Active Preference Learning for Personalized Calendar Scheduling Assistance

Melinda T. Gervasio*, Michael D. Moffitt†, Martha E. Pollack†,
Joseph M. Taylor†, Tomas E. Uribe*

*AI Center, SRI International
333 Ravenswood Ave.
Menlo Park, CA 94025
1-650-859-{4411,2444}
{melinda.gervasio,tomas.uribe}@sri.com

†Computer Science & Engineering, Univ. of Michigan
1301 Beal Ave.
Ann Arbor, MI 48109
1-734-{763-1563, 615-8048}
{mmoffitt,pollackm,jmtz}@eecs.umich.edu

ABSTRACT
We present PLIANT, a learning system that supports adaptive assistance in an open calendaring system. PLIANT learns user preferences from the feedback that naturally occurs during interactive scheduling. It contributes a novel application of active learning in a domain where the choice of candidate schedules to present to the user must balance usefulness to the learning module with immediate benefit to the user. Our experimental results provide evidence of PLIANT's ability to learn user preferences under various conditions and reveal the tradeoffs made by the different active learning selection strategies.

Categories and Subject Descriptors
I.2.6 [**Artificial Intelligence**]: Learning – *parameter learning*

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *scheduling*

General Terms
Algorithms, Performance, Design, Experimentation, Human Factors

Keywords
adaptive user interfaces, machine learning, active learning, learning preferences

1. INTRODUCTION
Recent years have seen a surge in commercial calendaring systems that focus on calendar sharing and display but leave most of the decision-making responsibilities to the user. Meanwhile, the research community has focused primarily on automation within *closed* calendar systems [5], wherein a single global calendar is maintained for all users and the sole consideration for meeting scheduling is users' availability. Because users generally prefer to control their own calendars, such centralized solutions have not always been easily accepted. Nonetheless, users may not want to manage all aspects of scheduling, and might be happy to delegate responsibility to a trusted assistant who knows their preferences.

In a long-running interactive application such as a calendar assistant, learning user preferences and adapting over time has numerous advantages over requiring the user to state a static set of preferences up front. For example, the user need not specify hypothetical preferences that never occur in practice, and the user can correct and refine the existing preferences to reflect circumstances that change over time. This is greatly facilitated if the system can automatically learn these preferences, through nonintrusive interaction with the user.

To support this, we have developed a scheduling system called PTIME (*Personal Time Manager*) that provides adaptive scheduling assistance within an *open* calendaring system [5]. In PTIME, users maintain control of their own calendars, but are assisted by scheduling tools that respect their preferences. This paper presents PLIANT (*Preference Learning through Interactive Advisable Nonintrusive Training*), the learning system that lets PTIME adapt its scheduling assistance to individual user preferences. Each time a user interacts with PTIME by making a scheduling request, PTIME suggests a small set of alternative solutions. An active learning component in PLIANT is responsible for selecting these alternatives. The user's choice of a solution is then fed back to PLIANT, which uses it to automatically update the preference profile.

2. PREVIOUS WORK
Machine learning techniques have been used to improve the performance of autonomous scheduling systems (e.g., [9,20]). However, much of the prior work has been aimed at learning more efficient strategies for automated scheduling, where neither the scheduling nor the learning process involves interaction with humans. In contrast, PLIANT learns human preferences over acceptable schedules, employing direct interaction with users.

PLIANT is thus more closely related to interactive scheduling assistants such as CAP, CABINS, and INCA. The Calendar Apprentice (CAP) [13] learns decision rules for predicting the values of schedule attributes such as day, time, and location based on other attributes such as meeting type and participants. CAP facilitates the scheduling process by using its predictions to

IUI'05, January 9–12, 2005, San Diego, California, USA.

automatically fill in scheduling forms, the acceptance or correction of which provides feedback to its learner. Instead of breaking the problem into a sequence of piecewise predictions, PLIANT suggests a small set of complete schedules to the user, simplifying the user's decision to one selection from options that provide a global view. In addition, PLIANT employs active learning to improve the learning rate.

CABINS is a revision-based job-shop scheduling system that uses case-based learning to acquire user optimization preferences over repair actions as well as repair outcomes [14]. CABINS was designed for job-shop scheduling tasks in dynamic environments, which favor repair-based methods to be able to continually adapt the schedule to changing situations. In contrast, in the domain of calendar scheduling, the primary task is one of incrementally adding events to a user's existing set of commitments, a problem that requires a constructive approach such as that in PTIME.

INCA is a constructive scheduling system for developing responses to hazardous materials (hazmat) incidents [8]. It uses case-based reasoning to provide initial candidate schedules and then iteratively refines the schedules through interaction with the user. Like PLIANT, INCA learns a schedule evaluation function based on user selections from suggested candidates. But while INCA's problem domain involves the construction of a completely new schedule for every new hazmat incident, PLIANT's calendar management domain involves a continually evolving schedule as new meetings are added over time.

