An Experts Approach to Strategy Selection in Multiagent Meeting Scheduling

Elisabeth Crawford and Manuela Veloso

Computer Science Department, Carnegie Mellon University,Pittsburgh PA 15213, USA

Abstract. In the multiagent meeting scheduling problem, agents negotiate with each other on behalf of their users to schedule meetings. While a number of negotiation approaches have been proposed for scheduling meetings, it is not well understood how agents can negotiate strategically in order to maximize their users' utility. To negotiate strategically, agents need to learn to pick good strategies for negotiating with other agents.
In this paper, we show how agents can learn online to negotiate strategically in order to better satisfy their users' preferences. We outline the applicability of experts algorithms to the problem of learning to select negotiation strategies. In particular, we show how two different experts approaches, plays [1] and Exploration-Exploitation Experts (EEE) [2], can be adapted to the task. We show experimentally the effectiveness of our approach for learning to negotiation strategically.

1 Introduction

Meeting scheduling can be a very time consuming task for computer users. It often requires the exchange of many email messages and the rescheduling of existing meetings, in order to accommodate new ones. Personalized meeting scheduling agents, with the ability to make decisions on behalf of their users, have been proposed as a way to automate meeting scheduling, e.g., [3, 4].

In this work, we consider each agent to be in control of its own calendar and of the information it reveals about its calendar and preferences. This differs to the set up in Microsoft Outlook where each user in the organization is required to make his/her calendar public. The main disadvantage of public calendars is that they violate users' privacy.[1] Negotiation is one way in which multiple agents can reach agreements about meeting times without having to make their calendars public. As such, a number of researchers have proposed negotiation protocols for multiagent meeting scheduling, e.g., [3, 4, 6].

In typical protocols, each meeting has an initiator. The initiator agent proposes meeting times, and collects all replies and counter proposals from the other agents. Consider the simplified protocol in Figure 1. In this context, we say that a *negotiation strategy* is an arbitrary function that decides what times to propose at each point in the process.

[1] For a more in depth discussion of the approach used in Microsoft Outlook see[5].

Simple Negotiation Protocol

while there is no intersection in proposals:

- the initiator proposes $x \geq 0$ times to the attendee agents
- each attendee agent proposes $y \geq 0$ times to the initiator

Fig. 1. A simplification of typical meeting negotiation protocols

While the use of negotiation approaches assumes that agents strive to reach agreements, users nonetheless have private preferences that their agents should try to maximize. Different strategies are likely to work well in different situations. For instance, when an agent negotiates with an agent that has similar preferences and offers a lot of times, it may be best to also offer a lot of times and/or to quickly agree to the other agent's proposals by proposing the same times. However, not all agents have the same preferences, and not all agents are very cooperative. For example, some agents may only offer their favorite times initially and be slow to compromise their preferences in order to find an intersection. When negotiating with such an agent it is likely to be best to offer less times and agree to the agent's proposals more slowly, in order to ensure the time eventually chosen satisfies the user's preferences.

In this article, we propose an approach to negotiation where agents learn what strategies to use in different situations. We show that an agent can learn to choose good negotiation strategies online using *experts approaches*. By observing how well each of the strategies performs, the agent can learn to select good strategies for different situations.

There are a number of existing algorithms for the experts problem and its variants (e.g., the k-armed bandits problem)[7, 8]. In an experts problem, an agent must make sequential decisions, by choosing which of a set of experts it should take advice from at each decision point. After each decision, the agent receives some reward. The aim is for the agent to make choices that lead to good rewards. Algorithms for experts problems must trade-off exploration and exploitation. By viewing a set of diverse negotiation strategies as a set experts, an agent can use experts algorithms to learn how to select strategies. In Section 2, we discuss experts algorithms in more detail, and in Section 3, we describe their applicability to strategy selection in multiagent meeting scheduling.

In Section 4, we look at the static environment case, where an agent needs to learn how to negotiate in the presence of fixed strategy agents. We show how we can adapt the *playbook* approach for team plan selection in robot soccer [1], to the problem of learning to select negotiation strategies. In Section 5, we look at the more difficult case, where the environment is reactive. We show how the Exploration-Exploitation Experts (EEE) Algorithm [2], can be used to learn effectively when *other agents are also adapting*. In Section 6, we present an experimental evaluation of the different approaches.

