Learning Dynamic Time Preferences in Multi-Agent Meeting Scheduling

Elisabeth Crawford **Manuela Veloso**

July 2005
CMU-CS-05-153

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research is sponsored by the Department of the Interior (DOI) - National Business Center (NBC) and the Defense Advanced Research Projects Agency (DARPA) under contract no. 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 the DOI, NBC, DARPA or the U.S. government.

Keywords: meeting scheduling, machine learning, multi-agent systems

Abstract

In many organizations, people are faced with the task of scheduling meetings subject to conflicting time constraints and preferences. We are working towards multi-agent scheduling systems in which each person has an agent that negotiates with other agents to schedule meetings. Such agents need to model the scheduling preferences of their users in order to make effective scheduling decisions. We consider that a user's preferences over meeting times are of two kinds: static time-of-day preferences, e.g., morning versus afternoon times; and dynamic preferences which change as meetings are added to a calendar, e.g., preferences to schedule meetings back-to-back (i.e. in succession). The dynamic nature of preferences has been understudied in previous work. In this paper, we present an algorithm that effectively learns static time-of-day preferences, as well as two important classes of dynamic preferences: back-to-back preferences and spread-out preferences (i.e. preferences for having gaps between meetings).

1 Introduction

In many organizations people find meeting scheduling a time-consuming task. Typically, meetings are scheduled via the exchange of email and it can be difficult to find mutually satisfactory times (particularly when there are many participants). Furthermore, in some cases existing meetings need to be moved to accommodate new meetings. The deployment of multi-agent meeting scheduling systems, in which each person has an agent that negotiates meeting times with other agents, has the potential to save computer users significant time. Two key hurdles need to be overcome before meeting scheduling agents can be produced. First, effective methods for learning the complex preferences users have about meeting times are required. Second, we need methods for negotiating meeting times between agents that satisfy users' preferences and constraints, and operate efficiently. Both of these problems have been studied before, but neither is yet well solved. The first problem, learning user preferences, is the focus of this paper.

Previous approaches to learning the meeting time preferences of users (discussed further in Section 2) have a common drawback — they do not deal with dynamic preferences. The time preferences of users are dynamic, often changing as new meetings are added to the calendar. Some aspects of time preferences are static, for instance, the general preference for afternoon meetings over morning meetings. However, someone who prefers afternoons may occasionally need to schedule a meeting in the morning. When this happens, such a person may be happy to have other new meetings scheduled that morning. A related and common dynamic preference is the preference for having meetings back-to-back. Many people find they work best in large, uninterrupted blocks of time. As such, they prefer to have their meetings bunched together, instead of spread-out over the day. Other people dislike having their meetings in close succession. From both cases, we see that a person's *preferences for times can change as meetings are added to their calendars.*

We have developed a simple memory-based approach (described further in Section 3) for selecting the most probable times for a new meeting given the user's current calendar. We use changes in the user's calendar, in particular the scheduling of new meetings, as our training data. This data contains noise from the perspective of learning a user's preferences, since meetings in a user's calendar are often scheduled according to the preferences of all the meeting participants, not just the user in question. However, as we discuss further in Section 3, it is unlikely a deployed system will have access to better data. The implicit assumption we are making, is that the most *probable* times for a new meeting according to our learned model, are a reasonable estimate of the most *preferred* times.

To evaluate our approach we have defined a variety of different preference profiles and some simple strategies that the agents use for negotiation. We have generated data for our learning algorithm by having a set of agents schedule meetings according to their preferences and the pre-defined negotiation strategies. We test how successfully our algorithm has learned an agent's preferences by scheduling meetings according to the learned preferences and according to the true preferences and then using the true preferences to evaluate the resulting schedules. Our evaluation methodology is described further in Section 4.

In Section 5 we show that on average the schedules produced by the learned preferences are only 6% worse than those produced by the true preferences. In Section 6 we conclude and discuss how our approach can be extended.

2 Related Work

The Multi-Agent meeting scheduling problem has been much studied in the last ten years e.g. [8, 12, 6] and more recently [10, 1, 5], however it is not yet satisfactorily solved. In this section we discuss existing approaches to learning a user's time preferences.

