Adaptive Cognitive Orthotics: Combining Reinforcement Learning and Constraint-Based Temporal Reasoning

Matthew Rudary MRUDARY@UMICH.EDU
Satinder Singh BAVEJA@UMICH.EDU
Martha Pollack POLLACKM@UMICH.EDU
Computer Science and Engineering, University of Michigan, Ann Arbor

Abstract

Reminder systems support people with impaired prospective memory and/or executive function, by providing them with reminders of their functional daily activities. We integrate temporal constraint reasoning with reinforcement learning (RL) to build an *adaptive* reminder system and in a simulated environment demonstrate that it can personalize to a user and adapt to both short- and long-term changes. In addition to advancing the application domain, our integrated algorithm contributes to research on temporal constraint reasoning by showing how RL can select an optimal policy from amongst a set of temporally consistent ones, and it contributes to the work on RL by showing how temporal constraint reasoning can be used to dramatically reduce the space of actions from which an RL agent needs to learn.

1. Introduction

Reinforcement learning (RL) has been successfully applied to a number of problems in control and operations research, but there have been relatively few applications to the design of human-computer interaction (HCI) systems; notable exceptions are Singh et al. (2002), Roy et al. (2000), and Langley (1997). In this paper, we describe the use of RL and temporal constraint reasoning to induce an effective interface for a *cognitive orthotic* system—a system intended to support people with impaired memory and/or executive function, by providing suitable reminders of functional daily activities. The goal of such systems is to increase the autonomy of cognitively impaired persons, allowing them to be more self-sufficient and/or to maintain self-sufficiency longer in the case of progressive decline. For these systems to be usable by cognitively impaired people, they must have interfaces that are extremely intuitive and straightforward, and hence the timing and content of the interactions must be carefully considered. Moreover, because people differ from one another in many regards, and because even an individual user will change over time—particularly if she has progressive cognitive decline—the interactions must be *personalized* to the needs of the user, and *adaptive* to both short- and long-term changes in those needs. For these reasons, simple interaction strategies such as always issuing a direct reminder for every activity at its earliest possible execution time are unlikely to maximize user compliance, user satisfaction, or the preservation of user autonomy.

We have therefore adopted an approach of learning effective strategies for interacting with the user of a cognitive orthotic system. Specifically, we use reinforcement learning (RL) to induce an interaction policy, i.e., a function from features of the current state (e.g., the time of day, the timing of the previous interaction, the user's mood, and the actions she is supposed to perform) to interface actions, including if and when to issue a reminder to perform a certain activity. From the perspective of RL, there is at least one rather unusual and interesting challenge in building adaptive cognitive orthotic systems. In general in RL systems, the set of actions available in every state is either fixed or quite easy to determine. In contrast, in our application, determining the set of actions available in the current state is itself an NP-hard problem. At any point in time, the system may issue reminders about any of the activities that the user might perform at that time and that would allow successful completion of the current plan. But this set of legitimate actions depends on the history of the user's activities

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

so far as well as on the details of the user's daily plan, which in general will contain a number of complex temporal constraints, including disjunctive temporal constraints; it is the extraction of the set of currently legitimate actions from the plan that is computationally hard. Although in principle we could specify that the fixed set of actions for every state is the collection of all possible reminders that the system might take at any time during the day, in practice this approach is highly inefficient. We therefore integrate two powerful technologies: constraint-based temporal reasoning, which employs powerful heuristics and pruning strategies to efficiently determine what actions are legitimate in the current state, and RL to learn from experience which of the legitimate actions is optimal there.

In a series of experiments with a simulated user and environment, we demonstrate that our approach results in a personalized and adaptive cognitive orthotic system. In addition to our contribution to the application domain, our integrated learning algorithm also contributes to research on temporal constraint reasoning by showing how RL can be used to select an optimal policy from among temporally consistent ones, and it contributes to the work on RL by showing how temporal constraint reasoning can be used to dramatically reduce the space of actions from which an RL agent needs to learn.

