A LEARNING INTERFACE AGENT FOR SCHEDULING MEETINGS

Robyn Kozierok
MIT Media-Lab
20 Ames Street Rm. 484c
Cambridge, MA 02139
robyn@ai.mit.edu
(617) 253-2137

Pattie Maes
MIT Media-Lab
20 Ames Street Rm. 489
Cambridge, MA 02139
pattie@media.mit.edu
(617) 253-7442

ABSTRACT
This paper describes a Learning Interface Agent for a meeting scheduling application. The agent employs Machine Learning techniques to customize itself to the user's personal scheduling rules and preferences by observing the user's actions and receiving direct user-feedback. Our approach provides the user with sophisticated control over the gradual delegation of scheduling tasks to the agent, as a trust relationship is built. We report upon an experiment in which a collection of such assistants became gradually more helpful to their users through the use of memory-based and reinforcement learning. The experimental data reported upon demonstrate that the learning approach to building intelligent interface agents is a very promising one which has several advantages over more standard approaches.

KEYWORDS: Interface Agents; Learning Interface Agents; Machine Learning; Personal Assistants; Software Agents

INTRODUCTION

An interface agent is a semi-intelligent, semi-autonomous system which assists a user in dealing with one or more computer applications. Interface agents typically behave as personal assistants: they have knowledge about the tasks, habits and preferences of their users and use this knowledge to perform actions on their behalf [4, 5, 7]. Recently, several computer manufacturers have adopted this idea to illustrate their vision of the interface of the future (cf. videos produced in 1990-1991 by Apple, Hewlett Packard, Digital and the Japanese FRIEND21 project). These videos, which portray interface agents "as they might be" have made this metaphor for Human-Computer Interaction a hot topic of discussion; however, current AI techniques leave us well short of the goal of actually implementing the sophisticated agent behavior envisioned by these vendors. One important problem is that of ***knowledge acquisition***: How does the agent acquire the knowledge about the user which it needs to provide effective personalized assistance? Maes and Kozierok [7] argue that neither of the currently popular approaches, the Knowledge-Engineer-programs-the-agent and the User-programs-the-agent approaches, is a satisfactory solution to this problem.

Maes and Kozierok [7] propose a third approach which relies on the use of Machine Learning techniques: the agent *learns* the knowledge it needs to effectively assist the user by observing and imitating that same user (*learning by observation*). The agent's learning may be accelerated by receiving direct feedback from the user (*reinforcement learning*) and by being given specific instructions by the user (*learning by being told*).

Advantages of this approach are that it requires less work from the end-user and application developer. Further, the agent is more adaptive over time and the agent can be customized to individual user preferences and habits. Another particular advantage of the learning approach is that the user and agent can gradually build up a *trust relationship*. The agent develops its abilities gradually, which allows the user to also gradually build up a model of how the agent makes decisions. The user can decide in how far to delegate tasks, and can even decide not to delegate any tasks at all until he or she becomes sufficiently confident in the agent's predictions.

In the remainder of this paper we present a semi-intelligent agent for meeting scheduling implemented using simple but very powerful learning techniques: *Memory-Based Learning* [10] and *Reinforcement Learning*. This concrete example demonstrates that Learning Interface Agents have several advantages over other approaches to building interface agents. The initial results from an experiment using this system show that

the approach is a realistic one, and that such an agent can acquire sufficient knowledge to begin being useful to its user within a short period of time.

OVERVIEW OF THE SYSTEM

The system under construction is composed of a calendar and scheduling component coupled with a learning agent. Each person in the organization has a personal copy of the system so that the agent portion can become customized to that individual's preferences and habits. The user's actions within the calendar/scheduling component provide the inputs to the agent, which learns by observing this behavior. The agent is also able to interact with the calendar/scheduling component on the user's behalf, but does so only if the user wishes it to.

The agent-user relation is modeled after the concept of a *personal secretary*: the agent has some initial general knowledge about scheduling meetings; however, initially most of the scheduling is done by the user. Gradually the agent gets to know the habits and preferences of its user (by observing the actions of the user, by making suggestions for scheduling decisions and receiving positive or negative feedback) and over time its suggestions become better (more often equal to what the user decides to do in some situation) and therefore more reliable. The user thus comes to trust the agent more and can decide to delegate more and more meeting scheduling decisions.