Mitchell *et al.* [9] present CAP, a learning apprentice designed to assist a user in updating their calendar. Each time the user adds a meeting to her calendar (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. CAP uses a technique called Greedy Attribute Selection [3] to decide which attributes to use in the decision trees. It is not clear if the pool of attributes covers the kind of dynamic preferences we are interested in, in this paper. 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 [2] was able to achieve an accuracy of almost 60%, using the Winnow algorithm. While this is a large improvement, the level of accuracy, given the amount of interaction with the user (the user explicitly accepts or rejects everything CAP does), is probably not high enough for this system to be deployable.

Oh and Smith [11] take a statistical approach to learning static 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 their agent can observe and understand negotiations between people about meeting times, 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. Making these assumptions allows Oh and Smith to learn static time preferences with good accuracy, but it restricts the immediate practical applicability of the learning algorithm.

Lin and Hsu [7] 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. Also, Lin and Hsu state that for some criteria more than 1000 training examples would be needed.

Kozierok and Maes [8] 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. This is an interesting approach, but the attributes that distinguish one situation from another are hard-coded and it restricts the choice of group decision mechanism in the multi-agent system to a particular type of negotiation approach.

The approaches described either require a great deal of user interaction, or very detailed (and probably unattainable) training data. Furthermore the handling of dynamic time preferences has been much understudied. Our approach is an attempt to address these two points.

In this paper we show that using only very limited training data, we can effectively schedule meetings for a user who has static and dynamic time-of-day preferences.

3 Learning Dynamic Time Preferences

In this section we describe our method for capturing information about user preferences. We also introduce a novel method for modeling the dynamic nature of user preferences and describe how we can learn models of this form.

3.1 Capturing User Preference Information

In the previous section we described a number of approaches to collecting the scheduling preferences of users. Mitchell *et al.* and Blum [9, 2] collected information about the user's preferences by asking the user. The user was not directly queried, rather they 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 the user in a negotiation setting we would require their participation unreasonably often.

Oh and Smith [11] proposed capturing user preferences by passively observing the user negotiate meeting times with others. An assistant agent could observe meetings scheduled via e-mail, however with current natural language processing technology it would be very hard to reliably extract the necessary information. A work around would be to require the user to schedule all their meetings through a structured interface that the agent controls and is thus able to interpret. Using such an interface, the user could negotiate meeting times via e-mail while the agent observed. The main difficulty with this approach is that users may not be willing to use a new interface. Assistant agents that work seamlessly with existing software are much more likely to be widely adopted.

We propose an approach whereby the user's preference information is captured by simply observing the changes in the user's electronic calendar. Data of this form is easy to collect and this approach allows us to assume as little as possible about how the user will be willing to schedule their meetings. Using this approach we do not have to assume the use of any specific negotiation procedure. Furthermore, this approach allows us to capture the dynamic aspects of the user's preferences. A possible difficulty with this approach is that changes in a user's calendar depend not only on the preferences of the user, but also on the preferences of the people the user meets with. However, note that we have much the same problem with data obtained from watching negotiations. On first glance it may seem that the times a user proposes for meetings *first* are their most preferred times. However, the times proposed might have been selected according to who the user is planning to meet. For instance, when a junior staff member needs to meet with a senior staff member she may offer times that she thinks the senior member will prefer. In this instance, the offered times tell the agent nothing about the junior staff member's time preferences.

In this paper we show how capturing user preferences by observing changes in the user's calendar can lead to effective learned models of dynamic user preferences.

3.2 Modeling User Preferences

We are interested in being able to model dynamic preferences. We don't simply want to represent a coarse model such as whether a user prefers meetings in the morning, or afternoon. Rather we want to model what times a user prefers subject to her changing calendar. 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 dynamic time preferences by placing probabilities on transitions between possible states of the user's calendar. The current state of the user's calendar determines the states of the calendar that are reachable through the addition of one new meeting. Moreover, some of these reachable states are more preferable according to the user's dynamic preferences than others. We account for these preferences by placing probabilities on the transitions.

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

- S is the set of all possible states of the user's *day_calendar*. *day_calendar* is a bit vector, one bit for each time slot in the user's calendar for a particular day of the week. A one in a time slot indicates there is a meeting at this time and a zero indicates the time is free.

- Consider $s, s' \in S$, then $(s, s') \in \vec{E}$ iff we can get s' from s by adding one meeting. So s' contains exactly one more one bit than s and s' has a one in every slot that s has a one.

- Each edge $(s, s') \in \vec{E}$ has a weighting corresponding to the probability of transitioning to s' from s.