2. Cognitive Orthotic Systems

Cognitive-orthotic systems (also called assistive technology for cognition) are designed to help people with cognitive impairment better manage their daily activities. Generally these systems fall into two classes: activity-cueing systems, which guide their users through multi-step functional activities such as bathing or simple meal preparation, and schedule-management systems, which provide individual reminders about multiple activities, in the context of a daily plan. For a survey of the state-of-the-art in cognitive orthotics, see (LoPresti et al., 2004).

In the current project, we modify and extend an existing schedule-management system, Autominder (Pollack et al., 2003), to allow personalization and short- and long-term adaptation. Autominder has three major components: a plan manager, which models and maintains status information about the user's plan of daily activities, a client modeler, which processes information obtained from sensors to infer whether and when activities have been performed, and a reminder generation module, which reasons about discrepancies between what the user is supposed to do and what she has been observed doing, and on that basis, determines what reminders to issue.

Table 1. A realistic plan.

Activity	Start time	Duration	Notes
TakeMeds1	6:00–7:00	1–2 min	1
TakeMeds2	9:00–10:00	1–2 min	1
TakeMeds3	12:00–13:00	1–2 min	1
TakeMeds4	14:00–15:00	1–2 min	1
TakeMeds5	17:00–18:00	1–2 min	1
TakeMeds6	20:00–21:00	1–2 min	1
EatBreakfast	6:30–8:00	10–20 min	2
GotoSC	8:30–8:35	1–5 min	4
PrepareLunch (disjunctive)	6:00–7:00 11:15–12:30	5–10 min	
EatLunch	11:30–13:00	10–20 min	3
CookDinner	17:40–19:10	20–30 min	
EatDinner	18:00–19:30	15–30 min	5
CleanKitchen (disjunctive)	6:40–7:40 18:15–21:00	15–20 min	6
Bathe	19:00–21:00	10–15 min	7
Exercise	16:30–20:30	20 min	8

Notes from last column of table above
1. At least 2.5 hrs and no more than 3.5 hrs between successive TakeMeds actions.
2. At least 30 minutes between the end of TakeMeds1 and the beginning of EatBreakfast.
3. Start of EatLunch must follow end of PrepareLunch.
4. Go to the senior center. Must follow end of EatBreakfast.
5. Start of EatDinner must follow end of CookDinner.
6. If in the morning, start of CleanKitchen must follow end of EatBreakfast. Otherwise, start of CleanKitchen must follow end of EatDinner.
7. Start of Bathe must follow end of EatDinner.
8. If before dinner, end of Exercise must precede start of EatDinner by at least 10 minutes. If after dinner, start of Exercise must follow end of EatDinner by at least 20 minutes. Must end before start of Bathe.

Table 1 shows an example of a plan of daily activities and constraints among them that any reminder system has to be able to handle. (We use this plan in our simulations later in the paper). This example plan, which may be created by the user herself, or by a caregiver, helps illustrate the need for actively reasoning about reminders. Note that in this plan, a user is supposed to eat breakfast no sooner than 30 minutes after taking the first medicine of the morning. Autominder will not have a fixed *a priori* time for issuing an eat-breakfast reminder, but will instead set the time for eating breakfast based on its determination of when the user actually takes the first medicine. But should a reminder be issued for eat-breakfast at all, and if so, when? At minimum, if the user is observed to eat breakfast on her own, before a prompt is issued, Autominder should not issue a reminder at all. Beyond this, a simple scheme, such as always issuing the reminder at the earliest possible time, may not be optimal for

several reasons. First, such an approach may make the user overly reliant on the system, with the undesirable effect of decreasing, rather than increasing, her independence. Second, if as in the example plan the kitchen can be cleaned in the morning or in the evening and the user prefers to do it in the evening, the morning reminder may only serve to annoy the user and be ignored. Finally, a failure to consider interactions between reminders may lead to unacceptable sequences of reminders, for example, reminding the user to get up and go take medicine, and then, just as she's returned to sit down, reminding her to get up and go prepare lunch. Other criticisms apply to other overly simplistic schemes.