3. ADAPTIVE ASSISTANCE IN PTIME

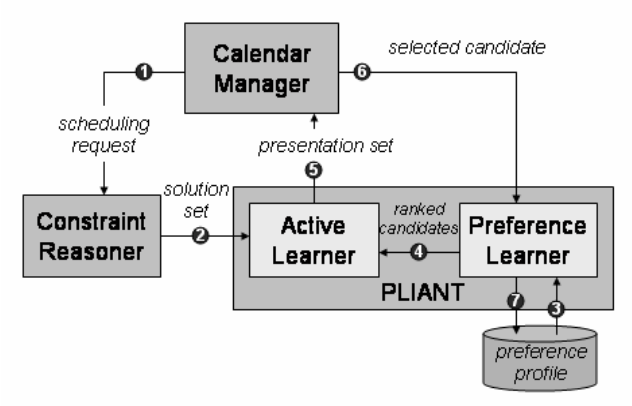

Figure 1. PTIME architecture.

PLIANT works within the PTIME scheduling assistant, whose architecture is shown in Figure 1. PLIANT is an active, online learner: it treats every scheduling request as a learning opportunity and updates the user profile after each scheduling session. The Calendar Manager is a SPARK [15] procedural reasoning module that manages the workflow involved in handling event scheduling requests. The Calendar Manager formulates a *scheduling problem* consisting of the new event constraints and the user's existing calendar. This is passed on to the Constraint Reasoner (1), which generates the *solution set* comprising all possible schedules satisfying the request. This solution set is passed on to PLIANT (2), which uses the current user preference profile (3) to rank the set of solution candidates (4), from which a subset (*presentation set*) is shown to the user (5). After the user schedules the event through the Calendar Manager, whether by selecting one of the presented candidates or scheduling the event manually, the decision is passed on to PLIANT (6), which uses the feedback to update the preference profile (7). By keeping the presentation set small and making sure it contains preferred options, PLIANT can provide more useful assistance. PLIANT thus has two main components: a *preference learner*, which induces the preference profile from the user's scheduling choices, and an *active learner*, which decides upon the scheduling alternatives to present.

3.1 Scheduling Problem

We cast the event scheduling problem as a standard constraint satisfaction problem, represented by

- a set of variables {*day, start, dur*}
- for each variable, a domain specifying its possible values
  - D_{day} = {mon, tue, wed, thu, fri}
  - D_{start} = [12:00am,11:59pm]
  - D_{dur} = (0,1440]min
- a set of constraints on one or more variables, where[1]
  - constraints on *day* are Boolean relations (e.g., (*day* = mon), (*day* ≠ tue)),
  - constraints on *start* and *dur* are linear inequalities (e.g., (*start* ≥ 10:00am), (60 ≤ *dur* ≤ 120min))
  - constraints may be arbitrary logical combinations of other constraints (e.g., ((*day* = mon) AND ((*start* ≥ 10:00am)) OR (*day* ≠ mon))

Definition. A ***meeting*** is a 3-tuple $\langle y_{day}, y_{start}, y_{dur} \rangle$ such that $y_{day} \in D_{day}$, $y_{start} \in D_{start}$, and $y_{dur} \in D_{dur}$.

Definition. A ***calendar*** is a set of meetings.

Definition. A ***scheduling problem*** (or ***meeting request***) is a pair $S = \langle C,X \rangle$, where C is a calendar and X is a set of constraints over *day*, *start*, and *dur*. Given a scheduling problem, $\langle C,X \rangle$, we can derive a set of implied constraints X_S, representing the requirement that the new meeting not overlap with any meetings already in C.

Definition. A ***solution*** to a scheduling problem $S = \langle C,X \rangle$ is a calendar $C' = C \cup M$, where M is a new meeting satisfying the constraints $X \cup X_S$.

For example, if X = {(*day* = Monday OR Tuesday) AND (*start* ≥ 10:00am) AND (*dur* = 60min)} and C = {{*day* = Monday, *start* = 1:00pm, *dur* = 60min}}, then $C' = C \cup M$ is a solution if M = ⟨Monday, 11:00am, 60min⟩ but not if M = ⟨Monday, 8:00am, 30min⟩ (violates a constraint in X) or ⟨Monday, 12:30pm, 60min⟩ (violates a constraint in X_S).

[1] In this paper, we consider only these temporal constraints and the scheduling of single meetings. In general, meetings can also include constraints over nontemporal variables such as participants and location as well as constraints involving existing meetings; requests may involve multiple new meetings.

3.2 Schedule Representation

PLIANT models the user's preferences as a schedule evaluation function—specifically, a linear cost function over a set of features that represents a candidate schedule. Given a scheduling problem $S = \langle C,X \rangle$, PLIANT evaluates a candidate solution $C' = C \cup M$ both in terms of ***local features*** of M and ***global features*** of C'. In both cases, we begin with ***basic features***, which may take on any of an arbitrary number of possible values. From each such feature, we derive an equivalent set of Boolean variables: each set represents the n possible values for an n-valued nominal feature (e.g., *day*), or n nonoverlapping ranges of values for a numeric feature (e.g., *dur*).

