Argumentation Based Decision Making for Trust in Multi-Agent Systems

Ruben Stranders

Delft University of Technology

Argumentation Based Decision Making for Trust in Multi-Agent Systems

Master's Thesis in Computer Science

Parallel and Distributed Systems group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Ruben Stranders

June, 2006

Author
Ruben Stranders

Title
Argumentation Based Decision Making for Trust in Multi-Agent Systems

MSc presentation
June 30th, 2006 at 16.00
Snijderszaal, Mekelweg 4, Delft

Graduation Committee

prof. dr. ir. H. J. Sips (chair)	Delft University of Technology
prof. dr. C. Witteveen	Delft University of Technology
dr. M.M. de Weerdt	Delft University of Technology
dr. T.B. Klos	Center for Mathematics and Computer Science
dr. drs. L.J.M. Rothkrantz	Delft University of Technology

Abstract

Agents in Multi-Agent Systems depend on assistance from others to attain their goals. Often, goals of agents conflict with each other, and agents can be unreliable or deceitful. Therefore, rational agents embedded in an open Multi-Agent System need to be equipped with algorithms to reason about trust.

Currently, existing algorithms mainly focus on performance of these algorithms in terms of utility, but do not provide explanation of their actions. This might hinder acceptance of agent-based technologies in sensitive applications where users rely on their personal agents.

In this thesis, we therefore propose a new approach to trust in Multi-Agent Systems based on argumentation that aims to expose the rationale behind these decisions. Our solution features a clear separation of opponent modeling and decision making. It uses fuzzy logic to model behavior of opponents, and argumentation to translate these models into well-supported decisions.

Preface

Ever since reading the book "Trust" by Fukuyama (1996) in the summer of 2002, trust as a social phenomenon has fascinated me. In this book, the author synthesizes economics, politics, social sciences, and shows how trust is a major factor in a society's welfare. Trust is intangible, vague, and complex, but it has such a profound influence on almost every aspect of human interaction: from personal relations on a small scale, to entire economies on a larger scale.

So, when my supervisor Mathijs de Weerdt showed me a list of possible topics for a Master's project in the Collective Agent-Based Systems (CABS) research program, and trust between agents was one of the options, my choice was obvious. Studying trust in Multi-Agent Systems would not only give me the opportunity to learn more about trust, but it would also allow me to do this while pursuing another passion: my studies in Computer Science.

Acknowledgments

This thesis is the fruit of my Master's project. The completion of this project would not have been possible without the help and support of many people. First and foremost, I would like to express my gratitude to my supervisor Mathijs de Weerdt for his input, inspiration, and advice, and for putting me back on track in times of adversity. I am also indebted to my supervisor Cees Witteveen, whose critical remarks helped me to improve the quality of my work significantly.

Furthermore, I would like to kindly thank Robert Babuška of the Delft Center for Systems and Control for taking time to help me select a fuzzy learner suitable for our approach, and providing references to relevant papers. To Henry Prakken at the Intelligent Systems Group at Utrecht University we are grateful for his advice on argumentation frameworks. Also, thanks to Tomas Klos at the Center of Mathematics and Computer Science for his advice on the ART Testbed, and taking time to discuss my project. Sicco Verwer, thank you for disambiguating a detail in the specification of one of the algorithms I used, which caused a nasty bug in my implementation.

We would like to thank Peter Kok for sharing his data sets, and many

lunches. Paul Klapwijk, thank you for your support in the summer of 2005, and your friendship. My sincere apologies to all my friends who I neglected during this project, and had to endure my impatience. They were still kind enough to support me.

Lastly and most importantly, I would like to thank my parents for their tireless support, and for reminding me that the journey is the goal, which must be taken one step at a time.

Ruben Stranders

Delft, The Netherlands
June 15, 2006

Contents

List of Figures

HPS Listings

1 Introduction

Imagine that some day, technology has progressed to the point that we rely on artificial intelligence for basic medical treatment. Suppose one morning you get up and feel somewhat queasy. It is a mix of symptoms you have not experienced before. Naturally, you walk up to a view screen and consult a virtual medical consultant.

Assume you have two different medical consultants installed: MEDIBOT and PHARMAGENT. From the user's point of view, they are very similar, because they require the same input. Both need a description of your symptoms and your medical history. After you entered all information the agents need to know, they roam the Internet to consult medical databases, and perhaps even other similar agents. It is in the presentation of their findings, however, that MEDIBOT and PHARMAGENT differ.

After investigating all available information, MEDIBOT replies: 'You should take Medicine X'. Apart from its quite blunt way of reporting your prescription, would you *trust* MEDIBOT? In its prescription, it does not refer to your symptoms, to the source of the information it has used, or to rejected alternatives. You have no idea how MEDIBOT reached its conclusion.

Let us assume for a moment that MEDIBOT's manufacturer acknowledges these weaknesses and after an extensive survey of user experiences they develop MEDIBOT 2.0. The difference between MEDIBOT 2.0 and its predecessor is that it not only presents alternatives to 'Medicine X', but also indicates the suitability of all these alternatives to your specific condition with a number between 0 and 100. Again, we would like to ask you: would you *trust* MEDIBOT 2.0?

Granted, MEDIBOT 2.0 is better than its predecessor, because it gives

you more information. However, you still do not know how why 'Medicine X' is best or where those numbers came from. From your point of view, you are little better off.

PHARMAGENT is different in this respect. It might support MEDIBOT's conclusion that you should take 'Medicine X' to counter the symptoms you are experiencing. This, however, is the only thing MEDIBOT's and PHARMAGENT's prescriptions have in common. PHARMAGENT explains *why* it thinks this pharmaceutical is your best option. To do this, it might refer to your symptoms, to medical data it deems relevant for your condition, and explains why the alternatives are less suitable. For the last time we ask our question: do you *trust* MEDIBOT?

Let us rephrase this question. Do you trust MEDIBOT *more* than PHARMAGENT? In contrast to MEDIBOT, PHARMAGENT gives you the idea that it has *diagnosed* your condition before *prescribing* a medicine. It has given you insight into the rationale behind its decision: the *reasons why* Medicine X is best. Its 'thought process' so to speak, is verifiable to you.

The scenario above has illustrated the importance of *explaining* the user why certain decisions were taken. This increases the user's trust in an agent (in this case a virtual consultant). Trust in an agent's actions is especially important when the nature of the task requires a more delicate approach. For example, if you were interested in the most nutritious cereal instead of the most suitable drug, the issue of whether or not you trust your agent's judgment would be far less important.

In this thesis, we are very interested in the trust an agent lies in other agents. Your agent might not be capable of performing the task you sent it out to do on its own. In that case, it needs to consult other agents (as in the scenario above), or delegate parts of the task. No one is satisfied with the answer 'someone said you should take Medicine X', so your agent really should explain why the judgment of particular agents is trusted.

1.1 Scope

The research presented in this thesis pertains to trust in open Multi-Agent Systems (MAS). As it name suggests, a MAS consists of multiple agents that are in contact with each other. Agents are entities that strive to achieve goals on behalf of their owners. In the scenario above, the owner was a regular user, but could also have been a company or an institute. Examples of contexts in which these agents operate and where trust plays a role of importance are Computer-Supported Cooperative Work (CSCW) (Barthés and Tacla, 2001), web services (Maximilien and Singh, 2002), e-Business (Resnick and Zeckhauser, 2000; Guttman et al., 1998), and Human-Computer interaction (Castelfranchi and Falcone, 2001).

Agents are generally autonomous, in that they are situated in an environment, are capable of sensing this environment, and making decisions based on what it believed to best bring about its goals, without intervention from the user (Franklin and Graesser, 1996). The predicate *autonomous,* however, should not be confused with *independent*. In most cases, agents in a MAS depend on other agents to fulfill their tasks. They need to delegate (parts of) their plans to others, because they themselves are incapable of executing them, or the cost of doing it themselves far exceed the cost of delegation.

Open Multi-Agent Systems are characterized by the freedom of agents to enter and exit the system as they please, making the system vulnerable to agents that exit, switch their identity, and enter again. Combined with the lack of independency of agents, this introduces a number of significant problems for both the agent, and its user. This is due to the fact that agents in a system of such kind are not reliable or trustworthy by default. An open MAS is not always regulated by a trusted third party that imposes and enforces rules of conduct or protocols that agents in the system should conform to. This being the case, agents need to take into account the *trustworthiness* of other agents when planning how to satisfy their owner's demands.

Several algorithms have already been devised to confront these issues. These algorithms primarily focus on reducing the possibility of less-than-satisfactory results from other agents. Less emphasis is laid on discovering patterns in the behavior of other agents or, —more challenging—their motives and incentives (or goals). Also, the *rationale* of the decision to delegate or trust often eludes the user: it is 'hidden' in a large amount of numerical data, or simply incomprehensible. At any rate, these approaches do not provide human readable information about these decisions, and were indeed not designed to do this.

In light of the scenario sketched at the beginning of the introduction, the incapability to explain this rationale, is a significant drawback of the current state of research on trust in MAS.

1.2 Research Objective

Because of this, we have made the process of finding a solution to this problem the center topic of our research. In this thesis, we explain how we used existing approaches as a starting point, designed an architecture that enables an agent to explain its actions, and designed and implemented a proof-of-concept.

Based on these goals, our research objective is formulated as follows:

> A prerequisite for acceptance of agent-based technology in areas where humans rely on agents to perform sensitive tasks, is

the ability of agents to explain why decisions were made. In open Multi-Agent Systems, agents rely on one another. A decision to trust other agents is therefore also subject to this requirement. Currently, research on trust in Multi-Agent Systems has not sufficiently addressed this issue. In this research, we explore the requirements for a new approach to trust in Multi-Agent Systems that lays emphasis on the rationale of trusting decisions, and work towards a proof-of-concept.

1.3 Contributions

The main contributions presented in this thesis to the research on trust in Multi-Agent Systems are summarized as follows:

- Extending existing requirements for trust algorithms with a new requirement that persuades researchers to make the rationale of decisions available, preferably in a format that is human comprehensible.
- Designing an architecture based on the clear subdivision between modeling and decision making, facilitating the reconstruction of the process in which a decision is brought about.
- Using fuzzy logic as the primary knowledge format to describe behavior of agents. This format is used by both the opponent modeling and decision making components.
- Using argumentation to weigh the pros and cons of delegation actions based on a fuzzy model of opponents.

1.4 Overview

In general, agents are not capable of reaching their goals without assistance from others. Often, the agent's goals conflict with each other, or agents are unreliable or deceitful. Therefore, rational agents embedded in an open Multi-Agent System need to be equipped with techniques to reason about trust. In the scenario outlined in the introduction, we noticed that techniques solely focused on finding a viable solution without explanation might not encourage acceptance of agent-based technology in sensitive applications. Our approach therefore aims at exposing the rationale behind decisions pertaining to trust in other agents. This creates extra requirements for the internal operation of an agent, but also increases the user's trust in the agent's decisions.

The remainder of this document is subdivided as follows. In Chapter 2 we study and discuss how related existing research solve the problem

of trust in Multi-Agent Systems. We investigate the gaps in the state-of-the-art, and show which elements of this research can aid in achieving the research objective laid down in Section 1.2. In Chapter 3 we translate this aim into a set of requirements the design should conform to. Based on these requirements, we present a framework based on a clear separation between opponent modeling and decision making. Both these components use a symbolic knowledge format. This framework enables the user to get a better insight into the reasons behind decisions made by the agent, than was previously possible with current research. Chapter 4 discusses experimental results. The thesis closes with Chapter 5, in which we present our conclusions and recommendations for future work.

2
Related Work

The aim of our research is to explore the requirements for a new trust approach focused on explaining the trusting actions of an agents. This aim (defined in Section 1.2) differs from the aim of existing research on trust in Multi-Agent Systems. Most approaches are primarily concerned with the quality of an agent's decisions, and less on human comprehensibility. Studying existing research is therefore important to understand which circumstances led to our different aim. Moreover, studying related approaches can help us find a solution to our problem.

In order to do this, we evaluate these existing approaches to identify their strengths and weaknesses. If indeed no suitable solution to our problem can be found in these existing approaches, we can attempt to combine the strengths in a new solution, and find a way to prevent the pitfalls.

In this chapter, we distinguish between the opponent modeling and decision making components of a trust algorithm. An opponent modeling algorithm uses observed behavior of opponents to build a behavioral model. In turn, the decision making component uses the behavioral model to decide which trusting-action towards it is most prudent. The distinction between opponent modeling and decision making is also used in the evaluation of trust algorithms in the literature survey by Stranders (2004) (from which most of the material of this chapter is derived), and by Fullam et al. (2004).

At the end of the chapter, we hope to have a clearer image of how to approach our own problem, based on both the capabilities and shortcomings of related work. Put differently, we hope to know how to fill the gap that spans between our desired solution, and existing research.

The remainder of this chapter is sectioned as follows. To evaluate ex-

isting work in the field of trust in Multi-Agent Systems, we need a basic understanding of both trust and related concepts. These concepts are therefore discussed in Section 2.1. Next, we discuss the two major components of a trust algorithm, and show how these components are implemented in existing research. Opponent modeling is discussed in Section 2.2, and decision making is discussed in Section 2.3. We draw our conclusions in Section 2.4.

2.1 Trust

As said above, the study of related work on trust in Multi-Agent Systems requires an understanding of trust itself. In this section, we therefore focus on trust, before continuing with existing research.

Trust is a very intricate social phenomenon. Perhaps because of this, the meaning of the word 'trust' is not only vague and ambiguous in daily life, but also in Multi-Agent Systems (Jones, 2002). In this section we present a definition of trust that is used throughout the rest of this thesis. A precise definition of trust (in particular trust in Multi-Agent Systems) helps us demarcate the area of interest, and can serve as a guideline for evaluating existing approaches, as well as our own.

We depart from the notion of a rational agent, and explain why trust is needed to cope with the complexities of interacting with other agents. Next, we show which of three characterizations of trust from a socio-cognitive point of view is most applicable in MAS. After presenting some essential properties of situations in which trust is used, we give a definition of trust. Finally, we briefly explain the related notion of reciprocity.

2.1.1 Rational Agents

This thesis is about trust in Multi-Agent Systems. It is therefore important to have at least some familiarity with the (rational) agents that occupy a Multi-Agent System. The most important aspects of agency are briefly summarized here.

Wooldridge and Jennings (1995) state that agents posses the following properties:

autonomy An agent should be able to choose its own actions, independently of others.

pro-activeness An agent should not only be reacting to its environment, but plan ahead. In other words, its actions should be directed towards a goal.

reactivity Even though an agent is goal-directed, it should be prepared for possible unexpected events that could thwart the execution of its plan.

social ability When multiple agents inhabit the same environment, it is possible that their actions interfere with each other. In this event, agents should be able to communicate to resolve conflicts and to cooperate when possible.

Apart from these properties, *rational* agents also have the property of being *self-interested*. This means that they have a very narrow perception of utility, as they only assign value to their own goals and not to goals of other agents. Working towards a joint goal (shared among multiple agents) is therefore not one of the likely activities of a rational agents if this is not the best alternative in terms of the payoff the agent expects to receive.

At first glance, rationality seems simple: each time the agent has to act, it chooses the action that leads to the most profit. But which action is this? And which actions does the agent have at its disposal? What if a task can not be performed by the agent itself, and has to be delegated to another? Which agents will perform the task well? These questions make the problem more complex than it may seem. Fortunately, *trust* can help an agent faced with the problem of answering these questions (especially the last two).

2.1.2 What is trust?

When an agent is faced with these complex questions, trust can be a tool to simplify decision making. More specifically, it can guide an agent in a certain direction, limiting the number of decisions that have to be considered. For example, a trust algorithm can interact with the planning algorithm of an agent. The latter attempts to find plans that can bring about the agent's goals. However, this can be done in a multitude of ways. When a trust algorithm introduced, it can limit the number of potential agents to which a task is delegated, since certain agents are deemed untrustworthy. This way, the number of plans that have to be considered can be reduced.

In short, trust can be used to help the agent limit the number of delegation partners. This is what trust in Multi-Agent Systems *does*. We have not yet discussed what trust *is*. Unfortunately, a lot of definitions and characterizations of trust exist. Using the literature survey by Stranders (2004), we elaborate on a single representative viewpoint here.

A leading research group on trust, led by Castelfranchi, advocate a socio-cognitive approach of trust. They state that trust is at the same time: a *mental attitude* towards another agent, a *decision* to rely on another, and a *behavior* (Falcone et al., 2004):

- Trust as a *mental attitude* is most common in daily life, and is based on *evaluation* of past behavior, and on the *expectation* of future behavior.
- Trust as a *decision* (the act of *entrusting* a task) puts a part of the trusting agent's welfare on the line and thus involves *risk*: however satisfactory the transaction history with another agent might be, it is never guaranteed that this will continue in the future.
- Lastly, viewing trust as a *behavior* emphasizes the actions (delegating and monitoring) of trusting agents and the relation between them. This relation generally intensifies as time progresses.

The notion of trust as a *mental attitude* gives us an important clue of how to determine the trustworthiness of others: we need to analyze past interactions with the agent. Not surprisingly, this is exactly what the majority of trust algorithms do.

2.1.3 Conditions for the applicability of trust

Trust can be a useful tool. However, it can not be under all circumstances. In some situations, trust has no function or meaning. Studying the preconditions for the applicability of trust could therefore lead to a better understanding of its function, and lead to a better definition later on. These conditions are (Stranders, 2004):

- Uncontrollablility. The more we can control the entity we rely on, the more we can predict its behavior, the more we are capable of determining the end-result (and increase our knowledge about of the expected behavior). So, the less the entity is controllable, the more we need trust to cope with uncertainty.
- Partially monitored. This condition is related to the previous one. If we monitor everything the entity does, uncertainty about the quality of the end result is significantly reduced, thereby reducing the need for trust.
- Private information. Private information is information that is private to an agent. Examples of private information are the agent's capabilities, its trusting-behavior (or decision making rules), and its disposition towards other agents. In the absence of private information, an agent would be highly predictable (the agent becomes a white box), making trust unimportant.
- Dependency (Poli and Hexmoor, 2002). Fully independent agents do not need to rely on other agents to attain their goals. Dependency is therefore a requirement of trust.

- Goal-oriented (Castelfranchi and Falcone, 2001). The utility (or welfare) of an agent is always relative to a goal. Without goals, nothing is at stake, and an agent could not care less if a task is performed well or not. Consequently, trust is irrelevant in the absence of goals.

Note that the first three properties refer to the uncertainty about the trusted agent, whereas the last two properties relate to the trusting agent itself.

2.1.4 Definitions

Using the properties of, and preconditions for trust discussed in this is section, Stranders (2004) defines trust as:

Definition 1 (Trust)**.** *The expectation that one will not be deceived when relying on an entity one does not control completely.*

The notion of *reciprocity* is related to trust. In short, reciprocity pertains to the exchange of favors, and can be the foundation for a trusting relationship. In fact, during our experiments, we use an agent that is based on the principle of reciprocity (see Section 4.3.3.2).

Definition 2 (Reciprocity)**.** Reciprocity *is the state of being reciprocal: cooperation is returned by cooperation, while defection is returned by defection. Reciprocity is a* means *of establishing cooperation and trust, and could be instrumental to the achievement of selfish goals.*

2.2 Opponent Modeling

As said in the introduction of this chapter, opponent modeling is one of two necessary components in a trust algorithm. It is responsible for detecting patterns in an opponent's behavior, and expressing these patterns in an opponent model.

In this section we review four different existing opponent modeling-algorithms. These algorithms are part of existing approaches to trust. The approaches themselves are not limited to opponent modeling, but at this time we are only interested in their opponent modeling components. These approaches are: FIRE by Huynh et al. (2004), Dempster-Shafer by Yu and Singh (2002), an approach using probabilistic reciprocity by Sen and Dutta (2002), and an approach using Bayesian analysis by Mui et al. (2002).

In order to build an opponent model of an agent using only information from direct interaction, these approaches share the same decomposition in three aspects:

1. A definition of trust, which specifies how trust is defined in a particular context. This definition can help determine which features of a transaction are relevant for obtaining a concrete trust measure for the agent.

2. A transaction memory, in which (parts) of the transaction history with the agent are stored. Maintaining a database of transaction history is an important aspect of an opponent modeling algorithm. On the one hand, using too much history causes the obtained model to be too 'inert': it reacts too slow to changes in behavior. On the other hand, using too little history causes the obtained model to be unreliable: when taking into account only very recent transactions, it is not very likely we accurately capture the behavior of the opponent.

3. An algorithm to aggregate the transaction history into a behavioral model. This model can be represented in various forms, ranging from a simple scalar or a vector to a more complex rule base. The nature of an aggregation algorithm depends on the structure and the kind of information in the transaction history, which in turn depends on the used notion of trust.

In the following, we use these three aspects to compare the aforementioned existing approaches. For a more detailed coverage of opponent modeling approaches, we would like to refer to Stranders (2004).

2.2.1 FIRE

Trust Definition Huynh et al. (2004) define trust as a measure for the *quality* and the *reliability* of the transaction-results. Put differently, trust towards an agent is high if it sustained a high quality of service (QoS) over a long period of time.

Transaction History To find a balance between the extremes of having too much or too little transaction history, FIRE is equipped with a discount rate: recent transactions are assigned higher weights than transactions that happened a long time ago. Of the four approaches discussed here, this is the only one that distinguishes between recent and older transactions.

Aggregation Algorithm The aggregation algorithm calculates two values: interaction trust $\mathcal{T}_I$ and reliability $\rho_{\mathcal{T}_I}$. The value of $\mathcal{T}_I$ is determined by multiplying the quality of the results of each transaction with its discount-factor. The reliability $\rho_{\mathcal{T}_I}$ is calculated using both the amount of evidence $\mathcal{T}_I$ is based on, and the deviation in the quality ratings. These two values together form the opponent model that can be utilized to predict future transactions.

2.2.2 Dempster-Shafer

Trust Definition In the approach of Yu and Singh (2002), trustworthiness increases as the amount of evidence of trustworthiness increases. The evidence comes from past transactions: a satisfactory transaction result is positive evidence of trustworthiness, whereas an unsatisfactory result is evidence of the contrary.

Transaction History Yu and Singh (2002) only use the last n transactions, and treat these equally.

Aggregation Algorithm In the aggregation algorithm, the Dempster-Shafer theory of evidence is employed to obtain a measure of trustworthiness. This theory dictates that evidence in favor of and against a theory $\{T\}$ should be explicitly reviewed. In this case $\{T\}$ denotes "the agent is trustworthy". Transactions with 'satisfactory' and 'unsatisfactory' quality are considered evidence in favor of theory $\{T\}$, and $\{\neg T\}$ respectively. The predicates 'satisfactory' and 'unsatisfactory' are defined by two thresholds: quality below the lower threshold is unsatisfactory, and quality above the upper threshold is satisfactory. Transactions with a quality that falls in neither category add to the uncertainty (this category is denoted by theory $\{T, \neg T\}$). The opponent model obtained in this fashion takes the form of a basic probability assignment (bpa), which assigns a probability to each possible theory.

2.2.3 Probabilistic Reciprocity

Trust Definition The interpretation of Sen and Dutta (2002) of trust is closely linked to the notion of reciprocity (see Definition 2). Trust towards an agent is low if that agent still 'owes' us a great deal. If the agent repays its debt towards us, trust increases again, and the probability of accepting a delegated task increases. Note that this notion of trust is slightly different from Huynh et al. (2004) and Yu and Singh (2002), because it is used for determining whether or not to accept a task *from*, rather than delegating a task *to* an agent.

Transaction History Sen and Dutta (2002) also do not distinguish between transactions, and use the complete transaction history.

Aggregation Algorithm This approach uses reciprocity balances to keep track of the amount of favors (or utility) provided and received from an agent. If a balance is positive, we owe the agent. When a task is performed for another agent, costs are incurred which reduce reciprocity bal-

ance. When the other agent performs a delegated task, the savings obtained increase the reciprocity balance.

2.2.4 Bayesian Analysis

Trust Definition According to Mui et al. (2002), trust is the probability that task delegation towards an agent is successful. This notion of trust is very similar to our own definition.

Transaction History Mui et al. (2002) use at least n transactions, where n is determined by statistical analysis to ensure a certain confidence interval.

Aggregation Algorithm The transaction history between two agents is seen as a repeated Bernoulli trial with probability p. From this perspective, the results of a transaction are 'good' with probability p, and 'bad' with probability $1 - p$ (the definition of 'good' and 'bad' are not given). The problem now reduces to determining this probability p of obtaining a 'good' result in the next transaction. This probability is estimated using a Beta distribution and the proportion $\frac{g}{n}$ as estimator (g and n being the amount of successful transactions and the size of the transaction history respectively). The probability obtained in this manner, together with the error bound, function as the opponent model.

2.3 Decision Making

The decision making part of a trust algorithm translates the model obtained during the opponent modeling phase into actual decisions. For example, an agent needs to decide whether or not to delegate a task to another. To do this, the decision model needs to know the agent's goals and risk-profile regarding to trust.

The decision making algorithms discussed here are more diverse than the opponent models we discussed earlier. It is therefore not possible to break down all these decision models into similar components. However, we try to categorize these models as best as possible.

2.3.1 Simple

Simple decision models operate on opponent models encoded as numerical vectors. Elements of these vectors describe different features of both the agent and the quality of delivered services or goods. We saw examples of these kind of opponent models in the previous section.

Of the approaches we discussed earlier, only Yu and Singh (2002) and Sen and Dutta (2002) explicitly define decision models.

Remember from the previous section that Yu and Singh (2002) use a basic probability assignment to assign probability to theories. Their decision model recommends delegating a task to the agent iff the difference of probability assigned to the theories $\{T\}$ (the agent is trustworthy), and $\{\neg T\}$ (the agent is untrustworthy) is greater than a predefined threshold.[1]

Sen and Dutta (2002) use a (rather complex) function that takes the reciprocity balance of the agent, and the cost of performing a new task offered by that agent to decide whether or not to accept it. The lower the reciprocity balance (the more the agent owes us), and the higher the extra cost of the task, the lower the probability of accepting the task. Unlike others, this decision model determines not whether to delegate, but rather whether to accept a task for another agent. Either way, it is still a decision to trust another agent with a part of our utility.[2]

2.3.2 Socio Cognitive Decision-models

The so-called socio-cognitive perspective of trust is advocated by a research group led by Castelfranchi. It is based on the belief that trust is cannot be reduced to a simple subjective probability. Rather, trust is a complex of beliefs the trusting agent has about others. Together, these beliefs form a 'model of the mind of the other', or—in our terminology—an opponent model.

In Castelfranchi and Falcone (2001), an opponent model consisting of seven different beliefs is used to create a decision model. Examples of these beliefs range from a competence belief (do I believe that the other agent is able to perform the task at hand?), and a willingness belief (do I believe that the other agent has some incentive to perform the task?), to the self-confidence belief (do I believe that the agent itself believes that it can perform the task?).

The reason this opponent model was not discussed in the previous section, is that Castelfranchi and Falcone (2001) do not present details of how the opponent model is obtained (i.e. how to determine the strengths of the beliefs). As such, the approach discussed in their paper cannot be classified as an opponent modeling algorithm. Their decision model, however, can be studied independently.

The decision model itself is a decision tree. Edges represent decision choices of the trusting and trusted agent. Leafs are outcomes and are labeled with the utility gained in the outcome-scenario. Using a function, the strength of the individual beliefs are combined in a so-called 'Degree of Trust'. This measure is subsequently used to determine the expected utility

[1] More formally, if m is a bpa, the decision to delegate is recommended iff $m(\{T\}) - m(\{\neg T\}) > \rho$.

[2] Whether an agent delegates a task or performing a task for another, it receives savings or incurs cost. In both cases, the agent's utility is at stake.

for delegation and non-delegation (or, in plain English: doing it yourself). Put differently, the Degree of Trust is used to determine the probability of a successful delegation attempt. A *necessary* condition of delegation is that the expected utility for delegation is strictly greater than the expected utility for non-delegation.

This condition is augmented with a risk-acceptance strategy, which determines the amount of risk the trusting agent is willing to take by delegating the task to the other agent.

Falcone et al. (2003) and Castelfranchi et al. (2003) use a similar decomposition of trust in separate beliefs. Additionally, beliefs are classified in 'internal' and 'external' attributable beliefs. The former beliefs are beliefs about the trusted agent itself, whereas the latter beliefs are concerned with the appropriate external conditions for performing the task (i.e. the environment).

Instead of a decision tree, Falcone et al. (2003) use a Fuzzy Cognitive Map (FCM). In general, an FCM can be used to analyze a very complex system that is otherwise difficult to understand because of complex and vague relations between its components. It is argued that an FCM closely approximates how human reasoning deals with a system of interconnected concepts, because it too uses vague and fuzzy relations to get a grasp on complex systems.[3]

Falcone et al. (2003) use an FCM to analyze the effect of the strengths of individual beliefs on trust itself. Using the strengths of individual beliefs, and the relations between them, Falcone et al. (2003) employ an FCM to obtain an aggregated measure of trust.

2.3.3 Doxastic Logic

The decision model described in Liau (2003) allows explicit reasoning about trust. His method employs a doxastic logic[4] called BIT to encode rules and facts in the trust domain. BIT stands for Belief, Inform and Trust. BIT is obtained by extending traditional doxastic logic with operators for information and trust.

With these extra operators, it is possible to not only denote an agent i's belief of a fact φ (usually denoted as $B_i\varphi$), but also information about the acquiring of information from another agent j (denoted as $I_{ij}\varphi$), and i's trust in j's judgment regarding the validity of fact φ (denoted as $T_{ij}\varphi$).

[3] See `http://www.ochoadeaspuru.com/fuzcogmap/tutorial.php` for a very interesting tutorial of Fuzzy Cognitive Maps. In this tutorial, the complexity of several complex systems is tackled with an FCM. Examples of these systems are the economy, a ecosystem with prey and predators, and the geopolitical situation in the Middle East.

[4] Doxastic logic is a modal logic of belief. *Doxastic* comes from the Greek word $\delta o\xi\alpha$: belief, opinion.

Using these doxastic operators, Liau (2003) creates an axiomatic system in which a basic set of rules about trust are encoded. This system can be used to infer basic trust-related facts from an agent's observation. A decision can be based on a fact φ, if, for example, it is possible to infer the validity of the fact, based on trust towards other agents. More importantly from our perspective, reversing this process is also possible: the reasons why a fact is believed is expressible in terms of trust towards other agents.

2.4 Conclusion

This concludes our study of related work. In this chapter, we worked towards a definition of trust, and explained under what conditions trust can be an instrument to deal with the complexities inherent in multi-agent delegation. Next, we studied existing research on trust algorithms. These algorithms were broken down in two major components. These components—opponent modeling and decision making—are necessary to establish trust towards agents with whom an agent directly interacts.

Using this decomposition, we evaluated and compared several existing approaches to trust. Our conclusions can be summarized as follows.

Opponent Modeling The opponent modeling algorithms discussed in Section 2.2 have two properties in common. First, they are fully developed and readily applicable. Put differently, all necessary details have been worked out, and these approaches have in fact been subject to experimentation. Second, they are simple. Simplicity is often a desired property, but in this case it has some disadvantages.

For one, the opponent models produced by these approaches are simple scalars, or small vectors at the most.[5] Because of the limited amount of information present in these models, much of the information gathered during interacting with an opponent is lost. Consequently, the decision models they support are quite limited.

In our opinion this simplicity does not do justice to the complex nature of trust. Their application is therefore limited to simple problem domains.

Decision Making Indeed, the decision models accompanied with these opponent models are limited to making very basic decisions. Moreover, in Section 2.3 we have seen that the rationale of these decisions is not available. The lack of sufficient information in the opponent models does not allow for more than simple decisions, let alone explanation.

[5]For example, in FIRE the opponent model consists of a 2-D vector with quality and reliability.

In that respect, the methods of Castelfranchi et al. are much more expressive. Their decomposition of trust in distinct beliefs opens up the possibility of implementing different intervention strategies, depending on the precise composition of trust, instead of just having a binary choice: delegation or non-delegation. On the down side, the reasons *why* an agent is trusted are still not very clear. In order to make these clear, we would need to be able to trace back the process that established a certain decomposition of trust for a specific agent. The extent to which this is possible highly depends on the supporting opponent modeling algorithm. Unfortunately, such an algorithm is much more complex than the ones we studied in Section 2.2.

So, to use these decision models, an opponent modeling algorithm is required that can calculate the strengths of different beliefs from a transaction history, and retain sufficient information to be able to supply an explanation why decisions are made. Unfortunately, Castelfranchi et al. give no pointers how such an opponent model should be designed.

The same is true for the approach of Liau (2003), but unlike the decision model of Castelfranchi et al., the modified doxastic logic BIT is more capable of explaining why certain facts are believed. For example, using BIT, an agent could be able to present the rationale of the decision to trust another. In terms of our aim, this is very appealing. The absence of a procedure that translates observations pertaining to trust to BIT, however, is not a trivial problem. Due to the inherent uncertain, vague and continuous nature of these observations it is not trivial to translate them into the propositions of modal logic. Modal logic has no 'native' support for *directly* representing such observations, so special care should be taken if one wants to accurately translate these observations to modal logic propositions.

In this light, the application of fuzzy logic in the approach by Falcone et al. (2003) is interesting. Vague and uncertain observations are more easily translated to fuzzy logic, while still having the possibility of reasoning using explicit rules (as we will see later on in this thesis).

In conclusion, our discussion of existing trust approaches shows that there exists a gap between powerful decision making approaches and opponent models that support them. The opponent models currently used can be applied to problem domains with limited complexity. On the other hand, the decision models we studied are capable of making more complex decisions (and in some cases explaining them), but still lack a full-fledged algorithm to supply a suitable opponent model.

In the next chapter, we use the identified strengths of the discussed approaches as a foundation for a trust approach based on fuzzy logic (like Falcone et al. (2003)) and argumentation. This approach combines an opponent modeling algorithm that is able to learn an explicit model of an opponent (as opposed to a numerical model), with the ability to make deci-

sions that are supported by arguments. These arguments enable the user to trace back the origin of the decision to parts of the opponent model (similar to BIT). This way, we attempt to close the identified gap between opponent modeling and decision making.

3 Design

In the previous chapter, we reviewed the state-of-the-art of trust in Multi-Agent Systems. It appears that these approaches share a major drawback: either they lack a a supporting opponent modeling algorithm, or they are unable to explain their decisions. Without an explanation, human operators are not able of understanding the reasons behind a decision. This drawback could hinder the acceptance of multi-agent technology in sensitive areas such as Human-Computer interaction, in which the human operator could be required to provide agents with personal information, or trust an agent with a part of their welfare (money, for example).

Therefore, we propose a new approach that is both capable of explaining decisions, and has a supporting opponent model. In this chapter we discuss the precise requirements for such an approach, and design a proof-of-concept.

The chapter is subdivided as follows. In the first section, we lay down the requirements that this design should conform to. These requirements are both related to our methodology (the process), and the design itself (the product).

Next, we describe our solution in two ways. First, we look at the solution as a black box, in order to explain *what* our proposed solution is supposed to do in terms of input-output behavior. Second, we take a more in-depth look, and describe the internal components of the design. Both perspectives combined should help to get a high-level view on our approach for the rest of the chapter, in which we focus on each of the components that together compose our solution: the black box description allows the reader to see the big picture, whereas the white box description helps the reader to understand the function of each individual component in the system.

For each component, we explain why the selected algorithms and techniques were chosen over the alternatives. We close the chapter by explaining how these components are combined to function in harmony.

3.1 Requirements

The requirements of the project are subdivided into two categories: methodological requirements and requirements of the final product.

3.1.1 Methodological Requirements

The methodological requirements pertain to the *process* of obtaining the end result, and are important in light of the explorative nature of this project: we are interested in creating a proof-of-concept of an agent that is capable of explaining its trusting decisions, not in obtaining an agent whose goal is to perform optimally (not at this time anyway). Therefore, it is not only important to discuss the characteristics of our proposed approach, but also to understand the rationale behind the decisions that were made.

"Pluralitas non est ponenda sine necessitate"[1] is the motto that was central during the development. This statement summarizes the philosophy of the 14th century philosopher William Occam. This philosophy urges one to keep explanations of phenomena as simple as possible. Simple explanations are usually the most probable. Simplicity of a theory or approach improves its accessibility. An accessible theory is more open to criticism, than an unnecessarily complicated one. So, in light of our goal to show that an agent can be built that is capable of explaining its actions, we would like to allow criticism, and—hopefully—to show that our approach can withstand this criticism.