In Section 7, we describe related work on negotiation for multiagent meeting scheduling. There is a substantial body of work on the multiagent meeting scheduling problem, however the issue of learning to negotiate *strategically* has not been addressed.

2 Experts Algorithms

2.1 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 article) 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. The 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.

2.2 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 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 [7], Hedge [8] and Exp3[2] [8]. 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.

2.3 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 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., [8, 10]), 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) [9] and the Exploration Exploitation Experts algorithm (EEE) [2] 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" [9]. 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

[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.

opponent actions. Stronger guarantees for SEA and EEE are given when some assumptions are made about the opponent [9].

3 Learning to Negotiate with Different Agents using Experts Approaches

3.1 Setting

In this article we focus on the problem of learning to negotiate with different agents. A meeting scheduling agent is faced with a number of different negotiation situations or environments. For instance, 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. It is largely the other agent's strategy, preferences, and calendar, that will determine the success of our learning agent's negotiation strategies.

3.2 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 scheduling a particular meeting, given an agent's limited knowledge of the situation (e.g., its limited knowledge of the calendars, preferences and strategies of other agents) is not trivial. Thirdly, the learning needs to occur online, so we face an exploration/exploitation trade-off.

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 calendars and preferences, and even the type of meeting being scheduled.

One approach, would be to try and model the relevant factors. An agent could try and identify different aspects, including, the strategy, preferences, and calendar 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, we propose that 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 Section 6 we experimentally demonstrate their usefulness in this domain.

3.3 An Experts Approach for Learning to Negotiate with Different Agents

We propose that in order to learn which strategy to use with a particular agent, say Agent B, the learning agent can use an experts algorithm. In order 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 schedules meetings 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 two person meetings. Suppose that our learning agent is initiating a large meeting. To do this, the learning agent negotiates separately with each agent involved in the meeting, i.e., it exchanges time proposals with each other agent. In this case, the learning agent will select a possibly different strategy for negotiation with each of the attendee agents according to the experts algorithm running for that agent. When the learning agent is an attendee of a large meeting, it has to negotiate with the meeting's initiator. In this case, the learning agent will select the negotiation strategy according to the experts algorithm it is running for the meeting initiator. We summarize our learning algorithm in Figure 2.

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 for the case of two-person meetings.

4 Playbook Approach to Strategy Selection

4.1 Introduction to Plays

In this section we show how an experts-based playbook approach [1] can be adapted to the problem of selecting strategies in multiagent meeting scheduling [11]. 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

Learning Algorithm Overview

Let the agent's name be A.

when a new meeting M arises:

- if A is the initiator of M:
  - for each agent $x \in attendees(M) - \{A\}$:
    - if A has not not negotiated with x before, initialize an experts algorithm for x
    - negotiate with x according to the experts algorithm for x until M is scheduled
  - for each agent $x \in attendees(M) - \{A\}$:
    - update the experts algorithm for x according to the reward received from scheduling M
- else:
  - if A has not negotiated with the initiator, i, of M before, initialize an experts algorithm for i
  - negotiate with i according to experts algorithm for i until M is scheduled
  - update the experts algorithm for i according to the reward received from scheduling M

Fig. 2. An overview of our learning algorithm.

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 [1].

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 [1].

4.2 Outline: Plays for Strategy Selection

The meeting negotiation problem has 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 meeting scheduling 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 meeting scheduling problem, we consider each agent a separate 'opponent'. At any one time a meeting scheduling agent may be negotiating with multiple 'opponent' agents, whereas in robot soccer only one opponent team is played at a time. In the meeting scheduling 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 meeting scheduling however, an agent's calendar changes as meetings are scheduled. As such, scheduling a meeting with one agent can affect how a meeting is scheduled with another agent. As we show in Section 6, 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 times (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 times (if any) an agent offers in each round. Furthermore, there is no restriction placed on what times the agent chooses, or how it makes the choice. Figure 3, shows an example strategy, Offer-k-b, that offers k new available times each round, and after b rounds starts trying to compromise according to the proposals it has received. If necessary, Offer-k-b will offer times 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.