Table 1. Schedule evaluation features.

Local Features	**Equivalent Boolean Features**
day of week	mon, tue, wed, thu, fri
start time	earlyam (8:00-9:59am), lateam (10:00-11:59am), lunch (12:00-12:59pm), earlypm (1:00-2:59pm), latepm (3:00-4:59pm)
duration	short (<60min), medium (60-119min), long (≥ 120min)
Global Single Features	
{short (<120min), medium (120-179min), long (≥ 180min)} meeting blocks	none (0%), few (1-25%), some (26-75%), many (76-100]%)
{short, medium, long} free blocks	none, few, some, many
{earlyam, lateam, lunch, earlypm, latepm} meetings	none, few, some, many
{mon, tue, wed, thu, fri} meetings	none, few, some, many
Global Feature Combinations	
{mon, tue, wed, thu, fri} × {earlyam, lateam, lunch, earlypm, latepm} meetings	none, few, some, many
{short, medium, long} meeting blocks × {mon, tue, wed, thu, fri}	none, few, some, many
{short, medium, long} free blocks × {mon, tue, wed, thu, fri}	none, few, some, many

Because of the nature of the training data (see Section 4.1), features are informative only if their values may differ between candidate schedules for the same problem: for example, day and location but not meeting type or agenda. Furthermore, we wanted features that correspond directly to meeting properties or are easily computable from the calendar. Table 1 summarizes the 297 features currently used (3 local features, 16 single global features, and 55 global feature combinations). The local features capture the temporal properties of a single meeting.[2] Meanwhile the global features characterize the distribution of meetings through fragmentation (relative proportions of free and busy time), the distribution of meetings over the day, and the distribution of meetings over the days of the week. Finally, we designed feature combinations that we felt captured the kinds of preferences people tend to have over schedules—for example, a preference for late afternoon meetings on Mondays or long blocks of free time on Fridays.

3.3 Scheduling Assistance Problem

We can now formally define PTIME's task of providing personalized scheduling assistance.

Definition. Let $D = \{d_1,d_2,\ldots,d_n\}$ be the set of derived Boolean features representing a solution. The ***feature vector representation*** of a solution C' is the vector $V_{C'} = \langle v_1,v_2,\ldots,v_n \rangle$, where $v_i = 0$ (false) or 1 (true), representing an assignment of values to the variables D.[3]

Definition. A ***schedule evaluation function*** is linear function f_W, associated with a real-valued vector $W = \langle w_1,w_2,\ldots,w_n \rangle$, $w_i \in \Re$, where w_i is the cost or weight associated with feature d_i. (In terms of preferences, lower weight thus entails higher preference.)

Definition. The ***cost*** of a solution C' according to a function W is the dot product $W \cdot V_{C'}$. Since $V_{C'}$ is a 0/1-valued vector, we have $W \cdot V_{C'} = \sum_i \{w_i . \mid v_i \neq 0\}$.

We state PTIME's ***performance task*** as the following optimization problem: Given a scheduling problem $S = \langle C,X \rangle$ and a schedule evaluation function f_W, find a solution $C' = C \cup M$, represented by the feature vector $V_{C'}$, with minimum cost $W \cdot V_{C'}$. We now turn our discussion to the acquisition of the schedule evaluation function f_W.

4. PREFERENCE LEARNING

PTIME's task is to find the lowest-cost schedule according to a schedule evaluation function. More accurately, it must find the lowest-cost schedule according to the user's *unknown* true evaluation function. Thus, PLIANT's learning task is to find the schedule evaluation function that most closely approximates this unknown target function.

4.1 Learning from Pairwise Preferences

PTIME assists users in scheduling events by presenting a list of candidate schedules in response to a scheduling request. By selecting one of these candidates or by manually scheduling the event instead, the user naturally provides feedback regarding the suggestions in the form of pairwise preferences. That is, the user's selection of a particular schedule indicates that user's preference for that schedule over all the others presented. Learning from pairwise preferences has been successfully applied in a number of domains (e.g., [6,8,10]). While feedback in the form of pairwise preferences may not be as rich as the information that might be obtained through utility elicitation methods, it is a much less

[2] While adding other single-valued features such as location would have been straightforward, we limit the local features to these as they are the only ones that will be available initially to the deployed system.

[3] For any set of derived features D_i corresponding to a basic feature, at most one $d_i \in D_i$ has the value 1, since D_i consists of mutually exclusive values for the basic feature.

intrusive form of interaction that, we feel, supports more natural user interfaces.

Learning from pairwise preferences can be achieved in at least two ways. The first involves converting each pair into two examples (a positive example indicating the correct preference and a negative one indicating the reverse). Then any learning algorithm associated with a separating hyperplane (e.g., perceptron, SVM) can be used to learn a binary classifier as in [6,8]. Alternatively, within a support vector machine (SVM) learning framework, the pairwise preferences can be converted to constraints and added to the quadratic optimization function of a standard SVM; PLIANT employs this approach, using the same SVMlight ranker used in [10].

