Learning to Improve Negotiation in Semi-Cooperative Agreement Problems

Elisabeth Crawford

CMU-CS-09-111
May, 2009

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Manuela Veloso, Chair
Manuel Blum
Stephen Smith
Barbara Grosz (Harvard University)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by SRI International under grant number 55-000691 and 71-000152, and Department of the Interior under grant number NBCHC030029. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Multiagent Negotiation, Multiagent Learning, Machine Learning, Personal Assistant Agents

For David, my partner in all things.

Abstract

In agreement problems, agents must assign options to variables, typically via negotiation, in order to gain reward. For an assignment to occur, agents with different individual preferences must agree on an option. We provide a formalism for agreement problems and negotiation, which encompasses important real-world problems faced by people daily, e.g., selecting a meeting time, deciding on a location for a group lunch, and assigning roles in a joint task.

We focus on the challenge of designing algorithms to improve agent negotiation. We formalize the negotiation process as a series of rounds, where the agents each make an offer until they reach agreement. The offers agents make, and hence the outcome of a negotiation, is influenced by the agents' negotiation strategies, preferences and utility functions. As an agent negotiates with other agents, it can keep a record of the offers the agents made, and the negotiation round in which they were made. We address the challenge of designing algorithms to improve agent negotiation, by providing learning algorithms, which agents can use to inform, and thereby improve, their negotiation. In particular, we show how an agent can learn complex models of its own user's preferences, by observing the outcomes of negotiations. We also address the problem of modeling the properties of other agents from negotiation histories. This problem is particularly challenging, because negotiation histories do not provide sufficient information to reconstruct a complete model of an agent. As such, we have developed a domain independent approach, to designing and learning abstractions of the properties of other agents. We demonstrate that an agent can learn these abstractions online while negotiating, and use the learned abstractions to increase the efficiency of its negotiation, without sacrificing its individual preferences. We also provide an algorithm for agents to model the reward they receive from different negotiation strategies. When an agent is faced with the problem of dealing with a wide variety of negotiation behavior in other agents, we show that the agent can improve its negotiation performance by using experts algorithms to adapt its selection of negotiation strategies according to their performance.

We observe that agreement problems are neither fully cooperative (agents have preferences), nor fully competitive (agents must agree to receive reward). We define semi-cooperative agents, to capture this space between cooperation and self-interest. The utility function of a semi-cooperative agent, trades-off preference satisfaction, and the time taken to reach agreement. The degree of the trade-off depends on the individual agent. We show how semi-cooperative agents can estimate the

utility of different negotiation outcomes, by learning online about other agents' behavior. We also provide an in-depth analysis of two different strategies for using the learned information.

The approaches presented apply to a wide range of agreement problems. We provide analytical and experimental analysis to demonstrate their effectiveness in representative domains.

Acknowledgments

I am very grateful for the support of my advisor, Manuela Veloso. It has been an honor to work with someone of her vision, enthusiasm, intellect and integrity. I have learnt a great deal from working with her and the members of the wonderful research lab (CORAL) that she has nurtured at CMU. I am grateful to all the members of CORAL for the very productive and friendly group we have had over the years, and particularly to Douglas Vail, Sonia Chernova, Colin McMillen and Nick Armstrong-Crews for their feedback, discussions, and friendship. I would also like to thank Jay Modi, Steve Smith and Jean Oh who worked with Manuela and I on the CMRadar project, where we explored meeting and room scheduling problems. Finally I would like to thank my wonderful thesis committee - Manuel Blum, Steve Smith and Barbara Grosz who have given me great feedback on my work, and inspired me with their own.

This thesis would not have been possible without the help of friends. Many thanks go to my friend Virgi, with whom I took many courses, TA-ed, and played a whole lot of sport. Also to my other awesome exercise buddies, Katrina and Kelly. Thanks too are due to our entire basketball team/s. We rocked, even if we never won a game! I got very lucky with officemates at CMU - Jon and Andy always made interesting conversation (at the right times). Basically they were perfect officemates. Many thanks also go to my other friends here, who I have not mentioned somewhere else in this acknowledgment, including: Shobha, Alice & Dave, Ryan, James, Dan, Kristen and Edi.

Finally I want to thank my parents for all their love, support, and good sense over the years. And Mike and Kelly for their friendship, help, and good humor. Most importantly I want to thank my partner David who has shared in all aspects of my life and direction for the last 13 years, has given me endless support, and who brings me my happiness.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

We study the problem of multiple agents needing to reach agreement subject to their privately owned constraints and preferences. Most existing approaches to modeling agent interaction either represent the agents as fully cooperative, or instead rationally self-interested. Neither model accurately captures the trade-off that occurs between self-interest and cooperation when people are negotiating an agreement problem. We define a model of semi-cooperative utility that allows our agents to trade-off their individual preferences with the length of the negotiation. We observe that as agents negotiate agreements, they reveal some limited information about their preferences and constraints. One of the main focuses of the thesis is to introduce methods for learning online about other agents from the way they negotiate. We show how agents can use what they learn about each other, to negotiate more efficiently, and achieve higher utility than agents that do not learn.

1.1 Agreement Problems and Thesis Approach

We outline our definition of agreement problems which we have designed to encompass important real-world agreement problems. Our research is motivated by the potential of computer agents to solve the many time consuming agreement problems currently faced by people. There are many challenges associated with building agents to solve agreement problems. We show how this thesis addresses some of these challenges.

1.1.1 Real-world Agreement Problems

Agreement problems arise in many real-world scenarios.

- **Meeting Scheduling:** Scheduling meetings is a daily and time-consuming task for many computer users. Meetings are often scheduled through negotiation, via email. A group can find it difficult to find a meeting time, because different people often have different preferences about when to hold the meeting, and people also have constraints about when they can meet - people can't attend two meetings at the same time.
- **Location Agreement:** Location agreement problems arise whenever multiple people need to agree on the place of meeting - e.g., where to eat dinner, where to spend Saturday night etc. This example also features preferences, and constraints e.g., the participants need to be able to reach the location in time.
- **Price Agreement** The problem of determining prices for products and services is an important multiagent agreement problem. Increasingly commerce is being conducted over the internet. People now negotiate prices for the sale of goods over email, via sites like craigslist, and negotiate prices for their services on consulting sites.
- **Role Assignment in Multiagent Tasks:** The completion of some tasks can require multiple people (or robots). In such cases, it is typical for the people to agree upon an assignment of roles. People often have preferences about the roles they undertake, and perhaps constraints on what roles they can perform. For example, suppose a group of TAs is dividing up the grading of an exam. There may be a question one of the TAs is not comfortable grading, and in general TAs enjoy grading different types of questions. In order to complete the grading task, the TAs agree upon an assignment of roles (who grades which questions).

1.1.2 Defining Agreement Problems

One of our contributions is a formalization of agreement problems that captures the important characteristics of real-world agreement domains. We view an agreement problem as consisting of the following elements:

- A set of *agents*.
- A set of *variables* (sometimes called issues). Each variable has a set of *decision maker* agents drawn from the set of all agents.
- A set of *options* (sometimes called values).

To completely solve the agreement problem, each variable must be assigned an option. *To assign an option to a variable, all the variable's decision maker agents must agree on the assignment.* We study *incremental* agreement problems where instead of receiving the set of variables upfront, variables are introduced to the agent system over time. Each variable must be assigned an option when it is introduced.

In most real-world agreement problems, options are assigned to variables via negotiation. In this thesis, negotiation is the approach agents use to reach agreement. The negotiation process works as follows:

- Decision maker agents negotiate by exchanging *offers* (sets of options they are willing to agree to), until *all the agents have proposed the same option.*
- A *negotiation protocol* is used to define the set of rules that specifies the types of messages the agents can exchange, and general rules for the negotiation. We use a negotiation protocol that operates in *rounds.*
- Each variable has an *initiator* agent that is designated to start the negotiation.
- A *round* consists of an offer by the initiator to the other decision making agents, and their offer in reply back to the initiator.
- Agents decide which options to offer in each round of the negotiation using an algorithm, which we call their *negotiation strategy.*

A simplified version of our negotiation protocol is given in Algorithm 1.1.1.

Algorithm 1.1.1 Simplified negotiation protocol for negotiating a variable v

```
round = 0
while no intersection in agent offers do
    initiator makes an offer to the other decision makers
    the other decision makers reply with an offer to the initiator
    round = round + 1
return intersection
```

As we noted in discussing the real-world examples, *agents have individual preferences* over the outcomes, i.e, which options are assigned to the variables. Furthermore, we noted that agents can have *constraints* about the way options are used. For instance, in the meeting scheduling problem, an agent cannot agree to two meetings for the same time-slot. Differing agent preferences, means agents desire different option-variable assignments. This raises one of the central question of agreement problems - how do agents reach agreement in the presence of individual preferences? Importantly, agreement problems are not fully competitive. In order to receive a reward/pay-off from the agreement process the agents need to reach an agreement - potentially compromising their individual preferences.

To represent the middle ground that the agents must occupy, between self-interest and cooperation, we contribute a definition of *semi-cooperative agents.* We define the utility a semi-cooperative agent receives from an agreement to be a function of its *reward from the option assignment (according*

Domain	Variables	Decision Makers	Options	Preferences	Constraints
Meeting Scheduling	meetings	meeting participants	time slots	preferences about meeting times	cannot have two meetings at the same time
Location Agreement	scheduled meetings requiring a location	meeting participants	possible locations	preferences about locations	location must be reachable in time from previous meeting
Price Agreement	item to for sale	buyer and seller	prices	seller wants high price, buyer wants low price	can have min sell, max buy prices
Role Assignment for Tasks	task	participants	roles	preferences for different roles	some agents can't perform some roles

Table 1.1: Formalism of example agreement problems

to its preference function) and the number of rounds it took to assign the option to the variable. For two options of the same reward, a higher round number means a lower utility value. The relative weight placed on the reward, versus the length of the negotiation varies between agents. Less cooperative agents will place more weight on the reward component.

To see how our formalism of agreement problems relates to real-world agreement domains, see Table 1.1. In Table 1.1 we show how the terminology we have introduced maps to the different domains we have described.

1.1.3 Thesis Approach

The central challenge in developing agents for agreement problems is solving the problem of how they should negotiate. Which options should an agent offer for a particular variable, and when? There are many challenging problems that pertain to answering this central question. In this thesis we focus on developing learning algorithms, so that agents can make better informed negotiation decisions. In particular, we address the following questions.

- *How can an agent learn from seeing the results of negotiations?*
  We show that if an agent is allowed to observe the current state of the variable assignments, and the assignment of options to new variables over time, that it can learn a model of the preferences which influenced the assignment. We apply our algorithm to the problem of an agent learning its *own user's preferences* by observing the user make variable assignments.

Our approach does not assume it can access the process that caused the assignments, making it more practical than many existing approaches. We show that we can learn a model of user preferences that takes into account preferences that depend on existing option assignments. For instance, a person's preference, in the meeting scheduling domain, for having meetings scheduled back-to-back.

- *How can an agent model properties of other agents from negotiation histories?*
  - We model preference and utility functions as being *privately owned* by the individual agents. Further, we consider information about variables that have already been assigned as privately owned by the agents involved in the variables. While this information is privately owned and not open to the public, it nonetheless influences the negotiation process. In particular, we observe that some information is necessarily revealed during negotiations. We show that agents can build *useful abstractions* of each other using the revealed information.
  - In addition to providing a method for learning abstractions of other agent's properties, in particular their preferences and constraints, we show how agents can use the abstractions to improve negotiation efficiency and thus their utility.

- *How can an agent learn to improve its negotiation when negotiating in a system where the behavior of other agents can vary greatly?* We show how in this situation, that instead of employing a single negotiation strategy and trying to improve performance by modeling the properties of other agents, an agent can instead improve its performance by *modeling the reward it receives from using* **different** *negotiation strategies.* Given a diverse set of negotiation strategies, we show how an agent can learn to choose which strategy works best with each of the agents it negotiates with - adapting its strategy selection if another agent starts behaving differently.

- *In a semi-cooperative agent setting, how can agents learn to increase their utility from negotiation?*
  - When we can assume or enforce a more bounded set of behaviors on the part of the agents, we can use different types of approaches. We have defined semi-cooperative agents to capture the balance between self-interest and cooperative tendencies that arise in real-world agreement problems. We study how semi-cooperative agents can learn to increase their utility in a setting where we assume, at a minimum, somewhat cooperative behavior on behalf of all the agents e.g., we use a negotiation protocol that requires each agent to offer at least one new option in each round of the negotiation.
  - With the aim of providing an agent with useful information for increasing its semi-cooperative utility, we contribute a method for *learning to predict in what round an agent will offer different options.* This model is learnt online from observing negotiation histories. We study two different approaches to exploiting the learned models online that directly attempt to increase the learning agent's utility. We demonstrate experimentally

that when all the agents attempt to increase their individual semi-cooperative utility, better system-wide outcomes result than without learning.

In addressing each of these questions, we show how agents can learn to improve their negotiation, in multiple ways, and under different conditions, in multiagent agreement problems.

1.2 Contributions and Reader's Guide

In this thesis we provide a formalization of agreement problems that captures the key features of real-world agreement problems. The algorithms we develop are useful across many different domains. We specifically demonstrate their flexibility by studying them in detail on two very different agreement problems: the multiagent meeting scheduling problem, and the role assignment for multiagent tasks problem.

More specifically this thesis makes the following contributions:

- Formalization of agreement problems and a negotiation process that encompasses many different types of domains (Chapter 2).
- Definition of Semi-Cooperative agents that effectively captures the conflict between the reward agents receive from satisfying their preferences over the options, and their desire to negotiate efficiently (Chapter 2).
- Theoretical analysis of negotiation in agreement problems, focusing on the behavior of particular negotiation strategies. (Chapter 3).
- Algorithm that allows an agent to learn its own user's option preferences. These option preferences can depend on the user's/agent's state, not just on the option being assigned. We demonstrate the effectiveness of the learning algorithm in the multiagent meeting scheduling domain (Chapter 4).
- A domain independent approach for building abstractions of the properties of other agents from negotiation histories. We demonstrate the usefulness of learning and using these abstractions to improve negotiation online, in the multiagent meeting scheduling problem (Chapter 5).
- A method for strategy selection when an agent is faced with the problem of deciding what negotiation strategy to use with another agent of unknown properties. We show that we can provide an agent with a small set of diverse strategies, and that the agent can learn to choose between the strategies using experts algorithms, in a way that results in increased utility for the agent (Chapter 6).

- A method for predicting, from negotiation histories, the round in which other agents will offer different options. We give two algorithms for exploiting this learned information that directly attempt to increase the agent's utility according to its semi-cooperative utility function. We demonstrate the effectiveness of this approach in a syntactic agreement domain (Chapter 7).
- A detailed analytical and controlled experimental analysis of the approaches developed in Chapter 7 to the multiagent meeting scheduling problem and the role assignment for multiagent tasks problem, (Chapter 8).
- An extension of the analysis in Chapter 8 to a larger scale (e.g., more agents, options) and under additional conditions (Chapter 9).

We provide an analysis of the related work on agreement problems in Chapter 10.

Chapter 2

Definition of Agreement Problems

In this chapter, we define the Extended Incremental Multiagent Agreement Problem with Preferences (EIMAPP). In EIMAPPs, variables, v, arise over time. For each variable, the set of decision makers, $\mathcal{D}_v$ negotiate in order to agree upon an option, o, from the set of all options, $\mathcal{O}$ to assign to v. If the agents reach an agreement, they each receive a reward according to their individual utility functions U_a. Each agent's utility function is privately owned. In this thesis, we look at a number of different utility functions for individual agents. These utility functions are generally based on both the agent's preferences and the length of negotiation. In this chapter, we include a discussion of utility functions that have two components: a preference function, p_a that specifies a's payoff for assigning v to each of the different options; and a constant γ_a, which controls a's utility for compromising its preferences as the number of negotiation rounds r increases. We call this utility function semi-cooperative and use it to define Semi-Cooperative EIMAPPs.

Our definitions in this chapter build upon the Incremental Multiagent Agreement Problem (IMAP), introduced by Modi & Veloso [36]. In particular, we extend it to cover a wider class of values/options, and to include agent preferences and utilities.

2.1 Negotiated Agreement Problems

In negotiated agreement problems, agents reach agreement via negotiation. A *negotiation protocol* is a set of rules, known to all the agents, that governs the mechanics of the negotiation. In this thesis, we use round-based negotiation protocols, where once an option is offered, it remains on the table. Agreement is reached when there is an intersection in the options the agents have offered.

The following is an example of the execution of a round-based protocol between two agents.

Example 2.1.1. *Agents a_1 and a_2, negotiate variable v:*

- *Round 1: a_1 offers o_1; a_2 offers o_4, o_5*
- *Round 2: a_1 offers o_6, o_7; a_2 offers o_3*
- *Round 3: a_1 offers o_8; a_2 offers o_{12}*
- *Round 4: a_1 offers o_9; a_2 offers o_7*
- *An intersection has been reached: o_7*

Note that the agents offer different numbers of options, and the negotiation ends once they have both offered the same option. This option was first proposed by a_1 in round 2, but a_2 counter proposed o_3 and o_{12} before agreeing to o_7 in round 4. In this example, neither agent achieved one of the options it proposed in the first round.

2.2 Problem Definitions

We introduce the Extended Incremental Multiagent Agreement Problem (EIMAP) to capture the important features of negotiated agreement problems, in particular the incremental nature of the process. For instance, in a typical agreement problem, such as meeting scheduling, a person does not negotiate their entire calendar with everyone at the beginning of the month, rather they negotiate the scheduling of each new meeting as it arises.

2.2.1 Extended Incremental Multiagent Agreement Problem

We formally define EIMAP according to its components.

Definition An **Extended Incremental Multiagent Agreement Problem (EIMAP)** is a tuple : $< \mathcal{A}, \mathcal{O}, \mathcal{V}, \mathcal{T} >$. Where:

- $\mathcal{A}$ is a set of agents.
- $\mathcal{O}$ is a set of values/options.

- $\mathcal{V}$ is a set of variables. For each variable, v, there is a set of decision maker agents $\mathcal{D}_v \in \mathcal{A}$. Each variable has an *initiator* agent. The initiator agent i is an agent in $\mathcal{D}_v$ with a special job with regards to v. When a variable v arises, i_v is notified of its occurrence. It is then $i'_v s$ job to start the negotiation as soon as it is notified about v. The initiator, i_v, has a unique role in the negotiation process - it sends negotiation proposals to all the other agents in $\mathcal{D}_v$, collects their responses, and determines when the negotiation is complete.
- $\mathcal{T}$ is the distribution (and associated parameters) that determines the arrival of new variables over time. The variables in $\mathcal{V}$ are not provided up-front. This gives rise to the *incremental* aspect of the problem. In some cases $\mathcal{T}$ may be unknown to the agents.

2.2.2 Solving EIMAPs

We define the negotiation process through which EIMAPs are solved as a tuple, $\mathcal{N} =< \mathcal{R}, \omega, \mathcal{M}, \mathcal{H} >$, where $\mathcal{R}$ restricts how options can be assigned to variables, ω defines the negotiation process, $\mathcal{M}$ *stores the mapping between variables and their agreed upon options*, and $\mathcal{H}$ is the history of the agent's interactions. In more detail, we define the components of $\mathcal{N}$ as follows:

- $\mathcal{R}$ is a function that can be used to restrict how options can be assigned to variables. We write $\mathcal{R}(a, \mathcal{O})$ to indicate a's *ineligible options*. For instance, in the multiagent meeting scheduling problem, we could define $\mathcal{R}(a, \mathcal{O})$: all $o \in \mathcal{O}$ s.t $(o, v') \in \mathcal{M}, \forall v'$. In other words, in this domain, the same option cannot be used for more than one variable.
- ω is a tuple, $< \mathcal{N}_p, \mathcal{N}_{\text{range}}, \mathcal{N}_{\text{rule}} >$ where:
  - $\mathcal{N}_p$ is the negotiation protocol that the agents, $\mathcal{D}_v$, use to assign an option o to the variable v when it arises. We use a round-based negotiation protocol, where once an option is offered, it remains on the table. Agreement is reached when there is an intersection in the options the agents have offered. Our general negotiation protocol is given in Algorithm 2.2.1.

    The protocol operates in rounds. In each round the initiator of the variable sends a message containing a set of options (an offer) to the other decision makers (Line 4). In the same "round", they respond by sending an offer back to the initiator. Once the initiator has received all the responses, the round ends. Negotiation concludes when there is an intersection in the agents' offers. When this occurs the initiator chooses an option from the intersection to assign to the variable. When implementing this protocol in practice, it is necessary to use standard conventions, such as time-outs, to ensure the process cannot deadlock if one of the agent's programs crashes, etc.
  - $\mathcal{N}_{\text{range}}$ is a set of integer pairs (s_a, e_a), one pair for each agent. s_a is the minimum number of options a must make per negotiation round, and e_a is the maximum. The minimum can be violated only if there are not enough eligible options according to $\mathcal{N}^a_{\text{rule}}$. (See Lines 4, 11 for use of $\mathcal{N}_{\text{range}}$)

Algorithm 2.2.1 General negotiation protocol (v)

1: Let round $r = 1$
2: For each a in $\mathcal{D}_v$, let $\text{offers}_a = \emptyset$ be the set of options a has offered
3: **while** $\bigcap_{\mathcal{D}_v} \text{offers}_a == \emptyset$ **do**
4: initiator, i, offers $i_r \subset \mathcal{R}(i, \mathcal{O})$ to the other agents in $\mathcal{D}_v$, s.t $i_r \subset \mathcal{N}_{\text{rule}}$ and $s_i \leq |i_r| \leq e_i$.
5:
6: **for** each $o \in i_r$ **do**
7: $\mathcal{H} = \mathcal{H} \cup (o, r, i)$
8: $\text{offers}_i = \text{offers}_i \cup i_r$
9:
10: **for** each $a \in \{\mathcal{D}_v - i\}$ **do**
11: offers $a_r \subset \mathcal{R}(a, \mathcal{O})$ to the initiator, s.t $a_r \subset \mathcal{N}_{\text{rule}}$ and $s_a \leq |a_r| \leq e_a$.
12:
13: **for** each $o \in a_r$ **do**
14: $\mathcal{H} = \mathcal{H} \cup (o, r, a)$
15: $\text{offers}_a = \text{offers}_a \cup a_r$
16: $r+ = 1$
17: $x = \bigcap_{a \in \mathcal{D}_v} \text{offers}_a$
18: initiator selects an o_c from x to assign to v and notifies all $a \in \{\mathcal{D}_v - i\}$.
19: $\mathcal{M} = \mathcal{M} + (v, o_c)$.

  - $\mathcal{N}_{\text{rule}}$ is a set of rules, one for each agent, that determines which options, in $\mathcal{R}(a, \mathcal{O})$, are eligible to be offered. For example, the rule for agent a, $\mathcal{N}^a_{\text{rule}}$, could specify that a can't repeat the offer of any options, i.e, every option offered for v in round r, must not have been offered for v before r. (See Lines 4, 11 for use of $\mathcal{N}_{\text{rule}}$)

- $\mathcal{M}$ is the mapping from variables to agreed upon options, $\mathcal{M} : \mathcal{V} \rightarrow \mathcal{O}$.
- $\mathcal{H} = \bigcup_v \mathcal{H}_v$, where $\mathcal{H}_v$ is a set containing a tuple for each offer made by each $a \in \mathcal{D}_v$. The tuple consists of the set of options offered, the agent name, and the round of the offer.

Definition A valid EIMAP solution is an assignment of a subset of $\mathcal{V}$ to options/values via a set of valid executions of the negotiation procedure $\mathcal{N}$.

In general, every time a new variable, v, arises, the agents $\mathcal{D}_v$ execute the negotiation procedure to *augment* the mapping, $\mathcal{M}$, from variables to agreed upon options. The negotiation process produces the histories, $\mathcal{H} = \bigcup_v \mathcal{H}_v$. These histories can be used to inform the negotiation of each agent. Later in this thesis we will discuss which parts of the history each agent can see. In some domains it may be possible to reassign some variables up to a deadline (e.g., reassign time-slots to meetings that have not yet occurred). Thus, the mapping $\mathcal{M}$ may change over time as well as be augmented.

Example Revisited

We can now formally define the EIMAP for the example we gave at the start of this chapter.

- $\mathcal{A} = \{a_1, a_2\}$
- $\mathcal{O} = \{o_1, o_2, ..., o_{12}\}$
- $\mathcal{V} = \{v\}$
- $\mathcal{T}$: if time is 1 then v (for now we just ignore the possibility of future variables).
- $\mathcal{R} = \emptyset$
- $\mathcal{N}_p$ is the protocol given in 2.2.1.
- $\mathcal{N}^{a_1}_{\text{range}} = (1, 2)$ and $\mathcal{N}^{a_2}_{\text{range}} = (1, 2)$
- $\mathcal{N}^{a_1}_{\text{rule}}$ = every option offered for some variable v, must not have been offered for v before. Same rule applies for a_2.
- $\mathcal{M}$ starts out as empty, and once the negotiation ends we have $\mathcal{M} = \{(v, o_7)\}$
- The history at the end of the negotiation is:

$$\begin{aligned}\mathcal{H} \;=\; & \{((\{o_1\}, 1, a_1), (\{o_4, o_5\}, 1, a_2), (\{o_6, o_7\}, 2, a_1), (\{o_3\}, 2, a_2),\\ & (\{o_8\}, 3, a_1), (\{o_{12}\}, 3, a_2), (\{o_9\}, 4, a_1), (\{o_7\}, 4, a_2)\}\end{aligned}$$

Solution quality

While we have defined valid solutions to EIMAP, we have said nothing about which solutions are more desirable. One obvious measure of quality, is the *size of the mapping* $\mathcal{M}$, the more variables assigned, the better the solution. However, in many agreement domains agents have preferences about option use. Our definition of EIMAP, like IMAP [36] does not include a built-in notion of agent preferences or utility. The goal in IMAP [36] is to maximize the number of assignments, subject to the constraint that an agent cannot use the same option more than once. In contrast our definition of EIMAP allows for arbitrary rule-based constraints to be expressed through $\mathcal{N}_{\text{rule}}$.

In the next section, we will extend our definition of EIMAP (and thus our notion of solution quality) to consider agent preferences.

Examples of EIMAPs

EIMAP is very expressive. It is designed to represent a large variety of real-world agreement problems. We give two examples here, (i) agreeing on locations for a regular group meeting, and (ii) scheduling meetings.

Example 2.2.1. EIMAP for location agreement problem

- $\mathcal{A} =$ *participants of the group meeting*
- $\mathcal{O} = \{$*Crazy Mocha, WeH 7200, Panera Bread*$\}$
- $\mathcal{V} =$ *regular meetings* m
- $\mathcal{D}_m = \mathcal{A}$*, the initiator of* m *is the meeting moderator.*
- $\mathcal{T} =$ *variables* m *arise with the frequency of the group's regular meetings.*

We define the negotiation solution process for the *location agreement EIMAP* as follows:

- $\mathcal{R} = \emptyset$
- $\mathcal{N}_p =$ Algorithm 2.2.1.
- $\mathcal{N}^a_{\text{range}} = (1, 1), \forall a \in \mathcal{A}$
- $\mathcal{N}^a_{\text{rule}} =$ every option offered for some variable v, must not have been offered for v before, $\forall a \in \mathcal{A}$.
- $\mathcal{M}$ starts out as empty and over time will contain pairs such as $(m_1, \text{WeH 7200})$.
- $\mathcal{H}$ records the negotiation history.

Note that the set $\mathcal{R}$ is empty. In this problem there is not necessarily a restriction on option usage. Options can be used for variables an unlimited number of times. There however could be constraints on options for particular agents. For instance if a person has a gluten allergy, they may have a constraint that says they cannot eat at Panera Bread.

Example 2.2.2. EIMAP for meeting scheduling

- $\mathcal{A} =$ *all the agents who schedule meetings*

- $\mathcal{O} = \{Mon\ 9am, Mon\ 10am, ...Fri\ 5pm\}$. *This is an example of how time could be discretized.*
- $\mathcal{V} =$ *meetings* m.
- $\mathcal{D}_m \subset \mathcal{A}$, *initiator of* m *is the meeting initiator.*
- $\mathcal{T}$ *: variables* m *with their associated participants arise as people initiate them.*

We define the negotiation solution process for the *meeting scheduling EIMAP* as follows:

- $\mathcal{R}$ represents that a person cannot attend more than one meeting at the same time – $\mathcal{R}(a, \mathcal{O})$ contains all $o \in \mathcal{O}$ s.t $(o, v') \in \mathcal{M}, \forall v'$.
- $\mathcal{N}_p =$ Algorithm 2.2.1.
- $\mathcal{N}^a_{\text{range}} = (1, 1), \forall a \in \mathcal{A}$
- $\mathcal{N}^a_{\text{rule}} =$ every option offered by a for v, must not have been offered by a for v before. Same rule applies $\forall a \in \mathcal{A}$.
- $\mathcal{M}$ starts out as empty and over time will contain pairs such as $(m_1, \text{Wed 3pm})$.
- $\mathcal{H}$ records the negotiation history.

A key difference between the meeting scheduling EIMAP and the location agreement EIMAP is that the options in the meetings scheduling EIMAP cannot be reused by the *same agent* for different variables. Another difference, is that in the meeting scheduling EIMAP the variables have different participants, but in the location agreement EIMAP all the agents are always involved in every variable.

2.2.3 EIMAP with Preferences

In many IMAP and EIMAP domains, the agents have preferences about which options are assigned to variables. For instance, in a multi-robot task domain, some options for completing a task might be more expensive for a particular robot than others. To represent such domains, we define EIMAP with preferences – EIMAPP.

Definition *EIMAP with Preferences, EIMAPP*, is an instance of EIMAP augmented by a set of preference functions, one for each agent : $< \mathcal{A}, \mathcal{O}, \mathcal{V}, \mathcal{T}, \mathcal{P} >$. Where:

- $\mathcal{P}$ is a set of preference functions, p_a: $\mathcal{O} \times \mathcal{V} \times \mathcal{M} \rightarrow \Re^+$. The function, indicates a's preference for assigning o, to variable v. We focus mostly on the case where $p_a(o, v, \mathcal{M}) = p(o, v', \mathcal{M}')$ $\forall v'$ and $\forall M'$.

EIMAPP examples

Example 2.2.3. *The* **EIMAPP for the meeting location problem** *is the EIMAP for the problem described in Example 2.2.1 augmented by preference functions for each of the agents. Some example preference functions for this domain are as follows:*

- *Given* (o, v), $p_a(o, v, \mathcal{M}) = 1.0 \times (o == \textit{Panera Bread}) + 0.0 \times (o == \textit{WeH 7220}) + 0.0 \times (o == \textit{Crazy Mocha})$
- *Given* (o, v), $p_a(o, v, \mathcal{M}) = 0.2 \times o == \textit{Panera Bread}) + 0.3(o == \textit{WeH 7220}) + 0.5 \times (o == \textit{Crazy Mocha})$
- *Let* $f(o) =$ *the number of times* o *appears in* $\mathcal{M}$*, then*

$$\begin{aligned} p_a(o, v, \mathcal{M}) &= \frac{0.2 \times f(o == \textit{Panera})}{0.2 \times f(o == \textit{Panera}) + 0.3 \times f(o == \textit{WeH7220}) + 0.5 \times f(o == \textit{Crazy Mocha})} \\ &+ \frac{0.3 \times f(o == \textit{WeH7220})}{0.2 \times f(o == \textit{Panera}) + 0.3 \times f(o == \textit{WeH7220}) + 0.5 \times f(o == \textit{Crazy Mocha})} \\ &+ \frac{0.5 \times f(o == \textit{Crazy Mocha})}{0.2 \times f(o == \textit{Panera}) + 0.3 \times f(o == \textit{WeH7220}) + 0.5 \times f(o == \textit{Crazy Mocha})} \end{aligned}$$

Example 2.2.4. *The* **EIMAPP for the meeting scheduling problem** *is the EIMAP described in Example 2.2.2 augmented by preference functions for each agent. The following are some example preference functions we could use:*

- *Let morn and arv be functions representing disjoint sets (morning and afternoon). Given* (o, v)*,*

  $$p_a(o, v, \mathcal{M}) = 0.8 \times \textit{morn}(o) + 0.2 \times \textit{arv}(o)$$

  where $\textit{morn}(o)$ *returns* 1 *if* $o \in \textit{morn}$ *and* 0 *otherwise, and similarly for arv.*
- *A function like the previous function, but with an increased preference for using options that are adjacent in time to options already in* $\mathcal{M}$ *i.e., with a preference for back-to-back meetings.*
- *A function with exceptions for various participants, e.g., if Alice is not present then the previously described* p_a *is used, but if Alice is present then* $p_a(o, v, \mathcal{M}) = 1$*.*

Solution Quality for EIMAPPs

A number of different quality metrics can be considered for EIMAPPs. We list some of them here. We discuss what quality metrics are the most suitable under different conditions in the next chapter.

- Let $\mathcal{M}^{-v}$ be the state of $\mathcal{M}$ prior to the addition of v. The *social welfare* of adding the assignment o, v is given by $\text{sw}(\mathcal{M}) = \sum_a p_a(o, v, \mathcal{M}^{-v})$. To evaluate the average social welfare value of all assigned variables in $\mathcal{M}$ we can sum for each (o, v) in $\mathcal{M}$ the preferences of the decision maker agents and divide by the number of variables.

- Another metric is whether the mapping $\mathcal{M}$ is *Pareto Optimal*. An outcome of a process, z, is Pareto Optimal, if there is no other outcome z', such that at least one agent is better off with z' and no agent is worse off with z'. Thus, a mapping $\mathcal{M}$ is Pareto Optimal if $\nexists \mathcal{M}'$, s.t $\exists a \in \mathcal{A}$, s.t $p_a(\mathcal{M}') > p_a(\mathcal{M}) \bigcap p_b(\mathcal{M}') \geq p_b(\mathcal{M})\ \forall b \in \mathcal{A} b \neq a$. In determining the Pareto Optimality of a mapping we have the option of evaluating the Pareto Optimality of each new addition to the mapping as it comes, while considering the rest of the mapping as given, versus, evaluating the Pareto Optimality of the entire mapping. In this thesis, since our focus is on incremental agreement problems, we will generally use the former definition.

- Without preferences the most obvious quality metric was the number of assignments - the size of $\mathcal{M}$. Thus a possible metric for EIMAPP would be to combine maximizing size with a preference based metric, or to trade off size and preference metrics.

2.2.4 Semi-Cooperative EIMAPP

In many agreement problems efficiency is important, e.g., when agents communicate with humans. Furthermore, users may want agents representing them to compromise as they would themselves. Grosz et al. [26] have demonstrated, that humans negotiating with computer agents expect co-operative behavior and that some co-operative behavior is beneficial to the agents.

We define *Semi-Cooperative EIMAPPs* to address the dual concerns of preference satisfaction and cooperation. We model the agents as having a self-interested component to their utility - the reward according to the agents' preference functions. This reward is discounted over the negotiation rounds to represent the agents' cooperative tendencies.

Definition *Semi-Cooperative EIMAPP, (SC-EIMAPP)*, is an instance of EIMAPP augmented by a set of semi-cooperative utility functions and a semi-cooperative behavior rule, one for each agent: $< \mathcal{A}, \mathcal{O}, \mathcal{V}, \mathcal{T}, \mathcal{P}, U, \zeta >$. Where:

- U is a set of utility functions, $U_a : \mathcal{O} \times \mathcal{V} \times I^+ \rightarrow \Re^+$, where the third parameter is a positive integer round, r, at which the negotiation ends with the agreement of o for v. More specifically we consider U_a as given by the following function:

$$U_a(o, v, r) = \gamma_a^r p_a(o, v), \gamma_a \in (0, 1)$$

- ζ is a set of values, ϵ_a, one for each agent, that specify the minimum estimated utility the agent will delay agreement to try to achieve. We define the role of the ζ values formally in the *semi-cooperative behavior rule*.

Agents with a high cost for negotiation time (low γ) are incentivized to compromise earlier (by agreeing to an offered option) than agents with a low cost for time (high γ).

Note: in some of the work that appears later in this thesis we trade-off option preferences and negotiation rounds in a linear utility function. The linear utility function requires two constant parameters, α and β

$$U_a(o, v, r) = \alpha \ \text{pref}_a(o, v) - \beta r$$

.

Due to privacy considerations, it is not desirable for agents to simply offer many options to reach agreement quickly. Restrictions on the minimum and maximum number of options offered per round can be set by the user in $\mathcal{N}_{\text{range}}$. We will assume that the agents have a maximum number of new options they can offer per round (e.g., set by the agent's user).

Definition Semi-Cooperative behavior rule - For a variable, v, agent a selects an option o from $O_{\text{un-Offered}_a}$ (the options it has not yet offered for v) to offer in round r as follows.

$$o_v(r) = \begin{cases} \underset{o \in O_{\text{un-Offered}_a}, r' > r}{\text{argmax}} \quad U(o, v, r') \text{ if } \exists o, r' : U(o, v, r') \geq \epsilon_a \\ \underset{o \in O_{\text{un-Offered}_a} \& \nexists o' \text{offered by more agents}}{\text{argmax}} \quad p_a(o). \end{cases}$$

In other words, the agent offers the option it believes will maximize its utility, if the utility will exceed the parameter ϵ_a, for some r'. The round r' is the estimated (from learning) round the option will be agreed upon. If the agent cannot achieve utility greater than ϵ_a, then it seeks to reach an agreement quickly, by selecting from amongst the options that have been offered *by the most agents*, the option it prefers.

Negotiation with discount factors has been looked at in game theory, e.g., in the context of the split the pie game [44]. Game theoretic equilibrium analysis (as a method for informing agent behavior) is most applicable when there is *complete information* or when the agents share accurate prior beliefs about type distributions.

SC-EIMAPP Example

Example 2.2.5. *The* **SC-EIMAPP for the meeting location problem** *is the EIMAPP for the problem described in Example 2.2.3 augmented by γ and ϵ values for each of the agents. If agent*

a values achieving a high preference location strongly and does not mind very much how long the negotiation takes, a would have a high γ value such as 0.9 and a ϵ value relative to their preference range. If a wants to be more cooperative, then a may choose a γ value such as 0.5, and a low ϵ value so it will be a while before a will simply start trying to agree. a would set values similarly in the meeting scheduling scenario.

Solution Quality for SC-EIMAPPs

We can consider a number of different quality metrics, including:

- Social welfare in terms of the agents' semi-cooperative utilities
- Pareto Optimality, as each option is assigned, in terms of the agents' semi-cooperative utilities.
- A trade-off between maximizing semi-cooperative utilities and the number of variables assigned, etc.
- Semi-Cooperative utility from the perspective of individual agents

When evaluating the strategies of semi-cooperative agents we generally focus on individual-agent metrics of solution quality. In particular, we look at the average utility to each agent of reaching agreements. We focus on individual metrics because we are evaluating a negotiation system, where each agent acts independently motivated by their own personal, privately owned utility function. However, we will see that when agents have semi-cooperative utility functions and use these to motivate their negotiation decisions we get good outcomes for the system as a whole e.g., in terms of social welfare.

2.2.5 Summary

In this chapter, we formally defined agreement problems. Our definition encompasses the incremental nature of agreement problems, agent preferences, and a formal notion of semi-cooperative utility. In the next chapter we contribute some theoretical analysis of agreement problems.

Chapter 3

Analysis of Agreement Problems

In this chapter, we formally analyze agreement problems. Our focus is on understanding agreement problems in the context of multiagent negotiation.

3.1 Analysis of Negotiation Strategies

A ***Negotiation Strategy*** is an algorithm that subject to a negotiation protocol, ($\mathcal{N}_p$ in the notation of Chapter 2) and other restrictions (e.g., $\mathcal{N}_{\text{rule}}, \mathcal{N}_{\text{range}}$) determines the agent's actions. The function of negotiation strategies, in our protocols is to determine what options to offer in each round of the negotiation. While the protocol is public knowledge, each agent's negotiation strategy is privately owned. There is a large range of possible negotiation strategies. For the protocol, defined earlier in Algorithm 2.2.1, where agents must offer a new option each round, the space of all negotiation strategies that do not depend on the offers of the other agents, is every possible ordering of options. If there are k options, then there are $k!$ distinct strategies. We analyze what happens when agents use certain types of strategies.

One reasonable strategy, is to offer options in order of preference.

Definition *1-Preference-Order Strategy*: Offers one option per round. If $p_a(o_i) > p_a(o_j)$, o_i offered before o_j. If $p_a(o_i) = p(o_j)$ then a offers the option already offered by the largest number of agents, tie-breaking in favor of the earliest offered option. Where possible, a Pareto-Optimal option is selected from the intersection by the initiator (assuming all agents use the strategy).

A typical performance criteria is whether a negotiation strategy leads to a *Pareto Optimal* outcome in self-play.

Definition ***Pareto Optimality in Terms of Preferences***: An outcome o_i is Pareto Optimal for v if there does not exist another outcome o_j such that for some $a \in \mathcal{D}_v$ $\text{pref}_a(o_j) > p_a(o_i)$ and $\forall b \neq a \in \mathcal{D}_v$ $\text{pref}_b(o_j) \geq p_b(o_i)$.

Theorem 3.1.1. *Let there be two agents a and b, who both use the 1-Preference-Order Strategy, then the outcome o_c is Pareto Optimal in terms of preferences.*

Proof. Let r_c be the agreement round. For a contradiction, suppose $\exists o_p$ such that $p_a(o_p) > p_a(o_c)$ and $p_b(o_p) \geq p_b(o_c)$. **Case 1**: Let a be the initiator. Since $p_a(o_p) > p_a(o_c)$, a must have offered o_p prior to r_c. If $p_b(o_p) > p_b(o_c)$ then b must have offered o_p prior to r_c. If $p_b(o_p) = p_b(o_c)$ then b must also have offered o_p prior to r_c (tie-breaking rule). Hence o_p would be the agreement. **Case 2**: Let b be the initiator. If $p_b(o_p) > p_b(o_c)$ same argument applies. If $p_b(o_p) = p_b(o_c)$ then b may offer o_c before o_p, but a will still offer o_p before o_c. This could result in o_c and o_p appearing in the intersection. However, b knows that a is either indifferent or prefers o_p, so by the Pareto-Optimality condition in the strategy, b will select o_p as the agreement. □

For two agents (offering one option a round) preference order leads to a Pareto Optimal outcome. We can extend the result to agents *offering k options*, if we require that agents *don't offer options of different preference in the same round*. For n agents, we run into difficulties.

Lemma 3.1.2. *When n agents use the 1-Preference-Order Strategy the intersection can contain a Pareto Dominated option, indistinguishable to the initiator from a Pareto Optimal option.*

Proof. We show that an intersection can contain two options o_p and o_c, s.t, $\exists$ agent i $s.t, pref_i(o_p) > pref_i(o_c)$ and $\forall$ agents $j \neq i, pref_j(o_p) \geq pref_j(o_c)$, and that the initiator cannot tell from the offers it has received that o_c is Pareto Dominated.

Let there be five agents a, b, c, d, e, where a is the initiator. The agents offer one option per round, and the order of these is as follows:

- a: o_5, o_1, o_7, o_3
- b: o_2, o_7, o_5, o_1
- c: o_3, o_5, o_1, o_2
- d: o_2, o_3, o_1, o_5
- e: o_1, o_3, o_5, o_4

In the example above, prior to round 4, there is no intersection in offers. At round 5 the intersection becomes o_1, o_5. The agents have the following preferences related to o_1 and o_5: $pref_a(o_1) =$

$pref_a(o_5)$, $pref_b(o_5) = pref_b(o_1)$, $pref_c(o_5) = pref_c(o_1)$, $pref_d(o_1) > pref_d(o_5)$ and $pref_e(o_1) = pref_e(o_3) = pref_e(o_5)$. Notice that all the offers are consistent with these preferences according to the requirements of the 1-Preference-Order strategy. Thus the intersection can contain a Pareto-dominated option, in this case, o_5. Note also, that if instead of preferring o_1, agent d was indifferent between o_1 and o_5, d's offer would still be consistent with the strategy. This is because prior to round 3, o_5 and o_1 have both been offered once in round 1, and once in round 2, meaning that the only preference ordering *inconsistent* with d's offer would be to offer o_5 before o_1. Thus, the initiator cannot tell the difference between d being indifferent and o_1 being preferred. Furthermore, the initiator cannot simply select o_1 to be 'safe', because for agent c (by a similar argument), the initiator cannot tell if c prefers o_5, or is indifferent. Thus the initiator cannot tell from the set of offers which option is Pareto Dominated. □

Corollary 3.1.3. *The outcome of n agents using the 1-Preference-Order Strategy is not always Pareto-Optimal.*

This problem could be avoided by using a protocol where only the initiator proposes options and the others just accept/reject. This could be very slow however.

Lemma 3.1.4. *If an agent offers an option out of preference order, an outcome that is not Pareto-Optimal* can *result.*

Proof. Let , a, b be agents, and a offers o_c out of order to agree with b. Then, $\exists o_p$, such that $p_a(o_p) > p_a(o_c)$. If $p_b(o_c) = p_b(o_p)$, then o_c is not Pareto Optimal. A similar argument works for greater than two agents. □

Furthermore, it is worth noting that an agent that cares only about its own preferences, can sometimes achieve a higher preference outcome by offering options out of preference order.

Lemma 3.1.5. *In some cases an agent can achieve an outcome with a higher preference value by offering options out of preference order.*

Proof. We can see this is true using an example. Let agent a have the preference ordering: $o_1, o_2, [o_3, o_4, o_5], o_6$, where the square brackets indicate options of equal preference. Let agent b have the preference ordering: $o_6, o_2, o_1, [o_3, o4, o_5]$. If agents a and b offer the options in order of preference, one option per round using the 1-Preference-Order strategy, they will agree on o_2 in round 2. However, suppose instead that agent b sticks to this strategy, but agent a offers o_4 in round 2 instead of o_2. In this case there will not be an agreement in round 2, and agent b will offer o_1 in round 3 and agent a will get an agreement for its top preference. □

Corollary 3.1.6. *In general all agents using the 1-Preference-Order Strategy does not constitute a Nash Equilibrium.*

Proof. As shown in the previous lemma, agent a benefits from changing strategy. □

In this thesis, our focus is on the behavior of semi-cooperative agents. With semi-cooperative agents, we get the following:

Lemma 3.1.7. *An agent may receive greater utility if it offers an option* **out of preference order***, than in order, when it has a SC-utility function.*

Proof. Suppose agent a offers o_1 in round 1. Agent b's utility function is $U_b(o, v, r) = \frac{1}{6}^r p_b(o)$. Suppose $p_b(o_1) = 2$ and $\forall o \neq o_1 \in \mathcal{O}, p_b(o) = 4$, then b will get a utility of $\frac{1}{3}$ for offering o_1 (out of preference order) and reaching agreement straight away. However if b offers any other option, the maximum utility b can receive is $\frac{1}{9}$. □

To design algorithms for SC-agents and evaluate them, we need an appropriate performance criteria. Unfortunately, *achieving Pareto Optimality relative to SC-utility is not realistically achievable in SC-EIMAPPs.* Let an outcome be an option round pair, e.g. $(o_i, 2)$, then Pareto Optimality relative to SC-utility is defined as previously, but using utility functions instead of preference functions.

Lemma 3.1.8. *For an outcome to be Pareto Optimal in terms of SC-utility it must be agreed upon in the first round.*

Proof. Let (o_i, r), s.t $r > 1$ be the outcome of negotiating v. By the definition of SC-utility, $U_a(o_i, 1) > U_a(o_i, r) \forall a \in \mathcal{D}_v$. □

Since agent preferences and any constraints are privately owned, we can't guarantee Pareto Optimality.

3.2 Negotiation Protocols

The negotiation protocol has a significant impact on the negotiation strategies and thus on the results we have described in this chapter. There are many possible negotiation protocols. Some examples from related work include the following:

1. Agents report preferences/utilities with their proposals (Garrido and Sycara [24] and Jennings and Jackson [28]) and the initiator uses the reported preferences/utilities to make the decision.
2. Initiator proposes options and the other agents can accept or reject but not make proposals of their own e.g., [46]

3. IMAP [37] designed for the Multiagent Meeting Scheduling problem, same as 2, but the other agents must accept a proposed time if it is available in their calendar.

4. Monotonic concession protocol for bilateral [43] and multi-lateral [18] negotiation.

Our choice of protocol features was motivated by concerns for allowing agents to preserve some privacy, encouraging efficient negotiation, and allowing agents the flexibility to act in their own interests.