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

[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.

Offer-k-b Negotiation Algorithm

1. **APPLICABILITY**: if the other agent is very-important and ($k < 20$ and $b > 5$) return false; else return true.
2. **OFFER-METHOD**: In any negotiation round offer a's k most preferred, available (not yet occupied by a meeting), un-offered times. If negotiation round $> b$ (i.e., b rounds have passed without an intersection in proposed times). Apply the simple compromiser sub-strategy which works as follows:
   - If a is an attendee of the meeting, but not the initiator, a searches for any times proposed by the initiator that a is available for, but has not offered. If one or more such times exist offer a offers its most preferred such time. Else, a offers the time proposed by the initiator that contains the meeting with the fewest participants.
   - If a is the initiator, a ranks all the times proposed by other agents according to the number of agents that have proposed that time. Out of all the times with the highest number of proposals if any of these times are available (for a), a offers its most preferred such time, otherwise a offers the time it has unavailable that contains the meeting with the fewest participants.
3. **ABANDON-METHOD**: if negotiation round > 50 return true.

Fig. 3. Description of the Offer-k-b negotiation algorithm.

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 meeting it was selected to schedule has been added to the agent's calendar, and
- any meetings that the learning agent is involved in that have been bumped have been rescheduled for new times.

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

4.3 Weight Adaptation and Strategy Selection for Negotiating with Different Agents

For each 'opponent' agent, the learning agent must learn which strategies to select. The learning algorithm [1] 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]

[4] For a more complete treatment we refer to the reader to [1].

Algorithms for selecting experts with no regret can be used to select plays [1]. Similarly to [1] 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* [12]. An expert is awake when it is applicable at a particular choice time, and asleep otherwise. Following the notation used in [1], 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 [1] combines elements of the Exp3 algorithm [8] (which handles the problem of unknown rewards) with the sleeping regret approach [12]. We describe the approach here, and use it to adapt playbook weights for each 'opponent' agent, and select the 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:

$$Pr(x_t = i) = \frac{a_i^t w_i^t}{\sum_j a_j^t w_j^t} \tag{1}$$

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

$$w_i^{t+1} = \hat{w}_i^t . N_i^t \tag{2}$$

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

$$\hat{w}_i^t = w_i^t (m_i^t)^{\frac{1}{Pr(x_t=i)}} \tag{3}$$

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:

$$N_i^t = \frac{\sum_j a_j^t w_j^{t-1}}{\sum_j a_j^t \hat{w}_j^t} \tag{4}$$

Plays Algorithm

- weights are initialized to 1 for all experts
- for each time step t:
  1. play/strategy is selected according to the probability distribution in equation 1
  2. weights are updated using equations 2, 3, and 4

Fig. 4. Plays Algorithm from [1]

The algorithm is summarized in Figure 4.

Instead of directly dealing with rewards the playbook approach assigns the multiplier 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 weight. We base the multiplier on a model of user utility. We let the utility a user derives from a negotiation strategy take into account three elements:

1. the user's preference for the time-of-day (tod) the new meeting is scheduled for: $val(tod)$.
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

$$\sum_{moved} val(tod_{new}) - \sum_{moved} val(tod_{old})$$

3. the number of negotiation rounds r required to schedule the new meeting and move any old meetings.

The user's utility function is parametrized by two constants α and β which specify the relative importance of time-of-day valuations and negotiation cost. Formally a user's utility for the outcome of a negotiation strategy is modeled as:

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

We use the user's utility function and highest time-of-day 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 1 that applies. Also note that if the negotiation strategy fails to schedule the new meeting, or to reschedule any bumped meetings, 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 meeting scheduled. Secondly, we wanted the number to evenly balance gains and losses.

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 [8], we are dealing with a *non-oblivious* adversary. As such, since the playbook approach builds on Exp3, the theoretical bounds are weaker.

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 1. The multiplier is given by the first row for which the entry in the first column evaluates to true

5 Exploration-Exploitation Experts Approach to Strategy Selection

5.1 Setting Definition

The Strategic Experts Algorithm (SEA) [9], as well as a generalized version, Exploration-Exploitation Experts (EEE) [2] 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.