4.2 Learning Task

We formulate PLIANT's learning problem as in [10], characterizing the learning task as an optimization problem in the space of ranking functions. Let (V_i,V_j) represent the preference of schedule V_i over schedule V_j. We say that an evaluation function f_W is *consistent* with the pairwise preference (V_i,V_j) iff $W \cdot V_i \leq W \cdot V_j$. Conversely, f_W is *inconsistent* with (V_i,V_j) iff $W \cdot V_i > W \cdot V_j$.

Let P_k be the set of all pairwise preferences implied by the user's selection during the interaction to schedule the request S_k. Then we can state PLIANT's ***learning task*** as follows: Given the set of pairwise preferences $\cup_{(k=1..n)}P_k$, derived from the user's interactions over n scheduling sessions with PTIME, find a linear cost function f_W that produces the same pairwise preferences. That is, find a weight vector W that is consistent with the maximum number of constraints:

$$(V_{1i},V_{1j}) \in P_1 : W \cdot V_{1j} \geq W \cdot V_{1i}$$
$$\ldots$$
$$(V_{ni},V_{nj}) \in P_n : W \cdot V_{nj} \geq W \cdot V_{ni}$$

This is an NP-hard problem that can be approximated through the addition of *slack variables* [10]. Intuitively, a (nonnegative) slack variable represents the degree of satisfaction of a constraint; by incorporating them into the objective function of a minimization problem as additional terms to be minimized, we transform the objective into a tractable one of finding a solution that maximally satisfies the constraints.

Let ξ_{ijk} be the slack variable introduced for the preference pair (V_{ki},V_{kj}) of the set of pairwise preferences P_k resulting from interaction k. Using the standard SVM learning problem formulation, we can formally state PLIANT's preference learning task as follows:

Minimize: $V(W,\xi) = \frac{1}{2}\, W \cdot W + C \sum \xi_{ijk}$

Subject to:

$$(V_{1i},V_{1j}) \in P_1 : W \cdot V_{1j} \geq W \cdot V_{1i} + 1 - \xi_{1ij}$$
$$\ldots$$
$$(V_{ni},V_{nj}) \in P_n : W \cdot V_{nj} \geq W \cdot V_{ni} + 1 - \xi_{nij}$$
$$\forall i \forall j \forall k : \xi_{ijk} \geq 0$$

Here we have chosen to use an SVM formulation that limits PLIANT to learning linear preference functions. Were we to allow nonlinear kernels, PLIANT could automatically explore arbitrary feature combinations in a higher-dimensional space, but at significantly higher computational cost. More importantly, the use of more complex kernels would reduce visibility into the learned model: the resulting user model would not explicitly specify which combinations of features are particularly important to the user. So, we instead advocate designing the feature vector to explicitly include certain feature combinations (see Table 1) and learning simpler linear functions.

4.3 Initializing Preferences

While it is important for PTIME to be able to learn from unobtrusively gathered data, as an interactive assistant it must also be able to learn from direct user instruction. We have thus designed PTIME to be able to take direct advice regarding scheduling preferences. Each of the binary variables in Table 1 corresponds to an easily specifiable preference—for example, "I like meetings on Tuesdays" or "I like to have long blocks of free time". We incorporate such explicit preferences into the learned model by converting them into an equivalent set of pairwise preferences.[4] Learned preferences will take precedence over explicit preferences in that the training examples PLIANT acquires over time will eventually overwhelm these initial pseudo-examples. This is not necessarily a bad property in that human users are notoriously bad at elaborating their own utility functions. In the future, however, we may explore more sophisticated methods for combining explicit preferences with inductively learned preferences [e.g., 7] to address other situations such as users always wanting their explicitly stated preferences to take precedence, or users radically revising their preferences because of external factors such as semester breaks or organizational changes.

5. ACTIVE LEARNING

In the preceding section, we discussed how the user's selection from a set of candidate schedules can be used to generate pairwise preferences for training the preference learner. If our sole objective were to learn the user's preferences as quickly as possible, we could maximize the training examples by presenting the user with *all* possibilities or even requesting a complete ranking over them. However, such an approach would make interaction with PTIME so unwieldy that users would be likely to resort to manual scheduling. Instead, to balance the demands of maximizing learning while remaining beneficial to the user, PLIANT relies on *active learning* [2,11,19] to determine which candidates from the solution set to present to the user. There are several constraints on the presentation set:

- It must be limited in size, to avoid overwhelming the user.
- It must be responsive to a particular meeting request; PLIANT cannot simply engage the user in hypothetical scheduling problems, just to increase its knowledge.
- It must include at least some relatively good solutions; otherwise it provides no value over manual scheduling.

