Chapter 1

Interfaces that Learn: A Learning Apprentice for Calendar Management

Jean Jourdan, Lisa Dent,
John McDermott, Tom Mitchell,
David Zabowski

1. Introduction

A chief obstacle to widespread proliferation of knowledge-based systems is the cost of software development and maintenance. For example, the XCS systems used by Digital to configure computers has a staff of approximately sixty people who maintain and update the knowledge base used by this system (Barker and O'Conner, 1989). While the development and maintenance overhead is worth the cost for this particular application, there are many other potential applications where high costs prohibit use in practice.

In this chapter, we consider an approach to solving this knowledge base development and maintenance problem by constructing systems that learn from their users. In particular, we present a learning apprentice system, as defined in (Mitchell et al., 1985):

> We define *learning apprentice systems* as the class of *interactive* knowledge-based consultants that directly assimilate new knowledge by observing and analyzing the problem-solving steps contributed by their users through their *normal* use of the system.

To see the advantages of such a learning apprentice, consider the problem of developing a computer program to assist a secretary in maintaining an online calendar of meetings for a particular university faculty member. This program is to provide an interface that allows editing the calendar (e.g., adding, moving, deleting meetings). We also want it to provide useful advice for each of the editing operations, such as when to schedule a specific meeting, how long it should last, where it should be located, whether to move an existing meeting to make room for the new meeting, whether and when to send reminder messages to the participants, and so on. Several difficulties arise:

- **Problem 1: The elusive specification.** Typical software engineering practice involves writing a software specification, then implementing it. The problem here is that the specification for the program is simply not known. For instance, given a request to schedule a new meeting, by what criterion should the program decide whether to bump an existing meeting to make room for the new meeting? By what criterion should it choose the duration of the meeting? In interviewing faculty members and secretaries, we have found that neither could provide a crisp specification for how the program should behave, despite the fact that both make such decisions daily.

- **Problem 2: User customization.** Suppose we wish to provide the calendar assistant to all the secretaries and faculty in the department. Since each individual has his/her own preferences (e.g., for choosing how long to meet with a particular graduate student), the knowledge base will have to be customized for each individual user.

- **Problem 3: Times change.** Even once the initial knowledge base is developed for a particular user, times will change. Once the summer ends and the new semester begins, the regularities in scheduling meetings may change. For example, a particular student may have passed her thesis proposal and now require fewer but longer meetings as a result. The cost of keeping the knowledge base and decision-making rules up to date can become the dominant one in many applications.

Learning apprentices offer a potential solution to all these difficulties. To the degree that it can observe and learn from the decisions of its users, a learning apprentice will be able to infer the desired program specification (and implementation) by itself, as well as adapt to the idiosyncratic and changing specifications of different users. Given this, the key technical issues we consider here are:

1. Can a convenient interactive assistant be structured so that it can

capture useful training examples of users' decisions as well as the information on which these decisions are based?

2. Can a learning mechanism be developed to correctly *generalize from such training examples* to formulate a useful decision-making strategy of its own?

This chapter explores these two issues in the context of a learning apprentice for calendar management, called CAP. We show here that CAP provides a convenient assistant that is able to capture useful training examples. The primary evidence in support of this claim is that CAP is presently in daily use by one secretary in our department, regularly collecting such training examples. We also show that it is possible to automatically generalize from the training examples acquired by CAP, to learn rules comparable to those developed by hand. This claim is supported by experimental results applying two different generalization methods (ID3 and Backpropagation) to training data summarizing meetings scheduled by this secretary over a five month period. Finally, we suggest that the approach embodied in CAP can be adapted to a variety of applications where personally customized knowledge-based assistants will be of use.

There have been a number of previous attempts to develop learning apprentice systems in various application domains. Mitchell, et al. (Mitchell et al., 1985) describe a learning apprentice for digital circuit design, called LEAP. LEAP provides interactive advice on how to decompose boolean circuit specifications into circuit submodules, resulting eventually in a gate-level circuit design. LEAP collects training examples whenever its advice is accepted or overridden by the user, and is able to learn new tactics for circuit decomposition from these training examples. Segre (Segre, 1988) describes a learning apprentice for robot planning, called ARMS. ARMS provides a user interface to a simulated robot, which a user may instruct to perform simple block-stacking tasks. ARMS generalizes from these user-supplied plans to form general planning strategies for subsequent use. Schlimmer describes a learning apprentice for document cataloging, called REDSTONE (Schlimmer, 1990). REDSTONE learns a grammar for translating a document abstract into library catalog form. Kodratoff and Tecuci describe a domain-independent learning apprentice called DISCIPLE (Kodratoff and Tecuci, 1986), which they have used for designing detailed manufacturing technologies, for designing assimilation strategies for new products, and for aiding the manager of a computer center. (Nakauchi et al., 1991) describes a prototype learning apprentice to assist users in filling in purchase orders, which learns preferences (e.g., which vendor to use for which parts).

These earlier learning apprentice systems explored a variety of generalization methods and various modes of interacting with a user (ranging from passive observation of the user, to querying the user regarding his or her reasons for a particular decision). However, no learning apprentice system has yet passed the important threshold of successful deployment within a real user community. Thus, no direct evidence has yet been collected to support the conjecture that learning apprentices can enable efficient development of a new generation of knowledge-based systems. The typical reasons for this appear to be that (1) in some cases users find the system does not offer sufficient value to make it worth learning and using, and (2) in some cases the learning methods are not sufficiently powerful to acquire high levels of competence. We present preliminary evidence here that CAP is viewed as useful by users, and that its learning methods compare favorably to manual methods for knowledge base development.

The following section describes the calendar management problem addressed in this chapter. Section 3 describes the design of CAP, the organization of its knowledge base, and its learning mechanisms (this section may be skimmed or skipped by readers who are primarily interested in results of the learning experiments). Section 4 presents experimental results showing the effectiveness of two learning procedures, and the final section analyzes the potential for learning apprentice advisors for a broad class of related problems.

2. The Problem

The problem considered here is to provide an intelligent computer assistant to help people manage calendars. A calendar management assistant should provide the user with a set of basic editing operations for adding, deleting, moving, and displaying meetings, as well as operations such as sending electronic mail reminders to meeting participants, or sending electronic mail requests to schedule rooms. It should also provide decision-making advice to the user. This advice might take the form of suggesting parameter values for operations initiated by the user. For

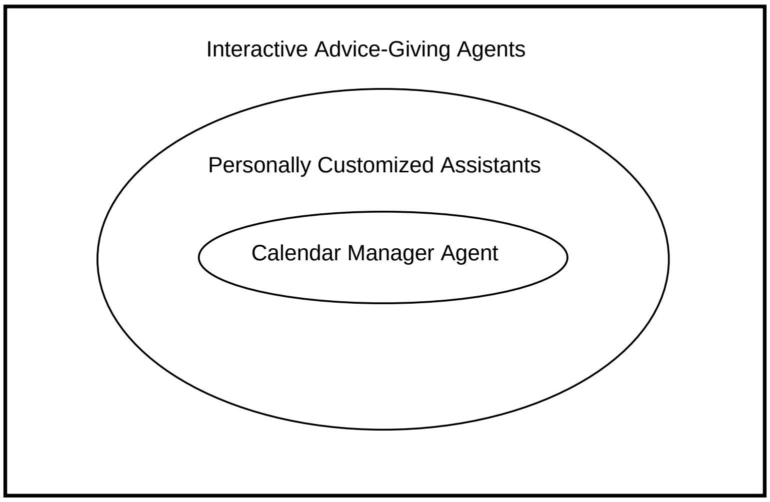

Figure 1. Classes of Interactive Advice-Giving Agents.

example, when the user initiates the AddMeeting operation, the agent should provide advice on the meeting time, location, etc. Beyond this, the agent could itself take the initiative to recommend whether and when certain operations should be performed (e.g., suggesting moving a meeting to make room for a new one, or suggesting sending an electronic message reminding the participants of the meeting time). For decisions at which it is especially competent, the agent could automatically perform the operation rather than trouble the user for approval of its advice. By observing the user's reaction to its advice, as well as the operations commanded by the user, the agent should learn to provide advice which corresponds more closely to the decision making strategies of its users.