The calendar/scheduling component consists of a graphical calendar and menu-based interface, as shown in Figure 1. The user or agent may initiate a new meeting, or change or cancel a meeting he/she/it had previously initiated. It is also possible to request a change in or cancellation of a meeting which had been scheduled by someone else. When initiating a meeting, a time and date may be selected directly, or a range of times and/or dates may be presented to the set of agents involved, who will then negotiate to come up with an optimal time within the given range. At initiation, a meeting may also have a frequency specified, i.e. once, weekly, monthly, etc. In response to a meeting invitation, the user (or agent) may choose to accept, decline, request a change in the time and/or date selected, or agree to attend part of the meeting (if, for example, there was an overlap between it and a previously scheduled meeting). In response to a request for a change in a meeting time, the choices are to refuse the request, to select a new time by hand, or to allow the agents involved to negotiate to suggest a new time. Similarly, a request to change or cancel a meeting may be honored or refused by the original initiator. (The user may also schedule outside meetings or personal appointments, but this, of course, is not something the agent could do on his or her behalf.)

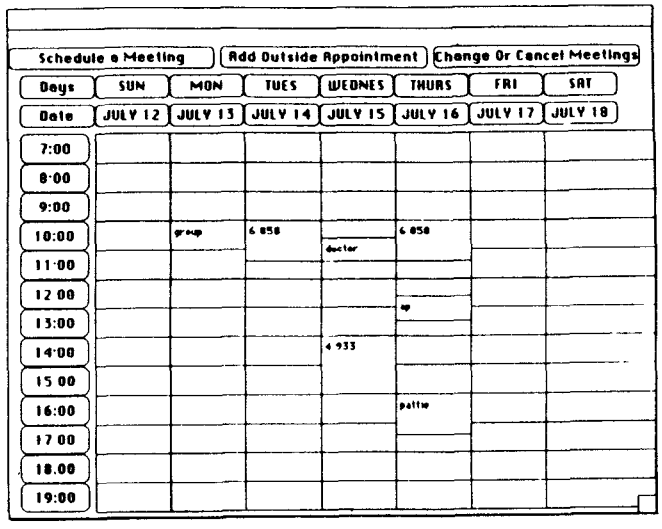

Figure 1: The calendar interface.

Communication among the different users or agents happens through the automatic generation of semi-structured electronic mail messages. For example, the command to initiate a meeting will result in the sending of a meeting invitation message to the calendar mailboxes of the invited users. The incoming structured messages are buffered and can either be handled by the agent or by the user. The agent can take advantage of its ability to communicate with other users' agents to enable it to suggest an optimal time and date for a given meeting. Because the agent has learned knowledge of the user's preferences, and the other agents have learned knowledge of their users' preferences, it is in an ideal position to suggest a meeting time that will be convenient for the user and also likely to be accepted by the others being invited.

At first the user has to manually choose the action to take in any given situation, but as time passes, the agent's observation of the user allows it to recognize patterns in the user's behavior and eventually learn to predict the user's actions. In each situation, (or, at the user's preference, in each situation in which the agent has at least the user-provided threshold of confidence in its prediction) the agent tells the user what it thought the user would do, how much confidence it had in its prediction, and, if desired, why the particular action seemed likely in the situation (based on similar situations in which the user had previously chosen that action). A user who has come to trust the agent's predictions can hand over control to it. This can be complete, or, more likely, partial, for example, allowing it to take the action without checking whenever the meeting in question is at least a certain amount of time away and the agent's confidence in the action is above a second

threshold (which will likely be higher than the one at which the user is interested in seeing the agent's predictions). The agent can provide a report to the user of the actions it has recently taken autonomously.