In the versions of Autominder that are currently being used in field tests, the decisions about whether and when to issue a reminder are made using an iterative refinement process, which starts with a simple initial reminder plan that in fact includes a reminder for every activity not yet done, at its earliest execution time. This initial plan is then successively rewritten, using hand-crafted rules and a local-search mechanism, until it is determined to be of sufficiently high quality when a hand-crafted evaluation rule is applied. This process is repeated whenever there is a change in the execution status of the user plan. This approach to reminder generation is limited for several reasons. First, it is difficult and costly to manually specify the rewrite rules and evaluation function, and in order to achieve the goal of personalization, they would have to be redesigned for each user. Second, even if this were done, there is no good way to validate their optimality. Third, manually specified rules are fixed, and thus not adaptive to changes in the user's needs.

3. RL-Based System Architecture

To address the limitations just discussed, we employ RL to infer an optimal interaction policy for each user of the cognitive orthotic system. Our learning architecture, depicted in Figure 1, has the same components as any standard RL-based learning architecture (e.g., Sutton & Barto, 1998) except for the additional dynamic action proposer component that is novel to this application. The environment, which consists of the user and her physical surroundings as well as the sensors are simulated in our current experiments and described in detail in Section 4. The state estimator and actuators components are original Autominder components. Autominder's Client Modeler performs state estimation; however, for our initial experiments, we assumed that the sensors provide perfect information; hence there is little actual inference being done by this module. Actuation is similarly quite simple in our experiments, and involves issuing a specific reminder, which is then input to the simulator. The most interesting component is the action proposer.

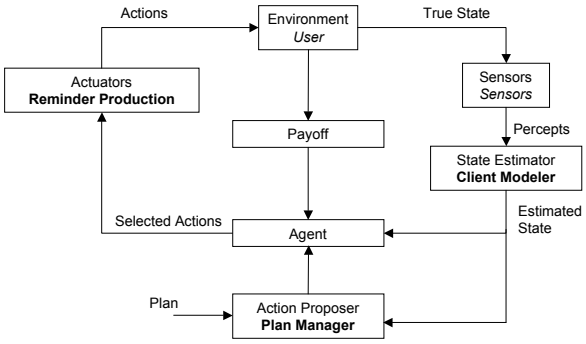

Figure 1. The architecture of the RL system for an adaptive interface to the Autominder cognitive orthotic. Generic RL components are in Roman typeface; components from Autominder are in **bold**; and components in the simulator are *italicized*.

3.1. Action Proposer: Temporal Constraint Based Reasoning

At the start of a day, the system is given the user's plan, i.e., a record of all activities the user is supposed to perform, along with constraints on the times and manner of their performance; Table 1 is an example. The action proposer has to compute which activities, if any, the user can do at each time step while still allowing the remaining activities to be done without violating any constraints (that are not already violated). This is a challenging task and we adapt Autominder's Plan Manager to this end.

The plan is modeled as a Disjunctive Temporal Problem (DTP), a constraint-satisfaction problem $< V, C >$, where V is a set of time points and C is a set of constraints of the form $[l_1 \leq x_1 - y_1 \leq u_1] \vee [l_2 \leq x_2 - y_2 \leq u_2] \vee \ldots [l_n \leq x_n - y_n \leq u_n]$, such that $x_i, y_i \in V$ and $l_i, u_i \in (\mathcal{R})$. The time points V represent the start and end of each modeled activity; additionally, there is a distinguished time point called the Temporal Reference Point (TR) that is used for encoding absolute (clock-time) constraints. Figure 2 provides some constraints from our example plan. We assume that TR is set to midnight, and that minutes are the time unit used. The start of an activity is subscripted with S, and the end with E.