Our initial implementation of CAP provides only part of the desired functionality described above. In particular, the current implementation of CAP provides advice only regarding parameter values for operations initiated by the user, and does not yet interface to the electronic mail system. To date, we have focussed primarily on allowing CAP to provide competent advice for the duration and location of newly added meetings. However, even with this partial set of capabilities we have demonstrated that CAP is useful, able to capture informative training examples, and able to learn from these examples. We see no obvious roadblocks to broadening the types of operations and advice that CAP supports in the future.

As depicted in Figure 1, a calendar management agent as described above is one kind of personally customized assistant, or more generally a kind of interactive advice-giving agent. We believe that the calendar management task is representative of a large class of tasks which require personally customized knowledge for high performance, and the goal of our research is to develop an approach that will be viable for this large class of tasks.

Note that the function of the learning calendar management agent is related to the function of typical scheduling algorithms (e.g., (Sadeh, 1991)). Scheduling algorithms accept as input a set of events to be scheduled, and precisely formulated constraints on the timing of these events. They produce as output a schedule that satisfies the given constraints. While calendar management might be viewed as a problem of this type, we believe that to do so misses the essence of the problem. The problem in calendar management is not so much to solve a set of well-formed constraints as it is to determine what these constraints are in the first place. For example, the most difficult problem for our system is not to determine that the newly requested meeting should be scheduled for Monday from 2-2:30 pm in order to satisfy the user's desire to meet for 30 minutes with a book publisher, but rather to learn the constraint that this particular user prefers to meet for 30 minutes with book publishers. Thus, the key difference in philosophy between traditional scheduling algorithms and the work described here is that we are concerned with how to organize scheduling assistants so that they learn the scheduling constraints from observing their users.

3. The Program

CAP is an implementation of a learning calendar management agent as described in the previous section. As shown in Figure 2, it provides an interactive editor for operating on the state of the calendar, contains a knowledge base capable of advising the user in selecting parameters such as duration and location of new meetings, and contains learning mechanisms for acquiring new advice-giving rules by generalizing from observed actions of the user.

Table 1 lists the available calendar operations. Each operation can be invoked directly by the user, via an editor command. CAP provides

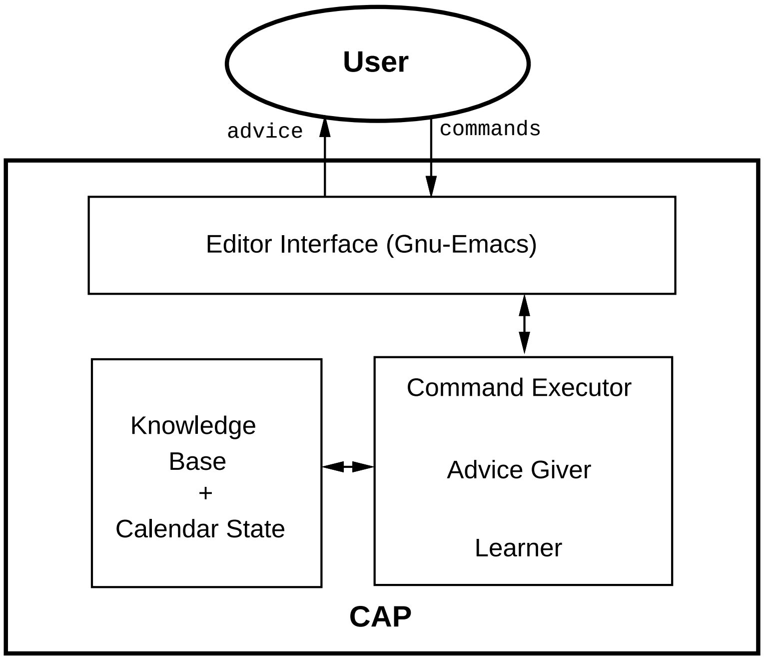

Figure 2. Organization of CAP.

Operation	**Parameters**
Changing State:	
Add Meeting	Attendees, Topic, SubTopic, Duration, Date, StartTime, Location.
Delete Meeting	Date, StartTime.
Move Meeting	Date, StartTime, NewDate, NewStartTime.
Change Meeting	Date, StartTime, Attendees, Location, Topic, SubTopic, Duration.
Annotate Meeting	Date, StartTime, Annotation.
Add Upcoming Event	Date, UpcomingEvent.
Delete Upcoming Event	Date, UpcomingEvent.
Observing State:	
View Meeting	Date, StartTime.
Go To Week	Date.
List Upcoming Events	No parameters.
Print Week	No parameters.
Print Month	No parameters.

Table 1. CAP Operations On Calendar State. Advice consists of recommended values for operation parameters.

advice by suggesting values for the parameters of each user-invoked operation. For example, when the user invokes the AddMeeting operation, CAP suggests values for the `Topic`, `SubTopic`, `Duration`, `Date`, `StartTime`, and `Location` of the meeting.

3.1 User Interface

The two key issues in designing a user interface for a learning apprentice such as CAP are (1) user acceptance, and (2) ability to capture training examples. To maximize usability and familiarity (and thus user acceptance) the interface was designed around an editor familiar to many users: Gnu-Emacs. Furthermore, the visual display of the calendar is based on a display evolved over several years by a secretary who had been using Gnu-Emacs to maintain an online calendar (and who has since become the first fulltime user of CAP).

CAP provides user commands to invoke each of the operations listed in Table 1. Each CAP command is defined as an Emacs editor macro, which communicates with a subordinate Lisp process which runs a frame-based system called Theo (Mitchell et al., 1991). Theo provides the representation, inference, and learning mechanisms on which CAP is built. When the user invokes an editor command (e.g., AddMeeting), CAP prompts the user for values of the various relevant parameters. For each such prompt, the CAP first queries Theo for advice and presents this advice to the user in the form a suggested value for the desired field (e.g. `Duration` = 60 minutes). The user can either accept this advice as correct or override it with another value. Once the user has supplied a value, it is passed back to Theo, which updates the calendar state according to the user's command, and stores the training example for later use.

