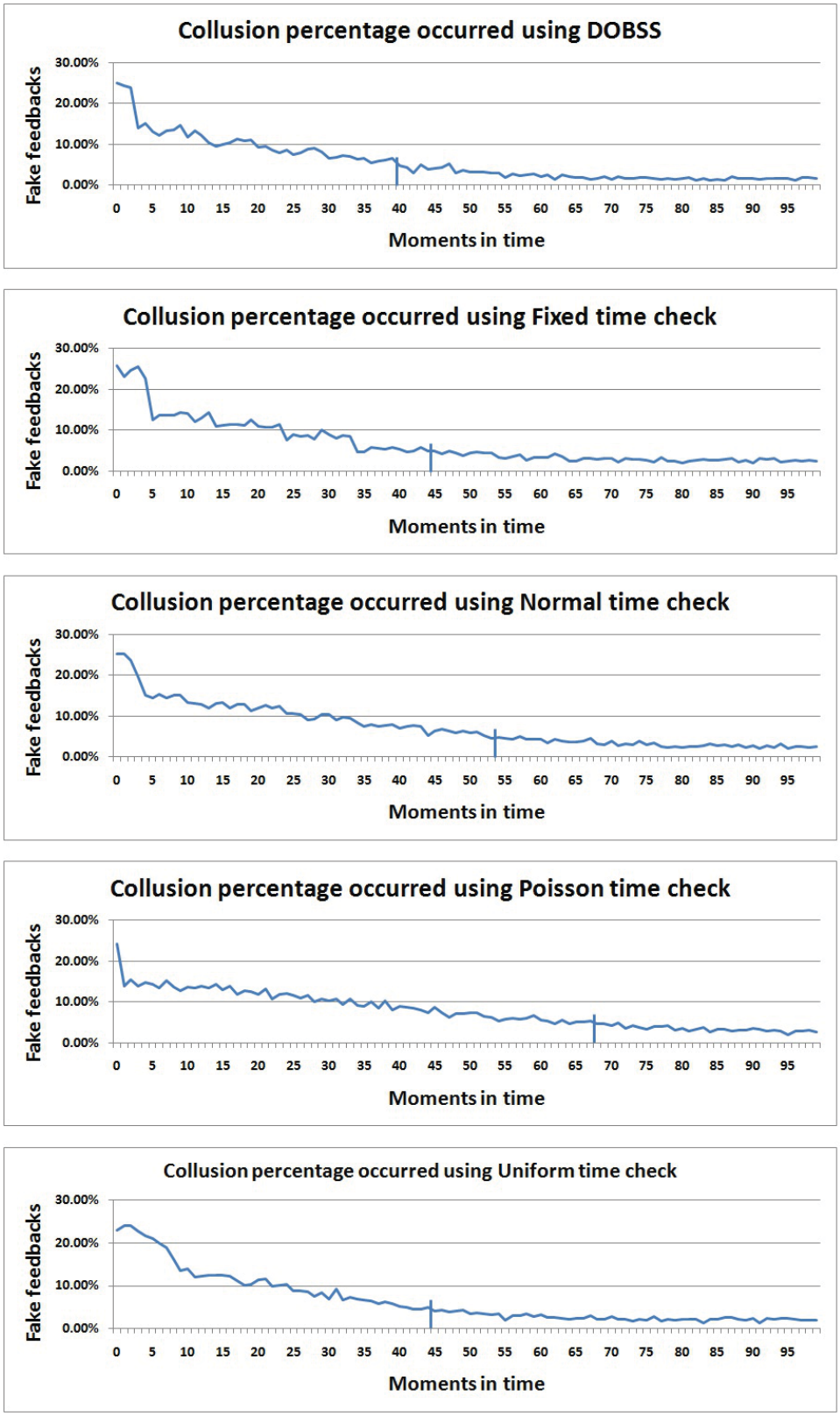

Figure 7: Comparative results between the scheduling algorithms in a static environment

In the next simulation, we analyzed the behavior of the five different algorithms in a dynamic environment, an environment where web services and agents join and leave the community dynamically. This simulation allows investigating two metrics: the first one is how can controller agents under different time check algorithms be able to cope with the environment change; the second one is how fast the controller agent can react according to a change of environment. Over 10 different runs on each strategy, the obtained results regarding the percentage of fake feedback in the community and the percentage of collusion at each moment in time across the five scheduling algorithms are shown in Figure 8.