At this time, we have implemented a system which allows users to keep track of their schedules, initiate meetings with other users and respond to their invitations. The system also includes an agent which observes the user's actions and predicts a response to each invitation the user receives. Additionally, this agent can initiate negotiations with other agents to suggest an optimal time for a meeting. The paper takes a sophisticated calendar and scheduling system (with graphical interface, etc.) such as *MeetingMaker*^{TM} for granted and instead elaborates upon the learning agent aspects. This implementation has been done in Lucid Common LISP on a Unix workstation, as well as in Macintosh Common LISP. The graphical interface to the calendar/scheduling component has been partially implemented on the Macintosh (see Figure 1).

HOW THE AGENT LEARNS

There are two things the agent does to enable it to eventually learn to predict the user's action in a particular situation. First, it keeps a memory of everything the user does, stored as situation-action pairs, which are simply raw data about what happened. Given a new situation, it can then use memory-based learning techniques to find the remembered situations which it believes to be closest to the new situation, and thus predict the most likely course of action.

Second, it maintains a set of priority weightings for meeting topic keywords, and for the relative importance of other participants. This information is used to help interpret the meaning of the raw information in the memory. For example, a person's impression of the initiator or other participants in a given meeting may influence the decision whether to accept an inconvenient time or ask that it be changed. Initially, all these priority weightings are set to a neutral value. They may be changed manually by the user, and are also adjusted automatically by reinforcement learning techniques whenever the agent's predicted action turns out to be incorrect. The ratings of the other participants are also used when the agents collaborate to suggest a new meeting time. Figure 2 shows the structure of the agent, and the flow of information between its various components and the user in the learning stage of operation.

Memory-Based Learning

The majority of the learning the agent does uses *memory-based learning* techniques based on [10]. The basic idea is to compare a new situation against each of the situations which have occurred before. Then the

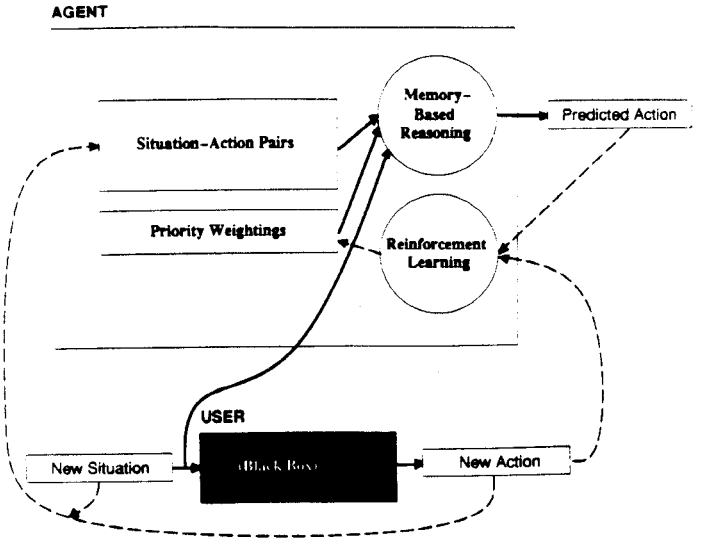

Figure 2: When a new situation arises, the memory-based reasoning module takes it, along with the stored situation-action pairs and priority weightings and attempts to predict the user's action. Meanwhile, the user considers the situation, selects an action, and performs it. (If the agent is confident enough in its prediction to make a suggestion and the user decides to accept it, this is treated in the same manner as the user deciding independently to select that action.) The reinforcement learning module then compares the predicted action to the user's action, and, if necessary, updates the priority weightings. At the same time, the situation is paired with the action the user took and added to the memory of situation-action pairs.

agent looks at the actions taken in a small number, m ($m = 5$ in this experiment), of the "closest" situations to predict the action in the new situation. The algorithm determines which features of the situation are most relevant to predicting the action by analyzing all of the data collected to date for correlations, as described below.

One of the main benefits of this type of learning is that all the information about situations and actions is remembered. Another advantage is that it allows the agent to give "explanations" for its reasoning and actions in a language that the user is familiar with, namely in terms of past examples which are similar to the current situation ("I thought you might want to take this action because this meeting and your current calendar are similar to a situation we have experienced before").