Autominder's plan manager checks the consistency of a plan initially, and subsequently updates it in response to four types of triggering events: (1) the addition of a new planned activity; (2) the deletion of or modification to the constraints on an existing planned activity;

"Breakfast starts between 6:30 and 8:00." $390 \leq Breakfast_S - TR \leq 480$
"Breakfast takes between 10 and 20 minutes." $10 \leq Breakfast_E - Breakfast_S \leq 20$
"Cleaning the kitchen can be done in the morning or the evening." $400 \leq CleanKitchen_S - TR \leq 460 \vee$ $1095 \leq CleanKitchen_S - TR \leq 1260$
"Breakfast should occur at least 30 minutes after taking morning meds." $30 \leq Breakfast_S - TakeMeds1_E \leq \infty$
"Bathing and exercising cannot overlap." $0 \leq Bathe_S - Exercise_E \leq \infty \vee$ $0 \leq Exercise_S - Bathe_E \leq \infty$

Figure 2. Examples of DTP Constraints in Autominder

(3) the execution of a planned activity; (4) the passage of a time boundary in the plan. In each case, the plan manager formulates a set of disjunctive temporal constraints that represent the triggering event, and then attempts to solve the DTP defined by the union of those constraints and the constraints already in the plan. Although solving a DTP is an NP-hard problem, heuristic techniques have been developed that make it feasible to perform DTP solving for problems of the size handled by Autominder (Stergiou & Koubarakis, 1998; Armando et al., 1999; Tsamardinos & Pollack, 2003). The approach used is to convert the original problem to one of selecting a single disjunct from each constraint, such that the result is consistent. By choosing a single disjunct from each constraint, one obtains a Simple Temporal Problem (STP) (Dechter et al., 1991), a temporal constraint-satisfaction problem like a DTP, but where each constraint is restricted to a single inequality. Given a DTP D, each STP S constructed in this fashion is called a *component STP of* D. A DTP is consistent iff one of its component STPs is; hence, to solve a DTP, it suffices to search for a consistent component STP. Checking the consistency of an STP requires only polynomial time. The complexity of DTP solving comes from the fact that there are exponentially many ways of selecting individual disjuncts, and hence, potentially exponentially many component STPs to be checked. However, a number of powerful CSP pruning techniques can be brought to bear in searching the space of component STPs for consistent ones.

The action proposer component in the learning architecture has the plan as input and can observe the estimated state sequence, and uses the plan manager to perform the following three steps at each time step. First, it encodes any triggering events that have occurred in this time step with disjunctive temporal constraints. Second, it solves the DTP that consists of the union of those constraints and the previous DTP.[1] In the process, it extracts component STPs from the (possibly augmented) DTP. As we noted earlier, there may be exponentially many of these, so we limited the number we extracted to 20, which turned out to be greater than the number actually in the DTPs in our experiments. For future work, in which the size of the plans we work with may be larger, we are exploring ways of extracting component STPs that vary significantly from one another, rather than simply taking the first 20 identified. Finally, for each component STP, the action proposer extracts all events that are both *live* and *enabled*: these become the actions that are proposed to the RL agent. To compute this set, the plan manager first derives the *d-graph*, the all-pairs shortest path matrix for the STP. The d-graph directly provides the *time window* for each event e: it is the interval $[l_e, u_e]$, where l_e is the minimal distance from e to TR, while u_e is the minimal distance from TR to e. An event is live iff the current time is within its time window, and it is enabled iff all the events that must necessarily precede it, which can also be directly read off the d-graph, have already occurred. Note that the d-graph also provides deadline information for each event, which is also used in the RL process.

Once the action proposer has proposed the set of actions available (the do-nothing action is always available), they become candidates for the learning step; one will be selected and executed in the simulation environment.

3.2. Payoff Function

Computing a suitable payoff function is also rather challenging in our problem. For example, in our sample plan, the user is supposed to take morning medication between 6:00 and 7:00, and to take her second dose of medication 2.5 to 3.5 hours later, within the 9:00-10:00 time frame. But suppose the user doesn't take that first dose at all; then how much payoff should result from taking the second dose? Or suppose the user takes her first dose late, making it impossible to satisfy both constraints on the second dose? To address such complexities, we perform *batch* training in our experiments, collecting data for a whole day before going back and training our agent on that day's data. This allows us to compute the payoff at each time step of the

[1] For the experiments we conducted, the attempt to solve the DTP will fail in only one situation: if the user fails to execute an activity. In this case, the plan manager adjusts the constraints of future dependent events so that they will be evaluated as if the missed event had occurred at its latest possible time.