Before pointing out the results from our dynamic simulation, it is very important to explain the sudden drop in the very first moments across the five algorithms. At the starting point of the simulation, web services receive the same number of requests. Once the maintenance is performed, our controller agent, regardless of the scheduling algorithm, rewards the ranked web services with an increase in terms of requests per interval. Therefore, since the malicious web services performed collusion to mislead the controller agent before the maintenance, they get more requests after the maintenance has been performed. Hence, the number of truthful feedback increases after the maintenance phase while the number of fake feedback are nearly the same.

As depicted in Figure 8, the DOBSS algorithm is more stable in terms of reducing the collusion percentage regardless of the dynamic environment. It also manages to get the minimum number of collusion, especially at the last moments starting from the moment 89 till the end of the simulation. It also shows that the controller agent under DOBSS algorithm was able to react quicker than the rest of the algorithms concerning penalizing the malicious web services before they have the chance to collude.

4.2.2 Controller's Overall Reward

To better show the results and merits of our scheduling algorithm, we conducted additional simulations under new simulation settings. The changes of the settings were applied on both the number

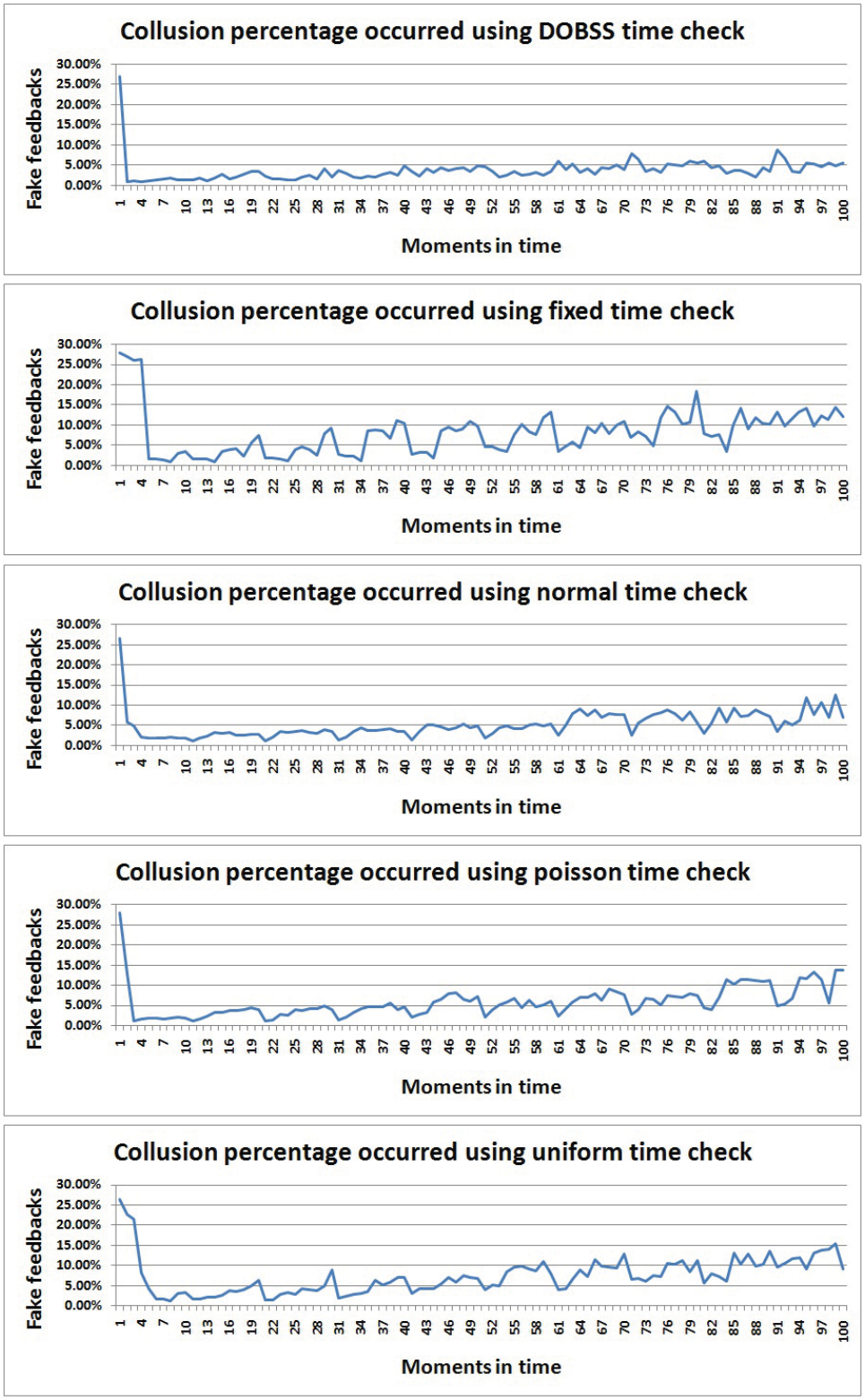

Figure 8: Comparative results between the scheduling algorithms in a dynamic environment