Currently the set of features (fields) which comprise a situation is hardcoded. It includes thirty-two details about the meeting at hand, about any conflicting meetings, and about the user's schedule for the day and week in question. In the **FUTURE WORK** section we discuss alternatives to hardcoding these features. Actions are

things like initiating a meeting, accepting an invitation, rescheduling a meeting, etc.

The distance between a new situation and a memorized situation is computed as a weighted sum of the distances between the values in each field. The distance between field-values is based on a metric computed by observing how often in the memory the two values in that field correspond to the same action. The weight given to a particular field depends upon the value of that field in the new situation being considered, and is computed by observing how well that value in the field has historically correlated with the action taken.

The agent predicts an action by computing a score for each action which occurs in the closest m memorized situations and selecting the one with the highest score. The score is computed as:

$$\sum_{s \in S} \frac{1}{d_s}$$

where S is the set of memorized situations predicting that action, and d_s is the distance between the current situation and the memorized situation s.

The weight and distance metrics theoretically need to be recomputed whenever a new situation is added to the memory; however, this is a relatively time-consuming ($O(n^2)$) procedure so we propose having the agent do this computation only once per day, preferably overnight. This has the effect that the agent cannot learn from anything you do until the next day, which seems a reasonable trade-off for the additional computation.

Once the above computation has been done, the actual prediction is only $O(n)$, which is quite reasonable provided n is not allowed to get too large. This will be ensured by keeping a bounded number of entries in the memory. This is probably advisable in any case, since entries which are too old may reflect user preferences and habits which have since changed.

Along with each prediction it makes, the agent computes a confidence in its prediction, as follows:

$$\left(1 - \frac{\frac{d_{predicted}}{n_{predicted}}}{\frac{d_{other}}{n_{other}}}\right) \times \frac{n_{total}}{m}$$

where:

- m is, as before, the number of situations considered in making a prediction,
- $d_{predicted}$ is the distance to the closest situation with the same action as the predicted one,
- d_{other} is the distance to the closest situation with a different action from the predicted one,
- $n_{predicted}$ is the number of the closest m situations with distances less than a given maximum (in this case 12.0) with the same action as the predicted one,
- n_{other} is the minimum of 1 or the number of the closest m situations with distances within the same maximum with different actions than the predicted one, and
- $n_{total} = n_{predicted} + n_{other}$, i.e. the total number of the closest m situations with distances below the maximum.

If the result is < 0, the confidence is truncated to be 0. This occurs when $d_{predicted}/n_{predicted} < d_{other}/n_{other}$ which is usually the result of several different actions occurring in the top m situations. If every situation in the memory has the same action attached to it, d_{other} has no value. In this case the first term of the confidence formula is assigned a value of 1 (but it is still multiplied by the second term, which in this case is very likely to lower the confidence value as this will usually only happen when the agent has had very little experience).

This computation takes into account the relative distances of the best situations predicting the selected action and another action, the proportion of the top m situations which predict the selected action, and the fraction of the top m situations which were closer to the current situation than the given maximum.

Reinforcement Learning

After each user action, whether correctly predicted by the agent or not, the new situation-action pair is added to the memory, giving the agent a better chance to deal correctly with similar situations in the future using the memory-based techniques selected above.

When the prediction made by the agent is incorrect, a reinforcement learning process is also used to help ensure that a similar mistake is less likely to occur in the future. The user is given an explanation of why the agent made the decision it did, and an opportunity to inform the agent if the decision was incorrect because the agent had attributed either too much or not enough importance to one of the features of the situation (e.g. the initiator, participants or topic of the meeting being considered).

This information is used to adjust the priority weightings the agent keeps. The values for the relevant objects are corrected by a small constant amount multiplied by a positive or negative modifier determined by how much more positive or negative the action taken was than the

action predicted. Thus if the user accepts an invitation the agent expected him/her to decline, and says that the reason (or one of the contributing factors) was the initiator's importance, the initiator's rating will be increased, as in this case the more positive action taken by the user causes the modifier to be positive. Similarly, if the user declines an invitation the agent thought he/she would accept, and claims an irrelevant topic as an excuse, that topic's priority rating will be decreased, because this time the user's action was more negative than the one the agent expected.

