A Learning Approach to Knowledge Acquisition for Intelligent Interface Agents

by

Robyn Arlene Edelson Kozierok

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1993

Author ....
Department of Electrical Engineering and Computer Science
May 17, 1993

Certified by ....
Patricia E. Maes
Assistant Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by ....
C. L. Searle
Chairman, Departmental Committee on Graduate Students

A Learning Approach to Knowledge Acquisition for Intelligent Interface Agents

by

Robyn Arlene Edelson Kozierok

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 1993, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Interface Agents are semi-intelligent, semi-autonomous computer programs — personal assistants that help their users in dealing with computer applications. They make use of knowledge about the tasks, habits and preferences of their users to simplify the use of the application by automating tasks on their users' behalf.

A difficult problem in the design of Interface Agents is that of Knowledge Acquisition. Forcing either a knowledge engineer or the end-user to program the agent can work in some situations but is very limited. A learning approach is indicated, yet no single learning technique is truly sufficient for an application of this complexity. An integrated approach is presented here, involving Memory-Based Reasoning augmented by Significance Feedback Learning and Rule-Based Induction.

This thesis discusses a particular Learning Interface Agent which has been implemented using this integrated approach — a Meeting Scheduling Agent. Complete testing of this agent would have required it to be used in a meeting-intensive environment for a period of several months — a requirement which it was not practical to meet. However, preliminary testing of the agent was performed in a variety of ways, which demonstrated the potential for the Meeting Scheduling Agent to learn a user's preferences within a reasonable amount of time, provided that the user's criteria for making a scheduling decision are things which the agent has access to.

The results suggest that meeting scheduling is a task of reasonable complexity for an agent of this sort; however, because of the time required to accumulate a sufficient number of examples for the agent to really learn the user's habits, this approach to developing agents may be more appropriate for applications in which a larger number of examples occur within a shorter time frame. An important but surprising result which arose from the testing was that simplistic rules can actually hinder the agent's learning by misleading it in many situations. It will be important to carefully educate users about the effective use of rules when facilities to enter them are provided.

Thesis Supervisor: Patricia E. Maes
Title: Assistant Professor of Media Arts and Sciences

Acknowledgments

I would like to gratefully acknowledge the help of the many people who have assisted me, both academically and personally, in the completion of my thesis.

First and foremost, I'd like to thank my thesis supervisor, Pattie Maes, for her assistance, guidance, support and availability throughout all phases of this project.

Second, very grateful thanks to UROP student Thuy (Cecile) Pham, without whose implementation of the graphical user interfaces for both the Meeting Scheduling Application and the Agent itself I could never have finished this thesis. I'd also like to thank Nick Cassimatis, Douglas Alan and Leonard Foner who helped with the implementation and debugging of the electronic mail interface allowing communication between the agents.

I'd like to offer special thanks to the other members of the Agents group at the MIT Media Laboratory, who, in addition to being a great bunch of people to work with, also agreed to use the Meeting Scheduling Agent for the last six weeks of this semester. Thanks to Pattie Maes, Henry Lieberman, Bruce Blumberg, Abbe Don, Leonard Foner, Beerud Sheth, Todd Drake, Max Metral, and Cecile Pham.

I'd also like to thank Max Metral and Beerud Sheth for providing helpful comments on an earlier draft of this thesis, Dan Ellis for help with tweaking PostScript figures, and Fred Martin, Tom Maglione and Trevor Darrell for their assistance in getting my PostScript figures incorporated into this LaTeX document.

On the financial end of things, I'd like to acknowledge the National Science Foundation, which supported me personally in the form of an NSF Graduate Research Fellowship, and supported the research agenda of which this work forms a part with grant number IRI-9205668. I'd also like to thank Apple Computer, Inc. for their donation of the equipment on which this work was implemented.

Finally I'd like to thank my family whose encouragement, love and support throughout my lifetime have helped to get me to where I am today. I'd especially like to thank my husband, Charles, whose love, friendship, companionship, understanding and support have enriched my life. His constant encouragement and confidence in

me have made especially these last few hectic months much more bearable. And a special thank you to my as yet unborn first child, who was kind enough to permit me an easy early pregnancy so that I could finish my thesis during it. I'm looking forward to meeting you in September.

Contents

List of Figures

Chapter 1

Introduction

As computers become more prominent in the home and work place, an increasing number of users come to rely upon them for more and more of their day-to-day tasks. This interaction tends to include a great deal of boring repetitive activities, such as deleting "junk" electronic mail (e-mail), filing away certain types of interesting news articles, or responding to meeting requests. In the name of efficiency and relief from tedium, it is desirable to be able to automate the most predictable of these actions.

One possibility that exists today is for users to write macros to automate truly repetitive tasks. Yet often a somewhat more flexible and/or easier to use solution is required. This is where *interface agents* can fill an important need.

An interface agent is a semi-intelligent, semi-autonomous computer program that 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 automate actions on their behalf.

In order for an interface agent to be useful and acceptable, it must satisfy the following criteria:

- **Competence:** The agent must be able to correctly predict the user's actions in order to automate tasks for him or her. In addition, the agent must recognize situations in which its competence is questionable, and defer to the user in these cases.

- **Trust:** The user must feel confident that the agent is in fact competent, as described above, and thus feel comfortable delegating tasks to it.

- **Helpfulness:** The user must feel that any effort that went into building or training the agent was worthwhile because of the savings in effort realized later.

One of the main problems in the actual implementation of Interface Agents is that of Knowledge Acquisition: How does the agent acquire the knowledge it needs to provide effective personalized assistance? The answer to this question is a significant factor in determining how well the agent can meet the above criteria.

This thesis argues for an integrated Machine Learning approach to the problem of Knowledge Acquisition. It presents an example of a Learning Interface Agent built in this way, integrating Memory-Based Reasoning, Significance Feedback and Rule-Based Induction.

1.1 Motivation

This thesis makes contributions to the field of Intelligent Interface Agents, and to the field of Machine Learning.

Contributions to the field of Intelligent Interface Agents

This thesis develops Pattie Maes' proposal to further explore the idea of a Machine Learning approach to Knowledge Acquisition for Intelligent Interface Agents. It does so by the implementation of a specific Learning Agent, namely the Meeting Scheduling Agent. This has led to the following contributions:

- This implementation has required investigation into what learning techniques would be appropriate for such an agent, and how they might be combined.

- The results from testing this agent confirm that the Machine Learning approach to Knowledge Acquisition is indeed a promising approach which should continue to be investigated further.

- Finally, the experience of implementing this agent has suggested various directions for future research, as presented in Section 7.3.

Contributions to the field of Machine Learning

This thesis also makes contributions in the area of Machine Learning:

- It validates the use of Memory-Based Reasoning [15] for an incremental learning task.
- It expands upon the work of Stanfill [16] in incorporating Rule-Based Induction into the Memory-Based Reasoning paradigm.
- It suggests a mechanism by which Memory-Based Reasoning, Significance Feedback Learning and Rule-Based Induction can be combined within a single system, allowing the system to share the advantages of all of these techniques.

1.2 Organization of this Thesis

The next chapter discusses contemporary approaches to Knowledge Acquisition, as well as our learning approach. It examines how well each approach deals with the three requirements introduced above: **competence**, **trust** and **usefulness**.

Chapter 3 introduces the Meeting Scheduling Agent, the particular Learning Interface Agent implemented for this thesis. It describes the functionality of the agent, from the user's point of view.

Chapter 4 describes the underlying algorithms used in the implementation of the Meeting Scheduling Agent.

Chapter 5 discusses issues involved in combining rules and examples within the Memory-Based Reasoning Paradigm.

Chapter 6 contains the results and analysis of tests run to evaluate the Meeting Scheduling Agent.

Chapter 7 describes related work and contains conclusions based on the experience of implementing, and upon the results of testing the Meeting Scheduling Agent.

It also contains suggestions for directions of future work.

Chapter 2

Knowledge Acquisition for Interface Agents

One of the most important obstacles in the quest to develop competent, trustworthy and helpful Interface Agents is that of Knowledge Acquisition — somehow the agent must gain access to the information which it requires in order to assist the user effectively.

In order for an agent to perform competently, it requires some way of determining what the user would want it to do in a particular situation. This requires knowledge at least about the user, and sometimes also about the particular application the agent is assisting the user with.

The other aspect of competence is recognizing when a prediction is weak, or being able to compute a confidence level in any prediction. This ability is closely tied to the Knowledge Acquisition strategy, as how the knowledge got there is a major factor influencing how likely it is to be correct. The knowledge acquisition strategy also determines what facilities the agent may have for judging the correctness of its own knowledge.

How trustworthy an agent is is largely dependent upon its competence; however it is also greatly influenced by how good the the user's understanding of it is. In addition to affecting the agent's competence, the choice of Knowledge Acquisition strategy also determines how well an agent can foster this very important understanding, thus having a significant impact on the user's level of trust in the agent.

In evaluating the helpfulness of an agent, we must consider both user effort in training the agent, and the amount of benefit the user is finally able to derive from using the system. The latter is again closely related to competence, while the former is probably the factor which is most directly influenced by the choice of an approach to Knowledge Acquisition, as this decision determines the degree to which the user is expected to help the agent acquire the knowledge it needs.

This chapter contrasts contemporary approaches to Knowledge Acquisition with our learning approach, in terms of how well each succeeds in meeting the requirements of competence, trust and helpfulness.

2.1 Contemporary Approaches

Contemporary approaches to the problem of knowledge acquisition have focused on programming the agent. Either a Knowledge Engineer programs it in advance with a hopefully comprehensive set of rules (cf. [1, 7]), or the end-user programs it himself or herself in order to create a truly personalized assistant (cf. [9]).

2.1.1 Knowledge-Based Approach

In the "knowledge-based" approach, the agent is endowed by its creator with a great deal of knowledge about the application and typical users (the *domain model* and *user models*, respectively). This is currently the most popular approach taken by the creators of interface agents [17]; however, no commercial products have yet appeared which employ this technique.

An example of an agent built using this approach is Chin's UCEgo [1]. This program is a help system for the UNIX operating system that uses built-in knowledge to help recognize a user's plans and provide appropriate help, by either answering the user's questions or volunteering information to help correct perceived misconceptions on the part of the user. UCEgo has a great deal of knowledge about UNIX (domain model) and its typical uses (user models), which it can use in deducing users' goals, allowing it to provide such help.

In this approach the agent is knowledgeable from the start, and can assist very naive users immediately. However, this approach requires a huge investment of time and expertise to build the domain and user models. Furthermore, once built, agents designed in this manner are neither customizable to individual users' (possibly changing) requirements, nor adaptable to other domain areas.

Competence

The agents' competence in this approach is determined by the ability of the Knowledge Engineer to predict in advance all situations in which the agent will find itself, and the correct action to take in each of these situations.

It is thus very difficult to develop a competent agent using this approach unless the application domain is very simple and well-defined, and all the users' actions fall into easily recognizable patterns of behavior.

The need for the agent to be able to recognize a weak prediction is minimal in this scenario. It is incumbent upon the Knowledge Engineer to determine in advance the situations for which rules can be developed, and those for which they cannot.

Trust

In this scenario, the agent is a Black Box. The user has no idea what the agent's rules are, and thus has little reason to trust such an agent until he or she has some experience with it. Unfortunately, having to gain the required experience while working with an agent one cannot trust is quite undesirable.

Helpfulness

The user has little effort invested in this agent. Provided that it does prove to be sufficiently competent and trustworthy, it will also probably be perceived by the user as being helpful, even if it is only able to assist with a few tasks; however, as discussed above, the likelihood of devising a competent and trustworthy agent of this type is small.

2.1.2 User-Programming Approach

At the other end of the spectrum we find the user-programming approach. In this case, the user programs agents to perform tasks he or she wishes to have automated. This is similar to writing macros, but can be more flexible due to the availability of a more expressive language.

For example, Malone and Lai's Oval system [9] allows the user to create "semi-autonomous agents" consisting of a collection of rules for processing information related to a particular task.

As this approach is the easiest for a developer to design and implement, commercial agents employing this approach are beginning to appear on the market. For example BeyondMail^{TM} is an electronic mail system which allows the user to specify rules for sorting, prioritizing and discarding incoming electronic mail. It also allows one to arrange for mail to be automatically answered with a canned message while on vacation. Magnet^{TM} is another user-programmed agent which allows the user to automate file-related tasks on the Macintosh, such as copying or moving files fitting a given profile, doing directed back-ups, cleaning up the desktop, and keeping track of shared files to ensure that the user has the most recent version.

Systems such as this one are extremely flexible. For example, an Oval user could create a personal mail sorting agent by generating rules that sort incoming mail messages into different folders, depending on various characteristics of the header information. The problems with this approach are that the user must recognize the opportunity to use an agent, have the motivation and ability required to program it, and take responsibility for its maintenance.

Competence

The competence of agents developed in this manner depends upon the user's ability to program them and the system's ability to support the user in this task. In most cases a very skilled user would be able to create competent agents in this type of system. However, in some cases a user may be unaware of all the true rules he or she uses in making certain types of decision, and thus even if skilled may end up being unable to create a sufficiently competent agent to handle this type of situation.

The issue of recognizing weak predictions is again less important with this approach, as the user sets up rules specifying the situations in which the agent should take control, and those in which he or she wishes to be deferred to.

Trust

This approach permits a great degree of trust (providing that the user trusts his or her own programming ability), as the user determines exactly how the agent is supposed to behave. However, even in this type of system, there is sometimes doubt as to how the user's instructions will be interpreted by the agent or what it will do if ambiguity arises, because the user generally programs the agent in a language that utilizes a high level of abstraction. It may also be difficult for the user to test his or her programs (i.e., agents) before letting them loose on real data. In this case the user may still be hesitant to trust it. (See [11] for an example of just such a misunderstanding which arose in Steven Levy's first use of the Magnet^{TM} product, causing a temporary panic until he found his missing files sitting in his Macintosh's virtual trash can.)

Helpfulness

The user must put a great deal of effort into developing such an agent, thus it will need to save the user a lot of time in order for it to be considered worthwhile to the user.

2.2 A Machine Learning Approach to Knowledge Acquisition

Professor Pattie Maes, of the MIT Media Laboratory, has argued that neither of the contemporary approaches is truly satisfactory, and has proposed a research agenda to explore a third possibility — having the agent learn the knowledge it needs to effectively assist the user.

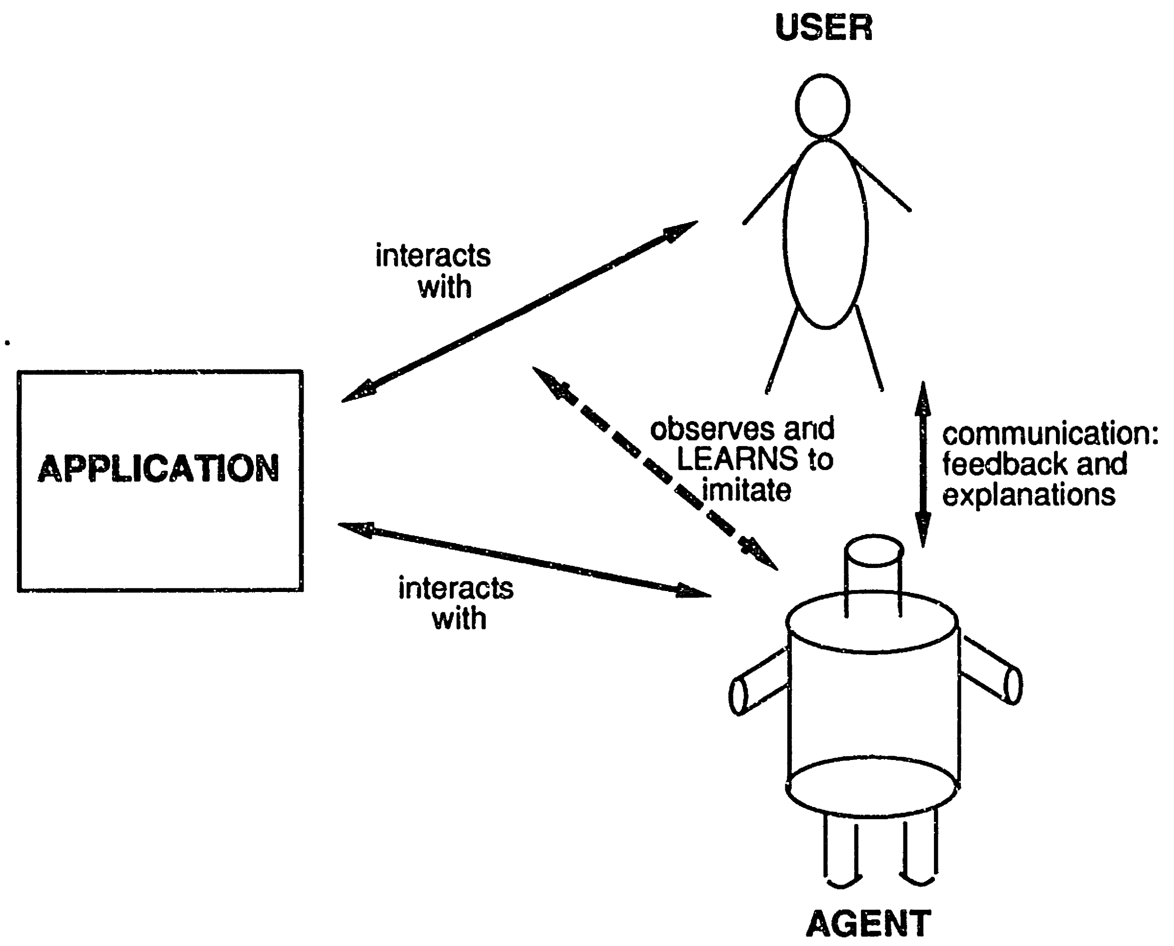

USER
interacts with
APPLICATION
observes and LEARNS to imitate
communication: feedback and explanations
interacts with
AGENT

Figure 2-1: Interaction between Agent, User and Application

In our conception, there is one agent per user, able to learn the complex preferences of that particular individual. In the current conception we also envision different agents for different tasks, but in the future, these could conceivably be combined into one overall personal assistant.

The agent learns the user's behavioral patterns by observing the user's interaction with the application, as shown in Figure 2-1. When it is confident that it knows what the user would do in a situation, it can automate that action for the user, by interacting with the application in the ways it has observed the user doing so. It can receive feedback from the user when its prediction is incorrect (regardless of whether or not that prediction was acted upon). It can also provide explanations of its predictions to the user.

Although it is called an "Interface Agent," as shown in this figure, the agent is not an interface between the application and the user. The user may still use the application directly, making only as much use of the agent as he or she desires.

Competence

A learning agent gradually builds up competence, learning what to do in a particular situation by observing and memorizing what the user has done in previous similar situations. Clearly this will only be possible in applications in which there is a fair degree of regularity and repetitiveness — if each situation is entirely independent from all prior ones, it will be impossible for the agent to make good judgments based on this type of knowledge acquisition.

However, the advantages of a learning approach are many. In addition to requiring less overhead of either a knowledge engineer or the user to program them, learning agents are able to learn very subtly complex and individualized rules — rules of which even the user himself or herself might not have been consciously aware.

In a learning scenario, some scheme is necessary whereby the agent is able to determine to what extent it has acquired the knowledge required to make a particular prediction. We accomplish this by having the agent compute a confidence in its prediction, which is made possible by the main type of learning algorithm we use, namely *Memory-Based Reasoning* (MBR) [15]. This is discussed in greater detail in Section 4.1.3. The agent makes use of user-provided thresholds (discussed further in Section 3.2.2) to determine when that confidence is great enough to take the action, and when the user should be deferred to instead.

Trust

Because the agent learns everything by observing the user, it is easier for the user to come to trust such a system than one in which, for example, rules had been provided in advance by an unknown knowledge engineer.