of users and number of web services. In addition, we modified the distribution of colluding web services in order to have a different view of the performance of our algorithm. We have set this simulation with the following data in a static environment, where a controller agent is dealing with the same agents across time:

- Number of users = 250.
- Number of web services = 75.
- Number of runs = 10.

The distribution of colluding web services in the static simulation is presented in Table 8.

Collusion degree distribution interval	Distribution percentage
0	8
]0, 20[	18.67
[20, 50[	33.34
[50, 70[	22.66
[70, 100]	17.33

Table 8: Distribution of the collusion degree in controller's overall reward in a static environment

Our results of analyzing the controller's reward output in the five scheduling algorithms are demonstrated in Figure 9. It is shown that most of the time, using DOBSS strategy, the controller insures higher reward. The fact that our solution is based on maximizing payoffs explains the results.

In order to determine which strategy works better under general settings, we extended the simulation under dynamic settings. The results of this simulation in terms of obtained reward are shown in Figure 10. As theoretically expected, we notice that under the DOBSS solving strategy, our controller gets more payoff than in other strategies. Moreover, the controller was able to maximize its benefits even under dynamic settings. Compared to other scheduling strategies, only after short time (moment 25) that our controller agent using the proposed scheduling algorithm start to increase its payoff.

4.2.3 Predictability Analysis

Predictability analysis, also called here randomization, of the scheduling pattern aims at addressing the issue of being or not easily predictable. Our ultimate objective is to see over long period of