5.2 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.

5.3 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. Our approach to learning to select meeting negotiation strategies, for the case where the other agents are also adapting, involves running the EEE algorithm for each agent the learning agent schedules meetings with. 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 meeting negotiation problem.

Let a *stage* denote the scheduling of one new meeting. 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 x, 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.

Figure 5 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 [2].

6 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. 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.

6.1 Communication Protocol

We have created a simulation environment consisting of a set of agents equipped with a common protocol for communicating about meetings. The protocol has three basic stages: a negotiation phase, in which agents exchange proposals,

EEE Algorithm

- R_s^x is the average reward a has achieved by using strategy s when negotiating with agent x.
- N_s^x is the number of phases in which a has followed s when negotiating with agent x.
- S_s^x is the number of times a has negotiated using s with agent x.
- i^x is a's phase count for agent x.

1. *Initialization*: $\forall x, s, R_s^x = N_s^x = S_s^x = 0$ and $i^x = 1$
2. *Explore/Exploit Decision*: Explore with probability p_i^x. If using the SEA restriction $p_i^x = 1/i^x$. With probability $1-p_i^x$ exploit. Let s be the strategy chosen.
3. *Use and Update*: Use s for the next n negotiations with agent x. If using the SEA variant, $n = N_s^x$. Then set $N_s^x = N_s^x + 1$, $S_s^x = S_s^x + n$. If we let $\hat{R}$ be the average payoff received over the n stages, then the update for R_s^x is given by the following formula:

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

4. $i^x = i^x + 1$, go to 2.

Fig. 5. EEE Algorithm, adapted from [2]

a pending stage, in which a time proposed by all the agents is agreed upon, and a confirmation stage, after which the meeting is entered into the agents' calendars. Support is also provided for *bumping* (canceling and rescheduling) meetings. There are a number of different types of messages that the agents exchange:

- meeting time proposals
- requests to bump meetings
- cancellation notices for meetings
- pending requests for times – when a meeting initiator finds an intersection in proposals, it sends a pending request for one of the times in the intersection to each of the participants.
- pending responses – when an attendee receives a pending request it responds with either:
  - a pending acceptance and marks the meeting as pending, or
  - a pending rejection (if the time is pending for another meeting, we require that the agent rejects the request).
- confirmation notices – sent out by the initiator when all attendees reply to a pending request with a pending acceptance.

Handling Bumping In the protocol outlined meetings can be bumped. 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

Availability Declarer Negotiation Algorithm

1. **APPLICABILITY**: if importance(other-agent) >= moderately-important return true.
2. **OFFER-RULE**:
   - In the first round a offers all its available times for the current week, in second round offers all its available times for the following week and so on, until all available times, up until the last possible time for the meeting, have been offered.
   - If negotiation round > 5, a applies the simple compromiser sub-strategy described in Figure 3.

   ABANDON: if negotiation round > 50 return true.

Fig. 6. Description of the Availability Declarer negotiation algorithm.

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.

6.2 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 article. The first strategy – Offer-k-b was previously described (see Figure 3.). This strategy is parametrized, and hence it covers a large number of distinct strategies. The second strategy we use is called Availability-Declarer (Figure 6.). This strategy can be very useful in practice, particularly in situations where 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.

6.3 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 in Section 4, i.e,

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

There exist more complicated models of user preferences. For instance, in [13] we discuss a model where the user's preferences over times can depend on the

state of the user's calendar. We chose to use a simpler model of user preferences for these experiments because it makes the interactions between different agents more understandable, enabling a clearer discussion of the results.

6.4 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 7 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 7 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 8 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 8 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,