In Figure1 we show an example graph for a 4 time slot day where the user prefers to meet at the first time and the last time but also has a strong preference for having meetings back-to-back. We have placed probabilities on some of the links in the example to demonstrate this time-of-day and back-to-back preference.

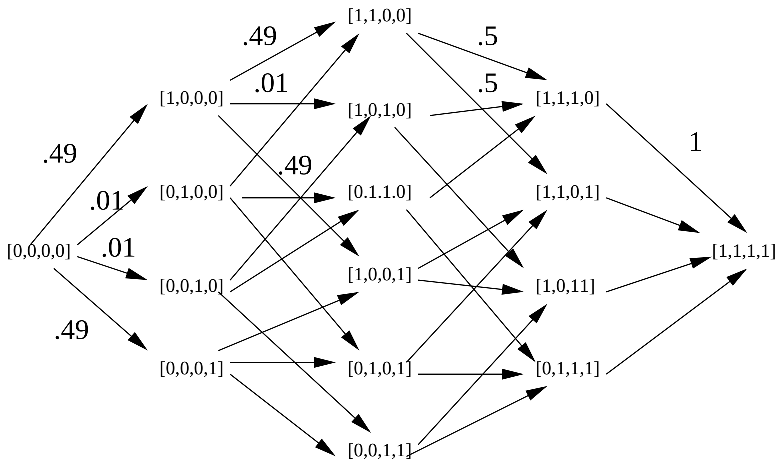

Figure 1: Example graph for four time slots

3.3 Learning the Model

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

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

- $t_i = \{current\ day\_calendar_i,\ new\ day\_calendar_i\}$
- *new day_calendar* is simply a calendar obtained by adding one extra meeting to *current day_calendar*.

Training data of this form is very easy to collect. Each time a meeting is added to the calendar we simply record the current calendar configuration and the new calendar configuration.

Given a set of training data, we create the graph and then use the data to annotate the graph in the following way. For each training instance pair t_i we locate the $current\ day\_calendar_i$ state in our graph and increment the state's frequency count. We then identify the edge that connects $current\ day\_calendar_i$ with the state for $new\ day\_calendar_i$ 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.

3.4 Using the Model

When a new meeting needs to be scheduled we use the graph to rank the possible times, according to their probability, as follows:

- We identify the state s corresponding to the current state of the calendar.
- 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 times.

- 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 $(s, s') \in \vec{E}$.

In this paper we consider *day_calendars* with 8 time slots. Each possible *day_calendar* is thus represented as an eight bit sequence. In total there are 256 possible bit sequences and hence 256 possible calendar configurations.

4 Experimental Setup

In order to evaluate our learning procedure we have created some realistic models of user preferences that include dynamic and static elements. To generate training data we log the changes in an agent's calendar that occur as the agent schedules meetings with other agents using a negotiation strategy and the 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 model. We compare the performance of the different preference models by evaluating the schedules that result (according to the true preferences). In this way we are able to evaluate the usefulness of the learned preference model.

4.1 Preference Models

4.1.1 Dynamic Preference Model

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 dynamic 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.

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

where:

- $TOD(t)$ 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 γ are constant weightings. In our experiments $\beta > 0$ iff $\gamma = 0$ and vice versa.

4.1.2 Random Preference Model

We also 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. 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. 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 TOD 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 Negotiation Strategies

We have used two negotiation strategies in our experiments. Each meeting to be negotiated has an initiator, a list of attendees and a preferred day. In both negotiation strategies agents offer times for a meeting in order of preference.

4.2.1 Offer-n-Negotiator

The negotiation strategy 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. A meeting initiator running this strategy behaves according to the following policy.

1. Offer n available, previously un-offered times for the specified day (or as close to n as possible) to the attendees.

2. When a response has been received from all attendees in this round:

   - If in negotiation mode, and there is an intersection in offered times, send a confirmation request for the best time in the intersection. Switch the meeting to pending mode. Go to 2.
   - If in negotiation mode, and there is no intersection go to 1.

- If in pending mode, and all attendees accepted the confirmation request, send a confirmation message and add the meeting to the calendar.
- If in pending mode, and one or more of the attendees rejected the confirmation request, cancel the confirmation request with the attendees, switch the meeting to negotiation mode and go to 1.

If in the above 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.