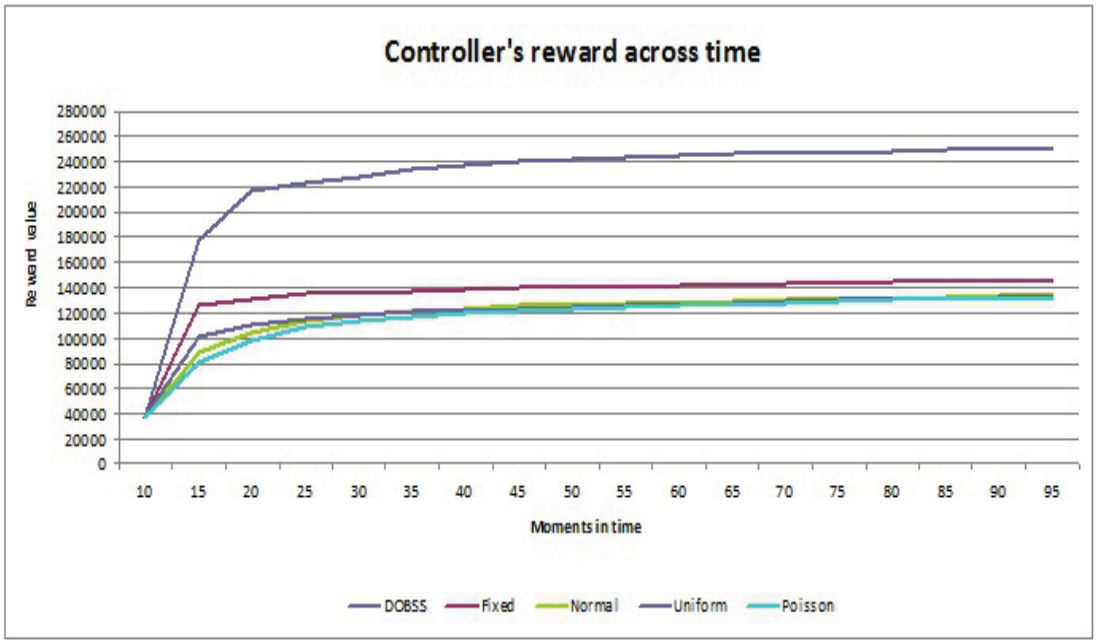

Figure 9: Comparative results in terms of controller reward in static settings

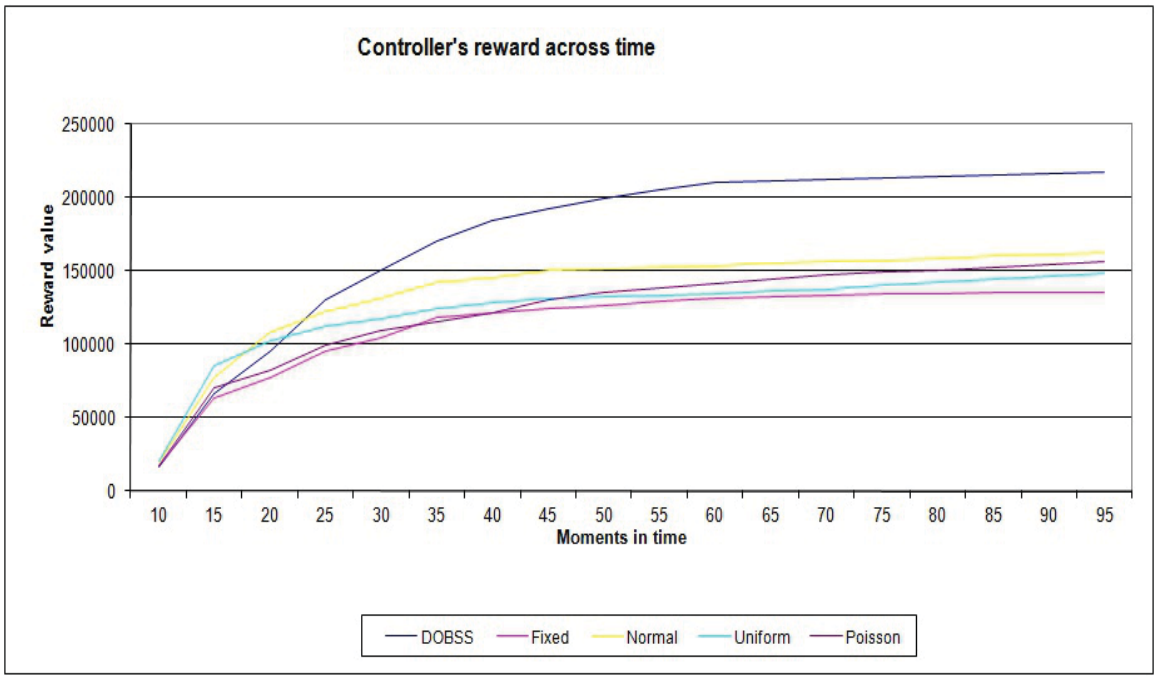

Figure 10: Comparative results in terms of controller reward in dynamic settings

time, while the controller is trying to maximize its profit, if the scheduled check moments can be guessed given previous check moments. In fact, being easily predictable will unintentionally allow malicious intelligent agents to benefit from this vulnerability. To this end, we conducted simulations and analysis over the output generated by the different scheduling algorithms as mixed strategies in the same settings as the previous experiment (Table 8). For static environment, our results are reported in Figure 11 and in Figure 12 for dynamic environment. We notice that at the same time that our controller maximized its output