Figure 3 shows the display presented by CAP to the user. In this figure, the user is in the process of adding a new meeting. CAP has prompted the user for the value of the meeting `Duration`, and has suggested the value of 60 minutes. In this case, the user overrides this advice by entering the value 120. Notice the calendar display is organized by weeks. At the top of the display for a week is a list of upcoming events for the week, which have been previously entered via the AddUpcomingEvent command. A display containing the complete list of all upcoming events is also available. Because of the limited amount of display space available, only the most important information about

```
04/01/1991  DEADLINE FOR CALENDAR PAPER TO MINTON!
04/04/1991  Foster Provost's thesis proposal

-----------------------------------------------------------------------------
           |   Monday   |  Tuesday   | Wednesday  |  Thursday  |   Friday   |
   TIME    |    4-1     |    4-2     |    4-3     |    4-4     |    4-5     |
---------- |------------|------------|------------|------------|------------|
           |            |            |            |            |            |
  8:00     |            |            |            |-  Mars-Rove|            |
           |            |            |            |Frc-Conf-Roo|            |
  8:30     |            |            |            |Research    |            |
           |            |            |            |------------|            |
  9:00     |            |            |            |            |            |
           |            |            |            |            |            |
  9:30     |            |            |            |*  Provost  |            |
           |            |            |            |Alumni-338  |            |
 10:00     |            |            |*  Tan      |Thesis-Propo|            |
           |            |            |Weh5309     |            |            |
 10:30     |            |            |            |     \/     |            |
           |            |            |            |------------|            |
 11:00     |            |            |            |*  Hsu Simmo|-  Schlimmer|
           |            |            |            |Weh5126     |Weh5309     |
 11:30     |            |            |            |Research    |Research    |
           |            |            |            |------------|------------|
 12:00     |-  Mason Chr|*  Many     |            |*  Many     |            |
           |Weh5309     |Ph225b      |            |Ph225b      |            |
 12:30     |Research    |Course      |            |Course      |            |
           |------------|            |            |            |            |
  1:00     |            |     \/     |*  Whittaker|     \/     |-  Lin      |
           |            |------------|Weh5327     |------------|Weh5309     |
  1:30     |            |            |Funding-Mtg |            |-  Ringuette|
           |            |            |            |            |Weh5309     |
  2:00     |*  Huang    |-  Zabowski |     \/     |            |-  Zabowski |
           |Weh5309     |Weh5309     |------------|            |Weh5309     |
  2:30     |-  Dent     |            |*  Dent     |            |-  Watanabe |
           |Weh5309     |            |Weh5309     |            |Weh5309     |
  3:00     |-  Theo-Grou|-  Chalasani|-  Many     |            |Research    |
           |Weh4605     |Weh5309     |Weh4605     |            |------------|
  3:30     |Research    |*  Jourdan D|Course      |*  Jourdan  |-  Many     |
           |            |Weh5309     |            |Weh5309     |Weh4605     |
  4:00     |    \/      |            |     \/     |-  Cheng    |Seminar     |
           |------------|            |     \/     |Weh5309     |------------|
  4:30     |-  Yamanouch|            |     \/     |Research    |            |
           |Weh5309     |            |------------|------------|            |
  5:00     |Research    |            |            |            |            |
           |------------|            |            |            |            |
  5:30     |            |            |            |            |            |
           |            |            |            |            |            |
  6:00     |            |            |            |            |            |
           |            |            |            |            |            |
-----------------------------------------------------------------------------


   TIME    |    4-1     |    4-2     |    4-3     |    4-4     |    4-5     |
Duration: [60] 120
```

Figure 3. A Typical CAP Display. The user is adding a meeting. At the bottom of the screen CAP prompts for the duration of a meeting and offers advice that it should be 60 minutes. The user overrides this advice by typing the desired value of 120.

Objects Describing the Calendar State:

DAY : a day in the calendar e.g. apr-1-1991

EVENT: an event (meeting) in the calendar

Objects Describing Permanent Background Knowledge:

PEOPLE : a person with whom the calendar owner meets

Objects Describing User-Generated Tasks:

REQUEST : a request for an update to the calendar

Table 2. Types of Objects Represented by CAP.

---

a meeting is displayed. The ViewMeeting command allows the user to request a display of all information about a particular meeting.

For many commands, the legal parameter values are restricted by CAP (e.g., there is a particular list of legal values for meeting `Location`). For these parameters CAP presents not only a suggested value, but also (upon demand) all of the legal values for this parameter. Constraining the interface in this way leads to more structured meeting descriptions, which, in turn, lead to more successful learning by the system.

3.2 Knowledge Base Organization

CAP's knowledge base consists of two parts: the objects or entities which represent the elements needed to manage a calendar, and a set of rules which define relations on these objects.

The Objects All the objects represented in the system are specializations of the four object types summarized in Table 2. As this table shows, the state of the calendar is represented by a collection of DAY entities and a collection of EVENT entities which represent meetings occurring on those days. An example DAY is:

```
apr-1-1991
   Generalizations: monday
   WeekDay?: yes
   Events: request-4-1, event4-2, event2-2, event3-2,
           event1-2
   RegularEvents: event1, event3, event2, event4
   SortedEvents: (event1-2 1200 1300),
                 (request-4-1 1400 1430),
                 (event2-2 1430 1500),
                 (event3-2 1500 1630),
                 (event4-2 1630 1730)
   ParityWeek: odd-week
   Date: (1 4 1991)
```

The name of this DAY entity is apr-1-1991. Each entity is represented by its slots and their values. For example, the first slot shown above, `Generalizations`, has a value `monday`. This indicates that apr-1-1991 is a member of the more general class of days which are mondays. The second slot represents the fact that apr-1-1991 is a weekday.[1] The slot `Events` stores a list of the meetings occurring on this day. One meeting, event2-2, is an instance of a regularly scheduled meeting (thus its template, event2, is stored in `RegularEvents`). In contrast, the meeting represented by request-4-1 is not an instance of a regularly scheduled meeting, but was scheduled as a one-time meeting. The meetings are also sorted by start time; event2-2 starts at 14:30 and ends at 15:00, request-4-1 starts at 14:00 and ends at 14:30.

Each EVENT is also represented by an entity with slots and slot values. For example, the following event represents a meeting occurring on April 1, 1991 at 14:30 for 30 minutes in room Wean Hall 5309 with Dent about a research problem related to the Theo project.

```
event2-2
   Location: weh5309
   Attendees: Dent
   Topic: research
   SubTopic: theo-project
   Duration: 30
   StartTime: 1430
```

1. Some slot values (e.g., `WeekDay?`) are automatically inferred from the values of other slots (e.g., `Generalizations`).

```
   Generalizations: event2
   DateEvent: apr-1-1991
   Creator: event2
   Regular: t
```

The DAY and EVENT objects are sufficient to describe the state of the calendar. However, in order to be able to give advice for command parameters, more information is needed. This background knowledge consists of facts about the people and activities with which the calendar owner is involved. For example, the decision of how long a meeting should last may be based on information about the attendees of the meeting: Who are they? What position do they have? Where are they working? How often do they meet? and so forth. Information about meeting attendees is stored in entities representing PEOPLE. For example, the following entity describes Dent, a third year graduate student at CMU whose advisor is Tom Mitchell.

```
Dent
   Generalizations: grad-student, tom-group-member
   Position: grad-student
   Individual?: yes
   InTomsGroup?: yes
   AtCMU?: yes
   Office: dh1323
   Advisor: Mitchell
   NumberofYearsAtCMU: 3
```

The last type of object in the knowledge base is the entity REQUEST. Requests represent instances of user-invoked editing commands, along with information about the meetings to which they pertain. They are the connection between the interface and the knowledge base, and they correspond to the training examples used for learning. An example request is shown below:

```
request-4
   Generalizations: requests
   Action: AddEvent
   ActionTime: 1116
   ActionDate: Apr-2-1991
   Result: request-4-1
```

```
   Attendees: cheng
   Duration: 30
   Day: Apr-3-1991
   StartTime: 1730
   Location: weh5309
   Topic: research
   SubTopic: misc-research

   ModifiableBySecretary?: yes
   DesiredDuration: 60
   NextSimilarMeeting: unknown
   Urgency: 2
   NumberOfPeople: 1
   SinglePerson?: yes
```

The above entity represents the user's request to add a meeting for Cheng on April 3 at 5:30 in Wean Hall 5309. There are three kinds of information stored in the REQUEST (separated in the above example by spaces). First there is information about the request itself. This includes the type of action requested (e.g., AddEvent), the time and date when the request occurred, and the outcome of the request (e.g., it succeeded, creating the event request-4-1 as a side effect). Second, there is information which corresponds to the values of command parameters for which the user is prompted (e.g., `Attendees`, `Duration`). Finally there is extra information inferred automatically by CAP, which is used when providing advice for this request (e.g the estimated `Urgency` of the requested meeting).