Translated to a concrete guideline, this means that we are looking for simple and suitable algorithms to proof our concept, and put less emphasis on performance and efficiency.

3.1.2 Design Requirements

As the final result is concerned, the goals defined in the introduction give rise to the following requirements:

1. The agent should be able to explain why certain decisions were made, and why alternatives were discarded,

2. formulate these explanations in terms of the perceived behavior of the agents in question, and

[1] Quite literally: "plurality should not be assumed without necessity"

3. present its explanations in a form that is comprehensible by human operators.

4. Because of requirements 1,2 and 3, the opponent's behavior as observed by the agent must be human readable (because it is part of the rationale behind decisions).

3.2 Black box description

In this section, we present a black box description: we focus on the *inputs* and *outputs* of our approach.

In this chapter, we propose a different approach to trust in Multi-Agent Systems. It allows an agent to explicitly weigh the pros and cons of a trusting decision and select the decision with the most desirable consequences.

The prediction of a decision's consequences is based on the observations of past performance of the agent itself, and other agents in the system: the information available in direct interaction. Therefore, these observations are the *inputs* of our approach.

The decisions and the rationale behind them form the *outputs* of the system. When requested, the agent is able to explain the rationale behind decisions made in the past and decisions that have yet to be made, by listing arguments in favor and against every decision under consideration, and show how these arguments were weighed to select the most appropriate decision(s).

This process of decision making is much more transparent to the agent's owner, by allowing insight into the rationale behind decisions.

3.3 White box description

The white box perspective shows how our approach achieves the input-output behavior described in the previous sections. In other words, we focus here on *processing*. Figure 3.1 shows the architecture of our agent. On a high level of abstraction, it consists of two components: opponent modeling and decision making.

The opponent modeling component is responsible for modeling the behavior of other agents, based on past experiences with these agents. These past experiences are stored in a transaction database. Data from the transaction database is used to identify behavioral patterns. This is done by applying a data mining algorithm. Together, the behavioral patterns form an opponent model—a description of how an opponent reacts in different situations.

As the name implies, the decision making component is responsible for making decisions. It uses the opponent models obtained in the pre-

vious step to predict the outcomes of each available action. Using these outcomes, and the knowledge used from the opponent model, arguments are constructed to support (or reject) the action. These arguments explicitly refer to the outcomes in terms of the agent's goals. The more the predicted outcomes are favorable in terms of these goals, the greater the strength of the argument supporting the action.

Based on this, the generated arguments can be paraphrased as follows: "when I select action a, the model that I have of the behavior of agent A predicts that the outcome is O, which conflicts with/attains positive goal G. Action a is therefore desirable/undesirable."

The final step in decision making is selecting the most appropriate action and executing it. In order to to this, the constructed arguments are evaluated using a decision criterion. The decision criterion specifies how arguments are weighed and compared to each other.

When the action has been executed, the actual outcomes are observed and recorded in the transaction database. These new results are subsequently used to refine the model of agent A once again, completing the circle.

In the upcoming sections of this chapter, we discuss each component in-depth. Opponent modeling is discussed in Section 3.4. Decision making is discussed in Section 3.5. After the coverage of the individual components, we show how they are interconnected to compose the complete system in Section 3.6.

3.4 Opponent Modeling

As said in the previous section, the first phase in our proposed method is to obtain a model of the behavior of the opponent agent. There are several ways to accomplish this, so we went through a selection process to identify a suitable means of representing the agent's knowledge about its opponent. This process consisted of three phases—identifying requirements, option generation, selection—which we would like to discuss briefly.

In the first phase, we focused on generating a list of requirements, that had to be met by the representation format of the opponent model. The format should:

1. be easily comprehensible by human operators,

2. be able to represent inherently uncertain, incomplete and ambiguous knowledge that is commonly found in the trust domain, and

3. be commonly used, to ensure the existence of sufficiently tested and accepted induction algorithms.

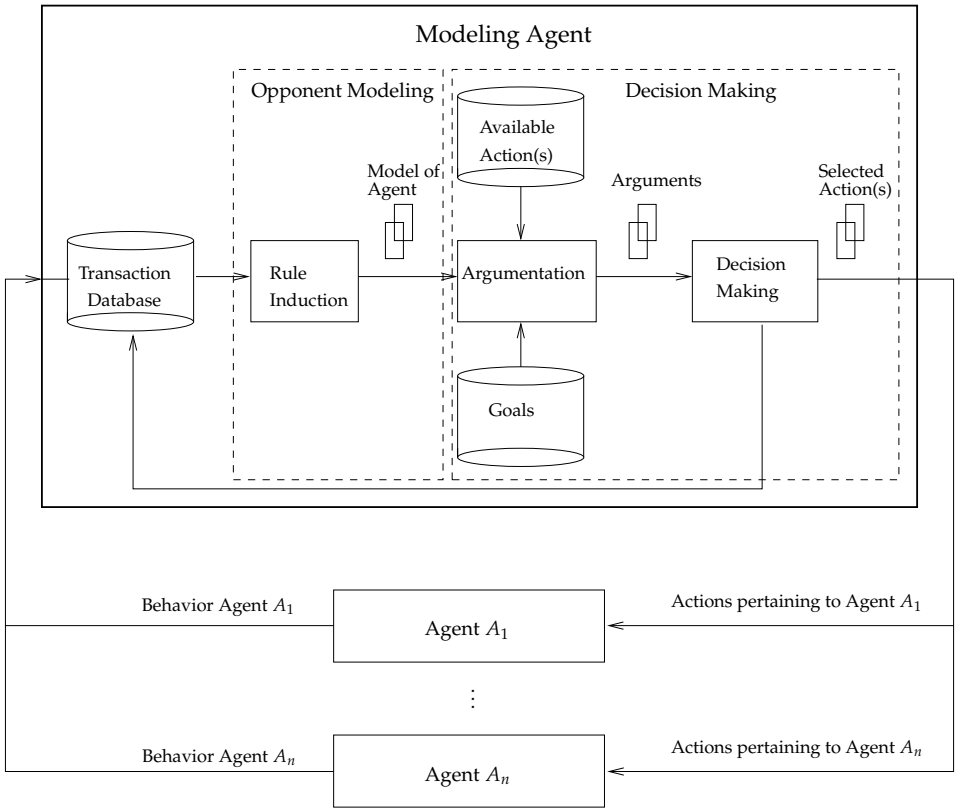


Figure 3.1: The architecture for the Modeling Agent

4. support an argumentation framework capable of making decisions and explaining them.

Based on these requirements, we identified two alternatives: defeasible logic and fuzzy logic.

Defeasible logic is a non-monotonic logic proposed by Nute (1993) to formalize a form of reasoning that is less stringent and formal than deductive reasoning. It represents knowledge using rules and a priority relation among these rules. If rules conflict with each other (i.e. they support opposite conclusions), the priority relation may help to resolve the conflict. If the conflict cannot be resolved, no conclusion can be derived.

On the other hand fuzzy logic is based on the concept of fuzzy sets. Fuzzy sets differ from 'classical' sets, in that the set boundaries are not sharp: objects can be partial members of a set. This allows for the representation of inherently vague and fuzzy knowledge (in Section 3.4.1, Fuzzy Logic is explained in more detail).

Both knowledge formats are explicit (or symbolic), in that they represent knowledge in a declarative form ('if-then' rules in this case), rather than implicit. In implicit knowledge formats, it is not possible to identify which part of the format contains a specific piece of knowledge. An example of an implicit knowledge formats is a neural network. We found implicit representation schemes to be violating the first requirement from Section 3.1.2 (i.e. human readable), because they store knowledge numerically, instead of as explicit rules.

We selected fuzzy logic over defeasible logic. The reason for this is twofold.

First of all, fuzzy logic was developed in 1965 by Zadeh (1996), and has won a lot of popularity since. As expected, this popularity coincides with a large body of research, and the development of many applications that use fuzzy logic. Of those applications, system identification is the one we are most interested in. As said before, the number of algorithms in fuzzy logic that is widely accepted, far exceeds the number of usable algorithms for defeasible logic.

Second of all, defeasible logic can not be easily adapted to operate on domains with ambiguous and continuous values. In the trust domain, agents are often faced with results that can be interpreted in a multitude of ways. On the one hand, transaction results are often not expressible in terms of just 'good' or 'bad', but in gray scales. Fuzzy logic is much more flexible in this respect, because it fades the boundaries for set membership.

The rest of this section is structured as follows. First we introduce fuzzy logic as a means to represent facts with an inherent uncertain and continuous nature. We explain why fuzzy logic is suitable to describe events in the trust domain. Second, we discuss the algorithm used to learn fuzzy rules

based on the facts we have gathered. This algorithm is called Fuzzy Rule Learner (FURL) and was developed by Rozich et al. (2002).

3.4.1 Fuzzy Logic

In this section, we study fuzzy logic applied to set theory and rule bases. In essence, fuzzy theory is an extension to classical set theory. In the first part of this section, we explain the difference between the classical and the fuzzy notion of a set. In the second part, we discuss the application of fuzzy set theory in rule bases. The material presented in this section is derived from Nguyen and Walker (1999).

3.4.1.1 Fuzzy Sets

In classical set theory, the boundaries of a set are well-defined and sharp. That is, an object is either a member of a set, or not a member. Such a set is also called a 'crisp' set. In mathematical terms, the membership function $\in$ of a crisp set can be written as follows:

$$\in(a, A) = \begin{cases} 0 & \text{if } a \text{ is not a member of } A \\ 1 & \text{if } a \text{ is a member of } A \end{cases} \tag{3.1}$$

Of course, a more common notation for the membership of an object in a set is the $\in$ *operator*: $a \in A$ if a *is*, and $a \notin A$ is *not* a member of set A.

In a fuzzy set however, the membership function $\in(a, A)$ can take other values than 0 or 1. More specifically, the range of $\in(a, A)$ is $[0, 1]$. In other words, an object a can be a partial member of a set A. This way the boundaries of a set become *fuzzy*. The consequence of this property of fuzzy logic that it deviates from classical bivalent logic because it contradicts the law of the excluded middle.[2]

The main benefit of using fuzzy sets, is that reasoning about continuous data becomes more intuitive. In fact, reasoning with fuzzy sets mimics the human way of reasoning about sets. For example, we do not think of predicates as 'old' or 'hot' as having very sharp boundaries.

When is someone or something old? The answer of this question of course depends on the context (for example a college student is considered too old to play with toys, but too young to retire). The question itself however cannot be answered with a clear-cut answer as 'children above the age of 12 are too old to play with toys'. We rather think in the gradual applicability of such a predicate, than in black or white. In Example 1 we show the use of fuzzy sets to describe the temperature.

[2]The law of exclude middle states that a statement p is either true of false, and that there is nothing in between. More formally: $\models (p \vee \neg p)$.

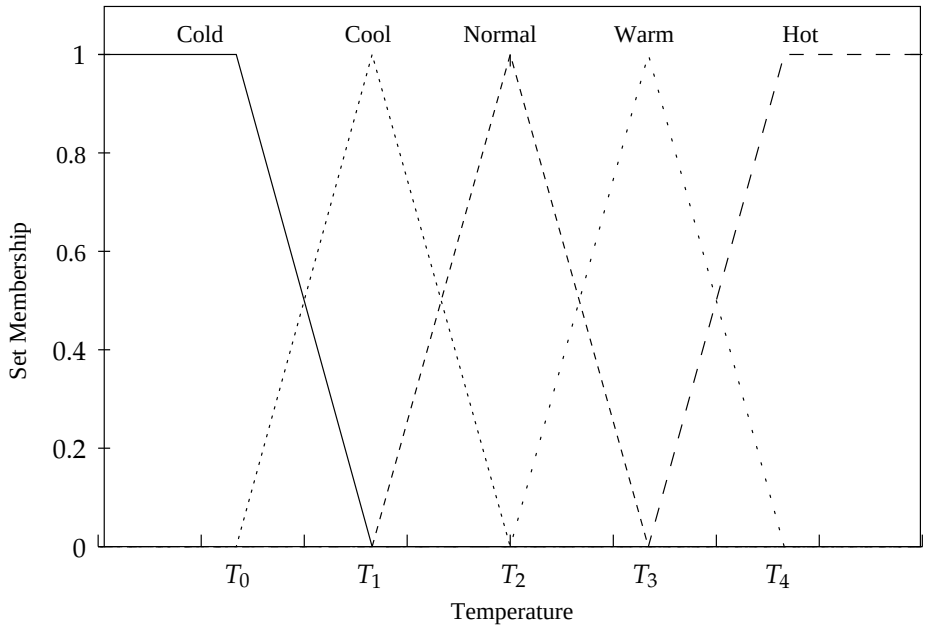


Figure 3.2: Fuzzy predicates for the temperature domain in Example 1

Example 1. *Figure 3.2 shows an example partitioning of the temperature domain in fuzzy sets. The predicates 'cold', 'cool', 'normal', 'warm', 'hot' are fuzzified to obtain corresponding fuzzy sets. The temperatures* $T_0, \ldots, T_4$ *are context dependent. For example,* $T_0 = 20$ *in the desert at noon, but* $T_0 = -15$ *in Amsterdam. Moreover, the boundaries of the sets also depend on the person using these predicates.*

In this example, a temperature T between T_0 *and* T_1 *is considered both 'cold', and 'cool' simultaneously to a certain degree. We would say something like 'it is not quite cool outside, but also not quite cold. Its somewhere in between'. The set memberships of the current temperature T in the fuzzy sets 'cold' and 'cool' will determine to what extent we consider T to be 'cold' or 'cool'.*

It is often desirable to convert a fuzzy value back to a numerical (or crisp) value. To bring about this conversion—also called defuzzification—a defuzzification algorithm is used. During this process information is lost, because a fuzzy value contains more information than a simple scalar.

Two simple defuzzification are the center of mass (or centroid), and the max-height method. As can be derived from its name, the former calculates the center of mass of a fuzzy value, whereas the latter determines the value of the variable where the set membership is at its maximum (or, in case of multiple maxima, it averages over all maxima). An example of the centroid defuzzification method is given at the bottom of Figure 3.4.

3.4.1.2 Fuzzy Rule Bases

Just as classical bivalent logic and classical set theory (on which bivalent logic is based) can be used to construct rule bases, it is also possible to build fuzzy rule bases using fuzzy set theory. A rule base is a set of rules that encodes knowledge about a (part of) a system. Most rule bases use simple if-then rules, with one or more antecedent clauses and a consequent clause. An antecedent clause is a condition for the rule to be applicable (or in rule base terms: fire), and are of the type 'x is Y', where x is a variable, and Y a value on the domain of x. Antecedent clauses thus make up the 'if' part of the rule. The consequent clause explains how the system the rule base describes or controls is expected to behave if the antecedents are true.

In fuzzy logic, however, truth is a continuous notion. The antecedents of a rule do not necessarily have to be 'true' or 'false'. In fuzzy logic, their truth can of course be somewhere between true and false. This poses the question how to deal with rules whose antecedents are not completely true in the classical sense. To solve this problem, fuzzy rule bases use a correlation function. A correlation function uses the truth of the antecedents to adapt the output value of the rule. A commonly used correlation function is the product function. This function scales the output value of the consequent set with the match strength of a rule. The match strength is the minimum truth value of the antecedents for the given inputs. It is thought that the match strength is a measure for the applicability of a rule for a specific set of inputs.

Example 2 shows an example of a fuzzy rule in an air conditioning system.

Example 2. *Figure 3.3 contains a graphical depiction of an example rule in an air conditioning system. It dictates that the air conditioning should set its cooling power to 'high' if the temperature is 'hot'. However, in this particular case the temperature is not 100% 'high', but rather somewhere along the lines of 30% (the match strength). The output value of the rule is therefore scaled to reflect its partial applicability to the current state of the world (i.e. the statement 'the temperature is high' is true for only 30%).*

After the output of each rule has been determined, a fuzzy rule base must deal with multiple firing rules that reference the same variable in their consequent, but have different outputs. To unite the outputs of these rules, and determine the final values of variables, the rule base needs a function that combines the outputs of each rule into a single fuzzy value. Such a function is called an implication function. A simple and often used implication function is called 'sum' implication. It simply sums the outputs of each rule (pertaining to the same variable) to obtain its final value. Example 3 shows the operation of a rule base with three rules.

This bring us to a drawback of fuzzy rule bases. Whereas the rule base

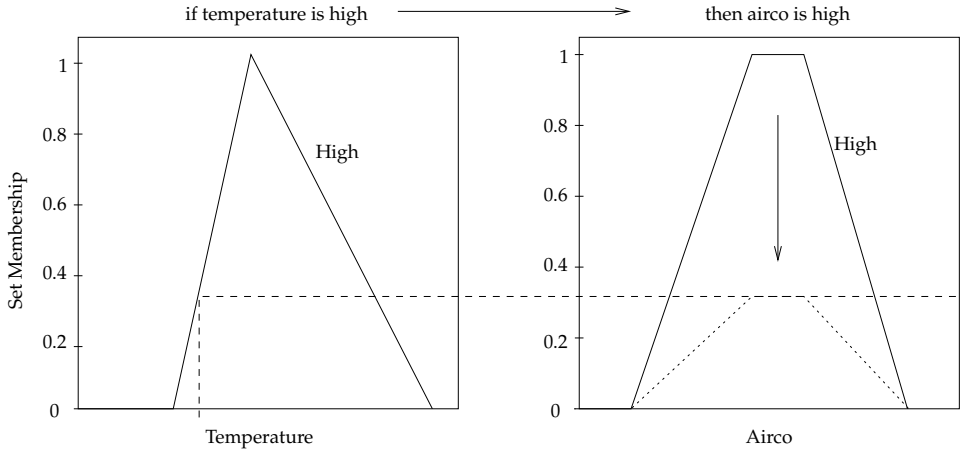


Figure 3.3: An example rule in an air conditioning system (Example 2).

internally uses fuzzy sets to determine output values, it is not capable of expressing these in terms of these fuzzy sets. For example, it is not trivial to express the output of the rule base in Figure 3 in terms of fuzzy sets D, E, and F. Instead, a fuzzy rule base converts the obtained fuzzy value back to a crisp value (a scalar). As mentioned already, this process is called defuzzification. These crisp values are the actual outputs of the rule base.

Example 3. *Figure 3.4 shows an example rule base with three rules. All three rules reference variable z in their consequent. To unify the outputs of all three rules, the rule base uses sum implication. The bottom part of Figure 3.4 shows the addition of the three fuzzy values.*

3.4.2 Learning Fuzzy Rules

In the previous section we described fuzzy logic in general. Now we will turn to identifying patterns in continuous data, and representing these patterns with fuzzy rules. This is necessary, because we wish to model an agent's behavior with fuzzy rules to create an opponent model. Fortunately, several algorithms exist that are capable of identifying these patterns.

To identify a suitable algorithm for our purposes, we selected and studied a number of fuzzy rule learners based on expert recommendation.[3] Herrera et al. (1998), Pal et al. (2003), and Cordon et al. (2001) propose algorithms for learning fuzzy rules from continuous data based on genetic algorithms. Nozaki et al. (1997) use a slightly different notion of a fuzzy rule base than discussed in Section 3.4.1.2. In this rule base, the output of a single rule is a numerical value, instead of a fuzzy one (see Figure 3.4).

[3] R. Babuška 2005, personal communication.

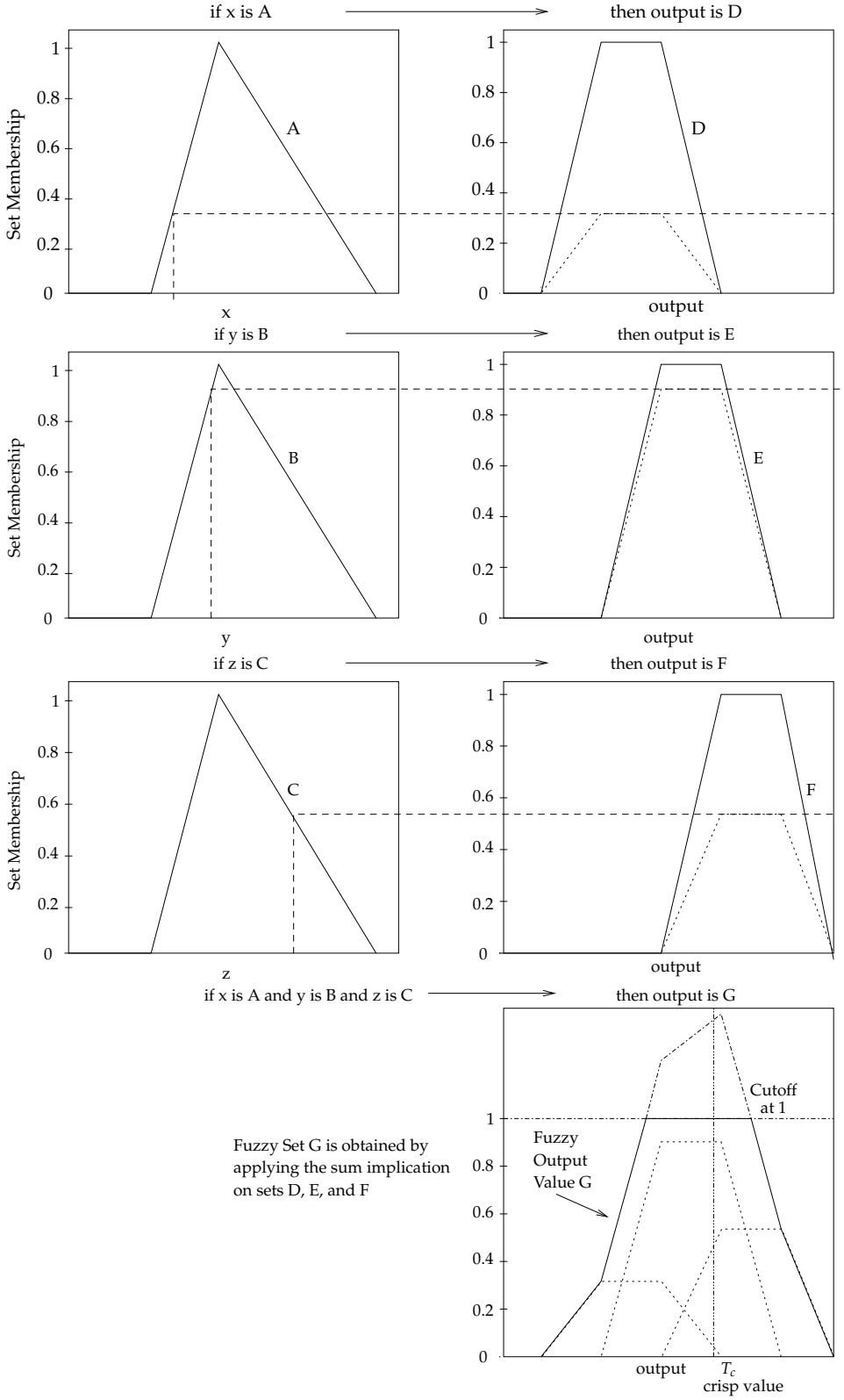


Figure 3.4: A rule base illustrating how three the outputs of three rules that reference the same variable in their consequent are combined (Example 3).

Based on this, they present a simple heuristic for learning fuzzy rules. Finally, Hong and Lee (1998) propose a rule learner based on the principle of decision tables.

In our approach, we chose for an algorithm called Fuzzy Rule Learner (or FURL for short) proposed by Rozich et al. (2002). FURL is a supervised, batch learning algorithm that uses theory revision techniques to induce fuzzy rules from data sets. 'Supervised' means that the algorithm must be supplied with input-vectors as well as the desired output. The 'batch' property means that FURL must recalculate the entire rule base if new instances are received. It cannot refine the existing rule base.

The reason for choosing FURL is threefold (in decreasing order of importance):

1. Unlike the genetic algorithms mentioned above, it is simple to understand: it iterates through the three basic steps of theory revision (error detection, credit assignment, and repair) until the developed rule base is good enough.

2. It uses the notion of exceptions to build a rule base, which is a attractive feature in trust. We come back to this in Section 3.4.4.

3. The algorithm is specified in detail in Rozich et al. (2002), making it relatively easy to implement.

In what follows, we explain this algorithm in more detail. First, however, we study the data structure on which it operates.

3.4.2.1 Hierarchical Prioritized Structure

FURL operates on a data structure called Hierarchical Prioritized Structure (HPS). This data structure was proposed by Yager (1993), as an answer to the problem of interacting rules with overlapping coverage. In particular, an HPS uses the concept of *exceptions* to deal with this problem.

To this end, an HPS consists of multiple fuzzy rule bases that are organized in layers (see Figure 3.5). Each of these (ordinary) fuzzy rule bases represent a different kind of knowledge in the HPS. In the bottom layer(s), rules representing general knowledge are stored. These rules fire for large classes of inputs. In higher layers, exceptions to lower rules are stored. These exceptions only fire for a smaller number of inputs. More specifically, if a rule fires on some class of inputs A, an exception will only fire on a subclass of A. The higher the level in which a rule is present, the higher its specificity.

Example 4 (Exception)**.** *For example, an exception to rule R*
'if x is A then z is C'

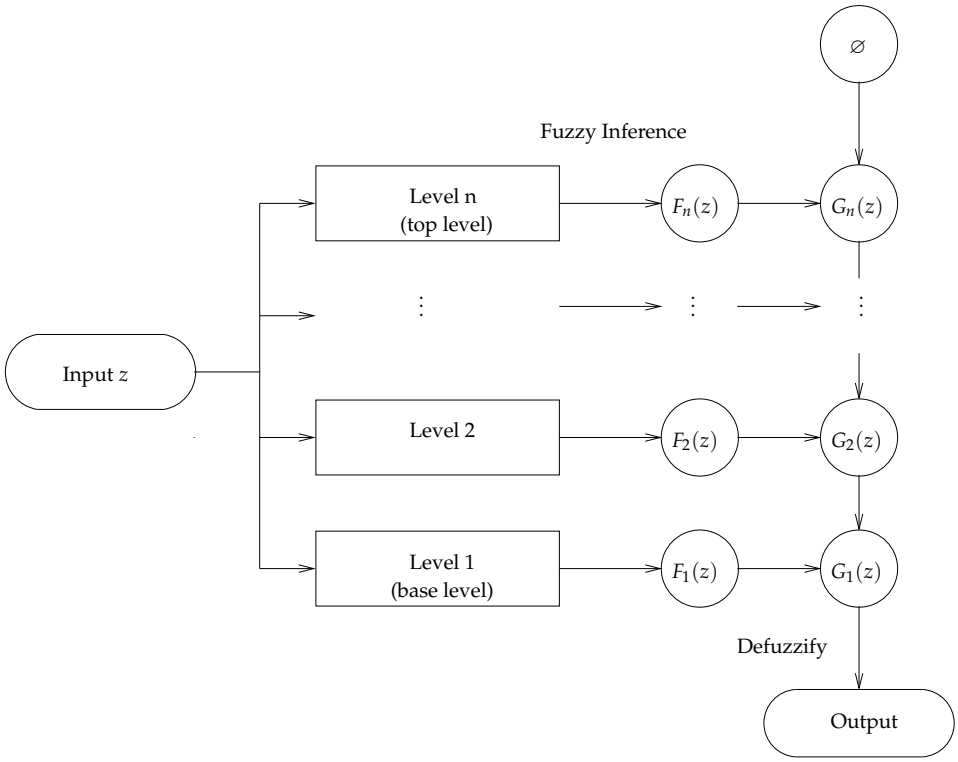


Figure 3.5: Inference in an HPS Rule Base. Adapted from Rozich et al. (2002).

is rule R′:

'if x is A and y is B then z is D'.

An HPS has a built-in mechanism to deal with overlapping rules. Rules are said to *overlap* if multiple rules from different layers fire on the same input. This of course happens if an input is subject to an exception. The HPS employs the so called Hierarchical Updation (HU) formula to calculate its output based on the input-vector z and the rules in the different rule bases. This process is illustrated in Figure 3.5. The Hierarchical Updation formula itself is defined as follows:

$$\begin{aligned} G_j(z) &= G_{j+1}(z) + (1 - \alpha_{j+1}) \times F_j(z) \quad 1 \leq j \leq n \\ G_{n+1}(z) &= \emptyset \end{aligned} \tag{3.2}$$

where n is the number of layers in the HPS, F_j denotes the output of the rule base in level j, and α_j is defined as:

$$\alpha_j = \max_z \left(G_j(z) \right) \tag{3.3}$$

which is the *maximal membership grade* in and above level j. Note that the '$+$' operator in Equation 3.2 is the *fuzzy addition* operator. The result of a

fuzzy addition of two fuzzy values is also a fuzzy value. The $\times$ operator is used to scale a fuzzy value with a scalar.

From the HU formula, we observe that the degree to which the output of a certain layer influences the output of the HPS, depends on the maximum match strength in higher layers.[4] So, if at any point an input matches an exception for 100%, rules in lower layers do not contribute in any way to the output of the HPS.

Indeed, this is the expected and desired behavior of an exception: if an exception applies in a certain situation, it overrides its parent rule. This way, the HU formula enforces the priority of the rules in the HPS.

The operation of the HPS and the HU formula is illustrated in Example 5.

Example 5. *In this example, we consider a HPS consisting of four layers. The separate (fuzzy) outputs of each of these layers are pictured in Figure 3.6. From these graphs, we can see that the values of α are as follows:*

j	$\max_z \left(F_j(z)\right)$	α_j	$1 - \alpha_j$
4	*0.2*	*0.2*	*0.8*
3	*0.6*	*0.6*	*0.4*
2	*0.4*	*0.6*	*0.4*
1	*0.9*	*0.9*	*0.1*

Using these values of α_j, we first scale and then fuzzy-add the outputs of each layer to finally end up with G_1. This final output of our example HPS is depicted in Figure 3.7.

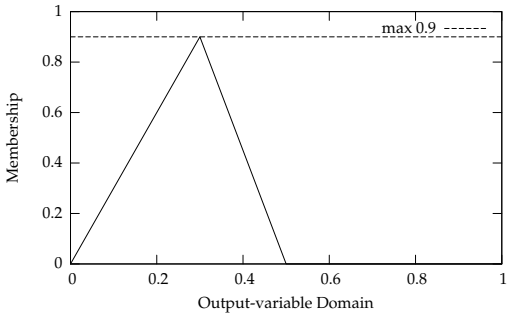
3.4.2.2 The FURL Algorithm

As said before, Rozich et al. (2002) have based FURL on techniques in the field of fuzzy logic and theory revision. Theory revision applied on a fuzzy rule base consists of the repeated execution of three steps (Ginsberg, 1989):

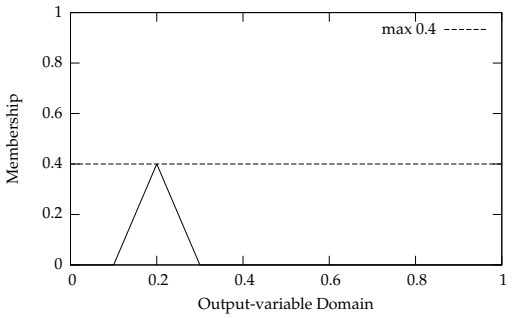
1. Error detection. This step focuses on detecting that the rule base contains an error and that refinement is necessary.

2. Credit assignment. In this step, individual rules within the rule base are assigned *credit* based on their relative responsibility for the errors detected in the previous step. The more credit a rule is assigned, the more responsible it is for the error in the rule base.

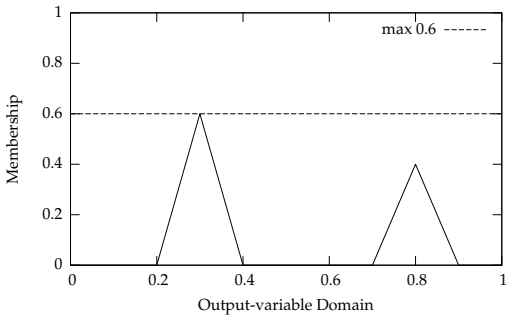
[4] Remember from Section 3.4.1.2 that the match strength is defined as the minimum truth value of the antecedents of a rule for a specific input sample.



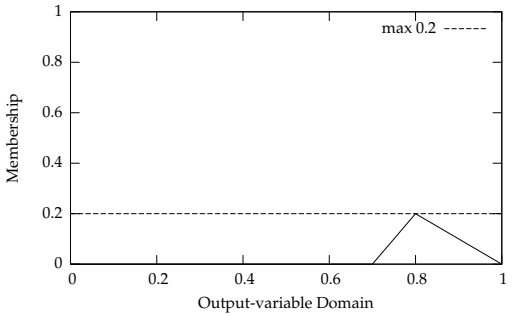
(a) F_1



(b) F_2



(c) F_3



(d) F_4

Figure 3.6: Fuzzy outputs for all four levels of the HPS in Example 5

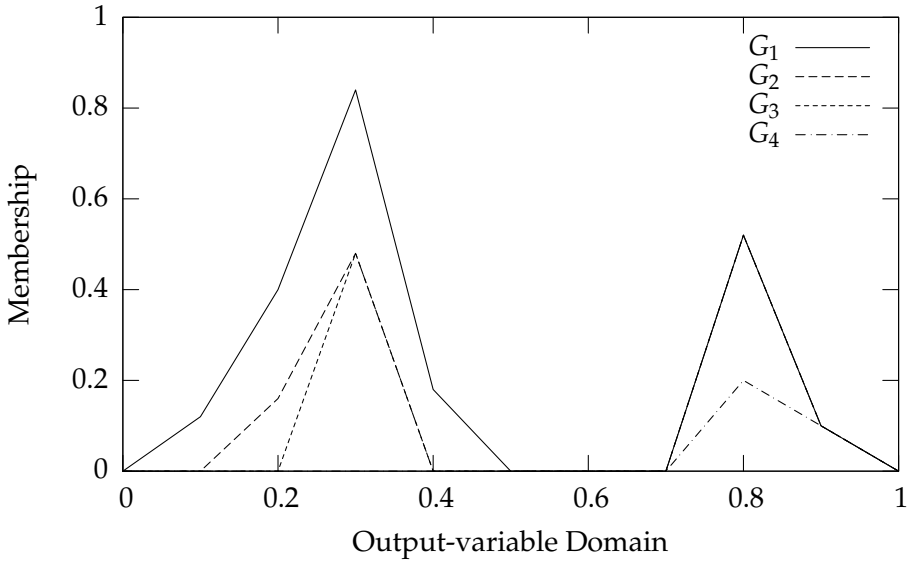


Figure 3.7: Combined fuzzy output of the HPS for Example 5

3. Repair. In this step, the N rules with the highest assigned credit are selected to be eligible for repair. For each of these N rules, refinements are generated. The refinement that reduces the error the most is added to the rule base.

Using the decomposition of theory revision in these three steps, we can describe the operation of FURL. A high-level description of the algorithm is presented in Algorithm 1. The first two steps (partitioning the input space and creating an initial rule base) are used to initialize the learning algorithm. The algorithm subsequently iterates through the three basic steps of theory revision, until a rule base is identified that sufficiently describes the set of examples the algorithm tried to learn. We will now take a closer look at each of these steps.

Algorithm 1 The Fuzzy Rule Learner Algorithm

```
partition input space of data set
HPS ← create initial rule base
repeat
   detect error
   assign credit to each rule in the HPS
   repair rule base by refining top N rules with highest credit
until stopping criterion is met
return HPS
```

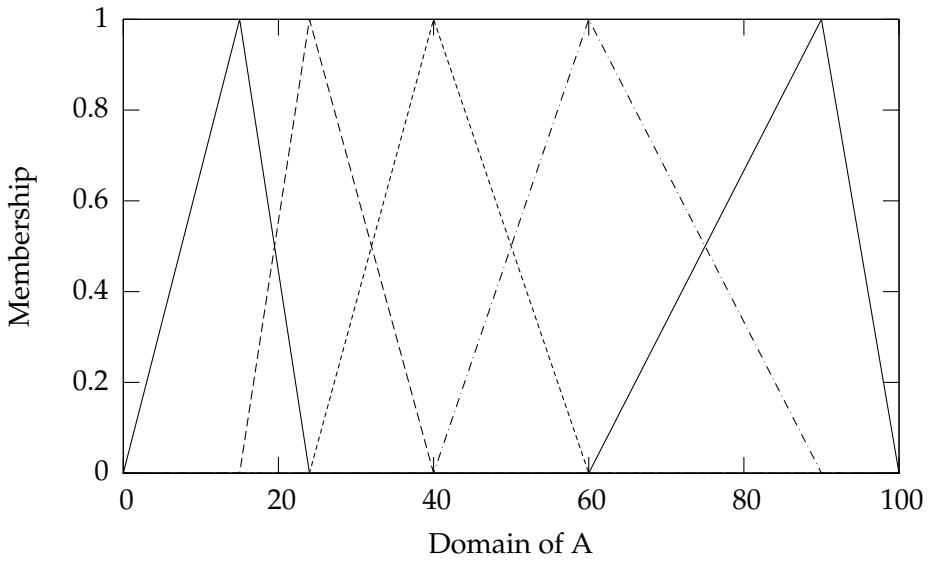


Figure 3.8: Sample partitioning of the input space of Variable A

Partitioning the Input Space This step of the algorithm focuses on defining fuzzy sets for every variable in the data set. The partitioning method used by FURL aims to identify sets, such that each set contains about an equal amount of data points from the examples.