day by looking at the whole day's experience. We build a chronologically sorted list containing the start and end times of each activity completed and the time of each reminder issued, and then consider the elements of this list in order. Each reminder results in a payoff of -0.6; this is because we prefer not to remind unless necessary. Each activity time point (beginning or end) whose prerequisites are satisfied and all of whose related constraints are satisfied gets a payoff of 1.0. If one or more prerequisites were not met, but all the other constraints were satisfied, a payoff of 0.1 is obtained; thus the agent is rewarded a small amount if the user takes her second dose of medicine even if she didn't take the first dose. Finally if any of the hard time constraints are not satisfied (e.g., lunch was supposed to be eaten between 11:30am and 1pm and was eaten at 2pm), the agent is punished with a payoff of -1.0. The payoff component uses the plan manager to determine whether any constraints and prerequisites were violated.

3.3. Learning Algorithm

The learning agent interacts with its environment and uses the observed state-action-payoff sequence to compute a policy that maximizes the expected summed payoff over a day. We use function approximation-based Q-learning (Watkins, 1989) for our learning algorithm. The agent has a separate linear neural network for each activity, plus one for the do-nothing action. For lack of space we omit those details of the learning algorithm that are now standard in RL and can be found in textbooks (e.g., Sutton & Barto, 1998). Details specific to our application such as the state input features and the specific training methodology used are described with the empirical results.

4. Simulation Environment

We conducted a set of experiments with a simulated user and environment. As mentioned earlier, our simulated sensors were trivial, reporting perfect information about performed activities, but we endeavored to build a richer, more realistic model of potential users, focusing on two key relevant aspects of their behavior: how they perform when they don't receive a reminder, and how they react to reminders.

Recall that the daily plan is expressed as an (evolving) DTP, which may contain several consistent component STPs. We begin simulation of each day by randomly selecting one component STP, which one can view as the simulated user's initial plan for that day. As time passes, this STP may become inconsistent; for instance, a reminder may cause the user to perform activities in a different order. If this occurs, the simulator selects a new component STP that is consistent with the actual execution times of activities. This process is equivalent to a person coming up with a rough schedule for the day, but modifying it as needed. We will use the phrase *current STP* to denote the STP that is currently driving the simulated user's activities.

Given a current STP with live and enabled action A, if there is no reminder for A, its simulated performance depends on three adjustable parameters: forgetfulness (f_A), punctuality (p), and variability (v). Note that a different forgetfulness setting can be associated with each action type, whereas punctuality and regularity are global features of the simulated user. The forgetfulness factor specifies the probability that the user will forget the activity. If a particular activity A is not forgotten, then the simulator selects a time for A randomly from within its time window. The mean of the randomly selected time depends on p (smaller p means that the time selected will be closer to the beginning of the time window), while the variance of the selected time depends on v.

These three parameters allow us to specify a range of user behaviors. For instance, high p and low v represent someone who habitually does things at the last minute. On the other hand, moderate values for p and v indicate a user who is erratic in the timing of her activity execution. These can then be coupled with particular values of f_A, to model, for instance, someone who usually forgets a certain type of activity, but generally does it at the earliest possible time when she remembers at all.

This so far specifies the user's behavior in the absence of reminders. This changes, however, when reminders are given. In our experiments, the user responds immediately to a reminder by performing the specified activity, except when annoyed by having received overly frequent reminders; in that case she does not respond at all (i.e., she fails to perform the activity for which the reminder was issued). A fourth parameter of the simulator, the annoyance time factor (a) specifies the minimum period of time that must pass after one reminder is issued before the user will respond to a subsequent reminder.

Of course, reliance on a simulated user is highly imperfect; to truly validate our results we will need to replicate them with real users. However, it was necessary first to demonstrate that in principle our approach is feasible, before trying it with people.

Table 2. The simple plan used in many of the experiments.