The agent gains its competence gradually. Our goal is to have it earn the user's trust over that same time period. In order to foster this gradually improving trust relationship between user and agent, we provide a means by which the user can provide confidence-level thresholds to control the level to which the agent takes autonomous actions on the user's behalf. We also provide a means by which the agent can demonstrate its progress to the user on a continuous basis in a relatively unobtrusive manner. Finally, we endow the agent with the ability to explain its suggestions and actions, permitting the user to understand it better, which will hopefully help to foster a greater degree of trust. The specific mechanisms by which these are accomplished in the Meeting Scheduling agent are discussed in Section 3.2.2.

Helpfulness

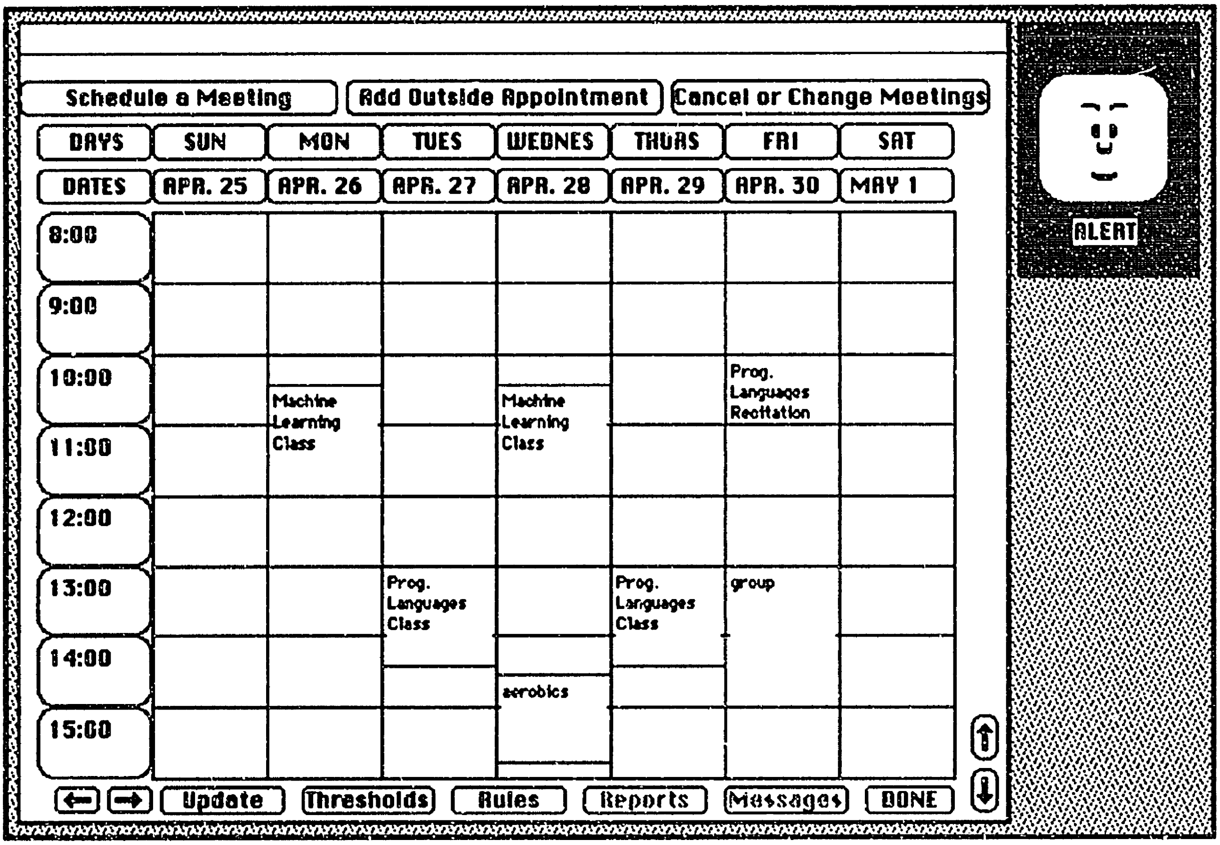
The investment made in training a learning agent falls somewhere between that of a Knowledge-Engineer-programmed agent (i.e., none) and a user-programmed agent (i.e., significant). Training a learning agent takes mainly patience — it takes very little additional effort on the part of the user, but does take time. It also takes a certain amount of monitoring so that the user feels confident about setting appropriate thresholds to allow the agent to take autonomous actions.

The main idea in accomplishing the goal of helpfulness is to make the agent as unobtrusive as possible during its training, so that whatever help it manages to provide (and after a while it will be significant) comes at small enough a cost to make it immediately worthwhile.

Chapter 3

A Meeting Scheduling Agent

The remainder of this thesis discusses the implementation of a Learning Interface Agent for Meeting Scheduling. This agent runs alongside a graphical calendar application (see Figure 3-1) and learns the user's scheduling preferences.



DAYS	SUN	MON	TUES	WEDNES	THURS	FRI	SAT
DATES	APR. 25	APR. 26	APR. 27	APR. 28	APR. 29	APR. 30	MAY 1
8:00							
9:00							
10:00		Machine Learning Class		Machine Learning Class		Prog. Languages Recitation	
11:00							
12:00							
13:00			Prog. Languages Class		Prog. Languages Class	group	
14:00				aerobics			
15:00							

Figure 3-1: The Calendar Application (with the Agent Alert and Ready to Learn)

When a meeting request arrives, the agent makes a prediction as to what action the user will take (i.e., accept the invitation, decline it, request renegotiation of the suggested time, accept it and skip or reschedule a conflicting meeting, etc.). Depending on its confidence in its prediction, the agent will either do nothing, suggest that action to the user, or take it autonomously, and prepare a report for the user describing the action which was taken.

If the agent does not take the action autonomously, it memorizes the action the user chooses in that situation. If it does take an autonomous action, it adds that action to its memory of examples when the user approves of the action described in the agent's report. Here approval consists of not changing the action after seeing the report — if the user does change the action the user's newly chosen action is memorized instead. As the agent memorizes more and more correct actions, it becomes more accurate in subsequent predictions.

3.1 The Calendar Application

The Meeting Scheduling Agent is intended to run alongside a top of the line calendar application, such as Meeting Maker^{TM}. However, in order to allow an agent the access it needs, such an application would need to be both "recordable" (meaning the agent could learn what actions are taking place within the application) and "scriptable" (meaning the agent could take actions in the application directly). Since currently available commercial calendar software is not written to provide these facilities, it was necessary to implement a calendar application from scratch as part of this project. Although care was taken to make the application usable and convenient, designing and implementing the calendar application was a minor part of this research, and thus this application does not compare favorably with those currently on the market. (Work is currently underway in our group to develop a protocol by which agents could communicate with any appropriately instrumented Macintosh applications by means of Apple Events [13]. This will make such implementations unnecessary in the future.)

Both the calendar application and the agent were implemented on a Macintosh Quadra 900 in Macintosh Common Lisp with CLOS. The calendar application takes advantage of the Macintosh's Ethernet connection to access the UNIX SMTP server and file system to send and receive mail.

The calendar application supports regular weekly, biweekly and monthly meetings, in addition to one-time-only meetings. It supports meetings scheduled within the system (internal meetings) as well as two types of meetings scheduled outside the system: external and personal meetings. (The types of meeting scheduled outside the system are provided as a convenience to the user — they do not have any *a priori* meaning to the application, though the agent can "learn" differences in the user's treatment of the two types, if applicable.) To initiate a meeting, the user specifies the details of the meeting (date, time, length, participants, description) and the application sends mail to the other users who were invited. Users receive e-mailed meeting invitations and choose a response. If the new meeting does not conflict with an existing appointment, the user may choose to **accept** the meeting, **decline** it, or **request renegotiation**, that is, appeal to the initiator of the meeting to select a more convenient meeting time. If the new meeting does conflict with one or more existing meetings, the user's choices become: **decline**, **request renegotiation**, or **accept and skip conflict** (simply don't show up to the conflicting meeting(s)), **accept and cancel conflict** (cancel the conflicting meetings if initiated by the user, otherwise send a message to the meeting initiator changing one's prior accept to a decline), **accept and renegotiate conflict** (renegotiate the meeting time(s) of conflicting meeting(s) if initiated by the user, otherwise send a message to the meeting initiator requesting renegotiation), or **partial accept** (inform the initiator that the user will be able to attend only part of the meeting, as the rest of it conflicts with another). Screen snaps of the application showing these choices are found in Figures 3-4 and 3-5.

Depending upon the responses the initiator gets to the invitation, and upon other demands on the initiator's schedule, he or she may at any time cancel or renegotiate the meeting, where renegotiating means selecting a new date and time for the meeting.

In the current implementation of the calendar application, all of the participants in a meeting must be using the application. A more sophisticated version could permit semi-structured electronic mail to provide an interface to people outside the system.

3.2 The Meeting Scheduling Agent

This section introduces the basic architecture and functionality of the Meeting Scheduling Agent. The details of the algorithms used to implement the agent are discussed in the next chapter.

The agent starts out with extremely minimal knowledge about the calendar application, and learns everything else it needs to know. Initially it is given a list of features of a meeting scheduling situation, and a list of possible actions a user might take. It has a partial ordering on the "positiveness" of the actions, but does not know which actions are appropriate or even permitted in a particular situation.

3.2.1 The Agent Architecture

A diagram of the agent architecture is shown in Figure 3-2.

The agent's learned knowledge is contained in two databases. The main one is an example base of situation-action pairs that represent all the actions the agent has observed. The other is a database of weights assigned to other users and meeting topics. These weights are used to interpret the raw information contained in the example base. For example, the meeting initiator is stored as part of the example, and it is possible to use the initiator's weight in the auxiliary database to determine the initiator's importance, which may in some cases predict the user's behavior more consistently than the specific identity of the initiator.

For the calendar application, a situation consists of a meeting invitation. Currently thirty-two features of such a situation are considered, including such things as day of the week, time of day, number of participants, initiator's importance, how busy your schedule for that day was before the meeting request arrived, whether there is already a conflicting meeting in your schedule, and how the average and total

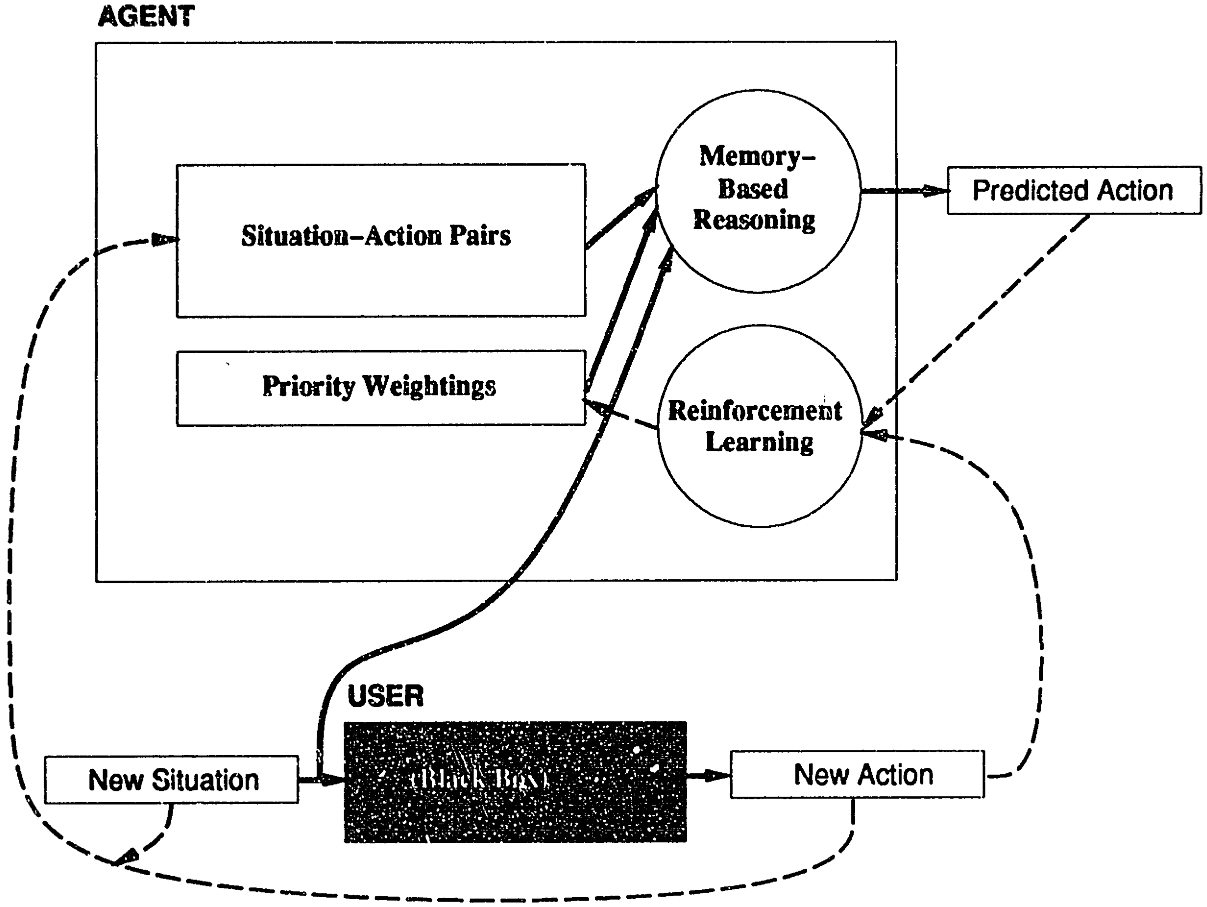


Figure 3-2: The Agent Architecture

importances of the participants in this meeting compare with those in a conflicting meeting. The full list of features used in the current implementation may be found in Appendix A.

As shown in Figure 3-2, the example base is updated and analyzed using Memory-Based Reasoning, a nearest-neighbor technique whereby a current action is predicted on the basis of previously memorized situation-action pairs. The database of weights is updated using Significance Feedback Learning, wherein weights are increased or decreased by a small amount each time feedback is given to suggest they are incorrect in the relevant direction. Details of these learning algorithms are given in Chapter 4.

3.2.2 Fostering a Trust Relationship

A major factor contributing to a user's trust relationship with a computer program must be a feeling of ultimate control. To this end, I allow the user to set two thresholds on the confidence level the agent has in its prediction. A "tell-me" threshold defines the confidence level above which the agent will tell the user what its prediction was and request additional feedback if its prediction was incorrect; a "do-it" threshold sets the confidence level above which the user authorizes the agent to take actions autonomously. These thresholds may be set on a per-situation basis — they may depend upon the action that the agent is predicting and the number of days before the meeting is scheduled to take place. In this way, the user may allow some (perhaps easier to recover from) actions to be taken at lower confidence levels than others. This gives the user control over when and in how far to entrust the agent to act on his or her behalf.

In order to help the agent gain the user's trust as it gains competence, I provide a means by which it can show the user its progress, in a way that is not too intrusive to the user. This is done by means of caricature faces which display the state of the agent, allowing the user to determine this state at a glance, as shown in Figure 3-3. When the agent makes a prediction, a caricature face is used to show it's degree of certainty, relative to the two user-set thresholds. Once the user takes an action (or reacts to an action taken autonomously by the agent), a new expression is used to show whether or not the agent's prediction (or action) was correct.

The user may make use of a relatively low "tell-me" threshold to observe the confidence levels at which the agent consistently makes correct predictions. However, with the caricature faces indicating the agent's success or failure on each prediction, the user can gain a feeling for how often the agent is correct, even when the "tell-me" threshold is high.

To further ensure that the agent understands and begins to trust the agent, a facility is provided by which the agent can explain its prediction to the user if asked to do so (as shown in Figure 3-6). This is again made possible by the Memory-Based Reasoning algorithm used as the agent's main learning technique, and is discussed in

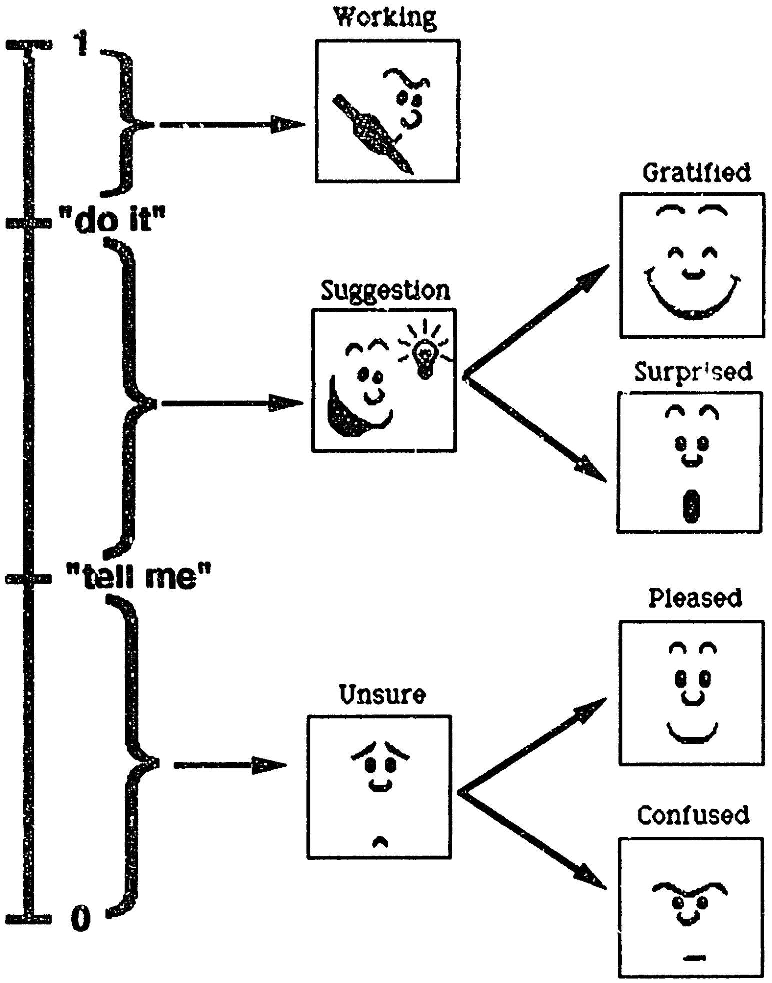


Figure 3-3: Caricature Faces Displaying the Agent's Current State

greater detail in Section 4.1.4. The user may also request that the agent's prediction and confidence be displayed in any particular situation (i.e., even if the confidence fell below the "tell-me" threshold) by simply clicking on the agent's caricature face.

3.2.3 Handling Meeting Requests

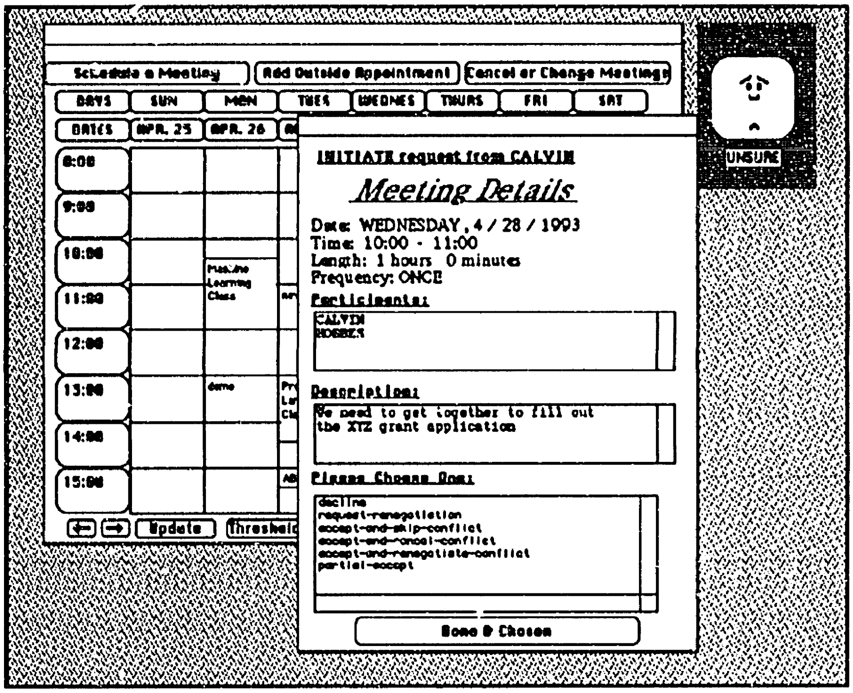
Whenever a new meeting request arrives, the agent intercepts it and makes a prediction as to what the user's response would be. Along with each prediction, the agent computes its confidence in the prediction. It then compares its confidence level to the "tell-me" and "do-it" thresholds set by the user.