For crisp sets, obtaining the cardinality simply reduces to counting the number of members of the sets. For fuzzy sets, however, the cardinality is obtained by summing the membership degrees of all the elements in the data set S:

$$card(M) = \sum_{i \in S} \mu_M(i) \tag{3.4}$$

where $\mu_M(i)$ is the membership of instance i in set M.

FURL uses triangular fuzzy sets, modeled as a piecewise linear function with a start point, middle point and an end point, such that each of these points is shared by three different sets. For example, a start point of set A, is the middle point for set B, and the end point of set C. An example partitioning is illustrated in Figure 3.8.

These sets have two properties, that are important for later steps of the algorithm:

- Each set only overlaps with its two neighbors.
- For each instance of the data set its memberships in the relevant sets sums exactly to 1.

Based on these two properties, the method for obtaining the set cover for each variable can be reduced to calculating the percentiles of the data set. So, for a partitioning of k data sets, the start, middle, and end points for the sets are the $\frac{1}{k}, \frac{2}{k}, \ldots, \frac{k}{k}$ percentile rank positions from the set of data points. While this does not ensure that the sets have exact equal cardinalities, the method produces an acceptable partitioning of the input space.

Along with the partitioning method based on percentiles, we also tried other partitioning methods, because the way the input space is partitioned greatly influences the end result.

Therefore, we selected another partitioning method that is more focused on identifying clusters of data points in the input space, instead of just ensuring that the cardinality of each set is more or less similar.

To this end, we investigated clustering algorithms included in the Weka Machine Learning Project.[5] Specifically, we studied both the K-Means and Fuzzy K-Means algorithms. The former calculates only the mean of each data cluster, whereas the latter identifies complete fuzzy sets.

We implemented both algorithms, and eventually chose the K-Means clustering method as the primary alternative to the percentile partitioning method, because it is very simple, and the difference with the partitioning produced by Fuzzy K-Means is minimal.

The K-Means algorithm partitions N data points in K disjoint clusters S_j in an aim to minimize the following sum of squares criterion:

$$J = \sum_{j=1}^{K} \sum_{n \in S_j} (x_n - \mu_j)^2 \tag{3.5}$$

where x_n is the nth data point assigned to cluster S_j, μ_j is the mean of data points in cluster S_j. After initialization, in which the centers of the clusters are randomly chosen,[6] the algorithm iterates through two steps. In the first step, data points are assigned to the nearest clusters. In the second step, the cluster centers are recalculated by taking the mean of the data points in the cluster. This process repeats until no data point is assigned to another cluster.

In Section 3.4.3, we show the performance of the FURL algorithm with both partitioning methods: the percentile partitioning method included in the specification of FURL, and the K-Means clustering algorithm that aims to identify clusters of data points.

[5] See `http://www.cs.waikato.ac.nz/~ml/` and `http://sourceforge.net/projects/weka/`

[6] Bottou and Bengio (1995) show that K-Means 'almost surely converge to a local minimum' of Equation 3.5, irrespective of the exact initialization of the clusters. In practice, K-Means is almost always initialized with random clusters. In our application, random initialization resulted in good partitionings of the input spaces.

Create Initial Rule Base Having partitioned the input space of each variable of the data set, the algorithm continues with building an initial rule base. In this step, FURL constructs a flat rule base with a minimal global error. The rationale behind choosing a low error, is that such a rule base is likely to contain rules that describe the most general properties of the learned data set, and is therefore likely to generalize well. The rule base that is obtained from this step is used as the base level of the HPS data structure (see Figure 3.5).

The algorithm used in this step is outlined in Algorithm 2. Simply put, the algorithm tries to construct rules with a single antecedent and consequent for each input variable A. Next, the rule base with the least global error is selected and returned.

Algorithm 2 Create Initial Rule Base

for all input-variable A_i **do**
  construct a set RB_{A_i} of k rules of the form
  if A_i is A_i^1 then C is C^{*1}
  $\vdots$
  if A_i is A_i^k then C is C^{*k}
  where A_i^k is a fuzzy set defined on the domain of variable A_i, and C^{*k} the *most suitable* fuzzy set defined on the domain of consequent variable C.
end for
return $RB_{A_o} = \arg\min_i (error(RB_{A_i}))$

Algorithm 2 seeks to find the *most suitable* fuzzy set for each fuzzy set defined on the domain of A_i. FURL uses Equation 3.6 to identify this set. It tries to maximize the sum of the product of the membership of the antecedent and the consequent, for each instance in the data set. It is easy to see that the higher this product, the more likely the antecedent and the consequent are correlated.

$$C^{*x} = \arg\max_{C^j} \left(\sum_{i \in S} \mu_{C^j}(i) \times \mu_{A^x}(i) \right) \tag{3.6}$$

Error Detection In theory revision, a fitness measure is a measure of how well the learned rule base describes the elements of the data set. It is obtained by comparing the output (or prediction) of the rule base with the actual data for each example in the data set. In FURL this is done by taking the mean absolute error, as follows:

$$\delta_{abs} = \frac{\sum_{i \in S} |t_i - o_i|}{|S|} \tag{3.7}$$

where t_i is the actual value in the data set, and o_i the output of the rule base for instance i.

Credit Assignment Credit assignment is the process of identifying the parts of the rule base that are responsible for the suboptimal predictive capability of the rule base. In FURL, rules are assigned a higher credit, if they are believed to be more responsible for this error. This credit is used in the final step of the iteration, in which rules with high credit are repaired.

To assign credit, FURL uses the notion of error contribution. In other words, it tries to identify which rules are responsible for a discrepancy of the output of the learned rule base, and the actual output in the training samples. In short, this process consists of four steps:

1. Determine to what extent γ_j a level j of the HPS is responsible for the output of the entire rule base:

$$\gamma_j = \sum_{i \in S} \alpha_j(i) - \alpha_{j-1}(i) \tag{3.8}$$

   where variable $\alpha_j(i)$ denotes the maximal membership grade in and above level j (see Equation 3.3), and S is the data set used for training.

2. Determine the relative error δ_j a level j is responsible for:

$$\delta_j = \gamma_j \times \sum_{i \in S} |t_i - o_i| \tag{3.9}$$

   where t_i is the actual value in the data set and o_i is the output of the rule base for entry i.

3. Determine the relative contribution rc_R of each rule R in $\mathbf{R_j}$ (the set of rules in level j):

$$rc_R = \sum_{i \in S} \frac{\mu_R(i)}{\sum_{R \in \mathbf{R_j}} \mu_R(i)} \tag{3.10}$$

   where $\mu_R(i)$ is the match strength of rule R on instance i.

4. Finally, calculate credit C_R for each rule R, by determining the overall error responsibility for rule R. This is obtained by combining the results of steps 2 and 3:

$$C_R = \delta_j \times rc_R \tag{3.11}$$

The rationale behind this method, is that when a rule R has a large influence on the output of the entire system for a certain set of inputs, and the output of the system contains an error for this set, R is believed to have an exception, and should therefore be assigned a large credit.

Rule Base Repair This step of the FURL algorithm uses the result of the credit assignment to select rules eligible for repair. After selecting these rules, it generates all possible modifications for these rules, and identifies the best one in terms of error. The selected modification is then added to the HPS at one level higher than the original rule. A modification to a rule is very similar to the concept of an exception.

More formally, an exception R' of rule R is obtained by adding an antecedent, and modifying the right hand-side of the consequent (see Example 4 on page 32).

In Algorithm 3 the procedure is outlined in pseudo-code. Note that, by selecting the top N responsible rules, the algorithm uses a beam search to find an acceptable refinement to the learned rule base. A *beam search* is a search algorithm that uses a heuristic function to identify the N most promising choices. The parameter N is called the *beam width*. In FURL, the credits function as a heuristic. Rozich et al. (2002) chose to implement FURL as a beam search algorithm, in order to limit the number of rules for which refinements must be evaluated.

Algorithm 3 The repair algorithm

for all rule R in the top N responsible rules **do**
  for all refinement j on R to produce exception R_j **do**
    add R_j to the rule base at one level higher than R
    $\delta_{abs,j} \leftarrow$ recompute error for this new rule base
  end for
end for
select modification k such that $R_k = \arg\max_k (\delta_{abs} - \delta_{abs,k})$
add R_k to the rule base

Stopping Criterion The last three steps we discussed in the previous sections (i.e. error detection, credit assignment, and rule base repair) are repeated until the stopping criterion is met (see Algorithm 1). In general,

a theory revision technique is applied, until the fitness measure of the learned theory has reached a certain value. In FURL, this fitness measure is the error obtained from Equation 3.7.

In theory, we could iterate through the algorithm, until we obtain the error does not decrease anymore. Paradoxically, however, such a small error would not be desirable, because of the problem of overfit. Whereas an overfitted rule base is capable of accurately describing a training set, it would be less suitable to predict outputs for examples that were not used during training (see Figure 3.9). To prevent this from happening, FURL continues to refine the learned rule base, until the change in error of two subsequent iteration is less than a predetermined percentage.

Admittedly, this is a quite rudimentary approach. Viable alternatives exist for preventing overfit. For example, the algorithm can be extended with a policy that puts a penalty on adding a new rule, thereby attempting to limit complexity. Unnecessary complexity is often the cause of overfit (cf. Occam's razor). If the benefits of adding a new rule (in terms of error) do not weigh up to the incurred penalty, the algorithm terminates.

Another often used method is cross-validation.[7] In cross-validation, a part of the data set is set aside for validating the quality of the learned model. This set, called the validation set, is not used while learning the model. The learning algorithm is stopped as soon as the error for the validation set starts to increase (as in Figure 3.9).

In the next section, we will support our choice for the stopping criterion in the original specification of FURL with a small experiment. In this experiment, we show that overfit does not occur if the number of partitions on the input space of the variables is chosen correctly. If this number is chosen correctly, using this simple stopping criterion gives adequate results.

3.4.3 A Small Experiment

To test our implementation of the FURL algorithm, and to calibrate the implementation by identifying suitable parameters, we ran FURL on a sample data set. This data set was also used by the creators of FURL to measure the algorithm's performance. The data set is called 'Auto-MPG' and contains data about cars and their fuel consumption. It is part of the Statlib library that is maintained at the Carnegie Mellon university.[8] The data set was originally used for the 1983 American Statistical Association Exposition.

In this experiment, FURL is used to predict the fuel consumption, given seven attributes. Among these are the car's origin, its weight, and its acceleration.

We tested several configurations to find out what choice of parameters

[7] For example, see `http://en.wikipedia.org/wiki/Cross-validation`

[8] See `http://lib.stat.cmu.edu/`

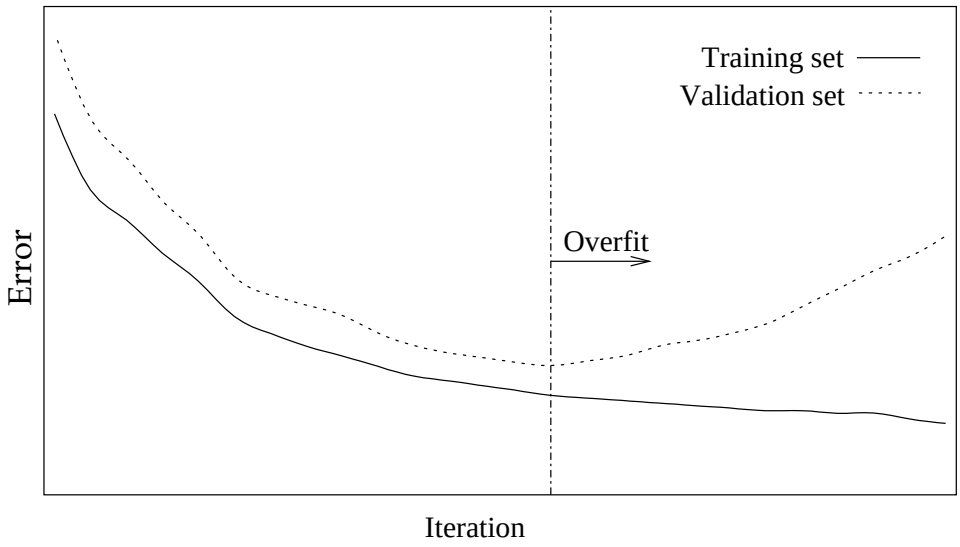


Figure 3.9: Overfitting causes the error for the training set to drop, but increases the error for inputs not used during training (the validation set)

results in best performance in terms of global error of the validation set. We were specifically interested in the influence of the choice of the beam width N (3, 4 or 5), the number of partitions used to partition the input-space, and the method used to calculate the partitions (i.e. percentiles or K-Means). For this experiment, we used a fixed stopping criterion of 25 iterations, in order to determine when overfit occurs.

Figure 3.10 shows FURL's performance on the training set. We validated the quality of the learned rule base using the validation set to see if and when overfit occurs. After each iteration of FURL (i.e. the addition of a new rule), the error on the validation set is determined. The results of this are displayed in Figure 3.11.[9]

In the results of this experiment, we can see the following patterns emerging.

1. A FURL configuration with a small number of partitions (3) has a high error on the training set, and causes slower convergence on the validation set.

2. A FURL configuration with a large number of partitions (8 or 10) reduces the error on the training set, but causes slower convergence on the validation set.

[9]The beam width parameter did not have any influence on the intermediate results (i.e. the result obtained after each iteration) nor on the final result. These results are therefore omitted the these plots.

3. The choice of partitioning method does not significantly influence the results, except for the following two points.

4. Configurations using K-Means perform more consistently than configurations using percentile partitioning. In particular, the percentile configuration with 10 partitions converges to a higher level of error than the other configurations, and the configuration with 3 partitions has a quite erratic error.

5. The final error on the validation set is more or less the same for all configurations (except for 3 partitions and 10 partitions with Percentile partitioning).

The first two results are more or less expected. A configuration with very few partitions causes very coarse rules to be created. These rules fire on relatively large sets of inputs. Consequently, it is likely that these individual inputs in these sets are unrelated, but are still treated alike. This results in a relatively high error on both the training set and the validation set.

On the other hand, the more partitions, the smaller the amount of training-instances that match on a particular rule. Therefore, it becomes possible to create rules with a high specificity, resulting in a lower training-error. However, the probability that these rules do not match on data set instances that were not used during training also increases, causing a poorer prediction for these instances. Fortunately, FURL's computational overhead decreases as the amount of partitions decreases, because of a smaller number of possible exceptions that have to be evaluated.

Based on these results (specifically the third), we are somewhat indifferent of the specific partitioning method used. In our experiments, we decided to use K-Means, mostly because this method produces results of consistent quality. Moreover, this experiment shows that the number of partitions should be chosen with care.

What overfit is concerned, Figure 3.11 shows that an increasing number of iterations (or rules added to the rule base), does not have a significant negative result on the error for the validation set. Based on this observation, we not think that a more sophisticated stopping criterion is necessary. We therefore prefer the simplicity of the stopping criterion in the original specification of FURL (i.e. stop after the error decreases with less than n%).

3.4.4 Discussion

This concludes our coverage of the FURL algorithm. We have seen that FURL is a theory revision algorithm. It iterates through three steps—error detection, credit assignment, and repair—until a sufficiently accurate rule base is constructed. In this section, each of these three steps have been

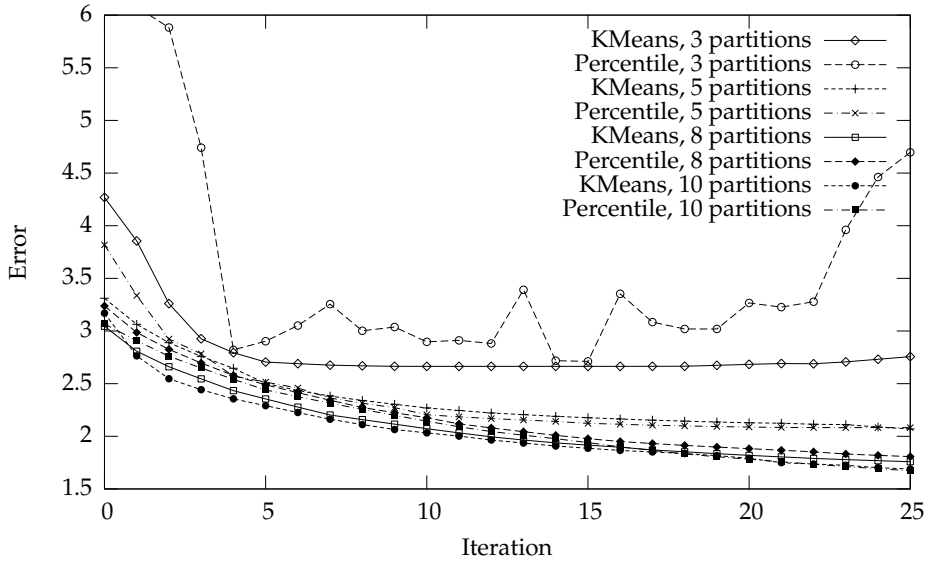


Figure 3.10: Errors for training set during FURL execution on the Auto-MPG data set

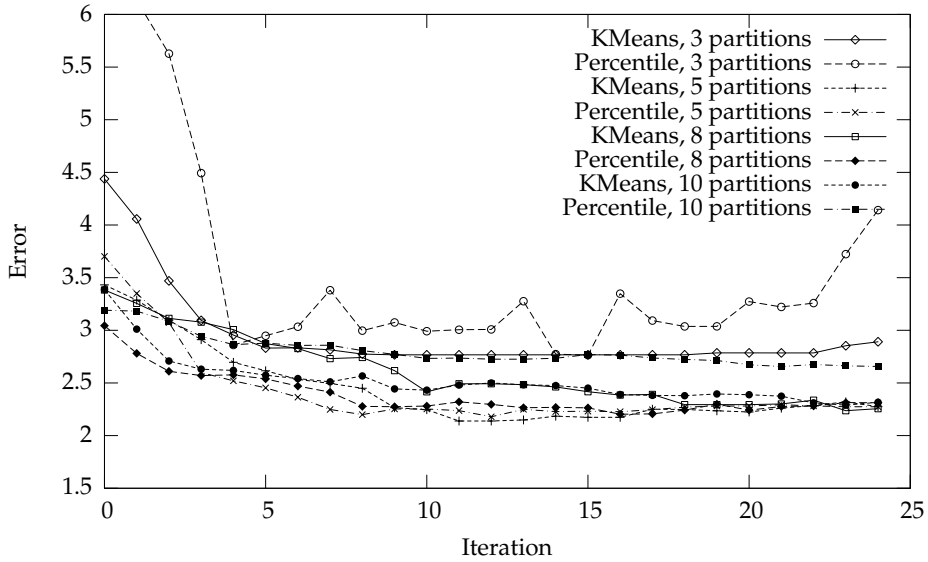


Figure 3.11: Errors for the validation set during FURL execution on the Auto-MPG data set

discussed in detail. At this point, we would like to discuss the applicability and suitability of FURL in our approach.

As we have seen, FURL uses a data structure that is comprised of several fuzzy rule bases in a hierarchy (the HPS). This structure very naturally supports the notion of exception. An exception to a rule is more specific, and demarcates a subclass of objects or situations for which the original rule applied. In our context, behavior of an agent is often expressible in terms of exceptions. For example, the behavior of the RECIPROCAL agent we encounter in Section 4.3.3.2, can be described as: 'honest, except when treated dishonestly'. FURL's exceptions could therefore be beneficial for human comprehensibility.

On the down side, however, it also needs several rules to describe this behavior, instead of a single sentence. For each partition on the input space of the variable in the base level, a rule needs to be created. For each of these rules, multiple exceptions could be detected. Of course, too many rules can clutter up the rule base, making it harder to understand.

To find out which of our two lines of reasoning would prevail (i.e. exceptions facilitate comprehensibility, while a large set of rules complicates matters), we requested expert input. Colleague Peter Kok was kind enough to act as an expert. He studied the application of neural networks on data sets obtained from diabetes patients, in order to predict the effect of medication, food and rest on glucose level at the end of certain time intervals. We used FURL on this data set, and asked Peter if the resulting rule base was both understandable and correct.

In his opinion, the rule base was indeed easy to understand, and described the most obvious patterns in the data set. However, unlike the neural networks he used in his research, FURL did not detect the finer patterns. On the other hand, FURL is capable of explicitly describe knowledge, which a neural network is incapable of. In light of our aim to create a proof-of-concept, high levels of accuracy are desirable, but not necessary. In this proof-of-concept, we much more desire comprehensibility. We believe that FURL is accurate enough to detect and describe simple behavior, and is therefore good enough for our experiments. In Chapter 4, in which we discuss our empirical results, we support this claim. For future work, however, we think that FURL should be further improved. In Section 5.2.2, we give some pointers on how this could be done.

3.5 Decision Making

In trust, the decision making component must be able to translate the computed opponent model into actual decisions. To do this, it uses the opponent model to find out the hypothetical consequences of a decision. In a way, the opponent model functions as a safe environment in which the

agent can try out different decisions and determine their effect.

In our approach, we propose to use argumentation for decision making. The rationale behind this choice, is that argumentation is a very natural tool for providing explanations, and convincing the user of the fact that the recommended decision is in fact justifiable and prudent.

3.5.1 Argumentation

As said, argumentation is a means to convince someone of the truth or falsity of a certain statement. Applied to decision making, it enables us to carefully weigh the pros and cons of each decision under consideration, and to select the decision that is most likely to have acceptable consequences.

In order to select a suitable argumentation framework for our approach, we studied a selection of unpublished work on decision making under uncertainty.[10] In a recently published paper by Amgoud and Prade (2004) that is based on this unpublished work, a fully worked-out argumentation framework is presented, capable of making decisions using uncertain knowledge from fuzzy rule bases. Based expert recommendation,[11] and the fact that it can be appropriately combined with our opponent modeling algorithm, we decided to select the framework proposed by Amgoud and Prade (2004).

In general, decision making consists of four distinct steps: option generation, drawing up selection criteria, weighing the criteria, and finally selecting an option. The main benefit of subdividing the decision making process in these phases, instead of regarding it as a single monolithic step, is that it enables us to reconstruct the process that brought about the decision. This allows an outsider insight into the reasons behind a decision (the rationale), which is exactly what we aim for in our approach.

The argumentation framework by Amgoud and Prade (2004) uses a similar decomposition of decision making in phases:

- argument generation (both pro and con),
- selecting acceptable arguments, and
- concluding.

The first phase uses the agent's knowledge and preferences to create arguments based on the *predicted* outcome of a decision. If a predicted outcome is consistent with the agent's goals, an argument can be constructed

[10] H. Prakken 2005, personal communication.
[11] *Ibid.*

that supports that particular decision. If the predicted outcome is inconsistent with the agent's goals, an argument is constructed that rejects the decision.

The second phase evaluates each available argument and decides whether or not the argument is acceptable in terms of the knowledge used, and the desirability of the arguments used.

In the third and final phase, the acceptable arguments are processed by an evaluation criterion, to obtain the final decision.

In what follows, we shall explain the theory the argumentation framework of Amgoud and Prade (2004) is based on, and study the framework itself.

3.5.1.1 Decision making under uncertainty, a possibilistic logic approach

The framework by Amgoud and Prade (2004) is based on the work of Dubois et al. (1994), in which an approach of decision making under uncertainty is explained. This approach uses possibilistic logic (another name for fuzzy logic) to model the agent's knowledge and its preferences. To this end, two distinct possibilistic knowledge bases $\mathcal{K}$ and $\mathcal{G}$ are used.

The first base $\mathcal{K}$ (containing the agent's knowledge) consists of tuples (k_j, ρ_j), where k_j denotes a proposition from the propositional language $\mathcal{L}$, and ρ_j is the necessity of k_j. The value of ρ_j is an element of the linearly ordered scale R. The top and bottom elements of R are denoted by 1 and 0 respectively.

The tuple (k_j, ρ_j) therefore states that $N(k_j) \geq \rho_j$, where N is a necessity measure. Put differently, (k_j, ρ_j) can be paraphrased as: "statement 'k_j is true' is certain with at least degree ρ_j".

Decisions are modeled as a conjunction of decision literals. These literals model the state of each intervention tool the decision maker has at its disposal.

Let $\mathcal{D}$ denote the set of all possible decisions, and let the base $\mathcal{K}_d = \mathcal{K} \cup \{(d, 1)\}$ describe the world after decision d is taken. The formula $(d, 1)$ asserts that d is true with certainty 1, which is exactly the case after decision d is taken.

However, after taking a decision (or doing an *action*), there exists uncertainty about the new state of the world (or the *outcomes*). In the framework, this uncertainty is captured in the possibilistic distribution $\pi_{\mathcal{K}_d}$, which orders the possible states ω of the world according to their relative possibility:

$$\pi_{\mathcal{K}_d}(\omega) = \min_j \max\left(v_\omega(k_j), n_R(\rho_j)\right) \tag{3.12}$$

where $v_\omega(k_j) = 1$ if ω is a model of k_j, and $v_\omega(k_j) = 0$ if ω falsifies k_j, and n_R is an order-reversing map of R.

Equation 3.12 states that world-state ω is less plausible, if $v_\omega(k_j) = 0$ (i.e. k_j is false), and ρ_j is high. This is consistent with our intuition of the notion of possibility: state ω is considered less plausible if we *expect* a certain statement k_j to be true (because of its high possibility), and yet k_j is falsified in world-state ω (i.e. $v_\omega(k_j) = 0$).

The second possibilistic logic base models the preferences of the decision maker. This base is denoted by $\mathcal{G}$, and consists of tuples (g_i, λ_i), where g_i is a proposition of language $\mathcal{L}$, and λ_i represents the extent to which the decision maker prefers goal g_i. These preferences λ_i take their value from a linearly ordered scale T, which has a top and bottom element, denoted by $\overline{1}$, and $\overline{0}$ respectively.

Using the goal set $\mathcal{G}$, Dubois et al. (1994) formulate an ordinal utility function:[12]

$$\mu_{\mathcal{G}}(\omega) = \min_i \max\left(v_\omega(g_i), n_T(\lambda_i)\right) \tag{3.13}$$

where n_T is the order reversing map of scale T. As can be seen from Equation 3.13, the *acceptability* of a state ω is low when a goal g_i with a high preference λ_i is violated. Put differently, $\mu_{\mathcal{G}}$ is inversely proportional to the preference of the most important goal that is violated.

If we combine $\mu_{\mathcal{G}}$ from Equation 3.13 and $\pi_{\mathcal{K}_d}$ from Equation 3.12, we obtain a measure of satisfaction that evaluates the consequences of a decision in terms of $\mathcal{G}$. One such measure is defined by Dubois et al. (1994):

$$E_*(d) = \min_{\omega \in \Omega} \max\left(\mu_{\mathcal{G}}(\omega), n(\pi_d(\omega))\right) \tag{3.14}$$

where

- d is the decision under consideration
- Ω is the (finite) set of possible consequences of decision d
- $\mu_{\mathcal{G}}$ is the utility function that assigns values from the preference scale T to the elements of Ω.
- π_d is a normalized possibility distribution from Ω to a linearly ordered plausibility scale R.[13]
- n is an order-reversing map from R to T.

The result of utility function $E_*(d)$ is small only if there exists a outcome ω of decision d that is both highly plausible and highly undesirable. This

[12] In ordinal utility theory, an agent is considered only capable of *ranking* the available alternatives. It is incapable of *measuring* utility, as in cardinal utility theory.

[13] i.e. $\exists\omega : \ \pi_d(\omega) = 1$, where 1 is the top element of R

corresponds with a risk-averse and pessimistic attitude, because it emphasizes only the negative outcomes of a decision, instead of taking possible positive outcomes into account. In this respect, it can be compared with a mini-max strategy in game-theory.

Note that a commensurability assumption is made between plausibility and preference, through the order-reversing map from R to T. More specifically, $R = T$ and $n_R = n_T = n$. This means that it is assumed that plausibility and preference are measurable using a common standard.

To maximize E_*, the following proposition is postulated:

Proposition 1. *$E_*(d)$ equals the maximum value of α, such that:*

$$(\mathcal{K}_d)_\alpha \vdash (\mathcal{G})_{\underline{n(\alpha)}} \tag{3.15}$$

where $(B)_\alpha$ is the set of propositions in a possibilistic logic base B with a level greater than or equal to α. The set $(B)_{\underline{\alpha}} \subseteq (B)_\alpha$ contains only propositions with a level strictly greater than α.

For example, $E_*(d)$ is equal to 1, if d is based only on the most certain part of $\mathcal{K}$ (i.e. $\mathcal{K}_1$), and it satisfies all goals (even those with low preference) in the set $\mathcal{G}$ (i.e. $\mathcal{G}_{\underline{n(1)}} = \mathcal{G}_{\overline{0}}$).

Dubois et al. (1994) propose a logical machinery for computing decisions using Proposition 1.[14] Amgoud and Prade (2004), however, deviate slightly from this algorithm, and propose a solution based on argumentation.

3.5.1.2 Argumentation-based decision making

According to Proposition 1, in the pessimistic view one aims to find a decision d such that $\mathcal{K}_\alpha \wedge d \vdash \mathcal{G}_\beta$, with α high and β low. In other words, d is preferably a decision that, based on the most certain part of $\mathcal{K}$ brings about even low priority goals in $\mathcal{G}$.

Step 1: Argument generation In this perspective, an argument *in favor* of a decision predicts the consequences of a decision, using available knowledge about the world, and evaluates these consequences using the preferences of the decision maker.

Definition 3 (Argument PRO)**.** *An argument in favor of a decision d is a triple $A = \langle S, C, d \rangle$, where:*

[14] In brief, this machinery attempts to raise α, until $(\mathcal{K}_d)_\alpha \vdash (\mathcal{G})_{\underline{n(\alpha)}}$ no longer holds. The value of α that is obtained in this way, is used to determine the optimal decision, and the desirability and plausibility of its consequences.

- *S is the support of the argument. The support of the argument contains the knowledge from the agent's knowledge base $\mathcal{K}$ used to predict the consequences of decision d.*
- *C are the consequences of the argument. These consequences are goals reached by decision d, and form subset of goal base $\mathcal{G}$.*
- *d is the conclusion of the argument, and is a member of the set of all available decisions $\mathcal{D}$. Decision d is recommended by argument A.*

Aside of these requirements, three additional requirements apply:

- *$S \cup d$ is consistent*
- *$S \cup d \vdash C$*
- *S is minimal and C is maximal among the sets satisfying the above conditions.*

The set $\mathcal{A}_P$ gathers all the arguments which can be constructed from $\langle \mathcal{K}, \mathcal{G}, \mathcal{D} \rangle$.

According to this definition, an argument in favor of a decision is an *explanation* why the decision should be taken. The explanation explicitly refers to the (predicted) positive consequences of the decision, in order to convince the decision maker of the advantages of the decision.

Step 2: Selecting acceptable arguments After the decision maker has constructed all possible arguments in the set $\mathcal{A}_P$, it needs an algorithm to evaluate and compare these arguments. In other words, it needs a measure of the *strength* of an argument. In the framework, this strength depends on both the knowledge used, and the goals reached. Not only does the strength of an argument increase as the certainty of the knowledge on which it is based increases, but also as the consequences of a decision are more desirable in terms of the preferences of the decision maker.

In the framework, the certainty of available knowledge and the desirability of the consequences are referred to as the *Level*, and the *Weight* respectively. The strength of an argument in favor of a decision, and the notions of Level and Weight, are formalized in Definition 4.

Definition 4 (Strength of an Argument PRO)**.** *Let $A = \langle S, C, d \rangle$ be an argument in $\mathcal{A}_P$. The strength of A is a tuple $\langle Level_P(A), Weight_P(A) \rangle$ such that:*

- *The certainty level of the argument is $Level_P(A) = \min\{\rho_i | k_i \in S \wedge (k_i, \rho_i) \in \mathcal{K}\}$. If $S = \emptyset$ then $Level_P(A) = 1$.*

- *The degree of satisfaction of the argument is* $Weight_P(A) = n(\beta)$ *with* $\beta = \max\{\lambda_j | (g_j, \lambda_i) \in \mathcal{G} \wedge g_j \notin C\}$. *If* $\beta = \overline{1}$ *then* $Weight_p(A) = \overline{0}$ *and if* $C = \mathcal{G}$ *then* $Weight_P(A) = \overline{1}$.

From Definition 4, we see that the *Level* of the argument is equal to the certainty of the least certain piece of knowledge used. The *Weight* of the argument is slightly more complicated. It is inversely proportional to the priority of the most important goal that is not satisfied by the decision. Again, this is consistent with a pessimistic view of utility, because the focus is on the negative aspects of a decision, instead of the positive (i.e. goals that *are* satisfied).

Step 3: Concluding Using Definition 4, Amgoud and Prade (2004) construct a preference relation among arguments:

Definition 5 (Preference relation among Arguments)**.** *Let A and B be two arguments in* $\mathcal{A}_P$. *A is preferred to B iff* $\min(Level_P(A), Weight_P(A)) \leq \min(Level_P(B), Weight_P(B))$.

This definition allows us to compare decisions that are supported by different arguments, and to create an ordering between these decision. In the end, this is what we are interested in, as it helps the decision maker to identify the best possible decision in terms of the pessimistic evaluation criterion.

Definition 6 (Preference relation among Decisions)**.** *Let* $d, d' \in \mathcal{D}$. *d is preferred to* d' *iff* $\exists A \in \mathcal{A}_P, Conclusion(A) = d$ *such that* $\forall B \in \mathcal{A}_P$, $Conclusion(B) = d'$, *A is preferred to B.*

Put differently, a decision d is preferred to decision d', if it is supported by stronger arguments. The decision with the strongest argument is therefore selected as the final decision.

As final word about the pessimistic criterion, Amgoud and Prade (2004) establish Theorem 1, which links the strength of arguments to the notion of pessimistic qualitative utility from Equation 3.14.

Theorem 1. *Let* $d \in \mathcal{D}$. $E_*(d) \geq \alpha$ *iff* $\exists A \in \mathcal{A}_P$ *such that* $Conclusion(A) = d$ *and* $\min(Level_P(A), Weight_P(A)) = \alpha$.

Proof. Follows from Equations 3.12, 3.13, 3.14, and the commensurability assumption. □

3.5.2 Modifications to the framework

The framework described in the previous section deals with propositions from classical logic. Because our opponent modeling algorithm produces

fuzzy rule bases, we needed to adapt it to support fuzzy logic, without losing its essential properties.

Our modifications are concentrated in following areas.

3.5.2.1 The *Level* of arguments

In Definition 4, no attention is paid to the amount of truth or applicability of the knowledge used. Only its necessity is taken into account when assessing the reliability of knowledge. We found the strength with which rules fire (or *match strength*) to be an important influence on the *Level* of arguments.

Our solution to this problem is based on the following rationale:

- The new definition of *Level* should be a generalization of Definition 4. In boundary cases (i.e. where match strengths are 0 or 1), both the new and old specification of *Level* should produce the same value.
- If for $(k_i, \rho_i), (k_j, \rho_j) \in \mathcal{K}$, ρ_i is equal to ρ_j, the knowledge with the highest match strength determines the *Level* of the argument. The reason behind this, is that the higher the match strength, the more the piece of knowledge is applicable. The higher the applicability of the piece of knowledge, the more reliable it is in this particular case.
- If for $(k_i, \rho_i), (k_j, \rho_j) \in \mathcal{K}$, the match strengths are equal, the knowledge with the lowest plausibility ρ determines the *Level* of the argument. This is consistent with the specification of *Level* in Definition 4, where it is determined by the least certain piece of knowledge.