These features arise from the fact that PLIANT is not a stand-alone learning system but instead is tightly integrated with a larger

[4] Because our current SVM learning algorithm is not incremental, we cannot use the explicit preferences to directly set the initial profile weights. Our future plans include transitioning to an incremental algorithm.

system, and they distinguish the active learner in PLIANT from most other applications of active learning, which are typically concerned only with finding the most informative examples (e.g., [2,4,12,16]). To address these constraints, we explored a variety of both ***undirected*** and ***directed*** selection schemes [19] (Table 2).

Table 2. Active learning selection schemes.

Undirected Schemes	Directed Schemes
Random	*Max Diversity*
Random + Best	*Max Diversity + Best*
Best N	*Max Novelty*
ϵ-greedy	*Max Novelty + Best*

5.1 Undirected Methods

The simplest undirected method available is to select candidates from the solution set with a uniformly random probability (*Random*). Such a technique performs pure exploration without any exploitation. A slight modification is to ***seed*** the presentation set by having it include the meeting currently believed to be the best option (*Random + Best*). A third technique is to greedily choose the schedules that the learner currently believes to be the best (*Best N*). Such a selection policy makes complete use of exploitation, and only explores the solution space "accidentally" when a random problem forces it into an unexplored area.

A hybrid of random and greedy techniques, called the ϵ-*greedy* algorithm, chooses with probability $1 - \epsilon$ a solution believed to be optimal, and otherwise chooses a random meeting, iterating this process until the required number of candidates has been selected. This selection policy, inspired by techniques in reinforcement learning, attempts to balance exploration and exploitation.

5.2 Directed Methods

Directed methods differ from undirected methods in that they utilize either domain-specific knowledge or information about the search performed thus far. One such policy presents the user with a diverse sampling of all the potential solutions. In the context of PTIME, in which the candidate solutions are meetings represented with binary features, a natural way to measure diversity is to count the number of features where two meetings differ in value. Using this metric, one could choose a subset of meetings that maximizes the total distance over all meetings in the set:

$$\sum_{x=1}^{n}\sum_{y=1}^{n} dist\left(meeting_x, meeting_y\right)$$

Unfortunately, performing this selection requires the evaluation of every possible subset of size *n*, a task that is exponential in the size of solution set, which can easily contain well over a hundred solutions. A greedy approximation to this technique can be found by iteratively populating the presentation set with one meeting after another, each time maximizing the total distance between the new meeting and all previously chosen meetings. The first meeting included can be chosen at random (*Max Diversity*) or can be the solution currently ranked highest (*Max Diversity + Best*)).

An alternative directed approach *(Max Novelty)* presents options unlike those that the user has already seen. It keeps track of how many times each feature value has been shown to the user. When determining which solutions to include in the presentation set, it evaluates each meeting by counting the number of Boolean features that are present (i.e., are equal to one) and that have not been present in any previously presented candidate, and selects the meetings with the smallest counts. A seeded variation (*Max Novelty + Best*) includes the current best meeting in the presentation set.

6. EXPERIMENTS

The first complete implementation of PTIME will be deployed in early 2005, at which time PLIANT will be evaluated in actual use by real users. In preparation for the deployed system, we evaluated PLIANT on synthetic users within a simulated environment. There are, of course, limitations to doing testing with simulated users. For example, how well our results translate to actual performance depends on whether real users exhibit similar preference patterns. The degree to which users will be influenced to choose a candidate from the presentation set, regardless of its absolute desirability, is also an empirical question. However, testing within a simulated environment provides us with a way to perform an initial evaluation of our approach prior to the availability of a complete system. It provides us with a way to validate our design decisions regarding the user interaction while also testing hypotheses about the effects of different factors on learning performance.

6.1 Experimental Setup

We designed our experiments to verify PLIANT's ability to learn from users' interactions with PTIME. We also aimed to investigate the effects of the different active learning selection strategies on PLIANT's performance and to explore the effects of certain design decisions such as presentation set size and the reliance on implicit pairwise preferences, as opposed to complete rankings. Our resulting experimental setup is shown in Table 3.

Table 3. Basic experiment.

```
learnOnline(target,utype,exprefs,select,feedback,show) {
  // where   target = target profile
  //         utype = user type
  //         exprefs = explicit preferences
  //         select = active learning selection strategy
  //         feedback = feedback type
  //         show = #candidates shown to user
  FOR cycle = 1 to 50
    profile = initProfile(exprefs)
    problem = getProblem(target,utype,cycle)
    solutions = generateAllSolutions(problem)
    presentset = selectPresentSet(profile,solutions,show)
    evaluateLearning(profile,target,solutions)
    newtrain = trainingData(target,solutions,feedback)
    profile = updateProfile(profile,newtrain)
  END FOR
}
```

To test PLIANT's basic adaptation ability, we created 50 random linear evaluation functions representing synthetic users and devised three user types based on the number of preexisting meetings they have in a week: heavily committed (18 meetings),

moderately committed (9), and lightly committed (3).[5] We used these to test the active learning strategies presented in Table 2.