	Activity	Start time	Duration
A	Go to the Living Room	2–6	1–3
B	Watch TV	10–18	4–6
C	Go to the Kitchen	28–33	1–5
D	Play Bingo	38–43	4–6

1. Activity A is a prerequisite for Activity B.
2. Activity C is a prerequisite for Activity D.

5. Experiments

We performed experiments using two different plans. First, we performed a suite of experiments using the simple plan shown in Table 2 to illustrate particular types of personalization and adaptation exhibited by our agent. Then we showed that our agent can handle more realistic plans by acting on the more complex plan shown in Table 1. Though we vary the forgetfulness and annoyance time parameters, in all experiments the punctuality and variability are fixed at moderate values.

5.1. The simple plan

Our experiments using the plan in Table 2 were aimed at showing that our approach can produce interaction policies that are (1) personalized to a user's particular behavior patterns, (2) capable of short-term adaptation to sudden day-to-day differences in behavior that are the result of observable factors, and (3) capable of long-term adaptation to mostly gradual behavioral changes over time, particularly those that are not associated with any observable state features.

Personalization #1 In the first experiment, we model a user who always remembers to perform activities A and B, but always forgets activities C and D. Clearly, the optimal policy here is to remind the client about activities C and D, but not about A and B. Though it is a very simple policy, it is not "cookie-cutter": it is a policy that could not be established for all users, but must be determined in response to this particular user's proclivities.

For this experiment, we collected data over 10 runs with different seeds for the random number generator. For each run, we collected 50 days of experience using a random policy; in this policy, a reminder decision is made independently for each activity. If an activity is not enabled, then no reminder is issued. Otherwise, with 50% probability, no reminder is issued for an activity; the rest of the time, a random time is chosen uniformly from the time window for that activity, and a reminder is issued at that time if the activity has not yet been performed. We then trained the agent using the first n of these days for several different values of n. The feature set used to train the nets consists of four binary features; these indicate, for each activity, whether or not the activity has been completed. To evaluate the policy learned in this experiment, it is sufficient to simulate a single day using the policy.

Figure 3A shows the average return over the 10 runs vs. the number of days of experience used in training. The value of the optimal policy is 6.8; this is the value achieved by all policies learned using at least 10 days of experience.

Personalization #2 In the second experiment activities A and B are forgotten while C and D are remembered. In addition, we added the complication of an annoyance period of 9 minutes; that is, any reminder that is given within 9 minutes of the previous reminder is ignored. Consequently, reminders for activity A and B will only be effective if the former occurs near the beginning of A's time window, while the latter occurs near the middle to end of B's time window.

The data collection and training methodology for this experiment is identical to that of the prior experiment. We added binary features that indicate for each activity whether a reminder has been issued for that activity within the last 5,10,15 minutes to the state features. The results from this experiment are shown in Figure 3B. The system converges to the optimal policy which has value 6.8 with about 20 days of data.

Personalization #3 In the third personalization experiment, we model a somewhat more realistic user, who always forgets activities C and D, but only probabilistically forgets A and B: we set f_A and f_B to 25%. Here, the optimal policy is a little more complicated. The agent should still always remind for activities C and D. But now, the agent should issue a reminder for activity A and B at the end of each one's time window, provided that the activity hasn't been executed yet. This gives the user ample opportunity to perform the activity on her own, but still maintains a "safety net" reminder if the time window is about to close without the activity being performed. In this experiment we provided state input features that corresponded to 1, 2, 3, 4, 5, 7 and 9 minutes remaining for the activity to be done.

The results of Figure 3C are averages of 4 runs and show that near-optimal performance results after about 50 days of data. Inspection of the policy learned showed that in all cases the agent learns to always remind early for activities C and D and to always remind at the very end for activities A and B. The graph in Figure 3C is less smooth than the graphs of the other