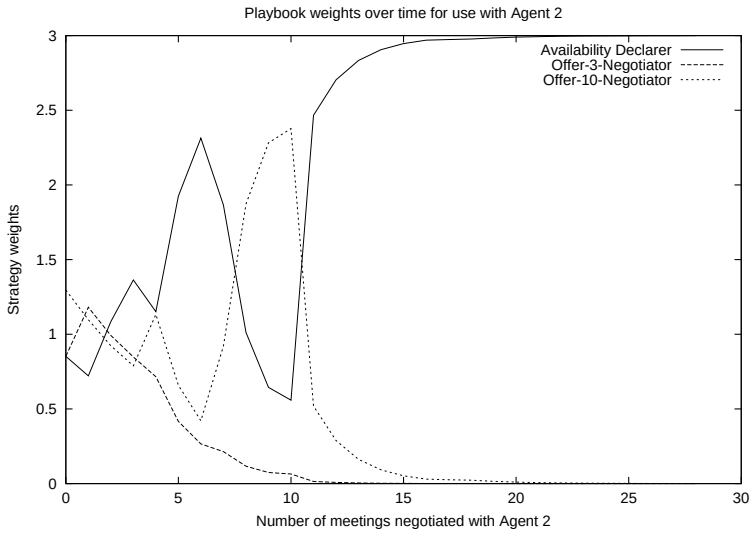

Fig. 7. Weights adaptation for Agent2

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 9 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 9. 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.

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.

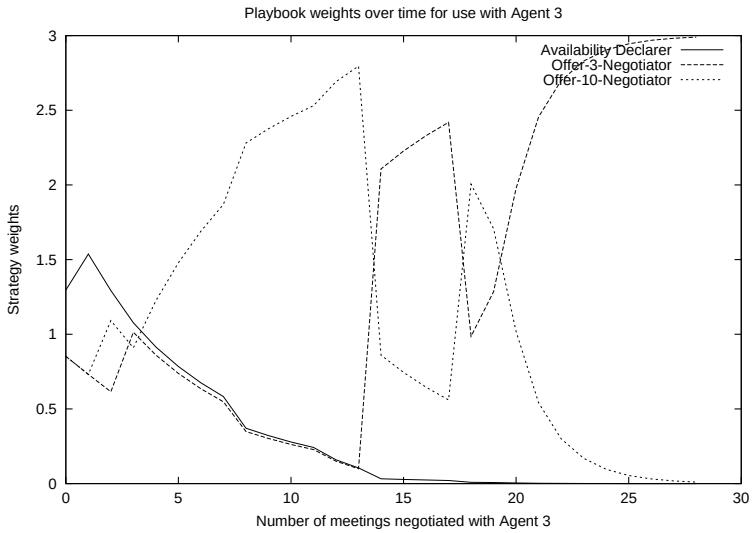

Fig. 8. Weight adaptation for Agent3

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 schedules 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 10 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:

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

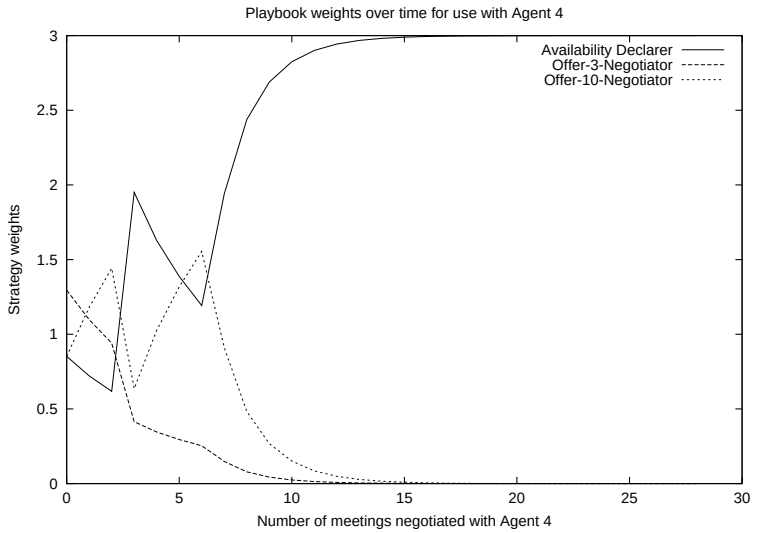

Fig. 9. Weight adaptation for Agent4

The utility of scheduling a series of meetings is then:

$$\alpha * (eval(finalCal) - eval(initialCal)) - \beta * r \tag{5}$$

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 5 and added to the current utility sum for the trial.

Figure 10 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 11 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. These results are typical of a variety of experimental configurations.

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 [8] (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.