If the confidence is above the "do-it" threshold for that type of situation, it takes the predicted action and prepares a report so that the user can learn what actions were taken on his or her behalf. If not, the agent queues the message for the user (remembering its prediction, confidence level and reasoning so that it may reveal these to the user if appropriate). The queued messages and reports are presented at the user's convenience.

Queued messages consist of items of information for the user, such as replies received from others to meeting invitations the user had issued, and new invitations that the agent was not authorized to reply to autonomously, by virtue of the fact that its confidence was below the "do-it" threshold. If the agent's confidence was also below the "tell-me" threshold, the agent will appear "unsure" of what to predict, and will simply present the user with a dialog containing the details of the invitation and the reply choices available (as determined by the calendar application). This is shown in Figure 3-4.

If the agent's confidence was above the "tell-me" threshold when making a prediction for a particular request, the prediction will be displayed to the user along with the dialog containing the details of the invitation and the replies to choose from, as shown in Figure 3-5. In this case, if the user makes a choice other than the predicted one, the agent will request feedback from the user. If the action the user takes is more "positive" than the one the agent predicted (the agent is provided with a partial ordering on actions as part of its initial knowledge), the agent will ask whether any of the initiator, the participants or the keywords were more important than the agent had previously thought. (The user should answer in the affirmative if these factors had an impact on his or her selection of the more positive response than the one the agent had predicted.) If the user's action is less positive than the one the agent had predicted, the agent will first ask if it was just a bad time (in which case no change to the weights occurs). If not, the agent will ask whether any of the initiator, participants or keywords were less important than it thought. If the user does indicate that any of these things were more or less important than the agent thought, the significance feedback algorithm will increase or decrease their importance ratings accordingly.

When reviewing reports of autonomous activity taken by the agent, the user may



INITIATE request from CALVIN

Meeting Details

Date: WEDNESDAY, 4 / 28 / 1993
Time: 10:00 - 11:00
Length: 1 hours 0 minutes
Frequency: ONCE

Participants:

CALVIN
HOBBES

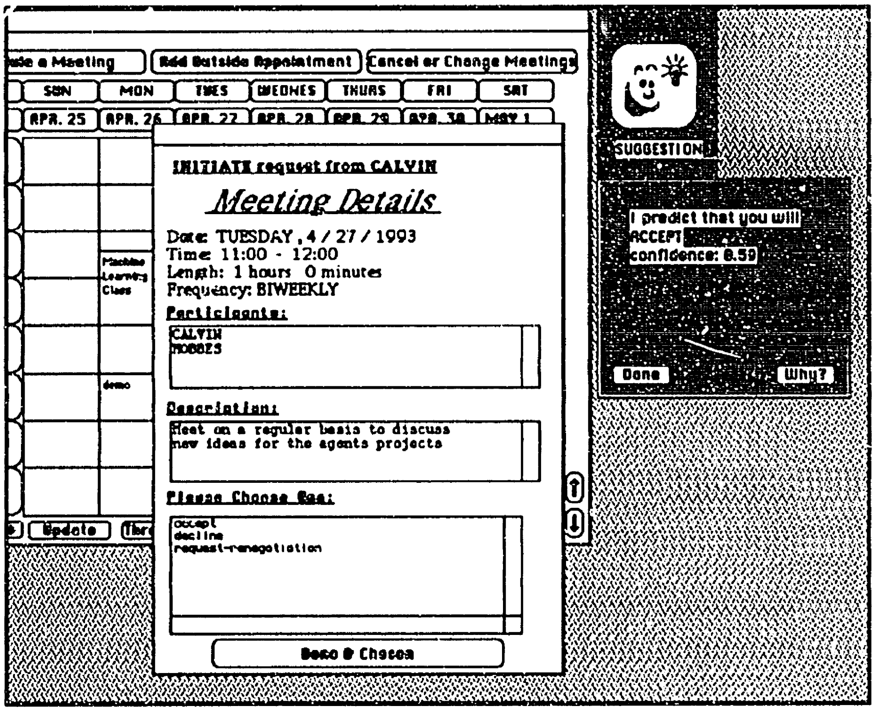
Description:

We need to get together to fill out the XYZ grant application

Please Choose One:

decline
request-renegotiation
accept-and-skip-conflict
accept-and-cancel-conflict
accept-and-renegotiate-conflict
partial-accept

Figure 3-4: The Agent is Unsure of its Prediction



INITIATE request from CALVIN

Meeting Details

Date: TUESDAY, 4 / 27 / 1993
Time: 11:00 - 12:00
Length: 1 hours 0 minutes
Frequency: BIWEEKLY

Participants:

CALVIN
HOBBES

Description:

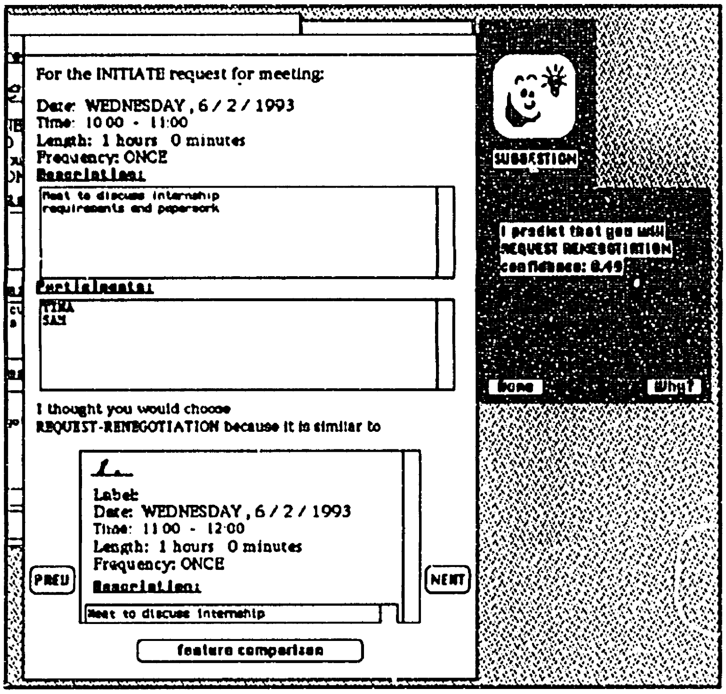
Meet on a regular basis to discuss new ideas for the agents projects

Please Choose One:

accept
decline
request-renegotiation

I predict that you will
ACCEPT
confidence: 0.59

Figure 3-5: The Agent Confidently Makes a Suggestion



For the INITIATE request for meeting:

Date: WEDNESDAY, 6 / 2 / 1993
Time: 10:00 - 11:00
Length: 1 hours 0 minutes
Frequency: ONCE
Description:

Meet to discuss internship requirements and paperwork

Participants:

TINA
SAM

I thought you would choose REQUEST-RENEGOTIATION because it is similar to

Label:
Date: WEDNESDAY, 6 / 2 / 1993
Time: 11:00 - 12:00
Length: 1 hours 0 minutes
Frequency: ONCE
Description:

Meet to discuss internship

I predict that you will REQUEST RENEGOTIATION
confidence: 0.49

Figure 3-6: The Agent Explains its Suggestion

change the action taken by the agent if necessary (causing a "changed reply" message to be sent out to the initiator). When this occurs, the agent also requests feedback as described above.

3.2.4 Accelerating Agent Training using Rules

Although the learning approach to knowledge acquisition has many advantages, agents built using this approach do take time to train. While users should not *have to* program rules into their agents, it is beneficial to be able to provide them with the *opportunity* to do so if they are so inclined.

Users who wish to take advantage of this facility may enter rules for their agents. Each rule consists of a user-generated hypothetical situation, with some or all of the details filled in, along with the action the user would have taken in this case. Each (possibly wild-carded) situation is paired with the given action and incorporated into the MBR Example Base, as shown in Figure 3-7. These rules may be specified to be either default or hard-and-fast rules. A description of how rules were included in the Meeting Scheduling Agent is found in Section 4.3. More details on the general issues

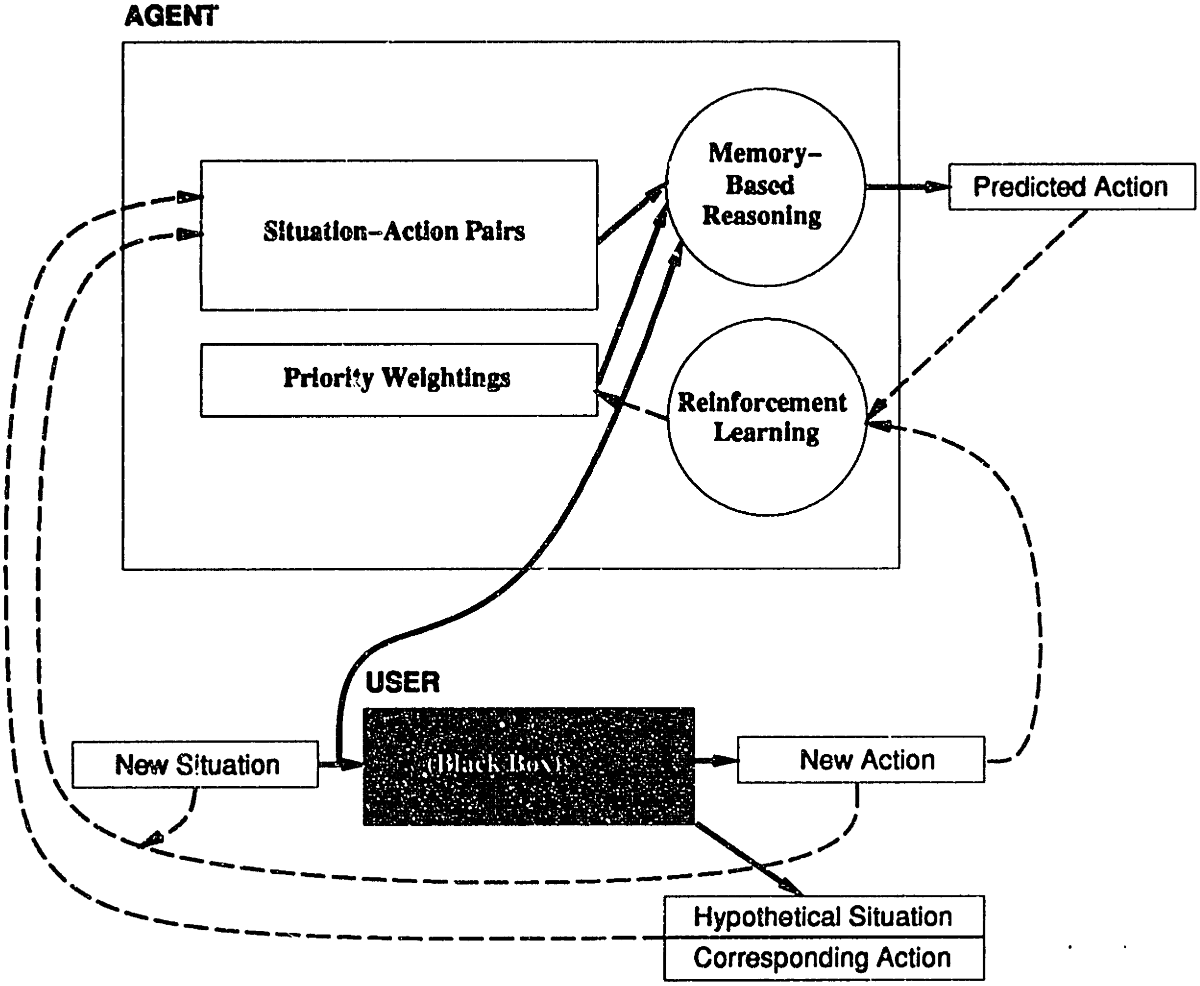
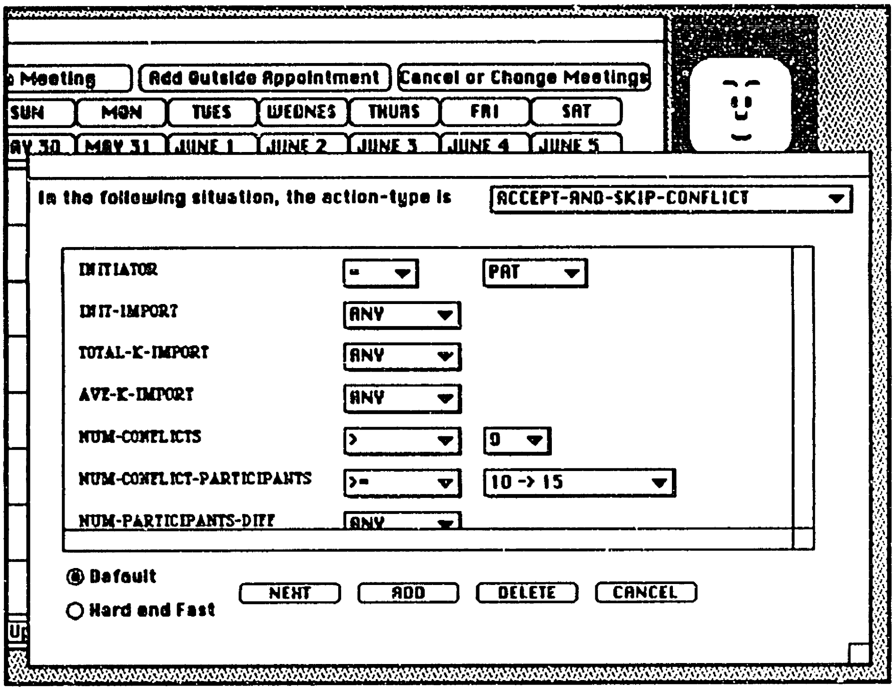


Figure 3-7: The Agent Architecture Incorporating Rules

involved in the integration of rules into the MBR paradigm are given in Chapter 5.

There are two levels of detail at which one can consider situations — there are the high-level details of the situation which are stored, and the lower-level features which are computed from these details (in a one-way transformation) for use in the Memory-Based Reasoning algorithm. In most cases the user would specify a hypothetical situation by setting some or all of the high-level details of the meeting request and the week it occurs in. Meeting details may be set using a form which closely resembles the dialog box for initiating a new meeting. Details of the surrounding week may be specified using an interface in which various blocks of time during the week can be set as either being free, or as containing internal, outside or personal meetings; or can remain unspecified.



In the following situation, the action-type is ACCEPT-AND-SKIP-CONFLICT

INITIATOR	=	PAT
INIT-IMPORT	ANY	
TOTAL-K-IMPORT	ANY	
AVE-K-IMPORT	ANY	
NUM-CONFLICTS	>	0
NUM-CONFLICT-PARTICIPANTS	>=	10 -> 15
NUM-PARTICIPANTS-DIFF	ANY	

● Default
○ Hard and Fast

Figure 3-8: The Low-Level Interface for Entering Rules

However, some features cannot easily be set using this type of interface. For example, it would not be possible for the user to indicate that there was one conflict with the new meeting, without specifying particular times for the meeting and the conflict. For cases like this the user is provided with an alternate interface by means of which it is possible to set the values of any of the lower-level features, such as number of conflicts, directly (see Figure 3-8). Both interfaces may be used together to specify a rule, with most details being entered on the high-level interface, and then a few details tweaked at the lower-level (but once the lower-level is used the user cannot go back to the higher-level as the transformation down to the lower-level features is not reversible).

[In the current implementation, the high-level rule interface has not yet been implemented, but the framework for allowing rules to be specified in either way is in place.]

3.2.5 Suggesting a Meeting Time

To take advantage of the agents that each user is training to his or her own preferences and habits, the initiator of a meeting may request that the agents suggest a convenient meeting time (and date) within date and time constraints (ranges) set by the initiator. When this happens the initiator's agent contacts the other invited users' agents and collects information from them in an attempt to suggest an optimal time for all users (with the preferences of users whom invitees considered most important as compared to themselves having greater impact on the decision than others'). The algorithm by which the agents predict an optimal meeting time is given in Section 4.4.

Chapter 4

Algorithms for the Meeting Scheduling Agent

The previous chapter described the Meeting Scheduling Agent from the user's point of view. This chapter provides details of the algorithms used in the implementation of this agent.

The Meeting Scheduling Agent learns by observing the user performing the task it is to assist with, remaining largely in the background until it has learned enough to become useful. This learning by observation is largely accomplished using the Memory-Based Reasoning technique discussed below. This technique is quite powerful, but is not ideally suited to all of the types of knowledge the agent needs to acquire, and can be slow in getting up to speed. Therefore, the agent's learning may be improved by receiving direct feedback from the user (Significance Feedback) and accelerated by being explicitly taught certain rules by the user in the form of hypothetical examples (Rule-Based Induction). The algorithms for these techniques are discussed in the first three sections of this chapter.

The final section of this chapter details the algorithm used to suggest an optimal meeting time based on the input of all the invitees' agents.

4.1 Memory-Based Reasoning

The main technique used for the agent's learning is a k-nearest-neighbors technique known as *Memory-Based Reasoning* [15]. This technique stores a large set (*example base*) of correspondences between situations and actions as learned from the training data. Whenever a new situation occurs, an action is selected by considering "closest" matches between the new situation and stored situations, and selecting the action which is most often and/or strongly associated with very similar situations (see Figure 4-1). The definition of similarity is based upon a weighted sum of distances between the values of a set of features defining the situation. Figure 4-2 shows some possible distances and weights for a few of the Meeting Scheduling Agent's features. (Note that it is possible for two values to have distance of zero even if they are not identical. This occurs when they both always predict the same actions.) The distances and weights for the various feature-values are based upon a statistical analysis of the example base which determines how consistently particular features correlate with the different possible actions.

This technique is particularly well suited to a learning agent because it lends itself relatively easily to the computation of confidence levels in the prediction, as discussed in Section 4.1.3.

Memory-Based Reasoning is also appropriate for our purposes because it provides a natural way of generating explanations of the agent's actions. Actions can be explained on the basis of the similar examples which the agent used to make its prediction, as discussed in Section 4.1.4.

A further advantage of Memory-Based Reasoning is that information is not lost or discarded — all of the information is retained, and is used to make increasingly more accurate predictions as time goes on and the amount of information available increases.

The distance computation algorithm given the next section comes directly from [15]. The action selection and confidence computations described in the following sections are my own.

4.1.1 Computing Distances between Old and New Situations

Each situation is represented as a set of values for various fields which represent different features of the situation (i.e., the number of participants in the meeting — see Appendix A for the full list of Meeting Agent Features). The distance between a new situation and a memorized situation (the situation part of a situation-action pair in the example base) is computed as a weighted sum of the distances between the values in each field, as follows:

$$\Delta(s_{new}, s_{mem}) = \sum_{f \in F} d_f(s_{new} \cdot f, s_{mem} \cdot f) w_f(s_{new} \cdot f)$$

where:

- F is the set of fields,
- s_{new} is the new situation and s_{mem} is the memorized situation,
- d_f is the field-distance between two given values in the field f,
- w_f is the field-weight to be assigned to the field f when it contains the given value, and
- for any situation s and field f, $s \cdot f$ represents the value in field f of situation s.

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. These are computed as follows:

$$d_f(s_{new} \cdot f, s_{mem} \cdot f) = \sum_{v \in V_g} \left(\frac{\|[f = s_{new} \cdot f][g = v]\|}{\|[f = s_{new} \cdot f]\|} - \frac{\|[f = s_{mem} \cdot f][g = v]\|}{\|[f = s_{mem} \cdot f]\|} \right)^2$$

$$w_f(s_{new}\cdot f) = \sqrt{\sum_{v \in V_g} \left(\frac{||[f = s_{new}\cdot f][g = v]||}{||[f = s_{new}\cdot f]||} \right)^2}$$

where:

- g is the goal field,
- V_g is the set of possible values for g, and
- the notation $||[f = s_{new}\cdot f][g = v]||$ represents the number of examples in the database where the field f contains value $s_{new}\cdot f$ and the field g contains the value v. (The notation where only the value of f is indicated represents the obvious analog.)