as seen in Figure 9, it was able to generate schedules (i.e. check moments) that are of significant variance than any other scheduling algorithm, even higher than stochastic randomization techniques. Having high variance is a strong indicator that the check moments are not following a well shaped pattern, which makes their predictability hard. In dynamic environment, our scheduling algorithm have higher variance than in static environment. This is mainly due to the fact that situation changes frequently through time intervals and our solver was able to adapt itself in terms of scheduling output. Our main conclusion is that it is harder to predict check moments under our scheduling algorithm than under regular randomization techniques.

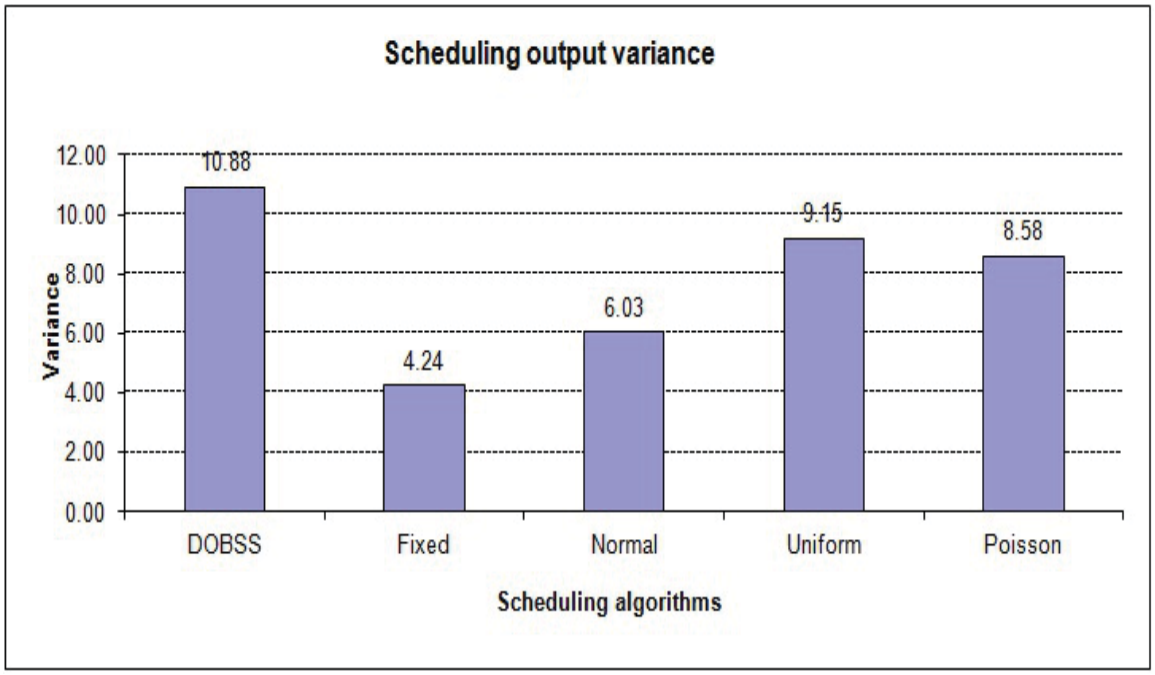

Figure 11: Comparative results in terms of scheduling output variance in static settings