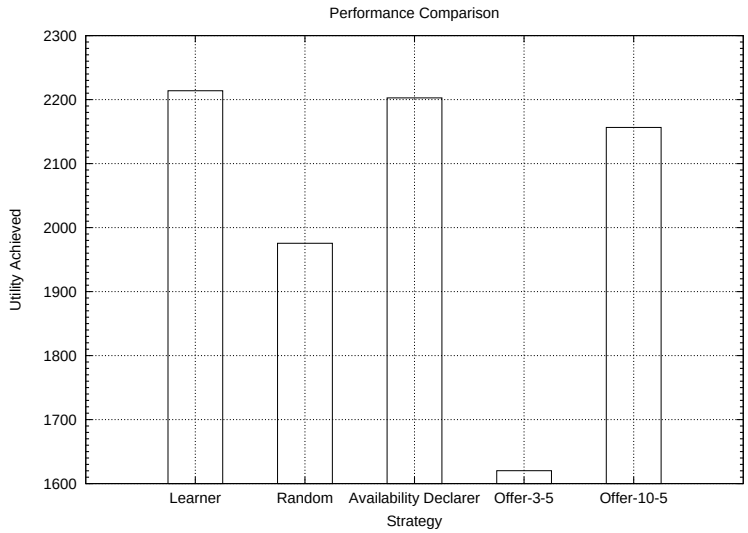

Fig. 10. Performance of the plays-based approach and other algorithms against only fixed agents.

6.5 Experiments and Results: EEE Approach

6.6 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 [14] three different exploration probability settings are discussed — explore-then-exploit, polynomially decreasing exploration, and constant-rate exploration. The SEA setting is version of polynomially decreasing exploration for which strong asymptotic results are provided [9]. 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 [9].

6.7 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

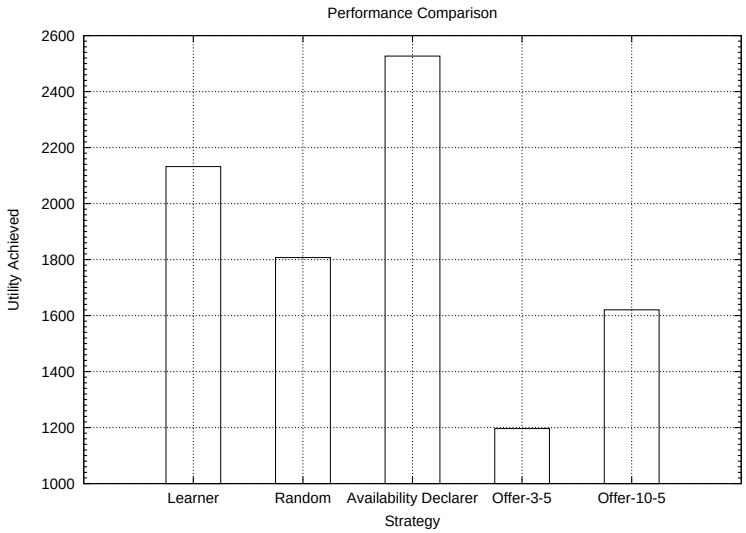

Fig. 11. Performance of the plays-based learner and other algorithms when another learning agent is added.

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 12 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 12 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).

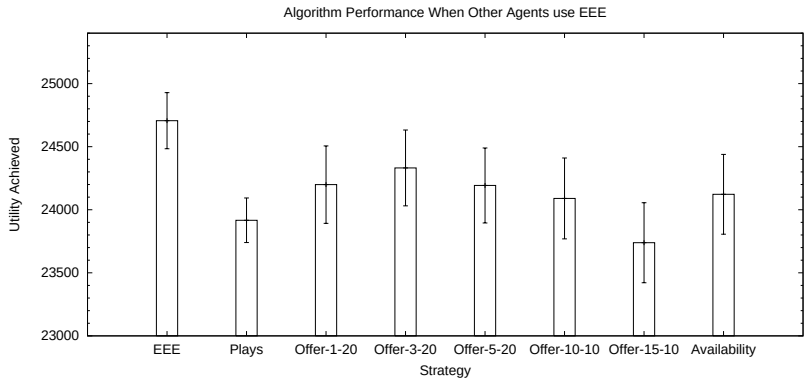

Fig. 12. Graph showing the performance of the different algorithms against 4 agents each using the EEE approach over 300 trials.