We also used these experiments to validate our design choices. To verify that incorporating explicit preferences would benefit learning, we looked at three conditions: no explicit preferences, some (specify ¼ of features indicating the strongest preferences), and many (½ of features). To determine the effects of limiting our presentation set size to 5 candidates, we compared it to performance with 10 and 25 candidates (both of which we felt would be infeasible for real use). We also suspect that users are likely to limit their decision to the presented set and so we compared this to the condition of users selecting their true best schedule overall. Finally, to measure the usefulness of learning from a single selection among a set of candidates, we compared it to learning from complete ranking.

To simulate online learning, we ran each simulated user over a sequence of 50 randomly generated scheduling problems. In generating problems, we created calendars biased toward meetings that respect the target preference function and then created new meeting requests with randomly selected constraints for day, time, and duration. The solution sets ranged from a mean of 36.89 (high of 137) in the heavy commitment case to 62.90 (high of 181) in the light commitment case. After the solution set was generated for each problem, we then simulated user feedback using the target profile and updated the learned profile accordingly.

6.2 Evaluation Metrics

We use two sets of evaluation criteria to measure how well PLIANT learns user preferences and to measure the desirability of the presented candidates.

Quality of Learned Profile: To measure how accurately the learned model represents the target preference function, we compute a modified Euclidean distance measure that directly compares the preference functions.[6] Lower values indicate closer correlation. Since we are learning a ranking function, we also use Spearman's rank correlation coefficient, a standard statistical measure for the similarity between two rankings—in our case, the rankings induced by the learned and target functions on the entire solution set.

Quality of Presentation Set: To be useful, PLIANT must present desirable schedules—that is, schedules ranked highly according to the target function. To capture the quality of the presentation set, we rank the entire solution set according to the target function and then compute the best rank and the average rank of members of the presentation set. Both range from 0 (highest rank) to the size of the solution set (lowest rank).

[5] A similar previous experiment with synthetic users [8] showed that nonlinear target functions and targets using features unknown to the learner can also be approximated using linear models at the cost of somewhat degraded learning performance.

[6] The modification involves normalizing the preference vector for the learned model, to eliminate linear scaling effects. Within our space of target preference vectors, this measure has an upper bound of 86.17.

6.3 Discussion of Experimental Results

We now turn the discussion to our experimental results, although space restrictions limit detailed presentation to only a subset of our findings. We hypothesized that directed selection schemes would result in faster learning than undirected ones since they are designed specifically to maximize exploration. However, the results generally did not bear this out. For instance, for lightly committed users, when there was a clear difference, the best strategy was often undirected (*Random* or *Greedy*) (Table 4).[7] Why did the undirected strategies outperform the directed ones? A possible explanation is that this results from the nature of learning from pairwise preferences. One may actually learn more from pairs that are very close to one another than from pairs that are quite different, because the former highlight fine-grained differences, while the latter are likely not to be very revealing once a reasonable hypothesis has been obtained. The directed learning schemes that we studied (*Max Diversity* and *Max Novelty*) are designed precisely to generate sets in which the differences are maximal.

Table 4. Euclidean distance and Spearman's correlation after fifty scheduling sessions for different strategies (light commitment, no explicit preferences, presentation set size 5).[8]

	select from presentation set				select best schedule overall			
	unseeded		seeded		unseeded		seeded	
	Euc	Sp	Euc	Sp	Euc	Sp	Euc	Sp
Rand	47.01	0.82	47.14	0.75	47.04	0.76	47.00	0.78
MaxDiv	47.07	0.80	47.16	0.75	47.02	0.77	47.00	0.78
MaxNov	47.13	0.77	47.21	0.70	47.08	0.73	47.01	0.77
Greedy	47.13	0.72	47.12	0.68	46.99	0.76	46.98	0.77

The data in Table 4 reflects another interesting and unexpected influence. When the user chooses from the presentation set, unseeded selection schemes (i.e., those that do not necessarily include the current best candidate) tend to result in better learning. But when the user selects the best schedule overall, the seeded strategies tend to outperform the others. This observation can be explained as follows. As the learned model improves, the current best solution is likely to have some features in common with the one that would actually be ranked best overall by the target function. If the user always chooses from a seeded presentation set, there is a good chance that the user will select the current best candidate, and PLIANT thus will learn very little; after all, the selected candidate was already ranked higher than all the other elements of the set. In contrast, if the set is not seeded, we are more likely to gain new information from the relative ranking of the set members. Now consider the case in which the user goes outside the presentation set, always selecting the best schedule overall. If the presentation set is seeded, we gain a great deal of fine-grained information, because the selection expresses a pairwise comparison between the actual best and the current best hypothesis. But if the presentation set is unseeded, this useful

[7] At higher levels of commitment, the results were similar but less pronounced.

[8] *Seeded* refers to the *+*Best* strategies. For simplicity, we use *Greedy Unseeded* to denote the ε-*greedy* selection strategy and *Greedy Seeded* to denote the *Best N* selection strategy. All results are averaged over the last five scheduling sessions.

comparison will not be expressed unless, by chance, the presentation set includes the current best candidate.