Each of these slot instances is also an entity which may have its own (sub)slots and values. CAP associates an AdvisedValue and GivenValue subslot with each of the slots corresponding to command parameters. These subslots record the parameter value advised by CAP and the value specified by the user, respectively. For example, the `Duration` associated with a particular request may have both `AdvisedValue` of 60 and a `GivenValue` of 30.

```
request-4
   Duration: 30
      AdvisedValue: 60
      GivenValue: 30
```

PEOPLE	DAY	REQUEST	EVENT
AtCMU?	RegularEvents	Action	StartTime
KnownByTom?	Events	Result	EndTime
Position	WeekDay?	ActionTime	Duration
Busy	Date	ActionDate	Topic
Projects	BusyNextWeek	Date	SubTopic
Office	ParityWeek	StartTime	Attendees
InTomsGroup?	SortedEvents	Notes	Day
NextMeeting		Duration	Location
HasRegularMeeting?		Topic	Creator
LastMeeting		SubTopic	Regular
Institution		Location	
Department		Attendees	
Advisor		Urgency	
TomInCommittee?		SinglePerson?	
NumberOfYearsAtCMU		Number of People	
Individual?		PreviousSimilarMeeting	
		NextSimilarMeeting	
		ModifiableBySecretary?	
		DesiredDuration	
		Day	
		DayInWeek	

Table 3. Features of Different Entities.

As we have seen, each entity in the knowledge base is characterized by a set of features called slots. A summary of the entities and their associated slots is given in Table 3. Note that in this knowledge base, the calendar owner's name is Tom. Thus, the slot name `KnownByTom?` refers to whether the person is known to the calendar owner. We intend to generalize the system to support multiple calendar owners, and to replace slots such as `KnownByTom?` by slots such as `KnownByCalendarOwner?`.

The Rule Base The knowledge base contains a set of rules for supplying advice for calendar commands. These rules were created by talking with two expert calendar managers. Currently CAP contains about 110 rules for parameters of the AddMeeting command[2]. Writing these rules took about two person-weeks of effort, and we feel the rules represent a reasonably complete summary of the knowledge of our experts. While additional effort might further improve the quality of these rules, we believe we are at the point of diminishing returns and that further effort will produce only modest improvements in the quality of the rule set.

Table 4 shows a sample of these hand-coded rules for predicting the `Duration` of a meeting. There are two stages to the computation of the `Duration` of a meeting. First, the `DesiredDuration` is found based on static features of the meeting. This value refers to the duration that would be prefered if there were no conflicting demands on time of the participants. Second, the current calendar is checked to see how busy the calendar owner is. If the owner is very busy, the final computed `Duration` is shorter than the `DesiredDuration`; otherwise the `Duration` is simply equal to the `DesiredDuration`. Note that the rules are order dependent.

Two kinds of predicates are tested by these rules. The first kind of predicate tests information directly available from the knowledge base (for example, `Position`). Other predicates are computed by rules from the basic knowledge (for example, `SinglePerson?`). Thus one can trace the dependencies of the advised value for `Duration` through several levels. The dependency of advised `Duration` on other features, based on all the hand-coded rules, is shown in Figure 4.

2. Rules for other calendar commands such as DeleteMeeting and MoveMeeting have not yet been fully developed.

```
If   The Topic of ?r is committee-mtg;
Then the DesiredDuration of ?r is 60.

If   The Topic of ?r is funding-mtg;
Then The DesiredDuration of ?r is 30.

If   The SinglePerson? of ?r is yes, and
     The Attendees of ?r is ?a, and
     The Position of ?a is grad-student;
Then The DesiredDuration of ?r is 30.

If   The Attendees of ?r is ?a, and
     The AtCMU? of ?a is no, and
     The KnownByTom? of ?a is yes;
Then The DesiredDuration of ?r is 60.

         .............

If   The DesiredDuration of ?r is ?d, and
     The ModifiableBySecretary? of ?r is yes, and
     ?r > 30, and
     Day of ?r is ?x, and
     BusyNextWeek of ?x is true, and
     ShorterDuration of ?d is ?d2
Then The Duration of ?r is ?d2.

If   The DesiredDuration of ?r is ?d;
Then The Duration of ?r is ?d.

The default Duration of ?r is 30.
```

Table 4. Selected Rules for Duration.

Figure 4. Dependency of Meeting Duration on other Knowledge.

3.3 Learning Mechanisms

Each time the user enters the value of some parameter of some requested calendar operation, CAP treats this input not only as a parameter value, but also as a training example. For instance, if the user executes the AddMeeting command, and enters a `Duration` of 60 minutes, then CAP obtains a very specific training example of a situation in which it should recommend a value of 60 for `Duration`. This training example includes the detailed state of the calendar, as well as any other parameter values which the user may already have entered for this command (e.g., before entering `Duration`, the user will already have been prompted for the `Attendees`, `Topic`, and `SubTopic` of the meeting).

For each parameter of each user command, CAP thus faces a learning problem: generalize from the specific encountered training examples to formulate a strategy for providing subsequent advice. The measure of correctness for this advice in our experiments is agreement with the user's decisions. To date we have experimented with two inductive learning methods which are provided by the THEO system: a symbolic decision-tree learning method which is a variant of ID3 (Quinlan, 1986) (Quinlan, 1987), and a connectionist algorithm using back-

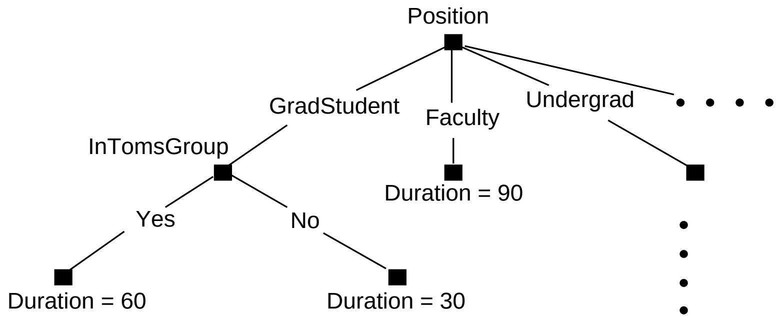

Figure 5. Part of an ID3 Decision Tree for Predicting Meeting Duration.

propagation (Rumelhart et al., 1986). Both these methods have previously been shown to be effective in learning from examples (see, for example, (Fisher and McKusick, 1989) (Mooney et al., 1989)).