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 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 13 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.

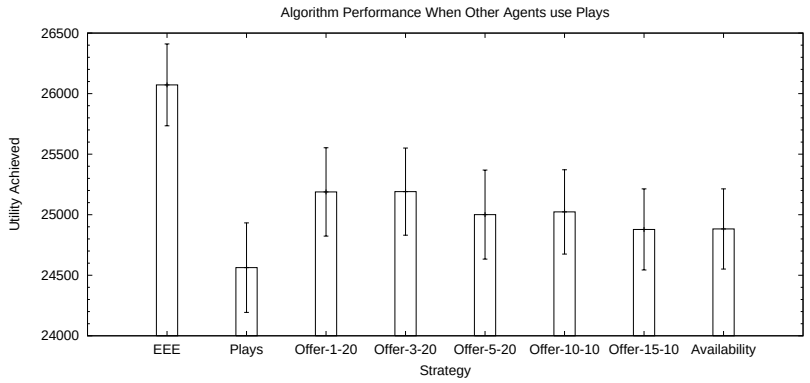

Fig. 13. Graph showing the performance of the different algorithms against 4 agents each using the plays-based approach over 300 trials.

In the previous section we showed that the plays approach is quite effective against agents using fixed strategies. In Figure14 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 6.7. 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 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 this instance, particularly if there was a poor strategy, EEE's tendency to explore more could hurt average utility. The fast convergence of the playbook approach was an advantage in this instance.

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 2. Fixed strategy agents.

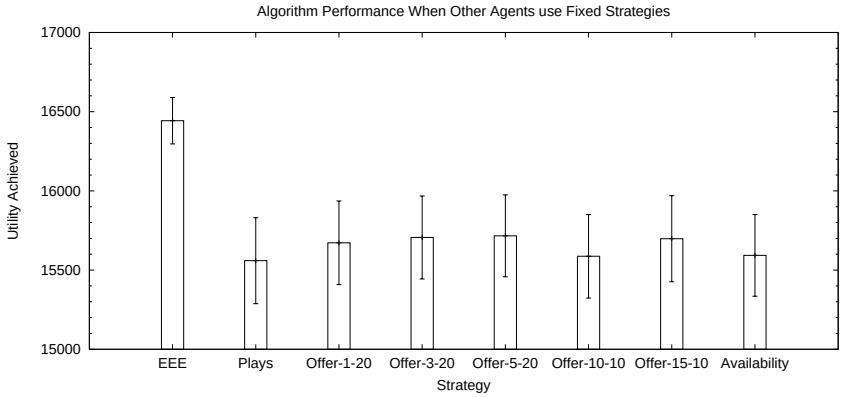

Fig. 14. Graph showing the performance of the different algorithms against 4 agents each using fixed strategies over 300 trials.

7 Related Work

A variety of methods for reaching agreements on meeting times have been proposed in the last ten years, including negotiation based approaches, e.g. [15, 3], Distributed Constraint Reasoning (DCR) approaches [16], and market based approaches [17]. In this section, we describe work on the first two methods, looking in particular at how user preferences are dealt with.

Sen and Durfee [3] conducted a probabilistic and simulation based analysis of negotiation strategies. The basic framework they considered was:

1. Host announces meeting
2. Host offers some times
3. Agents send host some availability information
4. Repeat 2 and 3 until an intersection is found.

Similar protocols have been looked at by other researchers, for example, [15], and [4], while [6] looked at a more complex protocol. These negotiation approaches have handled user preferences for meeting times in quite different ways. Shintani *et al.* [6] 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 [4] and Jennings and Jackson [15] 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, then makes 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.

While the approaches outlined are all concerned with user preferences they differ from the work described here in that we are interested in *how an agent can negotiate strategically in order to satisfy its user's preferences.*

Distributed Constraint Reasoning (DCR) approaches have also been applied to multi-agent meeting scheduling. For example Modi and Veloso [16] model the meeting scheduling problem according to the DCR paradigm and evaluate strategies for making *bumping decisions.* The way in which agents decide when to *bump* (i.e., move an existing meeting to accommodate a new meeting) can have implications for the efficiency of the meeting scheduling process. Intuitively, if the agents want the scheduling process to finish quickly, they should try to bump meetings that will be easy to reschedule. Similarly to the negotiation approaches, the work on DCR has not focused on how agents can act strategically, rather the agents have been assumed to be cooperative.