We note that the number of rounds it takes to reach agreement in the protocol given in Algorithm 2.2.1 relates to the number of agents, the number of options, as well as $\mathcal{N}_{\text{range}}$, and $\mathcal{N}_{\text{rule}}$. In most chapters in this thesis we will set $\mathcal{N}_{\text{range}}$ such that each agent must offer some number of options greater than one every round. Furthermore, in all cases we use $\mathcal{N}_{\text{rule}}$ to require that when an agent offers an option, it is an option it has not offered for that variable in a previous round. Under these conditions, *for any variable, if an agreement is possible, it will* always *be found in strictly less than* $|\mathcal{O}|$ *rounds.*

3.3 Summary

In this chapter, we analyzed aspects of the negotiation process in agreement problems, including an analysis of the 1-Preference-Order negotiation strategy. Our analysis showed that one of the standard metrics for evaluating the performance of outcomes in a multiagent system, Pareto Optimality, is not very suitable for SC-EIMAPPs. As such, in this thesis, we focus on the average utility achieved by each of the agents. In particular, we evaluate our approaches by looking at how well they improve performance.

Chapter 4

Modeling User Preferences

In this chapter, we look at the problem of an agent learning its own user's preferences over options in EIMAPP where $\mathcal{R}(a, \mathcal{O})$ contains all $o \in \mathcal{O}$ s.t $(o, v') \in \mathcal{M}, \forall v'$, i.e., an option can be assigned to at most one variable by a given agent. In domains such as meeting scheduling, EIMAPP agents represent users who have complex preferences. In order to negotiate on the behalf of their users, EIMAPP agents first need a model of their users' preferences. Our approach passively observes the assignment of variables to options and uses the information derived to induce the preference model. This method does not involve having to query the user, nor does it require the user to assign variables to options through some set process, during the initial learning phase. The class of option preferences we want to learn can depend not only on the options themselves, but on dependencies between the options. For example, in the multiagent meeting scheduling problem, many users have *time-of-day* preferences, they might prefer options that represent morning times over those in the afternoon for instance. But their preferences are also based on dependencies between options, such as the preferences for having meetings back-to-back or spread-out.

The problem of an agent learning its own user's preferences is closely related to one of the main focuses of this thesis, namely learning about *other* agents/users. The problems are similar, because if we were to design a structure to try and capture the full complexity of the underlying model of an agent's own user and the users of other agents, we would use the same structure for both. The problems are different, because we have access to very different training data.

When observing its own user, an agent generally has access to *many more examples of variables being assigned to options*, than when it learns about other agents. In addition, the agent also *knows which options its user already has assigned* (this is significant in domains with restrictions on agent option usage). In contrast, when we look at learning a model of *another* agent's/user's preferences we will start by trying to learn a simpler preference model, so we can focus more on learning the other agent's constraints (which options are already assigned). We will also assume our learning agent knows its user's preferences and is using negotiation to assign variables to options. It is from

these negotiation histories that we will learn about the other agents.

In the context of Multiagent Agreement Problems there has been a significant amount of work on learning user preferences about when meetings are scheduled. Mitchell *et al.* [35] present CAP, a learning apprentice designed to assist a user in updating his/her calendar. Each time the user adds a meeting (through CAP's structured interface); CAP suggests a duration, location, date, and start time for the meeting. CAP's suggestions are either accepted or corrected by the user and in this way training data is continually created. Oh and Smith [38] take a statistical approach to learning a user's time-of-day preferences, from negotiation histories. They assume that all agents are aware of the organization's hierarchy, and use a specific negotiation strategy that gives priority treatment to the preferences of people higher in the hierarchy. Lin and Hsu [32] use hierarchical decision trees to learn a detailed user model, but require the user to provide a very large amount of data. Kozierok and Maes [29] use a memory based approach. As the user takes scheduling actions, the agent stores in its memory a situation, action pair for every action. This memory is used to suggest the appropriate action to the user as the need arises.

Existing approaches have not explicitly dealt with learning preferences that involve dependencies, and most require training data that is costly or difficult to obtain. Our approach is designed to addresses these two points. We show that using only very limited training data, we can effectively learn option-based preferences as well as *assignment dependent preferences*. In the context of meeting scheduling, this means we can learn simple time-of-day preferences as well as preferences such as liking back-to-back meetings.

4.1 Learning Dependant Option Preferences

In this section, we abstract an approach we developed in [11, 12], for modeling dependent user preferences in the meeting scheduling problem to EIMAPPs.

4.1.1 Capturing User Preference Information

Previously, we described a number of approaches in related work to learning the scheduling preferences of users. Mitchell *et al.* and Blum [4, 35] collected information about the user's preferences by asking the user. The user was not directly queried, rather he/she had to accept or reject CAP's actions. However CAP's task was simply to predict the time of a meeting, not to negotiate with other agents to find a meeting time. If we were to query users in a negotiation setting, we would require their participation unreasonably often.

We examine an approach whereby, the user's preference information is captured by simply observing the changes in the user's option assignments. Using this approach we do not have to query the

user, and it should allow us to capture the dependent aspects of the user's preferences. A possible difficultly however, is that *option assignments depend not only on the preferences of the user, but also on the preferences of the people the user must agree with.* We would also have this difficulty were we to try and learn from negotiation traces. On first glance it may seem that the options a user proposes *first* are their most preferred options. However, the options proposed might have been selected according to who the user is trying to agree with. For instance, in the meeting scheduling problem, when a junior staff member needs to meet with a senior staff member, the junior staffer may offer times that he/she thinks the senior staffer will prefer. In this instance, the offered times tell the agent nothing about the junior staff member's time preferences. As such, the training data will necessarily contain some noise.

4.1.2 Modeling User Preferences

We are interested in being able to model dependent preferences. We don't simply want to represent a coarse model such as whether a user prefers option o_1 to o_2. Rather we want to model what options a user prefers subject to his/her changing option assignments. For example, we want to be able to describe user preferences for having meetings back-to-back or spread apart.

We model a user's option preferences by placing probabilities on transitions between possible states of the user's option assignments. The current state of the user's option assignments determines the states that are reachable through the assignment of one new variable. Moreover, some of these reachable states are more preferable according to the user's preferences than others. We account for these preferences by placing probabilities on the transitions.

Formally, we model a user's preferences over options using a weighted-directed-acyclic graph, $P = (S, \vec{E})$, where:

- S is the set of all possible states of the user's *assignments*. *assignments* is a bit vector, one bit for each possible option. For each option, a "1" indicates there is a variable assigned to the option, and a "0" indicates the option is unassigned.
- Consider $s, s' \in S$, then $(s, s') \in \vec{E}$ iff we can get s' from s by adding one new assignment. So s' contains exactly one more "1" bit than s, and s' has a "1" in every place that s has a "1".
- Each edge $(s, s') \in \vec{E}$ has a weighting corresponding to the probability of transitioning to s' from s.

In Figure 4.1 we show an example graph with 4 options where the user prefers the first and the last option, but also has a strong preference for consecutive option assignments e.g, in the meeting scheduling problem, users may prefer to have meetings scheduled back-to-back (i.e., in succession). We have placed probabilities on some of the links in the example to demonstrate this preference.

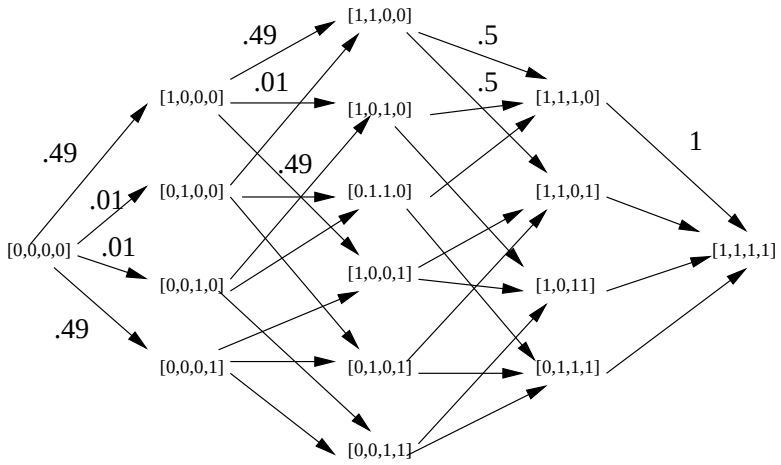


Figure 4.1: Example preference model for 4 options, where the user prefers the first and last option, but also has a strong preference for consecutive option assignments.

4.1.3 Learning the Model

We can use training data, obtained from logging changes in the user's option assignments, to learn the transition probabilities in our model.

Let the training data, T, be a set of training instance pairs, t_i, where:

- $t_i = (\text{current assignments}_i, \text{new assignments}_i)$
- new assignments$_i$ is obtained simply by adding one extra variable to current assignments$_i$.

Training data of this form is relatively easy to collect. Each time a variable is agreed upon we simply need a way of recording the current assignments$_i$ and the new assignments$_i$.

Given some training data, we first create the graph, $P = (S, \vec{E})$, and then use the data to annotate it. For each training instance pair t_i we locate the current assignments$_i$ state in P, and increment the state's frequency count. We then identify the edge that connects current assignments$_i$ with the state for current assignments$_i$ and increment its frequency count. Once all the training examples have been processed we can easily obtain the edge probabilities from the link and state frequencies, however it is also sufficient to simply store the information as frequencies.

4.1.4 Using the Model

When a new variable needs to be agreed upon we can use the graph to rank the possible options, according to their probability, as follows:

- We identify the state s corresponding to the current state of the user's option assignments.
- We then rank the successor states of the current state, i.e., the states s' such that $(s, s') \in \vec{E}$. This corresponds to a ranking of the possible options.
  - If the frequency count of s is 0 we rank the states s' according to their frequency.
  - Otherwise we rank the states s' according to the frequencies on the edges $(s, s') \in \vec{E}$.

4.2 Evaluation in the Meeting Scheduling Domain

In order to evaluate our learning procedure we have applied it to the Mutliagent Meeting Scheduling Problem (which we introduced in Chapters 1 and 2). We have created some realistic models of user preferences that include dependencies, such as the preference for having meetings scheduled back-to-back, as well as static time-of-day preferences (e.g., a preference for mornings over afternoons). To generate training data, we log the changes in an agent's calendar that occur as the agent schedules meetings with other agents using its negotiation strategy and its true preference model. On the basis of this training data, we produce a learned model of the agent's preferences. We then schedule a test set of meetings using the *true preference* model, the *learned preference* model, and a *random preference* model. We compare the performance of the different preference models by evaluating the schedules that result (according to the true preferences). We show that *an agent negotiating using the learned preference model performs almost as well as if it used the true preference model*, and much better than if it used the random preference model.

4.2.1 Time-based Preference Model with Dependencies

We have designed a realistic model of user preferences that will form the *true preference* model in our experiments. We are interested in whether our approach can learn dependent as well as static components of preferences. As such, our model of user preferences includes a static time-of-day preference, a back-to-back preference, and a spread-out preference. Formally, we define a user's utility for scheduling a meeting at time t, given the current calendar *cal* as follows.

$$\begin{aligned} u(t, cal) = \alpha * p(t) + \beta * \text{back-to-back-score}(t, \text{cal}) \\ + k * \text{spread-out-score}(t, \text{cal}) \end{aligned}$$

where:

- $p(t)$ quantifies how much the user likes having meetings at time t, regardless of the current configuration of the calendar. We assign values to times based on the user's preference for morning, mid-day and afternoons. For instance, if the agent's ranking over time classifications is morning, afternoon, then middle of the day, we assign t the highest value when t is in the morning, and the lowest when t is in the middle of the day.
- back-to-back-score(t, cal) is 2 if the bit corresponding to t in *cal* has a 1 on both sides, 1 if on one side there is a 1 and the other a 0, and 0 otherwise.
- spread-out-score(t, cal) is 2 if the bit corresponding to t in *cal* has a 0 on both sides, 1 if there is a 0 on one side and a 1 on the other, and 0 otherwise.
- α, β, and k are constant weightings. In our experiments $\beta > 0$ iff $k = 0$ and vice versa.

4.2.2 Random Preference Model

We evaluate a random preference model in order to form a baseline for comparison. The *select random* preference model does not maintain a consistent ranking of times. Instead, each time the preference model is queried, a random ordering of times is generated. Selecting a different random ordering each time gives a more effective base-line than choosing a single random ordering, since a single ordering could by chance be either very close to the true preferences or very far from them. We note that if a single random ordering was selected and used then we would no longer have an effective baseline, since performance could be arbitrarily bad when using such a model (it is possible to compensate for this, but it would require running a lot of experimental trials). Furthermore, if we were to use a simple guess or heuristic to decide the user's preferences we could have the same difficulty. For instance, if we guessed on the basis of a few examples that the user liked mornings and spread-out meetings then we could end up with very bad performance if these examples were misleading. Moreover, note that if we used a standard learning algorithm such as a decision tree learner to try and model the user's time-of-day preferences we would get very poor performance if the user had strong dynamic preferences. The advantage of using the *select random* model is that it forms a consistent baseline across different types of user preferences.

4.2.3 Negotiation Protocol and Strategies

We use a protocol based on the general negotiation protocol described in Chapter 2. The protocol operates in rounds. In each round the initiator and the attendees add more times to their original offer in search of an intersection of available times. For a complete description of the protocol used in this chapter please see the "Reply-To-Initiator protocol" in Appendix 1.

We have used two negotiation strategies in our experiments. In both negotiation strategies agents offer times for a meeting in order of preference. The Offer-n-negotiation strategy operates in the

Algorithm 4.2.1 Offer-n-Negotiator Algorithm for Initiator (meeting)

```
while meeting not added to the calendar do
  Oofer n available, previously un-offered times for the specified day to the attendees.
  if not response received from all attendees in this round then
    wait for more responses
  if in negotiation mode, and there is an intersection then
    send a confirmation request for the best time in the intersection
    switch the meeting to pending mode
    continue
  if there is no intersection then
    continue
  if in pending mode, and all attendees accepted the confirmation request then
    send a confirmation message and add the meeting to the calendar
    return
  if in pending mode, and one or more of the attendees rejected the confirmation request then
    cancel the confirmation request with the attendees
    switch the meeting to negotiation mode
```

context of the protocol, in each round the initiator and the attendees add more times to their original offer in search of an intersection of available times. Specifically an agent using this strategy offers n, available, previously unoffered times for the appropriate day. A meeting initiator running this strategy behaves according to the policy in Algorithm 4.2.1. If in the negotiation process the initiator detects that no one (including the initiator) is offering new times for the day in question, the initiator switches to requesting times for the next day. The corresponding algorithm for an attendee agent is shown in Algorithm 4.2.2. The Offer-n-negotiation algorithm can represent a wide range of negotiation behavior. It also corresponds to the kinds of strategies that have been proposed for automated meeting time negotiation e.g. in [10, 46].

The Hold-Out-for-n negotiation strategy specifies for how many rounds the agent will 'hold-out' for its top two choices i.e., the number of rounds the agent will wait before offering a time that is not its first or second choice. When the number n is large, this can be a very uncooperative negotiation strategy. In both strategies, if the initiator detects that no one is offering new times for the requested day, it switches to requesting times for the next day. This strategy is shown in Algorithm 4.2.3. When n is low the agent compromises quickly and when n is high it compromises slowly.

4.2.4 Training

In our experiments we use six agents with various *hidden true preference* functions and rankings of morning, middle-of-the-day and afternoon times. The meetings used for the creation of the training

Algorithm 4.2.2 Offer-n-Negotiator Algorithm for Attendee (message)

if this is a new meeting **then**
enter negotiation mode for this meeting
if in negotiation mode for this meeting, and this message is not a confirmation request **then**
offer n available, previously un-offered, times on the requested day, return
if in negotiation mode, message is a confirmation request and time not pending or taken **then**
send a message to the initiator accepting the time and enter pending mode, return
if in negotiation mode, message is a confirmation request and time pending or taken **then**
send a message rejecting the request, return
if in pending mode, and this message cancels the confirmation request **then**
move the meeting to negotiation mode
offer n available, previously un-offered, times on the requested day
if in pending mode, and this message is a final confirmation **then**
add the meeting to the calendar

Algorithm 4.2.3 Hold-Out-for-n-Negotiator

The mechanics of this strategy are similar to those of the Offer-n-Negotiator with the following exceptions:
Offer one new time per negotiation round.
After offering 2 times the initiator 'holds out'. If the attendees stop offering new times, the initiator proposes a new day.
An attendee waits n rounds ('holds out') before offering a time that is not in its top two choices.

data and for the testing phase are generated randomly. The day of the meeting is chosen at random as well as the initiator and the attendees. In any training or testing set, the number meetings with x participants is $\frac{1}{2^{x-1}} *$ total no. meetings for $x = 2...5$ and the rest of the meetings are of size six (i.e. all the agents are participants).

To generate the training data for the learning algorithm, we take a set of meetings and a set of agents. One of these agents is assigned to be the learning agent, and every meeting in the set involves this agent. The agents schedule the meetings using their true preferences and one of the negotiation strategies. Since the training set is large, we reinitialize the agents' calendars to empty at various intervals e.g after 15 meetings have been scheduled. Every time a meeting is added to the learning agent's calendar, we add an example to our training data consisting of the old and new calendars for that day. The learning algorithm, previously described, is then applied to the complete set of training examples.

4.2.5 Comparing Results

Our aim is to compare how well an agent can schedule meetings with the learned preferences versus the true preferences. We generate a large set of meetings for use in testing. We schedule these meetings with the learning agent's true preferences, the learned preferences and the select random model. Since the set of meetings for testing is large, we have to periodically reinitialize the agents' calendars as we schedule the meetings. Before we reinitialize however, we record the state of the learning agent's calendar. Once all of the meetings have been scheduled we take each of these saved calendars and evaluate the calendar according to the true preferences of the learning agent. These evaluations are averaged over all the calendars to produce the average calendar preference score for the preference model in use. We then compare these average calendar preference scores. In our discussion of the results, we refer to this summary measure of calendar preference score as average *utility*.

4.3 Results

Our results show that negotiating with the learned preferences, on average, results in schedules that are only marginally worse than negotiating with the true preferences.

Figure 4.2 shows how the average utility of the schedules produced by negotiating meetings using the learned preferences increases with the number of training examples. In this particular example, each agent was using the Offer-n negotiation method with $n = 2$ to schedule 250 randomly generated test meetings. The true preferences of the learning agent were for meetings later rather than earlier in the day, with a strong preference for meetings being scheduled back-to-back. The performance obtained using the learned preferences averaged over the first 30 training examples is *only 2.9% worse than that obtained using the true preferences*. So even for very small training sets the algorithm is able to perform well.

Negotiating using the random preference model was 25% worse than negotiating using the true preferences. Note that, no matter when the meeting is scheduled, it still yeilds some utility, and it is not just the learning agent's preferences that determine the meeting times. If the agent was always able to get his/her most preferred times, then by comparison, the random model would perform much more poorly. Since the agent has to negotiate the meeting times with other agents, the negative effect of using an imperfect model of user preferences is limited. The negotiation aspect also means it is possible for the use of the learned model to sometimes outperform the true model as we see in Figure 4.2. This is because it is possible to sometimes negotiate a better meeting time by not offering times in the exact order of true preference. For a proof of this fact, see Lemma 3.1.5.

Figure 4.2 also demonstrates that even having one training example helps the learned model outperform the random model. Since the test set contains 250 meetings and we reset the calendars after

Figure 4.2: Scheduled using TOD and back-to-back preferences with Offer-n-negotiator

scheduling 15 meetings it is possible that one training example can help the learned model multiple times. We see also that sometimes adding a new training example degrades performance. This can happen when the training example added is not a good indicator of the true preferences. Misleading training examples can occur because sometimes the only possible times for a meeting, are times the user does not like. Pruning training examples that degrade performance could alleviate this problem.

Figure 4.2 shows that we can learn back-to-back and time-of-day preferences effectively enough to achieve good negotiation outcomes. In the next two Figures we look at a harder example where the learning agent uses the Hold-out-for-n negotiation procedure with $n = 3$ while the other agents use the Offer-n negotiation procedure with $n = 3$. The training examples produced from scheduling meetings amongst these agents are better for the learning agent in the sense that the examples will more often demonstrate meetings being scheduled at the agent's favored times. However, it also means that when the agent schedules meetings with the learned preferences in the testing phase, misranked times are more costly since the agent is likely to get the times it holds out for.

Figure 4.3 shows how the effectiveness of the learned model increases with the number of training examples, when the learning agent uses the Hold-out-for-n negotiation method and the other agents use the Offer-n method. In this Figure, the true preferences of the learning agent have both a back-to-back and time-of-day component. It takes longer for the agent to learn a good model in this

Figure 4.3: Scheduled using back-to-back and time-of-day preferences and Hold-Out negotiator

case, however, the average difference across all the training sets, between the utility for the true model and the utility for the learned model is only 6%. So even in this harder case, the learning model performs well. We note that the performance of the learned model increases rapidly as the number of training examples increases from 1 to 50. However, the performance after only 20 training examples is already very reasonable. This is important because it is desirable that any deployed system be able to learn well from very few examples. Furthermore, it means there is potential for extending the class of preferences represented by the model while maintaining good performance.

Figure 4.4 demonstrates that our approach also effectively learns the preference for having meetings spread-out. The Figure shows how the learned model improves with the number of training examples when the learning agent's true preferences have both a time-of-day and a spread-out component. The learned model performs almost as well as the true model very quickly. If we average the difference in utility over the first 30 training sets we find that on average the learned model only performs 2.8% worse than the true model. We get better performance for spread-out preferences than for back-to-back preferences because it is possible to score some points for satisfying spread-out preferences simply by not having a full calendar. However, to satisfy back-to-back preferences the preference models have to cause meetings to be scheduled side-by-side.

Figure 4.4 also shows how sometimes performance can degrade with more training examples if the

new training examples are misleading. The figure however demonstrates that eventually adding more training examples reduces the effect of the misleading examples and the performance increases again.

We have explored the various parameters in our experiments. We found that varying the preferences of the agents, reinitializing the calendars at different intervals, and changing the n parameter in the negotiation strategies, had little effect on how well our algorithm learned.

Figure 4.4: Scheduled using spread-out and time-of-day preferences

4.4 Summary and Discussion

In this chapter, we reviewed a method we introduced to capture, model, and learn, the dependent option preferences of users. Many methods for learning user preferences in EIMAPP domains have assumed the use of data that in a deployed agent system would be hard or nearly impossible to obtain, due to the limited number of questions a system can ask a user without causing significant annoyance and the current capability of technology to process human communications (e.g., email negotiations). Furthermore, the dependent nature of user preferences has been largely ignored. We have shown how we can capture dependent user preferences by simply logging changes in a user's option assignments and use this data to learn effective models. We have demonstrated the

effectiveness of our approach by showing that in the multiagent meeting scheduling problem we can use learned time-of-day, back-to-back and spread-out preferences, to negotiate schedules that are within 6% of what can be achieved using true preferences.

By showing how dynamic user preferences can effectively be learned from realistic training data, we have made good progress towards the development of deployable multi-agent meeting scheduling systems. One aspect of user scheduling preferences that we have not considered, is how the particular type of meeting being scheduled affects the user's time preferences. For instance, someone may prefer administrative meetings in the morning and work meetings in the afternoon. The fact that using our model results in very good performance, after only a small number of training examples, leads us to believe it can be extended to effectively cover this case.

In the next chapter, we will look at the related problem of learning models of another agent. The training data available to us when learning about another agent is more limited. There are generally fewer examples, and since we do not know what options another agent already has assigned we are faced with the problem of simultaneously trying to learn about the agents *constraints* (for the case where $\mathcal{R} \neq \emptyset$), as well as preferences. As such, we start by trying to learn a less complex preference model. We will assume in the next section that our learning agent knows its own preferences (if it is representing a user, that these preferences have been previously learnt or supplied), and its task is now to learn the preferences and constraints of others in order to improve its negotiation.

Chapter 5

Modeling Agent Properties

In this chapter, we build upon our observation that as semi-cooperative agents negotiate they reveal information about their preferences and constraints. We show how an agent can learn abstractions of the preferences and constraints of the agents it negotiates with. Furthermore, we show that an agent can use these abstractions to improve its utility through, decreasing the number of negotiation rounds to reach agreements of the same preference value.

Our work differs from related work on property modeling in negotiation in that it focuses on how negotiation efficiency can be improved with very limited training data. We make weaker assumptions about agent behavior than in related work and our training data only consists of option round pairs[1].

5.1 Negotiation Output - Training Data

When an agent negotiates a variable with a set of agents it can record the offers that were made, and the rounds in which they were made. If the agents are negotiating via multi-cast (i.e., all messages are copied to all agents) then all the agents can see the offers of every decision maker. However, if each agent only sends its offers to the initiator, then the initiator is the only agent to see all the offers.

We let the *negotiation history* for a particular variable v, h_v, have the following form. For each: $A \in \mathcal{D}_v$, whose offers are visible to the agent

$$h_v(A) = [\text{offer}_A(v, r_1), \text{offer}_A(v, r_2), \ldots, \text{offer}_A(v, r_{\text{final}})],$$

[1]For details of related work, please see the related work chapter.

where $\text{offer}_A(v, r_i)$ is the set of options proposed by agent A for variable v in round r_i of the negotiation.

Over the course of negotiating multiple variables in the agent system, $\mathcal{A}$, an agent has the opportunity to observe other agents offer the same option multiple times. In the negotiation procedures we use, an agent can only offer the same option once for a particular variable (according to the $\mathcal{N}_{\text{rule}}$, so in order to observe multiple offers of the same option, the agent has to negotiate multiple variables.

5.2 Complexity of the Generating Function

The process that generates an agent's offers for a variable can be a function of many components, including:

- The agent's preferences
- The agent's constraints and the agent's current state with respect to those constraints. For example, in the meeting scheduling domain, the constraint that an agent can't attend two different meetings at the same time, combined with the current state of the agent's calendar affects the offers the agent makes.
- The agent's utility function - how it trades-offs preference satisfaction with negotiation cost
- The agent's beliefs about other agents
- The offers other agents have made
- The agent's negotiation strategy - i.e., the rules it uses to decide which options to offer (based on the other factors)

Given a negotiation history for an agent, we would like to be able to derive the function that generated it. However, due to the complexity of the generating function, the negotiation does not reveal enough information to accurately derive a single generating function.

Consider the following example. Two agents are negotiating with each other to schedule a meeting. One agent represents a Professor and the other a graduate student. After two rounds of negotiation, the graduate student agrees to the Professor's first proposal. What can the Professor conclude with *certainty* about the graduate student's properties? Unfortunately there are many possible reasons for the graduate student's behavior. Here are just some examples:

- The graduate student agent does not like the agreed upon time but has a very high negotiation cost with the Professor

- The graduate student agent reasonably likes the agreed upon time and has a low negotiation cost with the Professor
- The graduate student agent is using a negotiation strategy based on what it has learned about the Professor and thought the Professor would agree to the option it offered in the first round, when this failed the next option in line was the option that led to agreement.

Fact In general, the negotiation function cannot be uniquely derived from a negotiation history (regardless of length).

Argument We illustrate this fact by constructing an example where two different negotiation functions always map to the same negotiation output. For a practical example of the difficulty of deriving a negotiation function, see the previous example. Let the set of all options be $\mathcal{O} = \{o_1, o_2, o_3, o_4\}$, let there be no option constraints in this EIMAPP and let the agents use a protocol requiring exactly one option to be offered each round. Agent A_1 has the preference function, $p(o_1) = 1, p(o_2) = 1$. A_1's negotiation strategy is to offer options in order of preference, tie-breaking on the option with the lowest index. A_1's utility function is a semi-cooperative additive function with an α value of 3 and a β value of 1. Note that regardless of the options A_1 is offered, variables previously negotiated, A_1's beliefs etc., A_1's negotiation function will *always* output o_1 in the first round and o_2 when/if there is a second round. Now let A_2 have a preference function of $p(o_1) = 2, p(o_2) = 1$ and an α value of 4 and a β value of 1. A_2's negotiation strategy is to always offer options in the order of preference. Notice that this *always* produces the same output as A_1's negotiation function: o_1 in the first round and o_2 when/if there is a second round. Hence *two different negotiation functions can yield identical negotiation histories* and therefore it is not (in general) possible to uniquely derive a negotiation function from a negotiation history.

5.3 Abstracting Useful Agent Models

We need to define appropriate learning goals, given the complexity (and in general the impossibility) of obtaining negotiation functions from negotiation histories. Since we cannot learn an exact model of an agent's negotiation function (preferences, constraints, strategy, etc), we propose that we instead learn a *useful abstraction of the agent's preferences, constraints, and behavior*. We define an *abstraction* of an agent's preferences, constraints and behavior to be a representation that attempts to approximate the true model at a lower granularity than what is required for a full representation. A learned abstraction of another agent is *useful* if an agent can use it to increase the efficiency of negotiating with that agent (or to improve its utility in some other way).

Proposition 5.3.1. *It is possible to learn* **useful abstractions** *of negotiation functions from negotiation histories.*

In this chapter we provide a technique for defining useful abstractions in a domain independent manner. We then demonstrate this proposition empirically in the multiagent meeting scheduling domain where we show that by learning abstractions agents can increase negotiation efficiency, without sacrificing their preferences.s

5.3.1 Defining Useful Abstractions

To be useful, an abstraction of another agent's negotiation function, must be able to inform the agent's negotiation decisions. We observe that knowledge of the *likelihood of the other agent offering each of the different options* is useful for increasing negotiation efficiency. Consider the example in Figure 5.1. The example illustrates that if A_1 understands that A_2 is more likely to offer o_2 than o_1 then A_1 can speed up the negotiation.

A_1 as initiator needs to assign variable v with agent A_2, there are 10 options $o_1, ...o_{10}$ in the option set.

- A_1 has three options available for assignment, o_1 and o_2, and o_3, such that $pref_{A_1}(o_1) = 9$, $pref_{A_1}(o_2) = 9$ and $pref_{A_1}(o_3) = 1$
- A_1's cost per negotiation round $c_{A_1}(r) = 2r$.
- o_1 has a preference value of 1 to A_2 and A_2's negotiation strategy stipulates that A_2 will only agree to options with a preference value of less than 2 after 3 negotiation rounds have passed.
- o_2 however has a preference value of 4 for A_2 and A_2 will agree on it right away.

If A_1 has learned that A_2 is more likely to offer o_2 than o_1, then, A_1 knows it should offer o_2

- If A_1 offers o_2 in the first round, A_2 will agree (at the latest in round 2) by also offering o_2. So A_1 will get a total payoff of $9 - 2 * 2 = 2$.
- If A_1 tries to get an agreement for o_1 instead, it will take until the 4th round. So A_1 will get a total payoff of $9 - 4 * 2 = 1$.

So A_1 should offer o_2.

Figure 5.1: Negotiation example

5.3.2 Ordered Category Abstraction

We propose a domain independent technique for designing abstractions that involves creating a set of *disjoint option categories*. These disjoint categories should be *totally orderable based on how likely options with that category label are to be offered by the agent we are modeling*. For instance,

in an EIMAPP domain with use-once per agent options, we might define the following ordered categories:

- c_1 - unassigned and preferred;
- c_2 - unassigned but not preferred;
- c_3 - assigned

Notice that for a domain such as meeting scheduling where there is a cost associated with moving meetings, it is reasonable to assume that an agent is more likely to offer options with category label c_2 than with label c_3 and with c_1 than with c_2.

5.3.3 Probabilistic Agent Model

As demonstrated earlier it can be difficult or impossible to separate different components of an agent's negotiation function. Due to this fact, and the limited availability of training data (e.g., two agents may negotiate with each other infrequently) we use a *probabilistic model of agent beliefs*.

Let $b_A(o) \in B_A$ be a *belief vector* over the chosen abstraction, i.e, the ordered set of disjoint categorizations C for each option o. In other words, we maintain one belief vector for each option, indexed on category label. Each belief vector $b_A(o)$ contains the probability (according to the agent's belief) of the option having each of the category labels based on the observed data.

We use the ordering of the categories to define a *conditional probability distribution* which, for each category, gives the *probability that the agent proposes an option, given that the option fits the category*. If we let $c \in C$ be a category then the conditional probability distribution is given by $Pr(A \text{ proposes } o \mid o \in c)$. *This probability distribution represents the abstraction we are using to model the agent's behavior based on our abstraction of its preference's and constraints*. We will see in the experiments it is not important what exact number the conditional probabilities are set to. Rather the key is the that the categories are ordered relative to the probability that the agent proposes options as we have discussed.

5.3.4 Limited Training Data and Category Granularity

The amount of training data available helps to inform the granularity of the categories chosen. For example we could hypothesize 20 different categories of preference, but in many domains we would then end up with a lot of uncertainty over the categories. In the meeting scheduling domain for example, two agents may negotiate with each other only a couple of times a week. This

negotiation can occur from different starting calendars, causing the agents to sometimes have different availability when they negotiate. However, we don't want many weeks to have to pass before we have learnt a useful abstractions of other agents. In fact, we may never get enough training data to separate the options into large a number of categories in a domain like meeting scheduling. As such, it makes sense to choose only a small number of categories that will be useful, and relatively easy to distinguish.

5.4 Abstraction Learning from Negotiation Histories

If the agent has never negotiated a variable with a particular agent, A, before then $\forall o \in \mathcal{O}$ and $\forall c \in C$ the agent's belief vector, $b_A(o)$ is in the initial state. The initial state of the belief vector is defined subject to the domain and the chosen abstraction. Encoding *domain knowledge* into the initial belief vector allows the agent to learn more accurate abstractions from fewer negotiations. For example, suppose we have chosen the categorization:

- c_1 - unassigned and preferred;
- c_2 - unassigned but not preferred;
- c_3 - assigned

Furthermore, let our conditional probability abstraction of A be $Pr(A \text{ proposes } o \mid o \in c_1) = 0.8$, $Pr(A \text{ proposes } o \mid o \in c_2) = 0.4$, and $Pr(A \text{ proposes } o \mid o \in c_3) = 0.1$.

We could choose an initial belief that gives each category equal probability. Alternatively we could leverage our domain knowledge (about the specific agreement problem) and build into the initial belief the information that if A has not offered an option before it is more likely to be assigned than unassigned and more likely to be not preferred, than preferred. In the experiments we will show how using domain knowledge in this way can help the learning process.

Given an initial belief vector, the agent then needs to update it. Each time the agent receives an offer of an option o from A, the agent can use the conditional probability distribution to update its belief vector for that option.

5.4.1 Updating the Agent Model

The agent uses Bayes rule to make the update to the belief vector. Suppose the agent has belief $b^t_A(o)$ at time t, and then the agent receives an offer of option o from agent A. The agent performs

the following update to obtain a new belief vector for time $t+1$.

$$b_A^{t+1}(o)[c] = \frac{b_A^t(o)[c]Pr(A \text{ proposes } o|o \in c)}{\sum_{i \in C} b_A^r(o)[i]Pr(A \text{ proposes } o|o \in i)} \tag{5.1}$$

Using Additional Domain Information

As an aside, we note that in some cases there may be additional domain information available that can help the learner update the model. This is particularly the case where there are *semantic relationships between the different options*. For instance, at some universities, a particular class is generally held Monday/Wednesday or Tuesday/Thursday, at the same time on both days. As such, if the learner believes that agent A is unavailable at 10am on Tuesdays, it may want to increase slightly its belief that A is unavailable at 10am on Thursdays. Another example is, if the learner knows agent A likes to meet before mid-day on Monday, Tuesday and Wednesday, but not in the afternoons on those days, the agent could use domain knowledge to reason that the agent prefers morning meetings to afternoon meetings and adjust its beliefs for times on the other days. Domain knowledge, such as the examples given here can be represented as conditional probabilities and worked into the update process.

5.4.2 Summary of the Learning Procedure

We summarize the learning procedure as follows:

- For each $o \in \mathcal{O}$ create belief vector $b_A^0(o)$ and for each $c \in C$ initialize the component $b_A^0(o)[c]$
- For every option o offered by A in the negotiation of any variable at time t, apply the update rule in Equation 5.1 to $b_A^t(o)$ to obtain $b_A^{t+1}(o)$.

5.5 Using the Learned Abstraction to Inform Negotiation

As the agent negotiates with another agent, A, it is learning which category label to attach to each of the options. Recall that in our method the categories are chosen such that they are totally orderable based on how likely options with that category label are to be offered by A.

Therefore, an agent can use its beliefs b_A about the categories options fall into for a particular agent, A, in order to reason about which options A is most likely to agree to.

For example, suppose we are using the three category abstraction we outlined earlier, and the following inequalities hold for options o_1 and o_2:

1) $b_A(o_1)$[preferred & unassigned] $> b_A(o_1)$[¬preferred but unassigned]
2) $b_A(o_1)$[preferred & unassigned] $> b_A(o_1)$[assigned]
3) $b_A(o_2)$[¬preferred but unassigned] $> b_A(o_2)$[preferred & unassigned]
4) $b_A(o_2)$[¬preferred but unassigned] $> b_A(o_2)$[assigned]

then the agent can reason that for A, o_1 is "preferred and unassigned" and o_2 is "not preferred but unassigned", and therefore that A is more likely to agree to o_1 than o_2.

There are many ways in which this type of information could be used. Our aim in this chapter is to demonstrate that we can learn a useful abstraction from negotiation histories. We show that this is possible by *using the learned abstractions to break ties in our negotiation strategies*. If a negotiation strategy normally stipulates to a break ties randomly e.g., between options of equal preference, we instead use the learned model to break the tie according to the option the agent believes is the most likely to be offered by the other agent. This allows the agent to improve negotiation efficiency without sacrificing its preferences.

Since our agreement problem is incremental, *all learning and use of the learned information occurs online*. Furthermore, we evaluate our agents based on their ability to learn and exploit their learned models online, starting from zero training data.

5.6 Experimental Evaluation

In this section, we describe the domain and procedure we use for evaluation. We use the multiagent meeting scheduling domain to evaluate our approaches. Efficiency is of particular interest in this domain because in open systems, a deployed agent needs to be able to negotiate with humans (e.g., over email) as well as other agents in order to schedule meetings.

In the multiagent meeting scheduling domain all agents have the constraint that they cannot assign the same option to more than one variable. In these experiments, our agents use negotiation strategies that are capable of *bumping* a meeting. An agent *bumps* an existing meeting by removing the meeting from its calendar and notifying the meeting initiator that the meeting needs to be rescheduled. The meeting initiator then must notify the attendees that the meeting has been *bumped* and restart the negotiation.

5.6.1 Negotiation Strategies

In our experiments we use the negotiation protocol described in Chapter 2 and and a set of negotiation strategies called *Threshold-Offer-k* (Algorithm 5.6.1). These strategies select up to k options to offer per round. They are designed for domains where an agent can assign an option to at most one variable, and an agent's preferences over the options can be classed as preferred/not preferred. This level of detail about meeting time preferences could be elicited quickly from a computer user. The Algorithm 5.6.1 has a number of inputs:

- *comp-r* - the number of rounds to consider compromise after, i.e., if this number of rounds has passed the strategy will consider offering a non-preferred time
- *comp-num* - the number of options offered to consider compromise after, i.e., if the agent has offered this many options the strategy will consider offering a non-preferred time
- *bump-r* - the number of rounds to consider bumping after
- *bump-num* - the number of options offered to consider bumping after

Algorithm 5.6.1 Threshold-Offer-k (k: number options to offer, *comp-r*: number of rounds to compromise after), *comp-num*: number of option offers to compromise after, *bump-r*: number of rounds to bump after, *bump-num*: number of options offers to bump after) $\rightarrow$ offer

```
assert bum-num ≥comp-num and bump-r ≥ comp-r
if number of preferred unoffered options > 0 then
   sort these options according to most offers by other decision makers
   break ties randomly
   add top k options to the offer, return
if current-round ≥ comp-r || num-offers ≥ comp-num then
   sort not preferred, unassigned options according to most offers by other decision makers
   break ties randomly
   add the top k options to the offer, return
if current-round ≥ bump-r || num-offers ≥ bump-num then
   sort currently assigned options according to most offers by other decision makers
   break ties randomly
   add top option to the offer, return
```

Threshold-Offer-k **always offers preferred options before non-preferred options, and unassigned options before assigned options**. Once the agent has no more preferred options to offer, if one of its compromise thresholds is met it will consider non-preferred options. When deciding which options to offer, the agent sorts the possible options according to the number of other agents that have offered the option. If the agent is not the initiator, it only receives offers from one agent

(the initiator). An option's priority is raised by one, for every agent that has offered the option. In practice, this means that if two agents are negotiating a variable using this strategy, and one agent offers some options the other prefers (and has unassigned), the other agent will respond by offering at least one of these options.

5.6.2 Learning Online

Threshold-Offer-k can be combined with our learning algorithm, by using the learned models to inform the sorting of the options (i.e., tie-breaking between options of equal preference). In our experiments, when the agent is learning, we give an option d points for every agent that has offered it (or d multiplied the number of other decision makers if we are not the initiator), e points for every agent that we *believe to prefer the option*, and f points for every agent we *believe has the option unassigned (but not preferred)*. In our experiments $d = 3$, $e = 2$, and $f = 1$. As the agent negotiates, every time it receives an offer it updates its beliefs as described in the previous section. Over time the agent learns to make offers that reduce negotiation time, while still satisfying its preferences.

5.6.3 Simulation Scenario

We have simulated meetings being scheduled amongst a group of agents over a series of weeks. In each trial some times in each agent's calendar are selected at random to be blocked to represent unavailability caused by commitments outside the multiagent system. A set of initial meetings (these meetings range in size, from two person meetings, to meetings involving all agents), are also scheduled. We refer to an agent's blocked time slots, and initial meetings as the agent's *base calendar*. In each trial the agents schedule a set of new meetings. Each time a certain number of meetings from the new set, e.g., 10, is scheduled, all the agents' calendars are reset to their initial states (the base calendars). This simulates the regular meetings that most people have, and the new meetings that are scheduled each week.

5.6.4 Chosen Abstraction

In our experiments we use the three category abstraction introduced earlier:

- c_1 - unassigned and preferred;
- c_2 - unassigned but not preferred;
- c_3 - assigned

We abstract the negotiation strategy of the other agents using the following conditional probabilities for the belief update: $P(A \text{ proposes } o | o \in \text{ preferred }) = 0.9$, $P(A \text{ proposes } o | o \in \text{ unassigned}) = 0.5$, and $P(A \text{ proposes } o | o \in \text{ assigned}) = 0.1$. [2]

5.7 Results

We evaluate our approaches in terms of the quality of the learned models and the increase in efficiency we obtain from incorporating learning into the negotiation process.

5.7.1 Learned Models

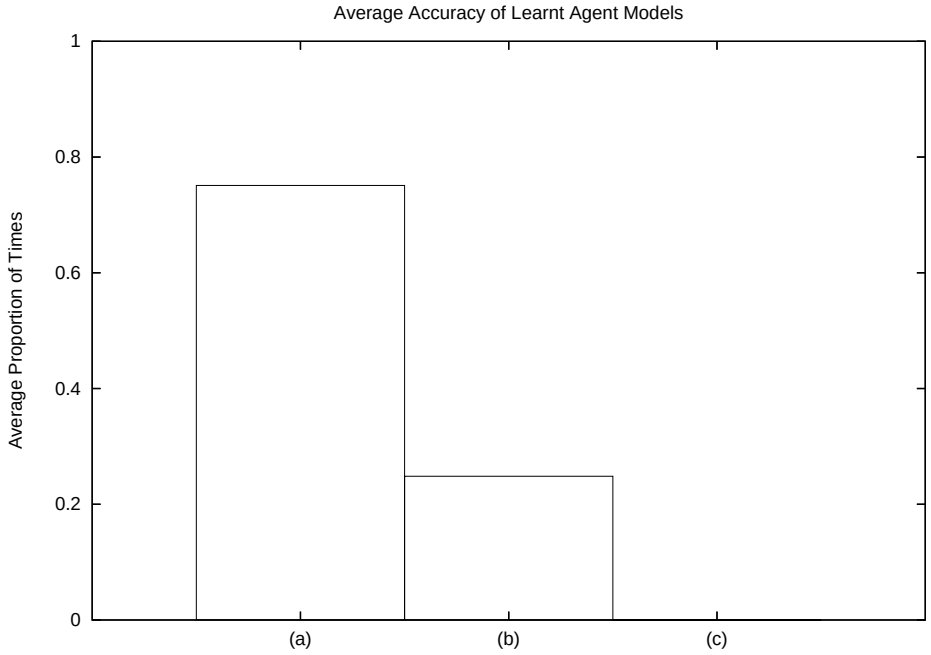


Figure 5.2: (a): $c_1 = c_1, c_2 = c_2, c_3 = c_3 (b) : c_1 = c_2, c_2 = c_1, c_2 = c_3, c_3 = c_2 (c) : c_1 = c_3, c_3 = c_1$. c_1 denotes the category preferred (and unassigned), c_2 the category unassigned (but not preferred) and c_3 the category assigned. The notation $c_1 = c_2$ indicates an option of category c_1 receiving label c_2. The graph shows the average accuracy of the learned models starting from an unbiased belief.

[2] We did not tune these values in an attempt to optimize results.

Figures 5.2 and 5.3 demonstrate that using our approach agents can learn models of other agents preferences and constraints with high accuracy. In these experiments, 5 learning agents scheduled 150 new meetings together. The agents' base-calendars were approximately one third full and 10 meetings were negotiated per week. The agents' preferences for options were randomly selected to be preferred or not preferred and the number of options offered per-round was set to 1.

We ran 100 trails and evaluated the average accuracy of one agent's models (of each of the other agents), the standard deviation was negligible. Figures 5.2 and 5.3 show the average proportion of options the learned model labeled correctly (1st bar). An option is considered to be labeled correctly if the category with highest belief in the learned model corresponds to the agent's true preferences and base calendar. For example, if an option is unassigned in A_2's base calendar and is preferred by A_2, then it is correctly labeled by the learned model if the highest belief is on *preferred*. Note that even if we have a perfectly accurate model it will not always reflect the current status of A_2's calendar since base calendar meetings can be moved, and new meetings are scheduled as the simulation runs. In this sense, the agents must *learn in the presence of changing constraints*.

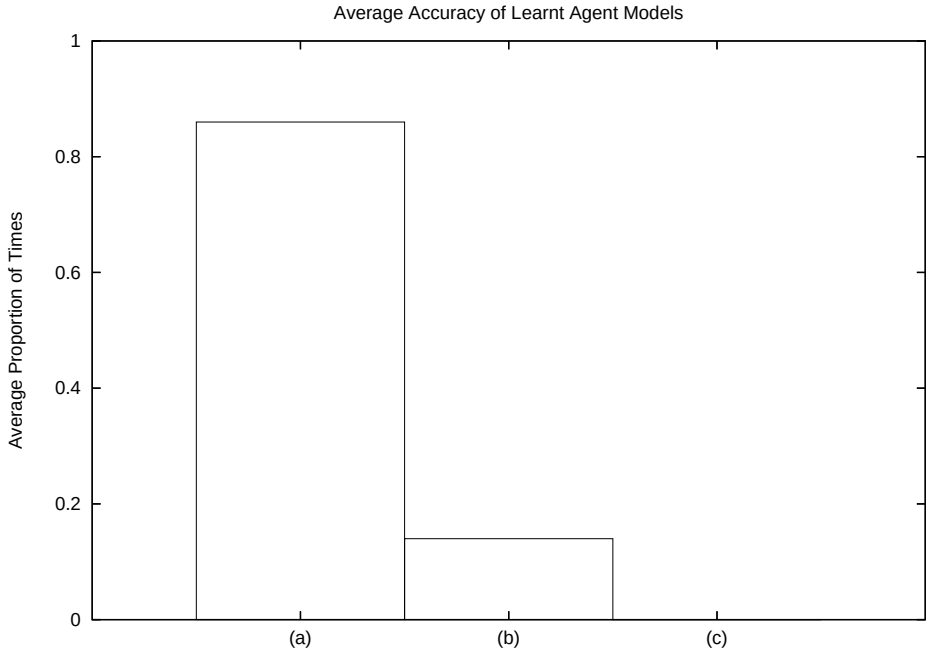


Figure 5.3: (a): $c_1 = c_1, c_2 = c_2, c_3 = c_3(b) : c_1 = c_2, c_2 = c_1, c_2 = c_3, c_3 = c_2(c) : c_1 = c_3, c_3 = c_1$. Average accuracy of the learned models starting from a biased belief.

Figures 5.2 and 5.3 also show the average proportion of options that are misclassified as a neighboring category (2nd bar) e.g., preferred options that are labeled as unassigned or unassigned options that are labeled as assigned. Figure 5.2 shows that when we start with a uniform belief over the

categorizations we get an average accuracy of almost 80%. We can improve this result simply by using domain knowledge to bias the inital belief towards the categorizations that imply we are less likely to recieve offers (assigned, unassigned). Figure 5.3 shows that with this bias we get an average accuracy of 86%.

Misclassifications occur for a number of reasons. Firstly, suppose two agents are negotiating. If there is some option that both agents have unassigned, but neither prefers, it is unlikely to be offered (unless the agents are very busy). As such each agent might believe the other has the option assigned. However, since neither agent wants to meet at this time, the misclassification is not very significant. Secondly, suppose two agents are negotiating and one agent is very busy. There may be some times that the busy agent regularly offers that it has unassigned but does not prefer. It is likely that the other agent will believe these are preferred options for the busy agent. This is a misclassification, but since the busy agent is willing to schedule meetings at these times, again, the misclassification is not particularly harmful. We have run a wide range of experiments (with different negotiation, preferences and initial calendar settings). We found that the most common misclassifications were mistaking the category unassigned for preferred and the category unassigned for assigned. The first misclassification mostly occurred when the agents had relatively full calendars (e.g., when there were no times that they both preferred) and the second, when the agents had relatively empty calendars.