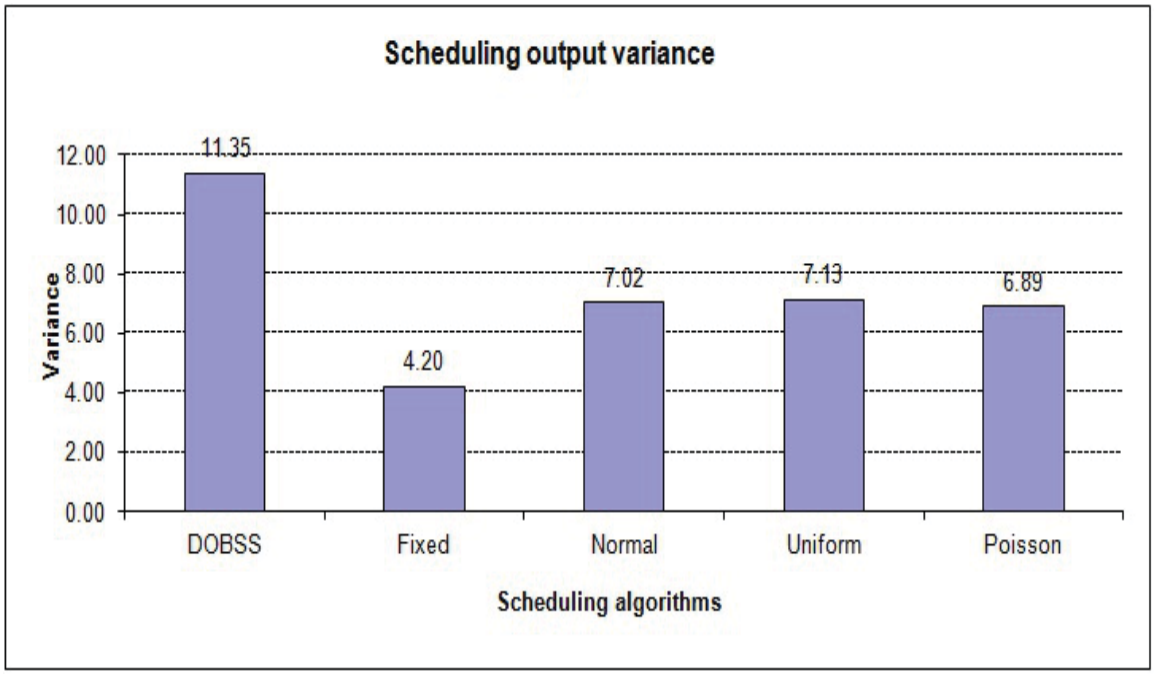

Figure 12: Comparative results in terms of scheduling output variance in dynamic settings

Chapter 5

Conclusion and Future Work

5.1 Contributions

The main contribution of this thesis is the proposition of a scheduling algorithm to decide about moments where the trust maintenance can be performed by the controller agent within a community of rational and selfish agents. We used a game theoretic model where different strategy profiles for the participating agents are considered. We theoretically presented the game and defined the associated payoff structure. One important fact to notice is that whatever reputation mechanism the controller agent applies, agents in the community will very likely have the chance to closely observe the behavior of the controller agent in terms of scheduling the maintenance (or check) update. To consider this fact, we proposed to use our Stackelberg game that the controller agent plays as a leader and the remaining agents in the community as followers. As we have demonstrated in this thesis, agents do differ in their strategies in terms of collusion; hence, our controller agent is facing multiple types which makes the game Bayesian. This makes the scheduling mechanism a Bayesian Stackelberg game, which is solved using optimization techniques. We also demonstrated the efficiency of the proposed algorithm in terms of reducing fake feedback in the log file against other scheduling algorithms. This algorithm allows our controller agent to maintain a sound reputation mechanism

in its community of agents.

5.2 Limitations and Future Work

This research demonstrated the potential of using game theoretic foundations in order to effectively and efficiently schedule maintenance update to allow the controller agent to force other agents to act truthfully. However, still further research is needed and interesting research questions are to be explored. Among which are:

- Include, in the game theoretic model, the possibility of optimizing the review window on each agent. In other words, investigating the question of how can we determine the portion of feedback that our controller agent needs to analyze since the last maintenance operation.
- Include the possibility of initiating malicious feedback from the user agent in our game scenario since we assumed that collusion is initiated from provider agents. This means that our game will be extended from two players to three players operations with different payoff structure.
- The third one is to consider different probabilities of the controller's accuracy, which implies using more parameters and introducing sophisticated solving techniques.