When comparing the user's action with the agent's prediction, things like canceling or rescheduling a meeting which used to be scheduled at a conflicting time to enable attendance at a new meeting (perhaps a personal audience with the president of the company) are considered very positive actions, while simply accepting an invitation is considered a moderately positive action. A moderately negative action would be to request that the time of the new meeting be changed to better fit the one's schedule, and a very negative action would be to decline the invitation outright.

SUGGESTING A MEETING TIME

A user wishing to initiate a meeting has the choice of manually selecting the time, or providing a range of acceptable dates and times and asking the agent to suggest one. If the agent is asked to suggest an appropriate time for a meeting, it first sends out a message to the other users' agents requesting that they each send a list of times their user is free during the range of dates in question. It then intersects these and makes a list of all time-slots (starting on 15-minute boundaries) in which everyone is free (or if this is impossible, at least everyone deemed by the meeting initiator to be mandatory to the meeting).

It then sends another message to each agent, asking it to send ratings for each of the possible time-slots and for each of the other participants in the meeting. An agent rates times by imagining that its user was invited to a meeting like the one being scheduled at the candidate time, and giving a score based on the goodness or badness of the action predicted and the strength of the prediction. Once all the candidate times have been rated the agent normalizes the scores onto a scale from 0 to 100. The initiating agent then uses the following algorithm to find the most convenient of the available times.

Given:

- candidate times $t_1, t_2, \ldots, t_m$,
- people $p_1, p_2, \ldots, p_n$,
- preferences r_{ij} defined as person p_i's preference rating for time t_j, and
- priorities q_{ij} defined as person p_i's assessment of the relative importance of person p_j. ($\forall i\ q_{ii} = 0$; other q values are relative to that in the range $[-100, +100]$.)

Then define the convenience of any given time t_k to be:

$$t_k = \sum_{i=1}^{n} \left(r_{ik} \sum_{j=1}^{n} q_{ji} \right)$$

In other words, the sum over all participants of each person's preference for that time, weighted by that person's overall importance, defined as the sum of the importance measures assigned to him or her by the other participants.

PERFORMANCE

The agent was tested on a set of simulated data in order to allow testing several months' worth of meeting scheduling in a short period of time. The simulated data was based on a hypothetical group of people which included one professor, her director and six of her students: three graduate students and three undergraduates. Each person was assigned a set of characteristics which included their regular schedules (mainly classes and regular outside meetings) and the types of times they preferred to meet. Hypothetical group dynamics were also devised which specified things like which people would have reasons to meet with one another, and approximately how often. Most of these were based on actual people and situations, but a few were made up since several of the people in the group had very similar preferences which did not present as much diversity as desired. A hypothetical series of meeting requests was devised based on the supposed dynamics of the group and then a response for each of the invited individuals was chosen based on the characteristics which had been assigned to that person. This series of meetings was then presented to the agents along with the hypothetical responses. The agents' predictions were recorded to allow analysis of their performance.

Figure 3 contains graphs displaying the confidence each agent had in each of its predictions, and whether in fact that prediction was correct. The first five predictions made by each agent were not plotted, as they were essentially random. (Each person was invited to a different number of meetings, and thus had a different number of predictions made.) It is clear that the frequency of incorrect predictions is drastically reduced as time goes on, and that the confidence in correct predictions begins to rise, while the confidence in incorrect predictions is kept quite low. It is our intention to eventually provide graphs such as these to the users of these