5.7.2 Improving Efficiency

Figure 5.4 shows the significant reduction achieved in the average number of negotiation rounds through learning. This Figure, clearly demonstrates the usefulness of the abstraction we constructed and the validity of the approach. In this experiment, we ran 100 trials and the agents both offered 1 time-slot per negotiation round. There were 5 agents in the system, and the initial meetings involved all 5 agents, but all the new meetings were only between two of the agents - the agent we were evaluating and one other. These two agents had a small overlap in preferences at midday. One agent liked morning times, the other afternoon times, but both liked midday. Despite the fact that the agents we were evaluating learned online, there was a significant performance improvement from learning after only 20 meetings were scheduled. After 80 meetings were scheduled the average number of negotiation rounds when both agents were learning was *almost half* that of the average number for the no learning case. Figure 5.4 shows that performance was slightly better for the case where both the agents learned, than the case where just one agent learned. The decrease in scheduling cost achieved through learning comes at no cost to schedule quality, so this reduction in cost represents an overall improvement in performance.

We found that the improvements were less pronounced for random preferences. Figure 5.5 shows a graph from an experiment where the agents had randomly assigned preferences, with a 20% probability of a time being preferred. Nonetheless, the reduction in the average number of rounds to schedule a meeting is almost 25%. In general, we found that the performance improvement was

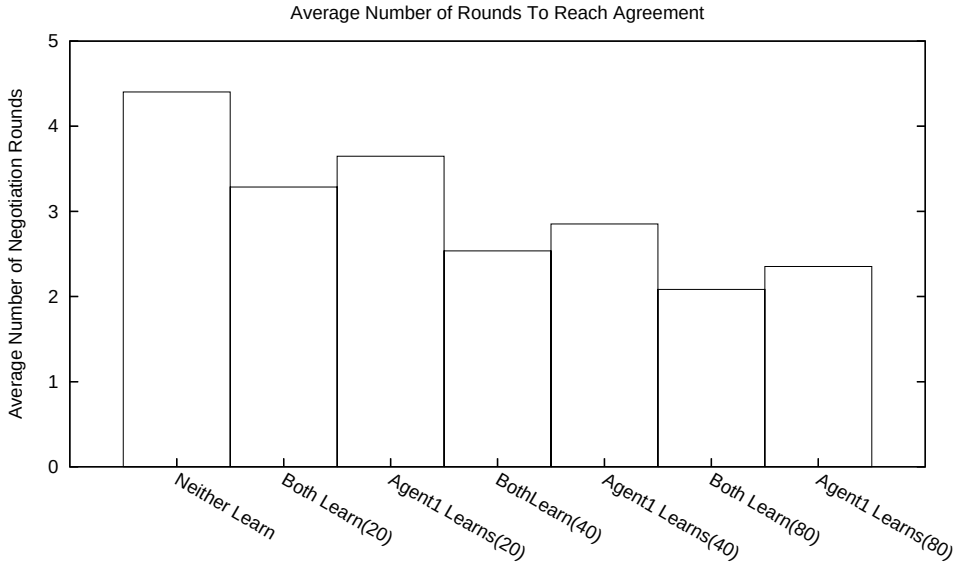


Figure 5.4: The graph shows the average number of rounds it took to agree upon meeting times between 2 agents after 20, 40 and 80 meetings had been scheduled. The agents had a small overlap in their preference functions.

most pronounced when there were a lot of potential options each agent could offer, but only a few options that were agreeable to all agents.

All of the gains in efficiency from learning shown, come at no cost in terms of preference to the agent. Despite the fact that the agent is learning to offer options that the other agent is more likely to accept, by combining learning with a negotiation strategy in the way we describe, the learning does not cause the agent to compromise its preferences more than is stipulated by the original strategy.

5.8 Summary

In this chapter, we introduced a domain independent method for modeling other agents in agreement problems. Our method is based on our observation that while the information revealed about an agent's negotiation function through the negotiation process is limited, we can nonetheless use it to build useful abstractions. We provide a method for constructing these abstractions, and

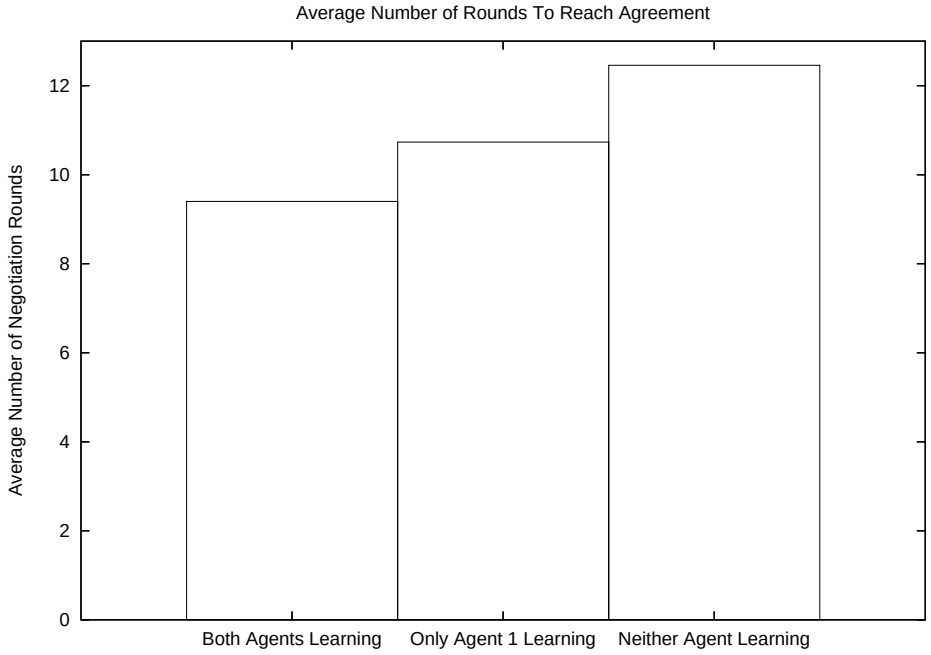


Figure 5.5: The graph shows the average number of rounds it took to agree upon meeting times between 2 agents after 20 meetings had been scheduled. The agents' preferences were randomly assigned.

demonstrate the usefulness of our approach in the multiagent meeting scheduling domain. We show that by learning models of other agents online, an agent can significantly improve the efficiency of its negotiation without sacrificing its preferences.

The success of the learning showed in this chapter relied on their being consistency in agents' offers related to their underlying preferences. In a future chapter we show how agents can learn online when each other are likely to offer particular options in order to trade off their preferences for negotiation time according to their individual utility functions. In another chapter, we show how an agent can learn online to choose the best negotiation strategies.

Chapter 6

Adapting Strategy Selection

In EIMAPPs, an agent may have to negotiate with other agents that exhibit a wide range of negotiation behavior. In this situation, learning to choose between a set of different negotiation strategies can benefit an agent. The best strategy for negotiating a variable is largely dependent on who the agent must negotiate with. In this chapter, we show how an agent can learn to select good strategies for negotiating with another agent using experts/bandit style algorithms.

6.1 Experts Algorithms

6.1.1 Outline

In an *experts* problem an agent must make a series of decisions about what action to take. For each action it takes, it receives some reward. Each time the agent needs to make a decision, it receives advice from a pool of experts. An *expert* is simply a strategy that recommends actions, possibly based on the previous actions taken and their rewards . Experts algorithms are methods for deciding what action to take based on the advice received from multiple experts.

One standard formulation of the experts problem is as follows. An agent is faced with a sequential decision problem where, at each decision point, it must choose an expert and take the action recommended by that expert. The agent then receives some reward from the environment based on the action it took, and on the environment. The environment can depend on the history of actions taken and/or on factors independent of the agent. In some settings (and in the setting we are concerned with in this chapter) the environment is viewed as taking an action at the same time as the agent. This action might be a move by Nature or by one or more other agents. An experts algorithm tells the agent at each decision point which expert's advice to follow. In the simplest

settings, the agent can compute the reward it would have received had it followed the advice of each of the different experts.

6.1.2 Regret

Regret is one performance criterion for an experts algorithm. We can compute for some history what expert would have yielded the highest reward overall. *Regret* is then the difference between how well the agent could have done, by always following the advice of this expert (against the sequence of actions chosen by the environment), versus how well it actually did. More formally, let the reward[1] received from following the advice of expert i at choice point t be r_i^t. The regret of the agent after T choices have been made is given by the following formula:

$$regret_T = \max_{over\ experts\ i} \sum_{t=0}^{T-1} r_i^t - \sum_{t=0}^{T-1} r_{x_t}^t$$

where x_t denotes the expert the agent chose at choice point t.

A common goal for an expert's algorithm is to *minimize regret*. We say that an algorithm minimizes regret if its average reward is as great as the reward of any one expert against any fixed sequence of opponent actions. An algorithm is said to be a *no-regret* algorithm if,

$$lim_{T->\infty} \frac{regret_T}{T} \leq 0$$

There exist algorithms that have provable regret bounds (and are no-regret), e.g., the randomized weighted majority algorithm [33], Hedge [2] and Exp3[2] [2]. These algorithms typically work by selecting strategies probabilistically. They start by having a uniform probability distribution over the strategies. As strategies are used, and reward received, the probability of choosing the higher reward strategies is increased, and the probability selecting the lower reward strategies is decreased.

6.1.3 Problems with the Regret Criterion

For the case where the opponent (or Nature) makes its moves simply on the basis of the current choice point, i.e., with no reference to the past behavior of the agent, regret is a very appealing performance metric. However, if the opponent bases its choice at each decision point t on the

[1]Note that this use of r to denote reward departs from our usage elsewhere in this thesis of r to denote the negotiation round. We use r for reward in this chapter for consistency with related work in the experts literature.

[2]Exp3 gives bounds on regret even for the partially observable case, where the algorithm can only see the reward of the expert it actually chooses.

past behavior of the agent, then regret is not a very good criterion. Regret simply looks at the difference between how well the algorithm's choices performed against the sequence of actions taken by the opponent, versus how well a fixed expert could have done against the *same sequence of actions*. However, if the opponent is basing its choices on the past behavior of the agent, then a different sequence of agent actions could have yielded a different sequence of opponent actions. For instance, the opponent would take a different sequence of actions if it were also learning. There exist algorithms that minimize regret against opponents whose choices can depend on the history (e.g., [2,5]), however, regret is not the best performance criteria in this case. Moreover since regret is not the best performance criterion, algorithms that aim at minimizing regret may not be the best choice.

In response to the problems with the minimum regret criterion, the Strategic Experts Algorithm (SEA) [17] and the Exploration Exploitation Experts algorithm (EEE) [15] have been developed. These algorithms aim at maximizing long-term average reward by taking into account the long-term effects an agent's choices have on the opponent. For certain settings these algorithms have a worst case guarantee that the average reward is asymptotically as large as that of the "expert that did best in the rounds of the game when it was played" [17]. More formally if we let avg_A be the average reward of the algorithm and avg_e be the average reward obtained from expert e when it was chosen by the algorithm, then (under certain conditions) $avg_A \geq avg_e$ in the limit. This differs a bit from a regret guarantee, because it compares the algorithm's average reward with the average reward each expert gave when it was used. The regret criterion compared the algorithm's average reward with the average reward each fixed strategy *might* have achieved at *each* choice point, assuming a fixed sequence of opponent actions. Stronger guarantees for SEA and EEE are given when some assumptions are made about the opponent [17].

6.2 Learning to Negotiate with Different Agents using Experts Approaches

6.2.1 Setting

In this chapter we focus on the problem of learning to negotiate with different agents. For example, a meeting scheduling agent is faced with a number of different negotiation situations or environments. Consider the problem of negotiating a two person meeting. One strategy may work well when negotiating a meeting with Agent B, and another with Agent C. We would like our meeting scheduling agent to be able to learn which strategy to use with Agent B and which to use with Agent C. There are other factors in the meeting scheduling problem that can differently affect the success of negotiation strategies, but who an agent negotiates with is likely the most important factor. We note that it is largely the other agent's strategy, preferences, and constraints, that will determine the success of our learning agent's negotiation strategies.

6.2.2 Motivation for our Experts-Based Approach

There are three main factors that make learning to negotiate with other agents very challenging. Firstly, the space of possible negotiation strategies is very large. Secondly, choosing the right strategy for negotiating a particular variable, given an agent's limited knowledge of the situation (e.g., its limited knowledge of the constraints, preferences and strategies of other agents) is not trivial. Thirdly, the learning needs to occur online, so we face an exploration/exploitation trade-off. At the start the agent has no information about strategy performance so it can just choose a strategy at random, but once it has collected some information, it must trade-off trying new strategies, exploring the use of strategies it has tried before more, or using the strategy that experience recommends.

Instead of having the agent learn in the entire space of negotiation strategies, we propose that the agent's attention should be restricted to a diverse set of useful strategies. These strategies could be learned, or chosen by a human. Given a set of diverse strategies, an agent then needs to *learn to choose the best strategies for negotiating with different agents*. Unfortunately, there are a huge variety of factors that can influence the effectiveness of a negotiation strategy. These factors include the strategies of other agents, their constraints and preferences, and any other significant properties of the variable being negotiated.

One approach, would be to try and model the relevant factors. An agent could try and identify different aspects, including, the strategy, preferences, and constraints of each other agent. However, given that agents may only rarely interact with each other, learning a model of another agent is clearly very difficult. Furthermore, even if an agent *does* have a lot of accurate information about the situation it currently needs to negotiate in, it may be very hard to work out what strategy would match the situation best. As such, it is worthwhile exploring an approach where agents learn what strategies to select by *observing their own rewards*, as opposed to trying to model other agents and the state of the system. Setting up the learning problem in this way allows us to use experts algorithms. Experts algorithms have a strong theoretical motivation, and in we demonstrate their usefulness experimentally for the multiagent meeting scheduling domain in this chapter.

6.2.3 An Experts Approach for Learning to Negotiate with Different Agents

We propose that to learn which strategy to use with a particular agent, say Agent B, the learning agent can use an experts algorithm. To do this, we simply think of each of the negotiation strategies available to the learning agent as one expert. The experts algorithm is then used to instruct the learning agent on what strategy to use each time it negotiates a meeting with Agent B. This gives the learning agent a principled way to trade-off exploration and exploitation to find the best strategy for negotiation with Agent B. So for every agent the learning agent negotiates with, we are going to use an experts algorithm to select the negotiation strategy. This allows the learning agent to learn over time what strategy to select for negotiating with each other agent.

So far we have only mentioned variables with two decision makers. Suppose that our learning agent is initiating a variable with many decision makers. The learning agent negotiates separately with each decision maker, i.e., it exchanges option proposals with each other agent. In this case, the learning agent will select a possibly different strategy for negotiation with each of the decision maker agents according to the experts algorithm running for that agent. When the learning agent is decision maker, but not an initiator of a variable, it has to negotiate with the variable's initiator. In this case, the learning agent will select the negotiation strategy according to the experts algorithm it is running for the variable's initiator. The Approach is summarized in Algorithm 6.2.1.

Algorithm 6.2.1 select-strategies-learning

```
1:  let the agent's name be A.
2:  if a new variable V arises then
3:
4:    if A is the initiator of V then
5:
6:      for each agent A_i ∈ D_v − {A} do
7:        if A has not not negotiated with A_i before, initialize an experts algorithm for A_i
8:        negotiate with A_i according to the experts algorithm for A_i until V is assigned
9:
10:     for each agent A_i ∈ D_v − {A} do
11:       update the experts algorithm for A_i according to the reward from assigning V
12:   else
13:     if A hasn't negotiated with initiator, i, of V before, initialize experts algorithm for i
14:     negotiate with i according to experts algorithm for i until V assigned
15:     update the experts algorithm for i according to the reward from assigning V
```

In the two sections that follow we describe in detail how we applied two different experts approaches to the problem of learning to negotiate with different agents. We then demonstrate the effectiveness of our approach experimentally in the multiagent meeting scheduling problem, in particular for the case of scheduling two-person meetings.

6.3 Playbook Approach to Strategy Selection

6.3.1 Introduction to Plays

In this section we show how an experts-based playbook approach [6] can be adapted to the problem of selecting strategies in multiagent meeting scheduling [13]. The playbook approach is based on a regret minimization algorithm so it is of most interest when the other agents are not also learning.

The playbook approach was introduced within the *small-size robot soccer* league, in which two teams of five small, wheeled, and fast robots compete against each other. An overhead camera above the playing field provides a complete view of all the robots to an external off-board computer that can then centrally perform coordinated team planning. A play is defined as a team plan, with applicability and termination conditions, roles, and actions for each robot in a team. An offensive play, for example, might be applicable whenever the ball is in the opponent's half of the field, and terminate either when a goal is scored, or the applicability condition is violated. A *playbook*, captures all the plays that are available to the team. A play is described in a simple language to ease its definition [6].

The playbook represents a set of strategies to potentially respond to different types of opponents. Each play has a specific weight that determines its probability of being selected. The playbook with its multiple weighted plays serves two main research goals: (i) the clear identification of a set of available team coordination strategies, and (ii) the ability to adapt the weights of the plays as a function of the achieved reward. Appropriate online adaptation of the weights of the plays provides a method to learn to respond to fixed opponents [6].

6.3.2 Outline: Plays for Strategy Selection

Negotiated agreement problems have a number of important features in common with small-size robot soccer. In both domains, the space of available strategies is huge. It is not possible for agents to adapt online if they must consider the entire space. Furthermore, the environment in both domains is dynamic, the models of the 'opponents' are unknown, and online learning is required. In this section, we discuss how we adapt the plays formulation to the problem of learning how to negotiate with different agents.

We can map the plays terminology, from robot soccer, to the multiagent agreement problem.

- Plays correspond to complete negotiation strategies.
- The opponent corresponds to the agent the learning agent is negotiating with.
- The playbook corresponds to the set of negotiation strategies available to the learning agent.

The playbook approach to robot soccer considered all the opponent agents as an opponent team and adapted play selection to the team, as opposed to individual opponent agents. In the multiagent agreement problem, we consider each agent a separate 'opponent'. At any one time a negotiation agent may be negotiating with multiple 'opponent' agents, whereas in robot soccer only one opponent team is played at a time. In the multiagent agreement problem the learning agent must adapt strategy selection for each of the different agents it negotiates with simultaneously.

In robot soccer, a team only plays against one team at a time. In this domain, it is fairly reasonable to assume that each game is independent, i.e., that a team's behavior in one game does not affect future games.[3] In agreement problems however (e.g., meeting scheduling), an agent's constraints can change as variables are assigned. As such, negotiating a variable with one agent can affect how a variable is negotiated with another agent. As we show in this chapter however, this does not stop the plays approach from being effective in the meeting scheduling domain when the other agents are fixed.

We let a negotiation strategy consist of 3 elements:

1. an applicability condition

2. a method for deciding at each negotiation round what options (if any) to offer

3. a method for deciding when to give up

This is a very flexible outline for a negotiation strategy. It makes no restrictions on the number of options (if any) an agent offers in each round. Furthermore, there is no restriction placed on what options the agent chooses, or how it makes the choice. Algorithms 6.3.1, 6.3.2, show an example strategy, Offer-k-b designed for the meeting scheduling problem, that offers k new available options each round, and after b rounds starts trying to compromise according to the proposals it has received. If necessary, Offer-k-b will offer options that would require an already scheduled meeting to be bumped (moved to another time). Depending on the values of k and b, this strategy can be very selfish and cause the negotiation to take a long time. As such, if the 'opponent' agent is very important the strategy is only applicable if the value of k is large and the value of b is small.

Algorithm 6.3.1 applicable-Offer-k-b(k,b,other-agent)

```
if the other-agent is very-important and (k < 20 and b > 5) then
    return false
else
    return true
```

[3]Dependencies *are* possible in the robot soccer domain, e.g., if the human designers of teams hand tune their robot teams according to the past play of their opponents.

Algorithm 6.3.2 Offer-k-b-negotiation-algorithm(round,meeting-negotiation)

```
if round > 50 then
  abandon negotiation
  return
else
  if round ≤ b then
    Offer A's k most preferred, available, un-offered times.
    return
  else
    use compromiser sub-strategy which works as follows
    if A is not the initiator then
      if ∃ times offered to A that A has available but has not offered then
        offer most preferred such time
        return
      else
        offer the time proposed by the initiator that corresponds to the existing meeting with
        fewest decision makers
        return
    else
      sort times proposed by other agents by number of proposals
      of times with highest number of proposals if any are available offer the most preferred
      such time
      otherwise offer the unavailable time corresponding to the meeting with the fewest decision
      makers
      return
```

The learning agent runs a different instance of the playbook algorithm (described in more detail in the next section) for each agent it negotiates with. When the learning agent needs to negotiate with a particular agent, e.g., agent A, the learning agent uses the playbook algorithm for agent A to select the strategy to use.

The learning agent considers the execution of a strategy to be complete when:

- the new variable has been assigned
- any variables that were bumped (that the learning agent is involved in) have been reassigned

A strategy is also considered to have been completely executed if the learning agent has given up on assigning the new variable, or on reassigning a bumped variable. Each time a strategy terminates, the playbook weights are updated according to the success of the strategy.

6.3.3 Weight Adaptation and Strategy Selection for Negotiation

For each 'opponent' agent, the learning agent must learn which strategies to select. The learning algorithm [6] has two key components:

1. a rule for updating the weights on strategies in the playbook, and
2. a rule for selecting the strategy to apply based on these weights.

In this section, we briefly describe the approach and its basis in the experts literature.[4]

Algorithms for selecting experts with no regret can be used to select plays [6]. Similarly to [6] we consider a formulation of regret that takes into account the fact that not all plays (or strategies) are applicable at each choice point by using the notion of *Sleeping Experts* [21]. An expert is awake when it is applicable at a particular choice point, and asleep otherwise. Following the notation used in [6], we let $a_i^t = 1$ if expert i is awake at time t, and $a_i^t = 0$ otherwise. Then if $\Delta(n)$ is the set of probability distributions over all n experts, the sleeping regret (SR) after T choices is:

$$SR_T = \left(\max_{x \in \Delta(n)} \sum_{t=0}^{T-1} \sum_{i=1}^{n} a_i^t \left(\frac{x(i)}{\sum_{j=1}^{n} x(j) a_j^t} \right) r_i^t \right) - \sum_{t=0}^{T-1} r_{x_t}^t$$

where x_t is the expert selected at time t, and r_i^t is the reward associated with expert i at time t. The formula gives the reward the agent could have received if the best possible distribution over awake experts had been selected at each choice point, minus the reward the agent actually achieved.

In the context of plays, unlike in the traditional experts problem, agents only find out the reward of the action they actually take. In order to account for this, the playbook approach [6] combines elements of the Exp3 algorithm [2] (which handles the problem of unknown rewards) with the sleeping regret approach [21]. We describe the approach here, and use it to adapt strategy weights for each 'opponent' agent, and select the negotiation strategy to use according to these weights.

To handle the problem of unknown rewards we let r_i^t be $\hat{r}_i^t$ where, $\hat{r}_i^t$ is 0 if i is not selected at point t, and $\frac{r_i^t}{Pr(x_t = i)}$ otherwise. Let the weight for strategy i at decision point t, be w_i^t. The value $e^{r_i^t}$ is denoted as m_i^t, and we refer to this value as the multiplier and use it to adjust the weights according to the reward received from carrying out the negotiation strategy (or play). The probability that the strategy chosen at point t, denoted x_t, is strategy i is given by the following equation:

[4] For a more complete treatment we refer to the reader to [6].

$$Pr(x_t = i) = \frac{a_i^t w_i^t}{\sum_j a_j^t w_j^t} \tag{6.1}$$

Once strategy x_t has been executed, and the reward $r_{x^t}^t$ received, we update the weights as follows:

$$w_i^{t+1} = \hat{w}_i^t . N_i^t \tag{6.2}$$

where $\hat{w}_i^t = w_i^t$ for i not selected, but for i selected:

$$\hat{w}_i^t = w_i^t (m_i^t)^{\frac{1}{Pr(x_t = i)}} \tag{6.3}$$

The N_i^t term is used to ensure that sleeping does not affect a strategy's probability of being chosen. $N_i^t = 1$ if $a_i^t = 0$ and otherwise:

$$N_i^t = \frac{\sum_j a_j^t w_j^{t-1}}{\sum_j a_j^t \hat{w}_j^t} \tag{6.4}$$

The algorithm is summarized in Algorithm 6.3.3.

Algorithm 6.3.3 plays-algorithm [6]

weights are initialized to 1 for all experts
for each time step t **do**
play/strategy is selected according to the probability distribution in equation 6.1
weights updated using equations 6.2, 6.3, and 6.4

Instead of directly dealing with rewards the playbook approach assigns the multiplier, (m_i^t), a value according to the success or failure of the play. To apply the approach to negotiation we need to decide how we are going to set the multiplier. The multiplier specifies the degree to which the success or failure of a strategy affects the strategy's weight. We base the multiplier on a model of semi-cooperative utility. Thus, the multiplier takes into account:

1. the user's preference for the option agreed upon using the strategy
2. the number of rounds taken to finish the agreement process

Utility	Multiplier
$U(i) > 0.75 * maxU$	1.75
$U(i) > 0.5 * maxU$	1.5
$U(i) > 0.25 * maxU$	1.25
$U(i) > 0$	1
$U(i) > 0 - 0.25 * maxU$	0.75
$U(i) > 0 - 0.5 * maxU$	0.5
$U(i) > 0 - 0.75 * maxU$	0.25

Table 6.1: The multiplier is given by the first row for which the entry in the first column evaluates to true

We use the user's specific utility function and highest preference value to estimate the maximum possible utility a negotiation strategy can achieve. We then set the multiplier according to how the reward actually achieved relates to this maximum. The multiplier is set according to the first row of Table 6.1 that applies. Also note that if the negotiation strategy fails to assign the new variable, or to reassign bumped variables, a failure has occurred. We use a multiplier of 0.25 for this case. There are many ways in which numbers for the multipliers could be chosen. The motivation behind the particular choices in the table is twofold. Firstly, we wanted the numbers to be from a small range so that the weights on strategies would not oscillate too much with each variable negotiated. Secondly, we wanted the number to evenly balance gains and losses.

In our experiments, we use the multiagent meeting scheduling domain, and an additive semi-cooperative utility function. The utility function is made up from the following components:

1. the user's preference for the time-of-day (tod) the new meeting is scheduled for: $p(tod)$.
2. the increase (or decrease) in utility from moving other meetings, i.e., for all meetings that were moved, the agent's utility is increased by
$$\sum_{moved} p(tod_{new}) - \sum_{moved} p(tod_{old})$$
3. the number of negotiation rounds r required to schedule the new meeting and move any old meetings.

Recall that the additive semi-cooperative is parametrized by two constants α and β which specify the relative importance of preferences and negotiation cost. Formally the user's utility for the outcome of a negotiation strategy in our experiments is modeled as:

$$U(i) = \alpha(p(tod) + \sum_{moved} p(tod_{new}) - \sum_{moved} p(tod_{old})) - \beta r$$

We note that the playbook approach is likely to perform best if the 'opponent' agent is using a fixed strategy and if changes to the environment (i.e., the calendars) are not affecting the rewards. If the other agent is also learning, then in the terminology of [2], we are dealing with a *non-oblivious* adversary. As such, since the playbook approach builds on Exp3, the theoretical bounds are weaker.

6.4 Exploration-Exploitation Experts Approach to Strategy Selection

6.4.1 Setting Definition

The Strategic Experts Algorithm (SEA) [17], as well as a generalized version, Exploration-Exploitation Experts (EEE) [15] are designed for the following setting. Let A be our learning agent. At each time stage t (same as time in our previous discussions), A must choose an action ω_t from some set Ω. Simultaneously the environment, in which A is acting, chooses some state $\beta_t \in B$. The action and the state combine to determine the reward A receives according to some function $R(\omega_t, \beta_t)$. The choice the environment makes can depend on many factors (e.g., stochastic variables), but in particular it can depend on A's past actions. The agent has a finite set of experts (or strategies) available to it. Each expert/strategy recommends *one* of the actions at each time stage t. An expert/strategy is a function, mapping possible histories up to time stage t, $h_t \in H_t$, to an action in Ω. Each possible history up to stage t is of the form $h_t = (\omega_1, \beta_1, ..., \omega_{t-1}, \beta_{t-1})$. In other words, the experts/strategies can take into account the history.

6.4.2 Relation to Repeated Games

Note, that if we consider the state choice of the environment to be simply an action choice, then each stage is a two player matrix game, and the whole process can be thought of as playing this game repeatedly. If we want to think about a n-player repeated game, we can consider the environment's state choice to be the joint-action of the other $(n-1)$ players.

6.4.3 EEE for Selecting Negotiation Strategies

The key idea behind SEA and EEE is that when actions affect the environment and other agents, each expert/strategy needs to be used multiple times in succession to gauge its effect. We propose an approach to selecting negotiation strategy when negotiating with adaptive 'opponents', that involves maintaining a separate EEE algorithm for each 'opponent' agent. This allows an agent A to learn online what strategies work best with each other agent. In this section, we describe SEA and EEE in terms of our negotiation problem.

Let a *stage* denote the assignment of one new variable. A *phase*, is a sequence of stages, for which the same expert/strategy is used. Let a be the learning agent and let S_a be the set of all strategies available to A. Using this notation we can define the exploration and exploitation components of the SEA and EEE algorithms.

- ***Exploration***: At the beginning of a new phase, A needs to select a strategy. With some probability A *explores*, by choosing a strategy $s \in S_A$ uniformly at random. A then uses this strategy in every stage of the new phase.

- ***Exploitation***: When A is not exploring it instead *exploits* its learned knowledge. When exploiting, A identifies the agent it is negotiating with, lets call the agent A_2, and picks the strategy with the highest past average reward for this agent (ties are broken randomly). As in the exploration step, A then uses this strategy in every stage of the new phase.

Algorithm 6.4.1 shows the details of the algorithms in terms of our negotiation problem. Bounds, of various kinds, on the performance difference between the best fixed strategy and the learning strategy in the limit, exist for EEE [15].

Algorithm 6.4.1 EEE Algorithm, adapted from [15]

$R_s^{A_x}$ is the average reward A has achieved by using strategy s when negotiating with agent A_x.
$N_s^{A_x}$ is the number of phases in which A has followed s when negotiating with agent A_x.
$S_s^{A_x}$ is the number of times A has negotiated using s with agent A_x.
i^{A_x} is A's phase count for agent A_x.
Initialization:
for $\forall A_x, s$ **do**
  $R_s^{A_x} = N_s^{A_x} = S_s^{A_x} = 0$ and $i^{A_x} = 1$
Explore/Exploit Decision: Explore with probability $p_i^{A_x}$.
if using the SEA restriction **then**
  $p_i^{A_x} = 1/i^{A_x}$.
With probability $1 - p_i^{A_x}$ exploit, else explore.
Let s be the strategy chosen.
Use and Update: Use s for the next n negotiations with agent A_x.
if using the SEA variant **then**
  $n = N_s^{A_x}$
  $N_s^{A_x} = N_s^{A_x} + 1$, $S_s^{A_x} = S_s^{A_x} + n$.
let $\hat{R}$ be the average payoff received over the n stages

$$R_s^{A_x} = R_s^{A_x} + \frac{n}{S_s^{A_x}}(\hat{R} - R_s^{A_x})$$

$i^{A_x} = i^{A_x} + 1$, go to 2.

6.5 Evaluation

In this section, we evaluate our approach to learning to select negotiation strategies. We study in simulation how well the two experts approaches we have discussed perform in the multiagent meeting scheduling domain. We start by describing the experimental set up, including the communication protocol, the negotiation strategies and the preference model. We then discuss the experiments and the results.

6.5.1 Communication Protocol and Bumping Meetings

In our experiments we have used a flexible negotiation protocol which we call, Reply-To-Initiator. The protocol algorithms for an initiator and non-initiator agent are in Appendix 1. The protocol operates in rounds. In each round the initiator makes an offer to the decision makers and waits for a return offer. When all the agents have responded, the initiator determines whether there is an intersection in the offers. An agent's offer in any round, is considered to be the offer they sent in that round, unioned with their offers in all previous rounds. If there is an intersection the agents go through a confirmation phase, to avoid conflicts with any variables that are being negotiated simultaneously. The protocol makes no restriction on which options the agents offer in each round or how many options they offer. The agents can even chose to make an empty offer. We note that in these experiments, unlike in some of the experiments we will discuss later, the agents schedule multiple meetings simultaneously.

The protocol allows for the bumping (moving or removal) of existing meetings. Sometimes in order to schedule a new meeting, a meeting the learning agent is involved in gets bumped. When this happens, the learning agent must negotiate a new time for the bumped meeting. We have setup the learning agents such that they do not learn from rescheduling bumped meetings, i.e., in the context of the playbook approach, the weights are not updated when a bumped meeting is rescheduled. However, the number of rounds it takes to schedule the bumped meeting is added to the number of rounds to schedule the new meeting, and thus it affects the utility of the strategy being used to schedule the new meeting.

6.5.2 Negotiation Strategies

We have implemented a number of negotiation strategies that comply with the protocol outlined. We use two of these strategies in our experiments in this chapter. The first strategy – Offer-k-b was previously described (see Algorithm 6.3.2.). This strategy is parametrized, and hence it covers a large number of distinct strategies. The second strategy we use is called Availability-Declarer (Algorithm 6.5.2.). This strategy can be useful in practice, when the agents are very busy. The key feature of this strategy is that it offers all the available times in the first week straight away.

In subsequent negotiation rounds it does the same for later weeks.

Algorithm 6.5.1 applicable-availability-declarer(other-agent)

```
if importance(other-agent) ≥ moderately-important then
  return true
else
  return false
```

Algorithm 6.5.2 availability-declarer-MS-algorithm

```
if round < 50 then
  if round > 5 then
    use compromiser sub-strategy from Algorithm 6.3.2
    return
  offer all its available times in current-week + round
  return
else
  abandon meeting
  return
```

6.5.3 Preferences

We use a simple model of time-of-day preferences. Each agent has a preference ordering over morning times, middle of the day times and afternoon times. Within these time ranges the agent's time-of-day preferences are equal. The agent's utility from scheduling a new meeting is defined by the equation given previously, i.e,

$$U(i) = \alpha(p(tod) + \sum_{moved} p(tod_{new}) - \sum_{moved} p(tod_{old})) - \beta r$$

6.5.4 Experiments and Results: Plays-based Approach

We have empirically evaluated the effectiveness of using a plays approach to select negotiation strategies. The experiments we describe consist of one learning agent, which we are evaluating, and three fixed strategy agents of varying preferences and busyness. The learning agents have three strategies in their playbooks – Availability-Declarer, Offer-10-5 and Offer-3-5. In the experiments discussed, these strategies are always applicable.

Convergence

In each experiment, the agents schedule approximately 80 new two person meetings (we restrict our attention to two-person meetings to simplify the discussion). The learning agent is an attendee (not an initiator) of each of these 80 meetings. We show how the learning agent's playbook weights converge to sensible strategies for each of the fixed strategy agents.

In our first experiment, the learning agent's time preference is morning, then midday and then afternoon. The α and β values of the learning agent's utility function are 4 and 0.1 respectively. The agent's calendar is approximately 25% full when the experiment is started. Unlike the meetings we schedule in the testing phase, the initial meetings in the calendar can involve any number of the agents.

Figure 6.1 shows how the learning agent's playbook weights adapt for Agent2. Agent2 starts out with a similar number of initial meetings to the learning agent, uses the Availability-Declarer strategy, and has the same time preferences as the learning agent. Figure 6.1 shows how the playbook weights quickly converge to the Availability-Declarer strategy. While the other two strategies are also likely to work well in this instance, the Availability Declarer strategy offers the possibility of resolving the negotiation faster. Since the learning agent and Agent2 have the same preferences, there is no strategic advantage to the learning agent only releasing its availability slowly.

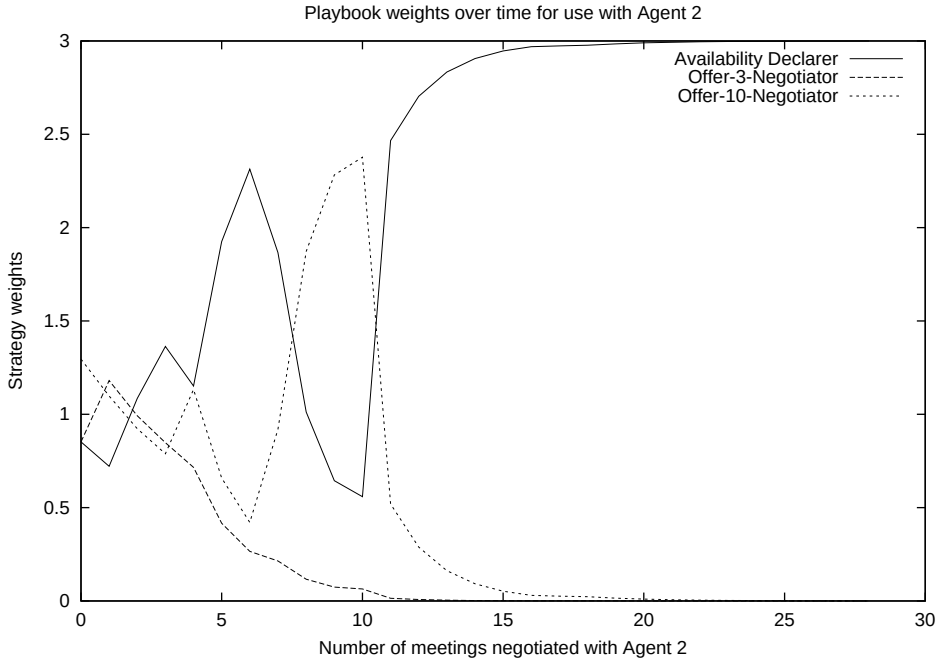


Figure 6.1: Playbook approach: weights adaptation for Agent2

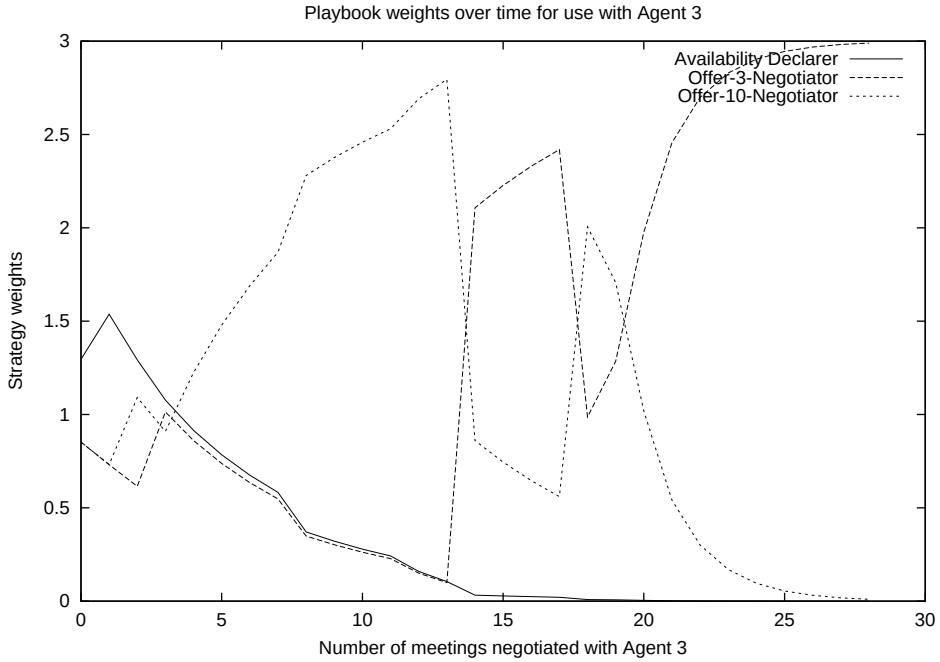


Figure 6.2: Playbook approach: weight adaptation for Agent3

Figure 6.2 shows the weight adaptation for Agent3. Agent3 uses the
Availability-Declarer strategy and starts out with a similar calendar density to the learning agent, but with opposite preferences. Agent3 most prefers afternoons, then the middle of the day, and then the morning. Figure 6.2 shows that the learning agent quickly establishes that the Availability-Declarer strategy is less useful for negotiating with Agent3 than the Offer-10-5 and Offer-3-5 strategies. After about 25 meetings have been scheduled the weights converge on the the Offer-3-5 strategy. Note that the Availability-Declarer strategy is a poor choice for use with Agent3. When both agents negotiate with this strategy, the initiator (always Agent3 in these experiments) is likely to quickly find a large intersection of available times. The initiator can choose its most preferred time in this intersection and since Agent3's and the learning agent's preferences clash, the time chosen will likely be bad for the learning agent. The learning agent has a clear strategic incentive to declare its available times more slowly and in order of preference. Since the learning agent's utility function rates achieving good times much higher than minimizing the number of negotiation rounds, it converges on the Offer-3-5 strategy rather than the Offer-10-5. This is despite the learning agent's calendar being quite full (93%), and hence mutally available slots fairly rare, by the time the experiment concludes.

Figure 6.3 shows the weight adaptation for Agent4. Agent4 has similar preferences (midday, morning, then afternoon) to the learning agent. Agent4 uses the Offer-10-5 negotiator and starts with a dense calendar (about 80% full). Figure 6.3. shows that the learning Agent quickly determines that

the Offer-3-5 strategy is not very effective when dealing with a very busy agent that has similar preferences. After approximately 15 meetings have been scheduled, the learning agent converges on the Availability-Declarer strategy.

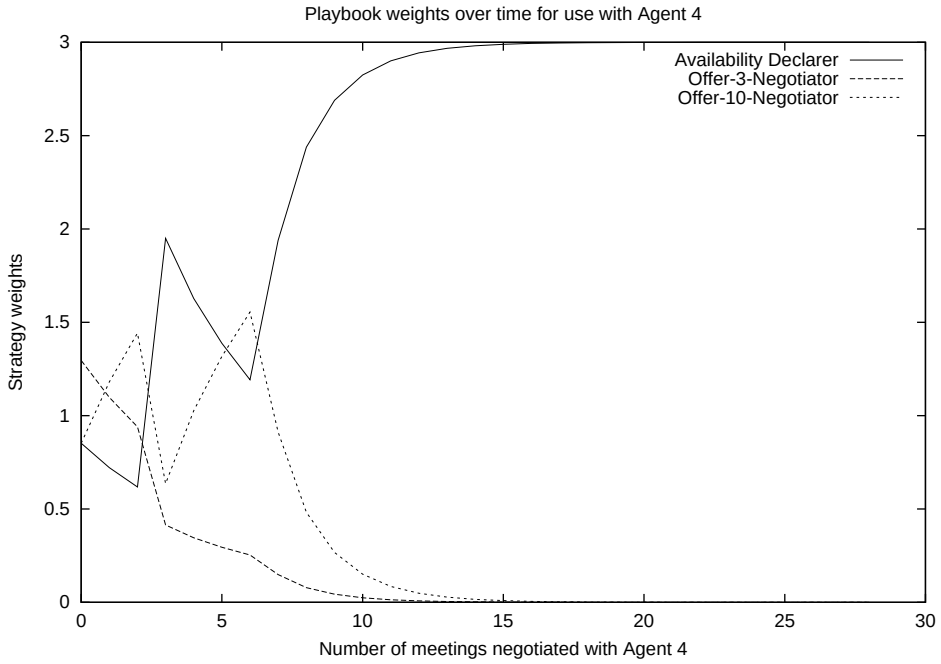


Figure 6.3: Playbook approach: weight adaptation for Agent4

We ran the same experiment described above but with a different utility function for the learning agent and different initial calendars. The utility function had α as 4, and β as 1. This change caused the weights to converge on Availability-Declarer for each of the agents, since the negative effect of negotiation length was greatly increased.

The figures show some oscillation in the weights, particularly initially. This oscillation is due to the way the algorithm works. Lower probability strategies (i.e., strategies with lower weights) have their weight boosted more when they perform well than higher probability strategies. However, as we have discussed the process converges over time in these experiments.

Performance

No regret algorithms bound the average difference between the performance of the learning algorithm, and the best fixed strategy in the limit. However, since a learning agent does not schedule an infinite number of meetings with each other agent, it is important to examine how well the learning algorithm performs in practice.

We used one agent designated as the learning agent and the three fixed strategy agents previously described and ran 10 trials. In each trial we had the designated agent schedule all the test meetings using the playbook approach, an approach that randomly selects one of the playbook strategies, and each of the fixed playbook strategies (clearing the calendars to their original state for each different setting). In each trial, the agents' calendars were randomly initialized with 160 meetings. The number of test meetings that the agents had to schedule was 200, but the calendars were cleared to their initial state after every 20 meetings. This reflects the common scenario where people have a set of meetings that occur weekly and new meetings that arise over time. Figure 6.4 shows the results.

The performance of each algorithm is measured in terms of the total utility it achieves averaged over each of the trials. This utility is calculated for an individual trial as follows. Each time the calendar needs to be cleared to its initial state (in this case it is cleared every 20 meetings) the calendar's utility is evaluated. The utility evaluation of a calendar, *cal* is defined as:

$$eval(cal) = \sum_{meeting\ times\ t \in cal} p(t)$$

The utility of scheduling a series of meetings is then:

$$\alpha * (eval(finalCal) - eval(initialCal)) - \beta * r \tag{6.5}$$

where r is the number of rounds it took to schedule all the meetings. When running each trial a record is kept of utility achieved so far. Each time the calendar is to be cleared, the utility from scheduling the new meetings is calculated according to Equation 6.5 and added to the current utility sum for the trial.

Figure 6.4 shows the play learning algorithm achieving higher average utility than playing a random strategy or using any fixed strategy. The learning algorithm used gives the strongest regret guarantees when the other agents are fixed. Figure 6.5 shows that the performance of the learning algorithm drops when we add a learning agent (that uses the same algorithm), to the three fixed agents. The overall performance drops because the plays algorithm does not perform well when scheduling meetings with the new plays-based agent that has been added. We found in general that the plays algorithm struggled when combined with learning agents. The aim of the approach is to minimize regret, and it does not consider the effect its action choices have on the other agents. This is clearly a problem when paired with agents that are also adapting. We noted that in self-play[5] the plays approach would fail to explore the strategy space in a steady manner - often converging too quickly to a strategy or experiencing large jumps in strategy weights in response to changes in strategy by another agent. This behavior resulted in reduced performance compared to the case where the plays approach learnt against fixed strategy agents. These results described here are typical of a variety of experimental configurations.

[5]We say an algorithm is run in self-play when multiple agents use the same algorithm (possibly with some different properties, e.g., preferences)

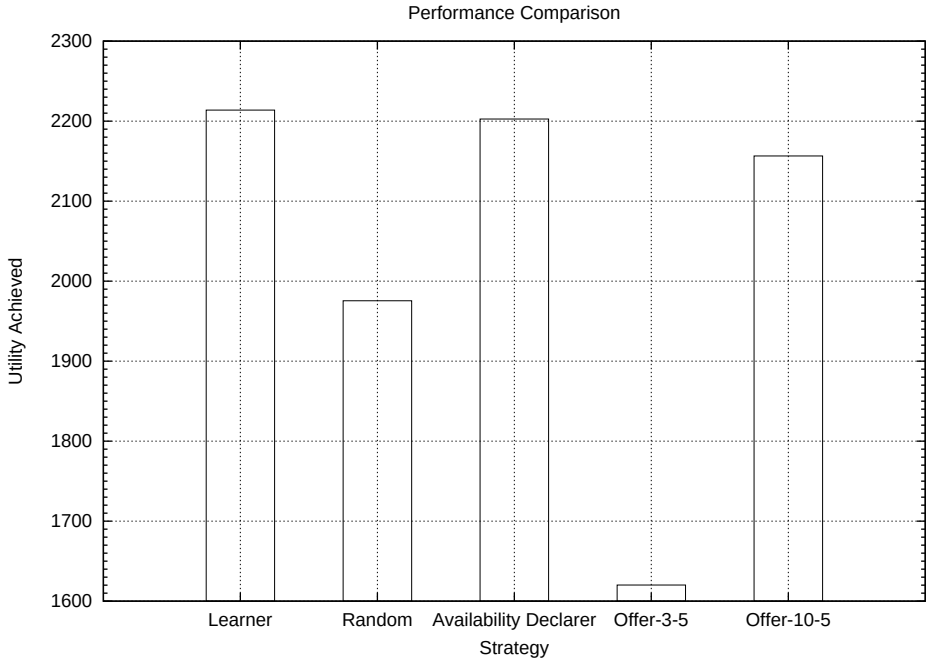


Figure 6.4: Performance of the plays-based approach and other algorithms *against only fixed agents*.