4.1.2 Action Selection

Once the distance between the new situaiion and all of the memorized situations has been predicted, a score is computed for each of the actions, A, predicted by the m closest memorized situations as follows:

$$Score_A = \sum_{s \in S_A} \frac{1}{\Delta(s_{new}, s)}$$

where:

- s_{new} is the situation for which an action is being predicted,
- S_A is the set of those memorized situations, among the m closest being considered, that predict the action A, and
- $\Delta(s_{new}, s)$ is the distance between the current situation s_{new} and the memorized situation s.

The action with the highest such score is selected.

4.1.3 Confidence Computation

Along with each prediction a confidence value is computed 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, d_{max} (the greatest distance which may still be considered "close"), with the same action as the predicted one,
- n_{other} is the maximum of 1 or the number of the closest m situations with distances less than d_{max} with different actions than the predicted one, and
- n_{total} is the total number of the closest m situations with distances below d_{max}. (If there is at least one situation with a distance less than d_{max} and a different action than the predicted one, then $n_{total} = n_{predicted} + n_{other}$.)

If the result of this computation 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). When $d_{other} = 0$ the confidence is set to 0. This occurs in the case when there are exact matches to two or more situations corresponding to different actions. When $n_{pred} = 0$ the confidence is also set to 0. This case arises when none of the situations with the same action as the predicted one have distances less than d_{max}.

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 are within distance d_{max} of the current situation.

4.1.4 Explanations

As mentioned above, one of the advantages of using a Memory-Based Reasoning algorithm 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. For example, the agent might say, "I thought you might want to take this action because this meeting and your current calendar are similar to the following situations we have experienced before."

If the user wants more details on why the given situations were considered similar to the current one, the agent can show the user which features were weighted heavily in the distance computation, and how similar each memorized value for that feature was judged to be to the value in the current situation.

4.1.5 Computational Complexity

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 ($O(n^2)$ where n is the number of examples in the example-base) procedure which may be done in batches if the application does not require immediate learning in response to every new action. (The new examples will be taken into account in the action selection stage of the algorithm, but the distances and weights for the various feature-values may be slightly off until the next update is run.)

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.

4.2 Significance Feedback

The agent also keeps a database of weights to be updated by Significance Feedback. For example, the Meeting Scheduling Agent stores weights for other participants, and for keywords within the meeting topic. Many of the features the MBR algorithm uses depend on these weights, as well as the raw information about the situation. For example, the "initiator importance" feature is computed by taking the initiator from the raw information, and looking up the user's rating of that person. By design, the features are recomputed in this way each time they are needed, rather than being fixed once and for all at the time the situation occurs. This allows the agent to take advantage of improved importance weightings to better interpret the details of what happened in past situations. This is appropriate in cases where there are "true" ratings which remain relatively constant, and which the agent is getting a better and better estimate of. In applications where the ratings would be more likely to change over time, so that the current values would not be appropriate when used in old situations, it would be better to fix the features' values when the situations occur.

When the agent makes a prediction with a confidence level greater than the "tell-me" threshold, but is incorrect, it can ask the user for feedback which will help it determine whether any of these weights should be adjusted. For example, the Meeting Scheduling Agent may ask the user, "Was the meeting topic more (or less, depending on whether the user took a more positive or more negative action than the agent had predicted) important than I thought?" and then update the scores for the keywords found in the meeting description accordingly.

The level of questioning was chosen to be reasonably unobtrusive, thus it does not hone in on which particular keywords were more or less important — all of the keywords get updated at the same time. This is not a limitation of the algorithm, however, but rather of the interface. It is quite conceivable that in some applications, more specific questions would be preferable (and tolerated). Another possibility is to have an option that allows the user to state that he or she wishes to assign credit specifically to certain keywords, and then request the detailed information only in those cases. A final possibility would be to permit the user to adjust the weights manually if desired.

4.3 Rule-Based Induction

The user may speed up the training of the agent by providing rules for it to use as a guideline. This allows the user to quickly bring the agent most of the way up to speed by specifying a few of the most important rules the user employs in deciding which actions to take in a given situation. The user can then allow the agent to learn the more subtle nuances of the desired behavior by observation. For example, a user might want to tell the agent up front that he or she does not like to meet before 10 a.m.. and that meeting requests from his or her boss are always to be accepted. (What the agent will do with a request from the boss for an 8 a.m. meeting will be determined by how these rules were implemented.)

Stanfill [16] introduced the idea of combining rules and examples within the MBR framework. He showed that pronunciation rules could be expressed within a Memory-Based paradigm by allowing wild-card values in some of the features of an example. He proceeded to implement a system that learned to pronounce English text using the Memory-Based approach with the examples augmented by pronunciation rules. In that work, the rules are stored separately from the training examples. Memory-Based induction is applied to a new situation using the example base and, in a separate induction, the rule base. The results are combined (the paper does not specify precisely how) to decide the output phoneme.

As Stanfill suggested, rules may be seen within the MBR paradigm as simply examples which contain wild-cards to indicate features in which the value does not matter. A simple extension of this idea is to allow wild-cards to either represent any of the possible values for that field (a true "don't care"), or to represent any subset of the possible field-values. For example, in the day-of-the-week feature, one possible wild-card could represent *Monday or Wednesday*, while in the initiator-importance field a possible wild-card is *greater-than 3*. This allows considerably more sophisticated rules to be expressed.

I have devised a method for combining the rules and examples in a single database, and for allowing multiple rule-types to co-exist within the same system. This work is discussed in greater detail in the following chapter.

For the Meeting Scheduling Agent, two kinds of rules have been implemented, *default* rules and *hard and fast* rules. Default rules are invoked only if there is no other close match in the example base, while hard and fast rules are invoked whenever they match the current situation.

4.3.1 Default Rules

In a default rule, matching a wild-card is treated as slightly worse than a natural match in the distance computation stage of the MBR algorithm. This allows actual examples to win out over similar default rules. It also allows more specific rules (of which examples are really just an extreme case) to win out over less specific rules, since the latter will contain more wild-cards.

Rules which do not match the situation exactly should never (or at least rarely) contribute to the selection of an action. This is ensured by assigning a greater than usual field-distance to any mismatches which occur in a rule. (When comparing two examples which mismatch in a particular feature, they contribute a *distance* × *weight* term to the weighted sum being computed as the distance between the examples, as described in the Memory-Based Reasoning section above. Here the distance may be a very low number, depending upon the particular values in question. When comparing an example to a rule which mismatches in the same way, we guarantee that the distance contributed is large, making mismatches which occur in rules more "serious" than those which occur in examples.)

Default rules do not affect the weights and distances computed for each of the features, as they are discarded when counting the number of occurrences of the various values in each field (which are needed for the weight and distance computations).

Default rules are given the same weight as a single example in the action selection stage of the MBR algorithm, so that a single close example will win out over a close default rule. However, they are given a somewhat greater weight in the confidence computation stage. This allows a reasonable level of confidence to be returned when the action is selected on the basis of a default rule alone.

4.3.2 Hard and Fast Rules

For hard and fast rules, matching a wild-card is treated just like a natural match (i.e., distance is zero). This is because for hard and fast rule behavior, the rule needs to look as good as an example.

As with default rules, mismatches are assigned a steep penalty. Hard and fast rules also do not contribute to the computation of weights and distances.

In order to ensure that hard and fast rules are selected over any matching examples, they are given significantly more weight than a single example in the action selection stage. To allow more specific rules to win out over less specific ones, this weight decreases with the generality of the rule, as determined by the number of examples the rule represents. (This is computed as the product of the number of values represented by each wild-card in the rule, and thus requires advance knowledge of what all of the possible values in each field are.)

At the confidence computation stage, hard and fast rules are given significantly more weight than any single example, allowing them to be applied with confidence.

4.3.3 Combining the Different Rule Types

The different types of rule behavior can be combined within a single database by permitting the slight modifications required in how the rules are to be treated to be made based on a tag identifying the particular rule's type.

In the case of the 8 a.m. meeting with the boss suggested above, if the user had specified the ***no meetings before 10 a.m.*** rule as a default rule and the ***always accept meetings with my boss*** rule as hard and fast, then the decision would be to accept the early meeting. If both rules had been of the same type, the decision would have likely come down to other examples in the database which closely match the features of this particular 8 a.m. meeting with the boss. (In other cases deciding between two rules of the same type often comes down to which is most specific, but in this case the two are equally general.)

4.4 Suggesting an Optimal Meeting Time

As discussed in the previous chapter, a user wishing to initiate a meeting may request that the group of involved users' agents suggest an appropriate meeting time. When this happens, mail is sent to all invitees requesting all available meeting times of the prescribed length within the specified date and time ranges, and the user's importance ratings for all of the other invitees. Though this could theoretically be handled by the human, in this implementation it is always intercepted by the agent. When the initiating agent has collected the replies, a set of candidate meeting times is assembled. If there is a non-empty intersection of the sets of available times returned by the invitees (and the initiator), then that set becomes the set of candidate times. Otherwise each timeslot in the meeting initiator's list is given a score which is the sum of the total importances of each invitee who is able to attend. The top k such times become the set of candidate times. The candidate times are then sent out to each invitee, who is asked to give each one a rating indicating how favorable a time it is for him or her. Again, this request is always intercepted and handled by the agent in the current implementation. (Responding to these specific requests are the only actions that the agents take autonomously without prior authorization from the user.) The agents reply based on how likely their user is (in their "opinion") to accept a meeting with the given characteristics at each of the possible candidate times. (The reason this is done in two steps, first narrowing down the list of possible times and then requesting ratings for the surviving candidates is that computing a rating for any given time-slot is a somewhat compute-intensive operation ($O(n)$ where n is the size of the example-base), so the number of times this is required is kept to a minimum in this way.) The replies are combined to arrive at a suggested meeting time as follows:

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. ($\forall i\ q_{ii} = 1$; other q values are relative to that in the range $\frac{1}{5}$ to 5.)

then define the optimality 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 his or her overall importance, defined as the sum of the importance measures assigned to him or her by the other participants.

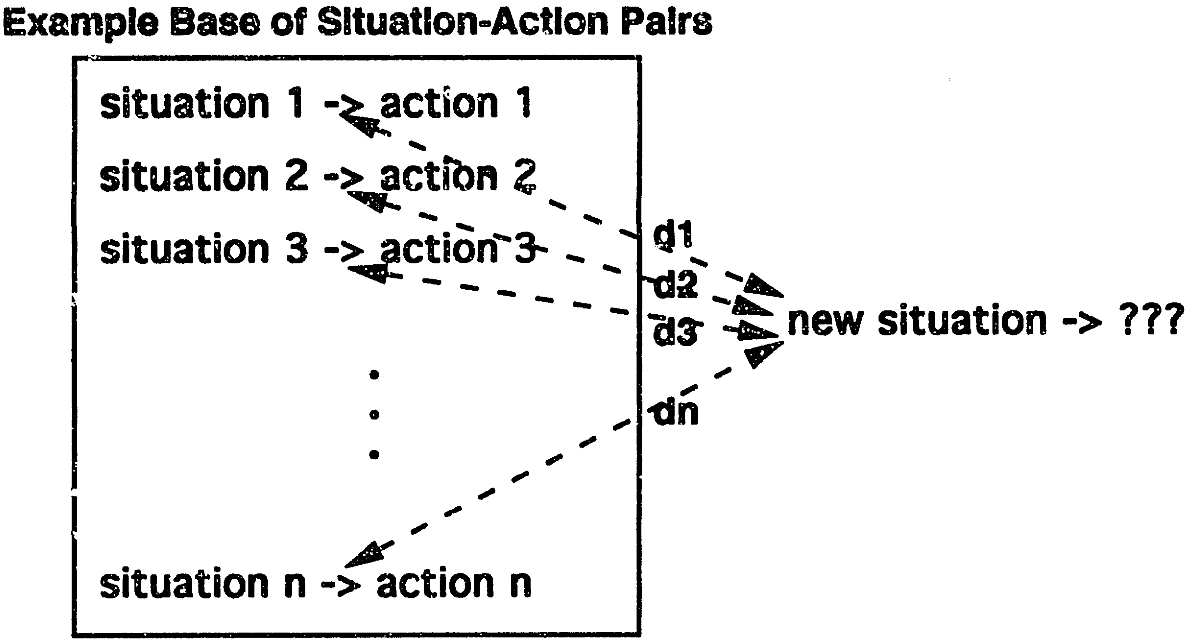


Figure 4-1: Memory-Based Reasoning — Action Selection

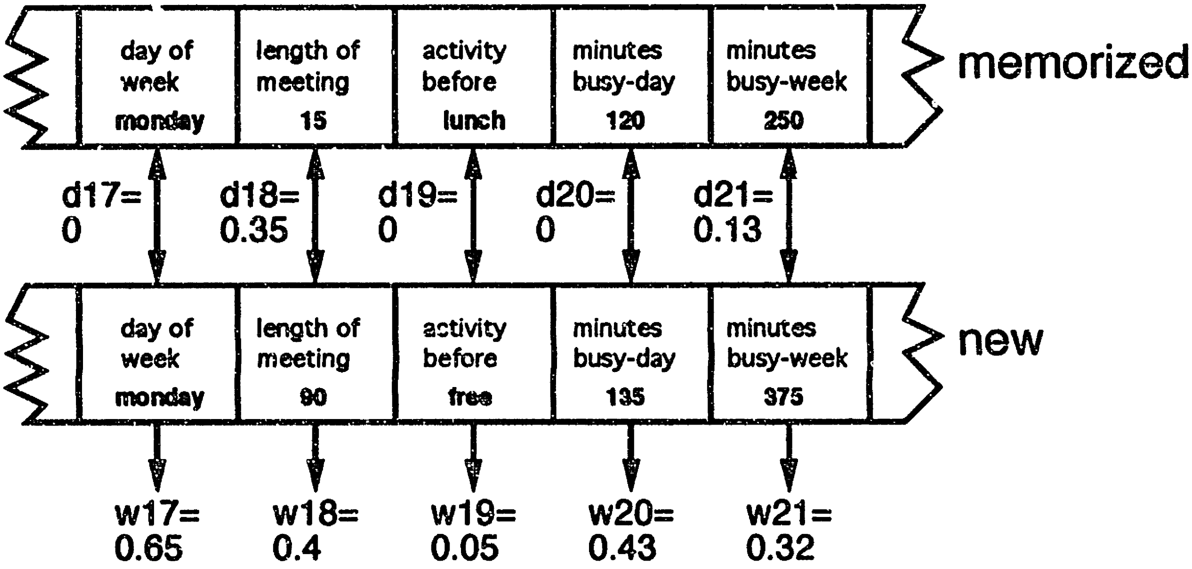


Figure 4-2: Memory-Based Reasoning — Computing Distances

Chapter 5

Incorporating Rules into the MBR Example Base

The main advantage of permitting rules to be incorporated into a Memory-Based Reasoning system is that it allows a user to provide rules which give the system some initial confidence, while allowing it to learn the subtle nuances from examples. This is particularly important when the examples are arriving one at a time, instead of having a pre-existing example base to work from.

As discussed in the previous chapter, Stanfill [16] introduced the idea of using wild-carded examples as rules. This chapter discusses an extension of that work. It explores how rules and examples may be combined within the same example-base and how different types of rule behavior may be easily achieved within the MBR paradigm.

The work described in this chapter is applicable to any application employing Memory-Based Reasoning — it is not specific to Learning Agents.

5.1 Rules in the Example Base

Rules within the example base may have differing effects depending upon how the wild-cards are treated in computing weights and distances, selecting actions and computing confidence values. This affords the possibility of creating several different types or strengths of rules by using different types of wild-cards, each with a different treatment.

Listed below are the areas in which different treatments could be considered, and some possibilities in each case.

1. **Occurrence Counts:** When counting the occurrences of each field-value which correspond with a given action (in order to enable the computation of d_f and w_f as described in Section 4.1.1), the weight given to information included in rules determines how much input the rule gets into deciding which fields are most important: the more weight each rule gets at enumeration time the more input it gives.

   - If a rule is not intended to offer information on which action to choose in cases where the situation does not match with it exactly, it should be disregarded during this phase of the algorithm.
   - There are several ways a rule's information can be included in the occurrence counts to allow it to provide guidance in situations that are close to it but do not match, if that is desired.
     - The simplest method is for the values which are not wild-cards to be counted as usual, disregarding the wild-cards.
     - Because rules represent more information than do single examples it may be desirable to give them more weight than a single example. On the other hand, if rules which do not match the situation exactly are expected to be less valuable in predicting the action than similar examples, it may be prudent to give them less weight than a single example in this computation. The weight given to a rule in occurrence counts represents the number of examples it will be treated as being equivalent to, and will be denoted by w.
     - In cases where the wild-cards can represent some but not all of the possible values in a field, it may be desirable to "credit" the values which are consistent with the wild-card as having "occurred". This can be accomplished by apportioning the rule's weight (w) over all of the

examples it can be seen as representing. Thus in fields where no wild-card occurs, the value seen will have its occurrence count incremented by w; wild-cards will result in all of the consistent values having their occurrence counts incremented by w/f, where f is the number of field-values consistent with that wild-card.

- In some cases, it may be desirable for the total weight contributed by a rule not to be constant, but instead to be proportional to the total number of examples it represents. In this case a (usually small) constant occurrence weight v is counted for each of the examples the rule represents. In this case, non-wild-card values will have their occurrence counts incremented by vn, where n is the number of examples represented by the rule; wild-cards will result in all of the consistent values having their occurrence counts incremented by vn/f, where once again f is the number of field-values consistent with the wild-card.
- In the above two approaches, it is also possible to weight natural values differently from wild-card values, by using different values of w or v for the wild-card vs. the regular fields.

2. **Distance Computation**: The way the distances between the current situation and the rules is computed contributes to the determination of what tradeoffs are made when choosing between conflicting rules and examples. (Recall that a wild-card represents all *or some* of the possible values for the field — thus it is possible to "mismatch" a wild-card.)

   - The simplest thing to do is count a field-distance of 0 whenever the value matches the wild-card, and some constant positive field-distance whenever it mismatches.
   - If it is desirable for the examples to take precedence over the rules when both match a situation, it is necessary make matching a wild-card a little worse than a natural match. To this end a wild-card match can count as a small constant field-distance, δ. Variations on this theme would be to make

δ a constant which depends upon what the field-distances are for other differences in the database (i.e., make it smaller than the smallest "true" field-distance, which could be arbitrarily small), or to have a non-constant δ which depends upon the number of situations the rule represents.

If δ is a constant, rules with fewer wild-cards will be preferred over those with more wild-cards; if instead δ increases with the number of examples represented by the rule, rules which match fewer situations will be preferred over those which match more situations. In most applications the latter would be more appropriate; however, in applications where the wild-cards will all represent approximately the same number of values, these two approaches will produce almost identical results, with the former being less computationally intensive, and having the advantage of not requiring advance knowledge of all the possible field-values, since the number of examples represented by the rule need not be computed.

- Rather than using a constant distance when a wild-card is mismatched, it may make sense to make the distance depend on the actual distances to the values which the wild-card represents. Any of the maximum, minimum or average of the distances to the values consistent with the wild-card may be appropriate.

- Sometimes it is important that the rules are never (or at least rarely) applied when their fully or partially specified fields (i.e., those containing non-wild-card values or those containing wild-cards which match only some of the possible values for the field) do not match exactly. This can be achieved by using a large (i.e., twice the usual maximum field-distance of 2) constant distance for wild-card mismatches, and by reassigning the field-distance contributed by a regular value mismatch to also be a large constant, or to be some multiple of the computed field-distance whenever the mismatch occurs in a rule (i.e., when there are wild-cards in other fields).