An attendee agent using this strategy operates as follows when it receives a message about a meeting:

1. If this is a new meeting enter negotiation mode for this meeting and go to 2.
2. If in negotiation mode for this meeting, and this message is not a confirmation request, offer n available, previously un-offered, times (or as close as possible) on the requested day.
3. If in negotiation mode and this message is a confirmation request:
   - if the time requested is not pending for another meeting or now occupied by another meeting send a message to the initiator accepting the time and enter pending mode
   - otherwise, send a message rejecting the request.
4. If in pending mode this and message cancels the confirmation request, move the meeting to negotiation mode, go to 2.
5. If in pending mode and this message is a final confirmation add the meeting to the calendar.

The Offer-n-negotiator 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 [12, 4].

4.2.2 Hold-Out-for-n-Negotiator

The mechanics of this strategy are similar to those of the Offer-n-Negotiator with the following exceptions:

1. The initiator and attendees only offer one new time per negotiation round.
2. After offering 2 times the initiator 'holds out'. If the attendees stop offering new times, the initiator proposes a new day.
3. An attendee waits n rounds before offering a time that is not in its top two choices.

4.3 Training Data

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

4.3.1 The Training Process

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 these meetings using the 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.

4.4 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 between the agents 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 agent's 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's utility according to the true preferences of the learning agent. These utilities are averaged over all the calendars to produce the average utility score for the preference model in use. We then compare these average utility scores.

5 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 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

Figure 2: Scheduled using TOD and back-to-back preferences with Offer-n-negotiator

being scheduled back-to-back. The performance obtained using the learned preferences averaged over the first 30 training sets 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 was only 25% worse than negotiating using the true preferences. This is partly because no matter when the meeting is scheduled, it still yeilds some utility. The other reason is that it is not just the learning agent's preferences that determine the meeting times. If the agent was always able to get her favourite times for meetings 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 limitted. 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 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.

Figure 2 also demonstrates that even having one training example helps the learned model outperform the random model. 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. However on the whole our experiments show that adding training examples improves performance.

Figure 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

Figure 3: Scheduled using back-to-back and TOD preferences and Hold-Out negotiator

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 mis-ranked times are more costly since the agent is likely to get the times it holds out for.

Figure 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 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 maintaing good performance.

Figure 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

Figure 4: Scheduled using spread-out and TOD preferences

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 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 of the misleading examples and the performance increases again.

We have also explored some of the 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.

6 Conclusions and Future Directions

We have introduced a method to capture, model and learn the dynamic scheduling preferences of users. Other methods for learning user preferences have assumed the use of hard (possibly impossible) to obtain data. Furthermore, the dynamic nature of user preferences has been largely ignored. In this paper we have shown how we can capture dynamic user preferences by simply logging changes in a user's calendar and use this data to learn effective models. We have demonstrated the effectiveness of our approach by showing that using learned time-of-day, back-to-back and spread-out preferences, we can negotiate schedules that are within 6% of what can be acheived 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 in this paper 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 even after only a small number of training examples leads us to believe it can be extended to effectively cover this case.

We are also interested in studying users' negotiation preferences. This encompasses problems such as learning who a user is prepared to sacrifice their scheduling preferences for and whose meetings they are prepared to move. Learning user negotiation preferences is likely to be a very challenging multi-agent learning problem given that any two agents will negotiate a meeting time quite infrequently.

References

[1] Pauline Berry, Melinda Gervasio, Tomas Uribe, Karen Myers, and Ken Nitz. A personalized calendar assistant. In *Working notes of the AAAI Spring Symposium Series, March*, 2004.

[2] 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.

[3] R Caruana and D. Freitag. Greedy attribute selection. In *Proceedings of the Eleventh International Conference on Machine Learning*, 1994.

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

[5] Elisabeth Crawford and Manuela Veloso. Learning to select negotiation strategies in multi-agent meeting scheduling. In *the working notes of the Multiagent Learning Workshop (to appear), AAAI, Pittsburgh USA*, 2005.

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

[7] Shih jui Lin and Jane Yung jen Hsu. Learning user's scheduling criteria in a personal calendar agent, 2000.

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

[9] 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.

[10] Pragnesh Jay Modi, Manuela Veloso, Stephen F. Smith, and Jean Oh. Cmradar: A personal assistant agent for calendar management. In *International Workshop on Agent-Oriented Information Systems*, 2004.

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

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