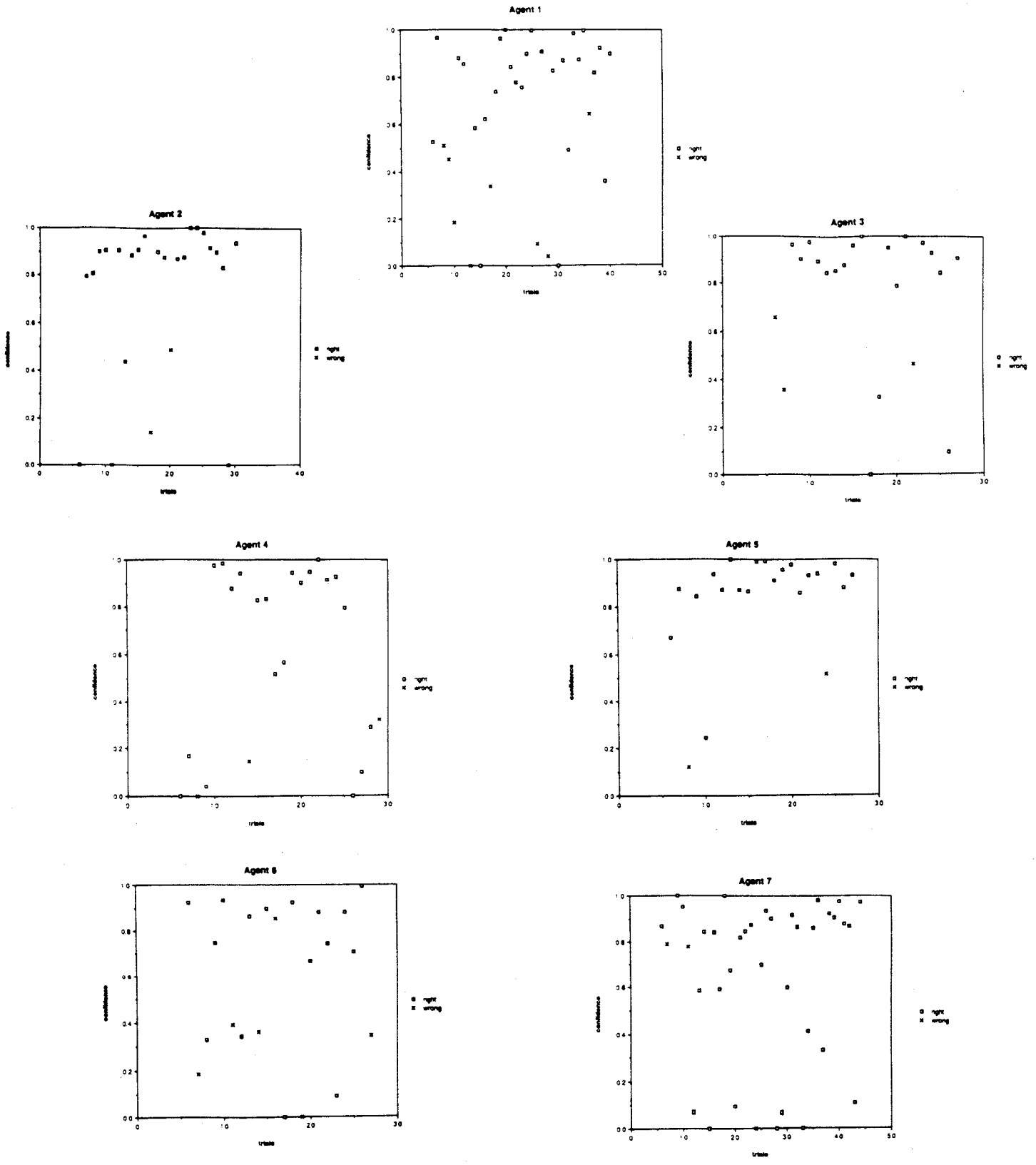

Figure 3: Confidence and Correctness of Predictions for each Agent.