3. **Action Selection**: The selection of an action based upon the m closest situations is another area in which the decision of how to treat wild-cards will affect the interaction between rules and examples which are similar to the given situation.

    - A rule can simply be counted as a single example, or, to give rules either more or less weight than examples, as more than one or a fraction of an example.
    - The number of examples a rule represents can depend upon number of situations it represents. Using a value which increases with the number of rules represented would cause more general rules to have more weight, while using a value which decreases with the number of rules represented would favor more specific rules, which will generally be preferred.

      [Note that the choice of δ in the previous stage also impacts the choice of more vs. less specific rules.]
    - Another approach is to break down any close rules into the individual examples they represent, and then compute the actual distance to each one. The weighting of each example in this case could be a constant or a fraction of some total weight assigned to the rule.

4. **Confidence Computation**: The computation of the confidence can be adjusted to reflect a difference in the confidence with which rules and examples are applied. This may stem, for instance, from a difference in the source of the rules and examples, so that one set is considered more reliable than the other.

    - When rules are not broken down they can be treated as one or some other constant number of examples. Again, they can instead be treated as a number of examples which depends upon the number of situations represented.

      A variation would be to also discount or increase the confidence rating computed in one of these ways by a factor related to the number of rules used.

- When rules are broken down confidence can be computed as if there were no rules, or the rule-generated examples can be weighted differently from natural ones, again either using some constant value or a fraction of some total weight allocated to the rule.
- It may also make sense in some situations to replace the confidence computation with something completely different in the case where there are rules involved.
- When rules are involved (and matching a wild-card is treated as an exact match), it is much more likely that two or more situations (rules and/or examples) will match a given situation exactly, yet suggest different actions. If one is confident in the method used to select the rule(s) and/or example(s) which win out in this case, it may be desirable to adjust the confidence computation, which would otherwise return 0 whenever this occurs.

Different combinations of treatments in these four phases will lead to different rule properties. Note that many of the options listed above require *a priori* knowledge of all the possible values for each field. (When breaking up a rule into the set of situations it represents, and whenever a weight depends on the number of examples the rule represents.) This may make those choices inappropriate in certain applications.

5.2 Types of Rules

In this section I explore a few different types of rules, and the way that such rule behavior could be generated in the above context. In particular I discuss *default* rules, ***hard and fast*** rules and rules directly representing *multiple examples.*

A default rule is intended to help in the decision process when there are few or no examples or other rules which match the current situation closely.

Hard and fast rules are rules which are intended to be applied regardless of any support for a contradictory action in the form of examples.

In some cases it makes sense to treat a rule simply as a replacement for the set of examples which are consistent with it. This type of rule could be used to increase efficiency by compacting the database. Rules such as these would be applied whenever they matched, since matching a rule would be just like finding an exact match among the examples. These rules do not behave as one expects "real" rules to behave in some cases, however. In particular, they do not handle the case of more specific rules in the usual way.

To illustrate these types of rules, consider the following small database:

$$\begin{array}{cccccccl}
A & A & A & A & A & \rightarrow & action1 \\
A & B & B & A & A & \rightarrow & action2 \\
A & B & * & * & * & \rightarrow & action3 \\
* & B & B & A & * & \rightarrow & action4 \\
B & B & B & A & A & \rightarrow & action5 \\
A & B & B & A & C & \rightarrow & action6
\end{array}$$

For the new situation $ABBAC$:

- If the rules have been implemented as default rules, action6 will be selected, as it is suggested by an exact match.
- If the rules have been implemented as hard and fast rules, action4 will be selected as it is suggested by the more specific of the matching rules.
- If the rules have been implemented as multiple examples, action3 will be selected as it is suggested by the rule which represents the most situations (assuming for simplicity that each field has the same number of possible values).

For the new situation $ABBAB$:

- If the rules have been implemented as default rules, action4 will be selected as there is no exact match, and the most specific matching rule suggests that action.

- If the rules have been implemented as hard and fast rules, action4 will again be selected for the same reason.
- If the rules have been implemented as multiple examples, action3 will again be selected for the same reason as in the previous case.

5.2.1 Default Rules

To implement default rules, in the distance computation stage a match with a wild-card is treated as slightly worse than a natural match, by counting it as contributing some small field-distance, δ, to the total. This can be either a small constant or a small value directly proportional to the numbe. of examples represented.

If it is desirable for the rule to be unlikely to be applied unless its non-wild-card fields match exactly, this can be accomplished by assessing a large field-distance for mismatches that occur in non-wild-card fields of a rule, increasing the total distance computed in this case. However, depending upon the weight of the mismatching field in determining the total distance, and upon the total number of fields contributing to this computation, the total distance may still be relatively small, and may still end up contributing to the decision of which action to take, particularly when there are few or no very close examples or other rules. This is inherent to the MBR paradigm and could not be changed without "going outside the system" and including special case rules at a higher level to provide the desired behavior. However, rather than considering this a problem with the MBR approach, I consider it an advantage. In the case where there are not enough close examples or precisely matching rules to decide what to do, a non-matching but nonetheless "close" rule can make a contribution. (Because the confidence computation considers how close the contributing rules and/or examples were, this will be done with very low confidence, which is as one would want it to be.)

In the action selection stage the rule is given a weight of 1 (same as a single example). This is necessary to ensure that it will not be able to override a single close example; however, it also means that if several not-so-close examples suggest the same action (which is different from the one the rule suggests) the action suggested by the multiple examples may win out. If this is not desirable, it will be necessary to select a very small δ to use when matching a wild-card, to ensure that the rule ends up matching so much more closely that its action gets selected. (However, using too small a constant for δ has the problem of a rule with many wild-cards being seen as a very close match for *every* situation, simply because it has a very close match with the many wild-card fields. Using a proportional δ can help get around this but may reintroduce the problem of multiple not-so-close examples winning out in cases where the rules are general enough that relatively large value of δ ends up getting used. The trade-offs here will need to be carefully considered in any application.)

Since the information contained in a default rule is not supposed to come into play except in the case where nothing else in the database informs us regarding a given situation, the rule information should not be incorporated into the correlation information computed in the occurrence counts stage of the algorithm. Thus default rules should be disregarded during this stage.

In the confidence computation, if the action has been selected based upon a rule, it is usually desirable for a higher confidence to be computed than would be the case if an action was based largely on a single example. Thus a weight higher than 1 should be used at this stage.

5.2.2 Hard and Fast Rules

Hard and fast rules can be implemented by treating wild-card matches as exact matches in the distance computation while wild-card mismatches contribute some reasonably large constant field-distance, and by giving this type of rule a high weight in the action selection stage. To ensure that more specific rules will be preferred over more general rules which also match, it will be necessary for the weight used in the action selection stage to decrease with the number of examples the rule represents.

Once again, if one wishes to discourage the rule from being applicable in situations where non-wild-card fields do not match exactly, one can apply the maximum possible field-distance to mismatches occurring in the rule's non-wild-card fields.

In most cases one would also want to assign a high confidence to actions selected in this way — this could be accomplished by assigning this type of rule a high weight in the **confidence computation** stage, or perhaps even bypassing the usual confidence computation in the case where such a rule is applied and instead returning a confidence which depends only or mostly upon the expected reliability of the rule.

Usually a rule such as this would not be expected to provide useful input into the correlation data being collected on the examples, and thus rules of this type should be disregarded in the **occurrence counts** stage of the algorithm.

5.2.3 Rules as Multiple Examples

One notable difference between this type of rule and the other types presented here is that one normally *would* want these rules to contribute to the correlation data being compiled in the **occurrence counts** stage of the algorithm.

Since the rule is supposed to be replacing a set of examples, it makes sense to give each example the same weight it would be given if it had actually occurred in the example base; however, when the number of examples represented by a rule is large in comparison to the number of examples in the example base, it will usually be desirable to discount the weight given each of the represented examples, lest they completely take control. Once again, this can be done by assigning a small constant weight to each example represented by the rule, or by distributing a total rule-weight among the examples represented by it.

During **distance computation** matching a wild-card should be the same as a natural match (i.e., 0 field-distance). Mismatching a wild-card should probably contribute a field-distance equal to the minimum of the distances to any of the consistent field-values. This allows a rule to match closely whenever any of the examples it represents would have matched closely, which is important if you want to truly simulate the effect of having all the examples there.

In the **action selection** stage, if there is a rule which is close enough to be considered, it will be necessary to break the rule down into the examples it represents and compute a precise distance for each one, and then use those distances along with the distances to other examples to do the selection as usual. (This makes this type of rule computationally more intensive than others when the learning algorithm is performed on a serial computer.)

The **confidence computation** could proceed in the usual (rule-free) way, with the possibility of weighting rule-generated examples differently from natural ones, as in the case of default rules.

One use for this type of rule is compressing the example base. If a set of examples in the example base is identical to the set that would be generated by a wild-card example, then those examples could automatically be replaced by the wild-card version to cut down on computational and storage requirements. (Computational requirements are lower because this new example would only need to be compared once to a new situation; it would just have a greater chance of matching it closely. Only if it turned out to be one of the closest matches would the full computation have to be done for each of the examples it represented.) In this case you would want each example generated to count as a full example (in **occurrence counts**, and **distance** and **confidence computations**) since it actually occurred.

As an extension of this idea, it would be possible to allow the agent to consider adding a rule whenever a certain proportion of the examples it represents (like 90%) make their way into the example base, perhaps contingent on acknowledgment of the user or a "teacher." In the case where no acknowledgment is sought, it may be desirable to decrease the confidence by a small amount whenever this rule is applied, but that would have the possibly undesirable effect of decreasing the confidence in all the cases that were really already there in addition to the cases that were newly inferred.

As discussed above, this type of rule does not deal with the occurrence of more and less specific rules in the expected way. Because more general rules represent more examples, they have a better chance of representing more very close matches to a given situation than do more specific rules matching the same situation.

One way around this problem would be to change the behavior in the **action selection** stage to not break down the rules and then assign weights to the rules which decrease with the number of examples they represent. This would cause the expected behavior in this case, but at the cost of no longer really representing the set of examples consistent with the rule. (It is probably better to simply recognize that rules of this type have some inherent value, but do not behave in the most intuitive way in cases like this.)

5.2.4 Other Types of Rules

Above I have discussed three types of rules possible in a Memory-Based learning system. Clearly they represent only a small fraction of the possible types of rule behavior achievable by combining the different options presented in novel ways. For example, it would be easy to create rules which behave like default rules, except that they also contribute to the correlation data collected during the occurrence counts stage.

A rule type well-suited to virtually any application can be easily constructed in this system. All that is required is to decide what combination of the possible behaviors is desired, and then do a little empirical experimentation with the weight parameters to get something that works well.

There is also no reason why only one type of rule must be selected for a given application. It would be quite simple to tag rules of different types with enough information to identify them, and then combine several different types of rules in the same database, as was done with default and hard and fast rules in the Meeting Scheduling Agent. This diversity and ease of combination allows very sophisticated rule behavior in a system.

Chapter 6

Testing the Meeting Scheduling Agent

6.1 Introduction

The Meeting Scheduling Agent was tested in three ways. It was used within our group at the Media Lab for a period of six weeks, during which time all meetings between members of our group (consisting of a professor, a research scientist, five graduate students and three undergraduate students) were scheduled through the system.

It was tested on a set of eight inexperienced experimental subjects role-playing various members of a company. This data was collected in a single evening, and covered two weeks worth of meetings.

It was also tested on a set of hypothetical data in which seven users, with behavioral patterns and interactions based upon those of members of our research group, scheduled five months worth of meetings. The users' behaviors were all determined by me, based upon profiles describing their meeting scheduling habits.

6.2 "Real User" Testing Within our Research Group

The real-user tests were mainly intended to glean information about how people felt about using such an agent, and to collect data regarding the ways in which they used it. This test period also served to help debug the system, and make minor improvements to the user interface.

Unfortunately, a lot of the raw data collected during this test period was lost, mainly due to computers crashing (either caused by the Meeting Scheduling Agent system or other software running on the same Macintosh) while the data files were still open. In addition, the total number of meetings scheduled by people in this research group with others in this group during the weeks in question was small enough that the agents did not end up having a chance to learn very much.

Peoples' impressions of the system were mainly positive, however, this was hardly an unbiased group of users. People did have practical concerns, such as not being able to access their calendar when dialing in over the modem. Most members of the group were quite dilligent in entering their outside meetings into the calendar (one of the usual problems with groupware calendar software [6]).

6.3 Tests of the Agent with Role-Playing Users

The agent was also tested by hiring eight people to come in and role-play corporate characters using the Meeting Scheduling Agent. The hope was that within an evening, we would be able to run through four to six months of meeting scheduling situations. The users' inputs were collected so that the scenario could be replayed using different versions of the program (as discussed in the next section).

Unfortunately the hastily developed calendar application did not stand up to this vigorous a test, and so there were problems with it crashing, slowing down the process considerably.

There were also difficulties with the method chosen to send mail between the agents, which worked quite well during the period of testing within our research group, but which also could not withstand the assault of several users sending multiple messages at the same time. Unfortunately, these problems often caused the Macintoshes we were running the system on to crash, resulting in a loss of data, and inconsistency between users. Additionally, sometimes mail between agents took up to 30 minutes to arrive, which would not be much of a problem in real life, but was in this simulation, where a day or two might have "passed" in the interim.

Furthermore, it was quite unrealistic to expect that such a long simulation could be completed in an evening — it is now clear that even under ideal situations (including faked e-mail), it is necessary to allot about 30 minutes per week of simulation, plus about 30 minutes training time at the beginning and an extra 30 minutes (i.e. allow a full hour) for the first week, in which peoples' schedules are set up, rules are generally entered and the majority of regular meetings are initiated.

In the end, only the first two weeks of the scenario were completed in the four hours allotted. However, the results were somewhat promising, and are thus presented here in any case. Because of the inconsistencies and data loss caused by machine crashes, however, comparative tests using different versions of the software were not run. (Such tests were instead run for the hypothetical inputs as described in the next section.)

6.3.1 The Scenario

Here is the general information distributed to all participants:

General Information:

You all work for the Bar company in their software division, on the Foo project. The company has recently reorganized, bringing Programmer/Analysts Alan, Beth, Carl and Dianne, along with new student interns Sam and Tina, together under Pat's management. Pat's manager, Chris, supervises 3 other groups like yours as well as yours.

Everyone works flex-time, with the company policy being merely that you make yourself sufficiently available that you can interact with others as necessary to get your job done. You are paid to work 40 hours/week. Because of poor building layout, you don't get much chance to talk to other group members except at scheduled meetings.

This Scenario will go from June 1 until the end of November, time permitting. There are holidays May 31, July 5 and Sept 6, all Mondays. The students' internships run 6 months, from June through November (i.e. the same as the course of this simulation)

You may be instructed in boldface to schedule a meeting or respond in a particular way to a request. However, you are free to schedule whatever other meetings you feel are appropriate to the situation.
[some logistical information deleted]
Feel free to be creative. As long as your character is not a meeting-hater, feel free to initiate meetings for reasons and at times other than those indicated on your information sheets.

Thanks for Participating!

In addition, each user was given a private information sheet containing details of his or her character. These may be found in Appendix B.

Each user then got a new private information sheet each week. For the Week of May 31, each contained a version of the following:

Monday is the Memorial Day holiday, so you start on Tuesday. You meet your new group, who discuss their preferred work hours.

Alan prefers to work 8-4, with more flexibility in the morning than the afternoon, as he has to leave by 4:30 at the latest to pick up his child at daycare.

Beth prefers 9-5 but is quite flexible about that.

Carl prefers 7-3, and claims to be relatively flexible about that, but you notice a twinge of hesitancy in his reply.

Dianne, a real night person, tends to work 10-6, and really hates to have to come in any earlier than 10.

Sam doesn't really know what hours he will end up keeping --- at school he kept very varied hours. He seems very flexible about making himself available for meetings and discussions, though.

Tina is a morning person, and prefers a 7-3 working day, but is quite flexible about that.

There are two main pieces to the Foo project, the Baz part and the Zam part. The Baz part is the larger part, and Alan, Beth and the students have been allocated for that part. Carl and Dianne will handle the Zam part.

6.3.2 Results

The two bosses, Chris and Pat, were not invited to enough meetings (7 and 4, respectively) for their agents to actually learn anything (the agents were wrong in every single case!) — their data is not graphed here.

The data for the other characters is presented in Figures 6-1 through 6-6. This data represents what could be salvaged from the audit files after several crashes in some cases, and thus only represent partial results (i.e., there were other meetings for which the results of the agents' predictions were lost, particularly in the cases of Beth, Sam and Tina).

6.3.3 User Questionnaires

The participants in this experiment were also asked to fill out questionnaires concerning their impressions of the system after the experiment.

The participants were split as to whether or not they'd like to use the system on a regular basis. (Some felt they could not look past the difficulties we'd encountered in order to really asses how it would feel to use a more polished version of the system.)

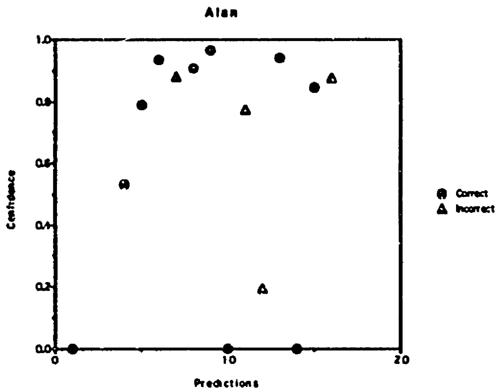


Figure 6-1: Scenario Results — Alan

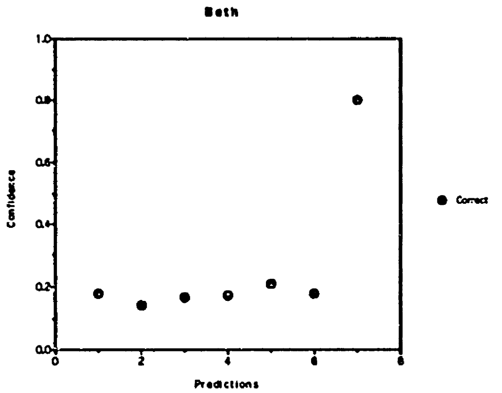


Figure 6-2: Scenario Results — Beth

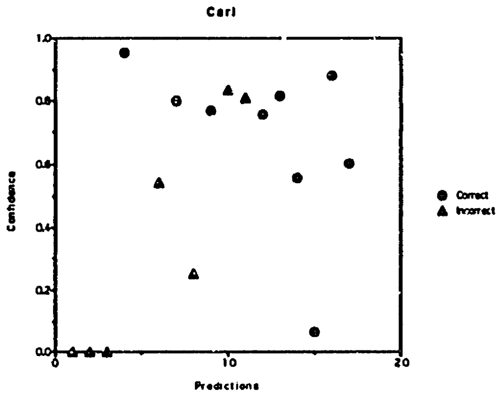


Figure 6-3: Scenario Results — Carl

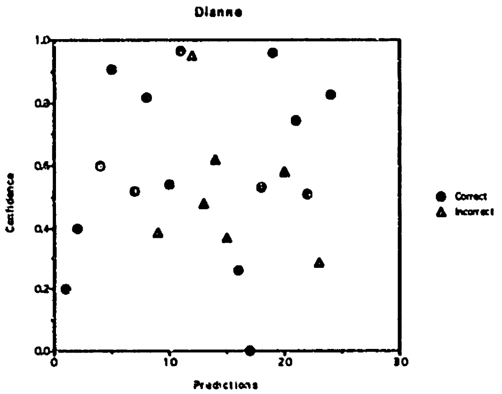


Figure 6-4: Scenario Results — Dianne

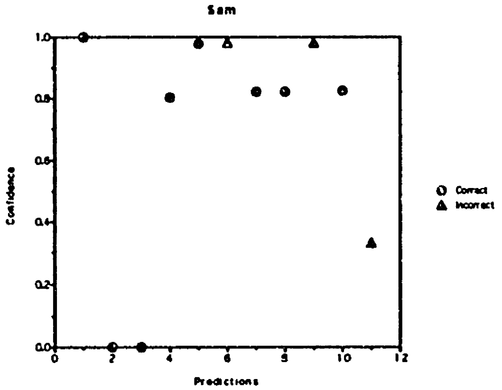


Figure 6-5: Scenario Results — Sam

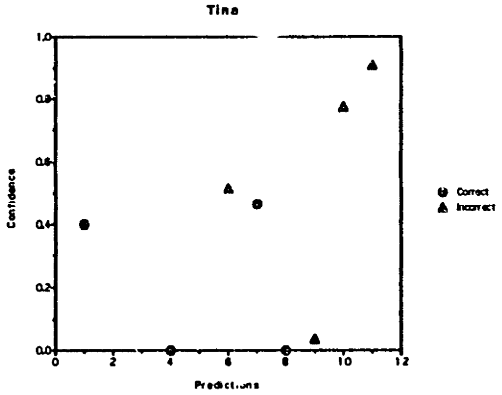


Figure 6-6: Scenario Results — Tina

During the experiment, all subjects used a single, all-situation "tell-me" threshold of 0.0 (i.e., they saw all predictions) and a single "do-it" threshold of 1.0 (i.e., the agents took no actions autonomously). This allowed the most general data collection to allow for re-simulation under different conditions. On the questionnaire, they were asked what thresholds they would want to use if they were using the system regularly. Answers for the "tell-me" level ranged from 0 to 0.8. For "do-it" users indicated values ranging from 0.9 to 1.0, with most indicating that this would be highly dependent upon the situation — certain actions would be more serious if taken in error. Three of the eight would use 1.0 in all cases, stating that they just would not feel comfortable having this type of action taken on their behalf, even by an extremely competent agent.