The performance of the playbook approach is greatly affected by the number of strategies in the playbook. Intuitively, we can see that it is harder to select the best strategy when faced with more possibilities. Analytically, we can also see this from the bounds given for Exp3 [2] (recall that the playbook approach is partially based on Exp3). These bounds include a $\sqrt{K}$ factor, where K corresponds to the number of negotiation strategies in our terminology. In the next section, we will show a case with a larger playbook where the plays algorithm is unable to perform better than the best fixed strategy when negotiating with fixed strategy agents. Given enough time the performance of the playbook approach would improve, but in meeting scheduling we need to see good performance very quickly.

6.5.5 Experiments and Results: EEE Approach

6.5.6 EEE Algorithm Setting

The two key parameters of EEE are the exploration probability and the phase length. We use the SEA exploration probability setting, i.e., $p_i^x = 1/i^x$. This causes exploration to decrease as the number of phases increases. In [16] three different exploration probability settings are discussed – explore-then-exploit, polynomially decreasing exploration, and constant-rate exploration. The

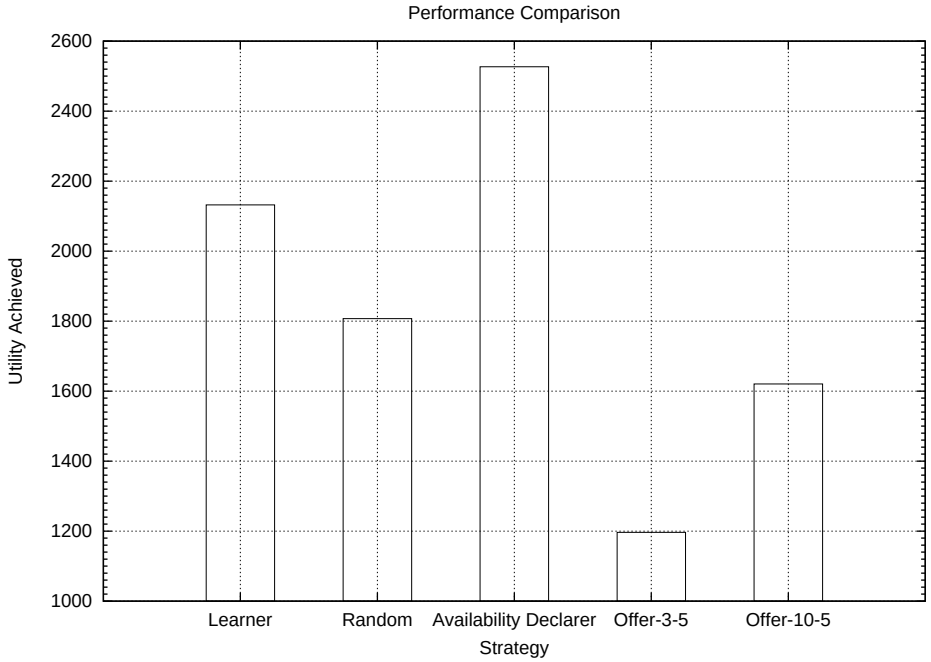


Figure 6.5: Performance of the plays-based learner and other algorithms *when another learning agent is added.*

SEA setting is version of polynomially decreasing exploration for which strong asymptotic results are provided [17]. The drawback of explore-then-exploit for the meeting scheduling domain is that if some aspect of another agent changes and the explore phase is over then the algorithm cannot adapt. In contrast, constant rate exploration can adapt quickly to changes, but in an environment like meeting scheduling, where there is likely to be a lot of consistency, it fails to take advantage of opportunities to exploit. The polynomially decreasing exploration takes advantage of consistency in the environment while still being able to adapt to changes. We have not attempted to tweak the polynomially decreasing exploration to achieve better performance in our experiments. Instead we have used the SEA exploration policy, for which performance bounds are given in [17].

6.5.7 Evaluation Procedure

The experimental procedure is similar to that described in the previous section. In all the experiments we describe, the meetings have two participants (the learning agent can initiate meetings). The initial calendars of the agents are populated with randomly generated meetings. The agents are then given a set of meetings to schedule. In each experiment one agent is designated as the agent to be evaluated. This agent uses the SEA/EEE learning approach to negotiate the set of meetings with the other agents. Before the agent schedules these meetings however, we record the

initial state of all the agents' calendars. This allows us to reset all the calendars and then have the agent we are evaluating schedule the meetings again with other strategies. In particular, we have the designated agent schedule the meetings using a random approach. Each time a meeting needs to be negotiated, the random approach simply selects from amongst the agent's strategy pool S_a uniformly at random. We also have the agent schedule the meetings using the plays-based approach and each fixed strategy from S_a. This allows us to compare the performance of our learning approach against the random approach and against each of the fixed strategies.

In Figure 6.6 we show that our EEE approach to strategy selection performs very effectively in self-play. We evaluated an agent learning with the SEA/EEE approach against 4 other agents also using the learning algorithm. The evaluation was done over 300 trials and in each trial 704 meetings were scheduled. The agents had the following strategies available to them: Offer-1-20, Offer-3-20, Offer-5-20, Offer-10-10, Offer-15-10, and AvailabilityDeclarer. We used a version of EEE with a fixed phase length of 5. The agents had a variety of preferences, favoring different times of the day and having different α and β values. This meant that different strategies generally performed differently against each of the agents.

Figure 6.6 shows the performance of our approach while it was learning. The Figure shows the utility each of the algorithms achieved averaged over the 300 trials. The error bars indicate 90% confidence intervals. The random approach does not appear on the graph. It achieved an average utility of 19720, signifigantly lower than any other approach. As the graph demonstrates, the EEE approach performed much better than the playbook approach. The playbook approach performed worse than most of the fixed strategies, but not signifigantly so. Given that the EEE algorithm is explicitly designed to learn in the presence of adaptive opponents it is not surprising that EEE outperforms the playbook approach. Significantly, EEE also performed better than using a single fixed strategy for all the agents. This demonstrates, that it is better to equip an agent with a pool of strategies and have it learn which to use, rather than to just give it the strategy that is best on average (supposing we even knew what this strategy was).

We found that the SEA/EEE approach was able to fairly quickly identify a good strategy for a given agent. However, since the other agents were also learning we did not generally see the algorithm fix on a very clear winner like the plays approach did against fixed strategy agents. One of the reasons the algorithm did not very clearly fix on a strategy, is that we used a fixed phase length. We found that when we used phases of length $n = N_s^{C_i^a}$ (i.e, phases that increased in length over time) that the algorithm did not get enough chance to explore, even if 100 meetings were scheduled with each agent. This lead to agents sometimes not identifying good strategies. By using a fixed phase length, we gave the algorithm a good chance of exploring all the strategies early on. However it was still sometimes the case that a particular strategy was never selected by the algorithm with a given agent. This could hurt the performance of the algorithm in the short term if it was unlucky.

There was some variation between trials, as shown by the confidence intervals. In most trials the SEA/EEE approach outperformed each of the fixed strategies. In some cases however its performance was lower than the best fixed strategy in the trial. The best fixed strategy in each

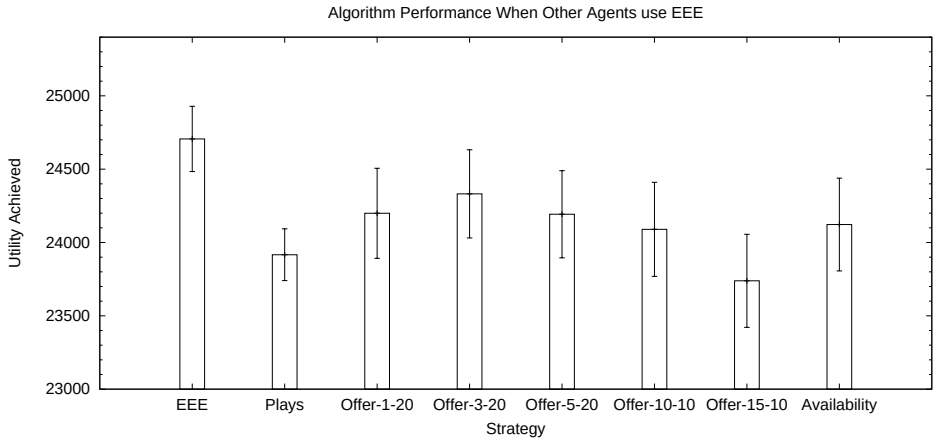


Figure 6.6: Graph showing the performance of the different algorithms against 4 agents each using the EEE approach over 300 trials.

trial varied. This further enforces the idea that it is better to equip an agent with a pool of strategies than a single fixed strategy.

Figure 6.7 shows that the SEA/EEE approach performs very well against agents using a no-regret approach. In this experiment we looked at performance against 4 agents that were each using the plays-based algorithm. The SEA/EEE approach significantly outperformed the plays-based approach, the random approach and each fixed strategy. It is interesting to note that the plays-based approach performed comparatively worse in self-play than against the SEA/EEE approach. This is mostly likely because the plays-based approach is more effective against fixed strategy opponents and our agents running SEA/EEE changed strategy less often (at least initially) than agents using a plays-based approach.

In the previous section we showed that the plays approach is quite effective against agents using fixed strategies. In Figure 6.8 we show an example of how the EEE algorithm performs against fixed strategy agents. For this experiment we chose a diverse set of fixed strategy agents. The agents are described in Table 6.2. The EEE approach significantly outperformed the playbook approach and all the fixed strategies. The playbook approach had a slightly lower average utility than each of the fixed strategies, but as the error bars indicate there was no significant difference. All the approaches greatly outperformed the random approach (not shown on the graph).

The playbook approach did not perform as well in this experiment as in the experiment in the previous section. The main reason for this, is the difference in the size of the playbooks. In this experiment, both the EEE and the playbook approach had 6 different strategies to choose from. In the previous experiment there was only 3 strategies in the playbook. We found that the playbook approach did not cope well with the increase in the number of strategies. In general, the weight became heavily concentrated on one strategy very quickly, causing the algorithm to not explore

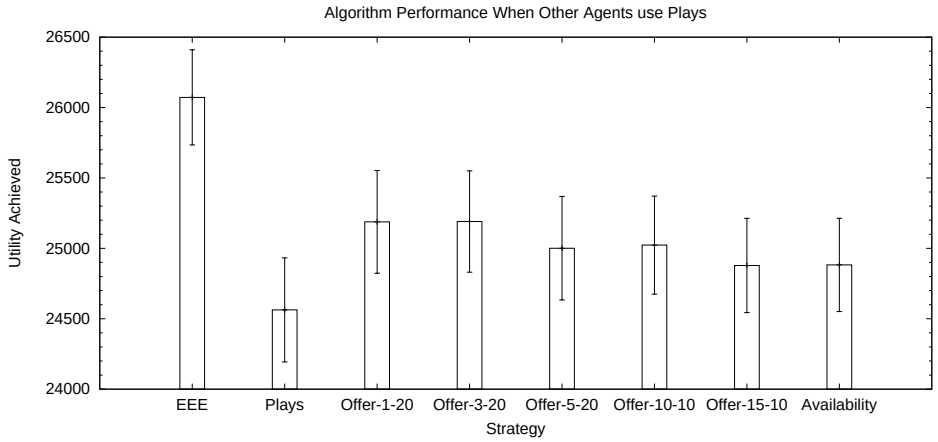


Figure 6.7: Graph showing the performance of the different algorithms against 4 agents each using the plays-based approach over 300 trials.

the different strategies enough early on (this is more of a problem when the playbook is larger). When another strategy was selected the weight of the new strategy tended to jump up very high if the strategy was successful (due to the probability term in the weight update formula). This sometimes caused the weight to converge too quickly on the new strategy. EEE was much more likely to explore all the strategies early on. In contrast, the playbook approach was sometimes able to outperform EEE when the strategy space was small. In that case, particularly if there was a poor strategy, EEE's tendency to explore more sometimes hurt average utility, and the fast convergence of the playbook approach was an advantage.

Agent Type	α	β	Initial Calendar Density
Availability Declarer	5	0.1	0.5
Offer-3-20	5	0.01	0.5
Offer-5-20	5	0.05	0.5
Offer-10-20	3	0.01	0.5

Table 6.2: Fixed strategy agents.

6.6 Summary

In this chapter, we showed how an agent can learn to improve its negotiation. The first key step in our approach, is to equip the agent with a small, diverse set of useful negotiation strategies. The space of all possible negotiation strategies is very large. As such, restricting the agent's attention to a relevant set cuts down the space the agent needs to learn in considerably. Given this

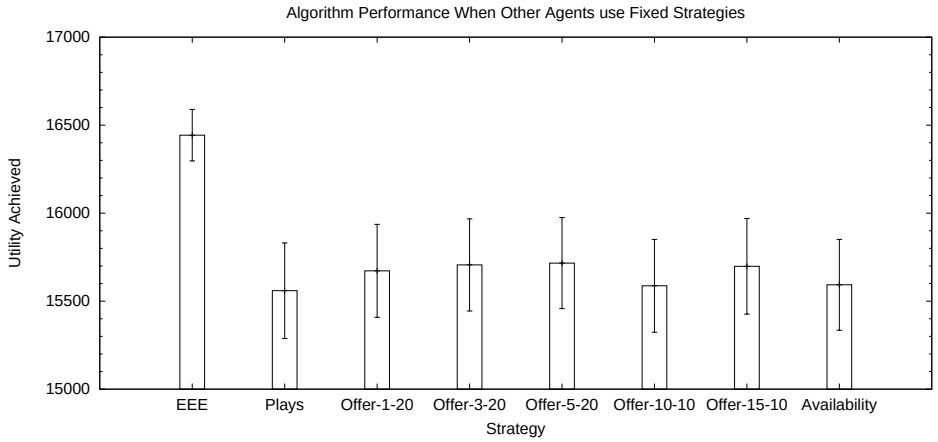


Figure 6.8: Graph showing the performance of the different algorithms against 4 agents each using fixed strategies over 300 trials.

restricted set of strategies, we showed how the problem of learning to negotiate with another agent can be framed as an experts/bandits problem. In particular we showed how two different experts algorithms, the playbook approach [6] and EEE [15] can be used to select negotiation strategies. We demonstrated experimentally in the meeting scheduling problem that the playbook approach works well in the presence of fixed agents, and that EEE works well in the presence of fixed agents, and agents that are also learning. Our approach to learning to improve negotiation through learning to select strategies is a departure from the previous work on agreement problems such as multiagent meeting scheduling, where the focus has been on attempts to build models of other agents - largely in cooperative settings.

Chapter 7

Predicting the Round of Offers to Increase SC-Utility

In this chapter, we focus *specifically* on SC-agents. In Chapter 2, we defined a semi-cooperative utility function, for an agent a, as having the following form:

$$U_a(o, v, r) = \gamma_a^r p_a(o, v), \gamma_a \in (0, 1)$$

where r is the round at which o is assigned to v. We further defined the rule that a semi-cooperative agent attempts to comply with when negotiating as follows:

For a variable, v, agent a selects an option o from $O_{\text{un-Offered}_a}$ (the options it has not yet offered for v) to offer in round r such that:

$$o_v(r) = \begin{cases} \underset{o \in O_{\text{un-Offered}_a}, r' > r}{\text{argmax}} U(o, v, r') \text{ if } \exists o, r' : U(o, v, r') \geq \epsilon_a \\ \underset{o \in O_{\text{un-Offered}_a} \& \nexists o' \text{offered by more agents}}{\text{argmax}} p_a(o) \end{cases}$$

In other words, the agent tries to offer the option that will maximize its utility, if that utility will exceed the minimum utility parameter ϵ_a, for some round r'.

In this chapter, we address the question, of what algorithms an agent can use to try and comply with this behavior rule and maximize its semi-cooperative utility function. We present two algorithms, supported by a method that learns models of other agents' behavior. Unlike in Chapter 5, we do not try to discover the motivations of the other agents, we instead model the result of their motivations only. We use regression algorithms to *predict, for each option, the round in which the agent will offer the option.* Our algorithms use these models of the other agents' behavior to inform their negotiation decisions and increase the utility they achieve.

7.1 Learning Goals

Our goal is to provide data that can assist a negotiation algorithm, which is trying to perform well against the function $U_a(o, v, r) = \gamma_a^r p_a(o, v)$. We restrict our attention to negotiation protocols where each of the agents has an $\mathcal{N}_{\text{range}}$ of $(1, 1)$ (must offer one option per-round) and an $\mathcal{N}_{\text{rule}}$ that requires that for a particular variable, each agent can only offer options they have not yet offered. From the form of the utility function, we see that in order to estimate the utility of an agreement (o, v), we need an estimate for the round in which agreement on o will occur.

Suppose the agent has an accurate estimate for the round in which each option will be agreed upon, if it were to offer the option. Then the agent would know its utility for each of the possible agreements (o, v). Intuitively, we can see that this information would be helpful to an algorithm trying to achieve a high utility agreement. Our goal is to learn to predict, for a given variable, the likely round of agreement for every possible option.

As discussed in Chapter 5, due to the complexity of agents' negotiation functions and the limited training data available in most agreement problems, it is difficult to build a detailed model of another agent's properties and negotiation behavior from negotiation histories. In Chapter 5, we addressed this problem by learning an abstraction that focused on how broad categories of preferences and constraints influenced behavior. In this Chapter, we focus solely on agent behavior, but work with a more detailed model.

7.2 Learning to Predict Rounds and Utility

From the negotiation history, we show that agents can learn a detailed model of when other agents offer options. Predicting the round in which other agents offer options allows an agent to estimate the utility of different negotiation outcomes, and weigh up the different offers it can make.

7.2.1 Negotiation History Data

For each variable the agent negotiates, it records the part of the negotiation history it sees. From these records, it extracts training data for each of the other agents it negotiates with: $\text{data}_a = \{(\text{option,round})\}$, e.g.,

$$\text{data}_a = \{(o_1, r_1), (o_2, r_1), (o_1, r_3), ..\}$$

The dataset specifies for each other agent the options our agent saw the other agents offering at specific negotiation rounds. Depending on the domain, the agent may include more features in the training data. For example, in the meeting scheduling domain, it may be helpful to add a feature

indicating if the time-slot (option) is a morning or afternoon time, since this could assist a learning algorithm in discovering further patterns in the data.

7.2.2 Prediction Method

In our approach, to predict the rounds when another agent will offer each option, the agent applies a regression algorithm to the training data. In the experiments in this thesis, we have used Linear Regression, but the regression algorithm and feature selection techniques can be chosen as is appropriate for the particular agreement domain.

For a two agent variable, the agent estimates it can achieve agreement for a particular option, by the round it predicts for the one other decision maker - provided of course it offers the option itself. In general however, for a variable, v, with decision makers $\mathcal{D}_v$, the agent can predict when option o will be agreed upon, by taking the maximum of the predictions for the agents in $\mathcal{D}_v$. If we let the predicted round for a_i and o_j be denoted by $\text{predR}_{a_i}(o_j)$ then the predicted round of agreement for option o_j for a variable v is:

$$\text{agreeR}(o_j, v) = \max_{a_i \in \mathcal{D}_v} \text{predR}_{a_i}(o_j)$$

The aim of making round predictions for options, is to be able to estimate the utility of offering different options for a given variable. The agent can use the round prediction method, to predict a round r for every option o it can offer for variable v. To estimate the utility of the outcome (o, v), the agent calculates $U(o, v, r)$. We say this is the agent's estimated utility for offering o for v.

7.2.3 Predicting Rounds for Unseen Options

An important consideration is how to handle options that are never offered. For a maximum number of decision makers and option set size, we can calculate the maximum number of rounds required (assuming the agents offer a new option in each round). Initially we provide a training example for every option where the round is this maximum, to represent the concept that the option is never offered. The first time an option is offered, the training example is removed. If an option is never offered, the training example remains.

7.3 Learning Online: Exploration and Exploitation

We provide an approach where agents learn to increase their semi-cooperative utility online, through a process of exploration and exploitation. Since the agents are always updating their models, they

are able to respond to changes in the behavior of other agents and adapt in an environment where other agents are also learning.

7.3.1 Exploration

Since SC-EIMAPPs are incremental, agents need to *learn online*. This also allows agents to adjust to changes in other agents' behavior. When an agent negotiates a variable with another agent, it *explores* with some probability, or *exploits*. The probability of exploration decreases as the number of variables negotiated with an agent increases. When exploring, agents use the 1-Preference-Order strategy (introduced in Chapter 2) and without learning, the SC-behavior rule. Using the SC-behavior rule without learning, simply means that if it is impossible for an agent, a, to do better than the minimum utility parameter ϵ_a, at some point in the negotiation (due to too many rounds having past) then a will try to reach agreement as quickly as possible, as specified in the ϵ_a condition of the behavior rule.

7.3.2 Exploiting the Learned Information

We consider the case where the agent offers one new option per round (according to its $\mathcal{N}_{\text{rule}}$). Let, o_w, be the option the agent estimates will maximize its utility. o_w has an associated predicted agreement round r_w. The agent would like to simply offer o_w and wait for it to be accepted (hopefully by r_w). Since the agent must offer a new option every round it may have to offer an option of lower estimated utility. There is a danger that such an option could be agreed upon. To try and avoid this, the agent can start offering options in the order of highest estimated round. If r_w is exceeded, the agent needs to revise its utility estimate for o_w, and offer any options with that utility. This is the idea behind the SC-Utility-Estimate-Strategic-Learner (Algorithm 7.3.1).

Algorithm 7.3.1 SC-Utility-Estimate-Strategic-Learner: Select-Option(v,options-trying-for, estimated-util, estimated-rounds, options-offered, eligible-options, curr-round)

```
1: sorted = sort-by-est-util(eligible-options, v, curr-round)
2: if |options-trying-for| < 0 then
3:    options-trying-for.add(top(sorted))
4:    estimated-util = est-util(top(sorted))
5:    estimated-rounds.add(est-round(top(sorted)))
6:    options-offered.add(top(sorted))
7:    curr-round+ = 1
8:    return top(sorted)
9: if ∃r ∈ estimated-rounds, s.t curr-round > r then
10:   re-evaluate estimated-util for curr-round
11:   estimated-util = max estimated util for o ∈ options-trying-for
12: if estimated-util < ε_a and est-util(top(sorted)) < ε_a then
13:   curr-round+ = 1
14:   execute the behavior to agree as quickly as possible given in behavior rule
15: else if estimated-util > est-util(top(sorted)) then
16:   sorted = sort-by-highest-est-round(eligible-options, v, curr-round)
17:   options-offered.add(top(sorted))
18:   curr-round+ = 1
19:   return top(sorted)
20: else
21:   options-trying-for.add(top(sorted))
22:   estimated-util = est-util(top(sorted))
23:   estimated-rounds.add(est-round(top(sorted)))
24:   options-offered.add(top(sorted))
25:   curr-round+ = 1
26:   return top(sorted)
```

Algorithm 7.3.1 shows the details of how the SC-Utility-Estimate-Strategic-Learner selects an option to offer. The algorithm shown is specific to the case where the negotiation process has an $\mathcal{N}_{\text{range}}$ of $(1, 1)$ (since it simplifies the algorithm). The algorithm takes the following parameters:

- v - the variable being negotiated
- *options-trying-for* - these are the options, o_w, that the agent has offered for v because the agent believes (or believed) they will maximize utility
- *estimated-util* - this is the estimated utility of the options in *options-trying-for* (or the utility of the option with the highest estimated utility in that set, if some utility estimates of have proved wrong, due to incorrect round estimates).

- *estimated-rounds* - the estimated rounds of agreement for the options in the set *options-trying-for*
- *options-offered* - the set of all options that the agent has offered for v
- *eligible-options* - the options that are eligible for offering (according to the SC-EIMAPP domain)
- *curr-round* - the current round of the negotiation for v

The first action of Algorithm 7.3.1(Line 1) is to sort the eligible options. Each option is sorted on its estimated utility. To estimate the utility of each of the options, rounds are predicted using the method discussed previously. However, if an agent has already offered an option (i.e., prior to *curr-round*), then *curr-round* is used, instead of predicting a round for that agent. Line 2 of Algorithm 7.3.1 only evaluates to true when the algorithm is executed for the first time on a new variable v. In all other cases, each time the method is called, the algorithm tests if any of the round estimates for options it was trying to get agreement for have proven incorrect (because the predicted round has now passed without agreement) (Line 9). If this is the case, the value of the variable *estimated-util* is updated. The algorithm then tests (Line 12), if it is possible, according to the utility estimates, to achieve better than the minimum utility parameter ϵ_a. If it is not, then the behavior required by the semi-cooperative behavior rule is executed. The algorithm next tests whether it should continue to try and achieve one of the options in *options-trying-for* (Lines 16-19) or if it should instead offer the eligible option of the highest estimated utility (Lines 21-26). If the agent continues to hold out for one of the options in *options-trying-for*, then the algorithm offers the eligible option with the highest estimated round of agreement.

A drawback of the SC-Utility-Estimate-Strategic-Learner is that if the estimates are inaccurate the agent could waste rounds trying to achieve an option. Also, recall that given a $\mathcal{N}_{\text{range}}$ such as $(1, 1)$ and an $\mathcal{N}_{\text{rule}}$ requiring no repeated offers for the same variable, the agents must offer one new option each round of the negotiation. This means that even if the round predications are perfect, the algorithm may run out of options with a higher round prediction than the options in the set *options-trying-for*. To see this, consider the case where the option of highest estimated utility was the option with second highest estimated round count.

To address the possible issues that can arise with the SC-Utility-Estimate-Strategic-Learner, we introduce a related exploitation strategy, which we call the SC-Utility-Estimate-Ordered-Learner (Algorithm 7.3.2). In the SC-Utility-Estimate-Ordered-Learner, the agent offers options in order of estimated utility. If we let $\mathcal{O}'$ be the set of eligible options then the agent's offer for variable v in round r, $o_{r,v}$ is,

$$o_{r,v} = \underset{o \in \mathcal{O}'}{\text{argmax}}\ \text{util-est}(o, v, r),$$

provided the utility estimate exceeds the agent's ϵ parameter.

Algorithm 7.3.2 SC-Utility-Estimate-Ordered-Learner: Select-Option(v,options-offered, eligible-options, curr-round)

```
1: sorted = sort-by-est-util(eligible-options, v, curr-round)
2: if max(est-util(options-offered)) < ε_a and est-util(top(sorted)) < ε_a then
3:    curr-round+ = 1
4:    execute the behavior to agree as quickly as possible given in behavior rule
5: else
6:    options-offered.add(top(sorted))
7:    curr-round+ = 1
8:    return top(sorted)
```

Algorithm 7.3.3 explore-or-exploit(v)

```
1: count = 0
2: for a ∈ other decision-makers(v) do
3:    if randNum < get-probability-explore(a) then
4:       count + = 1
5: if count == 0 then
6:    exploit using SC-Utility-Estimate-Ordered-Learner or SC-Utility-Estimate-Strategic-Learner
      to negotiate v
7: if count ≥ (|other decision-makers|)/2 then
8:    exploit using SC-Utility-Estimate-Ordered-Learner or SC-Utility-Estimate-Strategic-Learner
      to negotiate v
9: explore using 1-Preference-Ordered Strategy to negotiate v
```

The algorithm for deciding whether to use one of the learning strategies, or the 1-Preference-Ordered Strategy (see Chapter 3), to negotiate a particular variable, v, is summarized in Algorithm 7.3.3. The agent decides whether to use a learning strategy (exploit) or whether to use the 1-Preference-Ordered strategy (explore) based on how much information it has collected about each of the other agents involved in the variable. The algorithm maintains a probability of exploring for each other agent it negotiates with. This probability is decreased for each variable the agent negotiates with the agent. In Lines 3 and 4 the algorithm flips a biased coin based on these probabilities, one for each other agent involved in the variable, and counts the number "votes" for using the learned models. If more than half the coin flips "vote" for exploiting (line 7) the algorithm uses one of the learning algorithms to negotiate the variable. Otherwise the algorithm explores using the 1-Preference-Order Strategy.

Algorithm 7.3.4 details the algorithm that updates the probability of each agent's model voting for using a learning strategy. This update is executed every time a new variable is assigned. As the algorithm shows, every time the agent assigns a variable with another agent, the probability of exploring when negotiating with that agent, goes down by a constant multiple of the current

probability. In our experiments, we start the probability of exploration for each agent at 1.0. Note that the value of *const-multiplier* is between 0 and 1.

Algorithm 7.3.4 update-probabilities-explore(v-just-assigned, const-multiplier, probabilities-explore)

1: **for** $m \in$ decision-makers of v-just-assigned **do**
2: probabilities-explore$[m]$ = probabilities-explore$[m] *$ const-multiplier

The exploration algorithm that we use - the 1-Preference-Order Strategy is not random as is often the case in exploration/exploitation approaches. We chose to use the 1-Preference-Order-Strategy because our agents do not benefit from learning a complete model of the space. Rather, our agents want to learn about about when the other agents will offer the options they prefer. Using the 1-Preference-Order Strategy helps to focus the exploration on the part of option space that is most interest to the agent. Furthermore the 1-Preference-Order Strategy acts as a reasonable performance baseline. In fact, if one of the learning strategies is used for a variable where the agent has never negotiated with one of the decision makers before, then for every option, the agent will have a maximum round prediction for the option (recall the method for calculating round predictions). In this case, the rounds will have no effect on the relative utility estimates of the options, and the options will be offered in order of preference as in the 1-Preference-Order Strategy.

We experimented with a variety of *const-multipliers* in this thesis and a number of starting values for the probability of exploration. We found that small changes did not have a large impact on the average utility in the long run.

7.4 Experiments and Results

We have conducted a large number of experiments to determine the efficacy of our approaches and to understand the effects of different utility functions and preference profiles.

7.4.1 Experimental Domain

We use a SC-EIMAPP domain with $\mathcal{R} = \emptyset$, i.e., where the agents have no restrictions on the options they can assign to variables. This contrasts to domains like the multiagent meeting scheduling domain where an agent can't assign the same option to more than one variable. We set $\mathcal{N}_{\text{range}}$ to $(1, 1)$ for all agents and $\mathcal{N}^{a_1}_{\text{rule}}$ requires that every option offered for some variable v, must not have been offered for v before, for all agents. In the experiments shown we used the same formula to set the value of ϵ_a for each of the agents. $\epsilon_a = (\gamma^{\frac{\text{no. options}}{2}+1})$*preference range*. Where, we say the

preference range of an agent is N to indicate that for each option each agents' preference is chosen from the set of integers 1 to N uniformly at random. For a 2 agent variable, the number of rounds before agreement will not exceed the exponent. We chose this ϵ_a so the effects of different γ values would be clear.

In the experiments presented here, unless otherwise stated, variables were negotiated between two agents. As such, when we make comparisons to the 1-Preference-Order Strategy, we are comparing to the outcome that is Pareto Optimal in terms of preferences. At times we focus on the two agent case, which allows us to most readily analyze the effect of the exploitation strategies. We will also show the learning is very successful with more agents. Our experiments explore different preference ranges and different γ values. When graphs displayed do not show error bars the standard deviation calculated over the trials was insignificant.

7.4.2 Adaptation in Self-Play

Figure 7.1 demonstrates that both our exploitation approaches effectively adapt over time in self-play. In this experiment, γ is set to 0.3, the number of options is 40, and the preference range is 40. The graph shows the utility of Agent 1 increasing as the number of variables negotiated increases, relative to the non-learning approach (the graph for Agent 2 is similar). Using the 1-Preference-Order strategy (i.e., the non-learning approach) resulted in Agent 1 receiving approximately zero utility. The problem of learning when another agent will offer options is non-trivial in the self-play setting, since the SC-agents are not consistent in when they offer options. This is because SC-agents offer in the order of preference when they explore, and seek to compromise when they exploit. However, after only 10 variables have been negotiated, the learning approaches are already performing significantly better than the non-learning approach. *Adapting quickly online is important in many agreement problems*, e.g., in a domain like meeting scheduling, there may be few opportunities to learn about an agent.

7.4.3 Learning With More Than Two Agents

Our learning approach also significantly out-performs not learning when there are more than two agents. Figure 7.2 shows results from an experiment with 15 Agents. There were 20 options and the preference range was 40. A total of 300 variables were assigned each trial (200 trials). Agent 1 was a decision maker for every variable, and each variable had up to 10 participants. The number of decision makers for a variable was chosen uniformly at random from $[2-10]$. Decision makers, and the initiator for a variable were also randomly selected. The figure shows results for when all agents had a γ value of 0.5. Negotiating variables with many decision makers makes the exploitation process more challenging since an agent must rely on estimates about multiple agents (and if it is not the initiator of the current variable, the agent doesn't see the offers of most these agents for

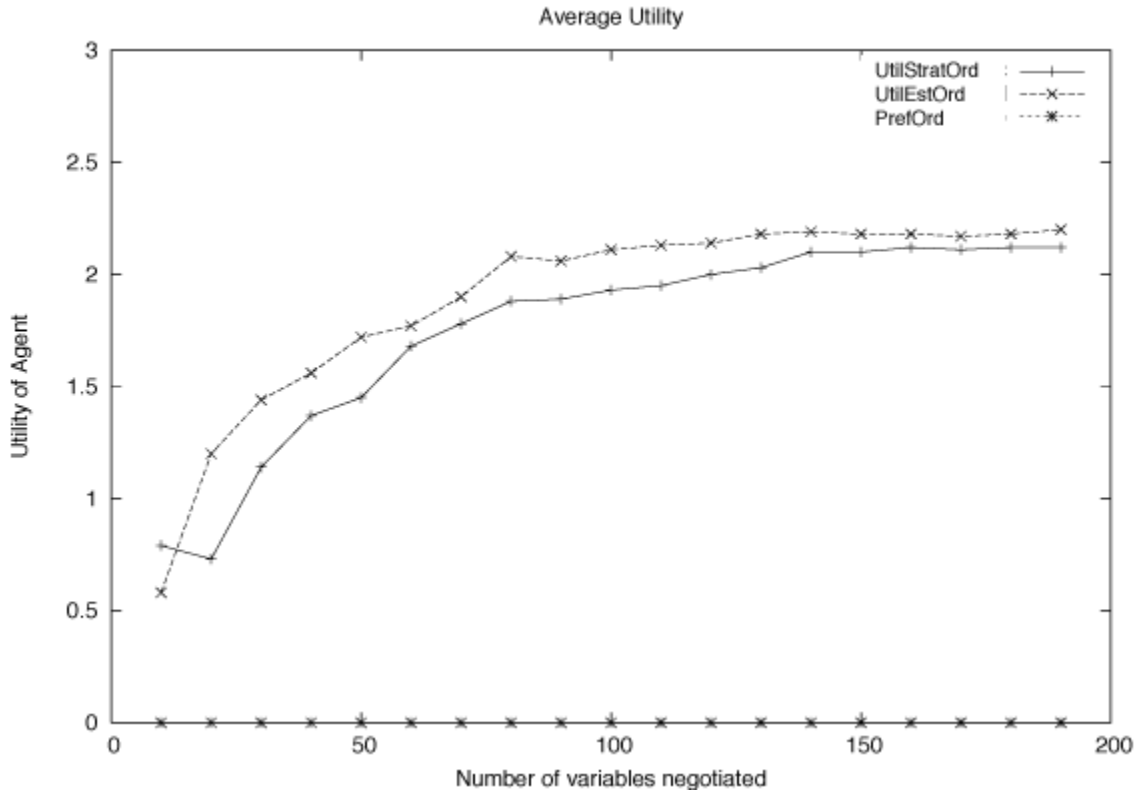


Figure 7.1: The SC-Utility-Estimate-Strategic-Learner (UtilStratOrd) and the SC-Utility-Estimate-Ordered-Learner (UtilEstOrd) increase the average utility they achieve over time in self-play

the current variable). Despite this challenge, Figure 7.2 clearly shows that both our exploitation approaches greatly out perform not learning online.

7.4.4 Effect of γ on Improvement Due to Learning

We examined the effect different values of γ had on how much learning improved performance (as a percentage of the performance of the 1-Preference-Order strategy). Figure 7.3 shows significant percentage improvements for most values of γ. For very low γ, $\gamma = 0.1$ we found that all the approaches achieved a utility of 0 when rounded to two decimal places. All the preferences in this experiment fell in the range 1 to 9, and the number of options was 40. When γ was high for the agent we were evaluating (0.9), the percentage improvement over the 1-Preference-Order strategy was moderate (17 to 27%) this was regardless of whether the other agent had a high or a low γ. As the γ of the agent we were evaluating decreased to medium high (MHi= 0.7) and medium

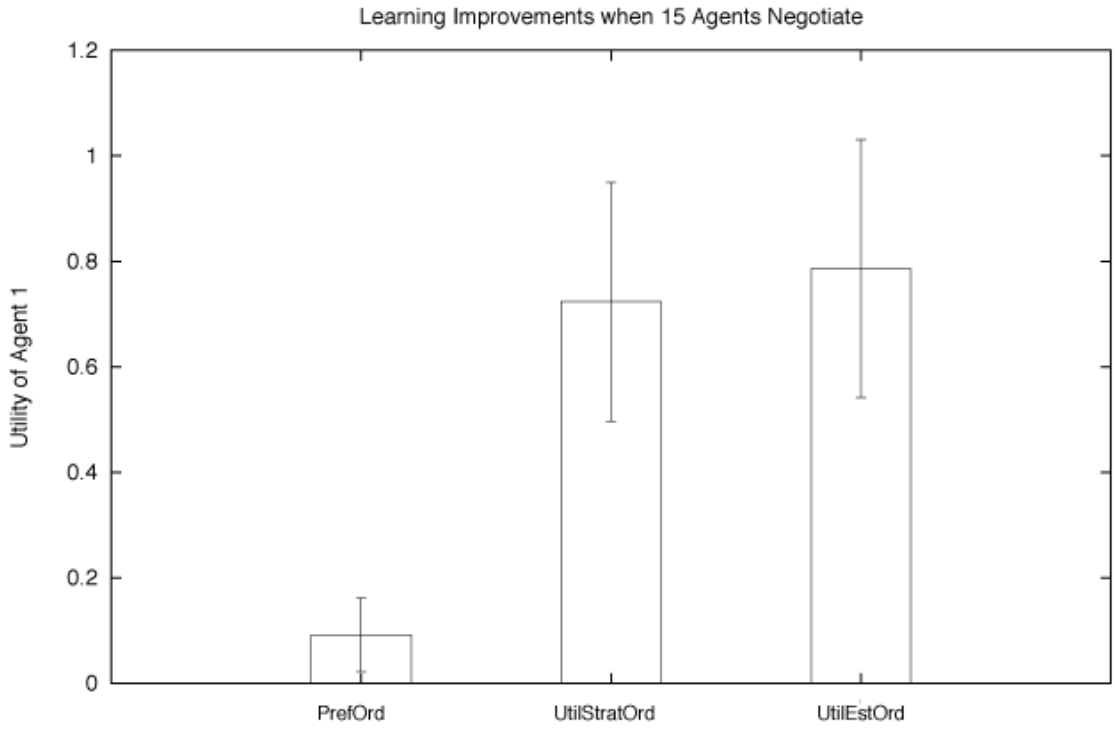


Figure 7.2: 15 agents negotiate variables with up to 10 decision makers in self-play. PrefOrd shows the average utility achieved by Agent 1 with the 1-Preference-Order strategy, UtilStratOrd shows the average utility for the SC-Utility-Estimate-Strategic-Learner and UtilEstOrd shows the average utility for the SC-Utility-Estimate-Ordered-Learner. The error bars indicate standard deviation

(MLow= 0.3) the percentage improvement increased dramatically, first to a little under 200%, and then to over 500%. In general we did not find a signifigant difference between the two different exploiatation approaches. The pattern described here was similar for other preference ranges.

7.4.5 Effect of Different Exploitation Strategies

Figure 7.4 looks at what happens when agents with different exploitation strategies negotiate together. In the experiments shown in this figure the number of options and the preference range was 40, and there were two agents. Each of the sub figures shows the utility of Agent 1, as it learns over time to negotiate better with Agent 2, for a different combination of γs for the two agents. For all the different γ combinations, we see that the lowest utility is achieved when neither agent uses learning. We see that Agent 1 does better for all the γ cases if it exploits (using either approach).

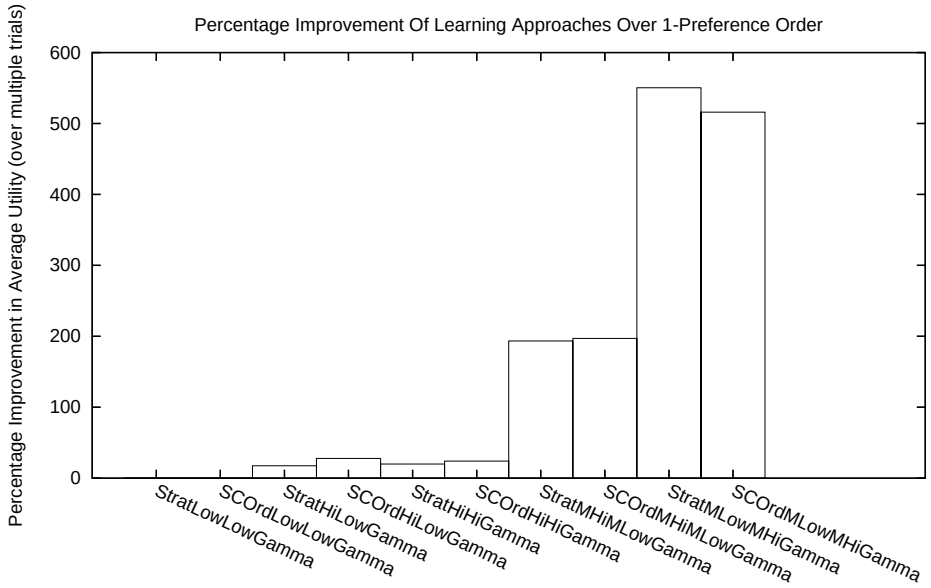


Figure 7.3: The figure shows the percentage improvement of the SC-Utility-Estimate-Strategic-Learner(Strat) and SC-Utility-Estimate-Ordered-Learner(Ord) over the 1-Preference-Order strategy. The maker "HiLowGamma" indicates that the agent we are evaluating had a high γ value and the agent it was negotiating with had a low γ value. The values of γ were as follows: Hi=0.9 MHi=0.7, MLow=0.3, Low=0.1. 200 variables were assigned and the std. dev across trials was negligible

Importantly we clearly see, that *Agent 1 has higher utility if Agent 2 also learns and exploits.* In other words, semi-cooperative agents are able to adapt to each other in a way that is mutually beneficial. Finally we see from these figures that how well an agent does is not particularly sensitive to which semi-cooperative exploitation strategy it chooses.

7.4.6 Effects on an Agent with Low γ

Table 7.1, shows the effect agents' with different γ values have on a very cooperative agent. Agent 1's γ value is 0.3. The table shows the utility, averaged over 200 trials, Agent 1 achieves when it negotiates with three different "Other" agents. The first agent's γ is 0.3, the second's 0.7, and the third's 0.9. In general we found that *regardless of the other agent's γ, an SC-agent benefited*

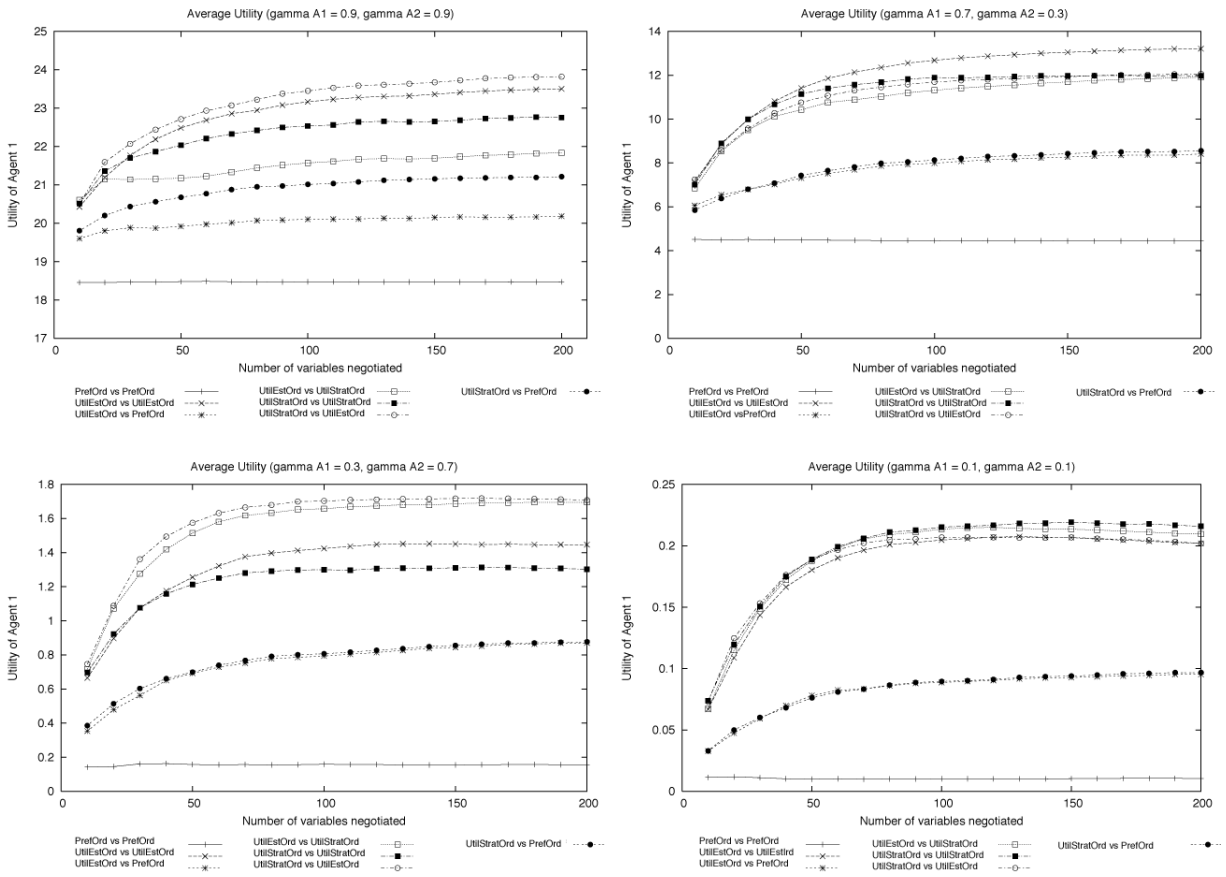


Figure 7.4: Effect of different exploitation strategies for different γ combinations.

from learning. However, we see that a low γ agent, does receive less utility (reduction is less than 50%) when negotiating with agents with increasing γs. This is intuitive, since higher γ values mean an agent is slower to compromise its preferences and agree. However, in many SC-domains, users may be encouraged by the presence of agents that agree quickly to emulate this behavior for social reasons. This points to an interesting area for further study.

γ		Av. Utility		Std. Dev.	
A1	Other	A1	Other	A1	Other
0.3	0.3	0.98116	1.01526	0.50872	0.51495
0.3	0.7	0.78669	10.5464	0.5013	2.3873
0.3	0.9	0.63201	23.4496	0.5295	3.1823

Table 7.1: Low fixed γ vs varying γ (UtilEstOrd)

.

7.5 Summary

In this chapter we showed how agents can learn to predict the round in which other agents will offer options from negotiation histories. We introduced two online learning algorithms based on this approach aimed at increasing the agent's semi-cooperative utility. We showed experimentally that our approach significantly outperforms not learning. The learning aspect of our work differs from previous work on learning in agreement problems, which has largely sort to learn explicit models of agents' properties and constraints.

The experiments presented in this chapter, are for an SC-EIMAPP with $\mathcal{R} = \emptyset$. Such SC-EIMAPPs can represent problems like a sports team repeatedly picking team plays. In domains with restrictions on option use, e.g., meeting scheduling, where a time-slot can't be used twice on the same day, learning can be more challenging. We show experimental results in this domain, and another real-world inspired domain in the next chapter.

Chapter 8

Controlled Experiments for SC-Agents

In this chapter, we illustrate the behavior of semi-cooperative agents in two real-world inspired domains. We use controlled experiments and examples to demonstrate properties of the learning algorithms we introduced in the previous chapter. Our experiments demonstrate that agents can effectively use the online learning algorithms to increase their semi-cooperative utility from the negotiation process.

8.1 Experimental Domains

In the previous chapter, we showed the effectiveness of the SC-Utility-Estimate-Ordered-Learner, and the SC-Utility-Estimate-Strategic-Learner on a general syntactic SC-EIMAPP domain. In this chapter we use the Multiagent Meeting Scheduling domain, which we introduced in Chapter 2, and have employed throughout this thesis. We also introduce a new domain, Role Assignment for Multiagent Tasks.

8.1.1 Multiagent Meeting Scheduling

Recall that in the multiagent meeting scheduling problem, unlike in the syntactic domain from the previous chapter, agents have constraints about which options they can assign to variables. In particular, they cannot assign the same option to more than one variable. Formally, we say that for the multiagent meeting scheduling problem, $\forall a \in \mathcal{A}$, a cannot assign variables to options in the set $\mathcal{R}(a, \mathcal{O})$, which equals, all $o \in \mathcal{O}$ s.t $(o, v') \in \mathcal{M}, \forall v'$. For further discussion of this domain,

please refer to the preceding chapters.