ID3 ID3 (Quinlan, 1986) (Quinlan, 1987) is an inductive algorithm which builds a decision tree for classification of objects. Given a set of objects with known classes and described by a fixed set of features, a decision tree is produced which predicts the class of a new object described by the same feature set. Each node in the tree represents a particular feature, and the branches from that node correspond to alternative values of the feature. Figure 5 depicts part of a decision tree for classifying the `Duration` of a calendar meeting. The root node of the tree represents the feature `Position` (position of the meeting attendee), and the branches depicted represent three possible positions: `Faculty`, `GradStudent` and `Undergrad`. The leaves of the tree represent the possible classes of the object; in the figure these are the duration times of 30, 60 and 90 minutes. A new object is classified by starting at the root of the decision tree and repeatedly taking the branch appropriate to the values of the new object, until a leaf is reached which gives the predicted classification of the object.

To construct a decision tree from a set of objects and classes, ID3 applies an information-theoretic evaluation function to select the single most discriminative feature. This feature becomes the feature tested at the root of the decision tree. ID3 then constructs a branch for each possible value of the feature and partitions the examples accordingly. For each of these branches it considers just the associated examples and determines the feature that will best discriminate these according to the target classification. The process is repeated recursively at each subtree until either all the examples are of the same class, or there are no more features to test[3].

Our variant of ID3 (Schlimmer, 1991) initially constructs a decision tree to discriminate between examples from different classes, and then converts the tree into a set of production rules. We use ID3 to construct rules which predict various calendar meeting parameters from other available features. To convert the tree into a set of rules, the algorithm constructs an initial rule for each leaf in the decision tree. The conditions of the rule are the features tested along the path from the root to the leaf together with their corresponding values. The action of the rule is the classification associated with the leaf. Each rule is individually pruned by iteratively testing the rule without each of its conditions. If removing a precondition does not increase the number of negative training instances covered, the rule is pruned (generalized) accordingly.

Back-Propagation Back-Propagation (Rumelhart et al., 1986) is an inductive algorithm which can be used to train a connectionist network for classification of objects. The network consists of an interconnected set of nodes which are typically arranged in 'layers'. Each node in a layer is connected to each node in the previous layer and to each node in the next layer, but not to nodes in the same layer. In the feed-forward networks which we use, each node computes a weighted sum of the values of its previous layer connections and passes the result along to the next layer.

A standard network consists of three layers of nodes: an input layer, in which nodes represent object features; an intermediate 'hidden' layer in which the nodes have no a priori meaning; and an output layer in which nodes represent possible classifications. The features of an object are encoded as one or more nodes in the input layer of the network, depending on the type of feature. Figure 6 depicts part of a neural network for classifying the `Duration` of a calendar meeting. Nominal features such as `Position` are encoded by using one node for each possible value of `Position` ((Sejnowski and Rosenberg, 1987)). When a particular object is to be classified by the network, the node which cor-

3. Note that some decision tree learning methods use a pruning heuristic to stop tree construction when the examples cannot be reliably discriminated based on the available features.

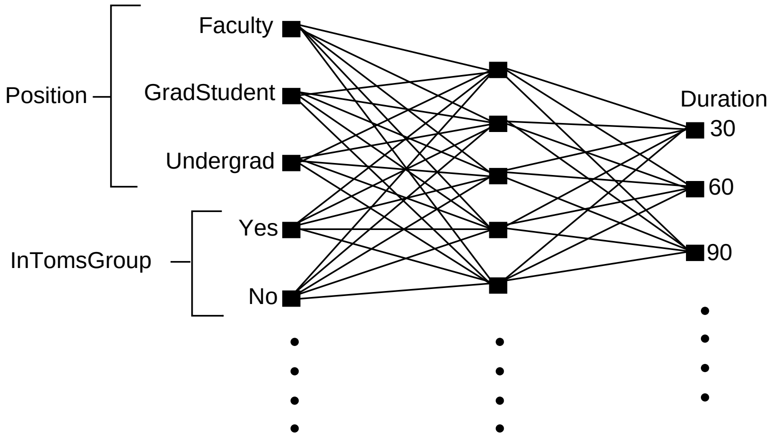

Figure 6. Part of a Neural Net for Predicting Meeting Duration.

responds to the value of `Position` for that object (say, `GradStudent`) will be set to 1, and all the other `Position` nodes will be set to 0. The nodes in the output layer of the network each represent one class; the node with the highest value is selected as the predicted class.

The network is trained by adjusting the weights of each node (which influence the weighted sums they compute). It is initialized with a random set of weights. As examples are presented to the network, back-propagation adjusts the node weights in a steepest descent search to improve the match between actual and desired classification output.

We built and trained a standard feed-forward network to predict each calendar parameter from a hand-selected set of input features (the same set used by ID3), using the standard back-propagation algorithm ((Rumelhart et al., 1986)) to train the network, with the learning rate set at 2.0, and momentum at 0.4. We keep the number of hidden units small (3 - 10 units) to avoid overfitting. At present we have no way of extracting rules from the final network.

4. Learning Experiments

As discussed in Section 1, the primary thesis of this research is that learning apprentice systems will allow the development of personally customized assistants which are not presently feasible because of the high cost of knowledge base development and maintenance. We present here a set of experiments demonstrating that CAP's learning mechanisms can acquire knowledge which correctly predicts a large proportion of the user's decisions regarding meeting `Location` and `Duration`. Furthermore, this learned knowledge compares favorably to the rule base we developed by hand for our initial user. We first describe the experimental design, then the experimental results.

4.1 Experimental Design

The most direct way of testing our thesis would be to monitor the success rate of advice given by CAP over time, in order to determine its success at learning. However, CAP has only been in routine use for approximately one month, and has not yet acquired sufficient training data to allow this direct test. Therefore, the experiments reported here are based on training data collected by hand, rather than by CAP. In particular, the training examples used here describe meetings of a single university faculty member which took place during the fall and spring semesters of 1990-91[4].

Because these examples were recreated from a text calendar file containing past meetings, some relevant information was no longer available. For example, for each training example of AddMeeting, CAP records the time at which the meeting is added to the calendar (not just the time at which the meeting is scheduled). However, this information was not available from the text calendar file from which we extracted training examples. This incompleteness of training information constrained our choice of which meeting parameters to learn to predict. We chose to predict meeting `Duration` and `Location`, because we suspect these parameters do not depend strongly on the missing information.

For each learning task, one must determine the set of features that will be used to describe the training examples (we shall refer to these as the 'input features', since these are used as inputs to the learned neural network or rules.). A very large set of input features is available to CAP. Each meeting can be characterized, for instance, by its date, start time, duration, topic, subtopic, its attendees, the office of each attendee, the position held by each attendee, the previous meeting held among these

4. These are meetings for the same faculty member, scheduled by the same secretary who is presently using CAP. Therefore, we believe they are a realistic approximation to the types of examples CAP will encounter in its current use.

INPUT FEATURES FOR PREDICTING DURATION:	
Feature set D1:	`Topic SinglePerson? *AtCmu? *InTomsGroup? *Position *KnownByTom?`
Feature set D2:	`NumberOfPeople SinglePerson? Topic Subtopic *AtCMU? *Busy *InTomsGroup? *NumberOfYearsAtCMU *Position *KnownByTom? **Duration`
INPUT FEATURES FOR PREDICTING LOCATION:	
Feature set L1:	`NumberOfPeople SinglePerson? Topic *InTomsGroup? *Position`
Feature set L2:	`DayInWeek Duration NumberOfPeople SinglePerson? StartTime SubTopic Topic *AtCMU? *InTomsGroup? *KnownByTom? *Position **Location`

* These are features of the attendees of the requested meeting.
** These are features of the previous meeting involving the same attendees.