These considerations led to the following specification of *Level*:

$$Level_P(A) = \rho_j \times v_\omega(k_j) \tag{3.16}$$

where v_ω is the (fuzzy) valuation corresponding with the state ω, which is obtained after executing decision $d = Conclusion(A)$, and j is obtained using the following equation:

$$j = \arg\min_i \frac{\rho_i}{v_\omega(k_i)} \quad \text{for } \{(k_i, \rho_i) \mid v_\omega(k_i) \neq 0\} \tag{3.17}$$

It is fairly easy to see that when match strengths are limited to 0 or 1, Equations 3.16 and 3.17 reduce to the specification of *Level* in Definition 4.

3.5.2.2 The *Weight* of arguments

Definition 4 does not mention goals that could be partially satisfied. In fuzzy logic, goal propositions need not be totally satisfied or rejected. This

being the case, it is no longer possible to define a crisp set C of goals that are satisfied. Of course, the extent of satisfaction of a goal is important in calculating the degree of satisfaction.

The key requirement a new measure of *Weight* should conform to, is that it should increase as the number of satisfied goals increases. More specifically, if a goal g with preference λ is 50% true, we expect the utility to increase with $\lambda/2$. Sum over all goals to obtain the new definition of *Weight*:

$$Weight_P(A) = \sum_{(g_j,\lambda_j)\in\mathcal{G}} v_\omega(g_j) \times \lambda_j \tag{3.18}$$

where v_ω is the valuation corresponding with state ω, which is obtained after performing decision $d = Conclusion(A)$.

3.5.2.3 The *preference relation* among a arguments

The original preference relation is laid down in Definition 5. The commensurability assumption discussed in Section 3.5.1.1 proved to be impractical. This assumption asserts that plausibility of knowledge and preference of goals are measurable using a common scale.

In reality, comparing the *Weight* and *Level* of an argument using the min operator (as suggested in Definition 4) does not do justice to the different nature of these concepts. The confidence in the used knowledge, and the utility assigned to goals are not mensurable using a common standard. Instead of using the min operator, we chose for a more intuitive multiplication. Our definition of the preference relation is therefore:

Definition 7 (Preference relation among Decisions (Modified))**.** *Let A and B be two arguments in $\mathcal{A}_P$. A is preferred to B iff $Level_P(A) \times Weight_P(A) \leq Level_P(B) \times Weight_P(B)$.*

3.6 The complete system: ABDM

In Section 3.4 and 3.5, we discussed the opponent modeling and the decision making component of our approach respectively. Using these two components, we can now construct the complete system. This system, pictured in Figure 3.1, is called ABDM: the Argumentation Based Decision Maker. The ABDM combines opponent modeling and argumentation based decision making in one single system. To to this, it interconnects its two major components using a number of algorithms. To do this, we need to find an answer to the following questions:

1. How frequently is the opponent model updated? More specifically, when is FURL used to create a new model of the data set?

2. What information is stored in the transaction database, and which part of this information is input to opponent modeling?
3. How are the opponent modeling and decision making components connected? Rephrasing this question for our choice of algorithms: how is the opponent model produced by FURL used by the argumentation framework?

In the following, we address each of these questions one at a time.

3.6.1 Update frequency

Strictly speaking, each time new data is added to the transaction database, the existing opponent model becomes stale. It no longer reflects the complete behavior of the opponent, because it does not take all its actions into account. Less strictly, however, staleness of the opponent model need not be a major problem. One of the properties of a good opponent model, is that it is able to predict the outcome of never-before seen situations. Hence, updating the model each time new data is received could potentially be a waste of computational resources.

Of course, this question would be moot or at least less relevant, if the chosen opponent modeling algorithm had been an on-line learning algorithm, instead of a batch algorithm. An on-line algorithm is capable of refining the learned model to reflect the changes in the data set. In theory, such an updation procedure is much faster than regenerating the model from scratch each time the data set is extended with a new datum. Unfortunately, FURL is a batch algorithm, which means that it must rebuild the model to reflect the changes in the data set, instead of refining it.

The choice for update policy should therefore be taken with some care. To this end, we studied a number of different update strategies. These alternatives were:

- Update when n new datums are added. This is the simplest update strategy. The parameter n has to be chosen based on the context.
- Update when the prediction error exceeds a predetermined threshold. The height of this threshold depends on the context. Based on our findings, it is not trivial to find a suitable threshold. As we will see in Section 4.4, modeling different types of agents cause very different levels of error. Using a single threshold on these diverse agents would have two possible negative side-effects. Either a poor quality model is obtained of agents whose behavior is easily detected (because the low errors do not exceed the threshold and no update command is issued), or a lot of computational resources are wasted on agents whose behavior is more difficultly detected, due to constant updates (because the error continuously exceeds the threshold).

- Update when the prediction error exceeds a varying threshold. The threshold is continuously adapted using resulting prediction errors. For example, we can use the moving average of the prediction errors as a threshold. If the error exceeds the average, an update command is issued. Experiments with this update strategy showed that the ABDM updates too frequently when an opponent suddenly changes its behavior,[15] and thus wastes resources.

The third update strategy seems to be a good alternative. However, it conflicts with our requirement to reduce complexity. We therefore selected the first update strategy for our experimentation. We think, however, that the third alternative deserves more attention in further research.

3.6.2 Selecting input features

The nature and amount of information that is extracted from the environment and stored in the transaction database is very context dependent. Based on the nature of the context, the user has to decide which features from the transaction database are relevant in predicting the opponent's behavior. For example, an important influence on an opponent's behavior is the quality of service delivered *to* this opponent (especially with the RECIPROCAL agent discussed in Section 4.3.3.2). We come back to this in Section 4.4.1 where we select the input features for our experimentation.

3.6.3 Connection between FURL and Argumentation

In Section 3.5.1 we have seen that the selected argumentation framework uses two knowledge bases. The first base contains the agents goals. These goals are determined by the user. The second base contains the knowledge the agent has about its opponent. This knowledge comes from the opponent model produced by FURL.

Remember from this section that the knowledge base consists of pairs (k_j, ρ_j), where k_j denotes a proposition, and ρ_j is a necessity measure for k_j. Note the similarity with the opponent model produced by FURL: an HPS consists of several rules describing the behavior of the opponent. These are the propositions k_j.

As for the necessity measure ρ_j is concerned, the most obvious choice is the credit assigned to the rules in the HPS. In Section 3.4.2.2 we saw that credit is a measure for the amount of error a rules is believed to cause. The higher the credit, the poorer the quality of the rule (or the less certain it is). The necessity measure ρ_j, however, *increases* as the rule is more certain, whereas credit *decreases* as the rule is more certain.

[15] For example, see the experimental results for SWITCH and RECIPROCAL agents in Sections 4.4.6 and 4.4.7.

We therefore inverted the *credit* to obtain a measure of necessity ρ:

$$\rho = \frac{1}{credit} \tag{3.19}$$

3.7 Summary

This concludes the discussion of the design of our architecture. In this chapter, we laid down the requirements that had to be met by our design. The most important requirement was the ability of the architecture to *explain* decisions related to trust in Multi-Agent System. We translated this requirement into two design decisions.

First, we chose to satisfy this requirement by clearly separating opponent modeling and decision making. This way, a clear distinction between the behavior of an opponent, and the goals of the agent itself is guaranteed. The main benefit of this separation, is that it allows reconstructing the process in which the decision was made (i.e. learning the opponent's behavior, weighing alternatives, and deciding).

Second, we designed these two components to work with explicit (or symbolic) forms of knowledge representation. This decision was made, because an explicit knowledge format is much easier to understand than a numerical knowledge format (for examples of the latter and their drawbacks, see Section 2.2). Due to the uncertain and continuous nature of data in the trust domain, we chose for fuzzy logic to represent knowledge. Suitable algorithms were subsequently selected for both opponent modeling and decision making.

In the next chapter, we subject the proof-of-concept of our design to experimentation. This gives us an opportunity to observe and study the both the architecture itself, and its components individually, and identify their strengths and weaknesses.

4
Empirical Results

In the previous chapter we discussed the design of our proposed approach and the theory behind it. Using an implementation of this design, we are now able to experiment with our approach, in an attempt to identify its strengths and weaknesses, and determine to what extent the design requirements from Section 3.1.2 are met. This involves running a number of carefully laid out experiments.

In this chapter, both the setup and the results of these experiments is discussed. It is structured as follows. First, we determine what guidelines need to be followed to setup our experiments. These guidelines are covered in Section 4.1. These should ensure that the results of the experiments are correct and useful, and that we can draw reliable conclusions from them. Next, in Section 4.2, we translate the aforementioned design-requirements from Section 3.1.2 to measurable objective criteria. These criteria allow us to objectively measure the performance of our approach. This enables us to determine if it can live up to the research community's expectations, as well as our own. Using these criteria, we are able to design and setup our experiments in Section 4.3.

In Sections 4.4 and 4.5, we focus our attention to the actual experiments. Each of these sections highlight one of the two major components of our approach. In Section 4.4, emphasis is laid on the opponent modeling capabilities of our experiment. The experiments in Section 4.5 focus on decision making. In Section 4.6, we focus on the complete system: the two components are brought together, and the performance of the complete system is subject to experimentation. We close the chapter by evaluating the results in light of the evaluation criteria from Section 4.2.

4.1 Experimentation Principles

To ensure that the results of our experiments are both usefulness and accurate, we adopted guidelines proposed by Johnson (2001). In his article, the author warns for several pitfalls of computational experimentation and gives several suggestions.

His emphasis is on algorithms for which the key metrics are running time and memory usage. Our approach, however, can be better characterized as an approximation algorithm. Therefore, our main evaluation criteria will focus on the quality of the solution (for both modeling and decision-making components). At this time, resource efficiency is secondary. We therefore disregard the principles and suggestions pertaining to this kind of efficiency. Nevertheless, the majority of the suggestions and principles presented by Johnson can be adapted to suit our needs, and help to make our experiments worthwhile.

Note that these principles are sometimes quite idealistic, in the sense that in theory, they are very self-evident and natural. In practice, we are not always able to conform these principles. In some cases, we found it hard to follow them, or even impossible. We will indicate when and where this was the case.

We have selected three principles from Johnson (2001) to which we paid special attention.

4.1.1 Effective and efficient experimental design

During the implementation phase, we have laid a strong emphasis on a flexibility and efficiency (see Appendix A). This really paid off during the experimental phase. Not only was it possible to easily modify parameters of both the framework and the testbed, but also to plug-in different algorithms.

Also, it was quite easy to add components responsible for saving all relevant data in a human readable format, to make the results *self-documenting* (a key suggestion of Johnson (2001)).

The article also warns for spending too much time on the wrong questions. For example, if the simulations contain too much (unnecessary) random elements, we have to run these simulations multiple times to compensate for the variability of single instances of the test runs. To avoid this, we aimed to reduce the number of random elements in our experiments. In Johnson (2001) this is referred to as *variance reduction*.

4.1.2 Reproducibility

In contrast to what one might expect, 'reproducibility' in context of computational experimentation does not only refer to the ability to obtain the exact

same results after running the same experiment with the same parameters on the same machine. Rather, viewed from a methodological perspective, it means that it should be possible to draw the same conclusions based on similar experiments. In the words of Johnson (2001): 'similar' amounts to using the 'same basic methods' and 'possibly different measuring techniques'.

The idea behind these requirements, is that a third party is capable of confirming that the conclusions drawn from the experiments are independent of the details of the experiments themselves.

So, what are the implications of this principle on the way we did our experiments? Firstly, we made sure that all relevant configuration details were saved with the data themselves (in accordance with the *self-documenting* suggestion), so that the majority of influences on the final result are accounted for. Secondly, we tried to use objective metrics where possible (see next section).

However, because the solution quality of the opponent-models is also measured in terms of human comprehensibility (e.g. 'is the description of the agent's behavior intuitive?'), we did have to sacrifice some objectivity. To somewhat counterbalance this, we do provide full disclosure: we provide a representative sample of the results that require a subjective assessment.

4.1.3 Use testbeds that support general conclusions

Trust in Multi-Agent Systems is a wide-spanning topic. It proved difficult to find a set of problem-instances that are representative for the topic as a whole.[1]

Fortunately, a number of researchers created the Agent Reputation Testbed (ART). On the one hand, ART is designed as a tool for experimentation, which researchers can use to perform easily-repeatable experiments. On the other hand, ART can be used to compare different trust-technologies with each other, using objective metrics (See Fullam et al. (2005)). These properties make ART a suitable testbed to focus our experimentation efforts on. In Section 4.3.1 we explain ART in more detail.

4.2 Evaluation Criteria

In light of the reproducibility principle above, we need metrics to evaluate the results of these experiments. These metrics help determine the value of our approach, and its strengths and weaknesses remain.

Fortunately, a number of evaluation criteria for trust algorithms in general already exist. These criteria are based on research objectives that the

[1] This might explain the plethora of experimental domains (See Fullam et al. (2005))

research community has reached consensus on. Fullam et al. (2004) lists eight of these: five for the opponent modeling capabilities of a trust algorithm, and three more for the decision making capabilities. In terms of a opponent modeling, a trust algorithm should be:

Accurate A model must accurately predict the behavior of an opponent. The similarity between the predicted behavior and the true behavior can be used as an indicator for accuracy.

Adaptive An agent's behavior is not static. It can change over time. A modeling algorithm should adapt its model to reflect the changes in the agent's behavior.

Quickly Converging Not only should a model adapt to changes, it must do this quickly. In open Multi-Agent Systems (where agents can enter and exit as they please), it is possible that agents leave and enter the system to take on a new identity. In this event, the modeling algorithm must quickly construct a model of this unknown agent. Otherwise, it would be vulnerable to exploitation. To assess convergence, the time is measured between the moment an agent changes its behavior (or enters the system), and the moment the model is again accurate enough.

Multidimensional The model must reflect an opponent's trustworthiness across multiple categories. For example, in a certain context, there could be multiple kinds of transactions.

Efficient An opponent modeling algorithm should be efficient in terms of computational resources and time. This is measurable by timing the amount of time a update costs on a predefined machine.

Concerning the decision making abilities, a trust algorithm should be able to:

Determine whether to interact For each possible transaction partner, an agent must be able to decide whether to interact with it. Because of this, the agent needs to predict whether the partner will live up to its end of the deal. This can be measured in terms of the number of successful decisions made by the agent, compared to the total number of initiated transactions.

Evaluate an interaction's utility Before deciding to participating in a transaction, an agent must be able to predict the resulting utility of that transaction. This is important, because knowing the utility, the agent is better able to negotiate payment.

Identify and isolating untrustworthy agents A trust algorithm must identify malicious agents and refuse to interact with them, in order to protect itself from exploitation.

Not all of these criteria are relevant in our experiments. In particular, *efficiency* was not one of the design goals. Our proof-of-concept has not been optimized for performance. In what follows, this requirement is not used. The same applies for the requirement of *multidimensionality*. The architecture in Figure 3.1 can be used on multiple types of transactions. However, in our experiments, we only use one type of transaction.

The requirements above consider both components as black boxes: they evaluate the *outputs* of both the modeling and the decision making phase. For example, modeling is evaluated in terms of accuracy and adaptiveness, not in terms of what the model actually looks like, or if it can be understood. Similarly, the decisions are evaluated in terms of expected utility and acceptable results, while neglecting the reasons *why* they are made.

In light of the aim of our research, we therefore propose to add two important requirements that specifically focus on explanation: the primary goal for which our approach is designed. For the modeling component, we have a single extra requirement:

Comprehensibility of the model The user must be able to comprehend the opponent model, and be able to understand the behavior of the opponent by looking only at the model. More specifically, is there a correspondence between the rules in the fuzzy rule base and the actual behavior of the opponent, and is this correspondence easily understood?

The explanations of the decisions produced by our agent are subject to the following requirement, analogous to the requirement for opponent modeling:

Comprehensibility of the arguments In other words, is the rationale behind a decision accessible to the user? If so, is it clear on which parts of the opponent model the decisions are based, how the expected utility is derived, and how decisions are weighed relative to each other?

4.3 Experimental Setup

Now that we have a set of measurement criteria at our disposal, we can discuss the general setup of our experiments. That is, we describe the setup common to the experiments that are discussed in the upcoming sections.

We start by giving a detailed description of the testbed used in our experiments. This testbed was specifically designed for testing and evaluating algorithms for trust in Multi-Agent Systems.

Next, we show which aspects of the testbed we adapted to suit our specific experimentation needs. These adaptations were performed to keep the influence of random elements in the testbed in check, in compliance with the principle of variance reduction described in Section 4.1.

In the final part of this section, we present the agents used as opponents.

4.3.1 The ART Testbed

In our experiments, we used the Agent Reputation and Trust (ART) Testbed. This testbed was created by a group of researchers in the field of trust in Multi-Agent Systems (Fullam et al., 2004). Among the reasons for creating this testbed was the desire to focus the attention of the trust community on one single problem, in an attempt to counter the uncontrolled growth of problem domains used by researchers in the trust domain. In theory, the testbed could facilitate the comparison of existing approaches, which was previously impossible due to the aforementioned plethora of experimental domains. Hopefully, it will increase the cohesion of the trust community, and stimulate a discussion on the strengths and weaknesses of existing approaches.

In what follows, we informally present the testbed. In the next section, we go into the formal details.

The acronym ART not only stands for "Agent Reputation and Trust", but also refers to the problem domain of the testbed. In the ART testbed, agents take on the role of art appraiser. As appraisers, they are visited by clients who wish to know the value of one of their paintings. For each client served, the agent is awarded a predetermined amount of money, irrespective of the accuracy of the appraisal. (A measure for the accuracy of an appraisal is difference between the appraisal and the true value of the painting.) However, the accuracy of the appraisal *does* determine the number of clients that return to the agent in the next round. So, if an agent's appraisals are relatively accurate compared to the appraisals of others, the agent can expect more clients. If not, these clients will visit its competitors in the hope that these will offer better services.

In ART, paintings are categorized in eras (e.g. baroque, neo-classical, cubism). Of course, to accurately appraise a painting, an agent should have enough expertise for the painting's era. At the beginning of the simulation, these expertises are determined at random. So, the most likely scenario is that, at the one hand, an agent has some eras for which it is able to accurately estimate the value of paintings, but on the other hand, it also has a lack of expertise in other eras.

4.3.1.1 Opinion Exchange

Despite this, an agent could still be requested to appraise a painting for which it has insufficient expertise. In this case, it is in its best interest to consult other agents, because appraising the painting on its own will result in an inaccurate appraisal. Fortunately, in ART, agents are able to exchange opinions about a paintings value. The protocol for such an exchange is depicted in Figure 4.1.

First, one agent (the requester) asks whether a second agent (the provider) is willing to help appraise a certain painting. If so, the provider lets the requester know how confident it is to appraise the painting by stating its certainty. Of course, the provider is free to misrepresent itself. For example, it can grossly overstate its certainty, thereby attempting to persuade the requester to send its inaccurate appraisal to the requester's client.

After receiving the certainty, the requesting agent decides whether or not it wishes to proceed. If so, it pays a small amount of money to the provider. Finally, the provider determines how much effort it wants to put into its appraisal. (For example, examining the painting requires time and money.) Of course, the more effort the provider puts in, the more accurate its opinion. After determining the amount of effort, the provider submits it to the testbed. At the end of the round, the testbed calculates the appraisal based on the provider's expertise and effort.

4.3.1.2 Appraisal Generation

After all opinions have been exchanged, the provider agent needs to determine how to weigh these in its final appraisal (the appraisal that is sent to the client). These weights are submitted to the testbed, which then proceeds to calculated the final appraisal by taking the weighted average of the received appraisals.

Not until the appraisals have been sent to the agent's clients, are the appraisals of other agents revealed by the testbed to the requesting agent. This way, the agent is kept in the dark about the actual quality of the appraisals it requested from other agents until the client has already been informed of the appraisal. This way, the requester has to *entrust* a part of its welfare to the provider, because the provider is given (partial) control of the appraisal that is sent to the customer.

4.3.1.3 Reputation Exchange

To reduce the uncertainty about the trustworthiness of the provider, the requesting agent is free to consult other agents about its reputation. The reputation of an agent is the opinion of its reliability and trustworthiness in the eyes of others. In ART, the exchange of reputation follows the proto-

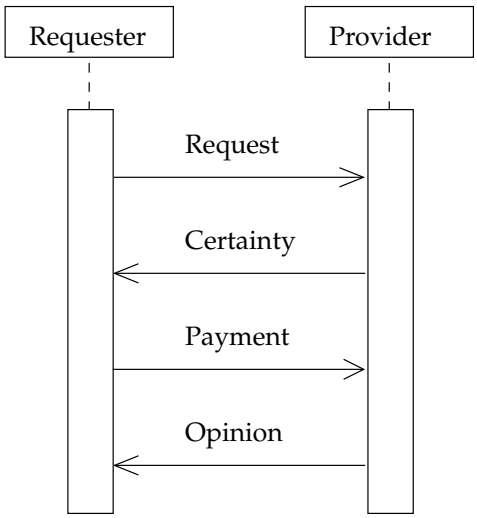


Figure 4.1: Opinion transaction protocol in the ART Testbed

col in Figure 4.2. This protocol is very similar to that of opinion exchange, with the exception that the reputation is directly revealed to the requester, instead of to the testbed. This allows the requester to use the received reputations in determining which agents will determine its final appraisal. Of course, these reputation need not be revealed truthfully.

4.3.1.4 Goal of the Competition

To complete the competitive element of the testbed, the goal of the competition is to maximize an agent's account balance. Most importantly, this is accomplished by servicing as many clients as possible by accurately appraising paintings.

4.3.2 Adapting ART for Experimentation

Although the official specification of ART is very suitable for competition, using an unmodified version in our experiments would introduce too much complexity. This would shift the emphasis of our experiments from our own approach to the testbed itself.[2] Of course, this is not desirable.

Therefore, we deviated from the official specification in several respects, reduced emphasis on some elements and laid focus on others. In the following, we briefly describe our modifications, and the rationale behind them.

[2]In fact, Johnson (2001) warns for this pitfall: to start with problem instances to evaluate the behavior of an algorithm, but end up evaluating the properties of those problem instances

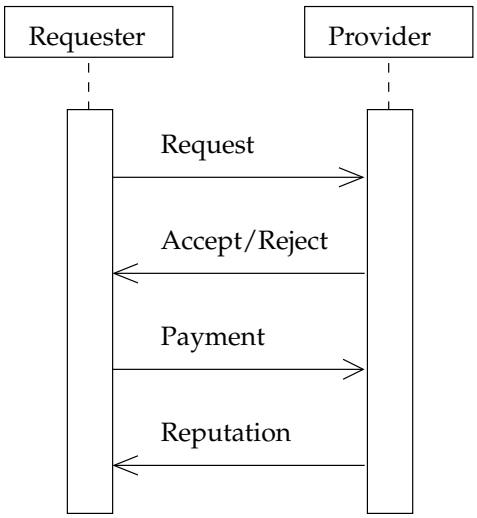


Figure 4.2: Reputation transaction protocol in the ART Testbed

Notation	**Meaning**
$owner(p)$	the agent responsible for appraising painting p
$era(p)$	the era painting p belongs to
$value(p)$	the value of painting p
$expertise_A(e)$	Agent A's expertise for era e
$expertise_A(p)$	Shorthand for $expertise_A(era(p))$ (p is a painting)
$expertise_{max}$	The maximum expertise an agent can have
$appraisal(A, p)$	appraisal of agent A for painting p
$certainty_A(p)$	asserted certainty for painting p by agent A towards $owner(p)$

Table 4.1: Notation for concepts in the ART Testbed

The notation used throughout this section is summarized in Table 4.1.

4.3.2.1 Ignore effort

In the discussion of the ART testbed, we briefly explained the effect of effort on the accuracy of the appraisal. By changing the amount of effort, an agent is capable of compensating for low expertise, or to deceiving an opponent by taking no effort at all.

Effort is certainly a valuable feature in the ART testbed, because it allows agents to vary the quality of their appraisals for a fixed certain era. Despite this, we decided to omit effort in our experiments, because it is not a necessary feature. Even without effort, an agent's trustworthiness is still determined by its mapping from expertise to certainty (see Figure 4.3): its trustworthiness depends on whether or not it honestly asserts its certainty

based on its expertise. An outsider only observes the asserted certainty and the resulting appraisal-error of a transaction, because the amount of effort is always private to the agent supplying an appraisal. It is for this reason that we leave out effort in our experiments, without loss of generality.

4.3.2.2 Reduce indeterminism

The specification of ART calls for randomly generated expertises. For each era, an agent's expertise is drawn from an uniform distribution. However, generating expertises this way, we would need to run an experiment many times to get useful results, and makes it hard to draw useful conclusions. Therefore, expertises are evenly distributed between zero and the maximum possible expertise in our experiments.

Also, the method for generating appraisals according to the ART specification would introduce too much variance in the results of our experiments.[3] The original specification uses a normal distribution to determine an agent's appraisal based on its expertise. This has the same drawbacks as mentioned above: not only would it require a large number of runs to obtain reliable results, but also a lot of statistical preprocessing before transaction data can be offered to FURL. After all, FURL was not designed to deal with a large *variance*, but rather with *fuzziness* in data. Experiments to determine FURL's capability to handle variance in data, or to determine what kind of preprocessing is needed, are considered outside the scope of our project.

Instead, we used a different appraisal generating function:

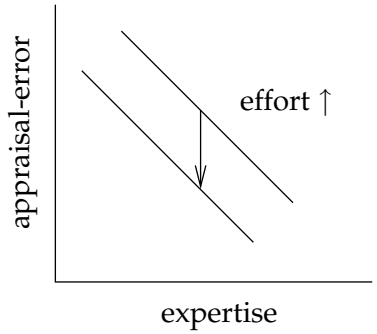
$$appraisal(A, p) = value(p) \times \left(2 - \frac{expertise_A(p)}{expertise_{max}}\right) \tag{4.1}$$

This function has two important properties. First, is deterministic, in the sense that its result only depends on the input parameters. The result is not drawn from a probability distribution, as was the case in the original specifications. Second, it makes sure that if an agent is very skilled (i.e. has a high expertise), the appraisal is completely accurate. If the agent has no expertise, the appraisal is twice as high as the true value of the painting. In that case, the appraisal error is 100%.

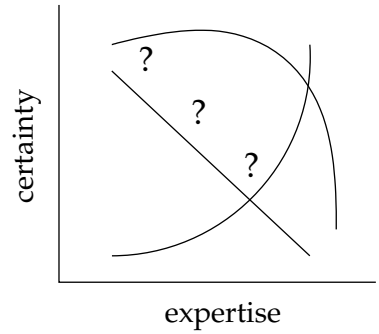
4.3.2.3 Focus on Appraisal Transactions

In this research, our focus lies on direct interaction between agents. Dealing with the reputation of potential transaction partners is outside the scope of our approach. Therefore, the possibility in ART to initiate reputation

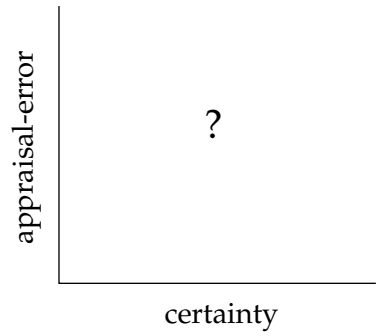
[3] For more details on the generation of appraisals in ART, we would like to refer to the official specification in Fullam et al. (2004).



(a) The mapping from expertise to error. If effort increases, error decreases



(b) The mapping from expertise to certainty is entirely up to the agent. This mapping determines its trustworthiness, and is private to the agent.



(c) The mapping from certainty to appraisal error is unknown to outsiders because the mapping from expertise to error is private to the agent. Even without the use of effort, the relation between certainty and appraisal error (which determines the agent's trustworthiness) can take on many different forms, depending on the mapping between expertise and certainty.

Figure 4.3: The relations between expertise, appraisal-error, and certainty in the ART Testbed.

transactions has been left unused in our experiments. In Section 5.2 we explore some possibilities to include reputation in our approach.

4.3.3 Opponent Agents

In our experiments, we used a number of agents to put our approach to the test. These agents all have different behavior, and emphasize different elements of both the identification and decision making abilities of our agent. The behavior of these agents is not exclusive to our work, but is also found in the work of others in this area of research. For example, Falcone et al. (2004) use a random agent, Axelrod (1984) describes a reciprocal (or Tit-for-Tat) agent, and Yu and Singh (2003) describe different deception models that are analogous with our HONEST, DISHONEST, and NEUTRAL agents. In the remainder of this section, we present the three classes of agents used in our experiments.

4.3.3.1 Unresponsive Agents

The class of unresponsive agents consists of agents that are insensitive to the behavior of other agents in the system. Remember from the description of the ART testbed, that agents have both an appraisal-provider and an appraisal-requester role: they request appraisals of their paintings from other agents, and they in turn can give appraisals of other agent's paintings. The essential property of unresponsive agents is therefore, that they do not take the quality of other agent's appraisals into consideration when appraising paintings themselves. Instead, they appraise paintings according to a predefined procedure that only takes the painting and their expertise into consideration, and ignores the identity of the requesting agent.

HONEST The HONEST (H) agent is the simplest of all agents. It simply and honestly asserts its true expertise for a particular painting to the requesting agent. In our implementation, the asserted certainty is equal to the normalized true expertise:

$$certainty_H(p) = \frac{expertise(p)}{expertise_{max}} \tag{4.2}$$

where $expertise(p)$ is equal to the expertise of the honest agent for painting p, and $expertise_{max}$ is the maximal expertise possible for a particular run of the testbed.

DISHONEST As the name suggests, DISHONEST (D) is the precise opposite of HONEST. Its asserted certainty is the exact opposite of what one

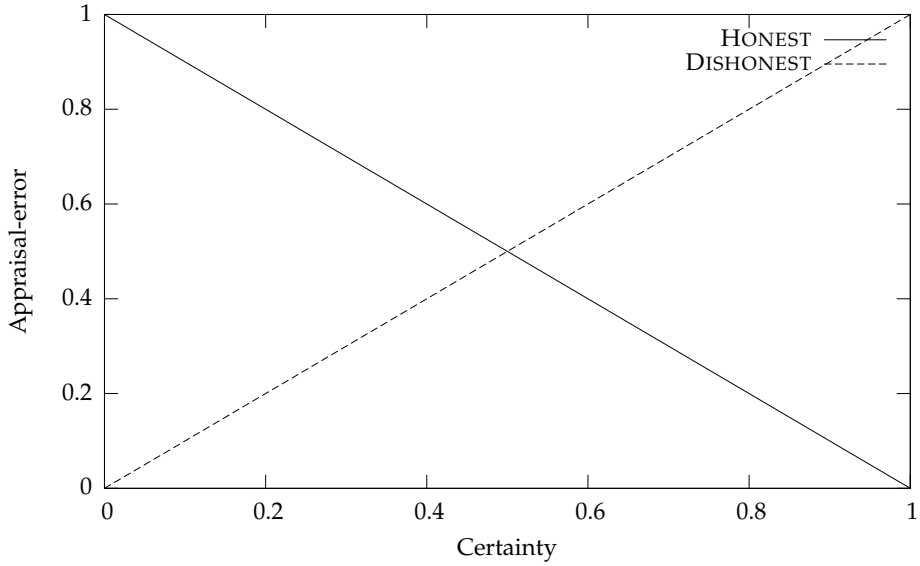


Figure 4.4: Relation between asserted certainty and expertise for HONEST and DISHONEST

would expect based on DISHONEST's expertise:

$$certainty_D(p) = 1 - \frac{expertise(p)}{expertise_{max}} \tag{4.3}$$

RANDOM RANDOM's (R) behavior is quite simple. Each time RANDOM receives a request for appraisal, it replies with a random certainty:

$$certainty_R(p) = rand() \tag{4.4}$$

where the $rand()$ function returns a random real in the interval $[0, 1\rangle$.

NEUTRAL The behavior of NEUTRAL (N) is constant: it always asserts the same certainty. Consequently, from the opponent's point of view, it is not possible to determine whether or not NEUTRAL has sufficient expertise:

$$certainty_N(p) = c \tag{4.5}$$

4.3.3.2 Responsive Agents

Whereas unresponsive agents do not respond in any way to the service provided by an opponent, responsive agents have the ability to adapt their behavior to that of their opponents. In the context of the ART testbed, this ability can materialize in the form of 'punishing' opponents for deceiving

the agent in previous round(s). Intuitively, we could imagine deception in ART as deliberately misrepresenting one's expertise. A formalization of the concept of deception is used to implement the most important agent of this class.

RECIPROCAL The essential property of the RECIPROCAL (RP) is that it (as its name suggests) reciprocates both deception and honesty to its transaction partners. That is, honesty is rewarded by honesty from RECIPROCAL, and dishonesty punished by dishonesty. The strategy this type of agent uses, is also referred to as 'Tit for Tat' (Axelrod, 1984).[4] A definition of reciprocity was given on page 11.

More specific, our implementation of RECIPROCAL takes the opponent's honesty in the previous round into account when deciding on the certainty to assert. When the opponent has been dishonest by misrepresenting its expertise, RECIPROCAL reciprocates by severely overstating its capability by asserting a higher certainty than is warranted for a particular era. Before discussing the details of this process, we first explain the concept of honesty.

Definition 8 (Honesty)**.** *A measure for honesty (or rather dishonesty) is the difference between the expected appraisal based on the asserted certainty, and the actual appraisal.* RECIPROCAL *assumes that the asserted certainty is a measure for the true expertise of its opponent A. Based on this assumption, the expected appraisal can be obtained by using the appraisal generation function from Equation 4.1 (see page 68):*

$$appraisal_{expected}(A,p) = value(p) \times \left(2 - \frac{certainty(A,p)}{expertise_{max}}\right) \tag{4.6}$$

Now, dishonesty *can be defined as:*

$$dishonesty(A,p) = \frac{\|appraisal_{expected}(A,p) - appraisal(A,p)\|}{appraisal_{expected}(A,p)} \tag{4.7}$$

The difference between the expected and actual appraisal is divided by the actual appraisal, to facilitate comparison of dishonesty between several paintings of different value.

Using the *dishonesty* function, we can now complete the details about RECIPROCAL's behavior. After receiving a request for appraisal, RECIPROCAL replies with a certainty according to the following function:

$$certainty_{RP}(p) = \frac{expertise(p)}{expertise_{max}} \times (1 + rs \times dishonesty_{avg}(owner(p)) \tag{4.8}$$

[4]The name of this strategy comes from an English saying meaning 'equivalent given in retaliation [...]' (Oxford Dictionary)

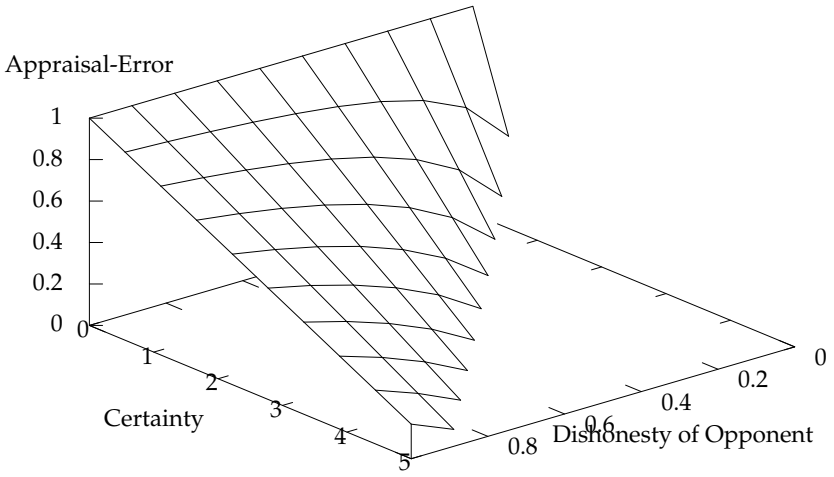


Figure 4.5: Relation between asserted certainty and appraisal-error in the behavior of RECIPROCAL

where rs is the retaliation-strength (the strength with which a deceitful agent is punished), and $dishonesty_{avg}(A)$ is the average dishonesty over the previous round of agent A. Less formally, Equation 4.8 states that RECIPROCAL overstates its certainty rs times *more* than its opponent overstated its certainty in the previous round.

Now, if $expertise_{max} = 1$, the relation between *dishonesty*, *certainty* and *appraisal-error* becomes:

$$appraisal\text{-}error(p) = 1 - \frac{certainty(p)}{1 + rs \times dishonesty_{avg}(owner(p))} \tag{4.9}$$

because $appraisal\text{-}error(p) = 1 - expertise(p)$.

The graph depicted in Figure 4.5 shows RECIPROCAL's behavior according to Equation 4.9 with $rs = 5$.