Bibliography

[1] [Online]. Available: http://en.wikipedia.org/wiki/File:IntelligentAgent-SimpleReflex.png

[2] J. Bentahar, Z. Maamar, W.Wan, D. Benslimane, P. Thiran, and S. Subramanian, "Agent-based communities of web services: an argumentation-driven approach," *Service Oriented Computing and Applications*, vol. 2, no. 4, pp. 219 – 238, 2008.

[3] B. Khosravifar, J. Bentahar, A. Moazin, and P. Thiran, "On the reputation of agent-based web services," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*, Atlanta, Georgia, USA, 2010, pp. 1352 – 1357.

[4] ——, "Analyzing communities of web services using incentives," *International Journal of Web Services Research*, vol. 7, no. 3, pp. 30 – 51, 2010.

[5] Z. Maamar, S. Subramanian, P. Thiran, D. Benslimane, and J. Bentahar, "An approach to engineer communities of web services: concepts, architecture, operation, and deployment," *International Journal of E-Business Research*, vol. 5, no. 4, pp. 1 – 21, 2009.

[6] J. Bentahar and B. Khosravifar, "Using trustworthy and referee agents to secure multi-agent systems," in *Proceedings of the Fifth International Conference on Information Technology: New Generations*, Washington, DC, USA, 2008, pp. 477 – 482.

[7] J. Bentahar, B. Khosravifar, and M. Gomrokchi, "Social network-based trust for agent-based services," in *Proceedings of International Conference on Advanced Information Networking and Applications Workshops*, Washington, DC, USA, 2009, pp. 298 – 303.

[8] J. Pita, M. Jain, J. Marecki, F. Ordonez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Krau, "Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles international airport," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, Estoril, Portugal, 2008, pp. 125 – 132.

[9] B. Khosravifar, M. Gomrokchi, J. Bentahar, and P. Thiran, "Maintenance-based trust for multi-agent systems," in *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, Budapest, Hungary, 2009, pp. 1017 – 1024.

[10] M. Wooldridge, *An Introduction to MultiAgent Systems*. John Wiley & Sons Ltd, 2002.

[11] G. Weiss, *Multiagent systems: A modern approach to distributed artificial intelligence*. MIT Press, 1999.

[12] S. Elnaffar, Z. Maamar, H. Yahyaoui, J. Bentahar, and P. Thiran, "Reputation of communities of web services - preliminary investigation," in *Proceedings of the 22nd International Conference on Advanced Information Networking and Applications - Workshops*, Washington, DC, USA, 2008, pp. 1603 – 1608.

[13] [Online]. Available: http://www.merriam-webster.com/dictionary/reputation

[14] [Online]. Available: http://www.merriam-webster.com/dictionary/trust

[15] Ramchurn, S. D., Huynh, Dong, Jennings, and N. R., "Trust in multi-agent systems," *The Knowledge Engineering Review*, vol. 19, no. 1, pp. 1 – 25, 2004.

[16] P. R. Thie and G. E. Keough, *An Introduction to Linear Programming and Game Theory*. John Wiley & Sons, Inc., 2008.

[17] D. C. Parkes, "Iterative combinatorial auctions: Achieving economic and computational efficiency," Ph.D. dissertation, Computer and Information Science, University of Pennsylvania, 2000.

[18] C. J. Hazard and M. P. Singh, "An architectural approach to combining trust and reputation," in *Proceedings of the 13th AAMAS Workshop on Trust in Agent Societies (Trust)*, Toronto, Canada, 2010.

[19] A. Jesang and R. Ismail, "The beta reputation system," in *Proceedings of the 15th Bled Electronic Commerce Conference*, Bled, Slovenia, 2002, pp. 324 – 337.

[20] W. T. Teacy, J. Patel, N. Jennings, and M. Luck, "Travos: Trust and reputation in the context of inaccurate information sources," *Autonomous Agents and Multi-Agent Systems*, vol. 12, no. 2, pp. 183 – 198, 2006.

[21] T. Dong-Huynh, N. Jennings, and N. Shadbolt, "Fire: An integrated trust and reputation model for open multi-agent systems," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 13, no. 2, pp. 119 – 154, 2006.