8.1.2 Role Assignment for Multiagent Tasks

We introduce a new domain, that contrasts with the multiagent meeting scheduling domain to help demonstrate the domain independence of our algorithms. In the Role Assignment for Multiagent Tasks problem, a group of agents have a task (variable), and each task has a set of participants (decision makers). In order to complete a task and receive reward, the participants must agree on who undertakes which role (an assignment of participants to roles). The roles have different types, and the agents have preferences over the types of the roles. The reward an agent gets from completing a task is equal to its preference for the role it is assigned.

SC-EIMAPP: Role Assignment for Multiagent Tasks Problem

- $\mathcal{A}$ = all the agents that participate in tasks
- $\mathcal{V}$ = tasks that arise over time. Each task is comprised of a set of roles, *roles*, and a set of *participants*. A *role* has a *role-id* and a *type*. The *participants* is a subset of $\mathcal{A}$. We have made the design decision to make $|roles| = |participants|$. This means that the domain expert must decide up-front how to group sub-parts of the task into roles that a single agent will end up carrying out. Our choice is motivated by keeping the problem of checking for an intersection in agent offers relatively straight-forward.
- $\mathcal{D}_v$ = *participants*
- $\mathcal{O}$ = each possible assignment of participant agents to roles.
- $\mathcal{T}$: tasks arise according to the need in the domain

We define the negotiation solution process as follows:

- $\mathcal{R} = \emptyset$
- $\mathcal{N}_p$ = Algorithm 2.2.1, except we use different conditions for determining when there is an intersection in offers (see the discussion that follows on agent offers).
- $\mathcal{N}^a_{\text{range}} = (1, 1), \forall a \in \mathcal{A}$
- $\mathcal{N}^a_{\text{rule}}$ = every option offered by a for v, must not have been offered by a for v before, $\forall a \in \mathcal{A}$.
- $\mathcal{M}$ = begins empty, and over time contains tasks with assignments of roles to agents, e.g., (task1, role1:agent1, role3:agent2, role2:agent3)

- initiator: one of the participant agents that plays the same role in the negotiation, as a meeting initiator does in the multiagent meeting scheduling problem.

We notice that the number of options for a given task, grows with the number of agents. As defined, an agent has no preference over the roles other agents take - only its own role, we have designed an approach to offering options, and recording the negotiation history, which ignores the roles of other agents.

Agent Offers and Negotiation History

Instead of an agent offering a fully specified option, i.e., an option that specifies its own role, and the role of every other agent, we let each agent simply offer its own role. Offering a role indicates willingness to perform the role. Instead of checking if there is an intersection in the offers of all the agents, as is done in Algorithm 2.2.1, we instead need to check if there is assignment of participants to the roles they have offered that covers all the roles.

We model this assignment problem as a bi-partite graph. On one side of the graph we have the participant vertices, and on the other the roles. We draw an edge between a participant and a role if the participant has offered that role in the negotiation process. The condition for agreement, is that graph admits a *prefect matching* (i.e., a set of edges, such that each vertex appears as the start or end point of exactly one edge in the set), or equivalently that the graph admits a maximum matching of size *roles*. It is possible that the initiator of the task will find more than one perfect matching, in which case it randomly selects between the matchings.

Despite having agents offer options that only specify their own role, we still apply the $\mathcal{N}^a_{\text{rule}}$. This means that each agent must offer a new role every round. Since the number of roles is the same as the number of participants, *the number of negotiation rounds is bounded by the number of participants.* To see this, note that by the time the current round is equal to the number of participants, every agent will have offered every role.

An agent records the negotiation history, in order to predict when other agents will offer roles for tasks in the future. Recall that a role is defined by a role-id, and a type. It is the role *type* that agents have preferences about. The role-id is required to distinguish different roles, since a task can have multiple roles with the same type. An agent represents the history of its negotiation with another agent as: $\mathcal{H} = \bigcup_v \mathcal{H}_v$, where $\mathcal{H}_v$ is a set containing a tuple for each offer made by each $a \in \mathcal{D}_v$, where the tuple contains: (i) the agent name, (ii) the type of the role offered, and (iii) the round of the offer.

SC-Utility

An agent's Semi-Cooperative Utility in the Role Assignment for Multiagent tasks domain is calculated in the same way as for other domains. The preference function over the options is based on the *type of role* the agent is assigned in the outcome being evaluated.

Potential Extensions to the Domain

An interesting potential extension to the Role Assignment domain, is the addition of limited resources. For instance, each role could involve the use of some units of various resources e.g., specified as follows:

$$[(\text{resource}_1, \text{units}_1), ...(\text{resource}_k, \text{units}_k)]$$

An agent would have to possess enough of the required resources in order to undertake a role. Agents would have different collections of resources that would be renewed periodically, and perhaps a cost function for resource usage. An agent's preference for undertaking a role would then depend both on the role's type as well as the resource cost. We propose to experiment with this extension in future work.

Incremental Nature of Role Assignment Domain

In the absence of resources or some other type of constraint affected by the assignment of variables there is nothing inherent in the structure of the Role Assignment domain that makes the problem incremental. This is in contrast to the Multiagent Meeting Scheduling Problem where the ability of an agent to only assign each option to a single variable makes the incremental arrival of variables significant.

However, in the Role Assignment domain agent preferences can make the incremental arrival of variables significant. In particular, if an agent's preferences for a role type depend on the role types it is already assigned then assignments affect each other. In our experiments we do not focus on this dimension of the task problem. Rather agent preferences depend only on the role type and as such in our experiments the fact that the tasks arrive over time is not significant for these experiments. However, we demonstrate using the multiagent meeting scheduling domain that our approach can handle incremental problems.

8.2 Analysis and Results - Task Domain

In this section, we describe two different scenarios in the task domain. In our first example there are two agents. We use this example to highlight the benefits to semi-cooperative agents of learning. The second example we give has four agents. Hence the negotiation of role assignments, for a single task, can last for up to four rounds. This longer negotiation process allows us to illustrate the differences between our two different approaches to exploiting learned information - making offers in order of estimated semi-cooperative utility and making offers strategically based on semi-cooperative utility.

The experimental results were obtained using the simulation system we have developed, which is described in Appendix A.

8.2.1 Two Agent Example

In this example we use a two agent system where there are two types of roles (type 1 and type 2). For the sake of simplicity, we won't consider tasks that require resources, or equivalently we assume that all tasks require zero resources.

Algorithm behavior when paired with a fixed agent

We start by looking at how each of the three negotiation strategies (1-Preference-Ordered, SC-Utility-Estimate-Ordered-Learner, and SC-Utility-Estimate-Strategic-Learner) operate when paired with an agent using the fixed 1-Preference-Ordered strategy.

We consider a Role Assignment problem, with two agents, Agent 1, and Agent 2, and roles of two types, type 1 and type 2. The agents have the following preferences over the role types:

Agent	type 1	type 2
Agent 1	19	20
Agent 2	5	20

Let Agent 1 have the following model for when Agent 2 offers roles with each of the possible types:

Type	Round
type 1	2.0
type 2	1.0

Suppose a task, t1, arrives:

- **Decision Makers:** Agent 1, Agent 2
- **Initiator:** Agent 1
- **Roles:** (role1, type 1), (role2, type 2)

We analyze the negotiation that results from the arrival of this task for the case where Agent 1 uses the 1-Preference-Ordered Negotiation strategy, the SC-Utility-Estimate-Ordered strategy and SC-Utility-Estimate-Strategic strategy.

1. **1-Preference-Ordered Negotiation Strategy**:
   In round 1, Agent 1 offers role 2, and Agent 2 offers role 2, since they are both offering in order of preference. There is no possible assignment of agents to roles, such that each role is filled, so the negotiation proceeds to round 2, where each agent offers role 1. There are now two possible matchings of agents to roles. In our protocol the initiator has to choose between these matchings randomly, so with probability 0.5 it assigns itself role 2 and Agent 2 role 1. Agent 1 has a utility payoff of $\gamma_{a_1}^2 20$ and Agent 2 has a payoff of $\gamma_{a_2}^2 5$. With probability 0.5 Agent 1 makes the alternative assignment.

2. **SC-Utility-Estimate-Ordered Strategy:**
   The utility calculations for Agent 1 are:

   $$U(\mathbf{role1}) = \gamma^{1.0} 19$$

   $$U(\mathbf{role2}) = \gamma^{2.0} 20$$

   In this strategy, Agent 1 offers roles in order of estimated semi-cooperative utility, So Agent 1 will only select role 1 before role 2 when

   $$\gamma_{a_1}^{1.0} 19 > \gamma^{2.0} 20,$$

   i.e when $\gamma_{a_1} < 0.95$. Suppose this is the case, and Agent 1 offers role 1 in round 1. The negotiation will end in one round (assuming Agent 1's model of Agent 2 is correct) and Agent 1 will have a payoff of $\gamma_{a_1}^{1.0} 19$, where $\gamma_{a_1} < 0.95$, and Agent 2 a payoff of $\gamma_{a_2}^{1.0} 5$. Thus the negotiation is shorter, Agent 1 has a higher utility, and Agent 2 a higher expected utility (i.e., there is increased social welfare) than if Agent 1 used the Preference Order Strategy.

   If $\gamma_{a_1} > 0.95$ then Agent 1 will make the same offer as for the Preference Order Negotiation Strategy and this will maximize Agent1's utility in this instance.

3. **SC-Utility-Estimate-Strategic Strategy:**
   Since there are at most two rounds in this example the strategic strategy has the same behavior as the SC-Utility-Estimate-Ordered strategy.

Algorithm behavior when both agents are learning (in self-play)

Both learning strategies have an explore and exploit mode. When in explore mode they operate according to the 1-Preference-Ordered Negotiation strategy (while recording data about the other agents). When exploiting, the learning strategies utilize the learned information according to their specifications - order of estimated utility or strategic. The probability of an agent being in exploit mode with another agent i starts at $p = 1.0$ and is decreased by multiplying p by d^k where d is a constant between 0 and 1, and k is the number of tasks the agent has negotiated with agent i.

To study what happens when both agents are learning, suppose the agents have collected the following data (produced by the agents interacting in explore mode):

Agent 1's data about Agent 2:

Type	Rounds
type 1	2, 2, 2
type 2	1, 1, 1

Agent 1's model of Agent 2:

Type	Round
type 1	2.0
type 2	1.0

Agent 2's data about Agent 1:

Type	Rounds
type 1	2, 2, 2
type 2	1, 1, 1

Agent 2's model of Agent 1:

Type	Round
type 1	2.0
type 2	1.0

To analyze the online learning process, we let all new tasks arriving have the following form:

- **Decision Makers:** Agent 1, Agent 2
- **Initiator:** Agent 1
- **Roles:** (role1, type 1), (role2, type 2)

Each task will have a role 1 of type 1 and a role 2 of type 2 for ease of discussion.

To analyze the behavior of the two learning exploitation strategies we compute the agents' utilities.

For Agent 1 we have: $U(\text{role1}) = \gamma_{a_1}^{1.0}19$ and $U(\text{role1}) = \gamma_{a_1}^{2.0}20$, for Agent 2 we have $U(\text{role1}) = \gamma_{a_2}^{1.0}5$ and $U(\text{role1}) = \gamma_{a_2}^{2.0}20$.

Note - the learning strategies are not specific to the task domain and thus do not model the fact that when there multiple possible matchings one is chosen at random.

Suppose both the agents enter the exploit mode. If Agent 1 has a γ value less than 0.95 it will offer role 1 in round 1. If Agent 2 has a γ greater than 0.25 it will offer role 2 in round 1. Agent 2 will not deviate from this behavior (whether exploring or exploiting). For this situation, and tasks of this form, the Nash Equilibrium is Agent 1 offering role 1 in round 1 and Agent 2 offering role 2 in round 1.

If Agent 2 has a γ value of less than 0.25 then both Agent 1 and Agent 2 will offer role 1 in round 1 and the agreement won't be reached until round 2. The Agents' models will then become:

Agent 1's model of Agent 2:

Type	Round
type 1	1.75
type 2	1.25

Agent 2's model of Agent 1:

Type	Round
type 1	1.75
type 2	1.25

We computed these new models by adding the data from negotiating the last task to the data table and then finding the new average round for each type.

In continuing this example, we fix Agent 1's γ value to 0.8 and Agent 2's to 0.2. Note that with these γ values both agents are not simply best off always offering roles in order of preference.

For the next task, Agent 1 has, $U(\text{role1}) = \gamma_{a_1}^{1.25}19 = 14.38$ and $U(\text{role1}) = \gamma_{a_1}^{1.75}20 = 13.53$. Agent 2 has $U(\text{role1}) = \gamma_{a_2}^{1.25}5 = 0.67$ and $U(\text{role1}) = \gamma_{a_2}^{1.75}20 = 1.20$. So Agent 1 will offer role 1 in round 1 and Agent 2 will offer role 2. Notice that this will decrease Agent 2's round prediction for Agent 1 offering type 1 in round 1, reinforcing it offering type 2 in round 1. Also it will decrease Agent 1's round prediction for Agent 2 offering type 1 in round 1. Thus, Agent 1 role 1, and Agent 2, role 2, will be the Nash Equilibrium for tasks of this form in this situation.[1]

Experimental Results

In the examples given thus far we have looked at specific γ values or ranges of γ values. In this section, we show results from our simulation system for a wide range of utility functions. Figures 8.1,8.2, and 8.3 show results that were averaged over 100 trials (with insignificant standard deviation).

Figure 8.1 shows Agent 1's average utility over assigning roles for 50 tasks with Agent 2 for a number of strategy pairings. For almost all pairings, Agent 1 had the highest average utility when learning versus using the Preference Ordered strategy. The only exception is when Agent 1 has a γ value of 0.9 and Agent 2 a γ value of 0.1 i.e., where Agent 2's γ value was so low that it was induced to offer role 1 despite its low preference for roles of type 1.

Figure 8.2 shows Agent 2's average utility for the same exact experiments as Figure 8.1. Note that in 4 out of the 5 strategy pairings, Agent 2 is learning. This graph clearly demonstrates that while Agent 2 like Agent 1 benefits from learning, *Agent 2 also benefits from Agent 1 learning.* What we see is that when both agents learn and exploit they increase the social welfare.

Figure 8.3 shows the average round count for the different utility functions from the same experiments. Clearly the shortest negotiations are consistently achieved when both agents learn. In this case, the agents end up assigning roles in only one round.

Figure 8.4 shows the rate at which the strategies improve their performance for Agent 1 as the number of tasks negotiated increases. In this particular example, both Agents had a utility function with a γ value of 0.6, and the plot is the result of 100 trials averaged.

[1]We conjecture that under certain conditions, e.g., static preferences, SC-agents in the task domain will converge to variants of NE using learning algorithms. However, we don't claim this is true for SC-EIMAPPs in general. Moreover, we believe it is important that agents in agreement problems can adapt in dynamic situations.

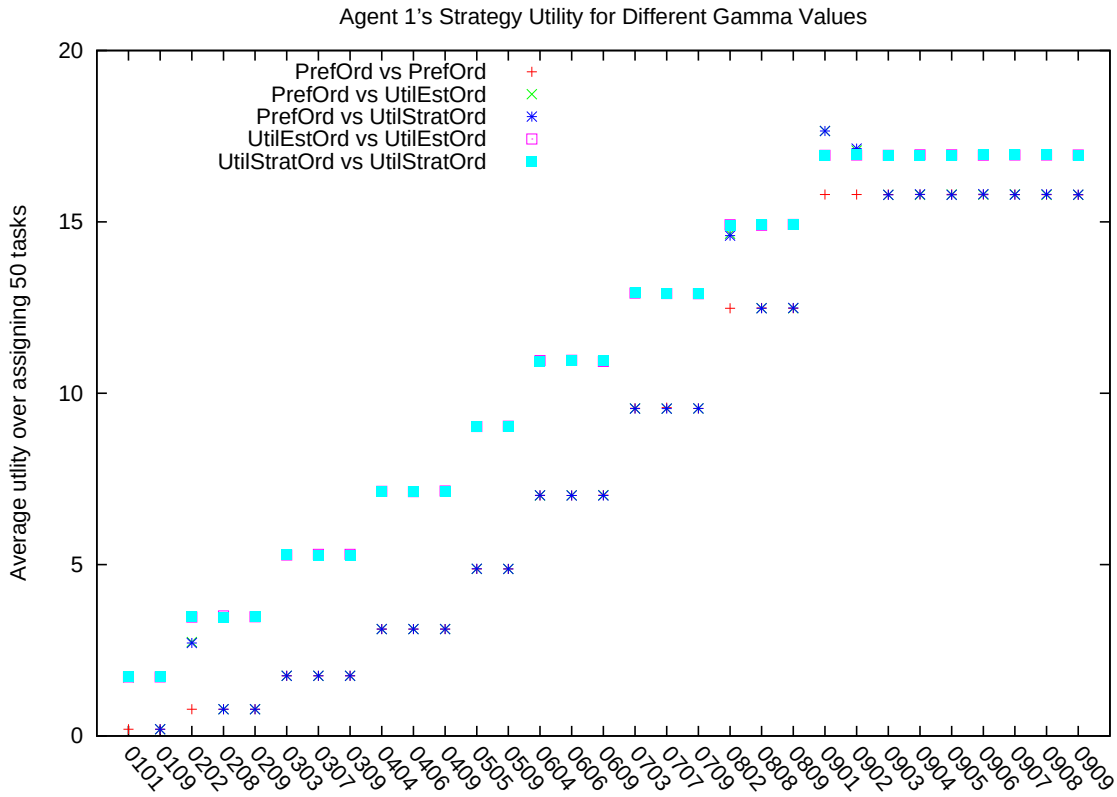


Figure 8.1: Agent 1: Strategy performance for different utility functions. A x-label of 0X0Y means that Agent 1 has a γ value of 0.X and Agent 2 has a γ value of 0.Y. PrefOrd, refers to the 1-Preference-Order strategy; UtilStratOrd, refers to the SC-Utility-Estimate-Strategic-Learner strategy; and UtilEstOrd, refers to the SC-Utility-Estimate-Ordered-Learner strategy. Note that for each of the experimental results plotted on this graph the agents have the same preferences and assign the same tasks, but each different x-axis label can represent an otherwise unrelated experiment to its neighbor since both agents' utility functions may differ between the experiments. We use these same short forms for strategies throughout this chapter in the graphs.

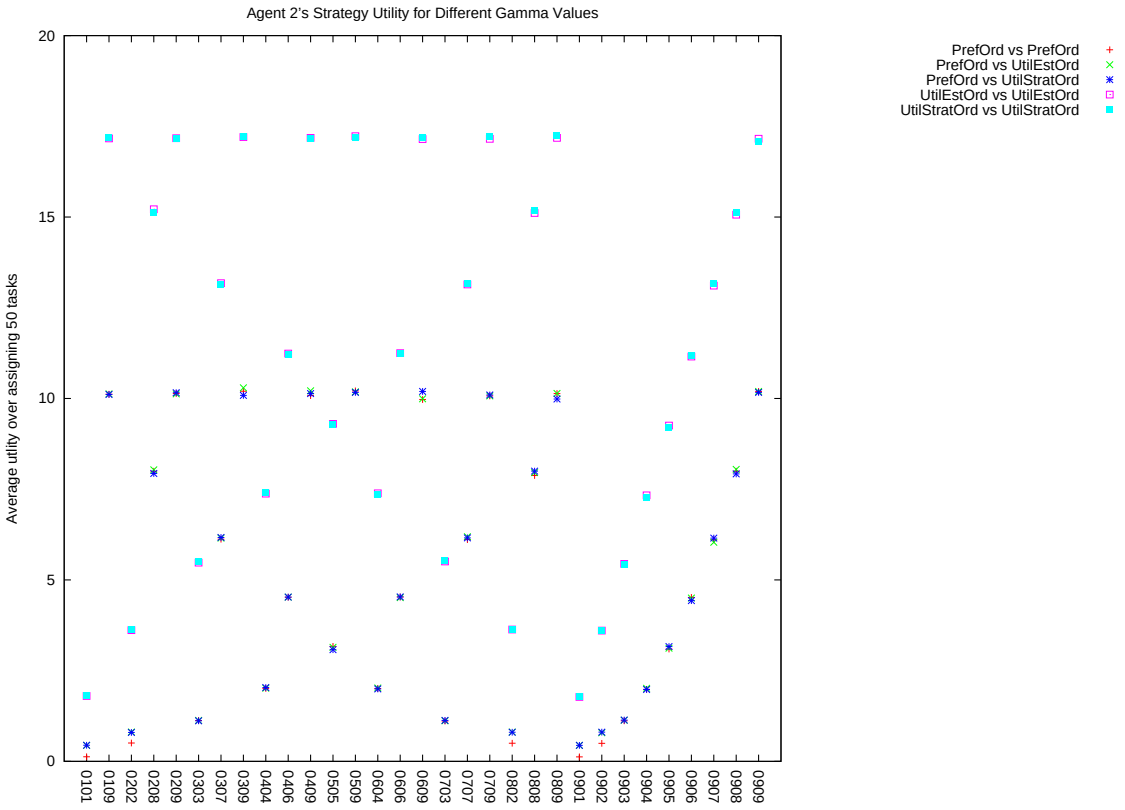


Figure 8.2: Agent 2: Strategy performance for different utility functions. A x-label of 0X0Y means that Agent 1 has a γ value of 0.X and Agent 2 has a γ value of 0.Y

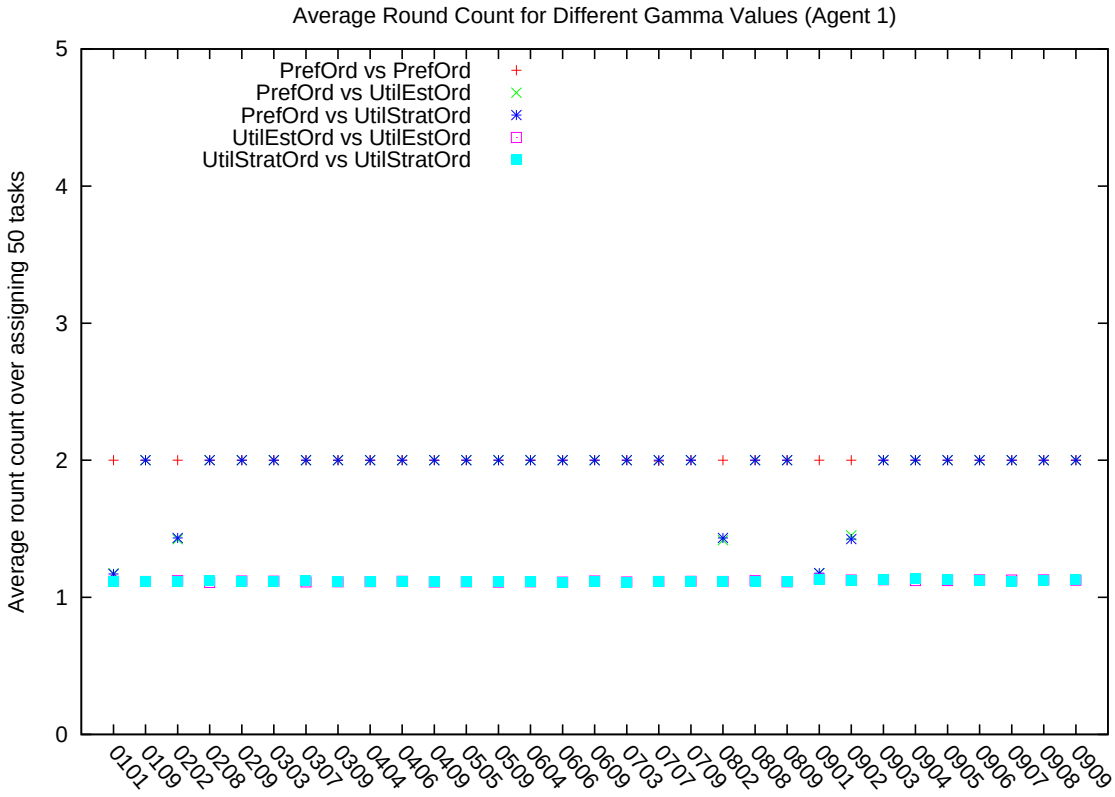


Figure 8.3: Average round count for different utility functions from Agent 1's perspective. A x-label of 0X0Y means that Agent 1 has a γ value of 0.X and Agent 2 has a γ value of 0.Y

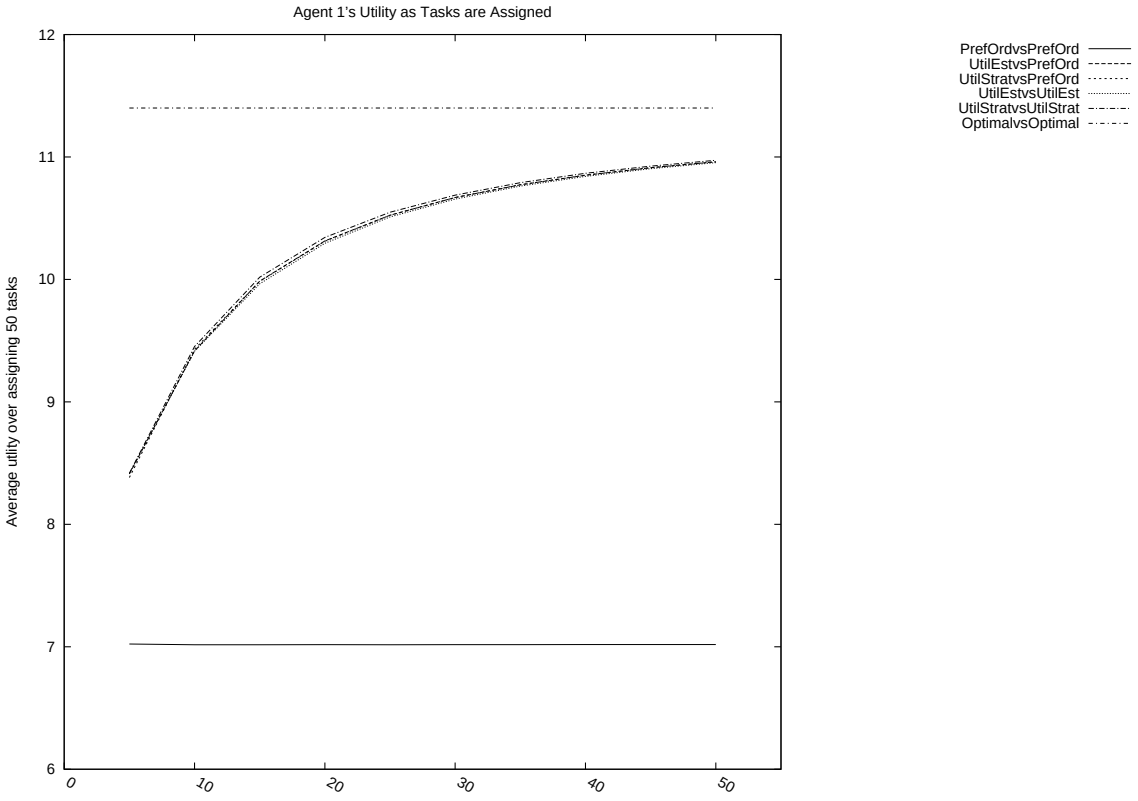


Figure 8.4: Utility for Agent 1 increasing as more tasks are assigned

8.2.2 Four Agents Example

In this example there are four agents, Agent1, Agent2, Agent3 and Agent4. We look at the situation from the perspective of Agent1. Let Agent1 have the following model of the other agents:

Type	Agent1	Agent2	Agent3
type 1	2	4	4
type 2	4	2	2
type 3	1	1	1
type 4	3	3	3

Agent1's preferences are:

Type	Agent1's preference
type 1	20
type 2	5
type 3	5
type 4	4

The task the agents need to negotiate is as follows:

- **Decision Makers:** Agent1, Agent2, Agent3, Agent 4
- **Initiator:** Agent 1
- **Roles:** (role 1, type 1), (role 2, type 2), (role 3, type 3), (role 4, type 4)

Notice that if Agent1 can delay agreement till round 3, Agent 1 will have a chance of being assigned role 1, which is of type1.

We calculate the utility estimates for Agent1 offering each of the roles where Agent1 has a γ value of 0.7. To do this, we need to estimate, at the current round, when agreement will be reached if Agent 1 offers, and is assigned, each of the different roles. There are two ways we can make this estimate. The first method, shown in Algorithm 8.2.1, finds the roles that no agent has yet offered and for those roles, calculates the round by which round Agent 1 can expect one of the other agents to have offered each role (according to Agent1's model of the other agents). This calculation method is an admissible heuristic (if the round estimates are correct) because it does not ensure that each role is being undertaken by a different agent, meaning that it can underestimate the rounds to reach agreement, but not overestimate it. The round this method calculates is just a maximum of the

round estimates for each role and hence we allow this round estimate to be a real number when using it in the utility estimate calculation.

An alternative method works by first rounding Agent 1's round predictions for each agent to integers. It then removes Agent 1 and the role being evaluated from consideration and looks for the first round greater than the current round such that a perfect matching can be formed between the remaining agents and the role offers they have made already and that Agent 1's model predicts they will have made by that round. This algorithm is described in Algorithm 8.2.2. This approach avoids problems with potentially significant underestimations, but it can be more costly (in terms of time) for the agents to execute. In most of our experiments we use the heuristic approach.

Algorithm 8.2.1 approximateRoundAdmissable (role$_i$, currentRound, task)

rolesNeeded = task.roles - role$_i$

for each role $\in$ rolesNeeded **do**
  if role previously offered by agent in {task.decisionMakers $-$ thisAgent} then rolesNeeded = {rolesNeeded $-$ role}
maxRound = currentRound

for role in rolesNeeded **do**
  predictedRoleRound = decisionMakers.size

  for agent in task.decisionMakers - thisAgent **do**
    if predictedRound(agent,role) < predictedRole-round then predictedRoleRound = predictedRound(agent,role)
  if predictedRoleRound > maxRound, then maxRound = predictedRoleRound
return maxRound

Algorithm 8.2.2 approximateRoundMatch (role$_i$, currentRound, task)

for agent $\in$ {task.decisionMakers $-$ thisAgent} **do**
  round-off round estimate to nearest integer
round = currentRound + 1
while not done **do**
  find max matching between {task.decisionMakers $-$ thisAgent} and roles, s.t there is an edge between an agent and a role if the predicted round for that role is <= round
  if matching is perfect then done = true
  else round = round + 1
return round

Role being offered by Agent 1	Estimated Agreement Round	Estimated Utility for Agent 1
role 1	3	6.86
role 2	3	1.715
role 3	3	1.715
role 4	2	1.96

We compare the behavior of the two learning strategies. Both the strategies will offer type 1 in the first round. In the second round the SC-Utility-Estimate-Ordered strategy will offer type 4 and the negotiation will end if the learned model is correct. In this case Agent 1 is guaranteed to be assigned role 4 of type 4 since it is the only agent that has offered this option, and its payout will be 1.96. The SC-Utility-Estimate-Strategic strategy will instead offer the role that is least likely to lead to agreement, so it will offer role 2 or 3 in the second round, and the other in the third round. Negotiation will end in the third round. In half of the possible matchings Agent 1 is assigned role 1 of type 1, so Agent 1 gets 6.86 with a chance of 50%. In the other half of the matchings Agent 1 gets 1.715. Notice that half of 6.89 is on its own greater than the payoff from using the Utility Estimate Ordered Strategy. In other words, in expectation the Strategic strategy performs better in this case.

We note that the Strategic strategy actually generates the same output as the Preference Ordered strategy would. The experimental results however show how agents in this system benefit from learning.

Figures 8.5,8.6,8.7, and 8.8 show the utility Agent 1 received under different experimental conditions. The x-axis gives the γ values of the agents in the experiment, where 0a0b0c0d corresponds to a Agent 1 having a γ =0.a, Agent 2 having γ =0.b and so forth. For each x-value, the results of multiple experiments are plotted. We show the agents' utilities for each of the following conditions:

1. all agents use the 1-Preference-Order strategy
2. Agent 1 uses the 1-Preference-Order strategy and all other agents use the SC-Utility-Estimate-Ordered-Learner strategy
3. Agent 1 uses the 1-Preference-Order strategy and all other agents use the SC-Utility-Estimate-Strategic-Learner strategy
4. Agent 1 uses the SC-Utility-Estimate-Ordered-Learner and all the other agents use the 1-Preference-Order Strategy
5. Agent 1 uses the SC-Utility-Estimate-Strategic-Learner strategy and all the other agents use the 1-Preference-Order strategy
6. all agents use the SC-Utility-Estimate-Ordered-Learner strategy
7. all agents use the SC-Utility-Estimate-Strategic-Learner strategy

We notice from looking at the the results for condition 1 (Agent 1 users 1-Preference-Order and all others use SC-Utility-Estimate-Ordered-Learner strategy), that when the other agents adapt to Agent 1 it is not so important for Agent 1 to learn. The other agents do the adapting for Agent 1. Between the self-play learning settings, Agent 1 generally preferred the setting where all the agents use the SC-Utility-Estimate-Ordered-Learner strategy. We however do observe the drawback of this strategy that we noted in our earlier analysis. By referring to the result for '07070707' (i.e., all agents have a γ value of 0.7) in Figure 8.5 and looking at the results for Agent 1 using SC-Utility-Estimate-Ordered-Learner when the other agents use the 1-Preference-Order strategy, we can see that this strategy does poorly in this setting. When the other agents use the 1-Preference-Order strategy, the SC-Utility-Estimate-Strategic-Learner strategy is the best choice in practice (as well as analytically) for this case.

Figure 8.6, shows that when the other agents all have a very high γ value, 0.9, it is not of much benefit to Agent 1 if they learn. In this case it was better under the different conditions for Agent 1 to adapt. Figure 8.7, clearly shows that Agent 1 prefers the condition where all agents use the SC-Utility-Estimate-Ordered-Learner strategy to the situation where they all use the Strategic strategy. The figure also shows the same preference if Agent 1 alone switches its strategy to 1-Preference-Order. Agent 1 benefits from the other agents using the more naive strategy. Overall we can see from the four graphs for Agent 1 that Agent 1's preferred condition is to use the 1-Preference-Order strategy and have all the other agents use the SC-Utility-Estimate-Ordered-Learner strategy. Agent 1's next most preferred condition was in most cases that all agents, including Agent 1 use the SC-Utility-Estimate-Ordered-Learner.

Figures 8.9 to 8.20 show the utilities for the other agents in the exact same experiments as the Agent 1 graphs. A careful examination of these graphs reveals that for Agents 2, 3, and 4 the SC-Utility-Estimate-Ordered-Learner strategy is the best or equal best strategy for almost all the different utility settings. What this clearly tells us is that the agent system as a whole *benefits from learning*. In trying to satisfy their own utility functions, the agents have achieved an outcome that is good in terms of social welfare.