4.3.3.3 Complex Agents

Complex agents do not behave according to a single predefined behavior. Instead, they combine simple behaviors. In our experiments, we focused on one single class of Complex Agents:

SWITCH SWITCH (S) is an agent that switches between behaviors on a predefined moment. For example, we defined SWITCH to first behave as HONEST, and subsequently switch to the behavior of DISHONEST after half

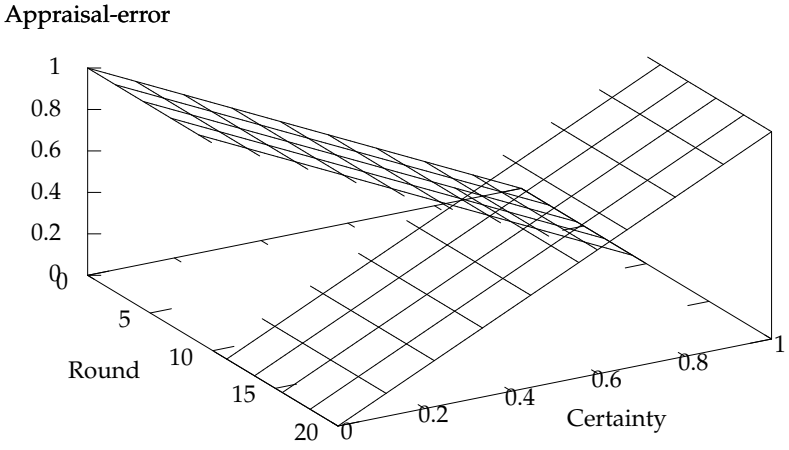


Figure 4.6: SWITCH's behavior with switch from honest to dishonest behavior in round 10

of the simulation is complete. More formally, the behavior of such an agent is expressed by:

$$certainty_S(p) = \begin{cases} certainty_H(p) & \text{if } round \leq n \\ certainty_D(p) & \text{if } round > n \end{cases} \tag{4.10}$$

This agent is first builds up a reputation of trustworthiness by acting honestly. After a while it switches to dishonest behavior, and 'spends' its built-up social capital, in an attempt to deceive its opponent. An example of this kind of behavior is presented in Figure 4.6.

4.3.4 Measures of Interest

For our experiments, we instrumented our code to record various interesting data. Of course, the generated data we are interested in differs for each set of experiments. For the identification experiments, we mainly recorded the difference between the predicted appraisal error and the true appraisal error of an opponent. Also we maintained a record of the resulting fuzzy rule bases. During the decision making experiments, we were more interested in the arguments supporting a decision, and their strengths. We come back to these measures in more detail in the discussion of the individual experiments.

However, there were several measures of key interest that we recorded for all our experiments. To make the data generated by our simulation

self-documenting, we recorded all parameters of both the testbed and the agents. Moreover, all transactions were recorded. These data make it possible to re-run the simulation exactly.

4.4 Identification

In this section, we put the opponent modeling component of our approach to the test. This is done by letting our approach compete against all the agents described in Section 4.3.3. During each experiment, we measure how well the results of the next transaction are predicted, based on data from transactions in the past (in accordance with the criterium of accuracy from Section 4.2). Also, after each experiment, we check how well the model can predict the complete behavior of its opponent (not only a single transaction), based on the finite number of observed transactions. Using both results, we make some general statements about the usefulness of FURL in our approach, and decide whether or not FURL meets the demands set in Section 4.2.

4.4.1 Setup

The experiments described in this section share a common setup. The relevant parameters of this setup are summarized in Tables 4.2 and 4.3. In the latter table, "Measured Features" are believed relevant for predicting transaction results. These are the certainty asserted by the opponent, the current round number, and the dishonesty towards the opponent. The "Learned Feature" is the actual property of transactions that is to be predicted. In these experiments, this feature is the appraisal error: the difference between the appraisal submitted by an opponent, and the true value of a painting.

In Section 3.4.3, we saw that the number of partitions influenced both the extent of overfit and accuracy of the obtained model. On the one hand, too much partitions cause the training data set to be perfectly described, while the validation data set cannot accurately be predicted due to overfit. On the other hand, too little partitions cause the learned model to be too crude, so that both the training and the validation data set are inaccurately predicted.

In a small experiment, we used a data set generated from the behavior of RECIPROCAL. From this data set, we chose three random subsets containing 60% of all elements in the original dataset. On each of these data sets, we ran the FURL algorithm. Next, we sampled the errors in a mesh of 50×50 points. The average over the thusly obtained 2500 points are plotted in Figure 4.7.

From this figure, we can see that 8, 10 or 11 partitions are good choice. In comparison with other configurations, these numbers produce a rela-

Parameter	Meaning
Minimum Expertise	0.1
Maximum Expertise	0.9
Number of Eras	9
Number of Clients per Agent per Round	10
Number of Rounds	20
Maximum Painting Value	1000

Table 4.2: Testbed Parameters

Parameter	Meaning
Update Strategy	Every 5 rounds
Measured Features	Round, Certainty, Dishonesty
Learned Feature	Appraisal Error
Number of Partitions	8
Stopping Criterion	1% drop in error
Partitioning Strategy	K-Means
Beam-width	5

Table 4.3: ABDM Agent Parameters

tively low error. However, because the difference is so small, we prefer 8 partitions, because it limits the amount of computation, and the number of rules in the resulting rule base.[5]

4.4.2 Results

4.4.2.1 HONEST

We begin our review of the results with the simplest agent of them all: HONEST. As said in Section 4.3.3.1, HONEST is an unresponsive agent, in that it does not adapt its behavior to that of others.

In this first experiment, we let our agent compete with HONEST using the setup described in the previous section. For each transaction, we measured the difference between the prediction of the learned behavioral model (the learned HPS) and the actual behavior of the HONEST. Figure 4.8 shows the progression of the prediction-error over time. HPS listing 4.1 shows the learned model of HONEST's behavior.

It is relatively easy to see that the rule-base in Listing 4.1 corresponds with HONEST's specification discussed in Section 4.3.3.1. Note that the input domain of the 'certainty' and 'appraisal-error' variables are partitioned in 6 partitions, instead of the expected 8 partitions (see Figure 4.9). This is

[5]Remember from Section 3.4.4 that as the number of partitions increases, the number of possible exceptions that have to be examined increases with it, resulting in more rules. A larger number of rules can clutter the rule base, making it harder to comprehend.

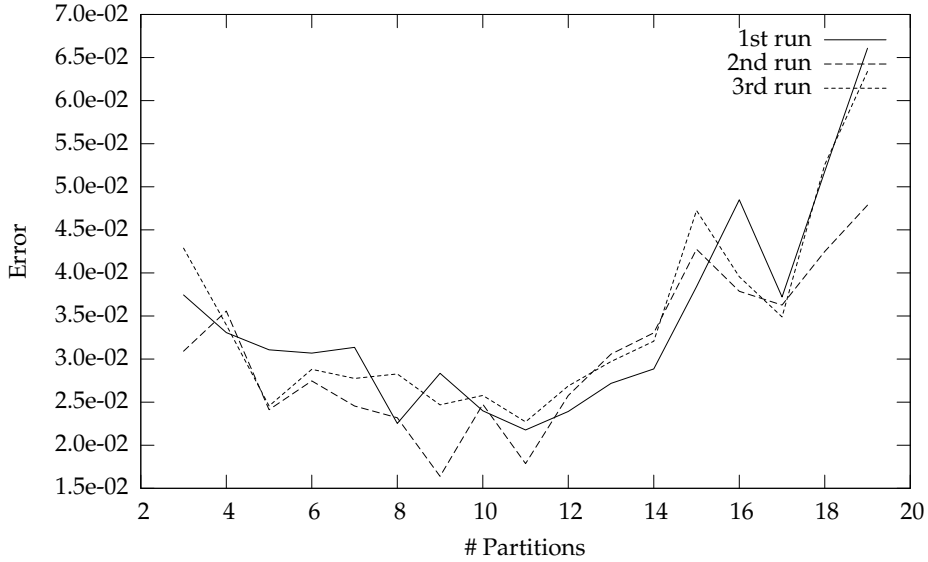


Figure 4.7: Errors of the learned models for different numbers of partitions on three 60% random subsets of the data set generated from RECIPROCAL's behavior

caused by the fact that the partitioning method (K-Means in this case) identified only 6 *distinct* partitions.

HPS Listing 4.1 Model of HONEST's behavior after 200 interactions

	Rule	Credit	Matches
1	**if** certainty is 0 **then** relative-error is 5	0.55590	52
2	**if** certainty is 1 **then** relative-error is 4	0.25440	56
3	**if** certainty is 2 **then** relative-error is 3	0.51898	61
4	**if** certainty is 3 **then** relative-error is 2	0.25006	63
5	**if** certainty is 4 **then** relative-error is 1	0.10544	62
6	**if** certainty is 5 **then** relative-error is 0	0.40729	48

From Figure 4.8, we can observe that, at first, the agent has some difficulty in accurately modeling the behavior of its opponent. Especially before observation 15, errors greater than 0.5 are not uncommon. After the update before observation 15, our agent is able to quite accurately predict the next move made by its opponent.

How can these errors be explained? The explanation is quite simple: behavior that has not yet been encountered, cannot be accurately predicted. Remember that, for each era, an agent has a different expertise, and paintings are randomly created by the testbed. It is therefore possible that after 15 paintings, a certain era has not yet been encountered. In this event, our agent is not yet capable of estimating HONEST's capability to assess the

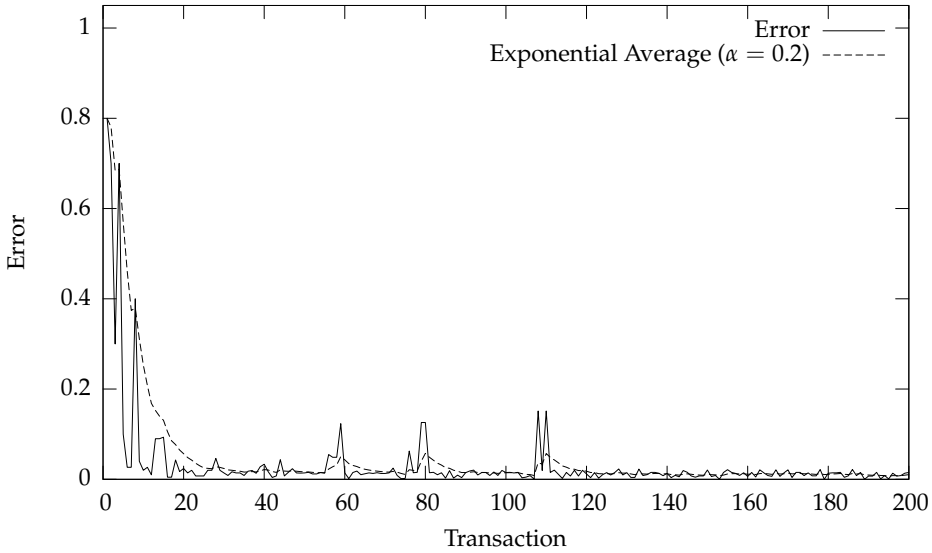


Figure 4.8: Prediction errors for HONEST

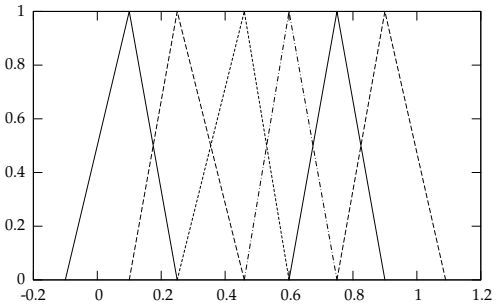
value of that painting's. Because of this, it doesn't predict anything, resulting in a prediction-error equal to the observed appraisal error.[6]

We can illustrate this phenomenon by looking at the incorrect predictions of HONEST's behavior between transaction 10 and 15. HPS listing 4.2 shows the model learned during the update before transaction 10. Compare this listing with the HPS learned after 15 transactions (listed in HPS Listing 4.3). The rule '**if** certainty is 2 **then** appraisal-error is 3' has not been learned yet, because HONEST had not yet asserted a certainty of level 1 (due to the fact that it had not yet been asked to appraise a painting from an era for which it has a certainty of level 1). Instead, the rule '**if** certainty is 3 **then** appraisal-error is 2' matches (see Figure 4.9 for partitioning of the input-spaces of variables 'certainty' and 'appraisal-error'), resulting in an error of about 0.1. This is corrected after the update after observation 15, and we obtain the model shown in listing 4.3.

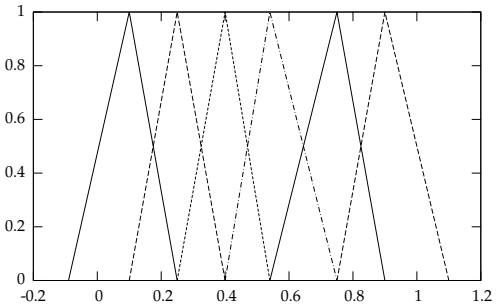
In this Listing 4.3, we also see that FURL incorrectly assumes that time has influence on HONEST's behavior. In particular, rule 7 asserts that the error at the start of the experiment is slightly higher than normal.

This phenomenon also occurs around iterations 60, 80, and 100. In all of these cases, the learned rule bases contains rules with the 'round' variable in their antecedents. Apparently, the inclusion of these kinds of rules slightly improves reduces the error on the training set, but cause an increase in prediction error. This problem is probably caused by overfit.

[6] More specifically, nothing is predicted because no rules fire, resulting in a prediction of 0.



(a) certainty



(b) relative-error

Figure 4.9: Input-space partitioning for the experiment with HONEST

HPS Listing 4.2 Model of HONEST's behavior after 10 interactions

	Rule	Credit	Matches
1	**if** certainty is 0 **then** relative-error is 5	0.01000	1
2	**if** certainty is 1 **then** relative-error is 4	0.10667	4
3	**if** certainty is 3 **then** relative-error is 2	0.00667	1
4	**if** certainty is 4 **then** relative-error is 1	0.06667	2
5	**if** certainty is 5 **then** relative-error is 0	0.04000	2

HPS Listing 4.3 Model of HONEST's behavior after 15 interactions

	Rule	Credit	Matches
1	**if** certainty is 0 **then** relative-error is 5	0.03820	4
2	**if** certainty is 1 **then** relative-error is 4	0.03006	7
3	**if** certainty is 2 **then** relative-error is 3	0.04637	6
4	**if** certainty is 3 **then** relative-error is 2	0.00000	1
5	**if** certainty is 4 **then** relative-error is 1	0.06667	2
6	**if** certainty is 5 **then** relative-error is 0	0.04000	2
7	**if** certainty is 2 **and** round is 0 **then** relative-error is 4	0.00287	4

The results depicted in Figure 4.8 do not tell the full story. In this experiment, we trained our model with data from all past transactions, to predict the next. Of course, these transactions do not necessarily cover the whole spectrum of possible transactions. To test how well the model generalizes, we compared the *predicted* appraisal error given the asserted certainty against the *actual* appraisal error for all possible transactions with HONEST.

The results are plotted in Figure 4.10. Based on this plot, we can conclude that the model shown in Listing 4.1 quite accurately describes the full extent of the actual behavior of HONEST.

4.4.3 DISHONEST

The behavior of DISHONEST was explained in Section 4.3.3.1. In essence, it is the opposite of that of HONEST. Our experimental results show exactly that. The model learned from DISHONEST's behavior tells that it asserts a completely different certainty than one would expect based on the observed error. HPS Listing 4.4, lists the model learned after 200 transactions.

Figure 4.11 shows the quality of the predictions over time. Based on the resulting model in Listing 4.4, and Figure 4.11 we can see that the learning process for HONEST and DISHONEST is very similar, if not equivalent.

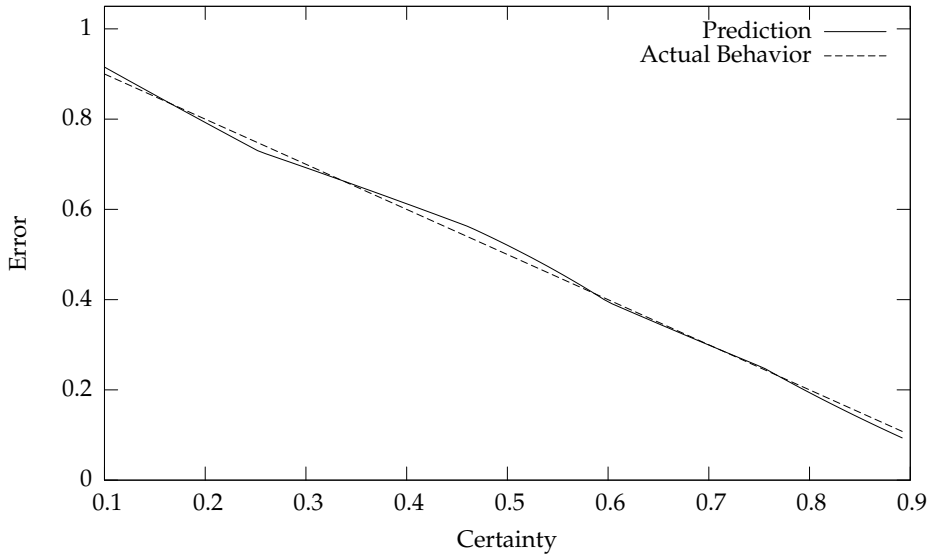


Figure 4.10: Prediction vs. Actual behavior for HONEST

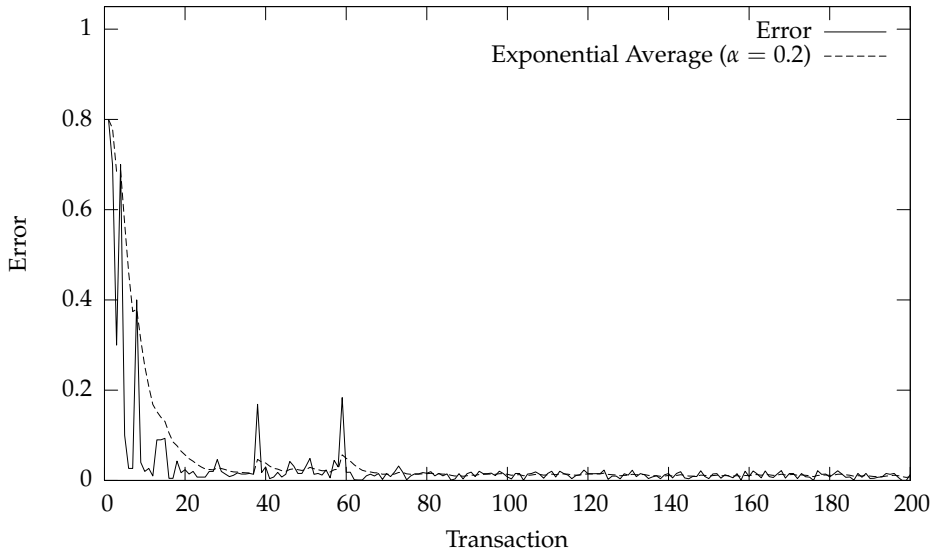


Figure 4.11: Prediction errors for DISHONEST

HPS Listing 4.4 Model of DISHONEST's behavior after 200 interactions

	Rule	Credit	Matches
1	**if** certainty is 0 **then** relative-error is 0	0.40729	48
2	**if** certainty is 1 **then** relative-error is 1	0.10544	62
3	**if** certainty is 2 **then** relative-error is 2	0.25006	63
4	**if** certainty is 3 **then** relative-error is 3	0.51898	61
5	**if** certainty is 4 **then** relative-error is 4	0.25440	56
6	**if** certainty is 5 **then** relative-error is 5	0.55590	52
7	**if** certainty is 3 **and** round is 0 **then** relative-error is 4	0.00036	4

4.4.4 RANDOM

RANDOM was discussed in Section 4.3.3.1. The main reason for pitting our agent against RANDOM in this experiment, is to make sure our approach does not identify patterns that do not exist, or at least that the produced model acknowledges a high amount of uncertainty in the patterns learned.

As can be seen from Figure 4.12, the attempt of our approach to model RANDOM is significantly less successful than similar attempts for HONEST and DISHONEST. Of course, the explanation for this failure is that no regularities or patterns can be found, and prediction is consequently impossible.

The resulting HPS after 200 transactions is shown in Listing 4.5. As we can see the rules identified by the modeling component are unusable, because they do not describe RANDOM's behavior in any way. However, in stark contrast to the experimental results obtained from runs with HONEST and DISHONEST, the credits for each rule are very high, indicating that the rules contain a large amount of error and uncertainty. Of course, this is exactly what we desire, because it fulfills our requirement that the model acknowledges the high degree of uncertainty in the rules it contains.

Note that in Figure 4.12, the running average increases somewhat from round 110 onwards. This is because FURL starts using the 'round' input feature in the base level of the HPS (just as in Listing 4.5). Consequently, most existing rules do not fire on new instances, because these instances contain a value for round that is not member of any of the existing fuzzy sets. Of course, if no rules match, the prediction is 0, resulting in higher errors. In Section 4.4.8, we discuss the special nature of the 'round' input feature in modeling behavior with FURL.

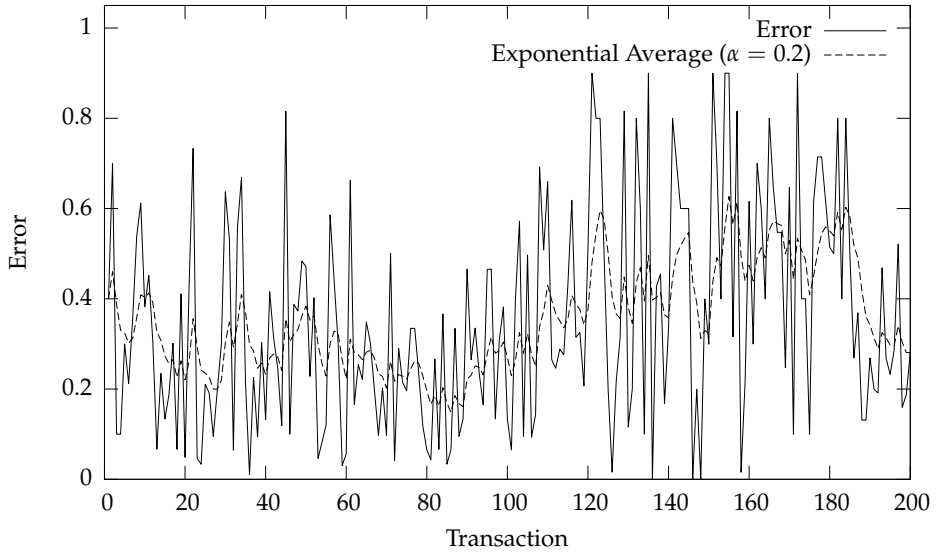


Figure 4.12: Prediction errors for RANDOM

HPS Listing 4.5 Model of RANDOM's behavior after 200 interactions

	Rule	Credit	Matches
1	**if** round is 0 **then** relative-error is 4	2.79728	10
2	**if** round is 1 **then** relative-error is 1	2.99501	30
3	**if** round is 2 **then** relative-error is 2	5.89392	60
4	**if** round is 3 **then** relative-error is 2	8.11381	70
5	**if** round is 4 **then** relative-error is 3	8.89005	80
6	**if** round is 5 **then** relative-error is 3	6.46727	70
7	**if** round is 6 **then** relative-error is 4	2.11757	30
8	**if** round is 7 **then** relative-error is 4	2.49932	10
9	**if** round is 1 **and** certainty is 4 **then** relative-error is 4	0.50848	7
10	**if** round is 2 **and** certainty is 6 **then** relative-error is 4	0.88761	14
11	**if** round is 5 **and** certainty is 1 **then** relative-error is 0	0.59890	13
12	**if** round is 6 **and** certainty is 5 **then** relative-error is 1	0.98184	11

4.4.5 NEUTRAL

Remember from Section 4.3.3.1 that NEUTRAL always asserts the same certainty, regardless of its expertise for the era of a painting. Consequently, it is impossible to predict the appraisal error made by NEUTRAL based on its asserted certainty. We therefore expect poor predictions of this agent's behavior.

In Figure 4.13 we see this expectation confirmed. The models learned in various rounds poorly predict NEUTRAL's appraisal error. The learned

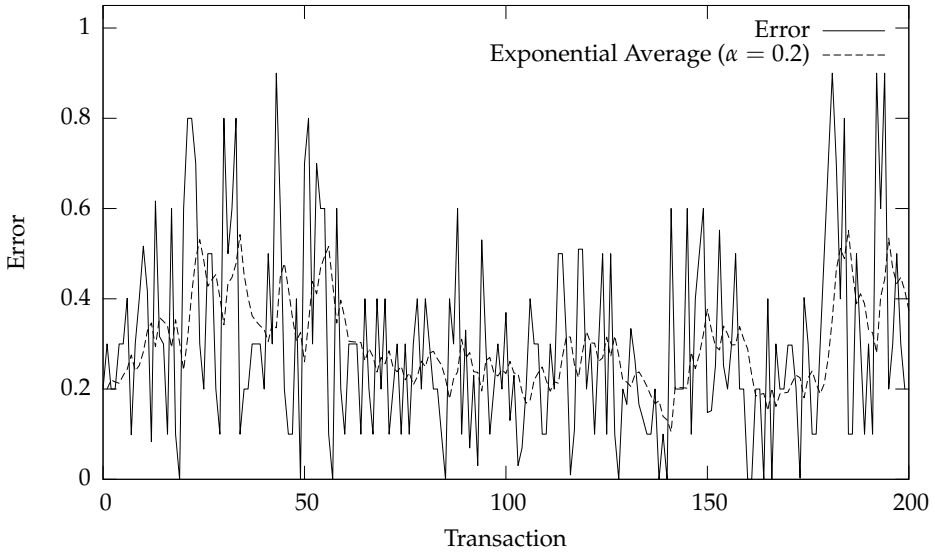


Figure 4.13: Prediction errors for NEUTRAL

model after 200 rounds is displayed in HPS Listing 4.6. Just as the models learned from the behavior of RANDOM, the rules in the model for NEUTRAL do not make any sense. The fact that NEUTRAL's actions are not predictable is reflected in the high credits of the learned rules.

Incidentally, note the varying levels of error. For example, between transaction 60 and 80, appraisal errors remain between 0.1 and 0.4, and are significantly lower than during the rest of the experiment. This is due to the fact that FURL intermittently models NEUTRAL's behavior based on its asserted certainty, instead of the current round. In Listing 4.7, the learned model after 60 transactions is shown. If we compare this listing to Listing 4.6, we see that certainty is indeed used in the base level of the HPS.[7]

We already discussed the effect of using the 'round' input feature in the model in the previous section. When used, it can cause no rule in the HPS to fire, resulting in no prediction at all. As a result, a HPS modeled based on the 'certainty' input feature can more accurately model behavior. This explains the reduced (variance of) error between transactions 60 and 80.

[7]Only a single rule is present in the base level, because the partitioning algorithm identified just one partition on the domain of certainty. Of course, this is correct, because all data points are projected on a single point on the 'certainty' axis, since NEUTRAL asserts a constant certainty.

HPS Listing 4.6 Model of NEUTRAL's behavior after 200 interactions

	Rule	Credit	Matches
1	**if** round is 0 **then** relative-error is 1	1.50000	10
2	**if** round is 1 **then** relative-error is 6	1.26562	20
3	**if** round is 2 **then** relative-error is 6	1.28812	40
4	**if** round is 3 **then** relative-error is 2	1.33287	40
5	**if** round is 3 **then** relative-error is 4	4.80208	50
6	**if** round is 4 **then** relative-error is 7	1.78518	50
7	**if** round is 5 **then** relative-error is 3	3.98060	40
8	**if** round is 5 **then** relative-error is 3	4.16502	40
9	**if** round is 6 **then** relative-error is 3	3.60000	40
10	**if** round is 6 **then** relative-error is 6	1.77500	20
11	**if** round is 7 **then** relative-error is 7	3.20000	10
12	**if** round is 2 **and** certainty is 0 **then** relative-error is 4	3.56125	40
13	**if** round is 3 **and** certainty is 0 **then** relative-error is 5	4.12405	40
14	**if** round is 5 **and** certainty is 0 **then** relative-error is 4	5.73402	50

HPS Listing 4.7 Model of NEUTRAL's behavior after 60 interactions

	Rule	Credit	Matches
1	**if** certainty is 0 **then** relative-error is 5	6.20000	60
2	**if** certainty is 0 **and** round is 0 **then** relative-error is 2	1.30000	10
3	**if** certainty is 0 **and** round is 2 **then** relative-error is 6	2.00000	10
4	**if** certainty is 0 **and** round is 6 **then** relative-error is 4	1.80000	10

4.4.6 RECIPROCAL

RECIPROCAL is the first agent that is capable of reacting to the behavior of others. It is therefore the first 'challenging' agent our approach has to compete with in. Remember that RECIPROCAL adapts its honesty (or the relation between certainty and appraisal-error in Equation 4.7 on page 72) based on the honesty of the behavior of opponents.

This requires a little different approach than we have seen so far in experiments in previous sections, because input behavior (i.e. the behavior 'fed' to RECIPROCAL) now influences the output behavior. We have to differ input as well. The experiments are therefore subdivided into two sub-experiments. In these two experiments, we vary the input behavior: honest behavior and switch behavior.

Throughout this experiment, *rs* (the retaliation-strength in Equation 4.9 on page 73) equals 5.

4.4.6.1 Honest Input Behavior

To verify that RECIPROCAL behaves as expected, we begin our experimentation by using honest input behavior. Honest behavior does exactly what the name says: it honestly asserts its expertise to its opponent. Our hypothesis (based on the specification of RECIPROCAL in Section 4.3.3.2) is that it should behave as HONEST when treated honestly. We therefore expect that the results of this experiment are similar to those of the experiment with HONEST described in Section 4.4.2.1.

In Figure 4.14 we see that is indeed the case. It is almost identical to Figure 4.8 and the resulting HPS listed in HPS Listing 4.8 confirms our hypothesis that we cannot distinguish between HONEST and (a honestly treated) RECIPROCAL in this experiment.

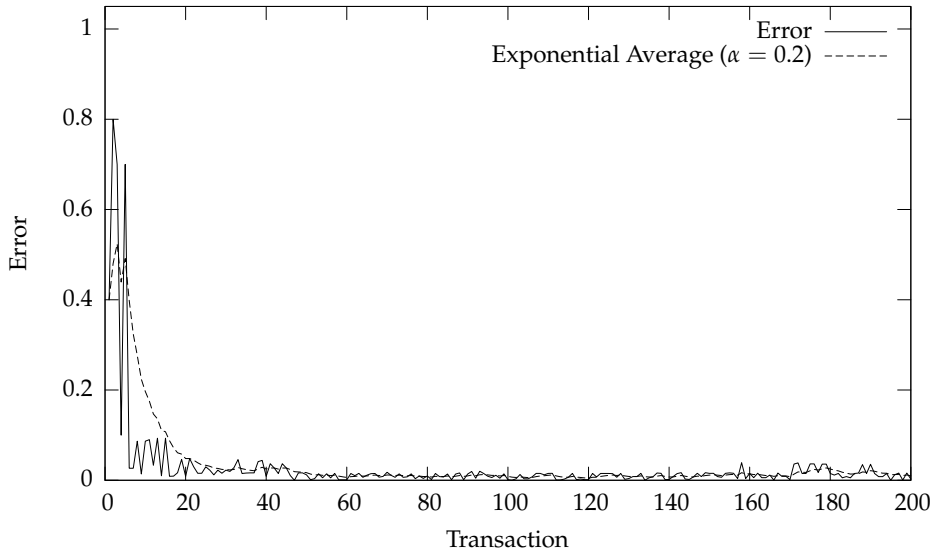


Figure 4.14: Prediction errors for RECIPROCAL with Honest input behavior

HPS Listing 4.8 Model of RECIPROCAL's behavior with Honest input behavior after 200 interactions

	Rule	Credit	Matches
1	**if** certainty is 0 **then** appraisal-error is 5	0.45287	47
2	**if** certainty is 1 **then** appraisal-error is 4	0.16079	69
3	**if** certainty is 2 **then** appraisal-error is 3	0.35663	72
4	**if** certainty is 3 **then** appraisal-error is 2	0.11930	59
5	**if** certainty is 4 **then** appraisal-error is 1	0.13328	54
6	**if** certainty is 5 **then** appraisal-error is 0	0.47491	44

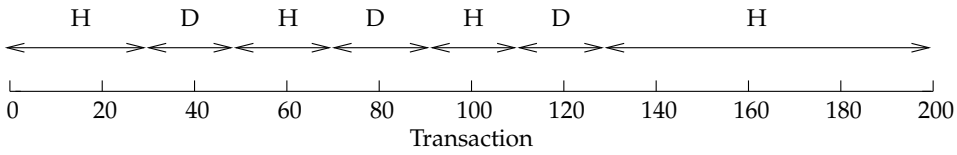


Figure 4.15: Time line for the Switch input behavior. 'H' and 'D' stands for 'Honest' and 'Dishonest' behavior respectively

4.4.6.2 Switch Input-behavior

In the second part of experimentation with RECIPROCAL we use a more complicated input behavior. The switch input behavior—as the name suggests—switches between two kinds of behavior. More specifically, it alternates between honest and dishonest at predetermined moments during the simulation (see Figure 4.15).

The resulting errors in the predictions are plotted in Figure 4.16. Looking at this figure, we immediately notice that in the first 40 transactions, the graph is very similar to Figure 4.8 for learning HONEST's behavior. This should come as no surprise, because RECIPROCAL punishes its opponent for dishonest behavior the first opportunity it gets: during the *next* round. So, although the modeling agent was dishonest in round 3 (transactions 30 to 39), it isn't punished until round 4. As we have seen in the previous experiment, when RECIPROCAL receives honest behavior from its opponent it behaves just like HONEST. Therefore, RECIPROCAL is indistinguishable from HONEST in the first 40 transactions of this experiment (not surprisingly, the model obtained after transaction 39 is identical to that of Listing 4.1).

Things get more interesting from transaction 40 onwards. Reacting to the dishonest behavior from its opponent in round 3, RECIPROCAL increases its own dishonesty using the retaliation-mechanism formalized on page 72. The results of this are clearly visible in Figure 4.16: the predictive capabilities of the learned model deteriorate immediately because this kind behavior has not been encountered yet.

In round 5 and 6 (transactions 50 to 69) RECIPROCAL is again treated honestly, and the error subsequently drops to lower levels in rounds 6 and 7 (from transaction 60 through 79).

In each subsequent alternation between both input behaviors, we observe a spike in error. Interestingly, the second group of spikes is lower than the first and third. However, our agent does not seem to actually learn RECIPROCAL's behavior, because the third spike is of the same height as the first. Let us review Listing 4.9 for an explanation for this phenomenon.

Contrary to our expectation, FURL has identified the 'round' input feature as an influence on RECIPROCAL's behavior. More specifically, rules 11, 12, and 13 match the spikes at the beginning of the experiment, and during

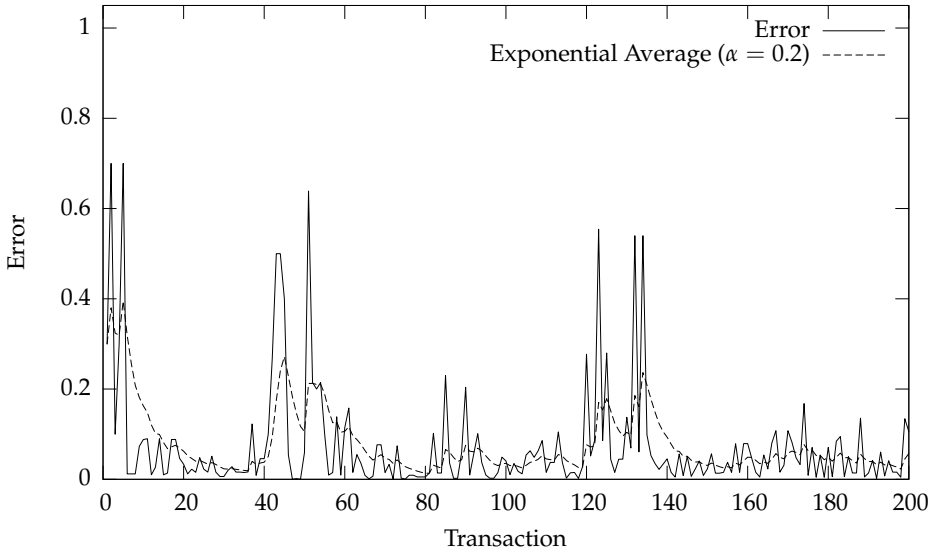


Figure 4.16: Prediction errors for RECIPROCAL with Switch input behavior using input features 'certainty', 'dishonesty', and 'round'

the first and second retaliation. Of course, this rule base was learned after the experiment was completed, but a short investigation shows that FURL also used the 'round' input feature in rule bases constructed *during* the experiment. The explanation for this lies in the fact that the dishonesty of our agent is indirectly influenced by the round-number, because of the nature of the Switch input behavior (see Figure 4.15). However, the strength of RECIPROCAL's retaliation depends on the average dishonesty of our agent in the previous round, which varies each round. A set of rules using dishonesty as the major cause of RECIPROCAL's behavior is therefore expected to be more accurate.