Participants were asked to rate their agent's competence. It was hoped to compare this rating to the agent's actual performance; however with so much lost data this was impossible. Participants also rated how helpful they felt the system would be, if used with their choice of thresholds. Responses on a scale of 1 to 10 (10 being the best) were: one 3, two 5's one 6, one 7, one 8, and two "unsures".

Finally, despite all the problems encountered in attempting this simulation, many participants left feeling enthusiastic about the future of such agents, and anxious to hear more about further developments.

6.4 Tests of the Agent using Hypothetical Input

A test suite for the agent was generated by querying several members of our research group about their meeting scheduling preferences, and then developing characters based upon their responses. A set of meetings that these characters might invite one another to was devised, and each character's response to the invitations was determined based upon that character's description. Five months of meeting scheduling data was generated in this way, which amounted to between 15 and 43 meetings per person.

The main benefit of the use of hypothetical data was that it allowed a longer period of meeting scheduling behavior to be monitored than was possible by any other method. Another benefit was that the data created was complete and consistent, in contrast to that developed by actual users of the program, who were plagued by both hardware and software problems, and more prone to error. This allowed the scenario to be replayed exactly in slightly different versions of the program, thus allowing an analysis of the roles of different parts of the agent.

6.4.1 The Characters

The Professor

The professor prefers to have most meetings between 10 a.m. and 6 p.m. Because of a class she teaches Thursday afternoons, for which she spends at least that morning preparing for, she does not meet on Thursdays until after 4 p.m. Because preparation for her class takes such a long time, she also tries to keep meetings on Tuesdays and Wednesdays to a minimum, preferring Monday and Friday meetings. The Director of the Lab is a very high priority for her — she will usually accept an invitation from him regardless of how convenient it is for her. Her workload varies greatly from week to week, making her scheduling behavior rather unpredictable.

She meets with her graduate students on a weekly basis, and the undergraduates biweekly.

The professor has four rules in the system:

1. Meeting begins before 10 a.m. or after 6 p.m. → Request Renegotiation (default)
2. Meeting on Thursday beginning before 4 p.m. → Request Renegotiation (hard and fast)
3. Meeting invitation from the Director with no conflicts → Accept (default)
4. Meeting invitation from the Director with one or more conflicts → Accept and Reschedule Conflicts (default)

Graduate Student #1

This student is in his first year of a Master's program. He prefers to meet between 9:30 a.m. and 5 p.m., and does not like to meet at all on Fridays. He tries to arrange his schedule so that meetings occur in clumps by attempting to add meetings immediately following other meetings (or classes, which show up as outside meetings) in his schedule. He is quite whimsical in his meeting preferences, often deciding about an invitation based on his mood that day and his workload that week, neither of which the agent has access to.

This student has three rules in the system:

1. Meeting begins before 9 a.m. or after 5 p.m. → Request Renegotiation (default)
2. Meeting on Friday → Request Renegotiation (default)
3. Meeting beginning immediately after another meeting ends → Accept (default)

Graduate Student #2

This student is also in her first year of a Master's degree. She prefers meetings which begin after 11 a.m., and if given her choice would meet in the evening rather than in the afternoon. She prefers to keep Tuesday and Thursday afternoons free, and has frequent lunch engagements on Fridays, so tries to leave that time open as well. She is interested in all topics which get discussed in this group.

This student has four rules in the system:

1. Meeting begins before 11 a.m. → Request Renegotiation (default)
2. Meeting on Tuesday or Thursday beginning after noon → Request Renegotiation (default)
3. Meeting on Friday beginning between noon and 2 p.m. → Request Renegotiation (default)

4. Meeting beginning after 5 p.m. and no conflicts → Accept (default)

Graduate Student #3

This student is finishing up his Master's degree. He prefers not to meet before 10:30 a.m., and is otherwise occupied most of the day on Mondays, so prefers not to come in on Mondays at all, and cannot possibly meet on a Monday until after 3 p.m. He also dislikes meeting on Friday afternoons. He tries to schedule meetings whenever possible for 2-3:30 p.m. on Tuesdays, Wednesdays and Thursdays. He will often try to renegotiate a meeting that does not fall within that time frame, particularly if it is only with one other person.

He has five rules in the system:

1. Meeting begins before 11 a.m. → Request Renegotiation (default)
2. Meeting on Monday → Request Renegotiation (default)
3. Meeting on Monday beginning before 3 p.m. → Request Renegotiation (hard and fast)
4. Meeting on Friday beginning after noon → Request Renegotiation (default)
5. Meeting on Tuesday through Thursday beginning between 2 and 3 p.m. lasting for less than 2 hours → Accept (default)

Undergraduate Student #1

This is a transfer student, taking a light load to finish up her degree. She does not have classes on Tuesdays or Fridays, and tries not to have to come to campus those days. She dislikes morning meetings, and prefers to schedule meetings right after a class or another meeting. She is doing her Bachelor's thesis in this research group, but is only interested in attending meetings whose topic directly relates to her thesis.

This student has three rules in the system:

1. Meeting begins before noon → Request Renegotiation (default)

2. Meeting on Tuesday or Friday → Request Renegotiation (default)

3. Meeting beginning immediately after another meeting ends → Accept (default)

Undergraduate Student #2

This student has a busy schedule of mostly afternoon and evening classes. He does not like to meet before 11 a.m. He prefers to avoid Wednesday and Friday meetings, and tries to schedule new meetings either immediately before or after other meetings or classes. He works in this group developing software which **Graduate Student #1** uses in his research, and thus needs to meet with him relatively frequently. He is not terribly interested in the research issues which come up in this group, and tends to avoid brainstorming meetings, reading groups, etc.

He has four rules in the system:

1. Meeting begins before 11 a.m. → Request Renegotiation (default)

2. Meeting on Wednesday or Friday → Request Renegotiation (default)

3. Meeting beginning immediately after another meeting ends → Accept (default)

4. Meeting ending immediately before another meeting begins → Accept (default)

Undergraduate Student #3

This student is also doing a Bachelor's thesis in this research group. She is a morning person, and likes to have all of her meetings between 7 a.m. and 2 p.m. She tries to avoid meeting on Mondays and Thursdays if possible. She does not like to schedule regular (weekly or biweekly) meetings, preferring to schedule each one as it comes along. She does not have much free time, and tries to avoid all unnecessary meetings, as educational as they might be.

She has 2 rules in the system:

1. Meeting begins before 7 a.m. or after 2 p.m. → Request Renegotiation (default)
2. Meeting on Monday or Thursday → Request Renegotiation (default)

Relationships

As mentioned above, **The Professor** considers invitations from the Lab Director (who enters this scenario only to invite **The Professor** to a few meetings) to have a very high priority.

In addition, all of the students consider **The Professor**'s invitations to have a high priority, and **Undergraduate Student #2** considers **Graduate Student #1**'s requests to have high priority as well.

6.4.2 Results

The scenario was replayed four times. The first time, the system was used intact. Then the scenario was replayed with one or both of Significance Feedback and Rule-Based Induction disabled. When rules were used, the rules listed in each user's description were entered into the system at the beginning of the scenario, before any meeting invitations arrived. The results from these trials are shown in Figures 6-7 to 6-13.

Not surprisingly, the agents for **The Professor** and **Graduate Student #1**, whose meeting scheduling behaviors depend on factors such as workload that the agents have no access to, did not fare particularly well, each achieving about a 50% hit rate. (This is still better than random, as there are many more than two options to choose between, including three common ones: accept, decline and request renegotiation.)

Graduate Student #2 was quite consistent in her behavior, and her agent did quite well, despite her having been invited to only 15 meetings.

Although **Graduate Student #3** was fairly consistent in his behavior, his habits are complex enough that his agent did not really have a chance to learn them well in the 21 meetings he was invited to. It did, however, improve from a 30% hit rate