Table 5. Features Used To Describe Training Examples.

attendees, the duration of this previous meeting, the meeting previous to that one, etc. All this information (and more) is accessible to CAP for each training example.

For each of the two predicted parameters (`Duration` and `Location`) we experimented with two sets of input features, as defined in Table 5. Feature set D1 is precisely the set of input features used by our hand-coded rules for predicting `Duration`, minus features which could not be easily handled by the learning methods. The feature `Attendees` was not included because it has so many different values (about 75), which causes the performance of the learning methods to degrade unless many more training examples are provided than we had available. Other features used by the hand-coded rules (e.g., `BusyNextWeek`) were omitted because the training data from the text calendar file did not contain values for these features. Similarly, L1 is the set of features used by the hand-coded rules for `Location`. Feature sets D2 and L2 are supersets of D1 and L1, extended to include a few additional features which we felt might be relevant, even though these features were not used by the hand-coded rules. The selection of features for D2 and L2 was based on experimentation. If too many features are included, the learning algorithms will not have enough training examples to distinguish the relevant features from the irrevelant ones and performance will suffer (this is known as 'overfitting the data'). If too few features are included, the learning algorithms will not be able to distinguish examples of different classes. Clearly, the selection of a good feature set is a critical part of the development of a learning apprentice.

A set of 180 training example meetings was used to train and test the learning algorithms. Of these 180, 120 were randomly selected for each run as training examples, and the remaining 60 were used as test cases. For each experiment, several runs (at least 4) were conducted and the results averaged. The experiments were run in 'batch' mode; all the training examples were presented to the algorithms at once as input.

4.2 Results

Figure 7 summarizes the results of the learning experiments for predicting meeting `Duration` and `Location`.

The `Duration` of a meeting has 6 legal values in the training data: 15, 30, 60, 90, 120 and 150 minutes. The most common value is 30 minutes

(44% of the meetings). The 21 manual rules suggest the correct value in 65% of the test examples. In contrast, ID3 creates 27 rules when using the same input features as the manual rules (i.e., feature set D1), and achieves on average 69% accuracy. When using the expanded feature set D2, ID3 creates 60 rules and the accuracy drops to 54%, perhaps due to overfitting the training data. The neural network created for the first set of input features has 32 input units, 6 hidden units and 6 output units. It produces correct advice 71% of the time. When the net is expanded to include more input features (74 inputs), it is correct 72% of the time. In both of the neural net experiments, back-propagation was run for 800 epochs, and the nets converged after about 400 epochs. Due to ambiguous test examples (examples with all features the same except for `Duration`), the maximum possible accuracy with feature set D1 was 79%, and for feature set D2, 87%.

The `Location` of a meeting has many more possible values (23) than `Duration`. However, the distribution of values is more skewed than for duration, as 71% of these meetings are in one location: the office of the calendar owner. The 11 rules for `Location` achieve 81% correct predictions on the test data. In contrast, ID3 created 32 rules using feature set L1, achieving 77% correct predictions. When using the augmented feature set L2, ID3 generated 45 rules and achieved only 71% correct predictions. However, many of the failures of this rule set were due to meetings to which no rule applied. When evaluated only for meetings to which the rules did apply, ID3 achieved 86% correct predictions using feature set L2. Backpropagation took longer (more epochs) to converge for `Location` than for `Duration`, presumably because of the greater skew in the distribution of meeting locations. We ran each feature set for 2000 epochs, and still had no evidence of overfitting (which we use as a signal that the net has converged). Due to ambiguous test examples (examples with all features the same except for `Location`), the maximum possible accuracy with feature set L1 was 91%, and for feature set L2, 99%.

Figures 4-1A and 4-1B summarize the results of predicting `Duration` and `Location` using the a number of methods:

1. **Default.** Predict the Duration (Location) will be 30 minutes (Room 5309). The default value is simply the most common value over the training examples.

Figure 7. Predicting Duration and Location of Test Examples

2. **Manual.** Apply the hand-coded rules for Duration (Location)
3. **ID3-D1.** Apply the rules learned by ID3 based on feature set D1 (L1); that is, only features which are also used by the manual rules.
4. **ID3-D1*.** Apply the same rules as in ID3-D1. However, measure the percentage of correct predictions only over those test examples to which some rule applies (as opposed to all test examples).
5. **BP-D1.** Apply the learned neural net based on features D1 (L1).
6. **Optimal-D1.** This indicates the maximum possible performance using feature set D1 (L1). This is limited by the fact that some examples have identical values for all features except for Duration (Location).
7. **ID3-D2.** Apply the rules learned by ID3 based on feature set D2 (L2).
8. **ID3-D2*.** Apply the same rules as in ID3-D2. However, measure the percentage of correct predictions only over those test examples to which some rule applies (as opposed to all test examples)
9. **BP-D2.** Apply the learned neural net, based on features D2 (L2).
10. **Optimal-D2.** This indicates the maximum possible performance using feature set D2 (L2).

Two observations are worth noting:

- Both the learned neural net and the learned ID3 rules performed well compared to the hand-coded rule sets. Note that this is despite the more limited representations employed by both learning methods. In particular, whereas the hand-coded rules are first-order Horn clauses, the rules learned by ID3 are restricted to a propositional representation. Thus, ID3 cannot represent rules such as 'the duration of this meeting should be the same as the duration of the previous meeting'. Furthermore, both ID3 and Back-Propagation have difficulty using input features with large numbers of possible values. The problem is the same as that of using too many features; many more training examples are required to differentiate relevant feature values from irrelevant values. In our version of ID3 once a

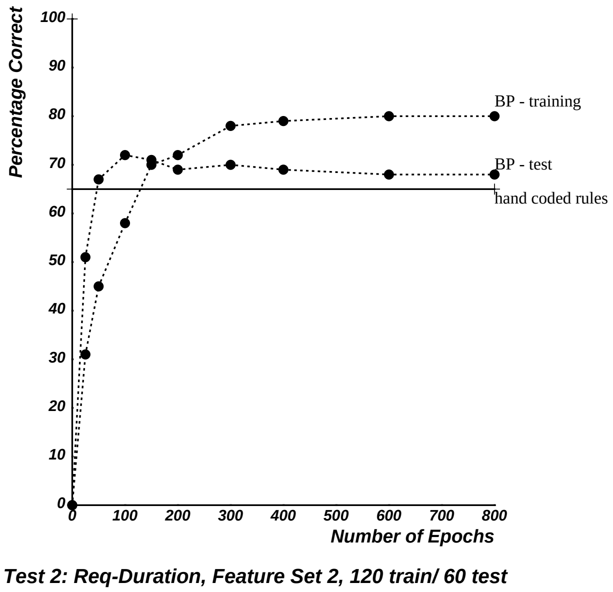

Figure 8. Training Curve for Backprop Predicting Duration from Feature Set 2. Curve labeled BP-training shows performance on training examples. Curve labeled BP-test shows performance on test examples. Solid line shows performance of hand-coded rules.

feature is chosen to be added to the decision tree, all the possible values of that feature must become branches[5]. As a result, we did not include meeting `Attendees`, which has about 75 different values, as an input feature, although it may be a good predictor.