During the discussion of the experiments conducted with RANDOM, we already mentioned the special nature of the 'round' input feature. In Section 4.4.8 we discuss this in more detail.

We conducted the same experiment again, without using the 'round' input feature, to see whether this would improve the quality of predictions.

From Figure 4.17 we see that this is the case: contrary to the previous experiment, our agent is increasingly more capable of predicting RECIPROCAL's behavior in subsequent alternations between both input behaviors (in particular the 'punishments' in response to dishonesty of our agent).

Let us take a look at listing 4.10, which shows the model learned after 200 transactions. (For details about the partitioning of the input-spaces of all three variables, see Figure 4.18.) This model is a little harder to interpret than the models we have seen so far. To understand it more easily, let us

HPS Listing 4.9 Model of RECIPROCAL's behavior with Switch input behavior after 200 interactions using input features 'certainty', 'dishonesty', and 'round'

	Rule	Credit	Matches
1	**if** certainty is 0 **then** relative-error is 5	0.57944	52
2	**if** certainty is 1 **then** relative-error is 3	2.28214	111
3	**if** certainty is 2 **then** relative-error is 0	1.69593	121
4	**if** certainty is 3 **then** relative-error is 3	0.28645	54
5	**if** certainty is 4 **then** relative-error is 1	0.23623	18
6	**if** certainty is 5 **then** relative-error is 1	0.23952	12
7	**if** certainty is 6 **then** relative-error is 0	0.10234	6
8	**if** certainty is 7 **then** relative-error is 0	0.04543	3
9	**if** certainty is 1 **and** dishonesty is 3 **then** relative-error is 5	0.21336	9
10	**if** certainty is 1 **and** dishonesty is 5 **then** relative-error is 5	0.04342	1
11	**if** certainty is 1 **and** round is 0 **then** relative-error is 4	0.19559	8
12	**if** certainty is 1 **and** round is 2 **then** relative-error is 4	0.40966	29
13	**if** certainty is 1 **and** round is 5 **then** relative-error is 4	1.23966	44
14	**if** certainty is 2 **and** dishonesty is 3 **then** relative-error is 5	0.10581	25
15	**if** certainty is 2 **and** dishonesty is 4 **then** relative-error is 5	0.09605	17
16	**if** certainty is 2 **and** dishonesty is 5 **then** relative-error is 5	0.08032	8
17	**if** certainty is 3 **and** dishonesty is 3 **then** relative-error is 2	0.31669	26
18	**if** certainty is 3 **and** dishonesty is 5 **then** relative-error is 4	0.10928	9
19	**if** certainty is 4 **and** dishonesty is 3 **then** relative-error is 0	0.14957	14

first examine the quality of the prediction in Figure 4.19.

Compare Figure 4.19 to Figure 4.5, which shows the actual behavior of RECIPROCAL. The most prominent difference between these figures, is the surface above the area described by $certainty \geq 1 + 5 \times dishonesty$ (this is the case when $appraisal\text{-}error = 0$ in Equation 4.9 on page 73). Ideally, this surface should be in the plane $appraisal\text{-}error = 0$. However, we must realize that tuples in the set $S = \{(certainty, dishonesty) \mid certainty \geq 1 + 5 \times dishonesty\}$ do not occur in the data set, because this would mean $expertise(p) > expertise_{max}$, which is absurd (see Equation 4.8 on page 72).

In other words, for tuples in the set S, the function $appraisal\text{-}error(p)$ is undefined. Clearly, FURL is not capable of correctly extrapolating this function beyond its domain. Of course, this should not be a problem. Ignoring the tuples in set S, the prediction-error plotted in Figure 4.20 are within the acceptable bounds.

First, let us examine the base-level rules. These describe the overall relation between certainty and appraisal-error. Simply put, these rules assert that when the certainty increases, the error drops irrespective of dishonesty, which corresponds with RECIPROCAL's specification.

Rule 1 states that very low certainty always precedes a an error of the highest possible level (see Figure 4.18). Rule 1 does not have any exceptions. This is correct, because regardless of dishonesty, RECIPROCAL always has a high appraisal error after asserting a very low certainty.

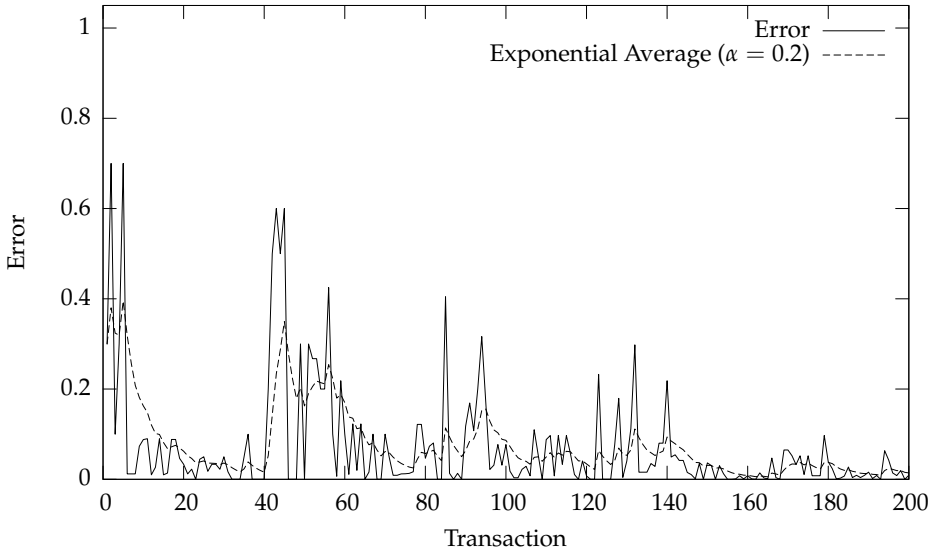


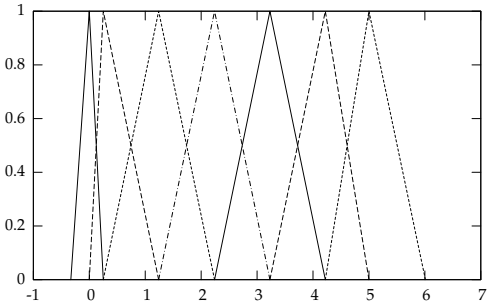
Figure 4.17: Prediction errors for RECIPROCAL with Switch input behavior using the input features 'certainty' and 'dishonesty'

Rules 8 to 12, which are exceptions to rule 2 state that the error is slightly lower than normal, when dishonesty is equal or higher than level 2. Rule 2 is thus only applicable at lower levels of dishonesty. At these levels of dishonesty, we expect a lower error from RECIPROCAL, which is exactly what Rule 2 asserts.

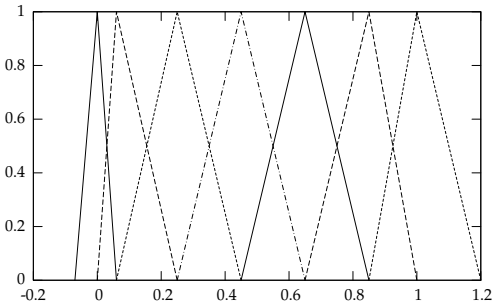
Rules 13 to 19, which are exceptions to rule 3. Rules 13, 14, and 15 have identical antecedents, and different consequents, which is counterintuitive. These rules were identified in iteration 24, 10, and 3 respectively, which leads us to assume that rules 13 and 14 were introduced to counter the effect of 'neighboring' rules identified after iteration 3 that assert higher levels of error (for example rules 6, 7, and 20). These rules (13 to 19) all reduce the error in the base case for lower levels of dishonesty. For higher levels of dishonesty, only the base-level rule 3 applicable, resulting in a higher error, as expected.

Rules 20 to 26 are exceptions to rule 4. The function of most of these rules is similar to that of rules we have seen so far. Rule 24 and 26 are different. Both rules assert a very high error at a high level of dishonesty, with a medium certainty. This is not what we expect, and it is not surprising that these rules are responsible for the 'spike' at $(certainty, dishonesty) = (2, 0.9)$. The reason behind this anomaly is the same as with rules 13, 14, and 15: the rules were identified early on, and interfere with newer rules.

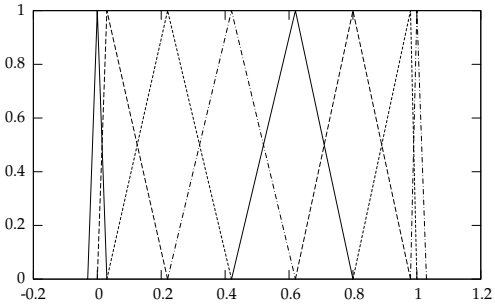
Rules 27, 28, and 29 are exceptions to rule 5. Compared to similar rules for a lower level of certainty (rules 21 to 23), they assert the same level of



(a) certainty



(b) dishonesty



(c) appraisal-error

Figure 4.18: Input-space partitioning for the experiment with RECIPROCAL

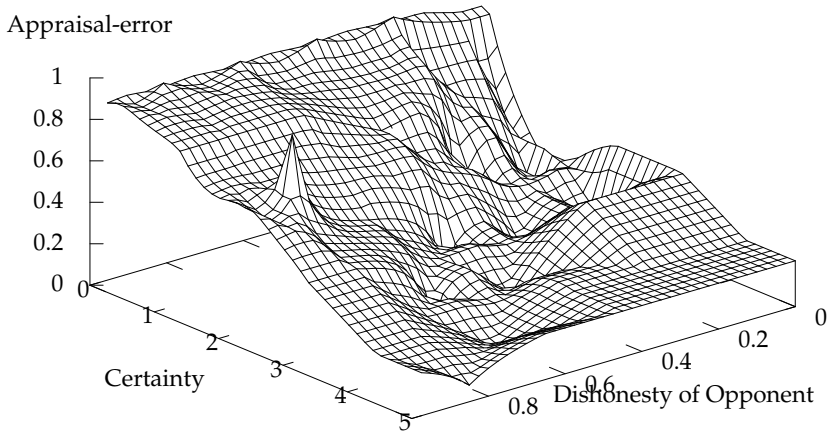


Figure 4.19: Model predictions for RECIPROCAL with Switch input behavior (see Figure 4.5 for the actual behavior)

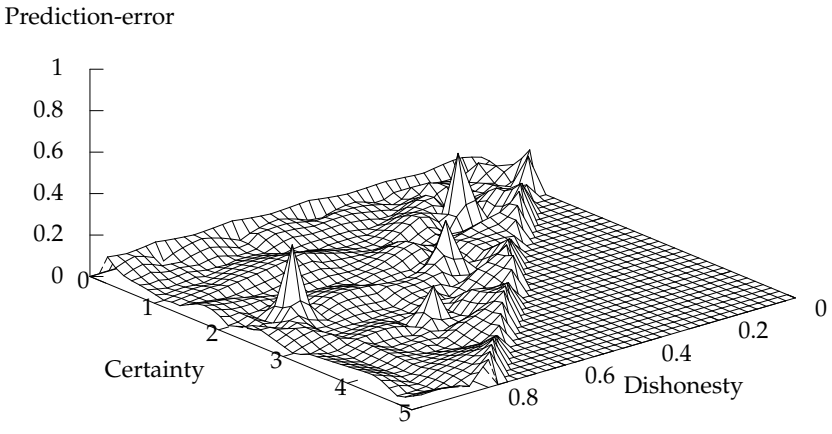


Figure 4.20: Model predictions errors for RECIPROCAL with Switch input behavior (ignoring impossible combinations of certainty and dishonesty)

HPS Listing 4.10 Model of RECIPROCAL's behavior with Switch input behavior after 200 interactions using input features 'certainty' and 'dishonesty'

	Rule	Credit	Matches
1	**if** certainty is 0 **then** appraisal-error is 7	0.02155	11
2	**if** certainty is 1 **then** appraisal-error is 5	0.14600	22
3	**if** certainty is 2 **then** appraisal-error is 5	0.35215	41
4	**if** certainty is 3 **then** appraisal-error is 3	0.27895	34
5	**if** certainty is 4 **then** appraisal-error is 3	0.24158	26
6	**if** certainty is 5 **then** appraisal-error is 2	0.20323	15
7	**if** certainty is 6 **then** appraisal-error is 2	0.15088	7
8	**if** certainty is 1 **and** dishonesty is 2 **then** appraisal-error is 6	0.05426	8
9	**if** certainty is 1 **and** dishonesty is 3 **then** appraisal-error is 6	0.05719	8
10	**if** certainty is 1 **and** dishonesty is 4 **then** appraisal-error is 6	0.04682	8
11	**if** certainty is 1 **and** dishonesty is 5 **then** appraisal-error is 6	0.03334	6
12	**if** certainty is 1 **and** dishonesty is 6 **then** appraisal-error is 6	0.04008	4
13	**if** certainty is 2 **and** dishonesty is 0 **then** appraisal-error is 0	0.06751	2
14	**if** certainty is 2 **and** dishonesty is 0 **then** appraisal-error is 1	0.06751	2
15	**if** certainty is 2 **and** dishonesty is 0 **then** appraisal-error is 2	0.06751	2
16	**if** certainty is 2 **and** dishonesty is 1 **then** appraisal-error is 0	0.07943	7
17	**if** certainty is 2 **and** dishonesty is 1 **then** appraisal-error is 1	0.07943	7
18	**if** certainty is 2 **and** dishonesty is 2 **then** appraisal-error is 0	0.09338	15
19	**if** certainty is 2 **and** dishonesty is 3 **then** appraisal-error is 4	0.08041	16
20	**if** certainty is 3 **and** dishonesty is 1 **then** appraisal-error is 1	0.03234	3
21	**if** certainty is 3 **and** dishonesty is 2 **then** appraisal-error is 0	0.13132	10
22	**if** certainty is 3 **and** dishonesty is 2 **then** appraisal-error is 1	0.13132	10
23	**if** certainty is 3 **and** dishonesty is 3 **then** appraisal-error is 0	0.10062	15
24	**if** certainty is 3 **and** dishonesty is 5 **then** appraisal-error is 7	0.07153	12
25	**if** certainty is 3 **and** dishonesty is 6 **then** appraisal-error is 0	0.04550	8
26	**if** certainty is 3 **and** dishonesty is 6 **then** appraisal-error is 5	0.04550	8
27	**if** certainty is 4 **and** dishonesty is 3 **then** appraisal-error is 0	0.12910	10
28	**if** certainty is 4 **and** dishonesty is 3 **then** appraisal-error is 1	0.12910	10
29	**if** certainty is 4 **and** dishonesty is 4 **then** appraisal-error is 0	0.08275	15
30	**if** certainty is 5 **and** dishonesty is 4 **then** appraisal-error is 0	0.09670	9
31	**if** certainty is 5 **and** dishonesty is 4 **then** appraisal-error is 1	0.09670	9
32	**if** certainty is 6 **and** dishonesty is 5 **then** appraisal-error is 1	0.16347	5

error at a higher level of dishonesty. This is exactly according to the specification of RECIPROCAL: if dishonesty increases, certainty must increase also to get a constant level of error.

Rules 30, 31, and 32 (exceptions to rules 6 and 7) are similar to exceptions to rule 5.

The final observation we would like to make in this experiment, is that we can clearly see from Figure 4.19 that exceptions reshape the surface created by the base-level rules by making 'dents' and 'spikes'. The less partitions we use, the less accurate the resulting model becomes, but we also need less exceptions, because the dents and spikes are larger. This might be preferable when we have a small number of data-points, and need im-

proved generalization capabilities from our learned model. This trade-off can be subject to further research.

4.4.7 SWITCH

The SWITCH agent that switches between the behavior of HONEST and DISHONEST was discussed in 4.3.3.3. The prediction errors for each transaction are plotted in Figure 4.21. As expected, in the first 10 rounds (transactions 0 to 99), SWITCH agent's behavior is reasonably approximated by the model. Of course, this is explained by the fact that, until then, SWITCH is indistinguishable from HONEST. In round 10, however, the quality of predictions is very low. SWITCH starts to behave exactly opposite to what is expected. This explains the first couple of 'spikes' in Figure 4.21. After 10 to 15 transactions, the model has adapted itself to the new behavior, and we see a subsequent drop in error. The average error then stabilizes around 0.1.

The model listed in Listing 4.11 shows that the base-rules, together with the exceptions explain SWITCH's behavior. Note that the base-rules individually are not very informative without also considering their exceptions. Some base-rules are similar to their counterparts in the model of HONEST (see Listing 4.1) and others to their counterparts in the model of DISHONEST (Listing 4.4). Examples of the former are rules 1, 5, and 6. Examples of the latter are rules 2 and 3.

Combining them with their exceptions, however, we see that these rules indeed differentiate between the first half and the last half rounds of the simulation.

In Figure 4.22, the overall prediction of the behavior SWITCH is shown. Compare this figure to figure 4.6. The differences between these figures are most prominent at the threshold of switching between honest and dishonest behavior. Figure 4.23 in which the prediction-errors are plotted shows this even more clearly.

From these figures, we can conclude that fuzzy rules are not very capable of approximating discontinuous functions. The way conflicting rules are handled (see Section 3.4.1), two rules asserting opposite appraisal-errors that fire simultaneously cause a gradual transition, where a abrupt transition is desired.

Despite these errors at the threshold, Figure 4.23 shows that the learned model is at the very least usable in the decision making phase.

The results from this experiment raises the question: "why not break the transaction history in two halves to simplify modeling?" This is a valid question, because this would make possible the modeling of SWITCH first as HONEST and later as DISHONEST. As we have seen in Sections 4.4.2.1, and 4.4.3 this can be done with high accuracy. Despite this, there is a reason for still choosing to consider the whole transaction history, instead of only a part of it.

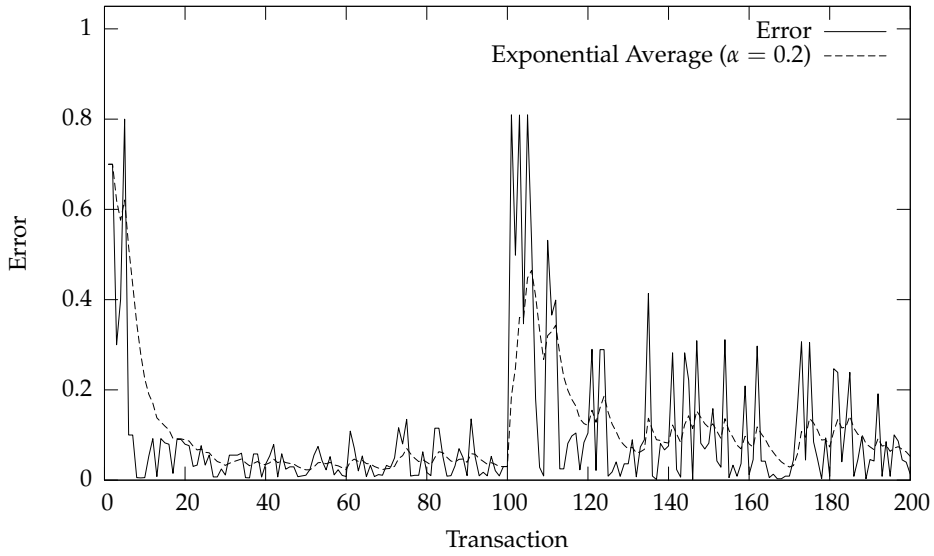


Figure 4.21: Prediction errors for SWITCH

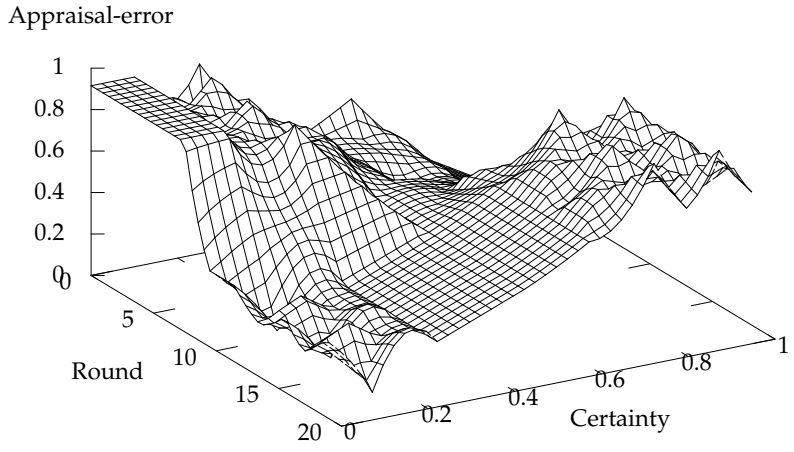


Figure 4.22: Model predictions for SWITCH

HPS Listing 4.11 Model of SWITCH's behavior after 200 interactions

	Rule	Credit	Matches
1	**if** certainty is 0 **then** appraisal-error is 5	1.20513	40
2	**if** certainty is 1 **then** appraisal-error is 1	1.15313	60
3	**if** certainty is 2 **then** appraisal-error is 2	0.75262	60
4	**if** certainty is 3 **then** appraisal-error is 3	0.72907	60
5	**if** certainty is 4 **then** appraisal-error is 1	0.93528	60
6	**if** certainty is 5 **then** appraisal-error is 0	1.05482	40
7	**if** certainty is 0 **and** round is 4 **then** appraisal-error is 0	0.81572	16
8	**if** certainty is 0 **and** round is 5 **then** appraisal-error is 0	0.47079	14
9	**if** certainty is 0 **and** round is 6 **then** appraisal-error is 0	0.20304	6
10	**if** certainty is 0 **and** round is 7 **then** appraisal-error is 0	0.06488	2
11	**if** certainty is 1 **and** round is 0 **then** appraisal-error is 5	0.08646	3
12	**if** certainty is 1 **and** round is 1 **then** appraisal-error is 5	0.12044	9
13	**if** certainty is 1 **and** round is 2 **then** appraisal-error is 5	0.12926	18
14	**if** certainty is 1 **and** round is 3 **then** appraisal-error is 5	0.53287	21
15	**if** certainty is 2 **and** round is 2 **then** appraisal-error is 3	0.15236	18
16	**if** certainty is 3 **and** round is 1 **then** appraisal-error is 2	0.12041	9
17	**if** certainty is 3 **and** round is 2 **then** appraisal-error is 1	0.25882	18
18	**if** certainty is 3 **and** round is 3 **then** appraisal-error is 2	0.34837	21
19	**if** certainty is 4 **and** round is 4 **then** appraisal-error is 5	0.34529	24
20	**if** certainty is 4 **and** round is 5 **then** appraisal-error is 5	0.18626	21
21	**if** certainty is 4 **and** round is 6 **then** appraisal-error is 5	0.07570	9
22	**if** certainty is 4 **and** round is 7 **then** appraisal-error is 5	0.05242	3
23	**if** certainty is 5 **and** round is 4 **then** appraisal-error is 5	0.71315	16
24	**if** certainty is 5 **and** round is 5 **then** appraisal-error is 5	0.38554	14
25	**if** certainty is 5 **and** round is 6 **then** appraisal-error is 5	0.16754	6
26	**if** certainty is 5 **and** round is 7 **then** appraisal-error is 5	0.02908	2

It can best be explained by considering RECIPROCAL, in particular the experiment discussed in Section 4.4.6.2. Should we decide to discard a part of the transaction history each time an event (such as a sudden increase in error) occurs, the model of RECIPROCAL would alternate between that of an HONEST and a less-than-honest agent. Each time this happens, we lose important information about RECIPROCAL's behavior. This way we never obtain a suitable model. The difference with SWITCH's behavior, is that the change in behavior depends on the round-number, which increases monotonically as the experiment progresses. This means that after a certain time, behavior can irreversibly change, and won't occur anymore. So, in order to decide whether or not a part of the data set can be discarded, the opponent modeling at least has to know which of the transaction-properties monotonically increase (like 'round'). Even then, it is not trivial to determine whether has "fundamentally" changed (which is the case with SWITCH), or that the change is inherent to the behavior of the agent in question (which is the case with RECIPROCAL). These questions could therefore be subject to further research.

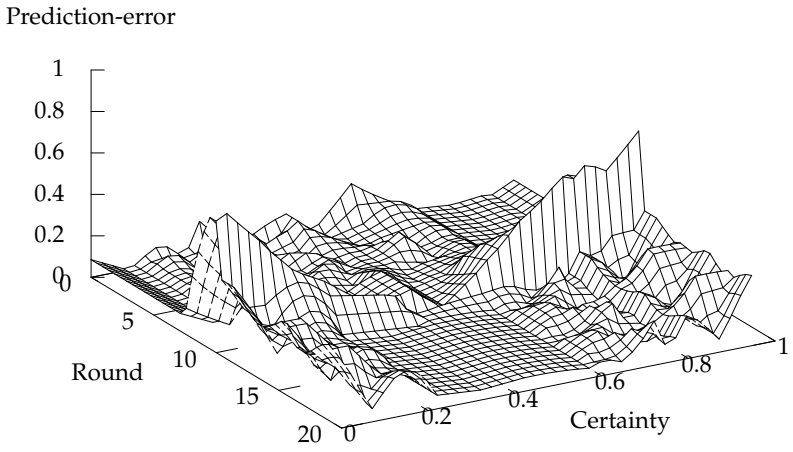


Figure 4.23: Model prediction-errors for SWITCH

4.4.8 Discussion: the 'round' input feature

The subject of the previous discussion is closely related to the special nature of the 'round' input feature we encountered during the experiments conducted with RANDOM, NEUTRAL, and RECIPROCAL. In all three experiments, we saw that rules using this input feature fire only on instances received during a certain time interval. Consequently, their usefulness to predict future instances is limited at the most.

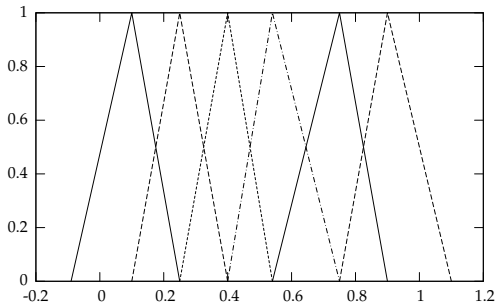
Moreover, using this input feature in FURL, we risk the possibility of ending up with a description of the past, instead of a tool to predict the future. An example of this is found in Listing 4.9, where rules 11, 12 and 13 do exactly this: describe the past. This explains the quite poor performance of predicting RECIPROCAL's behavior.

The case of SWITCH is a totally different one. Here, FURL uses the 'round' feature to discard a part of the transaction history. SWITCH fundamentally changes its behavior, and apparently, FURL sees no other possibility than to model this agent in a different way.

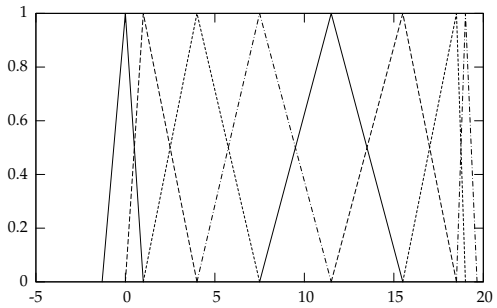
So, on the one hand, we prefer not to include 'round' in the rule base because it could interfere with our main goal of predicting behavior, but on the other hand, if no other possibility exists, allow FURL to include 'round' to indicate a fundamental change in behavior.

With this problem in mind, we explored two possible solutions for this problem.

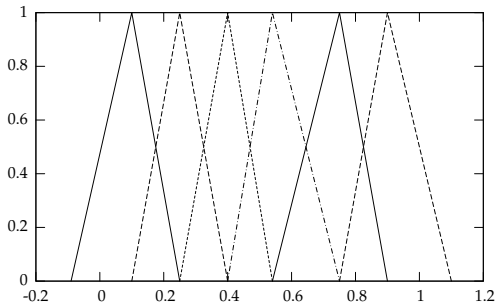
The first solution is to artificially increase the error for rules containing 'round' in their antecedents. This way, we assign a penalty to these rule



(a) certainty



(b) round



(c) relative-error

Figure 4.24: Input-space partitioning for the experiment with SWITCH

bases, and force FURL to look for other options.

A second possible solution is to look for input features that are statistically correlated with 'round'. In the experiment conducted with RECIPROCAL, we saw an example of such a correlation: the value 'dishonesty' was indirectly linked with 'round' by means of the way in which our agent changed behavior (see Figure 4.15). If such a correlation can be found, substitute 'round' for another feature. While this may result in a rule base with a higher global error, its rules play at least some role in predicting new behavior, instead of describing the past.

4.4.9 Evaluation

This concludes our discussion of experiments with the opponent modeling part of our approach. We saw how different agents were modeled with a HPS rule base using FURL, and how well these models predicted both future transactions and behavior that was not encountered during the experiment.

At this point, we can evaluate the results using the criteria formulated in Section 4.2.

Accurate For the unresponsive agents (HONEST, DISHONEST, RANDOM and NEUTRAL), our conclusion can be straightforward. The difference between the calculated model and the actual behavior of these agents is small. More specifically, the difference between the predicted and actual appraisal for HONEST and DISHONEST after a small number of transaction is negligible. For RANDOM and NEUTRAL, the prediction errors are high. This was expected for agents whose behavior is unpredictable. The models for these agents are nevertheless accurate, because the fact that the agents are unpredictable is reflected by the high credits assigned to the rules in the model. These high credits signal that the rules are highly uncertain.

The more complex agents (RECIPROCAL and SWITCH) proved to be more difficult to model. This is mainly due to changes in their behavior during the experiment. Judging From Figures 4.20 and 4.23, however, we can conclude that at the *end* of the simulation, FURL is more capable of modeling the behavior of RECIPROCAL than that of SWITCH. The reason for the quite poor results of FURL for SWITCH in a number of cases have been addressed in Section 4.4.7, and deserve further research (for example, FURL's difficulty modeling discontinuous functions).

Adaptive In the simulations with SWITCH and RECIPROCAL, we have seen how opponent modeling adapts to newly encountered behavior. As expected, each time these agents change behavior (or expose a new aspect of their behavior), the quality of the prediction diminishes. This cannot be

prevented. More important is how quickly the modeling algorithm detects the new behavior, refines the model, and is again capable of accurately predicting the agent's behavior.

As we have seen, the algorithm was more capable of detecting change in the behavior of RECIPROCAL, than it was for SWITCH. During a round in which RECIPROCAL retaliates, the model's predictions are clearly less accurate than in rounds in which RECIPROCAL act honestly. During these retaliations, the model does not quickly adapt to RECIPROCAL's behavior. Over time, however, the retaliations are more accurately predicted.

The model for SWITCH adapts after SWITCH's attempt to deceive our agent by altering its behavior. This adaptation takes a number of transactions. After this, the prediction errors do not return to their original levels, but are lower than at the moment of SWITCH's deception.

Quickly Converging The models for the predictable agents converge after a small number of transactions. After about 5 transactions, the models for newly encountered agents were found. As discussed above, adapting to changes proved more difficult.

Comprehensibility of the model For the simple agents, the models consisted of just a few rules. Especially for the HONEST and DISHONEST agents, the models are easily understandable. The models of NEUTRAL and RANDOM are less obvious. The rules are nonsensical, and special attention must be paid to the fact that the rules are uncertain to understand that the behavior of these agents is unpredictable.

Later on, we paid a lot of attention to explaining the function of individual rules of the models describing the RECIPROCAL and SWITCH agents. On the one hand, saw that there existed a correspondence between these rules, and the behavior of these agents. This alone is a major benefit compared to algorithms that use implicit knowledge formats (such as neural networks). On the other hand, the rule bases contain a lot of rules. Together, they state a simple fact. For example, in the case of SWITCH, they state: 'SWITCH is honest at first, but then it suddenly becomes dishonest'. Perhaps this is inherent to rule bases, but we believe this deserves further attention (see Section 5.2).

4.5 Decision making

In the previous section, we discussed experimentation with the opponent modeling component of our approach. In this section, we shift our attention to the decision making component. The aim of the experiments in this section is to illustrate how a opponent model is used to generate arguments that support delegation decisions.

4.5.1 Setup

In the scenarios that follow, we study the decision making process of our agent while in competition with both HONEST and RECIPROCAL. Because our focus is solely on decision making, we assume that the opponent modeling phase has obtained the opponent models listed in Listings 4.1 and 4.10. For clarity, both are repeated in Listings 4.12 and 4.13. In these listings, the FURL credits have been replaced by a measure of plausibility. Remember from Section 3.6.3, that the plausibility measures are obtained from these credits using Equation 3.19 on page 57.

The scenarios in this section correspond with the possible two roles in an opinion transaction in the ART Testbed: an agent can be the *provider* of an opinion, and the *requester*. In both roles, the agent has a different set of goals.

4.5.2 Results

4.5.2.1 Scenario 1: Requester Role

This scenario describes a single transaction, in which our agent has to appraise a painting at the request of one of its clients. Because it is unable to appraise the painting itself, it has to consult other agents, in this case HONEST and RECIPROCAL. For each agent, it searches for arguments to support the decision to delegate. The strengths of these arguments are used to determine to which extent the other agents influence the final appraisal (see Section 4.3.1.2).

HPS Listing 4.12 Model of HONEST's behavior after 200 interactions

	Rule	Plausibility
1	**if** certainty is 0 **then** relative-error is 5	0.00381
2	**if** certainty is 1 **then** relative-error is 4	0.00832
3	**if** certainty is 2 **then** relative-error is 3	0.00408
4	**if** certainty is 3 **then** relative-error is 2	0.00847
5	**if** certainty is 4 **then** relative-error is 1	0.02008
6	**if** certainty is 5 **then** relative-error is 0	0.00520

Goals Because it is in our agent's interest to appraise its clients painting as accurately as possible (in order to have them return in subsequent rounds), it has a single goal g_1 = (appraisal-error is acceptable, 1). *Acceptable* is a fuzzy set, which membership function is depicted in Figure 4.25. Put differently, goal g_1 states that our agent favors low appraisal errors from its opponents. So, in this particular transaction, our agent tries to find out from which agent it can get the most accurate appraisal.