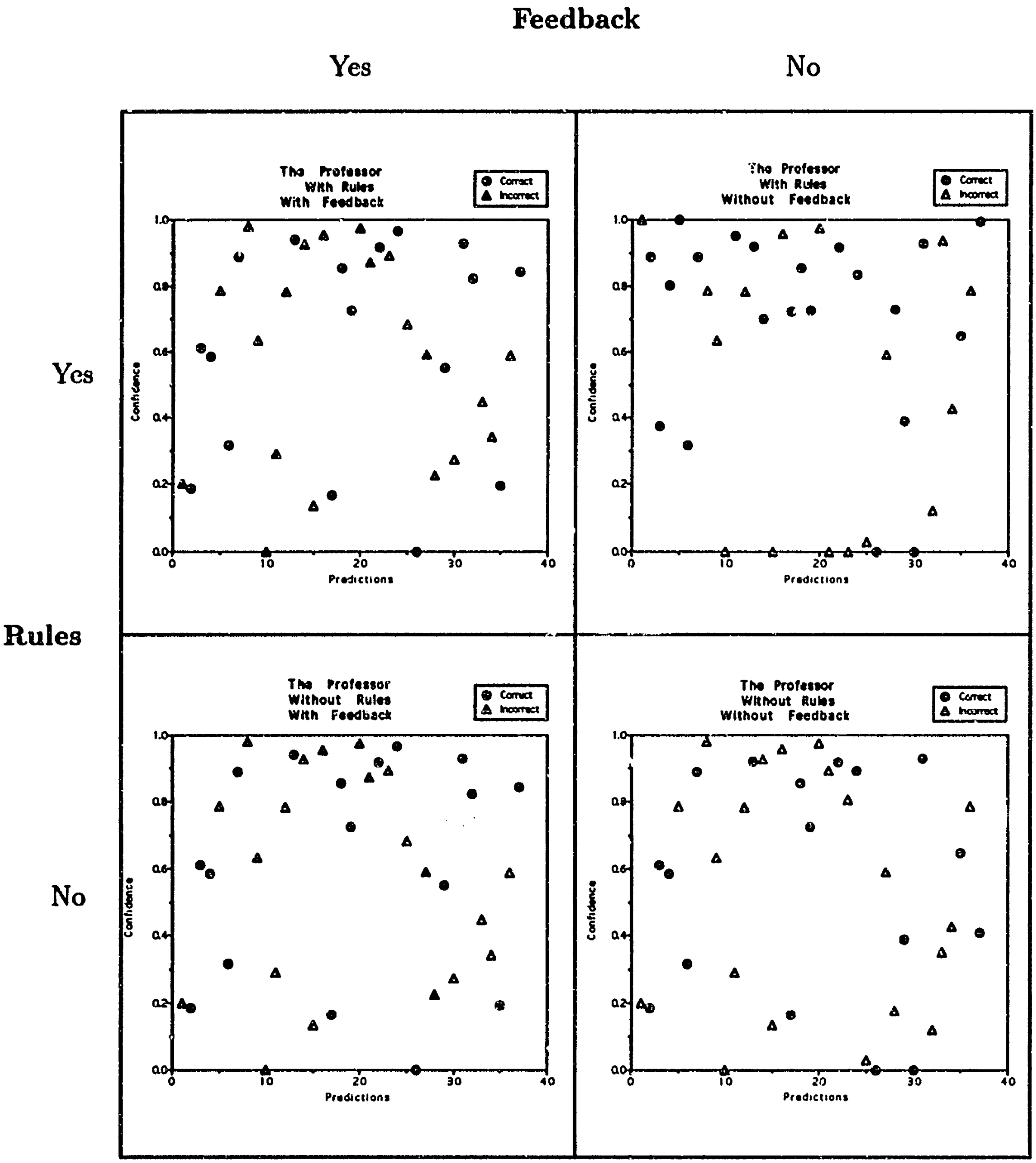


Figure 6-7: Results — The Professor

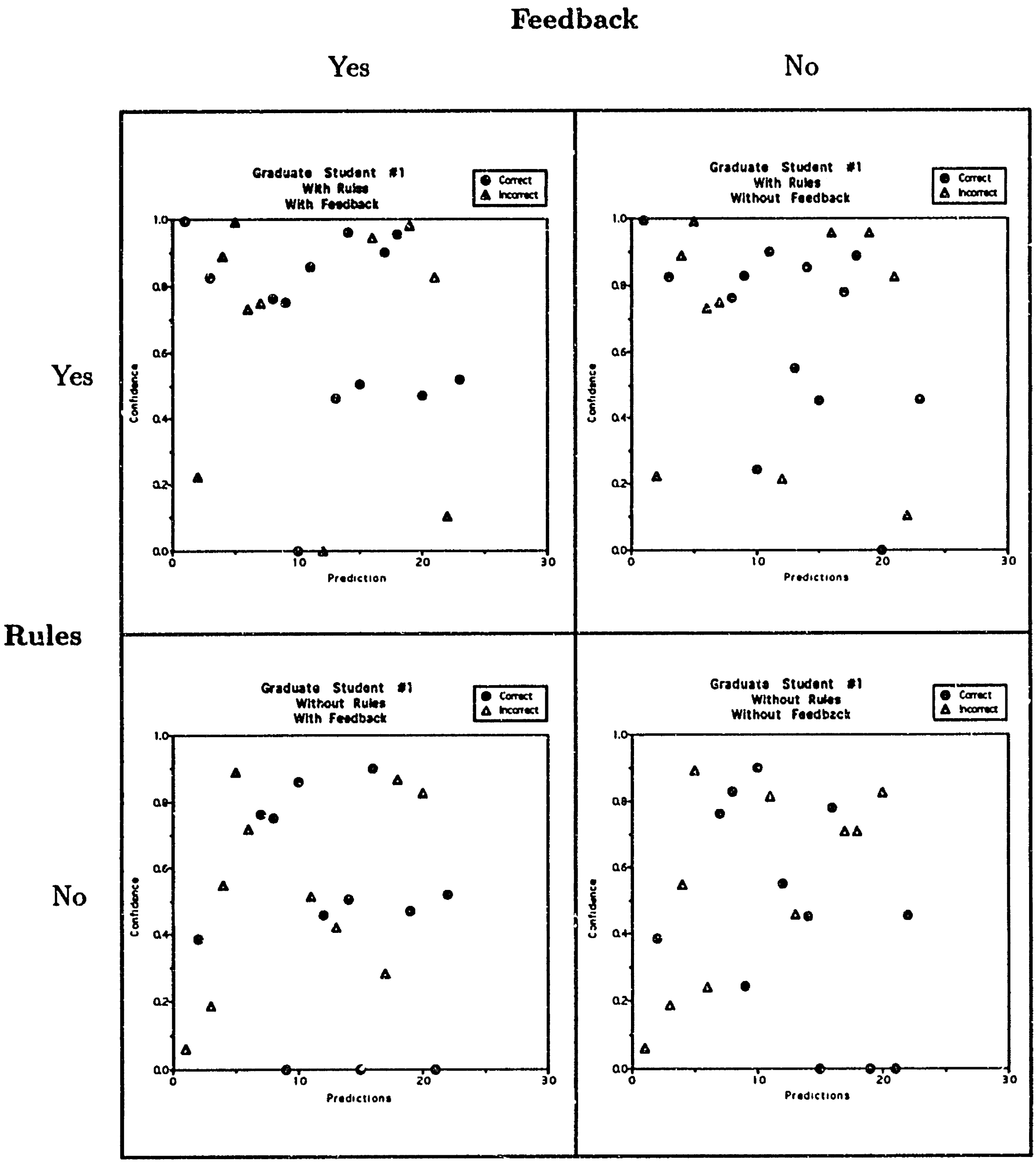


Figure 6-8: Results — Graduate Student #1

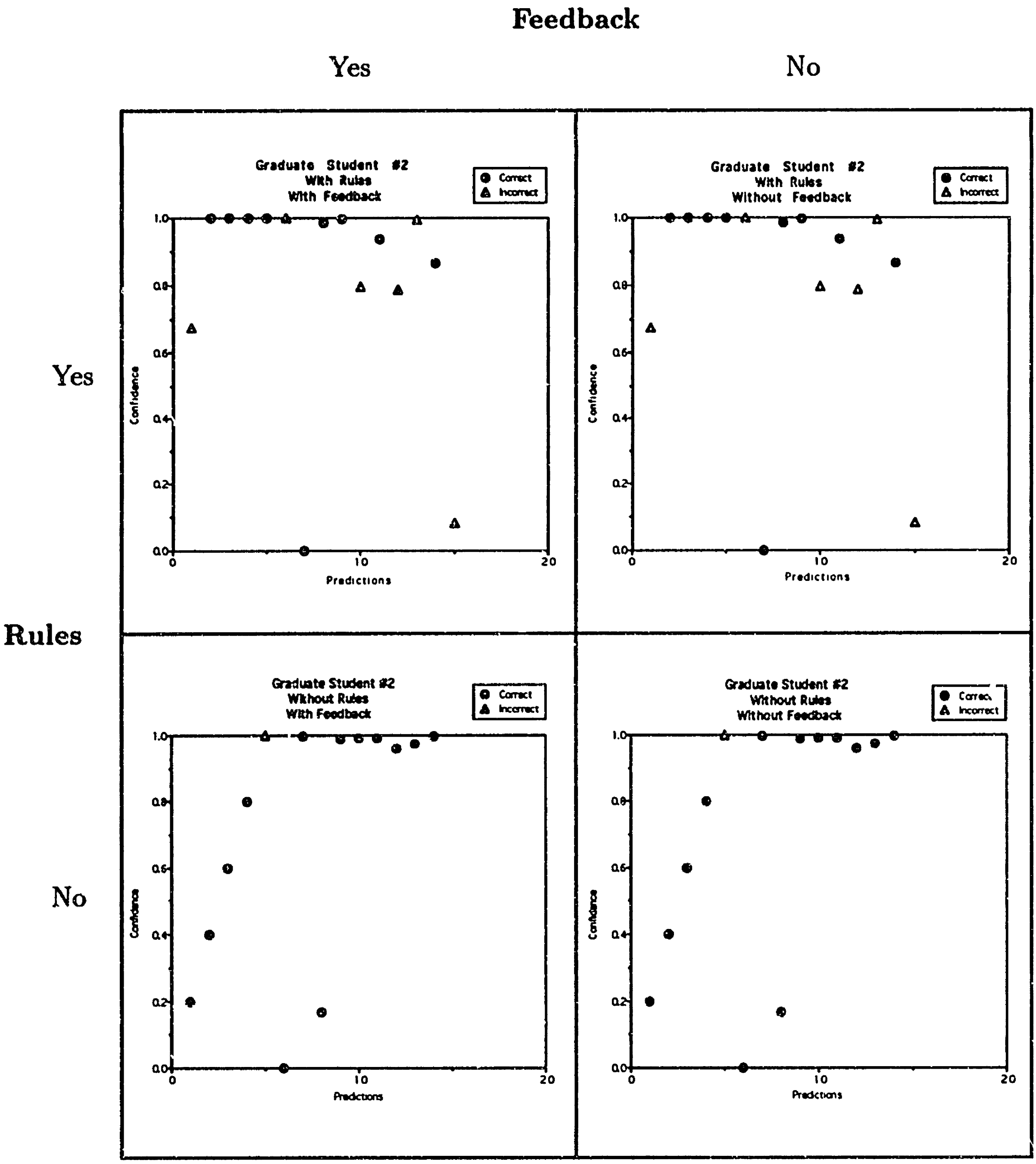


Figure 6-9: Results — Graduate Student #2

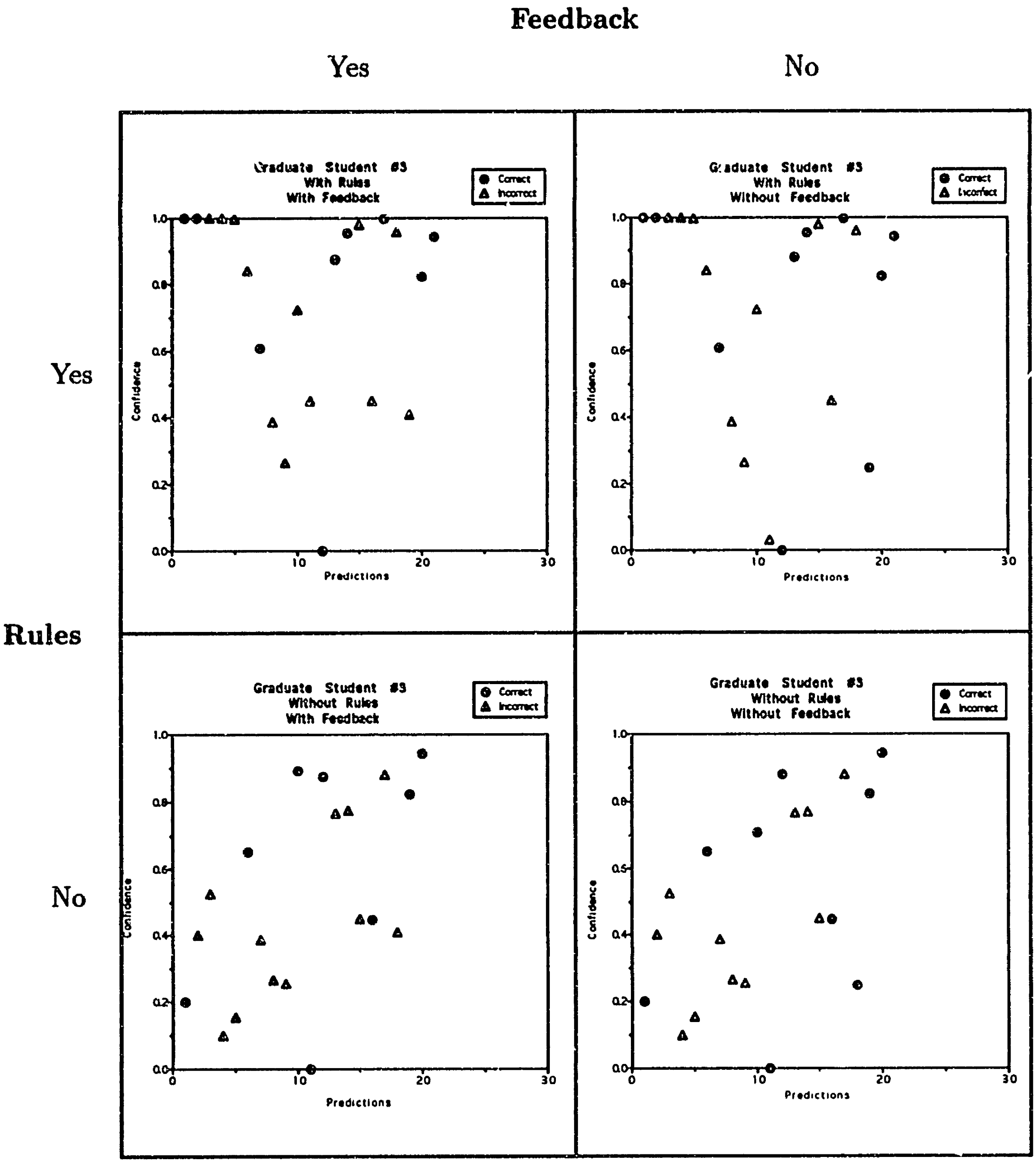


Figure 6-10: Results — Graduate Student #3

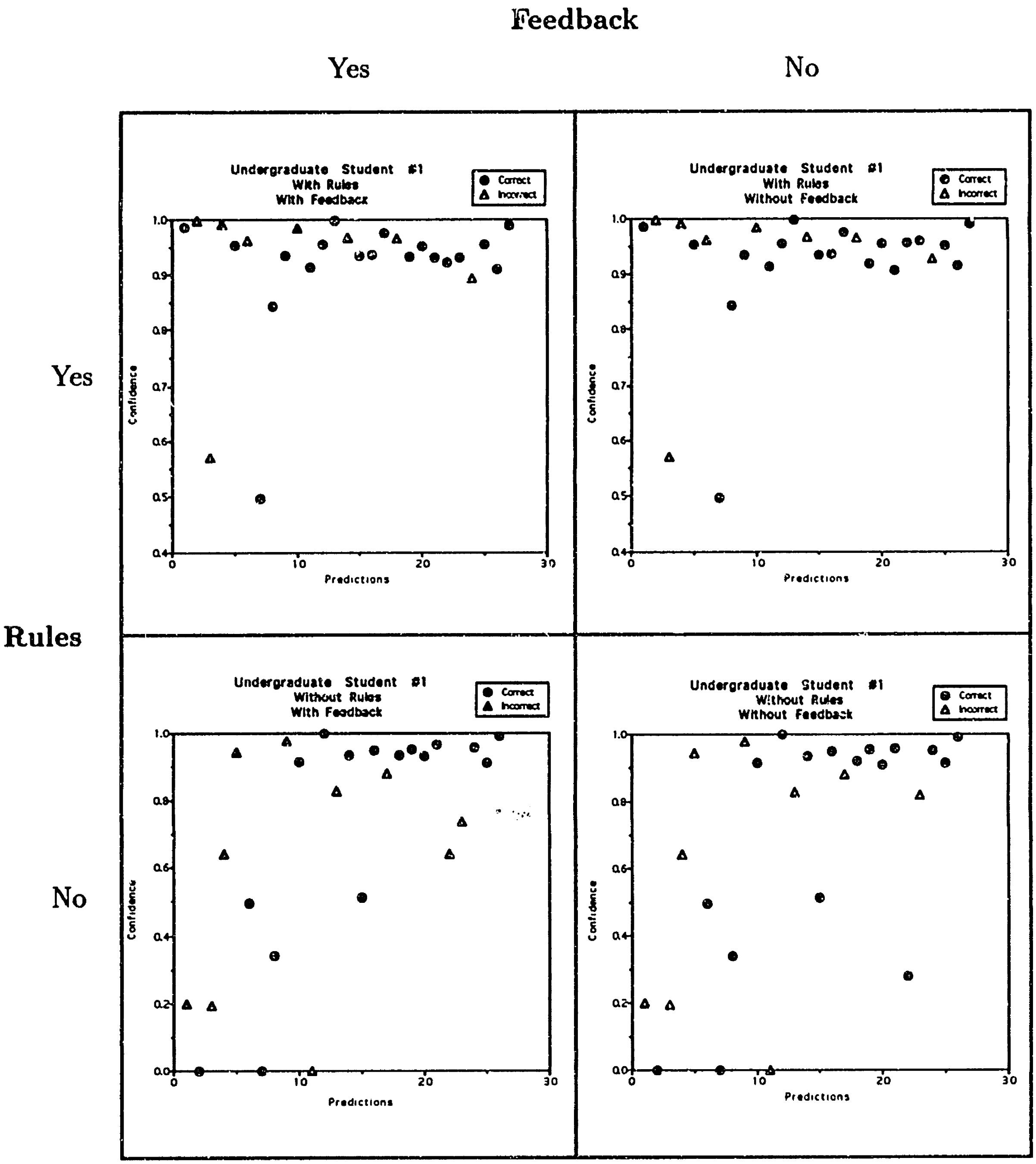


Figure 6-11: Results — Undergraduate Student #1

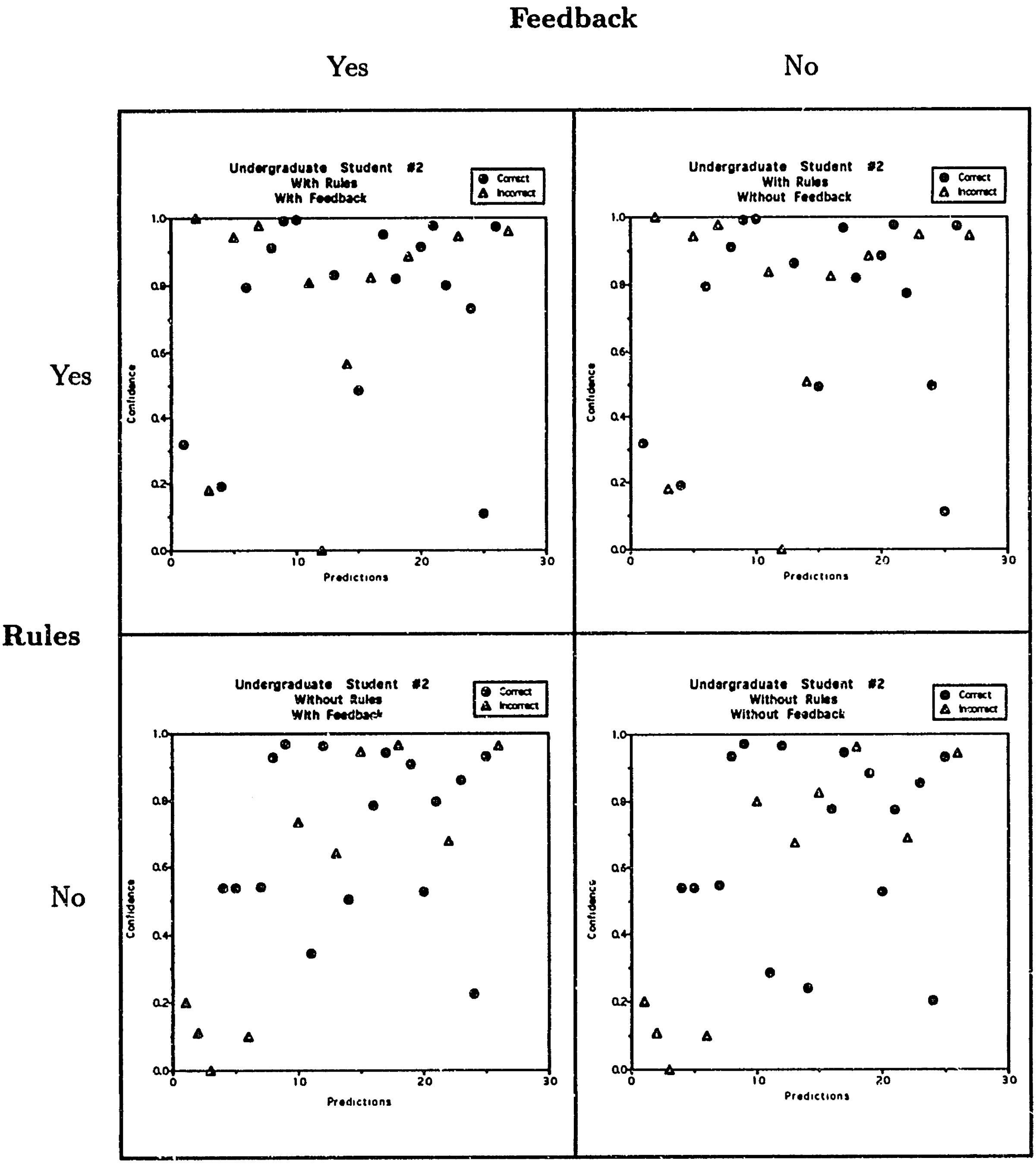


Figure 6-12: Results — Undergraduate Student #2

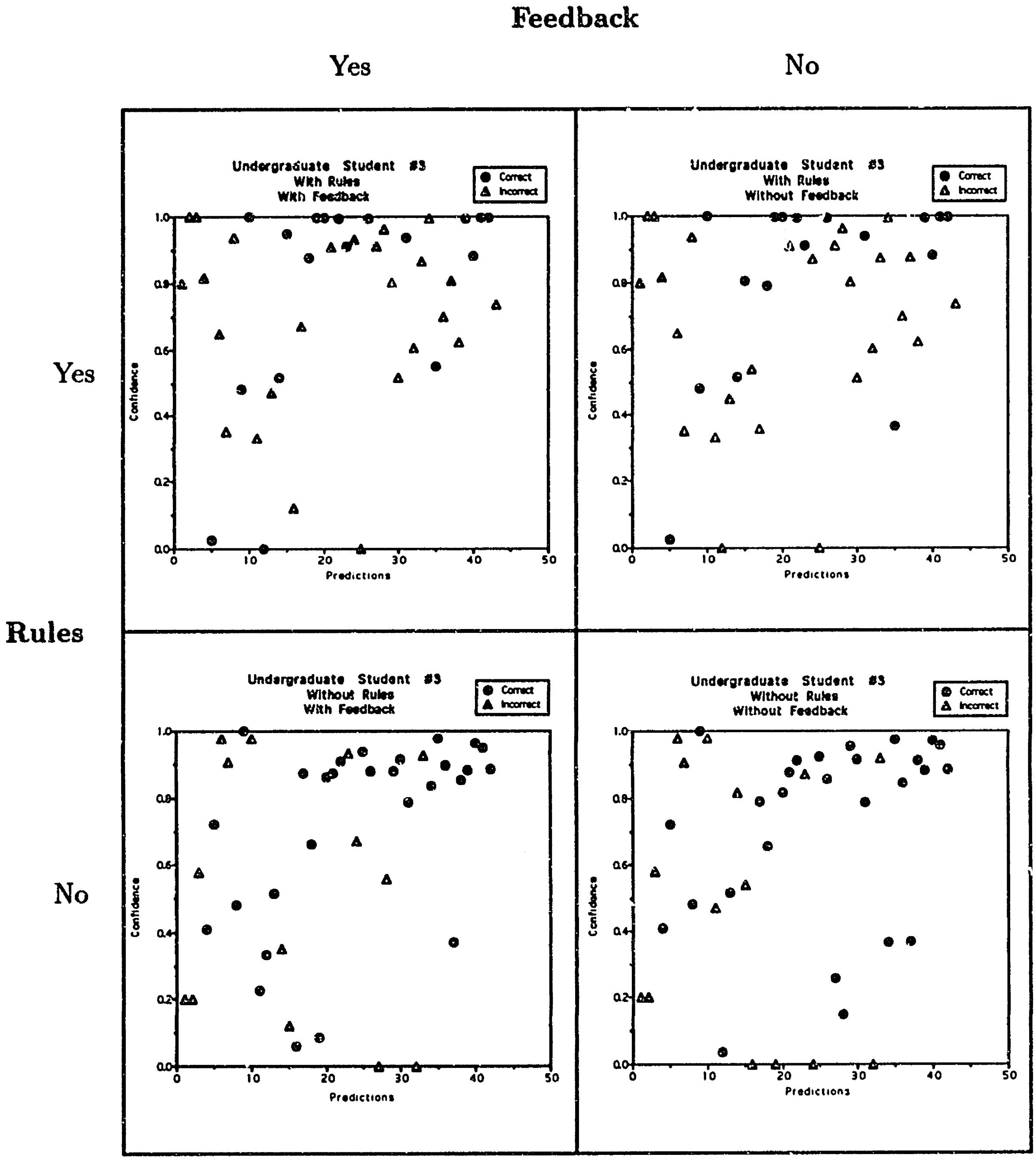


Figure 6-13: Results — Undergraduate Student #3

in the first 10 meetings to a 50–60% hit rate (depending on the particular test run considered) in the remaining meetings.

The agents for **Undergraduate Students #1** and **#2** did quite well overall, and decreased the proportion of incorrect predictions over time; however, they did quite poorly at recognizing when they were making a weak prediction, and had many incorrect predictions at high confidence levels.

Because **Undergraduate Student #3** had fairly simple scheduling habits, and an aversion to regular meetings (thus providing his agent with a large number of quite similar meetings as examples), his agent did the best of all (at least without rules — section 6.4.2 for more details.)

Effects of Significance Feedback

The inclusion of Significance Feedback tended to have a very small but positive effect. It usually allowed one or two extra correct predictions, and often decreased the confidence in at least one incorrect prediction or increased the confidence in a correct prediction or two.

For reasons which are unclear, Significance Feedback caused somewhat worse performance for **Undergraduate Student #1** when applied in the absence of rules, and for **Graduate Student #3** both with and without rules.

Significance Feedback would be expected to be more helpful in an application like news filtering, where many more keywords appear within a short time span, and where repetitions of keywords are more common.

Effects of Rules

Rules turned out to be somewhat of a mixed blessing — though they generally allowed a few additional correct predictions, they also tended to significantly raise the agent's confidence in its incorrect predictions. This occurred because the rules were entered at the beginning of the scenario, before the agent had any examples to serve as possible counter-examples. Thus the agent would confidently make a prediction based upon a rule, where before it would have made a very unconfident guess. Since the rules entered were quite general, there were lots of exceptions to them which the agents did not have a chance to learn before beginning to apply them.

The agents for **Graduate Student #2** and **Undergraduate Student #3** did significantly worse when using rules. This occurred because the rules were far to general, and did not really reflect the users' true behavior. For example although **Undergraduate Student #3** *says*: "Meeting on Monday or Thursday → Request Renegotiation," she really *means* something like, "If the meeting is on a Monday or Thursday, and I'm interested in going to it, or even if I might be interested in going to it if it were at a better time, then request renegotiation. If I'm not interested then decline. If this is a re-invitation to a meeting for which I had previously requested renegotiation but for which renegotiation has been declined, and I'm interested enough in the meeting topic to go at an inconvenient time, or if it was initiated by **The Professor** then accept, otherwise decline." Entering the simple rule misleads the agent in a number of situations. Eventually, the agent will gain enough counterexamples to know how to deal with these situations, but in the mean time it makes numerous mistakes by using that rule in situations where it should not really apply. Similarly, when a user has a rule which defaults to accepting the meeting, because it is at a time they like to meet, they usually also mean, "provided that there is no conflict, and it is a topic I am interested in." Again, eventually counter-examples will amass which allow the agent to correctly deal with the situation, but it would have been better off simply to learn the pattern from scratch than to start out with this often incorrect rule.

It would have been possible to get much better results by entering more complex sets of rules expressing things like the above, however the rules in this test were based upon the ones group members gave me when I interviewed them in order to set up this test. The rules used are the types of rules naive users are likely to use. An important lesson from these tests are that naive rules are at best marginally helpful, and may actually be quite harmful. It will be important to educate users on the construction of meaningful rules if they are to be used effectively.

6.5 Testing the Meeting Time Suggestion Algorithm

As described in Section 4.4, the Meeting Scheduling Agent has the ability to collaborate with other users' agents to suggest an optimal time for a group to meet. This facility was tested in an earlier version of the agent software, using a hypothetical scenario similar to the one described in 6.4 (see [8]). In this test, the seven agents were asked to collaborate to schedule two group meetings. One such instance occurred one third of the way through the scenario, and the other two thirds of the way through. In each case the agents were given a one-week range of dates in which to try to schedule the meeting. In both cases, the times suggested were acceptable to all of the participants. In the second case, the time was less convenient for two of the users, but they were able to accept it. It was confirmed manually that there were no times available that would have been more convenient for all of the participants.