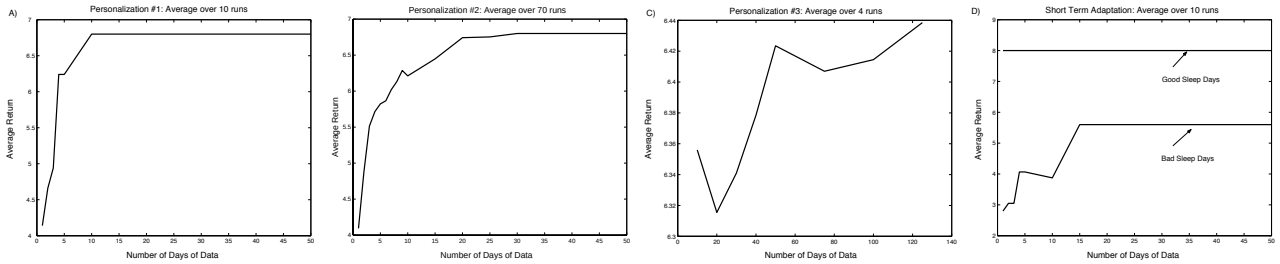

Figure 3. The results of the personalization and short-term adaptation experiments.

experiments because of the randomness introduced by the 25% forgetting of A and B.

Short-term Adaptation Here the goal was to demonstrate that the policies learned with our approach can adapt to short-term variations in the user's behavior. In this experiment, the user behaves differently depending on how well she slept the night before. If she slept well, her memory is relatively good, but if she slept poorly (something that could be determined from pressure sensors in a bed), she becomes forgetful. We conducted this experiment using extreme values: $f_A = 0\%$ for all A following a good night, and $f_A = 100\%$ for all A following a bad night.

The data collection methodology was the same here as in the previous experiments. We added a binary sleep quality feature, which is equally likely to be good or bad on any given day. The analysis of results is also slightly different: we produce different curves for behavior on good days and bad days, as shown in Figure 3D. The optimal policy on good days is never to remind; this has a value of 8.0 and is learned almost immediately (i.e., using only 1 day of data). The optimal policy on bad days is to remind for each activity; this has a value of 5.6, and is learned quickly, but not as quickly as the policy for good days; it requires approximately 20 days of training experience (however, note that these include the roughly 10 good days.)

From an RL perspective, there is little difference between the short-term adaptation experiment and the personalization experiments; in both cases the agent is learning how to map its state input features to good actions. However, from the point of view of caregivers the ability to adapt to short-term changes like sleep patterns is an important capability for a cognitive orthotic system.

Long-term Adaptation Experiment The goal here was to show that the agent can adapt to changes in the client's behavior over time, even when those changes don't correlate with observed state features. To model such change, we directly manipulate the f_A parameter for all activities A, using the function shown in the lower panel of Figure 4. This forgetfulness profile might be seen in a patient who suffers a mild stroke, represented by the jump in forgetfulness at day 50, and then enters a cognitive decline.

The data collection for this experiment differs from the previous experiments. Instead of collecting data using the random policy described earlier, we just let the agent act according to an ϵ-greedy policy (where it chooses the action that looks best currently with 90% probability and a random action with 10% probability). Every 10 days, the agent retrains using the prior 50 days of experience. The data we plot in the upper panel of Fig. 4 is the reward obtained each day.

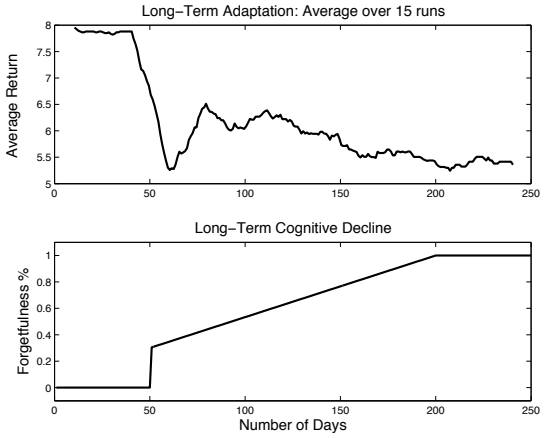

Figure 4. Results for the long-term adaptation experiment.

The results shown in Figure 4 are averaged over 15 such runs, and then smoothed by averaging together the results of 5 consecutive days. The average return is somewhat below optimal because of the epsilon-greedy strategy. However, we see that the agent adapts to the changing behavior readily. Note the dip at day 50 corresponding to the sudden forgetfulness of the client. This is followed quickly by a rise as the agent adapts to this behavior. Finally, the average return decreases somewhat as the forgetfulness increases, leveling out when the forgetfulness becomes complete.