[22] R. Jurca and B. Faltings, "Collusion-resistant, incentive compatible feedback payments," in *Proceedings of the 8th ACM Conference on Electronic Commerce (EC)*, New York, NY, USA, 2007, pp. 200 – 209.

[23] [Online]. Available: http://en.wikipedia.org/wiki/Nash_equilibrium

[24] B. Khosravifar, J. Bentahar, and M. Gomrokchi, "Declarative and numerical analysis of edge creation process in trust-based social networks," in *Proceedings of Declarative Agent Languages and Technologies - volume 5948 of LNAI*, Budapest, Hungary, 2009, pp. 141 – 161.

[25] S. Kalepu, S. Krishnaswamy, and S. Loke, "Verity: a QoS metric for selecting web services and providers," in *Proceedings of the Fourth international conference on Web information systems engineering workshops*, Roma, Italy, 2004, pp. 131 – 139.

[26] M. Serhani, E. Badidi, and A. B. M. Salem, "A cooperative approach for QoS-aware web services selection," in *Proceedings of the International Conference on Computer and Communication Engineering (ICCCE)*, Kuala Lumpur, Malaysia, 2008, pp. 1084 – 1088.

[27] A. Benharref, M. Serhani, S. Bouktif, and J. Bentahar, "A new approach for quality enforcement in communities of web services," in *Proceedings of the 2011 IEEE International Conference on Services Computing*, Washington, DC, USA, 2011, pp. 472 – 479.

[28] M. Serhani and A. Benharref, "Enforcing quality of service within web services communities," *Journal of Software*, vol. 6, no. 4, pp. 554 – 563, 2011.

[29] M. Serhani, R. Dssouli, H. Sahraoui, A. Benharref, and M. Badidi, "Vaqos: Architecture for end-to-end qos management of value added web services," *International journal of Intelligent Information Technologies*, vol. 2, no. 4, pp. 37 – 56, 2006.

[30] D. Banerjee and S. Sen, "Reaching Pareto-optimality in prisoner's dilemma using conditional joint action learning," *Autonomous Agents and Multi-Agent Systems*, vol. 15, no. 1, pp. 91 – 108, 2007.

[31] [Online]. Available: http://www2.econ.iastate.edu/classes/econ308/tesfatsion/axeltmts.pdf

[32] [Online]. Available: http://www.slideshare.net/amitsinha1964/prisoners-dilemma-and-axelrod-experiment

[33] B. Khosravifar, J. Bentahar, M. Gomrokchi, and M. Alishahi, "Collusion-resistant reputation mechanism for multi-agents systems," in *The 2nd International Conference on Ambient Systems, Networks and Technologies*, Niagara Falls, Ontario, Canada, 2011, pp. 181 – 189.

[34] M. M'hamdi and J. Bentahar, "Scheduling reputation maintenance in agent-based communities using game theory," *Journal of Software (Accepted)*, 2011.

[35] P. Parachuri, J. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus, "Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, Richland, SC, USA, 2008, pp. 895 – 902.

[36] P. Paruchuri, M. Tambe, F. Ordonez, and S. Kraus, "Security in multiagent systems by policy randomization," in *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, Hakodate, Japan, 2006, pp. 273 – 280.

[37] D. Fudenberg and J. Tirole, *Game theory*. MIT Press, 1991.

[38] V. Conitzer and T. Sandholm, "Computing the optimal strategy to commit to," in *Proceedings of the 7th ACM conference on Electronic commerce*, Ann Arbor, Michigan, USA, 2006, pp. 82 – 90.

[39] J. Pita, M. Jain, F. Ordonez, C. Portway, M. Tambe, C. Western, P. Parachuri, and S. Kraus, "Using game theory for Los Angeles airport security," *AI Magazine*, vol. 30, no. 1, pp. 43 – 57, 2009.

[40] A. Rubinstein, *Modeling Bounded Rationality*. MIT Press, 1998.

[41] J. C. Harsanyi and R. Selten, "A generalized Nash solution for two-person bargaining games with incomplete information," *Management Science*, vol. 18, no. 5, pp. 80 – 106, 1972.