Not surprisingly, seeded strategies—in particular, *Best N* (*Greedy Seeded*)—result in schedules with much higher average rank being presented to the user (Figure 2). Note that the difference is less pronounced early in the learning curves. That is, before a good user model has been learned, the other strategies are more likely to present schedules of similar rank as the greedy strategies. In addition, however, they present the user with a variety of options, something that may actually be more desirable early on, before a good user model has been learned.

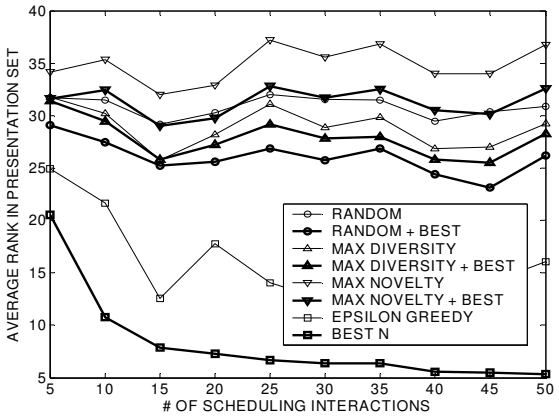

Figure 2. Average rank of presented candidates for different selection schemes (light commitment, presentation set size 5).[9]

In designing PLIANT, we decided to incorporate explicit preferences, and our experiments revealed dramatic improvement from incorporating such preferences, stressing the importance of using them when available (Table 5). We also decided to keep the presentation set small (5 candidates) both to let them be presented all at once and because we felt many users will not be willing to consider more than a handful of candidates at a time. Not surprisingly, given that a larger presentation set produces more training examples, the results did show that larger presentation sets improve performance (Table 6). However, preferences could still be effectively learned, even with small presentation sets. Our final design decision was to use as training data only the pairwise preferences implicit in a user's selection of one schedule from a set of suggestions, rather than to require a complete ranking of the presentation set from the user. The results show that while using feedback in the form of a complete ranking improves performance, it would do so greatly only for large presentation set sizes (Figure 3). This is particularly encouraging, since it shows that natural, unobtrusive user feedback can be nearly as effective as other techniques that require significantly more effort on behalf of the user.

[9] To control for the large variance in solution set sizes, which affects the rank-based metrics, we average the results every five scheduling sessions. Results for other conditions as well as for the other metrics (top rank and selection outside the presentation set) were consistent.

Table 5. Results for different explicit preferences conditions (moderate commitment, *Greedy* selection, presentation set size 5, selection from presentation set).

	Euclidean		Spearman's		Ave. Rank		Top Rank	
	start	end	start	end	start	end	start	end
none	47.38	46.86	0.35	0.70	18.38	4.82	9.41	0.31
some	45.63	45.39	0.74	0.81	7.16	3.73	1.76	0.24
many	45.04	44.85	0.80	0.85	5.92	3.58	1.09	0.18

Table 6. Euclidean distance and Spearman's correlation after fifty scheduling sessions for different feedback conditions (moderate commitment, *Best N* (*Greedy Seeded*) selection).[10]

	5		10		25	
	Euc	Sp	Euc	Sp	Euc	Sp
present	47.03	0.70	46.89	0.75	46.82	0.78
candidate	46.86	0.74	46.80	0.78	46.78	0.79
rank	46.94	0.76	46.62	0.88	46.32	0.96

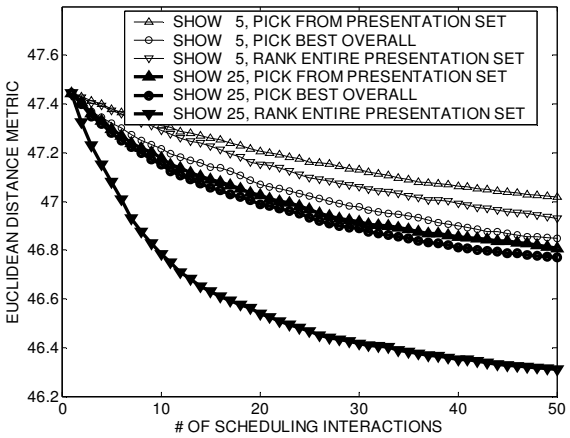

Figure 3. Euclidean distance for different feedback conditions (moderate commitment, greedy selection).

7. CONCLUSIONS AND FUTURE WORK

We have presented an open calendaring system that manages the trade-off between providing immediate benefit to its users and learning their scheduling preferences in an unobtrusive manner. Overall, our experimental results provide at least initial evidence that unobtrusive yet efficient learning of scheduling preferences is feasible. Learning occurs even when users are presented with relatively small presentation sets, from which they select a single option (as opposed to doing a complete ranking). Explicit preferences can be incorporated, and, when accurate, do further speed up learning. Finally, and interestingly, simple, undirected methods for selecting candidates to present to the user seem to be quite effective.