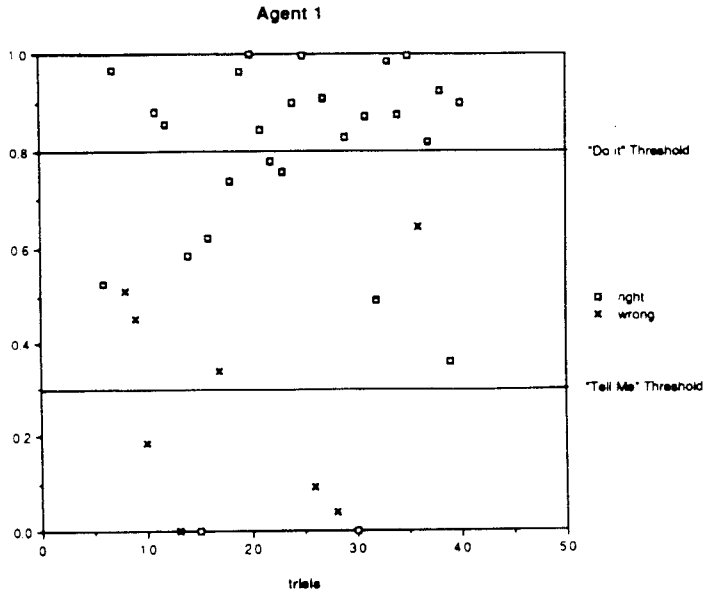

Figure 4: Hypothetical threshold settings for Agent 1.

agents, in order to help them decide when to hand over control to the agent, and what confidence-level cut-offs would be most appropriate for having the user present its suggestion and for allowing it to take the action autonomously. For example, Agent 1's user might wish to set thresholds as shown in Figure 4.

Agents 1 and 2 represented the users with the most complex and idiosyncratic behavioral patterns, while agent 3 represented the simplest user pattern. Agent 7 represented a user with fairly complex and unusual preferences, but with an aversion to scheduling regular (i.e. weekly) meetings. This resulted in the agent having the opportunity to deal with a large number of fairly similar requests, allowing it to more easily identify the important features and thus making accurate prediction easier. Agents 4 through 6 represented users with well defined, but fairly complex patterns of behavior. This type of consistency allowed fairly steady improvement on the part of their agents.

At two points in the experiment, once about 1/3 of the way through and again about 2/3 of the way through, the agents were asked to collaborate to suggest a time and date within a one-week range for a meeting of the entire group. In both cases they successfully came up with at time which all users found acceptable. In the first case, the time was truly convenient for all the participants. In the second case, the time was less convenient for two of the participants, but they were able to accept it. We were able to confirm with a manual check that in that case there were no times available which would have been more convenient for all the participants.

FUTURE WORK

Currently work is underway on a graphical user interface to the scheduler software. Once this is completed, actual field tests will be performed. The data from these tests will be used to drive future development.

Much of the future work on this project will focus on improving the learning algorithms. First, we would like to add support for *learning by being told* (or *programming by example*. There is already a limited form of this available in the commands which allow the user to edit the priority weightings being maintained by the agent. We also plan to allow the user to instruct the agent by adding to the memory hypothetical situation/action pairs, possibly including "don't cares" in some of the feature slots. Additional support for this type of learning would allow a user to also directly adjust the weighting given to particular features of the stored situations.

Another form of learning by being told would be allowing the user to tell the agent that particular parts of his or her recent behavior are going to have less predictive value in the future. This will be necessary in an environment like a university, where schedules and thus particular time preferences change frequently, but a person's more general types of preferences are less likely to change.

We would also like to eventually allow a user to specify some particular features for the agent to consider. The most open-ended way of doing this would be to allow the user to write some LISP code, but perhaps a way of specifying some of the more likely types of features can be devised for users who do not wish to program their agents directly in LISP.

Another option for improving the set of features examined, perhaps used in addition to user-programmed features, is automatic feature generation. Some work in this area has been done in [3]. This could be useful in cases where the importance of a field depends not only on the value in it, but also on the value of some *other* field. For example, there are a number of fields which contain information about conflicting meetings, which are only relevant when the number of conflicts (another field) is at least one. Automatic feature generation could be used to create a new feature which is an appropriate combination of the related fields.

We also intend to work on improving the way the values of the features are interpreted. The algorithm as discussed in [10] expects there to be a relatively small number of distinct possibilities for each field, whereas in this application, several of the fields have numerical values which can fall in a rather wide range (for example the amount of time that week which is already scheduled for meetings). Currently this is being dealt with by breaking the range into subranges, so that there

are a relatively small number of subranges a value can fall into. This was done manually in this version of the implementation, but should be something which can be done automatically.