- When the learning methods were allowed a larger set of input features to consider as possible predictors (i.e., using features sets D2 or L2), there was evidence of overfitting the training data due to an overly high ratio of features to training examples. In the case of ID3, this leads to a set of rules whose coverage of the test data is lower, though whose correctness is higher on those cases which are covered. In the case of the back-propagation algorithm, this results in the curve shown in figure 8. Notice that as training epochs increase performance increases monotonically in classifying the training set, but first rises and then falls on the test set.[6] This is due to the net overfitting the training data, to the detriment of generalizing correctly to cover the test set. It is a common phenomenon associated with neural network learning methods (Knight, 1989), which may be handled by terminating training once performance on the test set begins to degrade.

Table 6 shows a representative sample of the rules learned by ID3 for predicting `Duration`. Similarly, Table 7 shows rules for predicting meeting `Location`. In general, the learned rules were relatively simple, and most appear to be reasonable to the user and calendar owner. Some rules, such as the final rule for `Location` in Table 7, turn out to be statistical anomalies arising from the policy of ID3 to continue refining rules until it partitions the data as perfectly as possible. This rule, for example, covers just a single training example, and is unlikely to be useful in general.

There are many aspects of the Backpropagation algorithm which can be changed; for example, number of hidden units, learning rate, and encoding strategies. We have only investigated changing the number

5. At least one variation of ID3 has been proposed which avoids this (Cheng et al., 1988)
6. The BP-training curve is not the result of directly testing the net on the training cases, but rather an internal measure used by the back-propagation algorithm which tests output patterns against training data without decoding. Thus only the shape of the curve, not its absolute value, should be compared with the BP-test line .

```
If   The Position of Attendees of ?x is undergrad;
Then The Duration of ?x is 30.

If   The SubTopic of ?x is committee-mtg;
Then The Duration of ?x is 60.

If   The NumberOfPeople of ?x is 2, and
     The Position of Attendees of ?x is grad-student;
Then The Duration of ?x is 60.

If   The InTomsGroup? of Attendees of ?x is no, and
     The NumberOfPeople of ?x is 1, and
     The SubTopic of ?x is research, and
     The Position of Attendees of ?x is grad-student;
Then The Duration of ?x is 30.

If   The SubTopic of ?x is funding-mtg, and
     The Position of Attendees of ?x is funder;
Then The Duration of ?x is 90.
```

Table 6. Selected Rules Learned by ID3 for Duration.

```
If   The Position of Attendees of ?x is undergrad;
Then The Location of ?x is weh5309.

If   The Position of Attendees of ?x is dean;
Then The Location of ?x is weh4220.

If   The Position of Attendees of ?x is grad-student;
Then The Location of ?x is weh5309.

If   The SubTopic of ?x is thesis-proposal;
Then The Location of ?x is weh5409.

If   NumberOfPeople of ?x is 4, and
     The SubTopic of ?x is research;
Then The Location of ?x is frc-conf-room.

If   The SubTopic of ?x is interview;
Then The Location of ?x is weh5309.

If   The Position of Attendees of ?x is unknown, and
     The Topic of ?x is miscellaneous;
Then The Location of ?x is skibo107.
```

Table 7. Selected Rules Learned by ID3 for Location.

of epochs. Similarly, there are many variations of ID3 which may cope better with noisy data, overfitting of data, etc. Thus we cannot draw too much from the comparison of these learning methods. However, our initial results confirm those of (Fisher and McKusick, 1989) (Mooney et al., 1989): Backpropagation performs slightly better than ID3 on average with noisy data, but takes much longer to train. We are currently investigating the possibility of initializing the network from the hand-coded rules or ID3-generated rules, to try to improve the speed and accuracy of back-propagation.[7] One advantage of ID3 is the comparatively readable rules it produces; we are also looking into producing rules from the back-propagation network.

5. Summary and Analysis

The thesis of this research is that learning apprentice systems will enable the practical development of a variety of personalized software assistants which are not presently found in regular use due to (1) the costs of customizing such systems to each individual user or environment, and (2) the costs of maintaining systems as times change. Our argument in support of this thesis involves two claims:

1. It is possible to construct interactive assistants which users will find helpful even when the system's initial level of expertise is low, and which are able to *capture valuable training examples* of user decisions as a side-effect of their routine use.

2. Machine learning methods already exist which are able to *generalize from these examples* to formulate useful advice-giving strategies that improve the competence of the system over time.

This chapter explores this thesis and provides preliminary evidence in support of these two claims, by way of experiments with the CAP system for calendar management.

Support for the first claim is provided by the fact that CAP is in routine use by one secretary in our environment, and that it is currently collecting useful training examples of this user's calendar management decisions. Although the system has only been in use by one person

7. See (Shavlik and Towell, 1989) for one approach to using prior knowledge to initialize network weights.

for one month, and therefore we have only limited experience, the initial response is favorable. Our user finds the system significantly more convenient and useful than the calendar management system she was previously using (simply editing a text file using the Emacs editor). Partly this is due to the easier interface it provides (e.g., it provides a number of convenient commands specifically tuned to editing meetings), and partly it is due to the fact that its advice is helpful. However, there are several factors that limit our ability to draw conclusions from this experience. First, the design of CAP was influenced by suggestions from this user, and was specifically designed to be upward compatible with the calendar format she had evolved while using the previous Emacs-based editor. Second, even the initial version of CAP was provided with a hand-coded set of rules, so that it was not beginning with the low level of knowledge it would have with a completely new user. Third, part of the reason this user adopted CAP is that her supervisor requested that she try it! Despite these special circumstances which make this user somewhat different from others who may be used to different systems for managing calendars, or who may work with less enthusiastic supervisors, the fact remains that CAP is of value to the one user who has tried it thus far.

Support for the second claim is provided by the experimental results with the learning mechanisms reported here. Both ID3 and Backprop succeeded in learning from previous meetings to provide advice for durations and locations of new meetings. The level of advice achieved by the learned knowledge compares favorably with that provided by the hand-coded rules developed in advance of these experiments. Based on this evidence, we feel there is an excellent chance that CAP will be able to learn new advice-giving strategies customized to other users, without the need for significant manual development or maintenance of the knowledge base. Furthermore, in the experiments reported here, the learning programs were at a disadvantage compared to the envisioned system, since the training examples that are collected by CAP contain a richer set of features (e.g., the time at which the requested meeting is added to the calendar) than the training examples we were able to recover by hand. We have not yet tested how well the learning programs will be able to handle other types of advice such as recommending meeting times and dates or topics, but we plan to conduct such experiments soon.

Based on this analysis, we believe CAP has a good chance of succeeding as a learning apprentice for calendar management. Our future plans include development of new generalization methods (e.g., combining analytic and inductive approaches), as well as work to incorporate these multiple methods into CAP. This latter topic involves issues such as allowing CAP to determine at any given time whether to base its advice on the hand-coded knowledge, the learned neural net, or the learned rules.