HPS Listing 4.13 Model of RECIPROCAL's behavior after 200 interactions

	Rule	Plausibility
1	**if** certainty is 0 **then** appraisal-error is 7	0.09824
2	**if** certainty is 1 **then** appraisal-error is 5	0.01450
3	**if** certainty is 2 **then** appraisal-error is 5	0.00601
4	**if** certainty is 3 **then** appraisal-error is 3	0.00759
5	**if** certainty is 4 **then** appraisal-error is 3	0.00876
6	**if** certainty is 5 **then** appraisal-error is 2	0.01042
7	**if** certainty is 6 **then** appraisal-error is 2	0.01403
8	**if** certainty is 1 **and** dishonesty is 2 **then** appraisal-error is 6	0.03902
9	**if** certainty is 1 **and** dishonesty is 3 **then** appraisal-error is 6	0.03702
10	**if** certainty is 1 **and** dishonesty is 4 **then** appraisal-error is 6	0.04522
11	**if** certainty is 1 **and** dishonesty is 5 **then** appraisal-error is 6	0.06350
12	**if** certainty is 1 **and** dishonesty is 6 **then** appraisal-error is 6	0.05282
13	**if** certainty is 2 **and** dishonesty is 0 **then** appraisal-error is 0	0.03136
14	**if** certainty is 2 **and** dishonesty is 0 **then** appraisal-error is 1	0.03136
15	**if** certainty is 2 **and** dishonesty is 0 **then** appraisal-error is 2	0.03136
16	**if** certainty is 2 **and** dishonesty is 1 **then** appraisal-error is 0	0.02665
17	**if** certainty is 2 **and** dishonesty is 1 **then** appraisal-error is 1	0.02665
18	**if** certainty is 2 **and** dishonesty is 2 **then** appraisal-error is 0	0.02267
19	**if** certainty is 2 **and** dishonesty is 3 **then** appraisal-error is 4	0.02633
20	**if** certainty is 3 **and** dishonesty is 1 **then** appraisal-error is 1	0.06546
21	**if** certainty is 3 **and** dishonesty is 2 **then** appraisal-error is 0	0.01612
22	**if** certainty is 3 **and** dishonesty is 2 **then** appraisal-error is 1	0.01612
23	**if** certainty is 3 **and** dishonesty is 3 **then** appraisal-error is 0	0.02104
24	**if** certainty is 3 **and** dishonesty is 5 **then** appraisal-error is 7	0.02960
25	**if** certainty is 3 **and** dishonesty is 6 **then** appraisal-error is 0	0.04653
26	**if** certainty is 3 **and** dishonesty is 6 **then** appraisal-error is 5	0.04653
27	**if** certainty is 4 **and** dishonesty is 3 **then** appraisal-error is 0	0.01640
28	**if** certainty is 4 **and** dishonesty is 3 **then** appraisal-error is 1	0.01640
29	**if** certainty is 4 **and** dishonesty is 4 **then** appraisal-error is 0	0.02558
30	**if** certainty is 5 **and** dishonesty is 4 **then** appraisal-error is 0	0.02189
31	**if** certainty is 5 **and** dishonesty is 4 **then** appraisal-error is 1	0.02189
32	**if** certainty is 6 **and** dishonesty is 5 **then** appraisal-error is 1	0.01295

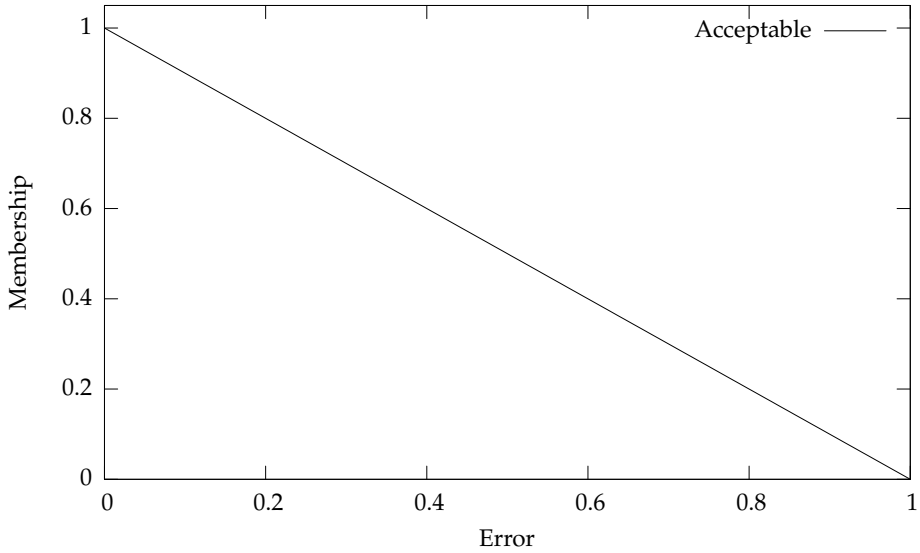


Figure 4.25: The *acceptable* set used to evaluate the acceptability of a received appraisal-error

Observations After our agent initiated a opinion transaction with both agents, they respond with their certainty. HONEST asserts a certainty of 0.25, while RECIPROCAL replies that it can appraise the painting with 3.73 certainty. In the previous round, our agent has been dishonest towards RECIPROCAL with 0.33.[8] (For the fuzzification of these values, see Figures 4.9 and 4.18.)

Decisions As said before, in this transaction, our agent can request an appraisal from two opponents. Consequently, it must consider two possible decisions: d_1: let HONEST determine the appraisal for its client, or d_2: let RECIPROCAL determine the appraisal. Of course, these decisions are not mutually exclusive. For example, our agent can decide to delegate the appraisal task to both agents equally. In that case, the final appraisal is the average over both agent's appraisals.

For HONEST, we expect a high error, because its asserted certainty is quite low (HONEST uses certainties between 0 and 1, the latter meaning that it is very capable). On the other hand, RECIPROCAL's certainty is very high, but our agent has to take its own dishonesty towards RECIPROCAL into account. The opponent model has to decide what the effect of this will be on RECIPROCAL's behavior.

[8]Note that no objective measure of certainty exists. For example, a certainty of 0.1 for one agent can lead to the same appraisal as a certainty of 0.2 for the second. See Section 4.3.3.1, and 4.3.3.2 for details about how these agents calculate their certainties.

Knowledge	Match	Plausibility
certainty is 1	100%	
if certainty is 1 **then** relative-error is 4	100%	0.00832
relative-error is 0.75	100%	

Table 4.4a: The support for Argument A_1

Goal	Match	Preference
g	0.25	1

Table 4.4b: The consequences of Argument A_1

Using these goals, observations, and decisions, our agent generates two arguments. The first argument A_1 supports decision d_1, the second argument A_2 supports decision d_2.

Decision for HONEST Remember from Definition 3 (on page 50) that an argument consists of three parts: support, consequences and conclusion.

The support of the argument is a subset of the knowledge base of the agent, and consists of knowledge used to predict the consequences of the decision under consideration. The support of A_1 consists of parts of the opponent model of HONEST relevant to this particular transaction. This is summarized in Table 4.4a.

The *consequences* of A_1 relate to the desirability of the consequences of decision d_1 in terms of the agent's goals. For an asserted certainty of 0.25, a single rule of the opponent model fires, and predicts an appraisal error of level 4 (corresponding to 0.75). Based on this, we can determine the utility in terms of goal g_1 (see Table 4.4b). It turns out that an error of 0.75 is only 25% member of the set *acceptable*.

Using the information from Tables 4.4a and 4.4b, we can now calculate the *Level* and *Weight* of argument A_1 (see Equations 3.16 on page 53, and 3.18 on page 54). Table 4.4c lists the steps for this calculation.

Our agent can now determine the the strength of the argument for HONEST: $0.00832 \times 0.25 = 0.00208$ (see Definition 7).

Decision for RECIPROCAL Next, it performs the same steps for RECIPROCAL. Argument A_2 supports the decision to delegate the task of appraising

Property	Calculation	Result
$Level_P(A_1)$	1×0.00832	0.00832
$Weight_P(A_1)$	1×0.25	0.25

Table 4.4c: The *Level* and *Weight* calculations of Argument A_1

Knowledge	Match	Plausibility
certainty is 4	50%	
certainty is 5	50%	
dishonesty is 3	40%	
if certainty is 4 **then** appraisal-error is 3	50%	0.00876
if certainty is 5 **then** appraisal-error is 2	50%	0.01042
if certainty is 4 **and** dishonesty is 3 **then** appraisal-error is 0	40%	0.01640
if certainty is 4 **and** dishonesty is 3 **then** appraisal-error is 1	40%	0.01640

Table 4.5a: The support for Argument A_2

Goal	Match	Preference
g_1	0.75	1

Table 4.5b: The consequences of Argument A_2

the painting to this agent. For determining the support and consequences of this argument, we follow the same procedure as above. They are summarized in Table 4.5a and 4.5b respectively. This time, four rules fire for the information RECIPROCAL provided. We can see that the appraisal error should be somewhere between level 0 and 3. Using the formulas in Equation 3.2 (see page 33), and defuzzifying the thusly obtained fuzzy set, we end up with a predicted appraisal error of 0.25. Consequently, goal g_1 is satisfied for 75%. Table 4.5c shows the calculation of the *Level* and *Weight* of this argument. Based on these measures, we now calculate the strength of the argument as follows: $0.00438 \times 0.75 = 0.00329$.

Concluding In the final step, our agent compares the strengths of both arguments to obtain the delegation weights towards both agents. This is done in Table 4.6. As we can see, RECIPROCAL is assigned 61% of the task. Apparently, our agents favors a low appraisal error, and more or less takes the reduced confidence of the knowledge of RECIPROCAL's behavior for granted.

In this scenario, we have seen that our agent had to choose between an agent for which a reliable opponent model is available (HONEST) and an agent that probably provides a more accurate appraisal (RECIPROCAL). The strengths of the arguments supporting both decision reflect this trade off. In the end, the lower predicted appraisal error for RECIPROCAL proved

Property	Calculation	Result
$Level_P(A_2)$	0.5×0.00876	0.00438
$Weight_P(A_2)$	1×0.75	0.75

Table 4.5c: The *Level* and *Weight* calculations of Argument A_2

Agent	Level	Weight	Strength	Delegation weight
HONEST	0.00832	0.25	0.00208	0.39
RECIPROCAL	0.00438	0.75	0.00329	0.61

Table 4.6: The delegation weights for HONEST and RECIPROCAL in scenario 1.

to be decisive. Consequently, it got the majority of the appraisal task.

4.5.2.2 Scenario 2: Provider Role

In this scenario our agent has the role of the opinion provider. It is slightly different from the scenario discussed in the previous section: we add a new goal, and apply the decision making procedure to the opinions generated by *our agent*, instead of its opponents. The new goal, called g_2, essentially encourages our agent to be as deceptive as possible towards other agents (by overstating its certainty of correctly appraising a painting). Achieving goal g_2, however, must not interfere with goal g_1. In other words, being deceptive towards another agent must not negatively influence the accuracy of appraisals received from that agent too much.

Deciding the extent of deception is different from deciding delegation weights in scenario 1. For one, the value of the decision variable is now not only a result from the decision making procedure, but also influences a part of the opponent model. In scenario 1, the decision variable was the delegation weight towards each agent. Now, the decision variable is 'dishonesty', which is part of the opponent model. Second, the decision pertains to transactions in the near future, instead of the current transaction. Our agent needs to predict the effect of its deception on future transactions.

This introduces a problem because information about a transaction in the future is not yet available. In particular, the certainty asserted by an opponent in a future transaction is important for predicting the appraisal error, but is not known beforehand. Using the opponent model without a value for certainty would cause none of the rules in the rule base to fire.[9] In this case, the opponent model does not produce a prediction for the appraisal error, rendering it essentially useless.

Our solution to this problem, is to generate a set of arguments for each decision for a number of hypothetical values of 'certainty'.[10] This way, we effectively removed the 'certainty' variable from the opponent model, leaving the relation between 'dishonesty' and appraisal error.

[9] All rules in Listing 4.13 contain the 'certainty' variable in their antecedents. If this variable is given no value, the value is not member of any fuzzy set. The result of this is a match strength of 0, which means that the rule does not fire.

[10] More specifically, we generated an argument for 100 equally spaced values of 'certainty' between 0 and 1.

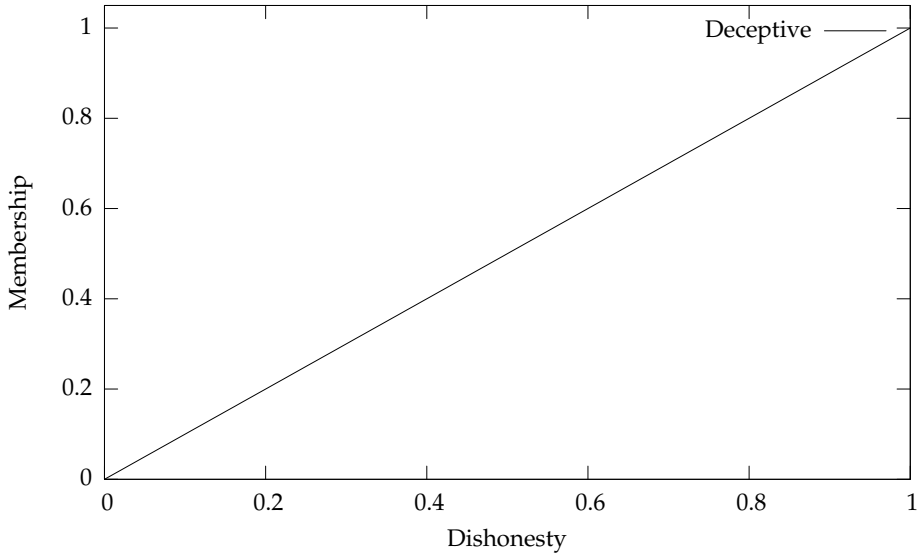


Figure 4.26: The *deceptive* set used to evaluate the deceptive nature of the certainty asserted to other agents

Next, the *Level* and *Weight* of each of these arguments is averaged and compared to obtain an aggregated *Level* and *Weight*. The recommended decision is then calculated in the normal fashion. In what follows, we illustrate this process by calculating the best level of deception for HONEST and RECIPROCAL individually.

Goals In addition to goal g_1 from scenario 1, goal g_2 = (dishonesty is deceptive, 0.5) is included in the goal base of our agent. *Deceptive* is a fuzzy set in the domain of dishonesty. The membership function is depicted in Figure 4.26.

Observations There are no relevant observations in this particular decision making process, because it pertains to transactions in the future.

Decisions For each agent, we consider five different decisions: d_1: dishonesty is 0.0, d_2: dishonesty is 0.25, ... d_5: dishonesty is 1.0.

Decision for HONEST As mentioned before, for each decision d_1 to d_5, we generate sets of arguments for a number of values of 'certainty'. The relevant properties of these sets are summarized in Table 4.7.

As expected, the extent of our agent's deception has no influence on the accuracy of appraisals generated by HONEST. The average appraisal error is the same for the different values of dishonesty. This is due to the fact

that HONEST is unresponsive: it does not react to other agent's behavior. Because the knowledge used to support these decisions is the same, the average *Level* of the arguments is also the same. The average weights of the arguments therefore are decisive. In this case, decision d_5 has the highest weight, and is most preferred.

	Dishonesty	**Avg. Error**	**Goal Satisfaction**		**Avg. Level**	**Avg. Weight**
			g_1	g_2		
d_1	0.00	0.50	0.50	0.00	2.95	0.50
d_2	0.25	0.50	0.50	0.25	2.95	0.63
d_3	0.50	0.50	0.50	0.50	2.95	0.75
d_4	0.75	0.50	0.50	0.75	2.95	0.88
d_5	1.00	0.50	0.50	1.00	2.95	1.00

Table 4.7: Properties of the sets of arguments supporting the decision to delegate to HONEST agent. The average appraisal error is obtained by averaging over the predicted appraisal errors for each value of 'certainty' in the set. Goal satisfaction refers to the average goal satisfaction based on the average appraisal error (goal g_1), and the deception towards HONEST (goal g_2).

Decision for RECIPROCAL Table 4.8 shows a summary of the sets of arguments generated for each decision pertaining to the dishonesty towards RECIPROCAL. In contrast to the HONEST agent, we see here that the extent of our agent's dishonesty towards this agent *does* influence the average appraisal error. Of course, due to the nature of RECIPROCAL, this is to be expected, because it punishes dishonesty with increasing its own. Consequently, for equal levels of certainty, appraisal error increases.

The interesting aspect of this scenario is the trade off between goals g_1 and g_2. Our agent has to decide what it values most: an accurate appraisal *from*, or its deception *towards* RECIPROCAL. With this particular goal base, and its associated priorities, we conclude from Table 4.8 that our agent favors the latter. Decision d_5 is preferred based on the fact that it has the highest weight.

We also experimented with alternative configurations, to investigate the influence of the goal-priorities on the preferred decision. Figure 4.27 shows the weights of decisions d_1 to d_5 for different priorities of goal g_2. As this priority decreases, goal g_1 becomes relatively more important. If the priority of g_2 is equal to 0.2, the scale suddenly tips over in favor of decision d_1.

	Dishonesty	Avg. Error	Goal Satisfaction		Avg. Level	Avg. Weight
			g_1	g_2		
d_1	0.00	0.63	0.37	0.00	1.49	0.37
d_2	0.25	0.75	0.25	0.25	1.49	0.38
d_3	0.50	0.85	0.15	0.50	1.49	0.40
d_4	0.75	0.87	0.13	0.75	1.49	0.51
d_5	1.00	0.85	0.15	1.00	1.49	0.65

Table 4.8: Properties of the set of arguments supporting the decision to delegate to RECIPROCAL

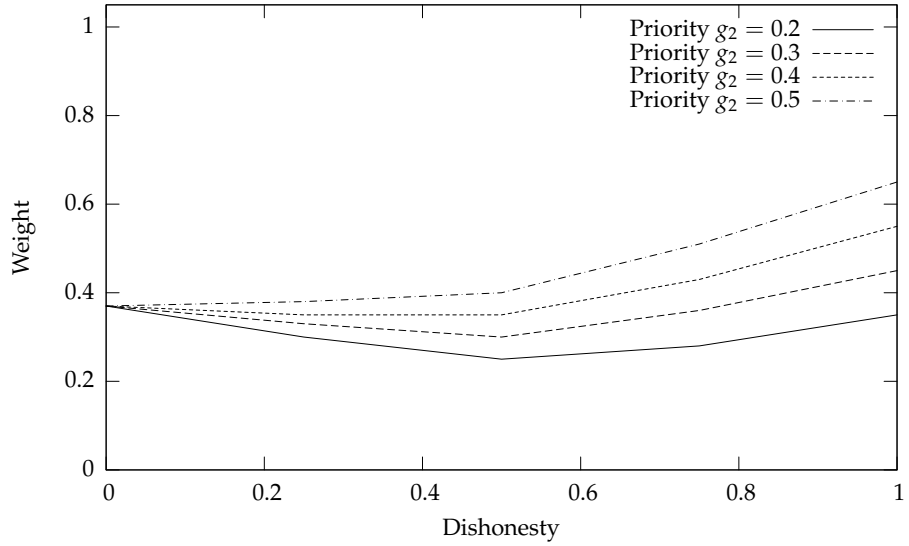


Figure 4.27: The weights assigned to decisions supporting different levels of dishonesty for various priorities of goal g_2.

4.5.3 Evaluation

In this section, we saw decision making applied in two (static) scenario's. In these scenario's, our agent took both the role of appraisal provider and appraisal requester. We departed from the opponent models obtained during the experiments with the opponent modeling component of our approach.

Using the results obtained from these scenario's, we are now able to use the evaluation criteria defined in Section 4.2. Of the four criteria for decision making, two are applicable on the *static* scenario's we sketched in this section. The two remaining criteria are postponed until next section, since they are better suited to evaluating the decisions made by the complete system (i.e. the *dynamic* behavior).

Evaluate an interaction's utility We have seen how arguments were constructed to recommend delegation agreements with opponents. In the argumentation framework applied in our approach, the strength of these arguments is not only influenced by utility of the decision's (predicted) consequences, but also by the confidence in the knowledge used to predict these consequences. Our chosen decision making component therefore surpasses this requirement: it is able to evaluate the utility *and* the certainty of the potential agreement separately.

Comprehensibility of the arguments The rationale behind the two decisions under consideration in the first scenario was made clear by the constructed arguments. We saw which parts of the opponent model were used to predict the outcome of each decision, and how the utility and certainty of these arguments are derived. The explicit nature of these arguments (and the process that constructs them) enable the user to trace back the steps made by the agent to come to a decision. As a result, the rationale of the decision is made accessible to the user.

In the second scenario, accessing the rationale is possible, but requires more effort. The fact that not all necessary information is known about future transactions complicated the decision making process. In contrast to the first scenario, *sets* of arguments needed to be constructed to support a decision, instead of just one. Of course, it is possible to explicitly list all of these arguments (which we omitted here), but digesting these could be a lot of work. Future work could be directed to find a way to summarize these arguments as to make them more accessible.

A possible way achieving this, is explicitly removing the certainty variable from the opponent model by evaluating the relation between dishonesty and the appraisal error for a large number of values of certainty. This relation can be expressed in a modified opponent model. This model can subsequently be used for argumentation in the usual fashion. In the second scenario, we essentially skipped this step, and immediately generated arguments.

4.6 ABDM

In Section 4.4 and 4.5, we experimented with the opponent modeling and decision making components in isolation. This allowed us to identify the pros and cons of these components individually. Now, we are interested to see how they function in unison. Therefore, in this section, the complete system is subject to experimentation.

4.6.1 Setup

The upcoming experiments discussed in this section are based on the two scenarios discussed in Section 4.5. In scenario 1, our agent took on the role of opinion provider, whereas in scenario 2, our agent was an opinion provider.

The first two experiments in this section are based on scenario 1. In these experiments, we compare our strategy to a benchmark strategy that is explained in the next section. Based on the results, we statistically determine which strategy is best.

The third experiment is based on scenario 2. In this experiment, our agent needs to decide the quality of the opinions sent *to* others, without negatively influencing the quality of appraisals obtained *from* others. Consequently, it has to weigh its interests in the requester role (scenario 1), with those of the provider role (scenario 2).

Based on the results of these experiments, we determine our agent's predicted performance in ART against HONEST and RECIPROCAL in Section 4.6.6.

4.6.1.1 Metrics

Before continuing with the results of the experiments themselves, we need a metric to measure the performance of the agent's decisions. We are especially interested in the *quality* of decisions made by our agent, which is a result of both the accuracy of the opponent model, and the goals of the agent. More specifically, if our agent competes against multiple opponents, we would like to know how well a task is delegated to these opponents, especially in light of the evaluation criteria from Section 4.2. To this end, we need a measure of the quality of these delegation decisions.

For each painting that has to be appraised by our agent, all other agents in the testbed submit their opinion. Each opinion contains an error (the difference between the appraisal and the actual value of the painting), depending on the submitting agent's expertise. Ideally, we would like our agent to assign maximum weight to the agent submitting the opinion with lowest error. In practice, however, due to the unpredictable nature of some agents, and imperfections in model of these agents, this is not feasible.

In Figure 4.28, this problem is presented graphically. The *min* and *max* errors are the opinions submitted by the agent with the most expertise, and the least expertise respectively. The *actual error* is the error made by *our* agent, by weighing the opinions received from others. These weights are based on the strength of the arguments supporting the decision to delegate. For example, if two opinions are received, and both are assigned equal weight, the resulting error is equal to the average of the error in both opinions.

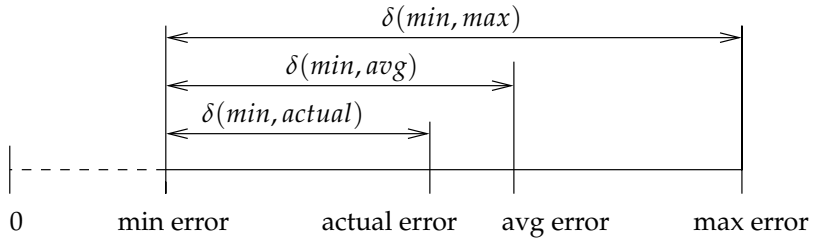


Figure 4.28: Example result of applying our strategy and the benchmark strategy on a single transaction.

We compare our agent's error to the error made by a trivial benchmark strategy. This strategy assigns equal weight to all agents. As a result, the appraisal error is always equal to the *average error*, regardless of the behavior of individual agents. If our strategy performs worse than this trivial strategy, we know for certain that we could have reached a better result with no effort at all.

Now, the difference between the minimum and actual error $\delta(min, actual)$, and the difference between the minimum and average error $\delta(min, average)$ can be used as an indicator of the quality of a decision. If $\delta(min, actual)$ is small compared with $\delta(min, average)$ (the error obtained by the benchmark strategy), our agent has given relatively more weight to better-than-average performing agents. So, the smaller the ratio between $\delta(min, actual)$ and $\delta(min, average)$, the better the decision of our agent compared to the benchmark strategy.

4.6.2 Two HONEST Agents (Requester Role)

In this first experiment let our agent compete against two instances of HONEST. In each transaction, our agent must determine which part of the task is delegated to each of them. We want to find out how our agent delegates the appraisal task to exact same agents. This gives us an opportunity to study the influence of the expected utility of a decision (or the *Weight* of an argument) in isolation, because the certainty of the knowledge used (or the *Level* of an argument) is approximately the same. After all, the opponent models of the same type of agent are almost identical.

The results of these experiments are shown in Figures 4.29 and 4.30. Figure 4.29 shows the quality of our strategy compared to that of the benchmark strategy. We can observe that—on average—the ratio $\delta(min, actual)/\delta(min, average)$ remains well under 1, the ratio at which both strategies perform the same.[11] A ratio of 1 would therefore be trivially

[11] In Figure 4.29, we used a *moving average* because the graph would otherwise be too erratic. As can be seen in Figure 4.30, the value of $\delta(min, actual)$ / $\delta(min, average)$ does in some cases exceed 1

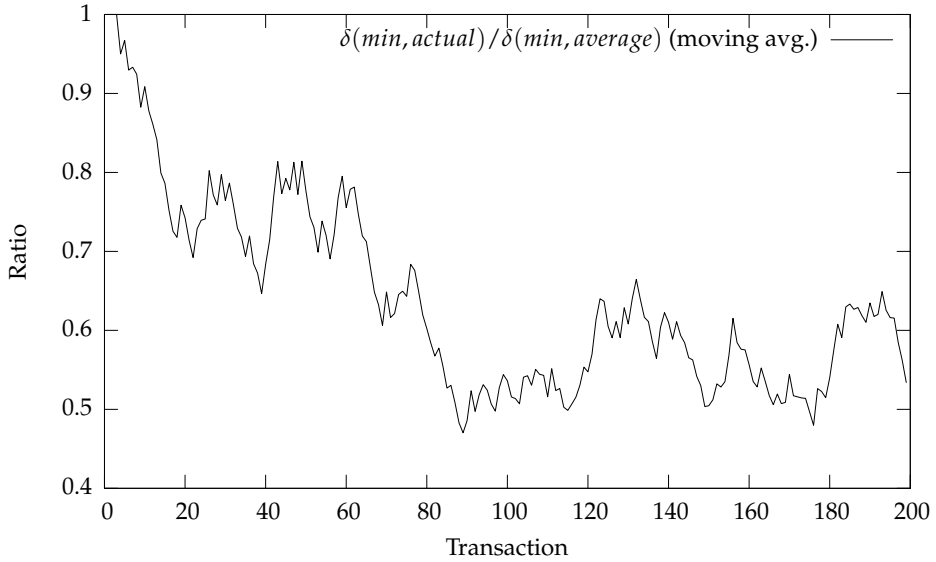


Figure 4.29: Relative performance of our strategy compared to the benchmark strategy during simulation against two instances of HONEST.

reachable by assigning equal weight to both instances of HONEST, or—on average—by randomly assigning weights.

In Figure 4.30, we plotted the relation between $\delta(min, actual)$ and $\delta(min, average)$. We see that for higher values of $\delta(min, average)$, the performance of our method is better than for lower values, compared to the performance of the benchmark.[12] In particular, for values of 0.1 and 0.2, our agent frequently makes poorer decisions than the benchmark. Fortunately, low values of $\delta(min, average)$ coincide with low values of $\delta(min, max)$, the *absolute* difference between the minimum possible and the actual error. Therefore, decisions with a low $\delta(min, max)$ value do not have a significant influence on the absolute performance, even if our agent makes a bad decision. On the other hand, decisions made for high values of $\delta(min, max)$ *are* decisive for the absolute performance of our agent.

Based on Figure 4.29 and 4.30, we can conclude that our agent has performed significantly better than the benchmark strategy.

4.6.3 All Agents (Requester Role)

In the second experiment, we included all opponent agents from Section 4.3.3. In contrast to the first experiment, where we used two instances of the same agent-type, we now deal with five different agents. Consequently,

[12] Of all data points, 19% lie above the line $\delta(min, actual) = \delta(min, average)$. Of these 19%, 94% lie left of the line $\delta(min, average) = 0.2$.

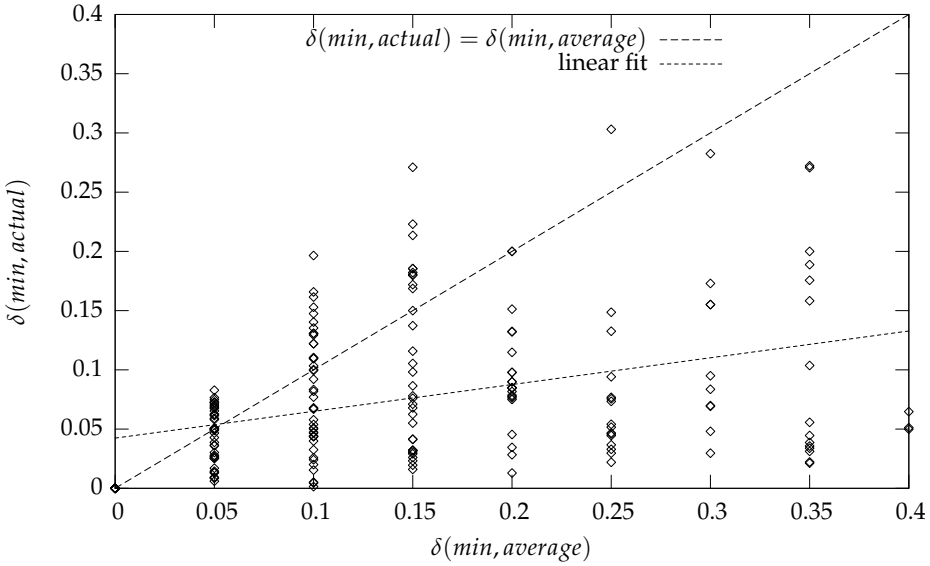


Figure 4.30: Appraisal errors obtained by our strategy plotted against the appraisal errors obtained by the benchmark strategy during simulation against two instances of HONEST.

the certainty of the knowledge used to predict the consequences now also plays an important role in decision making: for different agents, different opponent models are created, each with their own specific rules and confidence levels. This enables us to study the effect of the trade-off between expected utility, and our agent's confidence in the opponent models of each agent.

In order to do this, we measured both the metrics described in Section 4.6.1.1, and the weights assigned to each individual agent during the simulation. Figures 4.31 and 4.32 show the results. Just as in the previous section, our agent's performance $\delta(min, actual)$ is compared with the benchmark strategy's performance $\delta(min, average)$.

From Figure 4.32 we see that the relative performance increases during simulation, especially in the beginning (i.e. the ratio decreases). This is because as the opponent model increases in accuracy, so does the performance of our agent. This figure also shows that our strategy performs better than the benchmark strategy. Figure 4.31 corroborates this claim: the linear fit is well below the $\delta(min, actual) = \delta(min, average)$. In the next section, we will support this claim in a different way using a statistical test.

Let us focus on the weights assigned to the agents during simulation. These weights are plotted in Figure 4.33. In the first five transactions, all agents are assigned the same weights, because the data collected thus far is not enough to create an opponent model (remember that our update strat-

egy rebuild the opponent model after every 5 transactions). After the first opponent model is created after transaction 5, the first we observe is that the weight assigned to RANDOM drops very fast. Apparently, the amount of confidence in RANDOM's opponent model is low. This is consistent with our observations during our experiments with RANDOM (see Section 4.4.4).

Until transaction 30, HONEST, DISHONEST, SWITCH, and RECIPROCAL are assigned approximately the same weight. From transaction 30 onward, the weight assigned to RECIPROCAL is decreased. This is in line with our expectations, because the opponent model created after round 2 includes the retaliation in response to round 1. The relative imprecise manner in which the opponent modeling algorithm (FURL) describes this aspect of RECIPROCAL's behavior increases the amount of uncertainty in the model. We already addressed this phenomenon in Section 4.4.9. The consequences of these dropping plausibility ratings due to an increase in FURL credits, are clearly visible in dropping (relative) weights assigned to the RECIPROCAL agent.

The next event takes place when SWITCH switches from honest to dishonest behavior after transaction 100. As with RECIPROCAL, the delegation weight for SWITCH is significantly decreased in favor of both HONEST and DISHONEST. This drop in weight is caused by the fact that our agent does not consider the SWITCH agent reliable anymore. Again, this is due to FURL's relative inability to model SWITCH's behavior exactly. As can be seen in Listing 4.11, the exceptions that are introduced in the opponent model to describe the new behavior of the SWITCH agent are assigned a much higher credit, which corresponds to a low certainty.

As the simulation progresses, HONEST and DISHONEST are trusted with approximately two-thirds of the appraisal task. The RECIPROCAL and SWITCH agents together get about a third. RANDOM gets just 1 or 2 percent of the task.

This result is in accordance with intuition: HONEST and DISHONEST are most reliable and their performance is most consistent. RECIPROCAL and SWITCH change their behavior over time, and are therefore less reliable than HONEST and DISHONEST. The weights assigned to these former agents reflect this difference. Finally, RANDOM is assigned very low weights, and while it may have the same *average* performance as the other agents, its behavior is not predictable. In other words: RANDOM is untrustworthy and should not be considered as a possible transaction partner.

4.6.4 Statistical analysis (Requester Role)

In both previous experiments, we saw that—on average—our agent performed better than the benchmark strategy. In order to make sure this is not coincidental, and that our strategy is significantly better than the benchmark strategy in a statistical sense, we tested our hypothesis with a paired

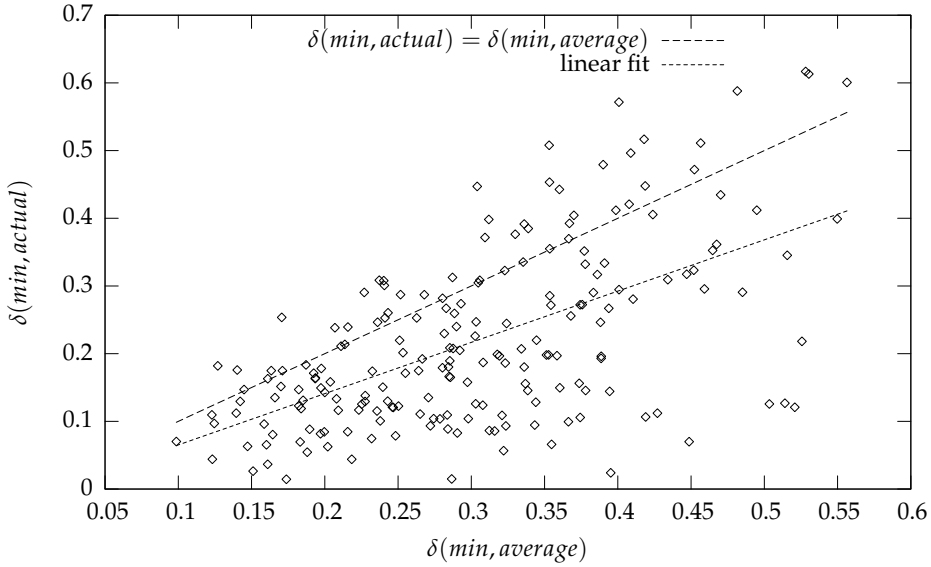


Figure 4.31: Appraisal errors obtained by our strategy plotted against the appraisal errors obtained by the benchmark strategy during simulation against all agents.

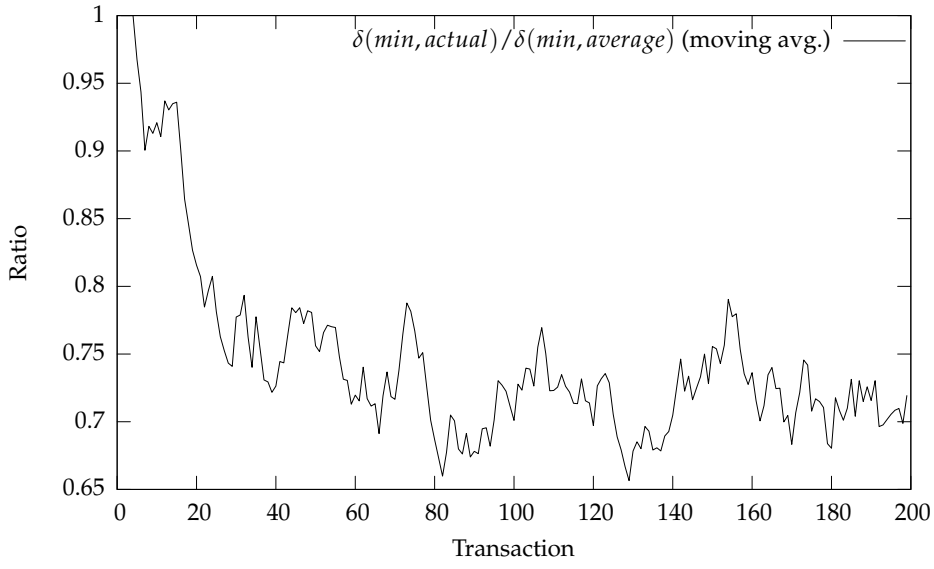


Figure 4.32: Relative performance of our strategy compared to the benchmark strategy during simulation against all agents.