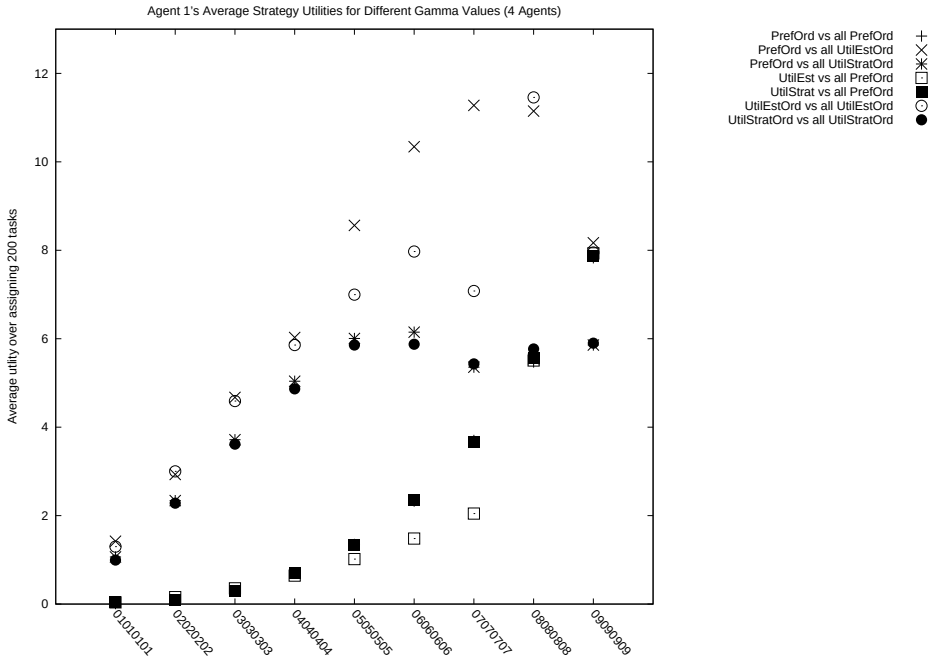


Figure 8.5: Set 1: Agent 1's utilities for different γ values

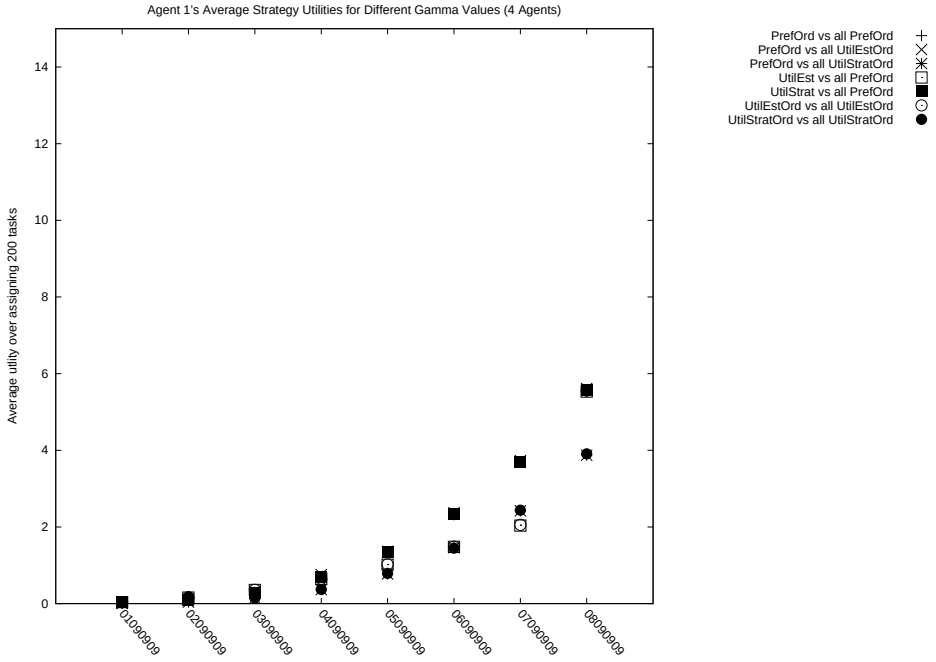


Figure 8.6: Set 2: Agent 1's utilities for different γ values

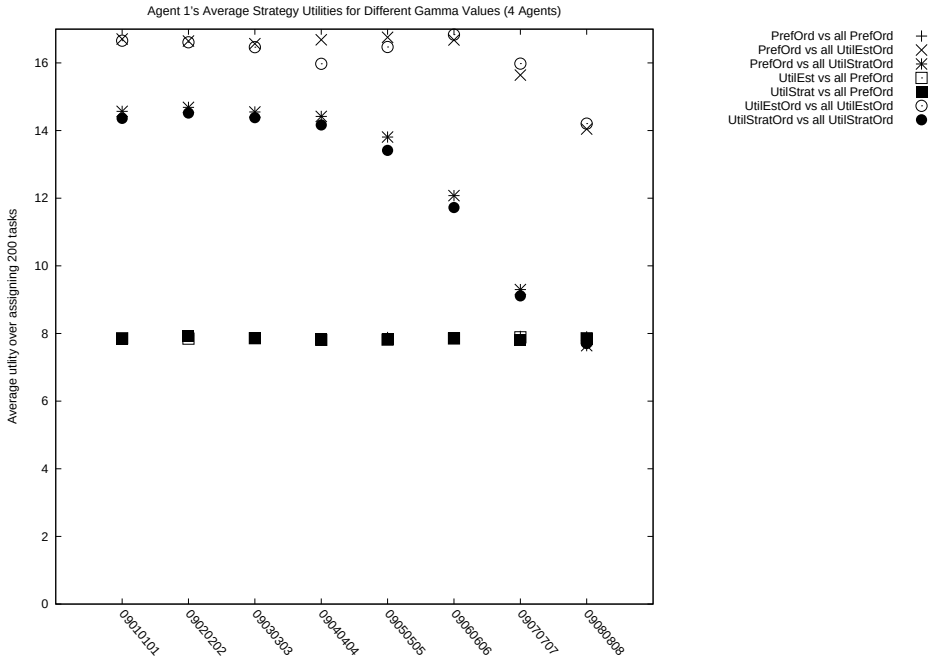


Figure 8.7: Set3: Agent 1's utilities for different γ values

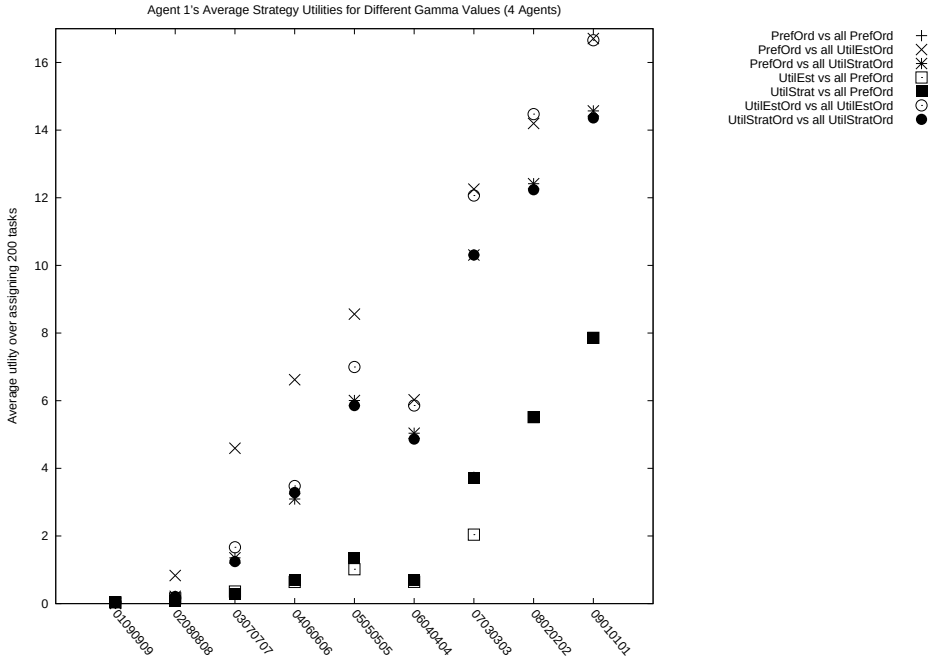


Figure 8.8: Set 4: Agent 1's utilities for different γ values

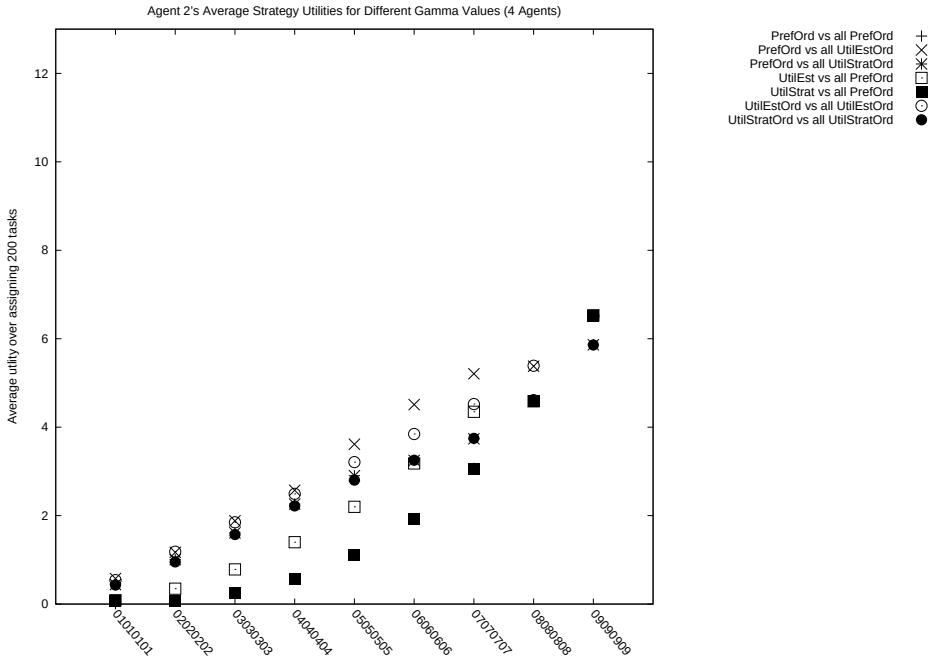


Figure 8.9: Set 1: Agent 2's utilities for different γ values

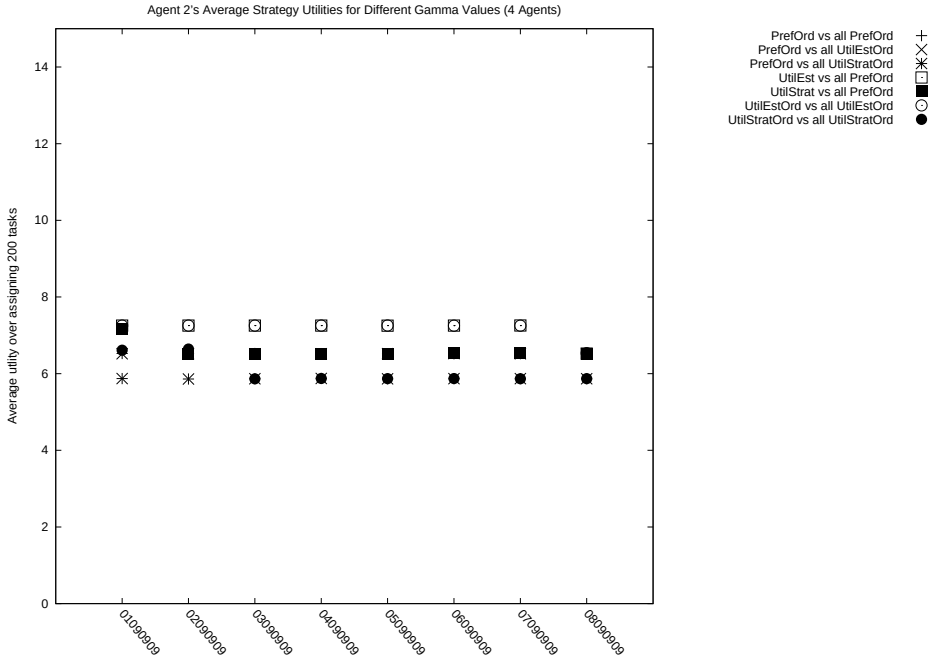


Figure 8.10: Set 2: Agent 2's utilities for different γ values

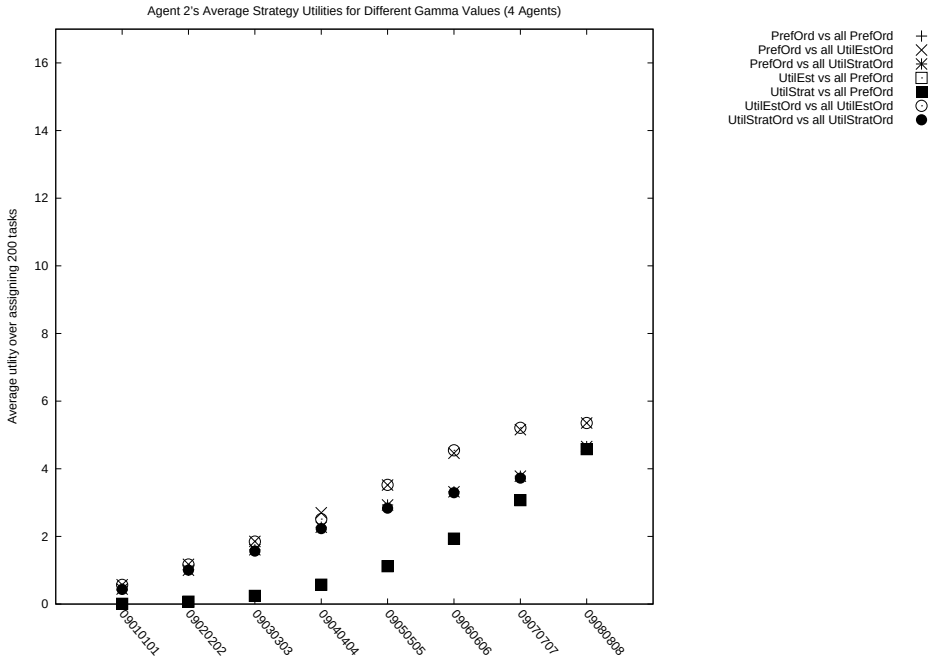


Figure 8.11: Set3: Agent 2's utilities for different γ values

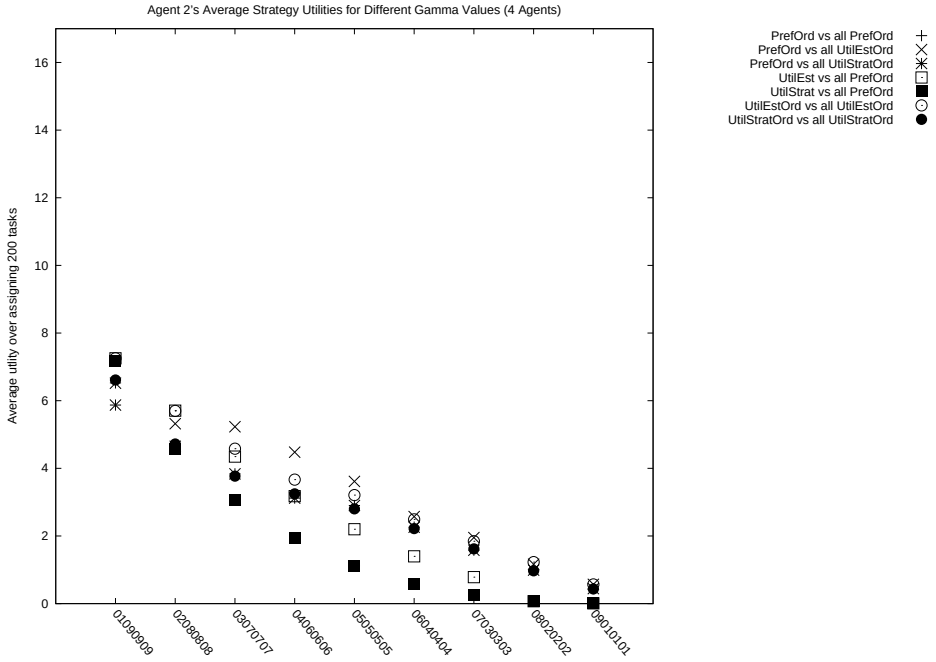


Figure 8.12: Set 4: Agent 2's utilities for different γ values

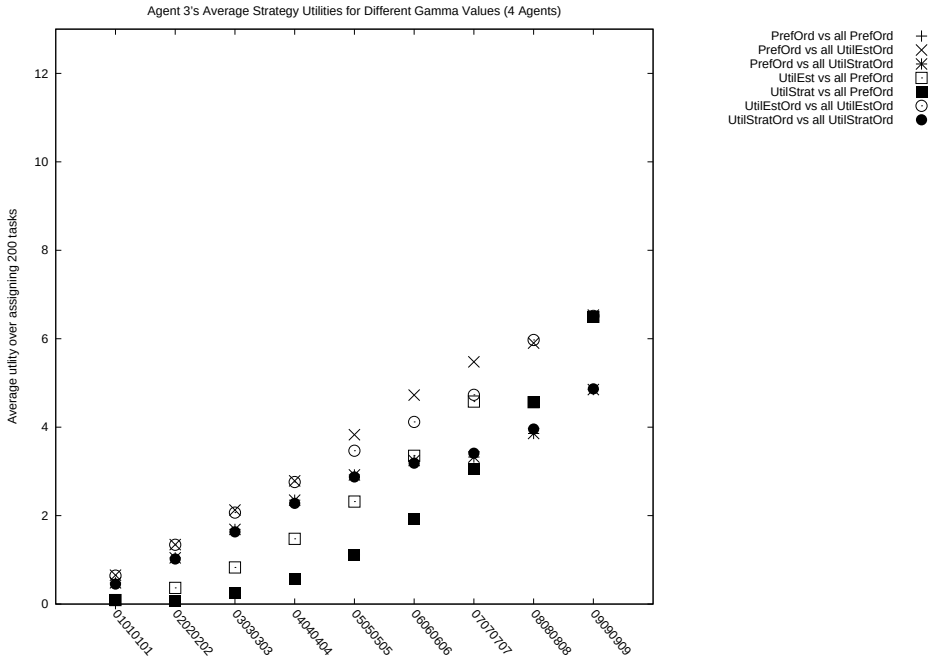


Figure 8.13: Set 1: Agent 3's utilities for different γ values

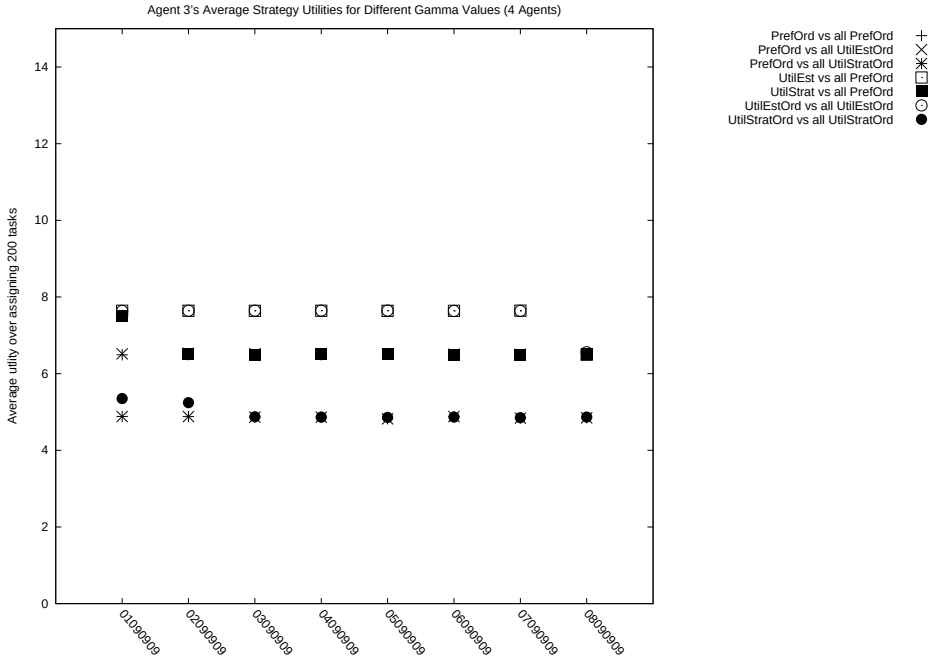


Figure 8.14: Set 2: Agent 3's utilities for different γ values

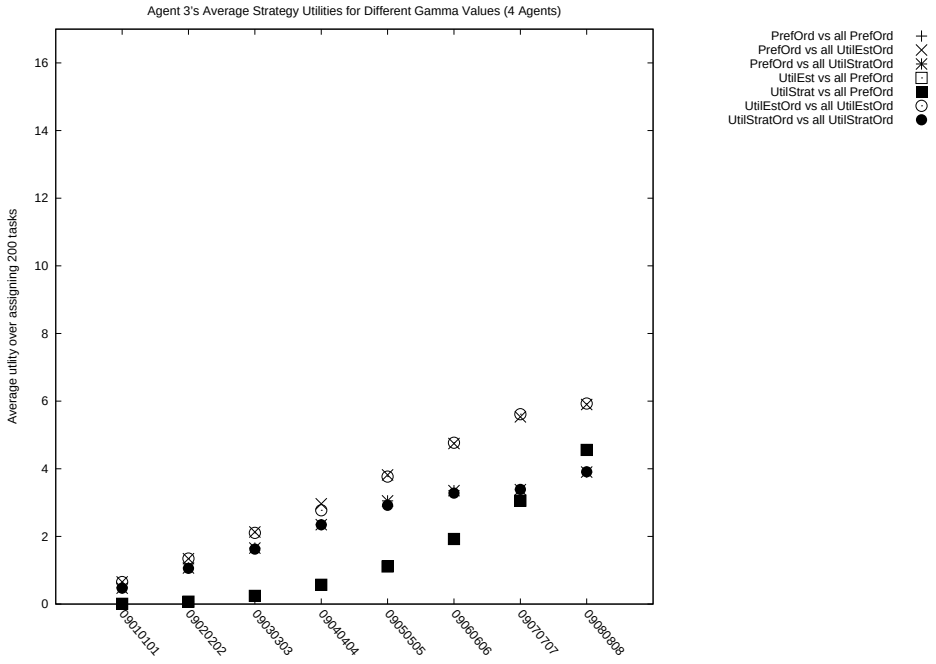


Figure 8.15: Set3: Agent 3's utilities for different γ values

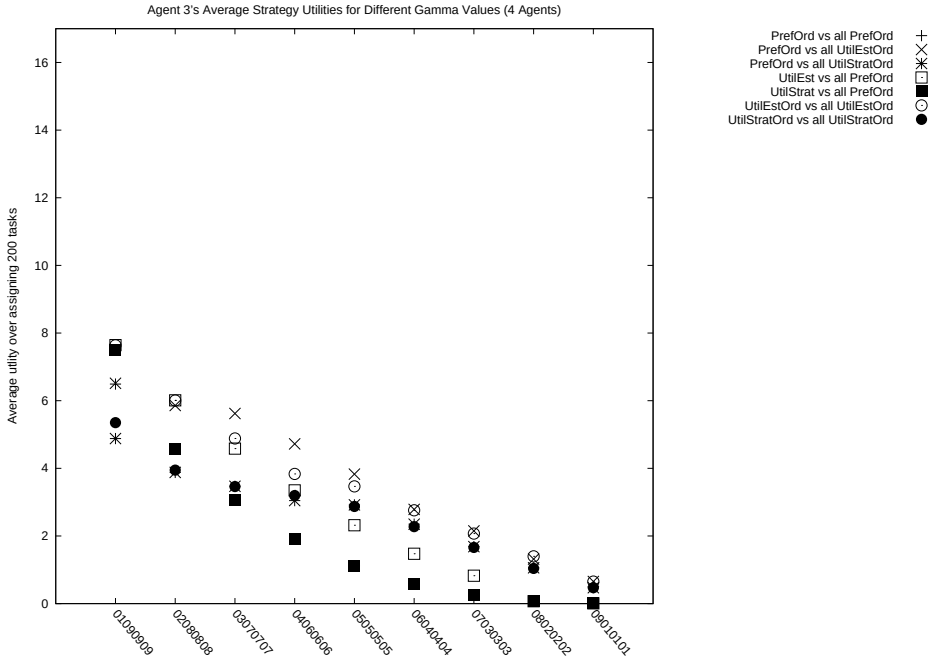


Figure 8.16: Set 4: Agent 3's utilities for different γ values

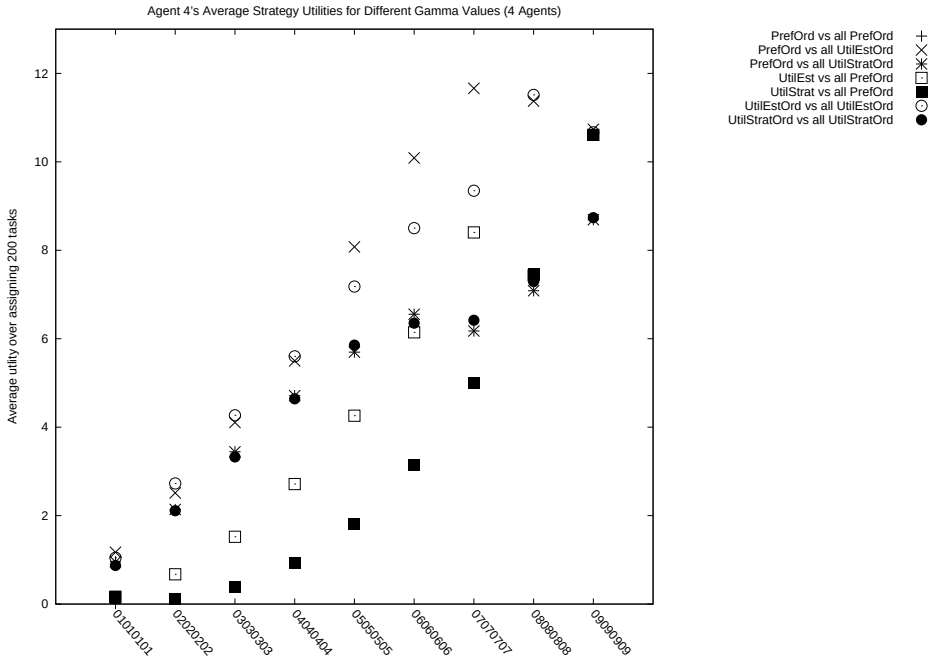


Figure 8.17: Set 1: Agent 4s utilities for different γ values

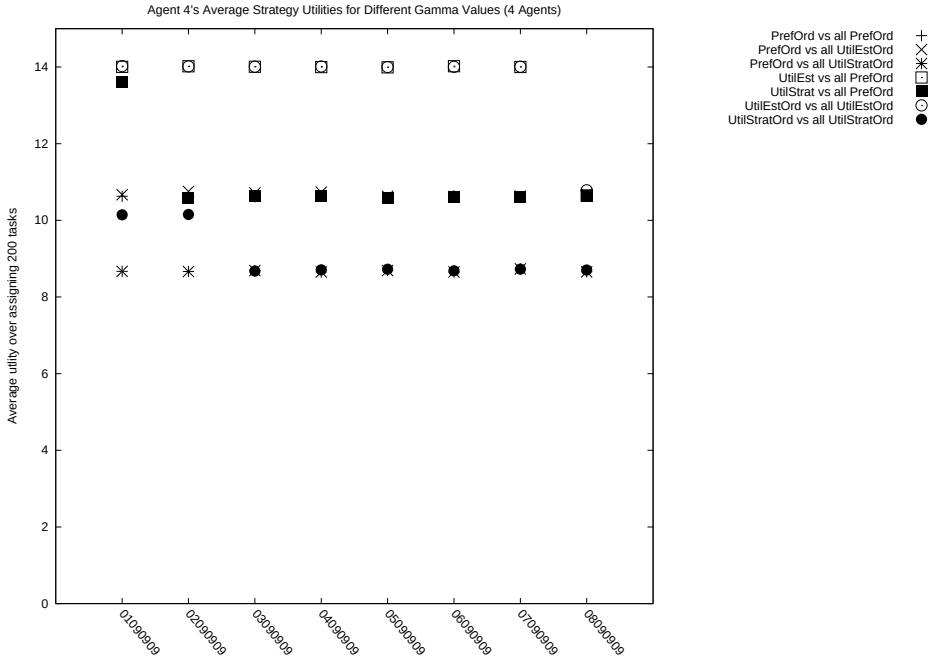


Figure 8.18: Set 2: Agent 4's utilities for different γ values

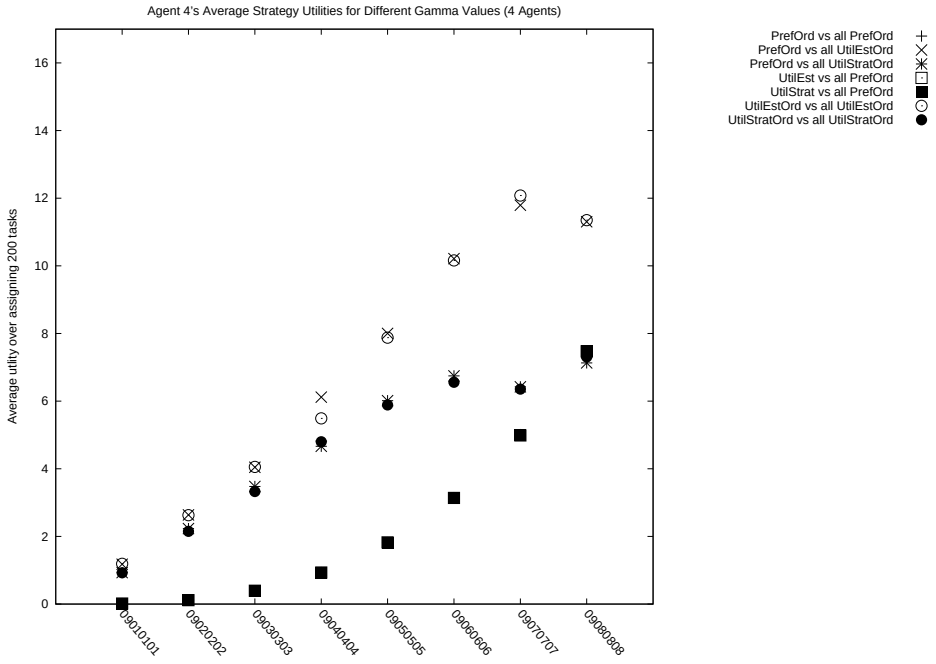


Figure 8.19: Set3: Agent 4's utilities for different γ values

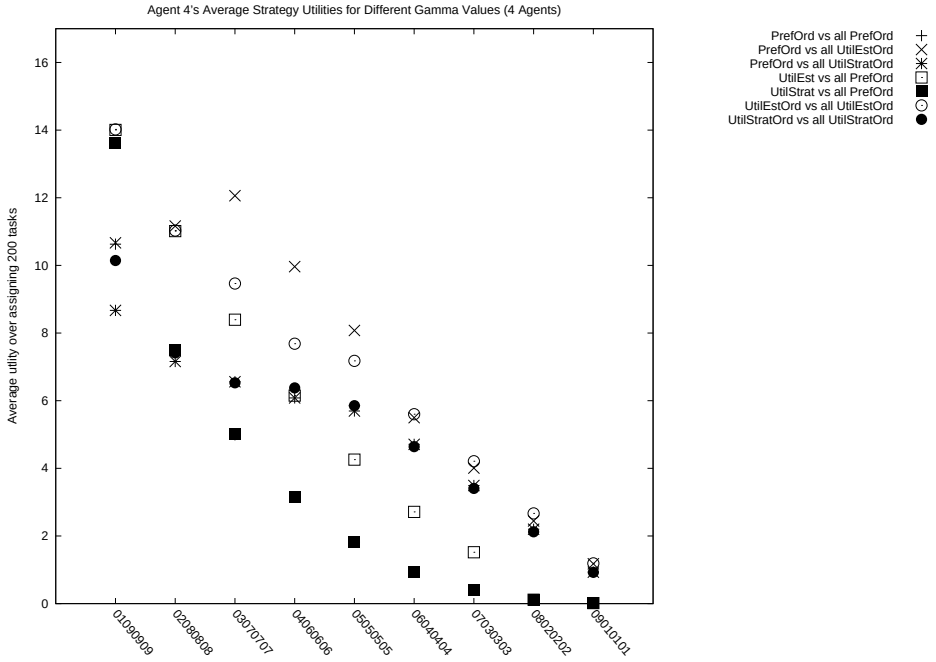


Figure 8.20: Set 4: Agent 4's utilities for different γ values

8.3 Analysis and Results - Meeting Scheduling Domain

In the Multiagent Meeting Scheduling Problem, Semi-Cooperative agents benefit the most if they learn to offer options early in the negotiation that they like, and that the other agents like (and will agree to). In other words, the ideal situation is when there is some overlap in the times preferred by the agents. When agents have opposite preferences for meeting times, the agents can use their round predictions to inform how they should compromise their preferences in order to perform well against their individual utility functions, and thereby perform better than if they did not learn, but the impact of learning is less significant.

We look at two meeting scheduling examples with two agents and different intersections of preferred times to illustrate the effect this has on performance. Suppose we have two agents, Agent 1 and Agent 2, who have no initial meetings. We use an option set (calendar) of size 8. One option for each hour of a normal work day. The agent's calendars are reset after two meetings are scheduled (We chose two meetings to simplify our analysis here - in our experiments we have looked at a wide range of parameters. In general increasing this parameter increases the constraints on when meetings can be scheduled). In the first example, the agents have the following preferences over the options.

Agent 1:

Time	Preference
9am	10
10am	10
11am	10
12Noon	10
1pm	10
2pm	5
3pm	5
4pm	5

Agent 2:

Time	Preference
9am	5
10am	5
11am	5
12Noon	10
1pm	10
2pm	10
3pm	10
4pm	10

The size of the intersection in the agents' preferred times is 2. The first time the two agents negotiate a variable with either learning strategy, they will explore (the initial probability of exploration should generally be set to 1), i.e., they will use the 1-Preference-Order strategy. Without loss of generality, suppose Agent 1 is the initiator. Agent 1, must offer one of the times it has a preference of 10 for, choosing between them at random. There is a $\frac{2}{5}$ chance that Agent 1 will offer an option in the preference intersection to Agent 2, resulting in Agent 2 offering the same option back to Agent 1 and the negotiation concluding in one round. If Agent 1 does not choose an option Agent 2 prefers in the first round, its chances improve to $\frac{2}{4}$ in the second round. Meanwhile, Agent 2 itself has a $\frac{2}{5}$ chance of offering an option in the intersection in the first round, leading to agreement in the second round. Clearly in this example, as agents explore, over the scheduling of multiple meetings, they will discover with high probability the intersection in their preferences. Once the agents have learnt that each will agree to Noon and 1pm in round 1, the agents can then schedule all meetings that arise in just one round, since the reset interval was set to 2. This is more efficient than what they can achieve using the 1-Preference-Order strategy, and they have the same preference for the outcome.

In our second example, we compare what happens when the agents have a large intersection (relative to the reset interval) in preferred times, with the previous example. The agents have the following preferences for the options:

Agent 1:

Time	Preference
9am	5
10am	10
11am	10
12Noon	10
1pm	10
2pm	10
3pm	5
4pm	5

Agent 2:

Time	Preference
9am	5
10am	5
11am	10
12Noon	10
1pm	10
2pm	10
3pm	10
4pm	5

In this case, when the agents use the 1-Preference-Order strategy they will offer an option in preference intersection with probability $\frac{4}{5}$ for the first meeting scheduled in the day, and with probability 1 for second meeting scheduled before the reset. The learning strategies can learn to offer options in the intersection, but the potential increase in performance that this can provide over the 1-Preference-Order strategy is minimal.

The two examples shown, illustrate some of the factors that affect what sort of benefits can be achieved through learning, including the reset interval and the size of intersection of preferences.

8.3.1 Experiments with Four Agents

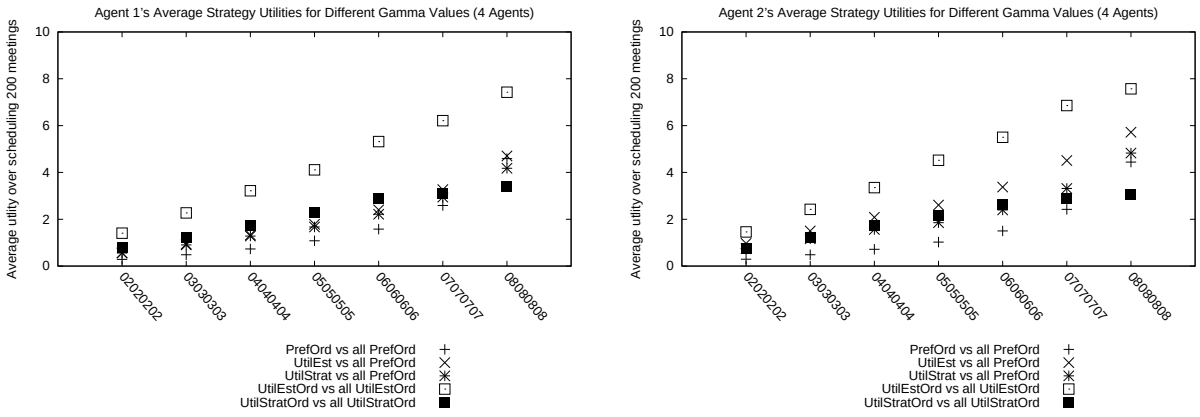
To evaluate and compare the performance of the different learning algorithms in the meeting scheduling domain we include results from two sets of experiments here. In both experiments there are 4 agents in the system, and the agents schedule both two, three, and four person meetings. The agents negotiate over a calendar of size one week, where there is a time slot for each hour in an 8 hour day. The experimental results were obtained using the simulation system we have developed, which is described in Appendix A. We use a reset interval of 5 meetings for both experiments and no initial meetings.

In the first set of experiments, the agents have an intersection in their preferred times of size 1. The agents' preference for this preferred time (and their other preferred times) is 40, and their preference for all other times is 5. The value of ϵ_a was set to 0 for all agents.

The graphs in Figures 8.21, 8.22, and 8.23 show the average utilities (the standard deviation was negligible) the Agents received for the following experimental settings:

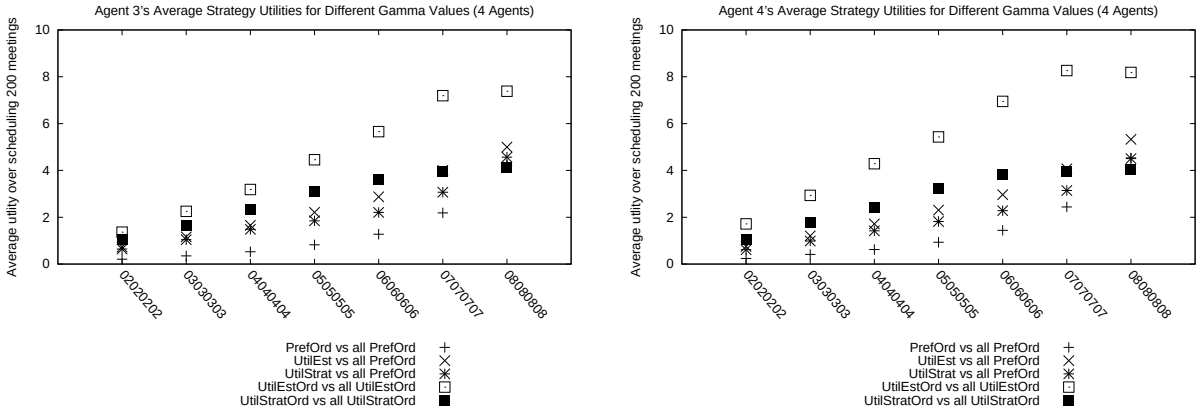
1. all agents use the 1-Preference-Order strategy
2. Agent 1 uses the SC-Utility-Estimate-Ordered-Learner strategy and all the other agents use the 1-Preference-Order strategy
3. Agent 1 uses the SC-Utility-Estimate-Strategic-Learner strategy and all the other agents use the 1-Preference-Order strategy
4. all agents use the SC-Utility-Estimate-Ordered-Learner strategy
5. all agents use the SC-Utility-Estimate-Strategic-Learner strategy

The graphs in Figure 8.21 show that the best condition for all the agents, when negotiating with agents that have the same γ value, is for every agent to use the SC-Utility-Estimate-Ordered-Learner strategy. In almost all cases, the worst condition was when all agents used the 1-Preference-Order-Strategy.



(a) Agent 1

(b) Agent 2



(c) Agent 3

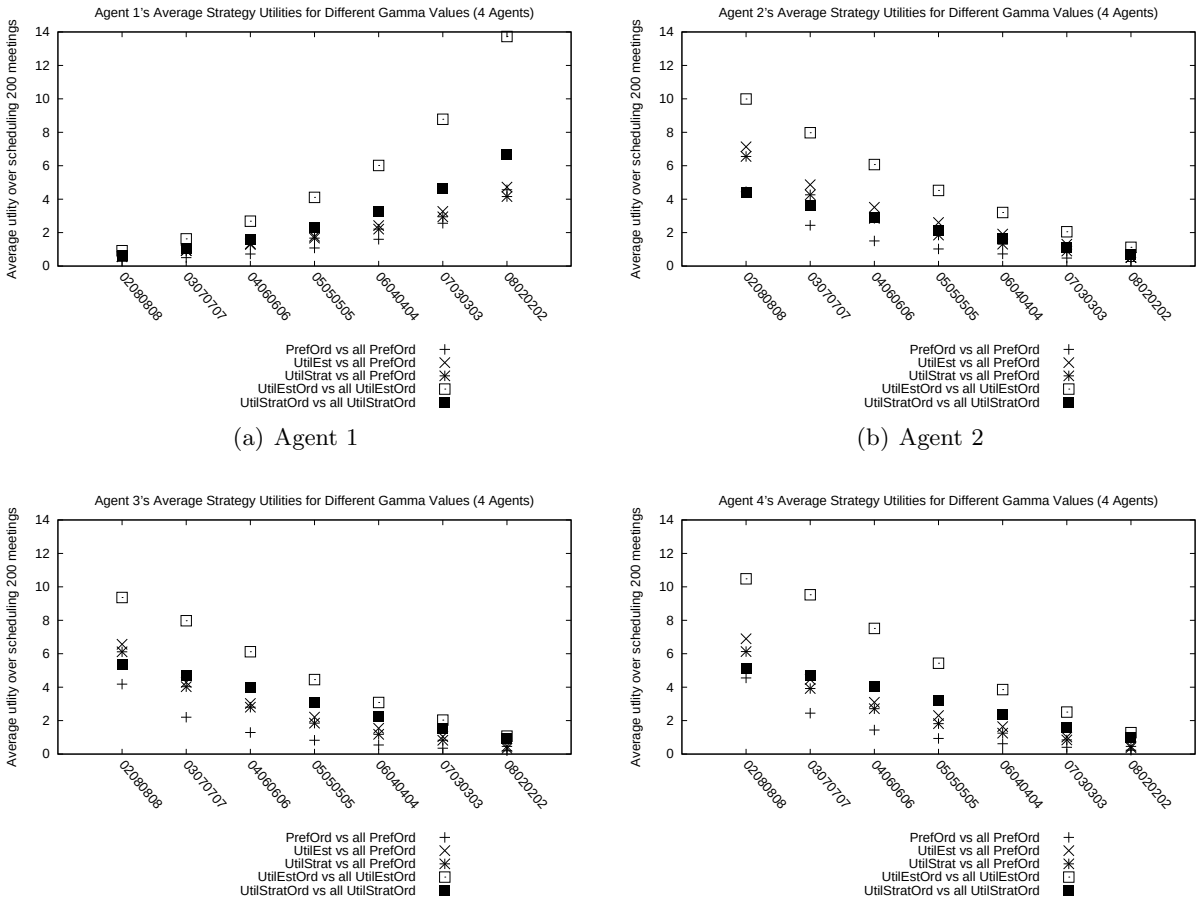
(d) Agent 4

Figure 8.21: Average utilities for agents with same γ, for each of the agents. Calendars reset after 5 meetings scheduled and only one overlap in preferred times.

The graphs in Figure 8.22 show the average utilities of the Agents under the condition, where Agent 1 has a different γ value to the other agents (it is always $1 - \gamma_x$, where γ_x is the γ value of the other agents). Despite this difference in γ values, the preferred condition for all agents is still when everyone uses the SC-Utility-Estimate-Ordered-Learner. The worst condition is consistently, when all agents use the 1-Preference-Ordered strategy. Agent 1's second most preferred condition, is when all agents use the SC-Utility-Estimate-Strategic-Learner, however this condition is slightly less preferred by the other agents.

Finally, we observe the same patterns in Figure 8.23, where two of the agents share the same γ value, and two of the agents have the mirrored $(1 - \gamma)$ value.

In the second set of experiments, we increased the size of the preference overlap to 5 times, leaving



(a) Agent 1

(b) Agent 2

(c) Agent 3

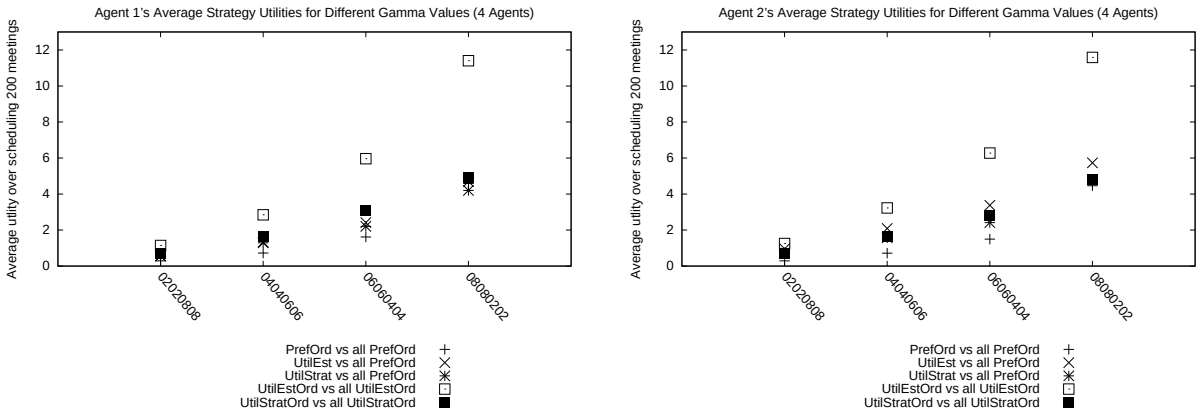
(d) Agent 4

Figure 8.22: Average utilities when Agent 1 has a different γ to the other agents. Calendars reset after 5 meetings scheduled and only one overlap in preferred times.

everything else the same. Figures 8.24, 8.25, and 8.26 demonstrate that this had some interesting effects.

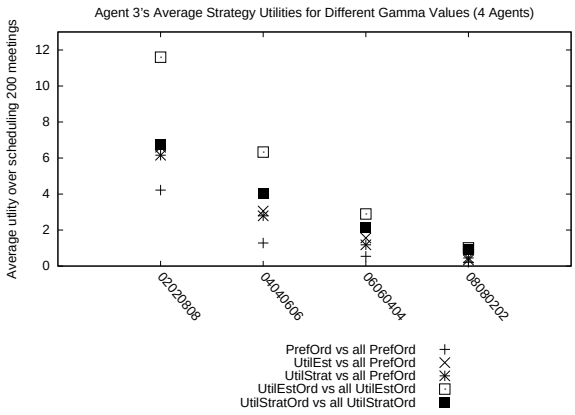
Notice that the scale on the graphs has changed. In the previous experiment, in the first set of graphs, the highest average utility achieved by an agent was just over 8. In the first set of graphs in the second experiment, Figure 8.24, the highest average utility is approximately 20. What this demonstrates is that the *learning algorithms have been able to identify the expanded preference overlap, and have learnt to offer options in that set.*

Another point of interest, is that similarly to our first experiment, for all the different γ value pairings, the best condition *for all the agents* is when each agent uses the SC-Utility-Estimate-Ordered-Learner. The increase in the size of the preference intersection, has had different relative effects on

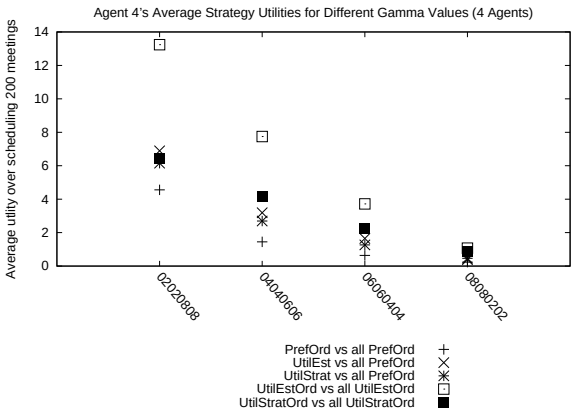


(a) Agent 1

(b) Agent 2



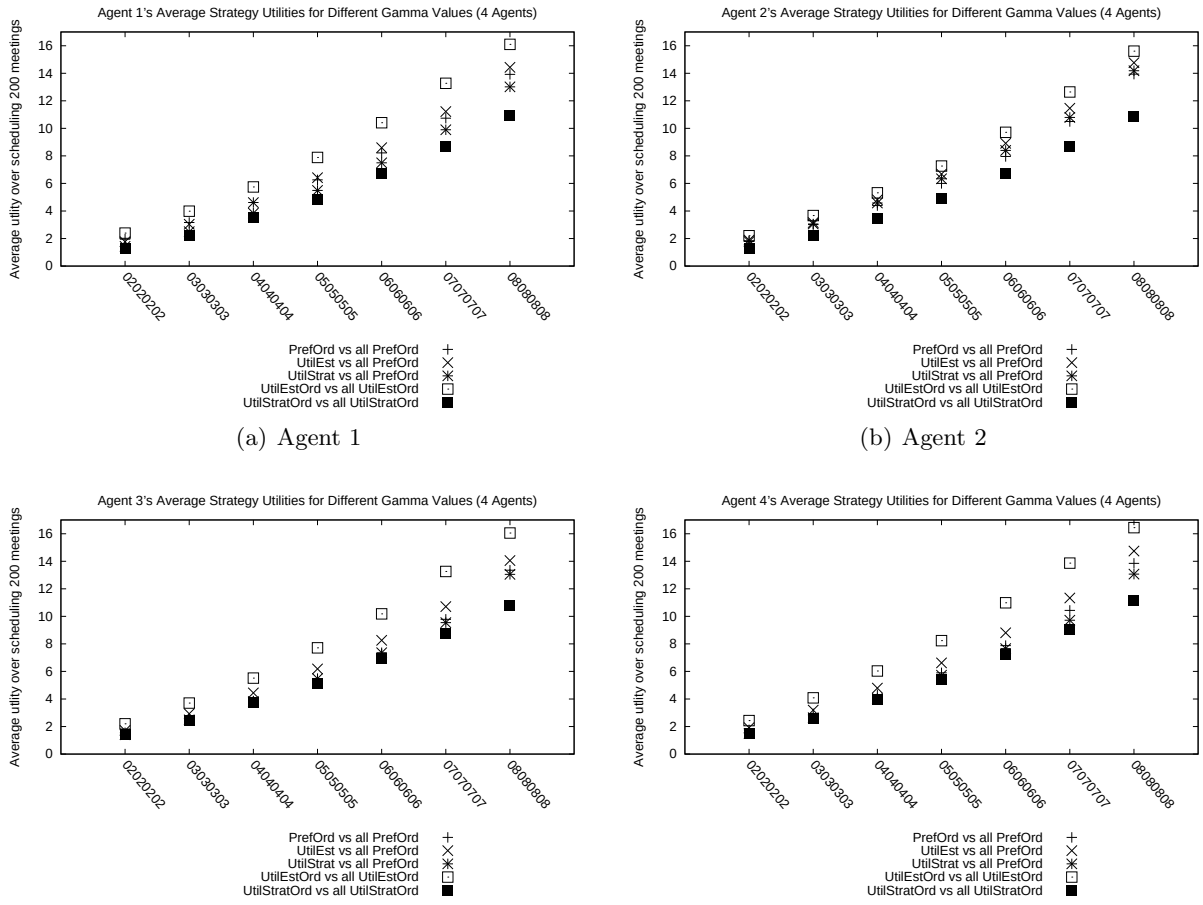
(c) Agent 3



(d) Agent 4

Figure 8.23: Average utilities when Agent 1 has a different γ to the other agents. Calendars reset after 5 meetings scheduled and only one overlap in preferred times.

the agents' performances between other conditions, however. We note that the 1-Preference-Order strategy is no longer always performing below the other strategies. In fact, under some conditions for some of the agents, it out performs the condition where all agents use the SC-Utility-Estimate-Strategic-Learner. The 1-Preference-Order strategy has benefited from the expansion in the size of the preference intersection, because it now has a higher chance of choosing an option of mutual preference. Conversely, the SC-Utility-Estimate-Strategic-Learner has suffered. The larger intersection means there are more times that the Strategic-Learner needs good round estimates for in order to perform comparatively with the SC-Utility-Estimate-Ordered-Learner. Unfortunately some poor round estimates lead the Strategic-Learner to incorrectly hold-out for times, dragging down its performance. Its performance is worst in self-play where the holding-out for times exacerbates the problem of incorrect round estimates (by providing misleading training data).



(a) Agent 1

(b) Agent 2

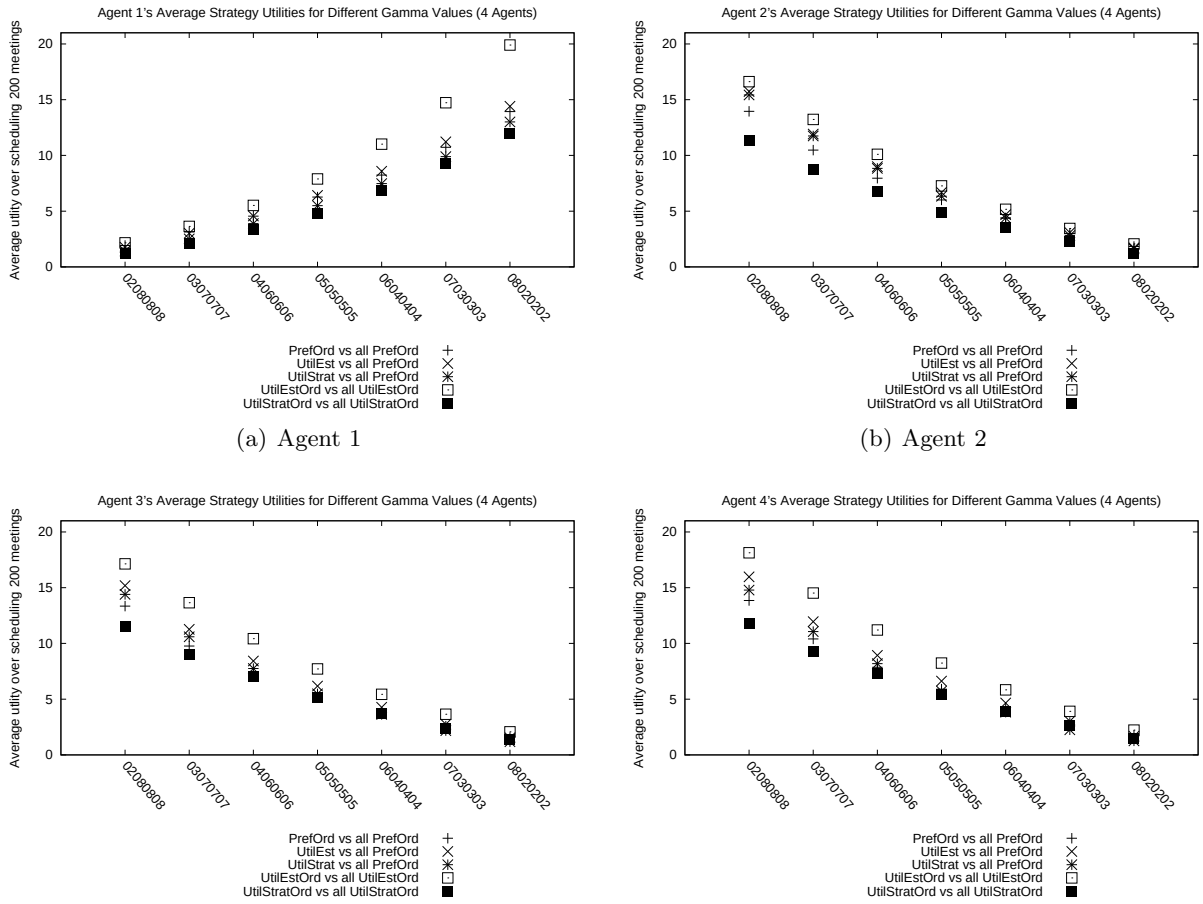
(c) Agent 3

(d) Agent 4

Figure 8.24: Average utilities for agents with same γ, for each of the agents. Calendars reset after 5 meetings scheduled and five preferred times overlapping.

8.4 Summary

In this chapter we explained in detail how the algorithms introduced in the previous chapter for learning online to increase semi-cooperative utility can be applied to two real-world inspired domains. Role Assignment for Multiagent Tasks, and Multiagent Meeting Scheduling, are very different domains, and we show analytically and experimentally that agents using our algorithms learn to increase semi-cooperative utility in these contexts. Furthermore we see that one agent's benefit from learning does not come at the expense of other agents. Rather, that in most cases, the preferred condition for all the agents is when each agent learns and acts according to its own semi-cooperative utility function.



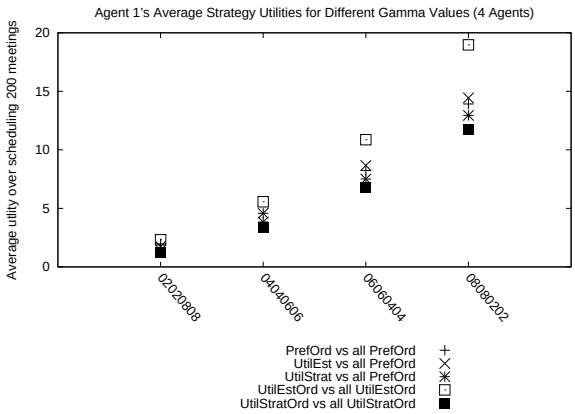
(a) Agent 1

(b) Agent 2

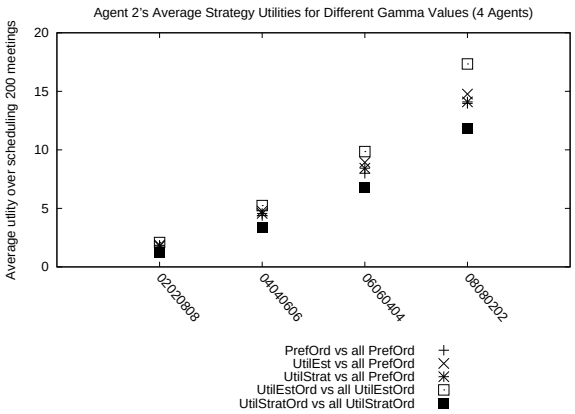
(c) Agent 3

(d) Agent 4

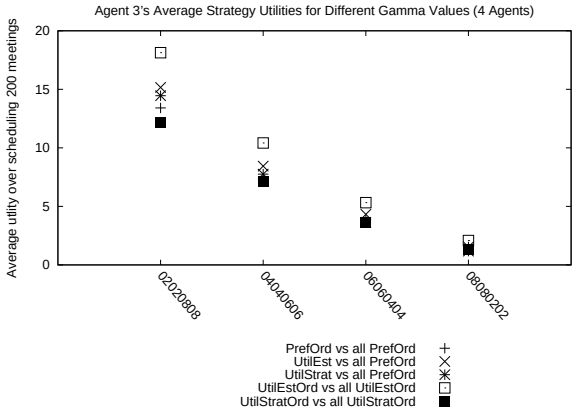
Figure 8.25: Average utilities when Agent 1 has a different γ to the other agents. Calendars reset after 5 meetings scheduled and five preferred times overlapping.



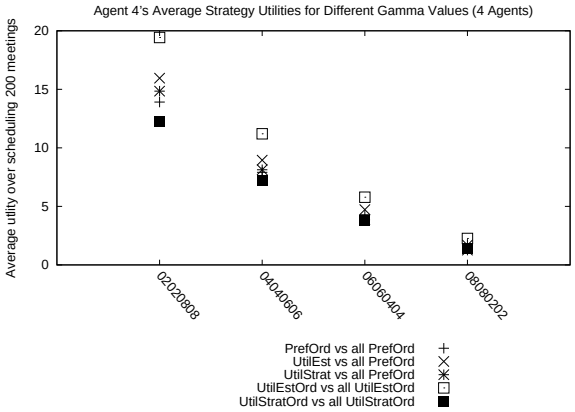
(a) Agent 1



(b) Agent 2



(c) Agent 3



(d) Agent 4

Figure 8.26: Average utilities when Agent 1 has a different γ to the other agents. Calendars reset after 5 meetings scheduled and five preferred times overlapping.

Chapter 9

Additional Results for SC-Agents

In the previous chapter, we demonstrated the effectiveness of algorithms for learning online in SC-EIMAPPs on two real-world inspired domains. In this chapter we include the results from some additional experiments on those domains. In the Role Assignment for Multiagent Tasks domain, we show that our approach achieves greater utility than not learning when negotiating with a large number of agents. In the Multiagent Meeting Scheduling domain, we also demonstrate the effectiveness of learning in the presence of a large number of agents. Additionally, we show that our algorithms perform well when more constraints are introduced into the system.