In addition to these plans to extend and integrate CAP's learning methods, we also plan to extend CAP to support additional calendar operations (e.g., sending electronic mail meeting reminders and requests for meeting rooms) as well as new kinds of advice (e.g., suggesting that a meeting be moved in favor of a higher priority meeting request, suggesting that reminder messages be sent). We also intend to experiment with multiple copies of CAP for multiple users. If several people in the environment were to use CAP, one could directly test the thesis that the system will be able to customize itself automatically to different users. This would also open the opportunity to study issues of multiple communicating CAP agents that exchange information directly with one another to further increase the competence of each. We believe a collection of self-customizing personal software assistants operating in a real work environment will provide a rich research testbed to study these issues.

While much of the discussion in this chapter centers around CAP in particular, our primary interest lies in understanding how to develop personalized learning apprentices more generally. The following are key lessons we have learned from the development of CAP that might provide insight into other tasks and environments where similar approaches might be considered:

- **Users find even the unintelligent interface desirable.** In order for the learning apprentice approach to work, the system must be able to collect training examples from users. Thus, users must find the system worth using even before it has had an opportunity to learn their personal preferences. Calendar management is a task for which many people already use computer text editors or interfaces, simply because of the convenience of editor interfaces and common file storage. By adding intelligent decision making to such systems one can significantly improve their value. However, for a learning apprentice approach to work, the initial system must already be above the threshold of user acceptability.

- **Perfect advice is not required.** CAP can be successful as long as it provides useful advice much of the time.

- **User decisions and some of the features which influence them are directly observable by the system.** In order to learn effectively, CAP must be able to observe sufficiently informative training examples. At present, those user decisions for which advice is learned correspond to choices of parameter values (e.g., Location) of calendar operations (e.g., AddMeeting). Such decisions are easy for the system to observe as a side effect of normal operation. Similarly, some of the features which influence the choice of these parameters (e.g., the choice of location is influenced by the number of attendees, meeting start time, etc.) are also directly observable to the system. We have added knowledge to CAP (by hand; see next point) which extends the set of accessible situation features beyond those parameter values entered directly by the user. This is primarily background knowledge about the people named in the `Attendees` parameter (their position within the organization, their office, their faculty advisor, their busyness, whether they are known to the calendar owner etc.) which allows CAP access to additional features which are helpful in characterizing regularities in the observed user decisions. While there are additional *un*observable features which may also influence the decision (e.g., the choice of Location may also be influenced by whether the desired room has been previously reserved), there is a sufficient correlation between the directly observable features, background knowledge, and the observable user decisions to produce training examples that lead to useful learned advice. Domains in which only directly observable features were required to predict user decision would be an even better match to the learning apprentice approach.

- **Some manual development is still required, but can be amortized over many users.** Despite our focus on learning, part of the knowledge base of CAP cannot be easily learned. For example, CAP currently contains knowledge of about 70 people in our environment (e.g., their offices, positions, etc.), as well as knowledge about specific work activities of the calendar owner. This must currently be entered by hand, and could constitute a significant maintenance effort in itself. However, much of this is knowledge that is not specific to a single user (it is specific to our university), and thus its development and maintenance can be amortized over all potential users of CAP within our organization. Furthermore, some of this information may already be available online in many work environments. For instance, in our environment, we hope to allow CAP to access an existing database which describes people who work in our department, including their office, position, telephone number, email address, etc.

- **The time constant of the learner is faster than the rate of change of the environment.** For CAP to be successful in adapting to changing decision strategies of its users, it must be able to learn new regularities faster than the rate at which they change. In our environment, for example, the beginning and end of each semester typically brings with it a change in calendar scheduling strategies. Therefore, CAP's learning methods must be able to collect sufficient data to learn these new regularities in a period which is short relative to a single semester. We have not yet explored this issue in detail for CAP. However, the learning experiments reported here do indicate that one semester's worth of training data is more than sufficient for learning useful regularities. Furthermore, while some regularities change at semester boundaries, many others carry over for much longer periods of time.

It is too early to say with certainty whether CAP will be a successful learning apprentice, though it has been adopted by one user, is presently collecting training data on a regular basis, and has been shown to be able to learn useful advice-giving strategies for at least meeting `Duration` and `Location`. We intend to evaluate it over the coming year by how well it learns to improve its advice for this initial user, and to adapt its advice to any changes that occur in this user's scheduling strategy. We also hope to attract new users, who will provide further data for evaluating CAP and more generally the concept of personalized learning apprentices.

Acknowledgements

CAP has been developed within the Theo architecture, and we gratefully acknowledge the help of the entire Theo group. We are especially indebted to Jeff Schlimmer, who developed the ID3 implementation used by Theo, and to Nobuo Watanabe, who developed Theo's neural net learning mechanism. Prasad Chalasani, Steve Minton, Jeff Schlimmer, and Nobuo Watanabe provided useful comments on an earlier draft of this chapter. We would also like to thank the first user of CAP for her assistance, patience and understanding.

This research was sponsored in part by the Avionics Lab, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597 and by a grant from Digital Equipment Corporation.

References

Barker, V. and O'Conner, D. (1989). Expert systems for configuration at digital: Xcon and beyond. *Communications of the ACM*, 32(3):298–318.

Cheng, J., Fayyad, U., Irani, K., and Qian, Z. (1988). Improved decision trees: A generalized version of id3. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 100–106. Morgan Kaufmann.

Fisher, D. and McKusick, K. (1989). An empirical comparison of id3 and back-propagation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.

Knight, K. (1989). A gentle introduction to subsymbolic computation: Connectionism for the a.i. researcher. Technical Report CMU-CS-89-150, Carnegie Mellon University.

Kodratoff, Y. and Tecuci, G. (1986). Disciple: An interative approach to learning apprentice systems. Technical Report UPS-293, Laboratoire de Recherche en Informatique, Universite de PARIS-SUD.

Mitchell, T., Allen, J., Chalasani, P., Cheng, J., Etzioni, O., Ringuette, M., and Schlimmer, J. (1991). Theo: A framework for selfimproving systems. In VanLehn, K., editor, *Architectures for Intelligence*. Erlbaum.

Mitchell, T., Mahadevan, S., and Steinberg, L. (1985). Leap: A learning apprentice for vlsi design. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.

Mooney, R., Shavlik, J., Towell, G., and Gove, A. (1989). An experimental comparison of symbolic and connectionist learning algorithms. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 775–780. Morgan Kaufmann.

Nakauchi, Y., Okada, T., and Anzai, Y. (1991). Groupware that learns. In *Proceedings of the IEEE Pacific Rim Communications, Computers and Signal Processing*. IEEE Press.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Quinlan, J. (1987). Generating production rules from decision trees. In *Proceedings of the International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.

Rumelhart, D., Hinton, G., and Williams., R. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., McClelland, G. L., and the PDP Research Group, editors, *Parallel Distributed Processing.*, volume 1, pages 318–362. MIT Press.

Sadeh, N. (1991). *Look-ahead techniques for micro-opportunistic job shop scheduling*. PhD thesis, Robotics Institute, Carnegie Mellon University.

Schlimmer, J. (1990). Redstone: A learning apprentice of text transformations for cataloging. Unpublished.

Schlimmer, J. (1991). Learning meta knowledge for database checking. In *Proceedings of the Ninth National Conference on Artificial Intelligence*. AAAI Press.

Segre, A. (1988). *Machine Learning of Robot Assembly Plans*. Kluwer Academic Press.

Sejnowski, T. and Rosenberg, C. (1987). Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168.

Shavlik, J. and Towell, G. (1989). An approach to combining explanation-based and neural learning algorithms. *Connection Science*, 1(3):231–253.