8 Conclusion

Previous work on negotiation for multiagent meeting scheduling has not looked at how agents can negotiate strategically in order to better satisfy their users'

preferences. In this article, we have shown how an agent can learn to negotiate strategically. The first key step in our approach is to equip an 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 set of diverse and effective strategies cuts down the space the agent needs to learn in considerably. Given this set, we show how the problem of learning to negotiate strategically with other agents can be framed as an experts problem. By considering each strategy to be an expert, a learning agent can use an experts algorithm to adapt strategy selection for each of the different agents it negotiates with. We show how two experts algorithms, the playbook approach [1] and EEE [2], can be used to select negotiation strategies. We demonstrate experimentally that by using EEE in particular, our approach leads to effective online adaptation of strategy selection.

References

1. Bowling, M., Browning, B., Veloso, M.: Plays as effective multiagent plans enabling opponent-adaptive play selection. In: Proceedings of International Conference on Automated Planning and Scheduling (ICAPS'04). (2004)
2. de Farias, D., Megiddo, N.: Exploration-exploitation tradeoffs for experts algorithms in reactive environments. In Saul, L.K., Weiss, Y., Bottou, L., eds.: Advances in Neural Information Processing Systems 17. MIT Press, Cambridge, MA (2005) 409–416
3. Sen, S., Durfee, E.: A formal study of distributed meeting scheduling. Group Decision and Negotiation **7** (1998) 265–289
4. Garrido, L., Sycara, K.: Multi-agent meeting scheduling: Preliminary experimental results. In: Proceedings of the First International Conference on Multi-Agent Systems. (1995)
5. Crawford, E., Veloso, M.: Opportunities for learning in multi-agent meeting scheduling. In: Proceedings of the AAAI Symposium on Artificial Multiagent Learning. (2004)
6. Shintani, T., Ito, T., Sycara, K.: Multiple negotiations among agents for a distributed meeting scheduler. In: Proceedings of the Fourth International Conference on MultiAgent Systems. (2000) 435 – 436
7. Littlestone, N., Warmuth, M.K.: The weighted majority algorithm. Inf. Comput. **108**(2) (1994) 212–261
8. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.: Gambling in a rigged casino: the adversarial multi-armed bandit problem. In: Proceedings of the 36th Annual FOCS. (1995)
9. de Farias, D.P., Megiddo, N.: How to combine expert (and novice) advice when actions impact the environment? In Thrun, S., Saul, L., Schölkopf, B., eds.: Advances in Neural Information Processing Systems 16. MIT Press, Cambridge, MA (2004)
10. Bowling, M.: Convergence and no-regret in multiagent learning. In: Advances in Neural Information Processing Systems 17. (2005) pages 209–216
11. Crawford, E., Veloso, M.: Learning to select negotiation strategies in multi-agent meeting scheduling. In: Proceedings of 12th Portugese Conference on Artificial Intelligence, Springer, LNCS. (2005)

12. Freund, Y., Schapire, R., Singer, Y., Warmuth, M.: Using and combining predictors that specialize. In: STOC. (1997)
13. Crawford, E., Veloso, M.: Learning dynamic preferences in multi-agent meeting scheduling. In: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT). (2005)
14. de Farias, D.P., Megiddo, N.: (Combining expert advice in reactive environments, url=)
15. Jennings, N.R., Jackson, A.J.: Agent based meeting scheduling: A design and implementation. IEE Electronics Letters **31**(5) (1995) 350–352
16. Modi, P.J., Veloso, M.: Bumping strategies for the private incremental multiagent agreement problem. In: AAAI Spring Symposium on Persistant Agents. (2005)
17. Ephrati, E., Zlotkin, G., Rosenschein, J.: A non–manipulable meeting scheduling system. In: Proc. International Workshop on Distributed Artificial Intelligence, Seatle, WA (1994)