In early 2005, we will be evaluating PTIME in actual use by real users. The results from this evaluation will naturally help in developing succeeding versions of the system by informing a

[10] These are consistent with the results for other selection strategies and commitment levels.

number of issues, including the feature engineering effort, the design of the user interaction, and the learning approach. It should also help us devise better simulated experiments, which will remain useful as a method for carrying out initial studies of new functionalities or techniques prior to actual deployment.

PLIANT's active learning strategies currently involve a post-hoc selection over generated candidates based on evaluation according to the profile. We plan to investigate approaches that instead drive the generation of candidates directly from the learned profile (e.g., [18]), which should lead to more efficient candidate generation and, potentially, faster learning. Also, while PLIANT is an online learner, it is not an incremental one: it relearns a user model on all previous examples after every scheduling session. By converting PLIANT to an incremental learner, we expect to accommodate more direct weight initialization as well as provide better online computational properties.

The work described in this paper addresses the problem of learning preferences primarily for underconstrained situations, where multiple feasible options are available. A potentially more interesting problem is that of learning preferences in overconstrained settings, which will require relaxing the scheduling constraints or rescheduling previous commitments. We are currently pursuing this issue along various fronts, including extending the feature set to capture partial constraint satisfaction, and learning procedures and strategies for resolving schedule conflicts.

Finally, there is the issue of nonstationary preferences. The calendar domain presents the problem not just of concept drift (gradually changing preferences) but also of concept shift (abruptly changing preferences) [13]. Simple techniques such as keeping a sliding window of training data may be sufficient for dealing with the former. However, addressing the latter will require the rapid detection of the start of a shift or even the prediction of an impending change. This problem remains an area for future work but one possibility is to cast this as a dynamic exploration/exploitation tradeoff and revisit active learning within this setting.

8. ACKNOWLEDGMENTS

We thank Pauline Berry and Kenneth Nitz for fruitful collaboration and discussions on PTIME, and the anonymous reviewers for their helpful comments and suggestions. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA or the Department of Interior-National Business Center (DOI-NBC).

9. REFERENCES

1. Berry, P. M, Gervasio, M., Uribe, T. E., Myers, K., & Nitz, K. (2004). A personalized calendar assistant. *Working Notes of the AAAI Spring Symposium on Interaction between Humans and Autonomous Systems over Extended Operation*.
2. Cohn, D., Ghaharamani, Z., & Jordan, M. I. (1994). Active learning with statistical models. *Advances in Neural Information Processing Systems 7*.
3. Cristianini, N. & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines*. Cambridge University Press.
4. Engelbrecht, A. P. (2001). Selective learning for multilayer feedforward neural networks. *Fundamenta Informaticae*, 45.
5. Ephrati, E., Zlotkin, G., & Rosenschein, J.S. (1994). A non manipulable meeting scheduling system. *Proceedings of the 13th International Distributed AI Workshop*. Seattle, WA.
6. Fiechter, C.-N. & Rogers, S. (2000). Learning subjective functions with large margins. *Proceedings of the Seventeenth International Conference on Machine Learning*, 287-294.
7. Fung, G., Mangasarian, O. L., & Shavlik, J. (2002). Knowledge-based support vector machine classifiers. *Advances in Neural Information Processing Systems 15*.
8. Gervasio, M., Iba, W., & Langley, P. (1999). Learning user evaluation functions for adaptive scheduling assistance. *Proceedings of the Sixteenth International Conference on Machine Learning*, 152-161.
9. Gratch, J. & Chien, S. (1996). Adaptive problem-solving for large-scale scheduling problems: a case study. *Journal of Artificial Intelligence Research*, 4:365-396.
10. Joachims, T. (2002). Optimizing search engines using clickthrough data. *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*.
11. Kiefer, J. (1959). Optimum experimental designs. *J. R. Stat. Soc.*, series B, 21:272-304.
12. MacKay, D. J. C. (1992). Information-based objective functions for active data selection. *Neural Computation*, 4 (4): 590-604.
13. Mitchell, T. M., Caruana, R., Freitag, D., McDermott, J., & Zabowski, D. (1994). Experience with a learning personal assistant. *Communications of the ACM* 37 (7):80-91.
14. Miyashita, K. & Sycara, K. (1995). CABINS: a framework of knowledge acquisition and iterative revision for schedule improvement and reactive repair. *Artificial Intelligence*, 76.
15. Morley, D. & Myers, K. (2004). The SPARK agent framework. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi Agent Systems*.
16. Murphy, K. P. (2003). Active learning of causal Bayes net structure. *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
17. Sandip, S. & Durfee, E.H. (1998). A formal study of distributed meeting scheduling. *Group Decision and Negotiation*, 7:265-298.
18. Tong, S. & Koller, D. (2000). Support vector machine active learning with applications to text classification. *Proceedings of the 17th International Conference on Machine Learning*.
19. Thrun, S.B. (2002). The role of exploration in learning control. *Neural Networks*, 15 (4):665-687.
20. Zhang, W. & Dietterich, T. (1995). A reinforcement learning approach to job-shop scheduling. *Proceedings of the 14th International Joint Conference on Artificial Intelligence*.