5.2. The complex plan

This experiment used the more realistic plan presented in Table 1. We added complexity to the plan by simulating a user that forgets to take her last 3 doses of medicine as well as to do her exercise. We collect data as in the long-term experiment—that is, using an epsilon-greedy strategy and learning as time goes on—even though the client model is static. The results are shown in Figure 5 where we see that the in about 30 days the system performs near-optimally having learned in terms of the average return to the agent. Inspection of the policy learned shows that the agent always reminds early for the forgotten activities and never reminds for the remembered ones.

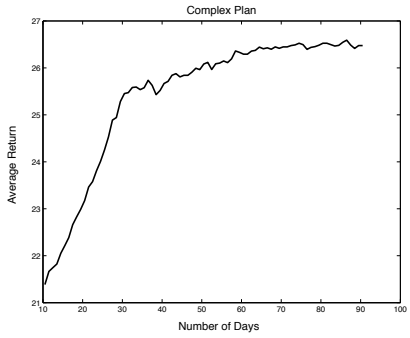

Figure 5. Results for the experiment with the complex plan.

6. Conclusion

In a series of simple experiments in a simulated environment we showed that a combination of RL and temporal constraint reasoning can produce a cognitive orthotic system that is personalized and adaptive to both short- and long-term changes in a user. This project was a feasibility study along a research trajectory in which we have deployed a non-adaptive Autominder system in field studies with real users and will move to deploying the adaptive Autominder system developed here. One issue that must be considered in this latter deployment is the length of time it takes for the reminder policy to converge to optimal; in some of our experiments, as much as 40 days of data were required. We note, however, that, by starting with a random policy in our experiments, we made the RL problem as difficult as possible. When we use the adaptive system with human users, we will instead begin with a more reasonable default policy, in which, for example, we identify certain types of activities as likely to be forgotten, and issue reminders for those more frequently. Not only do we hypothesize that convergence will be more rapid as a result, but it is also then reasonable to allow the system to interact with the user even before convergence.

In addition to deployment with human users, we have several other plans for continued work. One of the most interesting involves generalizing the interaction policy that we learn. Currently we only learn whether and when to issue reminders for individual activities, but it would also be useful to learn how much detail to include in each reminder, thereby enabling the development of an integrated schedule management/activity-cueing system.

Finally, our integrated learning architecture should extend the use of RL to a variety of planning problems in which currently temporal constraint reasoning is the method of choice.

References

Armando, A., Castellini, C., & Giunchiglia, E. (1999). Sat-based procedures for temporal reasoning. *5th European Conference on Planning.*

Dechter, R., Meiri, I., & Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, *49*, 61–95.

Langley, P. (1997). Machine learning for adaptive user interfaces. *KI - Kunstliche Intelligenz* (pp. 53–62).

LoPresti, E. F., Mihailidis, A., & Kirsch, N. (2004). Assistive technology for cognitive rehabilitation: State of the art. *Neuropsychological Rehabilitation.* In press.

Pollack, M. E., Brown, L., Colbry, D., McCarthy, C., Peintner, B., Ramakrishnan, S., & Tsamardinos, I. (2003). Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Robotics and Autonomous Systems*, *44*, 273–282.

Roy, N., Pineau, J., & Thrun, S. (2000). Spoken dialogue management for robots. *Proceedings of the 38th Annual Meeting of the Assn. for Computational Linguistics.*

Singh, S., Litman, D., Kearns, M., & Walker, M. (2002). Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *Journal of Artificial Intelligence Research*, *16*, 105–133.

Stergiou, K., & Koubarakis, M. (1998). Backtracking algorithms for disjunctions of temporal constraints. *15th National Conference on Artificial Intelligence (AAAI).*

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press.

Tsamardinos, I., & Pollack, M. E. (2003). Efficient solution techniques for disjunctive temporal problems. *Artificial Intelligence*, *151*, 43–90.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. Doctoral dissertation, King's College, Cambridge.