9.1 Role Assignment for Multiagent Tasks Experiment

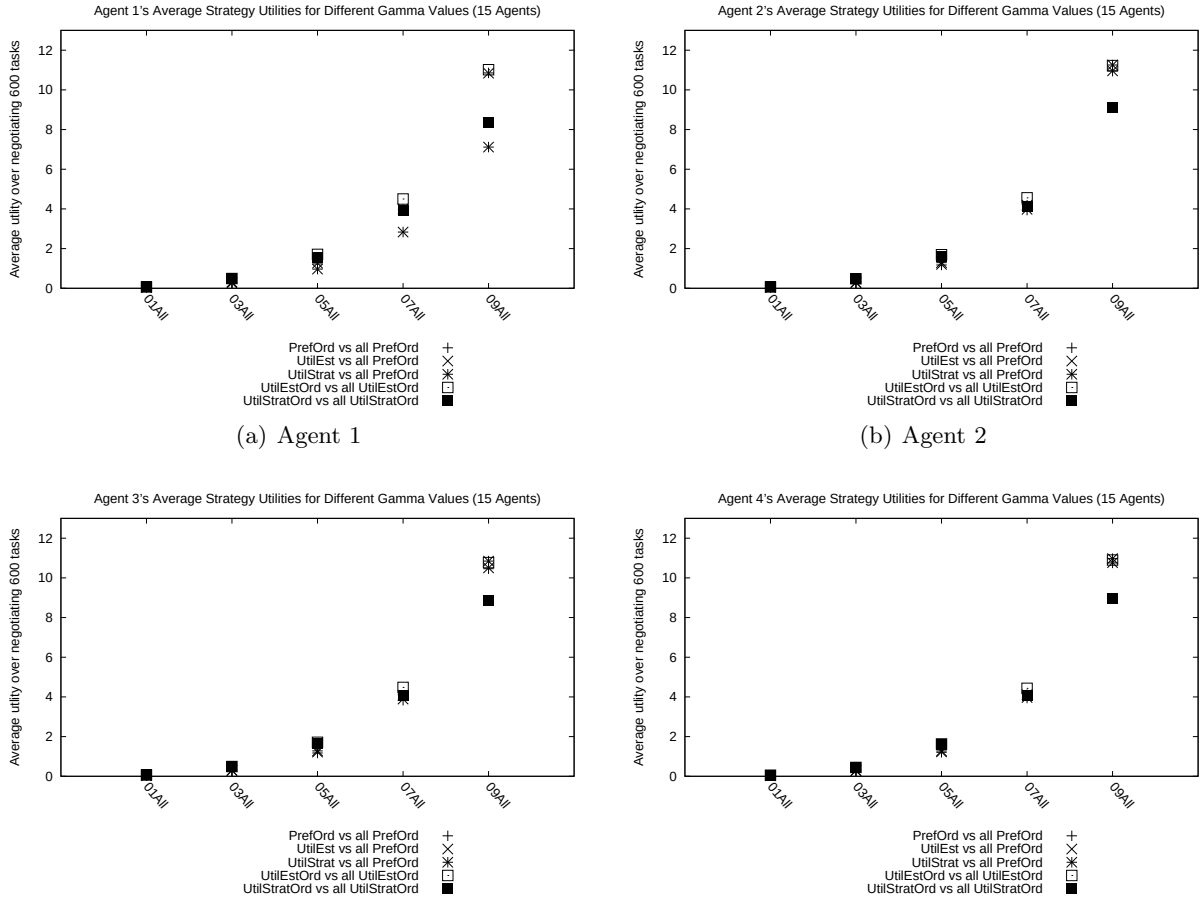
In this experiment we evaluate the performance of 15 agents under the following conditions:

1. all agents use the 1-Preference-Order strategy
2. Agent 1 uses the SC-Utility-Estimate-Ordered-Learner strategy and all the other agents use the 1-Preference-Order strategy
3. Agent 1 uses the SC-Utility-Estimate-Strategic-Learner strategy and all the other agents use the 1-Preference-Order strategy
4. all agents use the SC-Utility-Estimate-Ordered-Learner strategy
5. all agents use the SC-Utility-Estimate-Strategic-Learner strategy

The agents negotiate role assignments for tasks with a participant size of 2, up to 15. There are 20 different types of roles, and the agents have a preference range of 1 up to 20 over the role types.

The utility cut-off (ϵ_a) for all agents, was set 0. We evaluate the average utility the agents achieve in each condition for different utility functions after they have assigned a total of 600 tasks in the system. The results are averaged over 100 trials, and the standard deviation between trials was negligible.

The agents have a mix of preferences over the role types. In some instances they like different role types, and in some instances they prefer the same role type. We have included the the input file for the agents' preferences in Appendix 1.



(a) Agent 1

(b) Agent 2

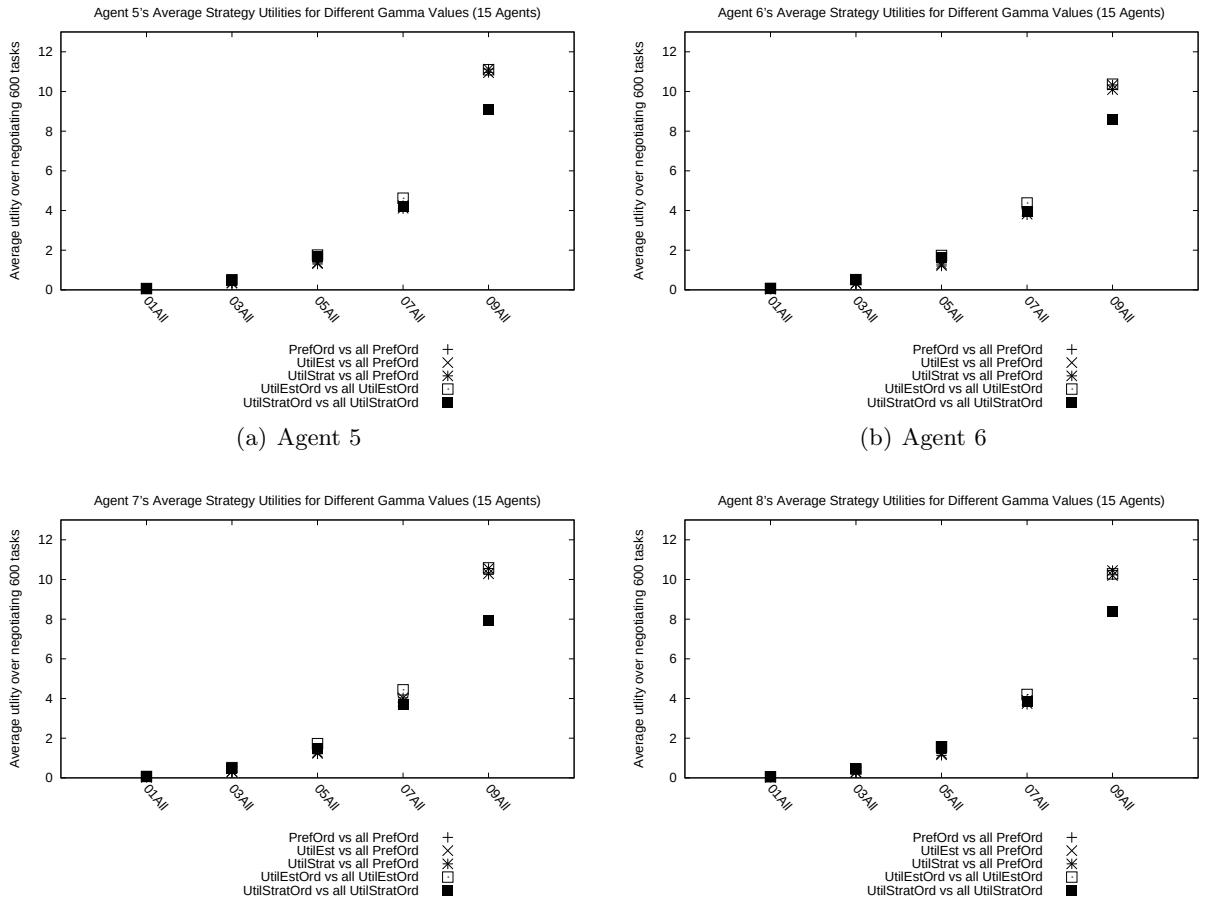
(c) Agent 3

(d) Agent 4

Figure 9.1: Role Assignment: Average utilities for agents with same γ, for agents 1 to 4.

Figures 9.1, 9.2, 9.3, and 9.4 show the average utilities each of the agents achieve under the conditions listed. These graphs show results for the case where all the agents have the same γ value (we demonstrated other pairings in the previous chapter). The graphs show, that the condition where all the agents use the SC-Utility-Estimate-Ordered-Learner is the best or very close to the

best condition, for all agents, for all γ values. Once again, we observed it clearly out-performed the SC-Utility-Estimate-Strategic-Learner for a number of utility values. The SC-Utility-Estimate-Ordered-Learner does not provide a large advantage over not learning for many of the agents in some of the γ conditions, however, it does find advantages where they are available, and does not under-perform not learning.



(a) Agent 5

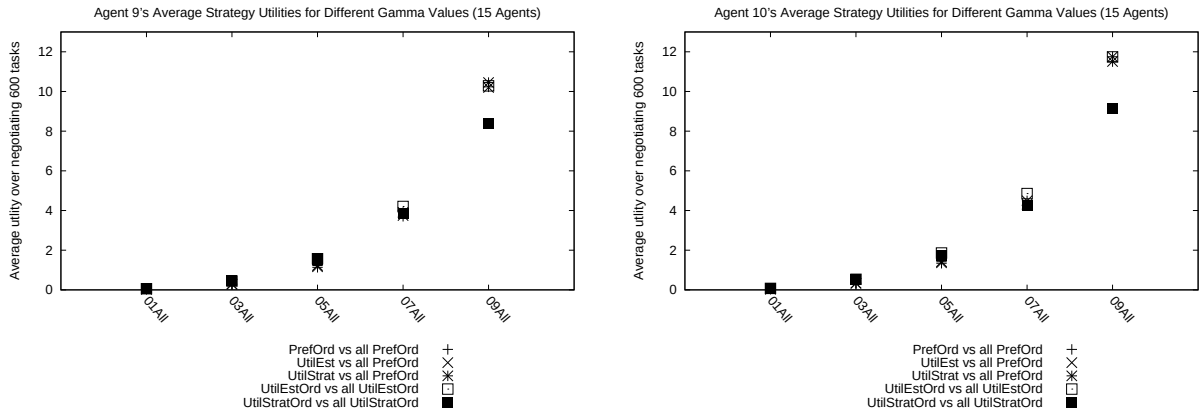
(b) Agent 6

(c) Agent 7

(d) Agent 8

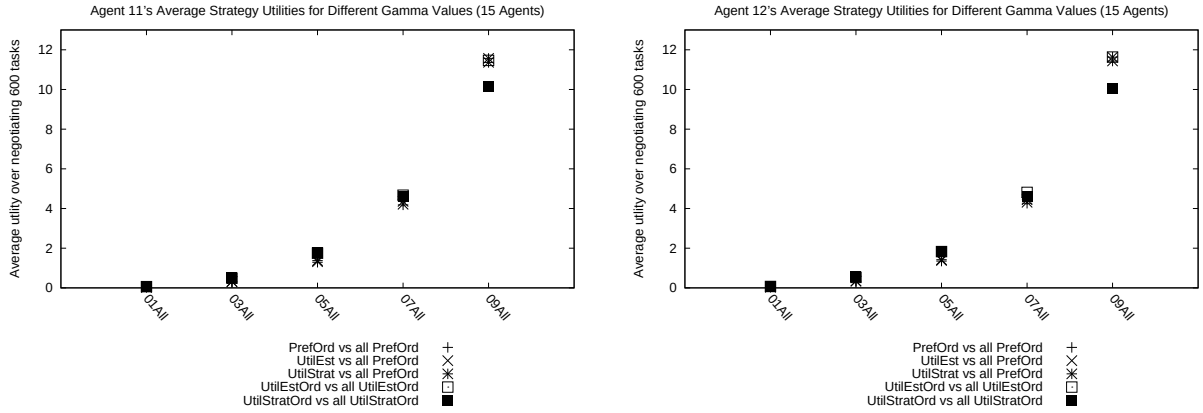
Figure 9.2: Role Assignment: Average utilities for agents with same γ, for agents 5 to 8.

In general, we found when experimenting with large numbers of agents, in the Role Assignment domain, that the preferences of the agents had a large effect on the difference learning could make. Given that an agent does not a-priori know the γ, or the preferences of the other agents it will be negotiating with, our experiments show it is clearly a good strategy to use the SC-Utility-Estimate-Ordered-Learner in the Role Assignment for Multiagent Task domain.



(a) Agent 9

(b) Agent 10



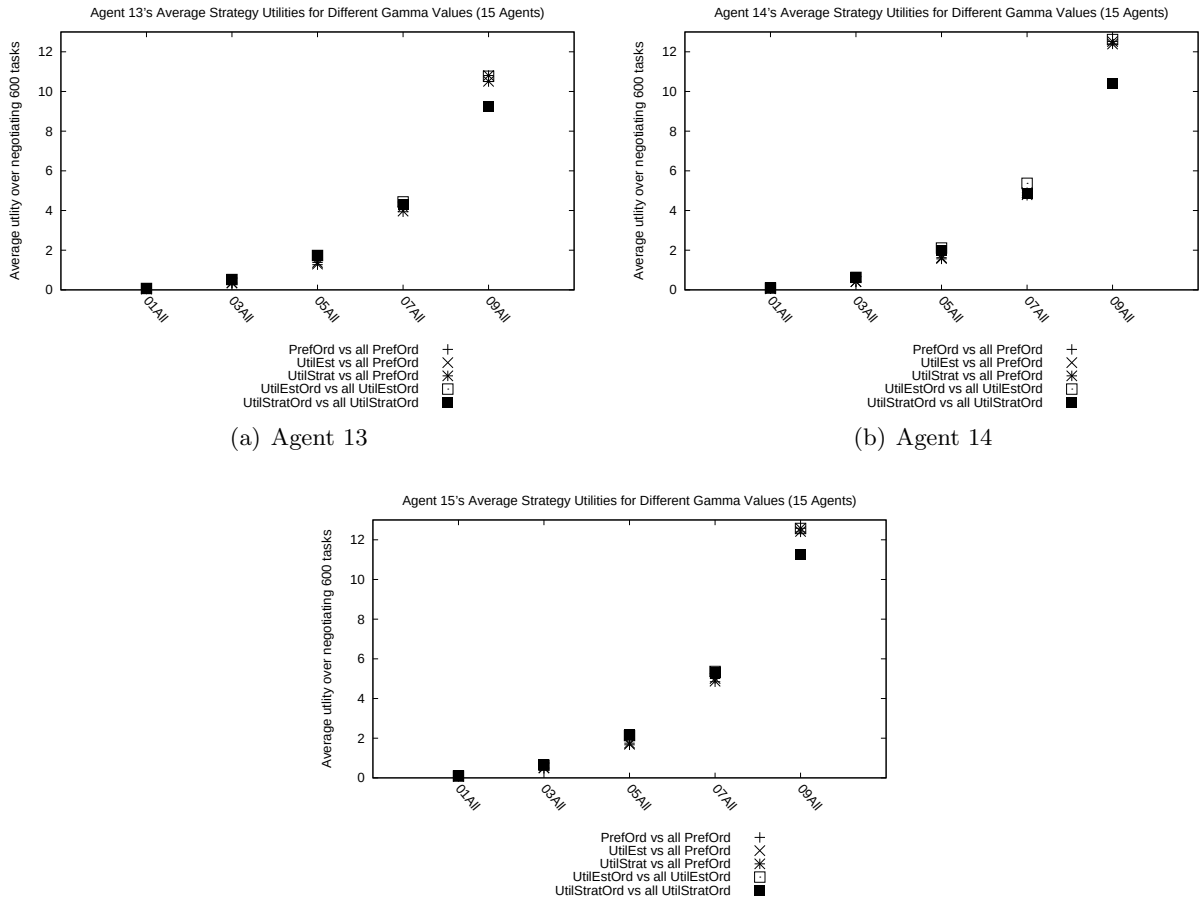
(c) Agent 11

(d) Agent 12

Figure 9.3: Role Assignment: Average utilities for agents with same γ, for agents 9 to 12.

9.2 Meeting Scheduling Domain

For the Multiagent Meeting Scheduling problem, we include two additional experiments. In the first experiment, we show how our learning algorithms perform when the agents have more constraints to deal with. In the second experiment, we demonstrate the effectiveness of our algorithms for learning in a larger agent system.



(a) Agent 13

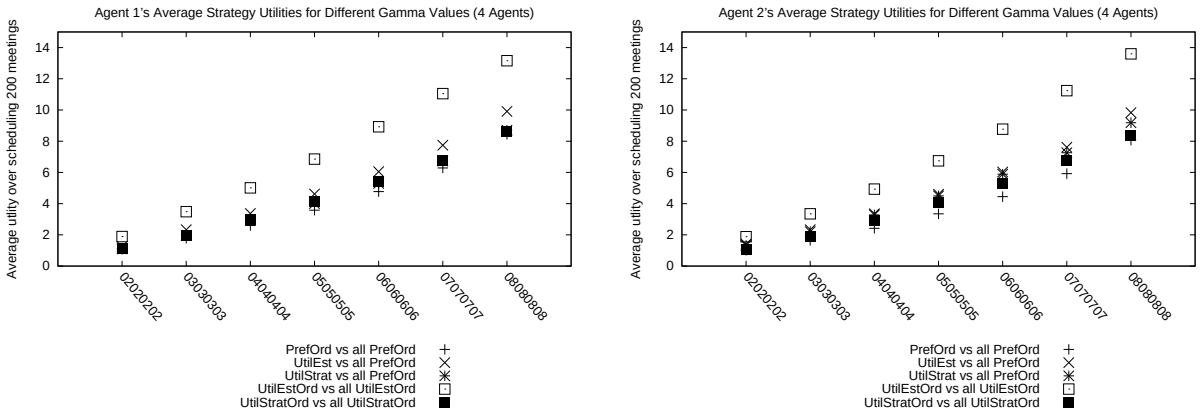
(b) Agent 14

(c) Agent 15

Figure 9.4: Role Assignment: Average utilities for agents with same γ, for agents 13 to 15.

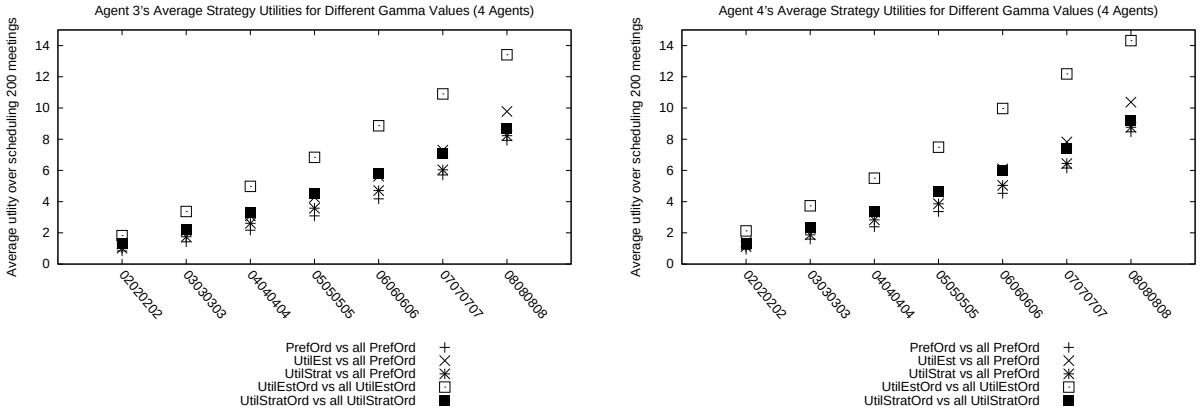
9.2.1 Experiment with Additional Constraints

In the experiments in the previous chapter, we looked at the case where the agents had no initial meetings. This meant that the only constraints on the agents arose from the scheduling of meetings before the calendars were reset. In this experiment, we return to our 4 agent meeting scheduling problem. This time, we include 25 randomly generated initial meetings. We also increase the reset interval to 10. Therefore, there can be up to 35 constraints (total) within the system by the time the agents are reset to their initial calendars. As in the previous experiments, the agents schedule 200 meetings and the average utility is evaluated over 100 trials (the standard deviation was found to be negligible). The agents have an overlap in preferences of size 3, with a preference value of 40, and a preference value of 5 for all other times.



(a) Agent 1

(b) Agent 2

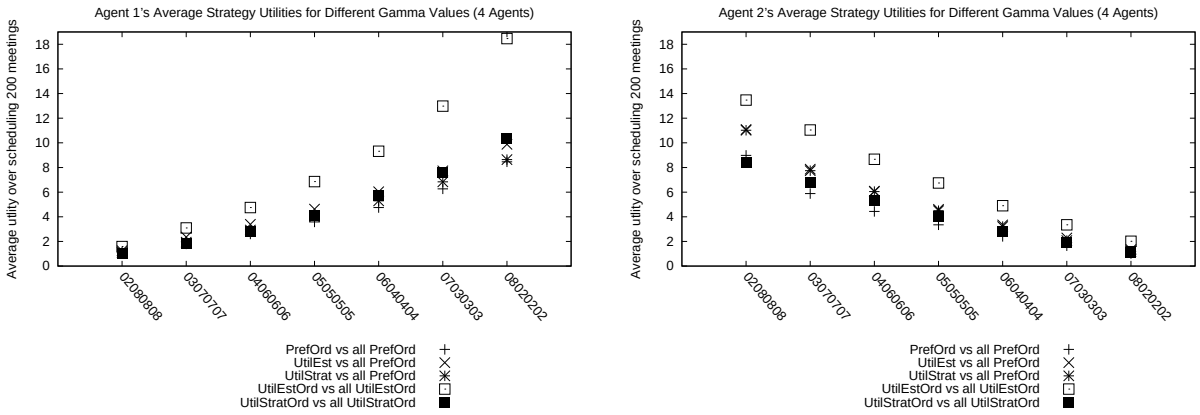


(c) Agent 3

(d) Agent 4

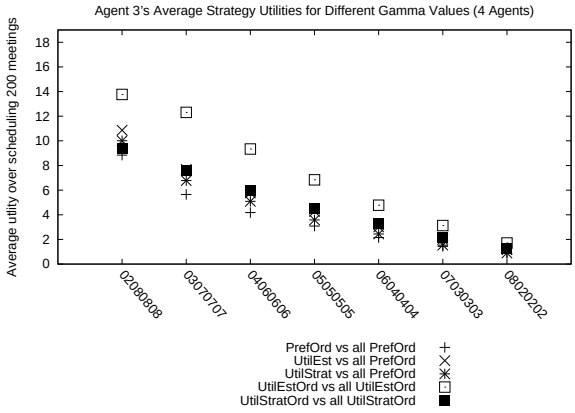
Figure 9.5: Meeting Scheduling: Average utilities for agents with same γ, for each of the agents. Calendars reset after 10 meetings scheduled and 3 preferred times overlapping.

Figures 9.5, 9.6, and 9.7, clearly demonstrate the benefits to all agents of using the SC-Utility-Estimate-Ordered-Learner. Under all the different utility conditions, including the conditions where the agents have very different γ values, this was the preferred condition for all agents. We also see that despite the additional random constraints, the condition where all agents use the SC-Utility-Estimate-Ordered-Learner significantly outperforms not learning. We note however, that if the initial meetings were randomly scheduled, such that all the times in the intersection of preferred times were taken, we would not have observed as great a boost from learning. We also note that the condition where all agents use the SC-Utility-Estimate-Strategic-Learner out-performs not learning in almost all cases (it is never worse), but that it achieves significantly lower average utility than the condition where all agents use the SC-Utility-Estimate-Ordered-Learner.

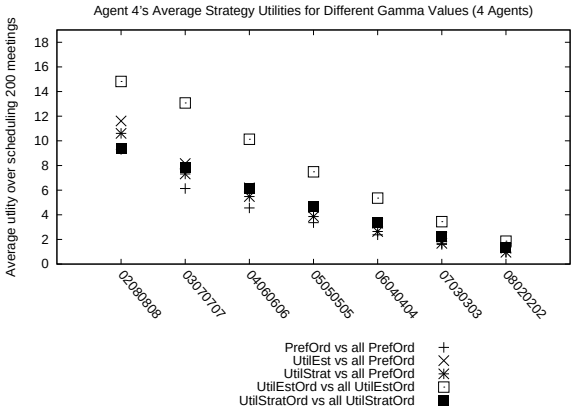


(a) Agent 1

(b) Agent 2



(c) Agent 3



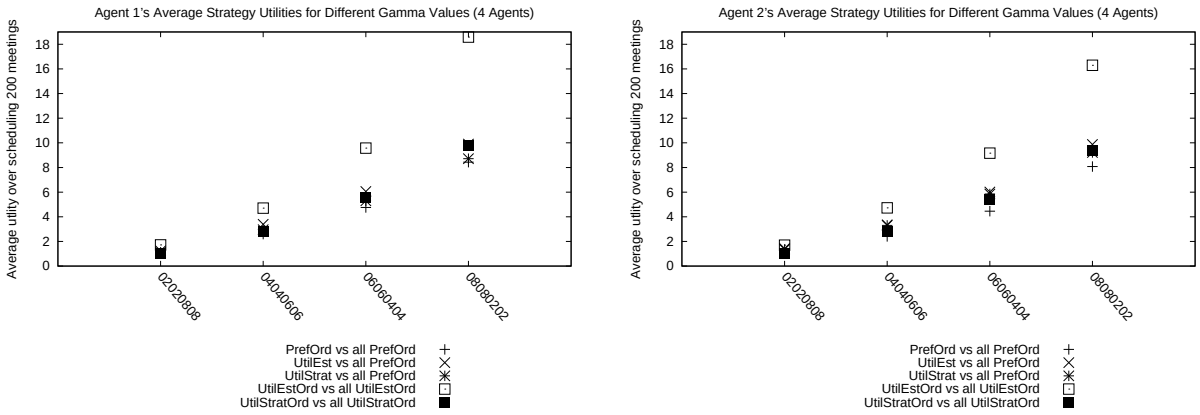
(d) Agent 4

Figure 9.6: Meeting Scheduling: Average utilities when Agent 1 has a different γ to the other agents. Calendars reset after 10 meetings scheduled and 3 preferred times overlapping.

This experiment demonstrated that in the presence of additional constraints (from a longer reset value, and initial meetings) our learning approach effectively adapts and increases semi-cooperative utility for all agents.

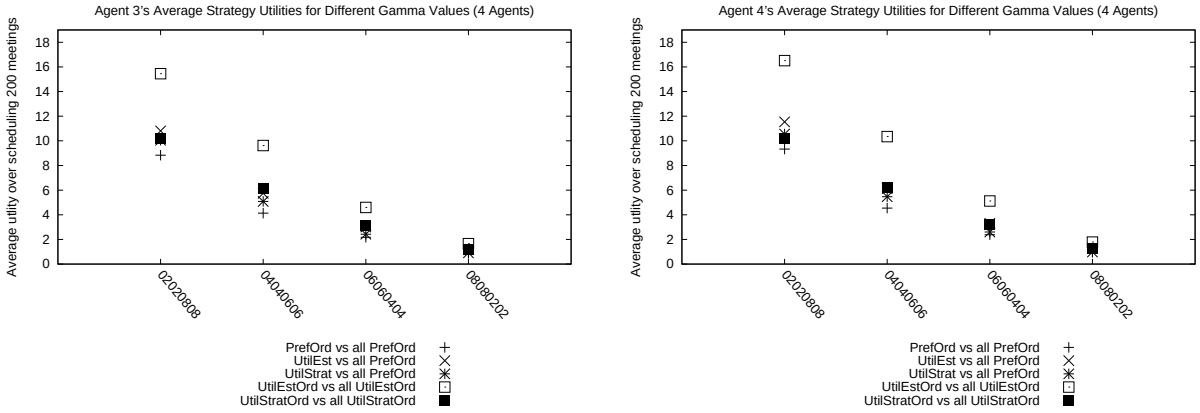
9.2.2 Experiment with 14 Agents

In this experiment, we evaluate 14 agents scheduling meetings of size 2, up to size 14. The agents schedule a total of 200 meetings and we ran 100 trials (the standard deviation was insignificant). Between all 14 agents there is an overlap in preferred times of size 4. The agents each have 5 other preferred times (preference value 40), which they share with one other agent only (Agent 1 with



(a) Agent 1

(b) Agent 2



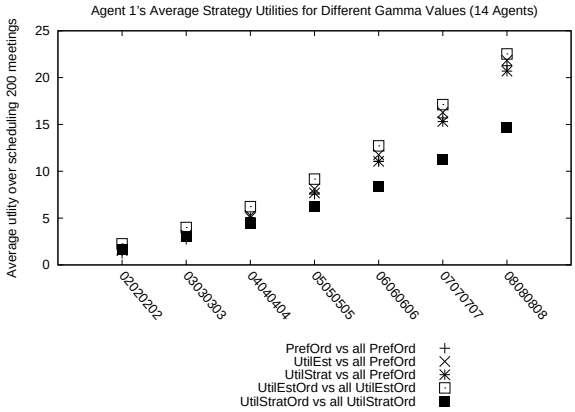
(c) Agent 3

(d) Agent 4

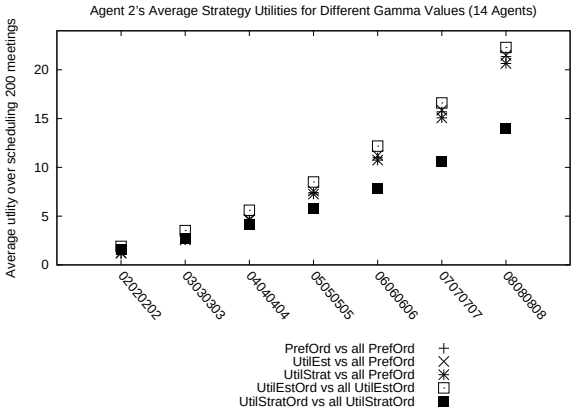
Figure 9.7: Meeting Scheduling: Average utilities when Agent 1 has a different γ to the other agents. Calendars reset after 10 meetings scheduled and 3 preferred times overlapping.

Agent 7, Agent 2 with Agent 8, etc.). Since the first 7 agents share preference values with the last 7 agents, we just show graphs for the first 7. We used a reset value of 5 in the experiment, and there were no initial meetings.

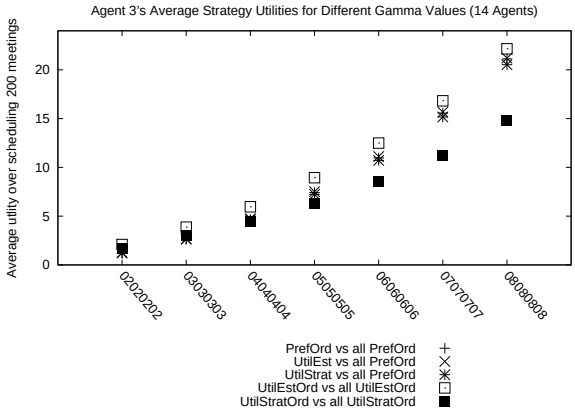
Figures 9.8, 9.9, 9.10, 9.11, 9.12, and 9.13, show that despite the small number of tasks over which to learn, compared to the number of agents to learn about, the preferred condition, for all agents, and utility values, is where all agents use the SC-Utility-Estimate-Ordered-Learner. We see the largest gains from learning in the situations where Agent 1 has a different γ value to the other agents (including, significantly, to this phenomenon the agent with the same preferences as it).



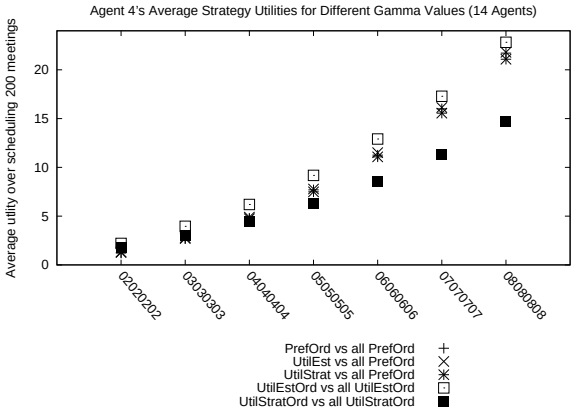
(a) Agent 1



(b) Agent 2

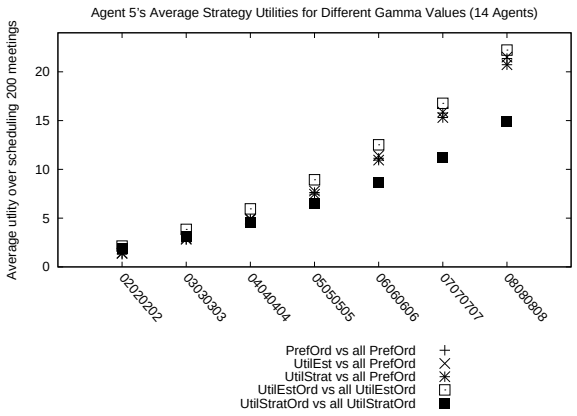


(c) Agent 3

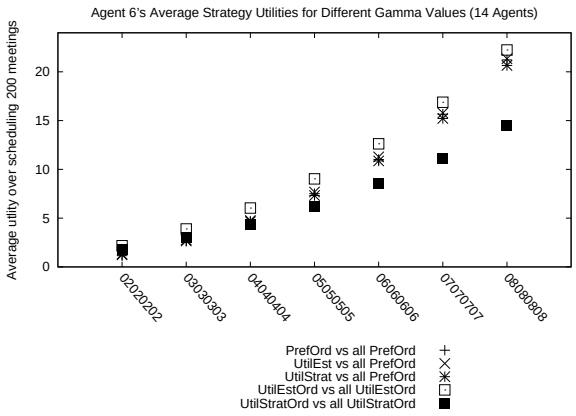


(d) Agent 4

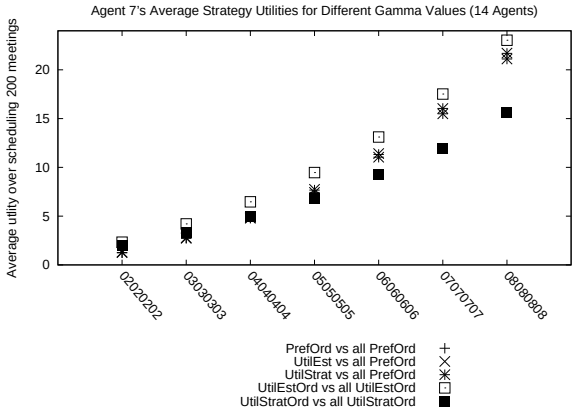
Figure 9.8: Meeting Scheduling: Average utilities for agents with same γ, for each of the agents. There are 14 agents in the system and the agents share 4 preferred times.



(a) Agent 5

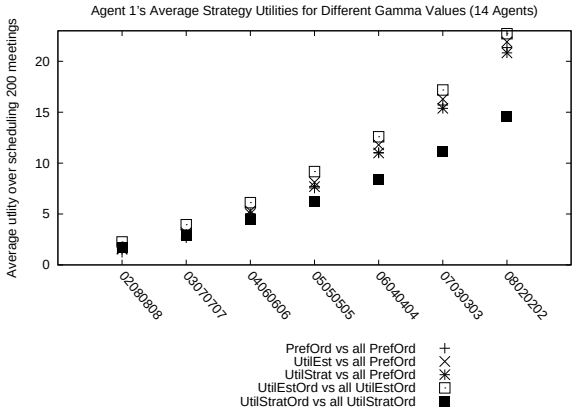


(b) Agent 6

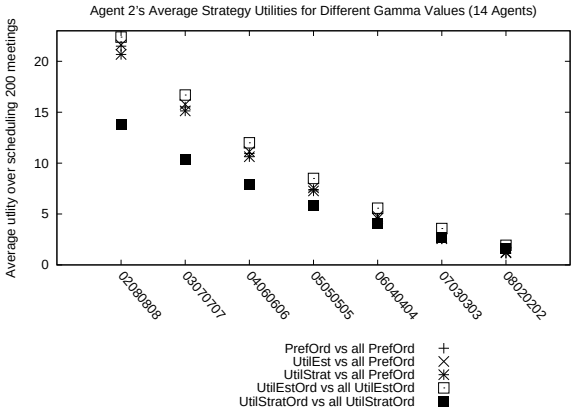


(c) Agent 7

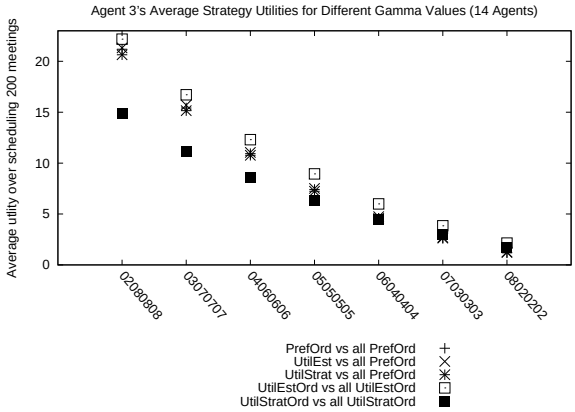
Figure 9.9: Meeting Scheduling: Average utilities for agents with same γ, for each of the agents. There are 14 agents in the system and the agents share 4 preferred times.



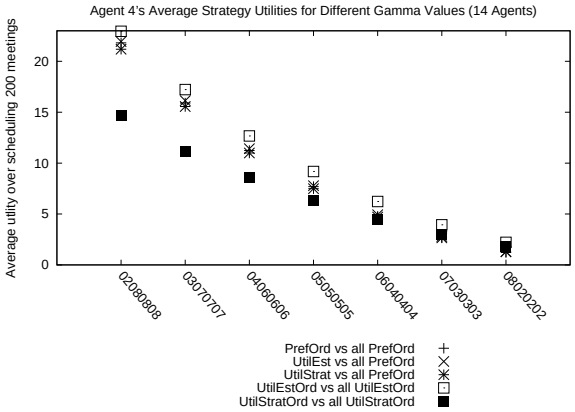
(a) Agent 1



(b) Agent 2

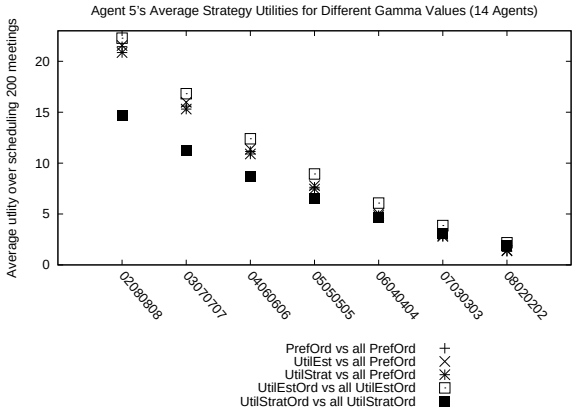


(c) Agent 3

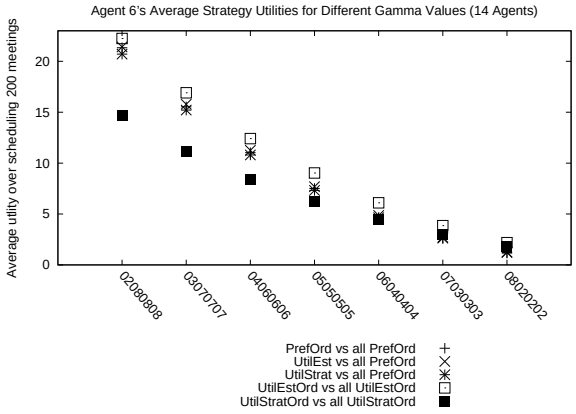


(d) Agent 4

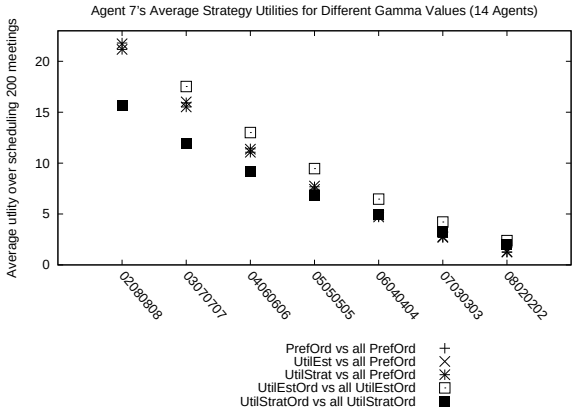
Figure 9.10: Meeting Scheduling: Average utilities when Agent 1 has a different γ to the other agents. There are 14 agents in the system and the agents share 4 preferred times.



(a) Agent 5

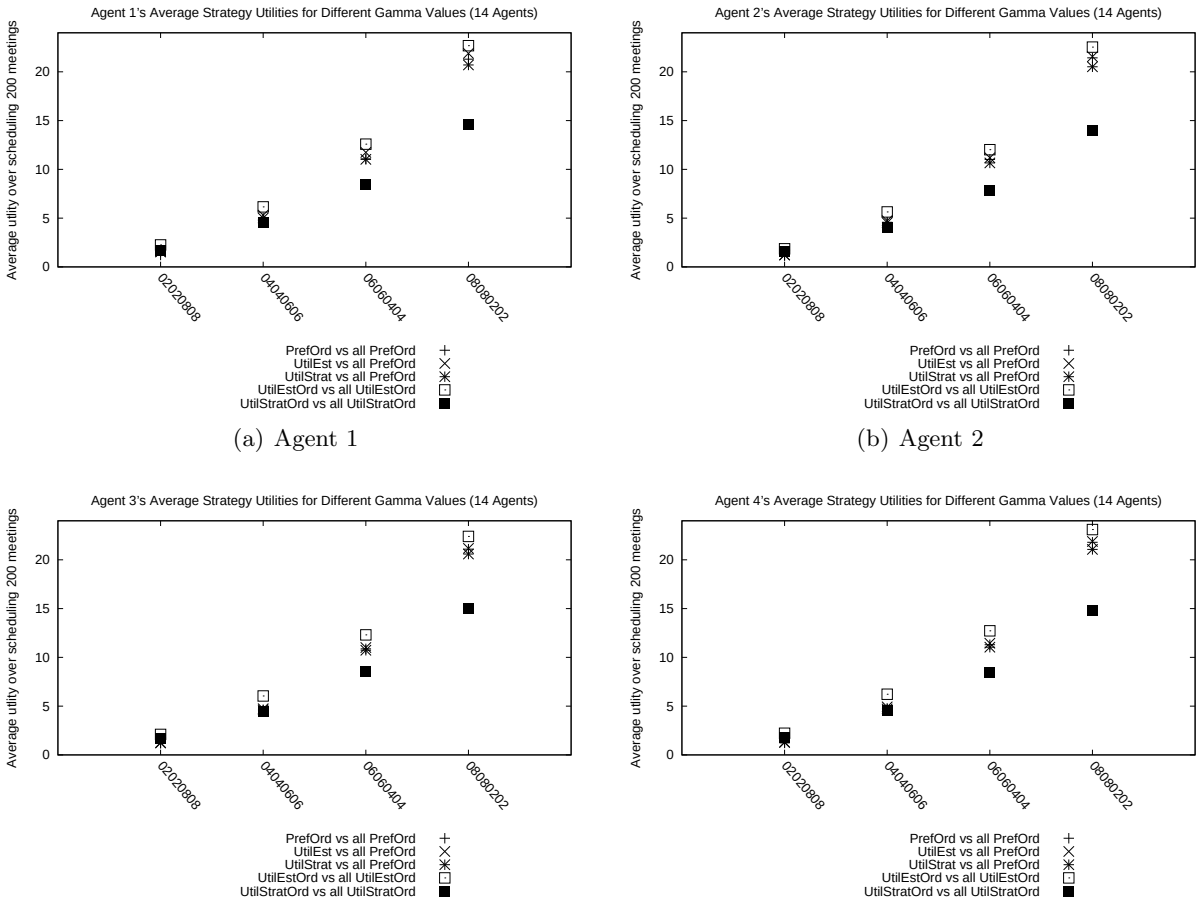


(b) Agent 6



(c) Agent 7

Figure 9.11: Meeting Scheduling: Average utilities when Agent 1 has a different γ to the other agents. There are 14 agents in the system and the agents share 4 preferred times.



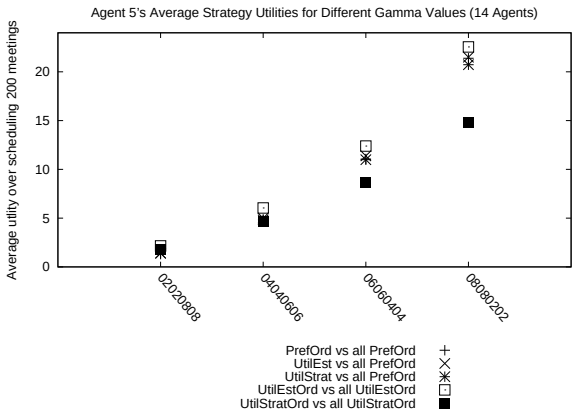
(a) Agent 1

(b) Agent 2

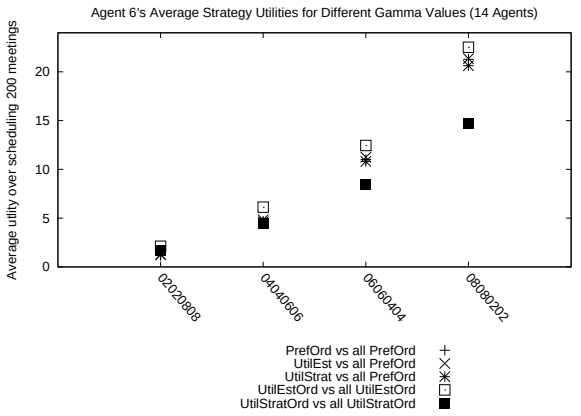
(c) Agent 3

(d) Agent 4

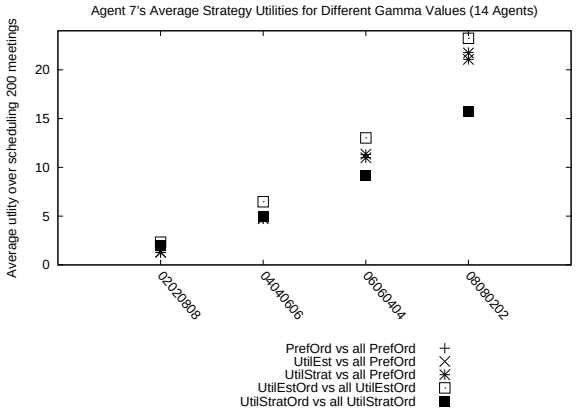
Figure 9.12: Meeting Scheduling: Average utilities when Agent 1 has a different γ to the other agents. There are 14 agents in the system and the agents share 4 preferred times.



(a) Agent 5



(b) Agent 6



(c) Agent 7

Figure 9.13: Meeting Scheduling: Average utilities when Agent 1 has a different γ to the other agents. There are 14 agents in the system and the agents share 4 preferred times.

9.3 Summary

We have demonstrated experimentally in this chapter, and in the previous chapter, that when semi-cooperative agents use the SC-Utility-Estimate-Ordered-Learner, all the agents involved benefit in the Role Assignment for Multiagent Tasks and Multiagent Meeting Scheduling domains. Given our results, we believe the SC-Utility-Estimate-Ordered-Learning approach is an effective method of online adaption for semi-cooperative agents to employ in a variety SC-EIMAPP domains.

Chapter 10

Related Work

In this chapter we describe some of the related work on agreement problems. We place particular emphasis on learning approaches studied in domains similar to the ones we use in this thesis.

10.1 Overview

Approaches to solving agreement problems can be classified according to two main axes: *cooperative* versus *self-interested* and *centralized* versus *distributed*.

Cooperative approaches to reaching agreements in the style of Distributed Constraint Satisfaction [51] such as that described in [37] are particularly appropriate in domains where it is reasonable to assume that agents will not attempt to game the system e.g., because the agents are part of the same team. In many real-world domains however, agents are representing partially self-interested users. This is not to say that agreement problems are *zero-sum*, but rather that *agents have individual utility functions* as opposed to a *joint team utility function*.

Centralized approaches to reaching agreements generally involve a *trusted central authority*. The agents are required to submit their constraints and preferences to the authority. If the setting is cooperative, the agents can be assumed to truthfully reveal all their privately owned information. In situations where the agents have some self-interest, truth revealing mechanisms e.g., second price auctions, are needed so the agents are incentivized to participate honestly. Truth revealing mechanisms often require the transfer of utility between agents. In domains that already involve money transfers such as the buying and selling of goods, these mechanisms can be a particularly effective solution. In domains that don't, introducing transfers, or creating artificial monetary systems can lead to complications. Furthermore, as we discuss in [14], it can be very difficult to design suitable mechanisms for complex agreement problems. For example, one of the issues

we address in [14], is that in incremental agreement problems the decision mechanism is executed multiple times. If the agents have preferences that depend on the outcome of future assignments, then a variety of incentive compatibility issues arise.

Instead of having agents express cardinal preference values we can think of an agent's preferences as an ordering over the set of options. In this context we can model agreement problems as a classical social choice/voting problem. In this case, *Arrow's Impossibility Theorem* [1] is informative about the results it is possible to obtain.

Let the set of options $\mathcal{O}$ contain three or more alternatives. Let each agent a_i have a *transitive preference* over $\mathcal{O}$, i.e., a complete ranking (from top to bottom) over the options in $\mathcal{O}$, allowing ties. In the notation of [25] we let a *constitution* be a function that associates with every tuple of agent preferences a *social-preference* which like the individual preferences is transitive.

A *constitution* satisfies *unanimity* if the *social-preference* produced puts o_i strictly higher than o_j whenever every agent puts o_i strictly higher o_j. It satisfies *independence of irrelevant alternatives* if the relative ranking of o_i and o_j in the *social-preference* only depends on the relative rankings of o_i and o_j in the agents' preferences (e.g., not on the rankings of any other options, say o_k). Finally a constitution is said to be a *dictatorship* if there is an agent a_i such that for all pairs o_i and o_j if a_i strictly prefers o_i to o_j then the *social-preference* strictly preference strictly prefers o_i to o_j.