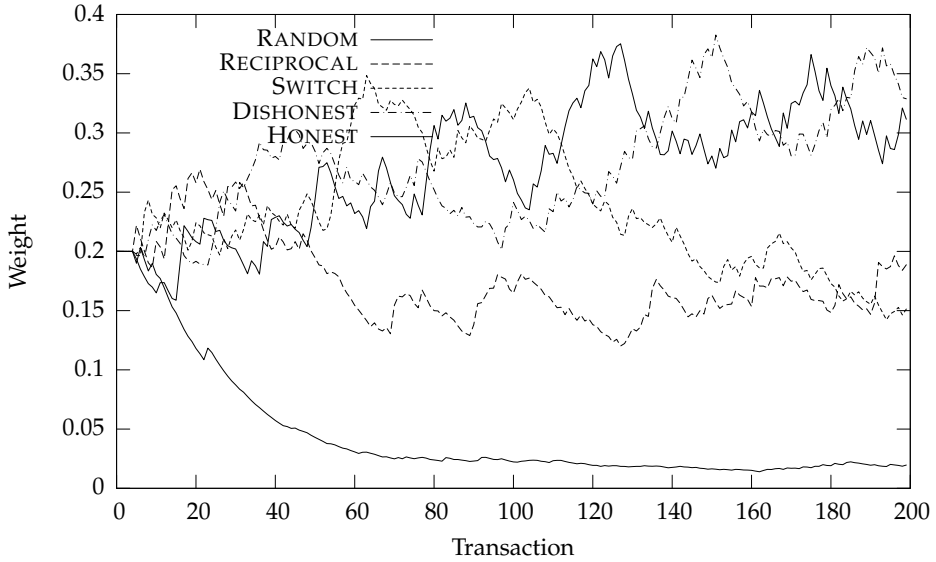


Figure 4.33: Moving averages of weights assigned to each agent during each transaction of the simulation

Student-t test. With a Student-t test, we can determine if two data sets are drawn from different distributions. More specifically, we tested if null-hypothesis H_0 'true means of error of both strategies are equal' could be rejected in favor of alternative hypothesis H_1 'the mean error of our strategy is lower than the error mean of the benchmark strategy'.

We performed this test on experimental data from five different experiments.[13] These experiments differ from each other in the types and number of opponents used. The results of the statistical analysis are summarized in Table 4.9.

For all but one experiment the test shows that our strategy produces significantly lower error than the benchmark strategy. The 95% confidence intervals indicate a true difference in means between both strategies ranging from approximately 0.04 to 0.1 in favor of our strategy. Comparing these figures to the average distance between minimum and maximum error, which is equal to approximately 0.3, this is a significant improvement (see Figure 4.28 for a visual aid).

The experiment with both RANDOM and HONEST did not provide conclusive evidence of better performance by our strategy. The explanation for this is simple. In Figure 4.33, we saw that our agent quickly responds to the lack of reliability of the RANDOM agent by considerably reducing its weight. This also happened in this experiment, leaving only the HON-

[13]Two of these were discussed in the previous sections. Three additional experiments were run to obtain more evidence.

Opponents	**p-value**[a]	**95% Conf. interval**[b]		**Significant**[c]
		From	To	
All	$\ll 0.01$	0.0688	0.0989	Yes
RANDOM, DISHONEST, HONEST	$\ll 0.01$	0.0381	0.0749	Yes
RANDOM, HONEST, RECIPROCAL	$\ll 0.01$	0.0445	0.0831	Yes
RANDOM, HONEST	0.43	-0.0126	0.0293	No
HONEST, HONEST	$\ll 0.01$	0.0521	0.0812	Yes

[a]The probability that null-hypothesis H_0: 'true means of error of both strategies are the same' is accepted in favor of alternative hypothesis H_1: 'the mean error of our strategy is lower than the error mean of the benchmark strategy'

[b]The 95% confidence interval for the true difference in means of both strategies. A positive difference indicates a higher average for the benchmark strategy than for our strategy.

[c]Whether or not the p-value is low enough (confidence level 5%) to reject H_0 and accept the alternative hypothesis H_1.

Table 4.9: Results of comparing our strategy with the benchmark strategy using the student t-test

EST agent as a viable delegation option. HONEST, honest it may be about its expected performance (i.e. honestly asserts its certainty), is not capable of adequately appraising *all* paintings. So, if the appraisal of all these paintings (which are randomly generated by the testbed) is delegated to the same agent, the quality of the obtained appraisals is totally random.

The random nature of these appraisals dissolves the differences between any pair of strategies. This fact is reflected in the high p-value in Table 4.9, which indicates no significant difference between our strategy and the benchmark strategy for this particular run.

4.6.5 Scenario 2: HONEST and RECIPROCAL (Provider Role)

The third and final experiment in this series is based on scenario 2 from Section 4.5.2.2. In this experiment, we study the effect of two possibly contradicting goals on the quality of the appraisals sent to other agents. Note that in contrast to scenario 1, our agent is now provider of appraisals, instead of requester (see Figure 4.1).

Our agent is asked by HONEST and RECIPROCAL to send its opinion. Each time, it needs to contemplate the extent of its deception towards these agents. Deception can of course provoke a retaliatory response from its opponents. This could possibly lead to less accurate appraisals of our own agent, which is not desirable in light of goal g_1: the error of appraisals obtained from others is low.

In this experiment, we wish to observe how the goal base discussed in scenario 2 can lead to an explicit trade off between two of our agent's desires: be deceptive towards others, unless this leads to less accurate ap-

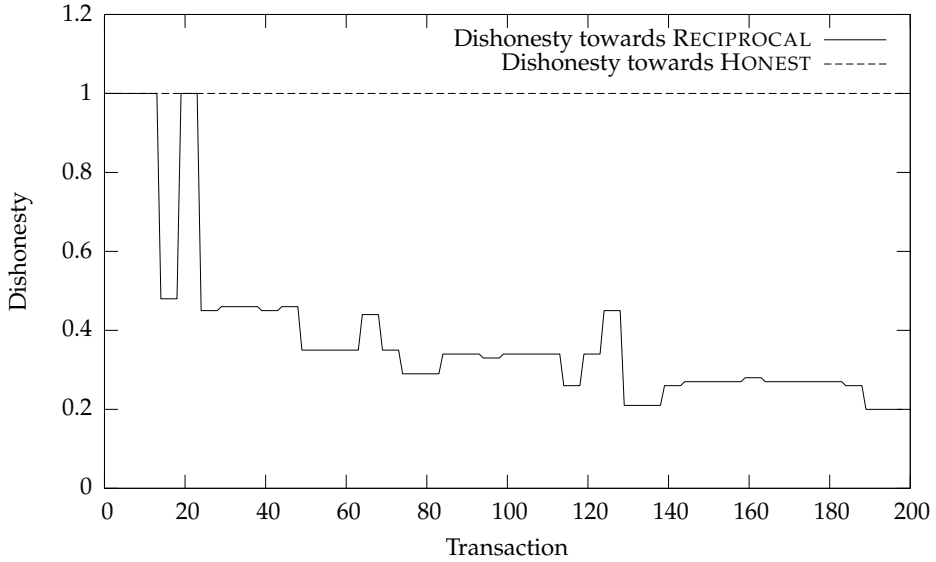


Figure 4.34: Deception towards HONEST and RECIPROCAL agents

praisals.

We expect that our agent's behavior is consistent with observation made in earlier research on trust in Multi-Agent Systems. Axelrod (1984) observes that when confronted with a Tit-for-Tat or reciprocal strategy, it is in one's best interest to cooperate. So, in competition with HONEST and RECIPROCAL that reciprocates both defection and cooperation, we expect that our agent will at least be less deceptive towards RECIPROCAL than towards the HONEST agent.

Figure 4.34 shows the results. Towards HONEST, our agent starts off being dishonest. Because the opponent model does not contain any indications that HONEST will retaliate, both our agent's goals are attainable. Consequently, HONEST is treated with maximal dishonesty.

Towards RECIPROCAL, however, dishonesty drops as soon as our agent detects a relation between deception *towards*, and the deception received *from* RECIPROCAL. As the simulation progresses, this relation becomes more certain and is modeled more accurately, resulting in less deceit towards RECIPROCAL. Apparently, our agent values accurate appraisals to such an extent, that it rather treats RECIPROCAL more kindly, than to deceive it. This observation is consistent with Axelrod (1984): a reciprocal agent persuades its opponent to behave trustworthy by punishing uncooperative behavior. This yields a win-win outcome: both agents benefit from RECIPROCAL's enforcement of honesty (this is consistent with Definition 2 on page 11).

4.6.6 Performance in the ART Testbed

The precise consequences of the results of the experiments above on the performance these agents in the ART testbed are heavily dependent on the competition parameters. However, based on the results from the experiments discussed in this section, we can give a *rudimentary* prediction of the market-shares. Remember from Section 4.3.1, that the market-share of agents participating in the simulation depends on the accuracy of appraisals in previous rounds. If an agent on average appraised their client's paintings better than its competitors, it will get a larger fraction of clients present in the market. This, of course, results in a higher turnover.

In order to calculate these market-shares, we made an assumption about how HONEST and RECIPROCAL calculate delegation weights towards their opponents. We decided that both HONEST and RECIPROCAL use the asserted certainty as an indicator for expected result. This means that they do not expect their opponents to lie. Certainties received from agents are therefore used to weigh their influence on the final appraisal.

Figure 4.35 shows the results of this simulation. The market-shares are not very far apart, but this figure shows that our strategy performs best, followed by RECIPROCAL and HONEST. RECIPROCAL beats HONEST, because HONEST is deceived by our agent, whereas RECIPROCAL persuades our agent to cooperate. In this particular simulation, our agent beats both agents, because it does not blindly trust the asserted certainties from its opponents. Instead, has built an opponent model that predicts the actual appraisal error based on a number of variables. For example, it can predict the appraisal error of RECIPROCAL based on the deception towards it in the previous round. That way, it can more accurately decide to whom the appraisal task is delegated, giving it a strategic edge over its competitors.

Again, note that these results are *indicative* of the performance of our strategy in a real ART competition. Exact results depend on testbed parameters, the precise implementation of both RECIPROCAL and HONEST, and the configuration of our strategy.

4.6.7 Evaluation

In this section, we have seen how the opponent modeling and decision making components work together, and how they perform compared against a benchmark strategy. In particular, these experiments showed how the computed opponent models are translated in concrete decisions by the decision maker. This allows us to use the remaining two evaluation criteria from Section 4.2.

Determine whether to interact In the ART Testbed, the question is not so much *whether* to interact with another agent, but rather *to what extent*.

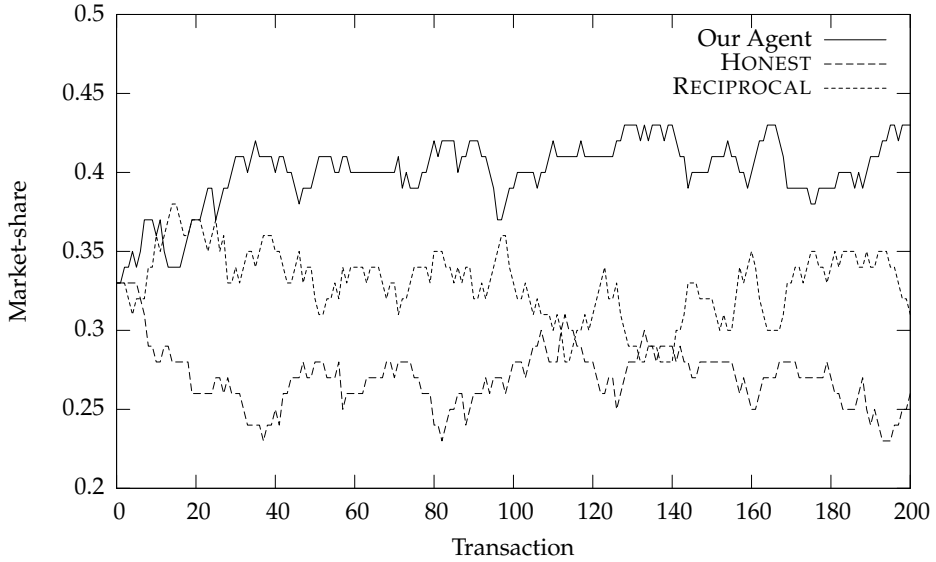


Figure 4.35: Market-shares during ART Testbed simulation with three agents

In these experiments, we saw how the strengths of arguments were used to determine what share of the task is delegated to others. The measures introduced in the beginning of this section subsequently allowed us to measure the quality of decisions in terms of the desirability of its outcomes. Using these measures, we showed that our approach can withstand comparison with a simple benchmark strategy (see Figures 4.30 and 4.31, and Table 4.9).

Identify and isolating untrustworthy agent Figure 4.33 clearly shows how less reliable or untrustworthy agents are isolated by assigning them less weight. For example, RANDOM—the most unreliable agent—is quickly excluded as a delegation partner. In contrast, reliable agents as HONEST and DISHONEST are given relatively more weight than others.

4.7 Conclusion

In this chapter, we have experimented with the system in its entirety and its two major components individually. At the closing of this chapter, we wish to take a moment to evaluate the results in terms of the evaluation criteria from Section 4.2. While these have already been addressed individually at the end of each experiment, we reiterate the most important observations here.

Remember from Section 4.2, that the criteria were subdivided into criteria for to opponent modeling, and criteria for decision making. As op-

ponent modeling is concerned, we concluded that choosing FURL as an opponent modeling algorithm has a number of advantages. In particular, in terms of the comprehensibility of the models, FURL proved to have significant improvements over numerical opponent modeling algorithms currently applied in trust. Also, we saw that this advantage does not have a negative impact on the rest of the criteria. In terms of accuracy and convergence FURL models the behavior of simple agents well. We did, however, conclude that on the issue of adaptiveness, FURL still needs some more improvements. We come back to this in Section 5.2.2.

Similarly, experimentation with decision making showed that the algorithms can live up to both our and the research community's expectations. Again, as comprehensibility of the decisions is concerned, our approach is not only able to explain the reasons for making a decision, but can also refer to the parts of the opponent model that were relevant in decision making. Experiments with the full system showed that our approach is very able to quickly set untrustworthy opponents apart from others. In addition, our approach also withstood comparison with a benchmark strategy in terms of the the quality of delegation decisions.

Based on these facts, we can conclude that our approach can combine the desire to understand the reasons behind a trusting decision with existing requirements for trust algorithms, and that these desires do not necessarily conflict.

5 Conclusions and Future Work

The aim of our research (formulated in Section 1.2) was to design an architecture for an agent capable of explaining why it trusts another agent and why it is willing to delegate a task to that agent. To show the viability of the architecture, we designed and implemented a proof-of-concept using existing algorithms.

Based on our experimentation with the proof-of-concept, we are now able to discuss both the advantages and disadvantages of our approach in Section 5.1. In Section 5.2, these give rise to further improvements of the current implementation of the system, and—more importantly—help to identify aims for future research.

5.1 Conclusions

The research presented in this thesis differs from existing research in the following respects. In comparison with the numerical models we discussed in Section 2.2, we see that our approach provides more extensive opponent modeling capabilities. In comparison with existing decision models (see Section 2.3), our approach is not only capable of capturing the rationale of a decision, but also features a fully operational opponent modeling algorithm.

These differences are due to the different aims of our research: our aim was to develop an trusting agent that is able to explaining why decisions are made. Until now, existing research has primarily focused on performance, not explanation. Consequently, our research has had a strong exploratory nature, even stronger than existing research on trust in Multi-

Agent Systems, a field that is itself still in its infancy.

Concerning the two components of our approach individually—opponent modeling and decision making—, we have seen that it is possible to construct a trust algorithm using existing algorithms: a rule learner and an argumentation framework. The rule learner we employed—FURL—proved capable of computing a fuzzy opponent model from the observed behavior of simple agents. In addition, the argumentation framework created by Amgoud and Prade (2004), proved to have sufficient abilities to convert this opponent model in actual trusting decisions. More importantly, it is able to explain these decisions in terms of observations (using parts of the learned opponent model), and the preferences of the agent.

Experimentation with our approach showed that it can live up not only to existing criteria in the trust community, but also to our newly proposed requirement (i.e. explanation). Based on this, we consider our first step towards a trust algorithm capable providing explanation of its decision a success. Hopefully, this will encourage others to adopt (elements of) our architecture, and continue where we left off.

5.2 Future Work

In the previous section, we discussed both the strength and weaknesses of our proposed approach and our proof-of-concept. On the one hand, the weaknesses can be subject of further research aimed at finding improvements of both the architecture and the proof-of-concept. On the other hand, the strengths of our approach can lead to new insights if applied to existing research on trust in Multi-Agent Systems, but also on other areas. In this section, we highlight the areas for improvement we deem most fruitful for future work.

5.2.1 Reputation

In our research, we concentrated on modeling behavior based on direct interaction. This means that our agent only uses information obtained during directly interacting with its opponents. For example, in the ART Testbed, these transactions involved the exchange of appraisals.

However, other sources of useful information about a possible interaction partner exist. The most important source of information about the partner is other agents: they also gather information about the behavior of their opponents. If communicated to other agents, this information is usually referred as *reputation*. Reputation is the opinion other agents have of an agent's reliability and trustworthiness. Until now, this information has not been used in our research.

Adding support for reputation in our approach could be a major improvement. This would involve broadening our architecture to include reputation, designing new algorithms to select agents from which reputation information is requested, and somehow aggregating these reputations.

A *possible* way of doing this, is to extend opponent modeling for a new kind of transaction: the exchange of reputation (the protocol in ART for such a transaction is depicted in 4.2). Of course, this kind of transaction is quite similar to the *actual* transaction. At the end of both types of transactions, something is exchanged. In the actual transaction, this can be a service, information or a product. In a reputation transaction, it is the reputation of another agent in the eyes of the agent that provides it. The quality of the result of both transactions can also be measured. For instance, the quality of a reputation depends on how well it describes the behavior of the particular agent. Now, our existing opponent modeling approach can be employed to model an agents behavior both for normal transactions, and the reputation transaction.

Not only opponent modeling should be extended, but also decision making is subject to change. At first, only a single source of information was available. Now, we need to deal with multiple sources that possibly contradict each other. This allows for the construction of multiple arguments (for example, one for each source). An improved argumentation framework could then be employed to reason about contradicting arguments. The decision that is obtained in this manner, would take into account both reputation, and information from direct interaction.

With these modifications, our approach can be extended to include reputation with a quite limited amount of effort.

5.2.2 Opponent Modeling

The FURL algorithm we used in our approach has a number of limitations as discussed in Section 5.1 and 3.4.4.

For one, FURL is a batch learning algorithm, which means that each time new data is obtained, the opponent model needs to be generated from scratch. In contrast, an on-line algorithm is capable of refining an existing model using data that has newly been obtained.

More importantly, FURL is incapable of detecting more complex behavior. By this, we mean that it is not able to accurately model data sets with a large number of input variables. Not only would FURL need an extensive amount of time due to its time complexity (see 3.4.4), but also will its results be increasingly less accurate as this number increases. The experiments we ran on two more complex data sets (the auto-mpg data set and the data set with information on diabetes showed that FURL has significant difficulty with more detailed patterns. Unfortunately, such patterns, present in just a few input-samples, are not accurately detected.

Lastly, FURL uses many fuzzy rules to describe the behavior of an agent. This could hamper the understanding of the behavioral model by the user. Further research could therefore try to find a way to summarize these rules in a brief description of the opponent.

5.2.3 Argumentation

To fully exploit the properties of an improved opponent model (for example, based on our suggestions above), more research needs to be done to study argumentation frameworks capable of providing the user with more extensive information. We already mentioned the possibility of using argumentation to aggregate reputations from different agents.

Other improvements involve using arguments to reason about other agents motives. For example, hypotheses about an agent's behavior can be tested using evidence from direct interaction and reputation. An example of such an hypothesis is the existence of a cartel of agents in the MAS. These agents collaborate tightly with each other, and try to deceive agents not part of the cartel. Arguments can be set up to support or reject each of these hypothesis. A decision criterion can then be employed to determine which hypotheses are deemed true.

The most important precondition for the application of a more extensive argumentation-framework, however, is that the opponent model is sufficiently rich and complex. In turn, this requires a sufficiently rich and complex environment, from which a sufficient amount information can be extracted. In Section 2.3, we observed how the application of complex decision models was limited by the absence of opponent models satisfying these requirements.

Based on these findings, we feel that the interrelation between the environment, the opponent model, and the decision making algorithms in trust deserves more study.

5.2.4 Application to other Contexts

Our proposed approach makes the rationale behind a decision available through a clear distinction between modeling and decision making. The applicability of a combination between an algorithm that can mine data sets to obtain fuzzy rules (like FURL does), and argumentation to explain a recommended decision to the user, is not limited to trust in MAS. We think it would be very interesting to explore its application to other contexts.

In the beginning of the introduction, we already gave an example of such a context. As another example, our approach could possibly be used to model data sets pertaining to glucose levels of diabetes patients. Such a model could then be used to give instructions to the patient for keeping

his glucose levels within acceptable boundaries, by recommending him on food, rest, and medication.

5.2.5 Comparison with other approaches

In this thesis, we have discussed and studied our approach in isolation. Comparison with other approaches in the field of trust in Multi-Agent Systems could therefore be an extension to the research presented in this thesis.

However, due to the different aim of our research compared to the other research in this area, comparison of our work to that of others will be difficult. In our literature survey (Stranders, 2004), we did not come across a fully operational approach using an explicit knowledge format. In our view, this is a precondition for providing the rationale that can be understood by the user. Moreover, the subjective nature of the requirement of human comprehensibility hampers the creation of a suitable benchmark. As a consequence, comparison of approaches on the basis of human understandability is thus far not possible.

Be this as it may, comparison of our approach using the objective criteria of the ART Testbed is still a possibility. While it has not been one of the aims of our research to develop an agent that outperforms other agents in this testbed, but it might be interesting to see how an agent equipped with our system performs relative to other agents. In order to do this, our system has to be extended to conform to the full set of ART game rules. Hopefully, our desire to use an explicit knowledge format can be combined with the desire to perform well in ART.

With these modifications, we believe our approach is an important step forwards in the field of trust in Multi-Agent Systems.

A

Appendix: Implementation

The design and implementation of the proof-of-concept has been a major part of our research. At the time of this writing, the project consists of approximately 30,000 lines of code.[1] We would like to give a brief account of the main packages our project consists of, and the tools we used.

A.1 Package description

In Figure A.1, the high-level packages and their relations are outlined. In the following, we summarize the function and implementation of each of them.

A.1.1 FUZZYRULEBASE

The FUZZYRULEBASE package contains our implementation of a simple fuzzy rule base. Its performance is optimized for piecewise linear fuzzy sets (triangular sets and aggregated sets such as the set illustrated in the bottom of Figure 3.4). We also implemented a special forward chaining algorithm that exploits the special nature of rules in an HPS (rules are either exceptions of other rules, or base rules).

Parts of our implementation of FUZZYRULEBASE are based on the book Bigus and Bigus (2001).

[1] We measured the size of our project using an open source tool called StatCVS (`http://statcvs.sourceforge.net/`). StatCVS is a tool that gathers several statistics from a CVS repository.

A.1.2 FURL

Unfortunately, no implementation of FURL was readily available. We therefore chose to implement it ourselves. FURL was implemented in a top-down fashion. First, we designed and created the high-level algorithm outlined in Algorithm 1, using interfaces for each component that had still to be implemented. For example, we created interfaces for the partitioning, repair, and stopping algorithms were created after the high-level algorithm itself was finished.

The highly flexible nature of the algorithm we obtained in this fashion, proved to be very welcome during experimentation. We were able to quickly implement alternatives for the existing components and install them into the high-level algorithm to study their performance.

A.1.3 HPS

The package HPS contains an implementation of the Hierarchical Prioritized Structure, which is specified by Yager (1993). For each level in the HPS, an instance of FUZZYRULEBASE is used. Only a little extra logic was required to aggregate the outputs of each level using Equation 3.2.

A.1.4 ARGUMENTATION

The argumentation framework as discussed in Section 3.5.1 is implemented in this package. It is implemented in such a fashion that any kind of opponent model (or system description) can be used in combination with it. If used in combination with FURL, it uses an instance of a HPS to predict the consequences of available decisions.

In our research, we have only used one decision criterium (pessimistic), but the implementation offers the possibility to plug-in many different criteria.

A.1.5 ABDM

ABDM stands for 'Argumentation Based Decision Maker'. The package ABDM provides a unified interface for the storage of input-output tuples of the observed system (in this case opponent agents), the generation of a model (using FURL), and decision-making (using the ARGUMENTATION package). One instance of ABDM can be used to model the behavior of one system, and make decisions about available actions towards the system.

A.1.6 ART

The ART package contains an implementation of the Agent Reputation and Trust Testbed. At the time of the start of our research, no implementation

of this testbed was yet available. We implemented a rudimentary version of this testbed, and made some modifications to facilitate our experiments (see Section 4.3.1).

The official implementation of ART is now available at `http://www.lips.utexas.edu/art-testbed/`.

A.1.7 AGENT

The various agents used in our experiments are implemented in the package AGENTS. To reflect the different roles an agent has in the ART Testbed (opinion requester/provider, reputation requester/provider), we split up an agent's behavior in roles (cf. 'behavior' in Jade[2]). This significantly improves the ease of implementation, and the reuse of roles in different agents. Our agent was also implemented using these roles.

A.1.8 EXPERIMENT

The EXPERIMENTATION package contains all code used for experimentation. It uses the ART Testbed, and the different agents from the Agent package (including our own).

A.2 Tools

During our research, we used several (open-source) tools to manage different aspects of our code. The three most important are listed below.

A.2.1 Eclipse

The Eclipse Integrated Development Environment (IDE) (`http://www.eclipse.org`) is an invaluable tool for Java programming. It allows for quick navigation of the code, it supports code completion which significantly reduces the amount of code the programmer has to write, and has on-the-fly syntax checking. Writing and maintaining our code would have been much harder if it was not for Eclipse.

A.2.2 Maven

Maven (`http://maven.apache.org`) is a project management tool. Among others, Maven can compile, build, and deploy projects with a push of a button. It needs a description of the project (called a Project Object Model or POM), in which the relevant properties of the project are described. For

[2]Jade is a framework that facilitates the implementation of Multi-Agent Systems. See `http://jade.tilab.com` for more details.

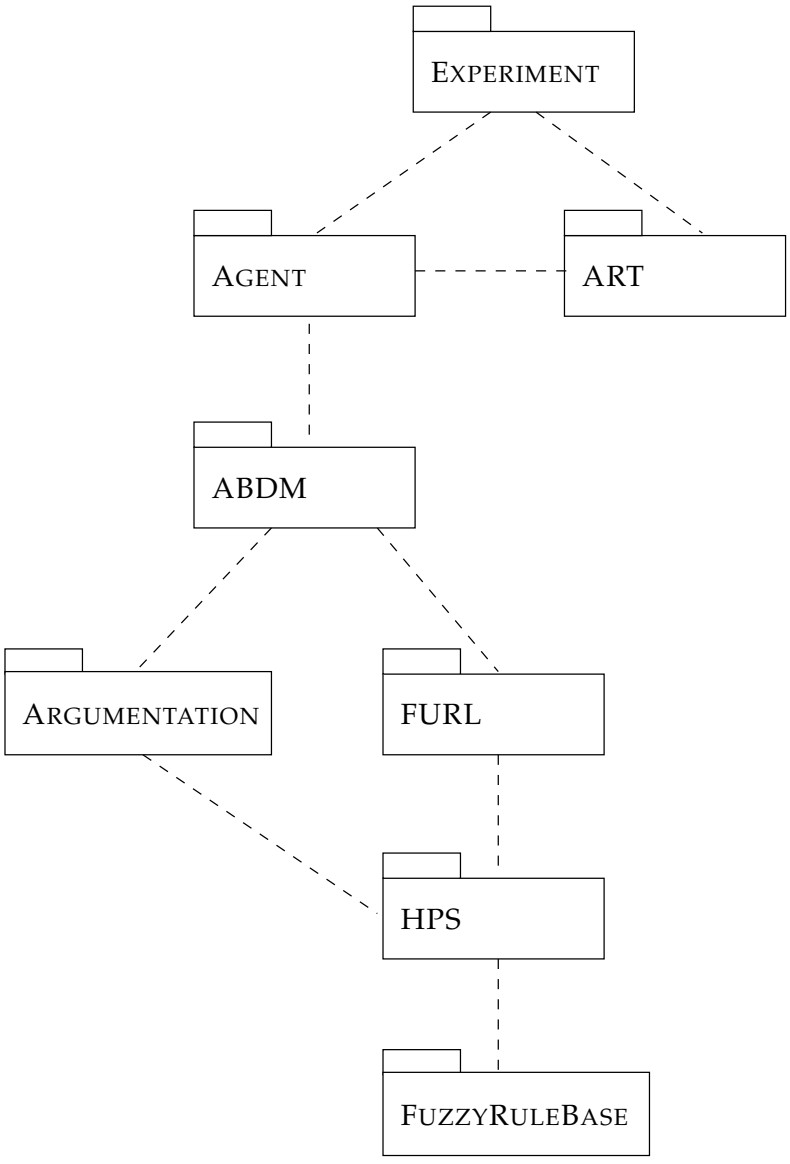


Figure A.1: Package diagram for the complete system.

one, Maven needs to know the name, and the layout of the project, and its dependencies (or libraries). These dependencies are automatically downloaded from a central repository, and need not be stored in the CVS repository.

Its advantage over Ant (a similar project management tool), is that the user does not need to describe *how* tasks, but *what* tasks must be performed.

A.2.3 JUnit

In our project, we used parts of the eXtreme Programming (XP) methodology. One technique from this methodology, is to write (automated) tests first, and implement classes later. The main benefit of this method, is the ability to get rapid feedback for the code that was just written.

The testing framework JUnit (`http://www.junit.org`) supports this technique. JUnit allows the programmer to write unit-tests. The aim of a well written test, is to falsify the theory 'the code is correct'. If this attempt fails, confidence in the correctness of the code is increased.

For each part vulnerable to critical bugs, we implemented a JUnit test. After each (seemingly trivial) improvement, we reran all tests to see if we did not accidentally introduced a bug.

Bibliography

Amgoud, L. and Prade, H. (2004). Using arguments for making decisions: a possibilistic logic approach. In *AUAI '04: Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 10–17. AUAI Press.

Axelrod, R. (1984). *The Evolution of Cooperation*. BasicBooks.

Barthés, J. A. and Tacla, C. (2001). Agent-supported portals and knowledge management in complex r&d projects. In *Sixth International Conference on CSCW in Design (CSCWD'01)*, pages 287–292.

Bigus, J. and Bigus, J. (2001). *Constructing Intelligent Agents using JAVA*. Wiley, New York.

Bottou, L. and Bengio, Y. (1995). Convergence properties of the *K*-means algorithms. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems*, volume 7, pages 585–592. The MIT Press.

Castelfranchi, C. and Falcone, R. (2001). Social trust: A cognitive approach. In Castelfranchi, C. and Tan, Y., editors, *Trust and Deception in Virtual Societies*, pages 55–90. Kluwer Academic Publishers.

Castelfranchi, C., Falcone, R., and Pezzulo, G. (2003). Trust in information sources as a source for trust: a fuzzy approach. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 89–96, New York, NY, USA. ACM Press.

Cordon, O., Herrera, F., and Villar, P. (2001). Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base. *IEEE Transactions on Fuzzy Systems*, 9:667–674.

Dubois, D., Lang, J., and Prade, H. (1994). Automated reasoning using possibilistic logic: Semantics, belief revision, and variable certainty weights. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):64–71.

Falcone, R., Pezzulo, G., and Castelfranchi, C. (2003). *Trust, reputation, and security: theories and practice*, chapter A Fuzzy Approach to a Belief-Based Trust Computation, pages 73–86. Springer-Verlag.

Falcone, R., Pezzulo, G., Castelfranchi, C., and Calvi, G. (2004). Why a cognitive trustier performs better: Simulating trust-based contract nets. In *Proceedings of AAMAS 2004*.

Franklin, S. and Graesser, A. (1996). Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35.

Fukuyama, F. (1996). *Trust: the social virtues and the creation of prosperity*. Free Press.

Fullam, K., Sabater, J., and Barber, K. S. (2005). Toward a testbed for trust and reputation models. *Trusting Agents for Trusting Electronic Societies*, pages 95–109.

Fullam, K. K., Klos, T. B., Muller, G., Sabater, J., Schlosser, A., Topol, Z., Barber, K. S., Rosenschein, J., Vercouter, L., and Voss, M. (2004). Trusting in agent societies competition game rules (version 1.0). Technical report, Laboratory for Intelligent Processes and Systems, University of Texas.

Ginsberg, A. (1989). Knowledge base refinement and theory revision. In *Proceedings of the sixth international workshop on Machine learning*, pages 260–265, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Guttman, R., Moukas, A., and Maes, P. (1998). Agent-mediated electronic commerce: a survey. *The Knowledge Engineering Review*, pages 147–159.

Herrera, F., Lozano, M., and Verdegay, J. L. (1998). A learning process for fuzzy control rules using genetic algorithms. *Fuzzy Sets and Systems*, 100(1-3):143–158.

Hong, T.-P. and Lee, C.-Y. (1998). Learning fuzzy knowledge from training examples. In *CIKM '98: Proceedings of the seventh international conference on Information and knowledge management*, pages 161–166, New York, NY, USA. ACM Press.

Huynh, D., Jennings, N. R., R., N., and Shadbolt (2004). Developing an integrated trust and reputation model for open multi-agent systems. In *Proceedings of 7th International Workshop on Trust in Agent Societies*, pages 62–77.

Johnson, D. (2001). A theoretician's guide to the experimental analysis of algorithms. In Goldwasser, M. and Johnson, D.S. McGeoch, C., editors, *Fifth and Sixth DIMACS Implementation Challenges*, pages 215–250. American Mathematical Society.

Jones, A. J. I. (2002). On the concept of trust. *Decision Support Systems*, 33(3):225–232.

Liau, C.-J. (2003). Belief, information acquisition, and trust in multi-agent systems: a modal logic formulation. *Artificial Intelligence*, 149(1):31–60.

Maximilien, E. and Singh, M. (2002). Reputation and endorsement for web services. *ACM SIGEcom Exchanges*, pages 24–31.

Mui, L., Mohtashemi, M., and Halberstadt, A. (2002). Notions of reputation in multi-agents systems: A review. In *First International Conference on Autonomous Agents and MAS*, pages 280–287, Bologna, Italy.

Nguyen, H. and Walker, E. (1999). *A First Course in Fuzzy Logic*. CRC Press, USA.

Nozaki, K., Ishibuchi, H., and Tanaka, H. (1997). A simple but powerful heuristic method for generating fuzzy rules from numerical data. *Fuzzy Sets Systems*, 86(3):251–270.

Nute, D. (1993). Handbook of logic in articial intelligence and logic programming. In D., G., C., H., and J., R., editors, *Defeasible logic*, volume 3. Oxford University Press.

Pal, T., Pal, N. R., and Pal, M. (2003). Learning fuzzy rules for controllers with genetic algorithms. *International Journal of Intelligent Systems*, 18:569—592.

Poli, P. and Hexmoor, H. (2002). Effect of power on trust and autonomy. In *Proceedings of AAMAS 2003*.

Resnick, P. and Zeckhauser, R. (2000). Trust among strangers in internet transactions: Empirical analysis of ebay's reputation system. In *Working Paper for the NBER Workshop on Empirical Studies of Electronic Commerce*.

Rozich, R., Ioerger, T., and Yager, R. (2002). FURL - a theory revision approach to learning fuzzy rules. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, pages 791–796.

Sen, S. and Dutta, P. S. (2002). The evolution and stability of cooperative traits. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1114–1120. ACM Press.

Stranders, R. (2004). Trust models in multi-agent systems. Technical report, TU Delft.

Wooldridge, M. and Jennings, N. (1995). Intelligent agents: theory and practice. *The Knowledge Engineering Review*, 10(2):115–152.

Yager, R. R. (1993). On the hierarchical structure for fuzzy modeling and control. *IEEE Transactions on Fuzzy Systems*, 23:1189–1197.

Yu, B. and Singh, M. P. (2002). An evidential model of distributed reputation management. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 294–301. ACM Press.

Yu, B. and Singh, M. P. (2003). Detecting deception in reputation management. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 73–80. ACM Press.

Zadeh, L. (1996). Fuzzy sets. In *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A. Zadeh*, pages 19–34. World Scientific Publishing Co., Inc., River Edge, NJ, USA.