Because this part of the Meeting Scheduling Agent has not changed since these tests were performed, and because of the difficulty in evaluating a suggested meeting time (which essentially requires manually confirming that other possible times would not have been as good), the meeting suggestion facility was not re-tested for this thesis.

Chapter 7

Related Work, Conclusions and Future Work

7.1 Related Work

7.1.1 Another Meeting Scheduling Agent

A similar project to the one described in this thesis is underway at Carnegie Mellon University [4]. Dent and others in Tom Mitchell's group have developed a personal learning apprentice, called CAP, that assists a user in managing a meeting calendar. Their experiments have concentrated on predicting meeting parameters such as location, duration and day of the week for a user attempting to initiate a meeting. (By contrast, my meeting scheduling agent concentrates on predicting a user's replies to meeting invitations.)

CAP makes a suggestion for every parameter it predicts. It is presented in such a way that the user can easily accept that suggestion, or simply enter another value. CAP never takes any action autonomously for its users, it simply speeds things up by often making the desired value available.

Their apprentice uses two competing learning methods: a decision tree learning algorithm and a back-propagation neural network. As discussed earlier, one advantages of my largely Memory-Based approach is that there is no loss of information:

when making a prediction the detailed information about individual examples is used, rather than general rules that have been abstracted beforehand. Another advantage of the Memory-Based approach is that it lends itself easily to the computation of a confidence level for any prediction. Thus the agent has a good idea of the accuracy of its suggestions. The main disadvantage of the Memory-Based approach, as compared to approaches such as CAP's, is that it requires more computation time to make a suggestion.

7.1.2 Demonstrational Interfaces

The agent presented in this thesis shares its learning-by-example approach with a class of programs called *demonstrational interfaces*, in which the user demonstrates a task and the agent learns to perform and generalize it.

EAGER [3] is an "eager personal assistant" for HyperCard which continuously observes the user, and offers to take over if it notices a pattern in the user's sequence of actions. Mondrian [10] and Peridot [14] are demonstrational systems for graphical applications in which the user tells explicitly demonstrates a task to be learned by means of an example, and then the agent generalizes it to other situations.

Some of the differences between these systems and mine are that the Meeting Scheduling Agent's learning takes place over a longer time frame (weeks and even months) while these programs learn very quickly. This means that they must learn relatively simple patterns, compared to the ones the Meeting Scheduling Agent is able to learn. These systems essentially allow the creation of macros (sequences of actions) based on examples.

Also, Mondrian and Peridot require the user to explicitly tell the agent when to observe, whereas the Meeting Scheduling Agent is constantly observing and learning, while still retaining the ability to be intentionally trained by its user, by means of rules.

7.2 Conclusions

The implementation of the Meeting Scheduling Agent demonstrates how Memory-Based Reasoning, Significance Feedback and Rule-Based Induction may be combined in one system. This method is application-insensitive, and thus provides a basis for other similar Learning Interface Agents to be built.

Ideally, to test a system such as this one, one would install it in a meeting-intensive environment for a period of several months, collecting statistics on the agents' performance, and interviewing the users regularly. However, several factors make this approach impractical. First there is the matter of time — such a long testing period would be very difficult to fit into the time frame of a Master's degree. Second, people would be reluctant, with good reason, to use a completely unproven system. Third, the fact that the calendar application is not state-of-the-art and merely a prototype would make it unreasonable to expect people to rely upon it.

The testing which was done and described in the previous chapter suggests that this approach is a promising one, but considerably more testing is required. A significantly longer (i.e., 100 invitations per person) hypothetical scenario would be helpful in determining just how much experience is really needed for the agent to become competent at predicting relatively complex behavior; however, what is really required is real field testing. Hopefully this will become more practical with the arrival of software written to support Apple Events, since a top-of-the-line application could then be coupled with an agent built in the manner described in this thesis. Since the agent can be made relatively unobtrusive by judicious setting of the thresholds, users could continue using the software they normally use, while training an agent with minimal inconvenience.

The results presented in the previous chapter demonstrated that the learning approaches used are sufficient to begin learning a user's habits within about 25 to 40 examples, provided that the user a) is fairly consistent in his or her behavior, and b) uses criteria the agent has access to to make scheduling decisions.

This generalizes to suggest that the integrated machine learning approach applied to build this agent is a good one for applications whose complexity is similar to that of the calendar application and whose users tend to exhibit consistent patterns of behavior.

However, for users with more complex patterns of behavior 20 to 40 examples were not enough for the agent to really gain any significant amount of competence. This is particularly true when the example set contained contradictory behavior. Although Memory-Based Reasoning has been shown to work on example bases containing exceptions [15, 2], these results apply to very large example bases. In a small incrementally growing example base, every example has a fairly strong weight, and atypical examples have the potential to confound the agent's learning. Some relief from this problem could come from user-directed forgetting, as discussed in Section 7.3.1.

One problem noted in the testing, especially using the hypothetical scenario, is that the agents did not succeed particularly well in recognizing when they were making weak predictions — confidence levels for incorrect predictions tended to be quite high for most of the agents. This is an important problem, as a user's trust depends in large part upon feeling that the agent not only knows what to predict most of the time, but also realizes when it cannot make a good prediction. The development of a more accurate confidence computation algorithm is thus an important area in which to focus future research into building this type of Learning Agent.

Although the task of meeting scheduling seems to be of a reasonable complexity for this type of agent, the amount of time (i.e., months) that it takes for most people to provide enough examples for the agent to learn makes this particular application somewhat less than ideal for this type of agent, except in very meeting-intensive environments.

The comparative testing run on the hypothetical scenario raised an important caveat regarding the use of rules — that carelessly added rules are often too general, and thus worse than no rules at all. This will need to be taken into account when deciding whether or not to provide facilities for adding rules in a particular Learning Interface Agent.

Interviews with people who had the experience of using the Meeting Scheduling Agent during the course of the testing confirmed that many people are enthusiastic about the upcoming arrival of Intelligent Interface Agents, and anxious to be able to use them in their everyday life.

The fact that some people were reluctant to hand over the meeting scheduling reins to their agents was not unexpected — this author feels the same way. However, the system still has benefits for people who do not allow the agent to take autonomous action. Provided that the system is used enough that the agent gets to know the user's preferences, it can provide (in collaboration with other users' agents) a very powerful method of suggesting meeting times. This method is significantly better than the usual method of simply intersecting the participants' schedules and taking the first available slot for two reasons: 1) it can handle the case where there is no intersection of the users' available time, and 2) it takes into account significantly more information than just who's free when.

7.3 Future Work

7.3.1 Intelligent Forgetting Strategies

In the Memory-Based Reasoning algorithm presented, it is necessary to keep the size of the example-base down to a reasonable size, where reasonable may be determined empirically. (On serial machines this is due to execution time getting out of hand, and on parallel machines the practical example-base size will be limited by the number of processors available, though the latter will normally permit more examples.) This has the additional advantage of allowing for behavior which changes over time — eventually the older behavior will be forgotten.

In the current system, a simple cut-off is used — after a certain number of examples are amassed, the oldest are discarded. I propose that numerous alternatives to this approach be considered.

One simple alternative is to consider decaying material before discarding it. While decayed rules would not directly help with the computational complexity issue, they may be a more accurate way of dealing with changing user behavior, and it may turn out that this is thus more desirable, even when the total example-base size is held to the same maximum as before.

Another option would be to consider when the examples originated, and perhaps use, for example, the past three months worth of examples (or the most recent N examples) plus the examples that arose during the previous year during this month. This could be a very effective approach in organizations where schedules tend to be predictably cyclic in some way. A further step in this approach would be to automatically analyze a set of examples (some of which are not in current use in making decisions, due to having been retired some time earlier) to determine whether there is a set of older ones which seem to correlate particularly highly with current situations, and reinstate those.

A more sophisticated approach would be to somehow allow certain parts of a person's recent behavior to be discarded, while other parts remain (perhaps abstracted into rules as discussed in Section 5.2.3). For this to occur, there would need to either be a way for the user to tell his agent what to keep and what to ignore, or for the agent to infer it automatically. This could be very important 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.

One way for the agent to decide which examples could be discarded would be to monitor the quality of each example by keeping track of 1) how many times the example was involved in making a prediction; and 2) how many of those times the prediction was correct.

It would also be useful if a user could just label a particular example as exceptional, and have it be ignored (or perhaps just weighted less strongly) by the agent. This type of user-directed forgetting would allow the agent to learn more reliably when dealing with a relatively small example base.

7.3.2 Confidence Computation

As mentioned in Section 7.2 above, the confidence computation algorithm did not do a very good job of recognizing weak predictions. As a reliable confidence computation algorithm is important to the development of a trustworthy agent, this is an area into which further research is required.

The current algorithm (described in Section 4.1.3) takes three factors into account:

- the relative distances of the best situations predicting the selected action and those suggesting another action,
- the proportion of the top m situations which predict the selected action, and
- the proportion of the top m situations which fall within a reasonable maximum distance from the current situation.

I believe that these factors are the correct ones to consider, but hopefully experimentation into different possible weightings of these factors can reveal a more accurate formula to use.

7.3.3 Feature Specification

Currently the set of features that are used to specify a situation are hand-coded and inflexible. A truly intelligent agent would need the capacity to allow change to the feature set (particularly additions — deletions would affect speed but not accuracy as irrelevant features are quickly detected as such by the MBR algorithm and given low or zero weight). The MBR code which was implemented for the Meeting Scheduling Agent does allow for the simple addition of features — the hard part is getting them specified.

User-Specified Features

It would be beneficial to eventually allow a user to specify some particular features for the agent to consider, so that even very idiosyncratic criteria could be included.

The most open-ended way of doing this would be to allow the user to write some Lisp code, but this is somewhat undesirable for obvious reasons. Ideally a nice interface could be provided which allows a user a way of specifying at least some of the more likely types of features for users who do not wish to program their agents directly in Lisp.

Automatic Feature Generation

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 by Fawcett and Utgoff [5]. The idea is that you have a (complete) set of very low-level features, and various combinations of these are examined in order to find meaningful ones. 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. Even better would be if a true low-level set of features could be devised for this (or any agent's) application, which could be used as input to the automatic feature generation algorithm.

7.3.4 Ranges of Feature Values

The basic Memory-Based Reasoning algorithm as discussed in [15] 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 future implementations might concentrate on how this might be done automatically.

Appendix A

Features used by the Meeting Scheduling Agent

This appendix describes the features used by the Meeting Scheduling Agent in the Memory-Based Reasoning algorithm. In what follows, all references to "the meeting" refer to the proposed meeting that the user is replying to.

Because the Memory-Based Reasoning algorithm requires each feature to have a reasonably small number of distinct values so that particular values will be repeated frequently, most of the numeric features used for this agent have their possible values broken down into a relatively small number of ranges of values, with the ranges then serving as the feature values. Features for which this is the case are indicated by an asterisk (*) preceding their name. (In the case of values based on other feature-values, the actual values of the component features are used, and the value is only translated into a range after the computation is done.)

Day of Week that the meeting was scheduled for.

* **Starting Time** of the meeting.

* **Length** of the meeting.

Frequency of the meeting.

Number of Participants invited to the meeting (including the initiator).

* **Total Participant Importance:** the sum of the user's importance ratings for all of the people invited to the meeting.

* **Average Participant Importance:** the average of the user's importance ratings for all of the people invited to the meeting.

Initiator of the meeting.

* **Initiator Importance:** the user's importance rating of the initiator.

* **Total Keyword Importance:** the sum of the user's importance ratings for all of the keywords found in the meeting description. (Keywords are all of the uncommon words in the meeting description.)

* **Average Keyword Importance:** the average of the user's importance ratings for all of the keywords found in the meeting description.

Number of Conflicts between the meeting and other items in the user's schedule.

* **Number of Conflict Participants:** the total number of participants in meetings which conflict with the proposed meeting. (Only "internal" conflicting meetings count in this and the other following participant importance totals and averages, as those are the only types of meeting for which the system has participant information.)

* **Difference in Participant Number = Number of Participants − Number of Conflict Participants.**

* **Total Conflict Participant Importance:** the total of the user's importance ratings for all of the participants in all of the conflicting meetings.

* **Difference in Total Participant Importance = Total Participant Importance − Total Conflict Participant Importance.**

* **Average Conflict Participant Importance:** the average of the user's importance ratings for all of the participants in all of the conflicting meetings.

* **Difference in Average Participant Importance = Average Participant Importance − Average Conflict Participant Importance.**

* **Total Conflict Keyword Importance:** the total of the user's importance ratings for all of the keywords found in the descriptions of all of the conflicting meetings.

* **Difference in Total Keyword Importance = Total Keyword Importance − Total Conflict Keyword Importance.**

* **Average Conflict Keyword Importance:** the average of the user's importance ratings for all of the keywords found in the descriptions of all of the conflicting meetings.

* **Difference in Average Keyword Importance = Average Keyword Importance − Average Conflict Keyword Importance.**

* **Scheduling Lead Time:** the number of days between the receipt of the invitation and the date of the meeting itself.

Activity Immediately Before the Meeting i.e., Free time, other meetings, arrival, etc.

* **Time for Activity Before the Meeting:** how long a contiguous block of time is dedicated to the **Activity Immediately Before the Meeting**.

Activity Immediately After the Meeting i.e., Free time, other meetings, arrival, etc.

* **Time for Activity After the Meeting:** how long a contiguous block of time is dedicated to the **Activity Immediately After the Meeting.**

* **Time Already Scheduled for Meetings** the day of the proposed meeting. (This and the next feature refer only to "internal" meetings.)

* **Weekly Total Time Already Scheduled for Meetings** during the week containing the proposed meeting.

* **Time Already Scheduled** the day of the proposed meeting. (This and the next feature refer to everything on the user's schedule.)

* **Weekly Total Time Already Scheduled** during the week containing the proposed meeting.

Request Type: indicates whether this is an initial invitation, a rescheduling attempt for a meeting or a re-invitation to a meeting for which the user requested renegotiation but waз turned down.

Appendix B

Role-Playing Scenario — Personal Profiles

Pat: Group Manager

You manage a group of 4 Programmer/Analysts on the Foo software project, named Alan, Beth, Carl and Dianne. You have also hired 2 student interns, named Sam and Tina.

Your manager is Chris. He also manages 3 other groups like yours.

The company has just been reorganized, so this group of people is new to you, though you have managed similar projects before. Through the grapevine, you've heard that Alan and Beth are independent workers, but that Carl prefers a lot of feedback. Dianne is new to the company, and you haven't met her. Both of the students are also new, but you have met them, and think they will turn out to be pretty independent, but will likely need more technical help than your regular group members.

You will do a little technical work on the project, but your main responsibility is to ensure that your group remains organized and efficient. You are also responsible for dealing with upper Management.

You tend to work more than your required 40 hours/week, which you believe to be in part responsible for the fact that you have been made a manager at a relatively young age. You usually arrive by 8 and leave sometime around 6:30 when your work is done.

It is standard in the Bar company for groups such as yours to meet once a week for an hour to discuss everyone's progress. You will want to set up a regular group meeting soon. You should also expect that Chris will want to set up a weekly meeting with you and a monthly meeting with the entire group, again to check on status. You may also want to set up regular individual meetings with your group members, depending upon their individual needs.

In your spare time you like to golf and play squash.

Chris: Senior Manager

You supervise 4 projects including the Foo project.

You work long hours, usually centered somewhere around 8-5, but often working up to 60 hours a week.

You are getting ulcers and see a doctor frequently.

Alan: Programmer/Analyst

You are a Programmer/Analyst on the Foo project.

You prefer to work 8-4, with more flexibility in the morning than the afternoon, as you have to leave by 4:30 at the latest to pick up your child at daycare.

You like having all of your meetings organized in clumps, to leave good chunks of time in which to actually work.

You are independent and rarely need or want feedback from your manager.

Beth: Programmer/Analyst

You prefer to work 9-5 most days, but are quite flexible about that.

You hate to have more than one meeting in a row, unless they are both very short.

You prefer to keep at least one day a week free from meetings to be sure you get a good clump of time to do "real work." You like it if Wednesday is your meeting-free day. As much as possible, you like to avoid meetings altogether, and make an effort to minimize their number in your group.

You are an independent worker, rarely wanting or needing reassurance from above.

Carl: Programmer/Analyst

A morning person, you prefer to work 7-3, and have regular squash dates with friends outside the company on Tuesdays and Fridays at 4p.m., as well as whenever else you can manage it.

You thrive on positive reinforcement — you like to hear about it when you have done something well, and like encouragement along the way that you are on the right track.

Dianne: Programmer/Analyst

You are new to the Bar company, having been hired during the reorganization.

A real night person, you tend to work 10-6, and really hate to have to come in any earlier than 10.

Sam: Student Intern

You don't really know what hours you will end up keeping — at school you kept very varied hours. You are quite eager to impress the Bar company so they will hire you when you graduate, so you are quite willing to be flexible about your work hours so that you can be accessible to others. In general though, you are quite the night person and will rarely arrive before noon unless you have a meeting.

You are independent, rarely seeking or needing the approval cf your supervisor. You do however want to make sure you have been keeping people happy in the long-term to make sure you make the right impression.

Tina: Student Intern

You are a morning person, and prefer a 7-3 working day, but are quite flexible about that.

You are an independent worker and would prefer to be left to do your work rather than constantly checking in with superiors.

Bibliography

[1] Chin, D., Intelligent Interfaces as Agents, in: J. Sullivan and S. Tyler (eds.), *Intelligent User Interfaces*, (ACM Press, New York, NY, 1991).

[2] Creecy, R., Masand, B., Smith, S., and Waltz, D., Trading MIPS and Memory for Knowledge Engineering: Classifying the Census Returns on the Connection Machine, in: *Communications of the ACM* 35, 8 (August 1992), pp. 48-64.

[3] Cypher, A., EAGER: Programming Repetitive Tasks by Example, in: *CHI '91 Conference Proceedings*, New Orleans, LA (1991) 33-39.

[4] Dent L., Boticario J., McDermott J., Mitchell T. and Zabowski D., A Personal Learning Apprentice, in: *Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA (1992) 96-103.

[5] 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).

[6] Grudin, J., Why Groupware Applications Fail: Problems in Design and Evaluation, in*Proceedings of the 1988 Conference on Computer-Supported Cooperative Work*, Portland, OR, (1988).

[7] Kaye, A.R. and Kram, G., Cooperating Knowledge-Based Assistants for the Office, in: *ACM Transactions On Office Information Systems* 5, 4 (1987).

[8] Kozierok, R. and Maes, P., A Learning Interface Agent for Scheduling Meetings, in: *Proceedings of the 1993 International Workshop on Intelligent User Interfaces*, Orlando, FL (1993) 81-88.

[9] Lai, K., Malone, T. and Yu, K., Object Lens: A "Spreadsheet" for Cooperative Work, in: *ACM Transactions on Office Information Systems*, 6, 4 (1988).

[10] Lieberman H., Mondrian: A Teachable Graphical Editor, in: A. Cypher (ed.), *Watch what I do: Programming by Demonstration*, in press (MIT Press, Cambridge, MA, spring 1993).

[11] Levy, S., Let Your Agents Do the Walking, in: *MacWorld* 10, 5 (May 1993).

[12] Maes, P. and Kozierok, R., Learning Interface Agents, to appear in: *Proceedings of AAAI-93*, Washington, DC (1993).

[13] Metral, M., Design of a Generic Learning Interface Agent, Bachelor's Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge, MA (1993).

[14] Myers, B., *Creating User Interfaces by Demonstration*, (Academic Press, San Diego, CA, 1988).

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

[16] Stanfill, C., Learning to Read: A Memory-Based Model, in: *Proceedings of the 1988 DARPA Workshop on Case-Based Reasoning* (1988).

[17] Sullivan, J.W. and Tyler, S.W. (eds.), *Intelligent User Interfaces*, (ACM Press, New York, NY, 1991).