We can state Arrow's theorem as *any constitution that satisfies transitivity, independence of irrelevant alternatives and unanimity is a dictatorship.* The key point is that there is no ideal voting method (constitution) since not all these minimal criteria can be satisfied at once. A number of work arounds to Arrow's Impossibility Theorem have been examined that involve relaxing one or more of the requirements. In modern democracies majority rule (*largest number of votes*) and Borda Count (e.g., [48]) are often used despite their problems. In terms of this thesis the implication of Arrow's impossibility theorem is that there does not exist a method for obtaining the true ordinal preferences of agents and selecting a good outcome based on some sensible combination of those preferences. As such, we focus on how much the learning approaches we introduce in this thesis improve utility from the perspective of each of the individual agents.

Most existing approaches to agreement problems are distributed, and generally feature some type of agent-based negotiation.

Game-theoretic negotiation approaches view the process as a general-sum, extensive-form game where the agents are perfectly rational and self-interested (and this is common knowledge). In game theory, when agents have private information, this is modeled as the agents each having some initial beliefs about each other. These beliefs are typically considered to be common knowledge to all the agents. Game-theoretic approaches work well when the assumptions hold, and when there is a *single equilibrium solution.* When some of the assumptions do not hold, or when there are multiple equilibria, it is not clear what strategy the agent should chose.

Given the difficulties with using game-theoretic reasoning, some approaches have the agents use domain appropriate heuristics and/or learn about the privately owned information of others to try and achieve high utility. Much of the work on learning in self-interested negotiation has been in the domain where a buyer and seller negotiate the price of an item or items. In this domain (e.g., [52]), the private information may be each agent's reservation price. Since the utility the agents receive from reaching an agreement is discounted according to negotiation time, learning about the other agent's reservation price can be helpful since the agent can use this information to speed up the negotiation by making acceptable offers.

Cooperative approaches to distributed negotiation have also been considered. In cooperative negotiation, the agents are assumed to comply with a restrictive protocol e.g., to accept a proposal if it does not violate a hard constraint. These approaches have also featured learning. Sen and Durfee [46] looked at how the agents could observe the state of the system in order to chose strategies that would improve overall efficiency. Others (e.g. [8, 50]) have used the information revealed in the course of negotiations to model other agents, then used the models to increase the efficiency of negotiation. Learning is generally easier in cooperative scenarios, since the assumption of compliance with a restrictive negotiation protocol supplies a lot of correctly labeled training data. For instance, if an agent is required to accept any proposal that does not violate a hard constraint, we know that any rejected proposal *does* violate a hard constraint. In the model where agents have their own utility functions however, an agent may instead reject a proposal because it does not *prefer* the option, or because its negotiation strategy stipulates a different behavior at this point in time etc.

Another approach to negotiation is argumentation. In argumentation, proposals are backed by persuasive arguments based on logical models of the agents' mental states (e.g., a model of mental state might represent beliefs, desires and intentions)(see [41] for an introduction). The aim of the arguments is to change the mental state of the other agents in order to reach agreements.

10.2 Approaches to Solving EIMAPP Style Agreement Problems

In this section we briefly discuss some different distributed approaches to solving agreement problems. For the purposes of our discussion, we categorize approaches as cooperative or self-interested, game-theoretic or heuristic, learning or non-learning.

Cooperative Approaches

We say an approach to solving agreement problems is cooperative, if the agents essentially have a joint utility function which they try to optimize. This differs from self-interested approaches, where despite the fact that the agents are incentivized to "cooperate" and reach an agreement, they are

motivated by individual utility functions.

One style of cooperative approach involves the use of the Distributed Constraint Satisfaction paradigm. The most similar work to ours is that of Modi et al. [37]. Modi et al., use a Distributed Constraint Reasoning (DCR) style of approach to modeling multiagent agreement problems. In particular, they introduce the *Iterative Agreement Protocol* for reaching agreements in IMAP with assign-once per-agent options. In this protocol one agent is designated as the initiator of the variable v. The protocol operates in rounds as follows:

1. initiator proposes one option to all the agents in $\mathcal{D}_v$
2. each agent in $\mathcal{D}_v$, sends an accept/reject response to the initiator
3. if all the agents have accepted, then there is an agreement and the variable is assigned
4. else go to 1. and repeat

A key feature of the protocol is that an agent α accepts a proposal *whenever* it won't violate the use-once per-agent constraint, i.e, whenever α currently has the option available for assignment. If the option is assigned to another variable, then α must decide whether to bump the current assignment. Modi et al., introduce a number of strategies an agent can use to decide when to bump one variable in favor of another.

The approach proposed by Modi et al. [37], is most useful when agents do not have preferences. Modi et al., suggest that preferences about options could be expressed as additional constraints. For instance, a constraint that option o should never be used. This works very well for preferences that are easily modeled by "hard" constraints. However, the approach does not provide a ready solution for handling preferences in general i.e., "soft constraints".

In Modi et al.'s Iterative Agreement Protocol, described above, the initiating agent has some control over what information it reveals, since it chooses which options to propose. The other agents however have much less control. They must explicitly accept or reject each proposal. In the protocol a rejection means that the agent has already assigned the option to another variable. An acceptance reveals less information, since the agent might have the option free or simply be willing to bump the variable currently assigned to the option. The key advantage of this approach, is that since agents are required to accept a proposal whenever the assign-once per-agent constraint is not violated, the process should settle on a option fairly quickly, thereby keeping the number of options the agents must accept/reject comparatively low. The disadvantage is the lack of control the agents have over what information they reveal. The protocol assumes the agents are cooperative in the sense that they will comply with the protocol. Only the individual agents knows when its constraints are not violated and it can accept proposals, there is no enforcement. In this thesis we focus on less constrained protocols.

There are many examples of cooperative approaches to solving agreement problems that are not based on the DCR paradigm. For example, in the domain of multi-agent meeting scheduling, Shintani et al. [47] propose a persuasion based approach. The persuasion mechanism involves compromising agents adjusting their preferences so that their most preferred times are the persuading agent's most preferred times. This method relies strongly on the agents complying with the protocol. Garrido and Sycara [24] and Jennings and Jackson [28] take the approach of allowing agents to not only propose meeting times, but also to quantify their preferences for proposals. The agent that is collecting the proposals, can then make decisions based on the reported utilities of all the meeting participants. This style of approach involves a lot of trust, since for the procedure to work well all the agents must report their preferences truthfully.

Cooperative approaches work well in domains where the agents naturally have no self-interest, i.e., where they do not represent people or entities that have individual preferences. Cooperative approaches can also work well in closed systems where it can be reasonably assumed that all agents in the system implement the protocol and strategies as required. In this thesis we focus on domains where there is some self-interested element to agent utility.

Game-Theoretic Negotiation

In the game-theoretic model agents are assumed to be self-interested and rational, in other words that they will act strategically in order to optimize their utility [22]. Furthermore, it is assumed to be common knowledge that each agent is self-interest and rational (i.e., agent 1 knows agent 2 is self-interested and rational and agent 2 knows that agent 1 knows this and so forth) [22]. When these assumptions are appropriate game theory can sometimes be used to reason about agent interactions.

Azoulay-Schwartz and Kraus [3, 30] apply a game theory based approach to the problem of data allocation across multiple self-interested serves. The servers negotiate in order to decide how data is allocated across the servers. The negotiation protocol operates in rounds. In the $i'th$ round, the $i'th$ agent proposes an allocation. The other agents must *accept or reject* the proposal, or *opt-out* of the negotiation. If all agents accept, then an agreement has been reached. If one or more agents opts-out then some pre-set allocation is used (called the *conflict allocation*). Otherwise negotiation proceeds to the next round. The negotiation is analyzed game theoretically by looking at the *sub-game perfect equilibria*. A strategy profile specifies a strategy for each agent in the game (in this case a negotiation strategy for each agent). If at every round in the negotiation, each agent has no incentive to unilaterally deviate from its strategy in some strategy profile X, then we say X is a sub-game perfect equilibrium. A standard problem with using game-theoretic approaches is that there are typically multiple equilibria. So the question becomes, how do the agents arrive at using one of the equilibrium strategy profiles? For the server problem Azoulay-Schwartz and Kraus [3,30] solved this difficulty by attaching a truth revealing mechanism to the process. The servers have to broadcast their private information at the start of the process and this information is used to

select the equilibrium. When the negotiation starts the first agent just proposes the appropriate allocation and all the agents will accept it. It is not really necessary however to model this process as a negotiation - the whole process could be modeled as a mechanism.

In some domains, such as the buying and selling of goods, mechanisms (e.g., second-price auctions) are very useful, but in others we can't necessarily design a suitable truth-revealing mechanism. In the data allocation problem, Azoulay-Schwartz and Kraus, were able to prove truth-telling was a Nash Equilibrium with regards to some of the privately owned information, since the information was partly shared and conflicting reports were punished. However for information that is solely owned by one agent, Azoulay-Schwartz and Kraus's claim is much weaker. In this case, they claim that in general it is intractable for an agent to find a beneficial lie. This of course may not be true for many specific cases. This problem is particularly important when we consider agreement problems in general (as we do in this thesis), not just data allocation problems. For example, some difficulties with designing a mechanism for the meeting scheduling problem are discussed in [14].

One well known protocol that has been analyzed game-theoretically is the *monotonic concession protocol* [43]. This is another round-based protocol. In the bi-lateral scenario, both agents start by making a proposal. In each round the agents can make a concession by making a new proposal that the other agent will prefer, or refuse to concede and just maintain their current proposal. The negotiation ends when neither agent concedes or when one agent makes a proposal that the other agent thinks is as good or better than what it is proposing itself. If the protocol ends without an agreement (i.e., because neither agent conceded) the outcome is the pre-defined 'conflict deal'. The *Zeuthen strategy* [53] for this protocol has been shown to have some interesting properties [27]. In this strategy, the agent with the most to risk by not conceding makes the next concession and this concession should be the minimal concession such that the other agent will have the most to risk by not conceding in the next round. This strategy represents an intuitive way to negotiate and results in an outcome that maximizes the product of the agents' utilities. However the strategy profile where both players use the Zeuthen strategy is not stable. There is a case in the second last round where an agent can benefit from deviating from the strategy. The monotonic concession protocol was originally presented for bi-lateral negotiation. Endriss [18] recently made significant progress in adapting the protocol to the multilateral case. In this thesis, we focus on domains where the utility functions of the other agents are not common knowledge. The monotonic concession protocol and the one step variant [43] require the utilities of all the agents to be common knowledge.

In game theory, when agents have incomplete information, this is modeled as the agents each having some initial beliefs. However, these initial beliefs are typically considered to be common knowledge to all agents, which is not a realistic assumption in many agreement problems. Saha and Sen [45] take a game theoretic approach to the problem of multiple indivisible resources *where the agents' utilities are not common knowledge*. The use a reasonable assumption about the risk taking behavior of the agents and exploit the structure of the problem to design a protocol that produces Pareto-optimal outcomes. The same kind of protocol cannot be used in a domain like meeting scheduling since the problem structure is very different, but their approach demonstrates that game theoretic techniques can be useful solving agreement problems even when the agents

have private information. Fatima et al. [20] consider a domain where a buyer and seller must agree upon a price. There are two possibilities for the buyer and sellers respective deadlines and reserve prices. The agents' probabilistic beliefs about the deadline and reserve of the other agent is private. In this restricted scenario, they are able to analyze a negotiation protocol and show the existence of a unique equilibrium in some cases and two equilibrium in other cases.

We have given some examples of how game theory has been applied to various agreement problems. The usefulness of taking a game theoretic approach depends largely on the nature of the problem in question. Some common problems that arise are:

- violation of the perfect rationality assumption - agents often have bounded computational resources and the amount of reasoning required to act strategically can exceed these resources.
- multiple equilibrium - if there are multiple equilibria it is not clear how the agents should act.
- private information - in many multiagent agreement problems the agents have private information, this makes game theoretic reasoning much more complex (and in many cases intractable).

Heuristic Style Negotiation

Since it may not be possible in a given situation to use game theoretic reasoning to reach an optimal solution (e.g., due to one of the required assumptions, such as perfect rationality being violated) some approaches simply seek to produce *good* solutions according to the agent's utility function. These approaches may use domain knowledge, or learning, in order to make negotiation decisions in the context of the protocol that is in place. As an example, Sen and Durfee [46] studied the efficiency of different negotiation strategies in the multiagent meeting scheduling domain. They found that the efficiency (defined in terms of how long it took to schedule meetings, and how many meetings were scheduled) of different strategies depended on various aspects of the system state, such as how busy the agents were, and how many meetings were being scheduled at once. If they are to detect aspects of the system state, this sort of domain knowledge can be used by the agent to chose their strategy.

10.3 Learning to Improve Negotiation

In this thesis, we focus in particular on heuristic approaches that involve learning. Bayesian methods have been applied to the problem of preference learning in negotiation, by for example [7,8,52]. Zeng and Sycara [52] propose Bayesian learning as a generally useful approach to learning in negotiation. In the domain where a buyer and seller negotiate a price for a single good they show that the agents

can use a Bayesian approach to learning each other's reservation prices to increase joint utility and efficiency in the context of a particular negotiation protocol and strategy.

Bui et al., [8] apply Bayesian learning to the problem of learning others' preferences in a negotiation setting and use meeting scheduling as their test domain. The setting they explore is quite different to the one we focus on this thesis however. The agents are assumed to be cooperative, and negotiate by truthfully revealing their preferences for different negotiation outcomes. This gives the Bayesian learning algorithm correctly labeled training data to work with. In our setting the agents have their own interested as well as cooperative tendencies, and there are no labeled training examples.

Buffett and Spencer [7] look at preference learning in a domain where agents negotiate over subsets of a set of objects. It is a property of the domain whether an agent prefers to have a subset or a superset of a given set of items. Buffet and Spencer assume that an agent values its offer at time $t+1$ strictly less than the offer it made at time t. Utility for objects is also assumed to be independent. Buffett and Spencer consider a number of predefined classes of preferences relations, with known priors. For each offer another agent makes, the learning agent can evaluate whether the offer is consistent with the preference relations in each of the classes. After receiving some number of offers, Buffett and Spencer use Bayesian reasoning to update the learning agent's belief about the other agent's preference class. The main difference between our work and that of Buffett and Spencer is that we consider how the agents feed back what they have learnt into the negotiation process. A potential problem with doing this in the model that Buffett and Spencer use is that the assumption of strictly decreasing utility could be strongly violated. In their utility model, utility can be a function of negotiation time, thus if an agent has a good model of another agent and can propose a subset the other agent is likely to agree to, this would be a good strategy, but in self-play, this would violate the learning assumption.

Wallace and Freuder [50] study a meeting scheduling problem where the meetings can be in different cities and the agents are constrained by travel time. The agents are assumed to be completely cooperative. The agents will accept a proposal if it is feasible according to their constraints. Wallace and Freuder show that by recording the proposals agents reject and by reasoning about the causes of the rejection, the agents can model each other's constraints. The experimental evaluation, focuses on the trade-off between increased efficiency and privacy loss.

Coehorn and Jennings [9] focus on negotiation in the domain where a buyer and seller negotiate prices for a set of issues simultaneously. The agents have a value for each issue, and a weight function over the issues which comprises their utility function. Coehorn and Jennings use Kernel Density Estimation to try and learn the weight function (this involves an offline training step, as well as an online learning component). The learned information is used in the trade-off step of Faratin's [19] negotiation algorithm. Coehorn and Jennings run experiments using a particular negotiation strategy, with the buyer learning and the seller having full information. These experiments showed a higher product of agent utilities and shorter negotiation time with learning than without. Restificar and Haddawy also look at the buyer/seller negotiation domain, but for single issues. Their focus however is on learning an agent's negotiation preferences in the context of a

Zeuthen style decision making model.

Lau et al. [31] have used a genetic algorithms approach to adapt to the opponent in the buyer/seller price negotiation domain. The genetic algorithm is used to gradually bring the offers made by one agent closer to the offers made by the other.

Much work on learning in negotiation has focused on identifying particular characteristics of agents. For instance, Talman et al. [49], try to identify whether agents are "helpful" and "reliable" in the game *Colored Trails* [26] (this game involves negotiation). When agents adapt their game play according to what they have learnt, this outperforms not adapting. A significant amount of work has been done in the area of *trust and reputation*, Ramchurn et al. [42] is a useful introduction to the area.

Our approach differs from related work on learning in agreement problems in a number of ways. Our reward-based learning approach, for example, differs in terms of method. Our work is also distinguished by the challenges it seeks to address. Much of the work on learning in negotiation, has looked at domains, like the price negotiation domain, where the private information is less complex than it is in the Multiagent Meeting Scheduling problem for example. Work on learning models of other agents when the private information is more complex, has tended to assume the agents are completely cooperative (or use a restrictive negotiation protocol), thereby supplying the learning algorithm with many correctly labeled training examples.

10.4 Models of Semi-Cooperativeness

Our approach to representing semi-cooperative agents involves discounting the agents' preferences for an option according to how long it takes for the agents to agree to that option. Other models of semi-cooperativeness have been considered in the agents literature, e.g., Gal et al. [23] looked at modeling an agent's utility as a function of self-interest and others' interest. This is a good choice when the preferences of the agents are public knowledge, but in our problem, agent preferences are privately owned. Manisterski, Katz & Kraus [34] define semi-cooperativeness, not according to utility functions, but use other properties, e.g., they require a Pareto Optimal outcome when identical agents negotiate. In Chapter 3, we show that in EIMAPP, when agents care about negotiation time, and have privately owned preferences, Pareto Optimality is not a suitable goal. Zhang and Lesser [54] also base their definition on agent properties, in particular they define an agent that is truthful and collaborative, but *not willing to voluntarily sacrifice its own interest* for others, as semi-cooperative.

The utility functions we use to represent semi-cooperativeness are similar those used by Azoulay-Schwartz and Kraus [3] to represent cost in a data allocation domain (discussed earlier in this chapter). One of the utility functions they study is of the form $U(\text{alloc}, t) = (1-p)^t U(\text{alloc}, 0) - t.C$ where $C > 0$ and $0 \geq p < 1$. This is essentially a combination of the two different round modified

utility functions we use in this thesis. In contrast however, we do not use "p" to just represent domain related costs, but rather each agent's level of cooperativeness. We note that despite the similarity in utility functions, our solution approach is very different to that of Azoulay-Schwartz and Kraus, for the reasons discussed earlier in the chapter.

We have shown that our representation of semi-cooperativeness can lead to efficient negotiation that satisfies agent preferences when agents learn to estimate the rounds in which the other agents will offer options.

10.5 Learning User Preferences

In the context of Multiagent Agreement Problems there has been a significant amount of work on learning user preferences about *when meetings are scheduled*. We describe some of that work in this section.

Mitchell *et al.* [35] present CAP, a learning apprentice designed to assist a user in updating his/her calendar. Each time the user adds a meeting (through CAP's structured interface); CAP suggests a duration, location, date, and start time for the meeting. CAP's suggestions are either accepted or corrected by the user and in this way training data is continually created. CAP learns decision trees for each of the target concepts. These decision trees are then used to make the suggestions to the user. Learning to predict the exact start time of a meeting appears to be quite a hard problem, and CAP achieves an average accuracy of just over 31% (based on data from two users). Using the same data Blum [4] was able to achieve an improved accuracy of almost 60%, using the Winnow algorithm.

Oh and Smith [38] take a statistical approach to learning a user's time-of-day preferences. One nice feature of their approach is that the learning occurs through the agent passively observing the user scheduling meetings with other participants. In order to interpret how the user's actions relate to the user's preferences, Oh and Smith make a number of assumptions. In particular, they assume that all parties in the system are aware of the organization's hierarchy, and that each one of these parties uses a specific negotiation strategy that gives priority treatment to the preferences of people higher in the hierarchy. When this assumption holds, Oh and Smith show that their approach is able to learn time preferences with very good accuracy. In [39], Oh and Smith extend their approach to learn more complex preferences. For example, they focus on learning the relative importance of meetings in order to be able to determine which meetings the user considers suitable for cancellation to accommodate more important meetings.

Lin and Hsu [32] take a hierarchical decision tree approach to learning a user's scheduling criteria e.g. the times of day the user prefers and restrictions on how different activity types can be ordered throughout the day. The learner is provided with training data that sets out the scheduling criteria for different activities. Lin and Hsu's approach has the potential to represent a very rich class of

user preferences, however, it requires very detailed training data that can only be provided by the user. Lin and Hsu state that for some criteria more than 1000 training examples would be needed.

Kozierok and Maes [29] have designed a software agent with an interface through which users can schedule meetings with others. The agent stores in its memory a situation, action pair for every action taken by the user. This memory is used to suggest the appropriate action to the user as the need arises. Reinforcement learning techniques are also applied to improve the model when the wrong action is suggested. The approach taken by Kozierok and Maes essentially rolls the learning of preferences into the problem of learning how to act like the user.

The approaches described have not explicitly dealt with learning preferences that involve dependencies, some also require training data that may be costly or difficult to obtain. Our approach addresses these two points.

We show that using only very limited training data, we can effectively learn option-based preferences as well as assignment dependent preferences. In the context of meeting scheduling, this means we can learn simple time-of-day preferences as well as preferences like back-to-back.

10.6 Summary

In this chapter we summarized some of the related work on agreement problems. We focused in particular on learning approaches and work in domains similar to our experimental domains. In the next chapter we revisit the contributions of this thesis and discuss future work.

Chapter 11

Contributions and Future Directions

11.1 Contributions

In this thesis, we looked at the problem of how agents should negotiate in agreement problems, with a focus on how agents can improve their negotiation through learning. In particular, we addressed the questions:

- How can an agent learn from seeing the results of negotiations?
- How can an agent model properties of other agents from negotiation histories?
- How can an agent learn to improve its negotiation when negotiating in a system where the behavior of other agents can vary greatly?
- In a semi-cooperative agent setting, how can agents learn to increase their utility from negotiation?

As a result of studying these questions, the major contributions of this thesis include:

- Algorithm that allows an agent to effectively learn its own user's option preferences. In particular our algorithm can learn option preferences that depend on the user's/agent's state, not just on the option being assigned.
- A domain independent approach for building abstractions of the properties of other agents from negotiation histories. And and approach to use these models of other agents to improve negotiation performance.

- A method for strategy selection when an agent is faced with the problem of deciding what negotiation strategy to use with another agent of unknown properties. Our approach provides an agent with a small set of diverse strategies that the agent learns to choose between strategies using experts algorithms.
- A definition of semi-cooperative agents that effectively captures the conflict between the reward agents receive from satisfying their preferences over the options, and their desire to negotiate efficiently.
- An approach for semi-cooperative agents to learn online to increase their utility. Our approach includes a method for predicting, from negotiation histories, the round in which other agents will offer different options. We provide two algorithms for exploiting this learned information, which directly attempt to increase an agent's utility according to its semi-cooperative utility function.
- An experimental evaluation in two real-world inspired domains demonstrating that Semi-Cooperative agents can effectively use one of our approaches, the SC-Utility-Estimate-Ordered-Learner, to learn online to increase their utility, in a manner that benefits all the agents involved.

11.2 Future Directions

This thesis opens up a number of directions for future research, a few of which we discuss here.

11.2.1 Learning in the Presence of More Complex Preference Models

Our experiments in this thesis focused on a relatively simple models of agent preferences. An interesting extension of our work is to explore the approaches we have introduced in the presence of more complex models of agent preferences. For instance, we could consider the case where an agent's preference for assigning a variable to an option is based on features of the variable in addition to the option. We could also look at preference functions that contain option dependencies as we did in Chapter 4. We believe our expert's based approach and our Semi-Cooperative learning approaches are particularly suitable for handling agents with complex preference models. The expert's based approach has the advantage that is does not attempt to learn a model of the other agents in order to improve its performance, rather it models the rewards the agent receives when using different strategies. As such, this approach will continue to perform well as long as there is some consistency in how well different strategies perform with different agents. The advantage of our Semi-Cooperative Learning approach is that we can encode any number of relevant features about an agent's offer of an option when we record it as training data for the regression algorithm. We can then use standard feature selection, and regression algorithms to produce a function that

will take a new variable, an option, the current option assignments, and any other relevant features, and map it to a predicted round.

11.2.2 Learning the γ Values of Other Agents

A natural question raised by our research on SC-agents, is whether or not in addition to predicting the rounds of offers, agents can estimate each other's γ values. Since an agent's γ value relates to how cooperative the agent is, learning to estimate an agent's γ value is similar to the concept of *reputation learning*(e.g., [40]). Learning another agent's γ value is a very difficult problem in the general case. As we discussed in Chapter 5, the function that generates our training data (negotiation histories) cannot in general be reconstructed from observing its output. Intuitively the problem is how can we separate the effects of an agent's γ value from the possible effects of its preferences or constraints? As such learning to estimate the γ values of other agents is most realistic in domains where agents don't have many constraints and where they have good apriori models of each other's preferences (due to the structure of the domain e.g., in a buyer/seller domain we know one agent prefers low prices and the other high prices).

11.2.3 Experts Algorithms and SC-agents

In our experiments with SC-agents we found that the SC-Utility-Estimate-Ordered-Learner had the best performance in most cases. However, our analysis showed that when we have perfect estimates (or very good estimates) the SC-Strategic-Ordered-Learner can perform better in some cases. In order to take advantage of such situations it is worth exploring the use of the experts approaches we introduced in Chapter 6 to choose between different learning strategies. In particular, a worthwhile direction for future research is to apply the EEE algorithm to the problem of selecting between the SC-Utility-Estimate-Ordered-Learner and the SC-Strategic-Ordered-Learner to evaluate the types of domains this approach would be useful in.

11.2.4 Implementing our Approaches for the Real-World

To implement our approaches in a real-world system e.g., in a system for scheduling meetings for computer users, additional work is required, including:

- Methods for agents to learn more complex models of *their own users' preferences*
- Ability for the user to change their γ value according to who they are negotiating with. This is something we believe our approach can readily extend to. The only problem it introduces

is that it adds some noise to the training data, when multiple learning agents schedule multi-person meetings.

- User studies need to be done on how an online learning agent can most effectively build trust with its user, receive and incorporate feedback from its user and communicate its decisions and models to its user.

Appendix A

Appendix

A.1 Simulation System

The simulation system has been designed to allow for the easy integration of new agreement domains. Agents exchange offers of *options*, regarding *variables* through a message exchange interface. Each agent has a negotiator, which takes care of deciding which options to offer. The different negotiation strategies and domains are represented through inheritance. We also define an interface for an Evaluator. Each agent has a set of evaluators registered with it, which are called on periodically to asses the agent's performance. The results are then written out to file. The system supports multiple trials of each experiment.

Our system provides post processing scripts to produce summary statistics and graphs from the output files. In this section we describe some of the specifics of the simulation system for the different agreement domains.

A.1.1 General Agreement Problems

In the general agreement problem there are no restrictions on the options that can be assigned to issues. As such, we only need to reset the agent's assignments at the beginning of each new trial.

Input Parameters

The input to an experiment includes:

- number of agents
- agent strategies
- agent preferences
- number of trials
- ...

Output Files

The format of the output files is consistent across all the domains. There is a separate file for each agent, for each negotiation strategy that is being evaluated for that agent, and for each evaluation metric. For example, there would be a file for Agent1Strategy1Metric1 and a different file for Agent1Strategy1Metric2 and so forth. One line of each of these files contains the evaluation according to the metric taken at each of the evaluation intervals in a trial. Different lines in the file correspond to different trials. For the learning strategies, the agent's learned models are also recorded.

A.1.2 Reset Intervals

Our simulation system features the concept of *reset intervals*. Each agent has an (i) initial state, defined by its initial variable-to-option assignments, (ii) a current state, and (iii) previous states that have been saved. A *reset interval* is the number of variables an agent assigns before its current state is saved with the previous states, and reset to the initial state. Consider the Multiagent Meeting Scheduling problem. The initial state of the agent is the meetings it has every week - like the agent's classes. Each week the agent schedules some new meetings, using up some of its scheduling options e.g., Monday 9am, Tuesday 3pm, Thursday 4pm, etc. After a certain number (the *reset interval* number) of meetings have been scheduled, we say the week is over. The agent's calendar is reset to the initial state, and the options, Monday 9am, Tuesday 3pm, Thursday 4pm, are once again available for scheduling. In our simulation system, we evaluate the *reset-interval* according to the number of variables assigned in the entire system (not on an individual, per-agent basis).

A.1.3 Meeting Scheduling Problems

In the meeting scheduling problem, the initial meetings (i.e., the meetings the agents have every week), can have a significant impact on the agents' behavior. As such, we provide the facility to

specify the initial meetings in an input file, in addition to the capability to enter parameters for their random generation.

Input Parameters

- Number of Agents
- Agent Strategies
- Agent preferences - randomly generated or specified in a file
- Weekly meetings - some number of weekly meetings randomly generated according to parameters or in a file (*initial state*)
- Blocked times - some number of blocked times per agent, randomly selected or in a file
- Week interval - number of meetings scheduled at which calendars will be reset to the base weekly calendars (*reset interval*)
- Evaluation interval - number of meetings after which the agents will be evaluated (the agent should at least be evaluated every reset interval).

Example Input Files

The experiment file appears as follows:

```
experimentName:Test1
noMeetingsToSchedAWeek:10
slotsInDay:8
noAgents:4
noTrials:2
evalInterval:10
learning:true
exploreProbStart:1.0
exploreProbMultiplier:0.93
negotiators:UtilEstOrd,UtilEstOrd,UtilEstOrd,UtilEstOrd;UtilStratOrd,PrefOrd,PrefOrd,PrefOrd;
initialMeetingsFile:5pInitM.txt
maxSizeInit:4
spreadInit:uniform
blockedSlots:None
numberBlocked:3,3,3,3
toSchedMeetingsFile:5pGenNew.txt
```

```
noMeetingsToSched:None
maxSizeToSched:None
spreadToSched:None
prefsFile:None
maxPref:40
prefRanges:20,20,20,20
prefStyle:random
utilsFile:None
gammas:0.6,0.6,0.6,0.6
epsilons:useRule
```

Note that the system can be instructed to evaluate different sets of negotiation strategies on the same input data. The "negotiators" line specifies different sets of negotiation strategies separated by semi-colons. In each set, the agents in the system are assigned a negotiation strategy in alpha-numeric order.

An extract from the weekly base meetings file is as follows:

```
---
ID: m0
Decision Makers: Agent3 Agent1
Initiator: Agent1
Time: 0_0_4
---
ID: m1
Decision Makers: Agent4 Agent3
Initiator: Agent4
Time: 0_1_1
---
ID: m2
Decision Makers: Agent3 Agent1
Initiator: Agent1
Time: 0_1_5
---
ID: m3
Decision Makers: Agent1 Agent3
Initiator: Agent1
Time: 0_3_2
---
ID: m4
Decision Makers: Agent5 Agent1 Agent4 Agent2
Initiator: Agent4
```

```
Time: 0_3_4
```

The "Time" field is interpreted as week_day_hour. The meetings to schedule file is similar, but the "Time" field is missing.

A.1.4 Role Assignment Domain

We use the same simulation system for the Role Assignment domain.

Input Parameters

The input parameters are similar to the meeting scheduling parameters, although some of the labels are different. Tasks for assignment can be supplied in a file or generated randomly by the simulation according to preferences given in the input file.

Example Input Files

Example task experiment file:

```
experimentName:TaskTest1
noTasksToScheduleBeforeReset:6
noAgents:4
noTrials:2
evalInterval:6
exploreProbStart:1.0
exploreProbMultiplier:0.93
negotiators:PrefOrd,PrefOrd,PrefOrd,PrefOrd;UtilEstOrd,UtilEstOrd,UtilEstOrd,UtilEstOrd;
newTasks:5pTasks.txt
numTaskTypes:3
maxPref:10
agentDetails:agentTaskDetails.txt
utilsFile:None
gammas:0.8,0.8,0.8,0.8,0.8
epsilons:0.0,0.0,0.0,0.0
```

Example agent input file:

```
---
ID:Agent1
Prefs:(type1 10);(type2 1);(type3 4)
---
ID:Agent2
Prefs:(type1 5);(type2 5);(type3 5)
---
ID:Agent3
Prefs:(type1 2);(type2 10);(type3 4)
---
ID:Agent4
Prefs:(type1 10);(type2 1);(type3 4)
```

Example new tasks file:

```
---
ID:t1
Decision Makers:Agent1 Agent2
Initiator:Agent1
Roles:{role1,type1}#{role2,type2}
---
ID:t2
Decision Makers:Agent3 Agent4
Initiator:Agent4
Roles:{role1,type2}#{role2,type3}
---
ID: t3
Decision Makers:Agent1 Agent2
Initiator:Agent2
Roles:{role1,type1}#{role2,type2}
---
ID: t4
Decision Makers:Agent3 Agent4
Initiator:Agent3
Roles:{role1,type2}#{role2,type3}
---
ID: t5
Decision Makers:Agent1 Agent2
Initiator:Agent1
Roles:{role1,type1}#{role2,type2}
```

A.2 Preferences Input

Preferences used in the experiment described in 9.1.

```
---
ID:Agent1
Prefs:(type1 20);(type2 20);(type3 19);(type4 18);(type5 18);(type6 10);
(type7 10);(type8 9);(type9 9);type10 5);(type11 5);(type12 3);
(type13 4);(type14 20);(type15 17);(type16 2);(type17 2);(type18 2);
(type19 19);(type20 3)
---
ID:Agent2
Prefs:(type1 20);(type2 20);(type3 19);(type4 18);(type5 18);
(type6 10);(type7 10);(type8 9);(type9 9);(type10 5);(type11 5);
(type12 3);(type13 4);(type14 20);(type15 17);(type16 2);
(type17 2);(type18 2);(type19 19);(type20 3)
---
ID:Agent3
Prefs:(type1 20);(type2 20);(type3 19);(type4 18);(type5 9);
(type6 10);(type7 10);(type8 9);(type9 9);(type10 5);(type11 5);
(type12 3);(type3 4);(type14 20);(type15 17);(type16 2);(type17 2);
(type18 2);(type19 19);(type20 3)
---
ID:Agent4
Prefs:(type1 2);(type2 20);(type3 10);(type4 18);(type5 18);
(type6 3);(type7 10);(type8 10);(type9 20);(type10 19);
(type11 19);(type12 5);(type13 5);(type14 10);(type15 10);
(type16 17);(type17 5);(type18 7);(type19 4);(type20 5)
---
ID:Agent5
Prefs:(type1 2);(type2 20);(type3 10);(type4 18);(type5 18);
(type6 3);(type7 10);(type8 10);(type9 20);(type10 19);
(type11 19);(type12 5);(type13 5);(type14 10);(type15 10);
(type16 17);(type17 5);(type18 7);(type19 4);(type20 5)
---
ID:Agent6
Prefs:(type1 2);(type2 20);(type3 10);(type4 18);(type5 4);
(type6 3);(type7 10);(type8 10);(type9 20);(type10 19);
(type11 19);(type12 5);(type13 5);(type14 10);
(type15 10);(type16 17);(type17 5);(type18 7);
(type19 4);(type20 5)
```

```
---
ID:Agent7
Prefs:(type1 10);(type2 5);(type3 3);(type4 15);(type5 5);
(type6 20);(type7 4);(type8 20);(type9 3);(type10 4);(type11 20);
(type12 3);(type13 10);(type14 5);(type15 10);(type16 4);
(type17 5);(type18 19);(type19 19);(type20 15)
---
ID:Agent8
Prefs:(type1 10);(type2 5);(type3 3);(type4 15);(type5 5);
(type6 20);(type7 4);(type8 20);(type9 3);(type10 4);(type11 20);
(type12 3);(type13 10);(type14 5);(type15 10);(type16 4);
(type17 5);(type18 19);(type19 19);(type20 15)
---
ID:Agent9
Prefs:(type1 10);(type2 5);(type3 3);(type4 15);(type5 5);
(type6 20);(type7 4);(type8 20);(type9 3);(type10 4);
(type11 20);(type12 3);(type13 10);(type14 10);
(type15 10);(type16 4);(type17 5);(type18 19);
(type19 19);(type20 15)
---
ID:Agent10
Prefs:(type1 10);(type2 5);(type3 3);(type4 15);(type5 5);
(type6 20);(type7 4);(type8 20);(type9 3);(type10 20);
(type11 20);(type12 20);(type13 10);(type14 5);
(type15 5);(type16 4);(type17 5);(type18 15);(type19 19);
(type20 15)
---
ID:Agent11
Prefs:(type1 20);(type2 20);(type3 15);(type4 15);(type5 15);
(type6 5);(type7 20);(type8 20);(type9 5);(type10 5);
(type11 14);(type12 14);(type13 19);(type14 18);(type15 14);
(type16 9);(type17 9);(type18 4);(type19 4);(type20 16)
---
ID:Agent12
Prefs:(type1 20);(type2 20);(type3 15);(type4 15);
(type5 15);(type6 5);(type7 20);(type8 20);(type9 5);(type10 5);
(type11 14);(type12 14);(type13 19);(type14 18);
(type15 14);(type16 9);(type17 9);(type18 4);(type19 4);(type20 16)
---
ID:Agent13
Prefs:(type1 20);(type2 20);(type3 15);(type4 15);(type5 15);
(type6 5);(type7 20);(type8 20);(type9 5);(type10 5);
```

```
(type11 14);(type12 14);(type13 5);(type14 15);
(type15 14);(type16 9);(type17 9);(type18 4);(type19 4);(type20 16)
---
ID:Agent14
Prefs:(type1 15);(type2 9);(type3 3);(type4 15);(type5 5);
(type6 20);(type7 20);(type8 20);(type9 3);(type10 19);(type11 20);
(type12 3);(type13 10);(type14 10);(type15 5);(type16 4);
(type17 20);(type18 19);(type19 19);(type20 15)
---
ID:Agent15
Prefs:(type1 20);(type2 5);(type3 19);(type4 15);
(type5 15);(type6 20);(type7 4);(type8 20);(type9 14);
(type10 20);(type11 19);(type12 20);(type13 10);(type14 10);
(type15 5);(type16 19);(type17 5);(type18 15);(type19 19);(type20 15)
```

A.3 Reply-To-Initiator Protocol

Algorithm A.3.1 Initiator Algorithm (variable: v, agent a)

Initialization: offer a set of options ${P_0}^a$ to $\{\mathcal{D}_v - a\}$, set mode(v) to *negotiation*.
while no option has been assigned to v **do**
  if a response has been received from all $\{\mathcal{D}_v - a\}$ in this round i, **then**

    let each agent's offer at round i, ${P_i}^{a'}$ be the union of the current off and all their previous offers

    if in *negotiation* mode, and $\bigcap_{a' \in \mathcal{D}_v} {P_i}^{a'} \neq \emptyset$ and one of these options is not now pending or assigned to another variable, **then**
      send a confirmation request for an option in the intersection to $\{\mathcal{D}_v - a\}$
      set mode(v) to *pending*
      break.

    if in *negotiation* mode, and there is no option that can be made pending above **then**
      break

    if in *pending* mode, and all $\{\mathcal{D}_v - a\}$ accepted the confirmation request **then**
      send a confirmation message and assign the option to v
      return

    if in *pending* mode, and one or more of $\{\mathcal{D}_v - a\}$ rejected the confirmation request **then**
      send a message to $\{\mathcal{D}_v - a\}$ canceling the confirmation request
      set mode(v) to *negotiation*
      break
  else
    wait for more responses
  offer a set of options ${P_{i+1}}^a$ to $\{\mathcal{D}_v - a\}$

Algorithm A.3.2 Non-Initiator Algorithm (variable: v, agent: a)

if receive a message about v **then**

if if this is the first offer **then**
set mode(v) to *negotiation*

if in *negotiation* mode, and message is not a confirmation request **then**
make an offer to initiator(v)

if in *negotiation* mode and message is a confirmation request, set mode(v) to *pending* **then**

if the option requested is not pending for another variable, or now assigned to another variable **then**
send a message to the initiator accepting the option
else
send a message rejecting the request

if in *pending mode*, and this message cancels the confirmation request **then**
set mode(v) to *negotiation*
make an offer to initiator(v)

if in *pending* mode, and this message is a final confirmation **then**
make the assignment

Bibliography

[1] Kenneth J. Arrow. *Social Choice and Individual Values, Second edition (Cowles Foundation Monographs Series)*. Yale University Press, September 1970.

[2] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual FOCS*, 1995.

[3] Rina Azoulay-Schwartz and Sarit Kraus. Negotiation on data allocation in multi-agent environments. *Autonomous Agents and Multi-Agent Systems*, 5(2):123–172, 2002.

[4] Avrim Blum. Empirical support for winnow and weighted-majority based algorithms: results on a calendar scheduling domain. In *Proceedings of the 12th International Conference on Machine Learning*, 1995.

[5] Michael Bowling. Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems 17*, pages pages 209–216, 2005.

[6] Michael Bowling, Brett Browning, and Manuela Veloso. Plays as effective multiagent plans enabling opponent-adaptive play selection. In *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'04)*, 2004.

[7] Scott Buffett and Bruce Spencer. Learning opponents' preferences in multi-object automated negotiation. In *ICEC '05: Proceedings of the 7th international conference on Electronic commerce*, pages 300–305, New York, NY, USA, 2005. ACM Press.

[8] H. H. Bui, D. Kieronska, and S. Venkatesh. Learning other agents' preferences in multiagent negotiation. In *Proceedings of AAAI*, 1996.

[9] Robert M. Coehoorn and Nicholas R. Jennings. Learning on opponent's preferences to make effective multi-issue negotiation trade-offs. In *ICEC '04: Proceedings of the 6th international conference on Electronic commerce*, pages 59–68, New York, NY, USA, 2004. ACM Press.

[10] Elisabeth Crawford and Manuela Veloso. Opportunities for learning in multi-agent meeting scheduling. In *Proceedings of the AAAI Symposium on Artificial Multiagent Learning*, 2004.

[11] Elisabeth Crawford and Manuela Veloso. Learning dynamic preferences in multi-agent meeting scheduling. In *Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT)*, 2005.

[12] Elisabeth Crawford and Manuela Veloso. Learning dynamic time preferences in multi-agent meeting scheduling. Technical Report CMU-CS-05-153, Computer Science Dept., Carnegie Mellon University, 2005.

[13] Elisabeth Crawford and Manuela Veloso. Learning to select negotiation strategies in multi-agent meeting scheduling. In *Proceedings of 12th Portugese Conference on Artificial Intelligence, Springer, LNCS*, 2005.

[14] Elisabeth Crawford and Manuela Veloso. Mechanism design for multi-agent meeting scheduling. *Web Intelligence and Agent Systems*, 4(2), 2006.

[15] Daniela de Farias and Nimrod Megiddo. Exploration-exploitation tradeoffs for experts algorithms in reactive environments. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 409–416. MIT Press, Cambridge, MA, 2005.

[16] Daniela Pucci de Farias and Nimrod Megiddo. Combining expert advice in reactive environments, 2004.

[17] Daniela Pucci de Farias and Nimrod Megiddo. How to combine expert (and novice) advice when actions impact the environment? In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[18] Ulle Endriss. Monotonic concession protocols for multilateral negotiation. In *Proceedings of AAMAS*, 2006.

[19] P. Faratin, C. Sierra, and N. Jennings. Using similarity criteria to make issue tradeoffs in automated negotiations. *Artificial Intelligence*, 142(2):205–237, 2002.

[20] S. Shaheen Fatima, Michael Wooldridge, and Nicholas R. Jennings. Optimal negotiation strategies for agents with incomplete information. In *ATAL '01: Revised Papers from the 8th International Workshop on Intelligent Agents VIII*, pages 377–392, London, UK, 2002. Springer-Verlag.

[21] Yoav Freund, Robert Schapire, Yoram Singer, and Manfred Warmuth. Using and combining predictors that specialize. In *STOC*, 1997.

[22] Drew Fudenberg and Jean Tirole. *Game Theory*. MIT Press, 1991.

[23] Ya'akov Gal, Avi Pfeffer, Francesca Marzo, and Barbara J. Grosz. Learning social preferences in games. In *Proceedings of AAAI*, 2004.

[24] Leonardo Garrido and Katia Sycara. Multi-agent meeting scheduling: Preliminary experimental results. In *Proceedings of the First International Conference on Multi-Agent Systems*, 1995.

[25] John Geanakoplos. Three brief proofs of arrow?s impossibility theorem. *Economic Theory*, 26(1):211–215, July 2005.

[26] Barbara Grosz, Sarit Kraus, Shavit Talman, Boaz Stossel, and Moti Havlin. The influence of social dependencies on decision-making: Initial investigations with a new game. In *Proceedings of AAMAS*, 2004.

[27] John C. Harsanyi. Approaches to the bargaining problem before and after the theory of games: A critical discussion of zeuthen's, hicks', and nash's theories. *Econometrica*, 24(2):144–157, April 1965.

[28] N. R. Jennings and A. J. Jackson. Agent based meeting scheduling: A design and implementation. *IEE Electronics Letters*, 31(5):350–352, 1995.

[29] Robyn Kozierok and Pattie Maes. A learning interface agent for scheduling meetings. In *Proceedings of the 1st international conference on Intelligent user interfaces*, 1993.

[30] Sarit Kraus. Automated negotiation and decision making in multiagent environments. *Mutliagents systems and applications*, pages 150–172, 2001.

[31] Raymond Y. K. Lau, Maolin Tang, On Wong, Stephen W. Milliner, and Yi-Ping Phoebe Chen. An evolutionary learning approach for adaptive negotiation agents: Research articles. *Int. J. Intell. Syst.*, 21(1):41–72, 2006.

[32] S Lin and J Hsu. Learning user's scheduling criteria in a personal calendar agent, 2000.

[33] Nick Littlestone and Manfred K. Warmuth. The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261, 1994.

[34] Efrat Manisterski, Ron Katz, and Sarit Kraus. Providing a recommended trading agent to a population: a novel approach. In *Proceedings of IJCAI*, 2007.

[35] Tom M. Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7):80–91, 1994.

[36] Pragnesh Jay Modi and Manuela Veloso. Bumping strategies for the multiagent agreement problem. In *Proceedings of The Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS'05)*, 2005.

[37] Pragnesh Jay Modi and Manuela Veloso. Bumping strategies for the multiagent agreement problem. In *Proceedings of AAMAS*, 2005.

[38] Jean Oh and Stephen F. Smith. Learning calendar scheduling preferences in hierarchical organizations. In *6th International Workshop on Preferences and Soft Constraints*, 2004.

[39] Jean Oh and Stephen F. Smith. Calendar assistants that learn preferences. In *Persistent Assistants: Living and Working with AI (AAAI Spring Symposium)*, 2005.

[40] Josep M. Pujol, Ramon Sangüesa, and Jordi Delgado. Extracting reputation in multi agent systems by means of social network topology. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 467–474, New York, NY, USA, 2002. ACM.

[41] Iyad Rahwan, Sarvapali D. Ramchurn, Nicholas R. Jennings, Peter Mcburney, Simon Parsons, and Liz Sonenberg. Argumentation-based negotiation. *Knowl. Eng. Rev.*, 18(4):343–375, 2003.

[42] Sarvapali D. Ramchurn, Nicholas R. Jennings, Carles Sierra, and Lluis Godo. Devising a trust model for multi-agent interactions using confidence and reputation. *Applied Artificial Intelligence*, 18(9-10):833–852, 2004.

[43] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of encounter: designing conventions for automated negotiation among computers*. MIT Press, Cambridge, MA, USA, 1994.

[44] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, January 1982.

[45] Sabyasachi Saha and Sandip Sen. An efficient protocol for negotiation over multiple indivisible resources. In *Twelth International Conference on Artificial Intelligence, IJCAI*, 2007.

[46] Sandip Sen and Edmund Durfee. A formal study of distributed meeting scheduling. *Group Decision and Negotiation*, 7:265–289, 1998.

[47] Toramatsu Shintani, T. Ito, and Katia Sycara. Multiple negotiations among agents for a distributed meeting scheduler. In *Proceedings of the Fourth International Conference on MultiAgent Systems*, pages 435 – 436, July 2000.

[48] Electoral Reform Society.

[49] Shavit Talman, Meirav Hadad, Ya'akov Gal, and Sarit Kraus. Adapting to agents' personalities in negotiation. In *AAMAS '05: Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 383–389, New York, NY, USA, 2005. ACM Press.

[50] Richard J. Wallace and Eugene C. Freuder. Constraint-based reasoning and privacy/efficiency tradeoffs in multi-agent problem solving. *Artif. Intell.*, 161(1-2):209–227, 2005.

[51] Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *Knowledge and Data Engineering*, 10(5):673–685, 1998.

[52] Dajun Zeng and Katia Sycara. Bayesian learning in negotiation. *Int. J. Hum.-Comput. Stud.*, 48(1):125–141, 1998.

[53] F. Zeuthen. *Problems of Monopoly and Economic Warfare*. Routledge and Sons, London, UK, 1930.

[54] Xiaqin Zhang and Victor Lesser. Meta-level coordination for solving negotiation chains in semi-cooperative multi-agent systems. In *Proceedings of AAMAS*, 2007.