Finally, we would like to test whether a gradual decay of older actions in the memory produces better results than having a sharp cut-off (i.e. older examples being deleted when the size of the memory exceeds a given maximum).

RELATED WORK

Learning agents such as the one discussed in this paper are related to the work on *demonstrational interfaces* [1, 6, 8, 9]. The ways in which these differ from the approach of a learning agent are discussed in [7].

The work presented in this paper is also related to a similar project under way at CMU. Dent et. al. [2] describe a personal learning apprentice which assists a user in managing a meeting calendar. So far their experiments have concentrated on the prediction of meeting parameters such as location, duration and day-of-week. Also they report results that show that meeting scheduling assistants which use machine learning techniques are very promising and often out-perform hand-coded systems. Their apprentice uses two competing learning methods: a decision tree learning method and a back-propagation neural network. One difference between their project and ours is that memory-based learning potentially makes better predictions because there is no "loss" of information: when suggesting a decision, the detailed information about individual examples is used, rather than general rules that have been abstracted beforehand. On the other hand, the memory-based technique requires more computation time to make a particular suggestion (which we don't consider to be a problem because scheduling decisions do not have to be made "right away"). An advantage of our approach is that our scheduling agent has an estimate of the quality or accuracy of its suggestion. This estimate can be used to decide whether the prediction is good enough to be offered as a suggestion to the user or even to automate the task at hand (by setting two different thresholds on the agent's confidence in its decision).

CONCLUSION

We presented an intelligent agent which learns to assist an individual user in scheduling group meetings. The data presented show that the agents' guesses as to which action to take in a particular scheduling situation gradually improve as they have had more time to observe their user and receive user feedback. It can be concluded that using Machine Learning techniques is a promising way to build agents which become gradually more helpful to their users and at the same time can be trusted by their users.

ACKNOWLEDGMENTS

We would like to thank Thuy (Cecile) Pham for her work on the graphical interface to the scheduling application. We would also like to thank Robert Ramstadt for his helpful comments on an earlier draft of this paper.

The first author is an NSF fellow and has been partially supported by both the MIT Media-Lab and the MIT AI Lab. Additional resources for this research have been provided by a grant from Apple Computer, Inc.

REFERENCES

[1] Cypher, A. EAGER: Programming Repetitive Tasks by Example. In Proceedings of CHI, 1991 (New Orleans, Louisiana, April 28 - May 2). ACM, New York, 1991, pp. 33-39.

[2] Dent L., Boticario J., McDermott J., Mitchell T. and Zabowski D. A Personal Learning Apprentice. Submitted to the 1992 National Conference on Artificial Intelligence. 1992.

[3] Fawcett, T., and Utgoff, P. Automatic Feature Generation for Problem Solving Systems, (COINS Technical Report 92-9). University of Massachusetts, Department of Computer and Information Science, Amherst, MA, 1992.

[4] Kay A. Computer Software. Scientific American 251, 3 (March 1984).

[5] Laurel B. Interface Agents: Metaphors with Character. In: B. Laurel (ed), The Art of Human-Computer Interface Design. Addison-Wesley, 1990.

[6] Lieberman, H. Capturing Graphical Expertise Interactively by Example. To be published in Proceedings of the International Center for Scientific and Technical Information (Moscow) Workshop on Human-Computer Interaction (St. Petersburg, Russia, August 1992).

[7] Maes, P. and Kozierok, R. Learning Interface Agents. Submitted to INTERCHI'93 (Amsterdam, The Netherlands, April 25-29) ACM, 1993.

[8] Myers, B. and Buxton, W. Creating Highly Interactive and Graphical User Interfaces by Demonstration. In Proceedings of SIGGRAPH 1986, Vol. 20, No. 4. (Dallas, TX, Aug. 18-22) ACM, 1986.

[9] Myers, B. Creating User Interfaces by Demonstrations. Academic Press, 1988.

[10] Stanfill C. and Waltz D. Toward Memory-Based Reasoning. Communications of the ACM 29, 12 (Dec. 1986), 1213-1228.