PROBABILISTIC PLAN RECOGNITION FOR INTELLIGENT INFORMATION AGENTS

Towards proactive software assistant agents

Jean Oh, Felipe Meneguzzi, and Katia Sycara
Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA

{*jeanoh,meneguzz,katia*}*@cs.cmu.edu*

Keywords: proactive assistant agents, probabilistic plan recognition, information agents, agent architecture

Abstract: In this paper, we present a software assistant agent that can *proactively* manage information on behalf of cognitively overloaded users. We develop an agent architecture, known here as ANTicipatory Information and Planning Agent (*ANTIPA*), for recognizing user plan in an unobtrusive manner and reasoning about time constraints to provide relevant information at a right time. *ANTIPA* integrates probabilistic plan recognition with constraint-based information gathering. This paper focuses on our probabilistic plan prediction algorithm inspired by a decision theory that human decision making is based on long-term outcomes. A proof of concept user study shows a promising result.

1 INTRODUCTION

When humans engage in complex activities that challenge their cognitive skills and divide their attention among multiple competing tasks, the quality of their task performance generally degrades. Consider, for example, an operator (or a user) at an emergency center who needs to coordinate rescue teams for two simultaneous fires within her jurisdiction. The user needs to collect the current local information regarding each fire incident in order to make adequate decisions concurrently. Due to the amount of needed information and the constraints that the decisions must be made urgently the user can be cognitive overloaded, resulting in low quality decisions. In order to assist cognitively overloaded users, research on intelligent software agents has been vigorous, as illustrated by numerous recent projects (Chalupsky et al., 2002; Freed et al., 2008; Yorke-Smith et al., 2009).

In this paper, we present an agent architecture known here as ANTicipatory Information and Planning Agent (*ANTIPA*) that can *recognize* the user's high-level goals (and the plans towards those goals) and *prefetch* various information that is relevant to the user's planning context, allowing the user to focus on problem solving. In contrast to a *reactive* approach to assistance that uses certain *cues* to trigger assistive actions, we aim to *predict* the user's future plan in order to *proactively* seek information ahead of time in anticipation of the users's need, offsetting possible delays and unreliability of distributed information.

In particular, we focus on an algorithm for predicting potential information needs following a decision-theoretic assumption that the user tries to reach more valuable world states (goals). Our algorithm is based on a decision-theoretic model known as Markov Decision Processes (MDP), and predicts a stochastic user behavior such that the better the consequence of an action is, the more likely the user takes the action. We first present the algorithm for a fully observable environment, and then generalize the algorithm to handle a partially observable case where the assistant agent may not be able to fully observe the user's current states and actions.

The main contributions of this paper are the followings. We present the *ANTIPA* agent architecture: anticipatory Information and planning agent. The *ANTIPA* architecture enables the agent to perform proactive information management by seamlessly integrating information gathering with plan recognition. We describe a probabilistic plan recognition algorithm for predicting the user's time-constrained needs for assistance. The agent continuously updates its prediction of the user plan, and adjusts its information-gathering

plan to satisfy the user's changing needs. For a proof of concept evaluation, we design and implement an abstract game that is simple yet conveys the core characteristics of information-dependent planning problem, and reports promising preliminary user study results.

2 RELATED WORK

Plan recognition refers to the task of identifying the user's high-level goals (or intentions) by observing the user's current activities (Armentano and Amandi, 2007). The majority of existing work in plan recognition relies on a *plan library* that represents a set of alternative ways to solve a domain-specific problem, and aims to find a plan in the library that best explains the observed behavior. In order to avoid the cumbersome process of constructing elaborate plan libraries of all possible plan alternatives, recent work proposed the idea of formulating plan recognition as a planning problem using classical planners (Ramírez and Geffner, 2009) or decision-theoretic planners (Baker et al., 2009). In this paper, we develop a plan recognition algorithm using a decision-theoretic planner.

A Markov Decision Process (MDP) is a rich decision-theoretic model that can concisely represent various real-life decision-making problems (Bellman, 1957). In cognitive science, MDP-based cognition models have been proposed to represent computationally how people predict the behavior of other (rational) agents (Baker et al., 2009). Based on the assumption that the observed actor tries to achieve some goals, human observers predict that the actor would act optimally towards the goals; the MDP-based models were shown to reflect such human observers' predictions. In this paper, we use an MDP model to design how a software assistant agent should recognize user behavior. In this regard, we can say that our algorithm is similar to how human assistants would predict the user's behavior.

A Partially Observable MDP (POMDP) approach was used in (Boger et al., 2005) to assist dementia patients, where the agent learns an optimal policy to take a single best assistive action in the current context. In contrast, the *ANTIPA* architecture separates plan recognition from the agent's action selection (*e.g.*, gathering or presenting information), which allows the agent to plan and execute multiple alternative information-gathering (or information-presenting) actions, while reasoning about time constraints.

3 THE *ANTIPA* ARCHITECTURE

In order to address the challenges of proactive information assistance, we have designed the *ANTIPA* architecture (Oh et al., 2010) around four major modules: *observation*, *cognition*, *assistance*, and *interaction* as illustrated in Figure 1.

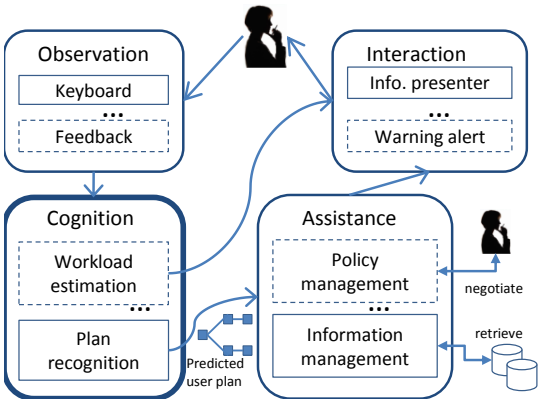

Figure 1: The *ANTIPA* agent architecture.

Observation Module receives the inputs from the user's current activities in the environment, *e.g.*, the keyboard inputs from the user's computing environment or the user feedback on the agent assistance. It collects inputs that are relevant to the user's cognitive activities and translates them into observations suitable for the cognition module. Here, the types of observations include domain-specific planning activities and more generic measures such as idle durations.

Cognition Module uses the observations (received from the observation module) to model the user behavior. For instance, the *plan recognition* submodule continuously interpret the observations to recognize the user's plans for current and future activities. When offering new information to the user, however, presenting too much information can hinder the user's task performance by overloading user cognitive abilities. In order to prevent such overloading, the *workload estimation* submodule is responsible for assessing the user's current mental workload. Here, workload can be estimated using various observable metrics, *e.g.*, a job processing time or a response time, to determine the level of assistance that the user needs.

Assistance Module is responsible for deciding the actual actions that the agent can perform to assist the user. From a predicted user plan, each assistance module identifies the user's specific needs, *e.g.*, information needs. For instance, the policy management module can verify a predicted plan according to the policies that the user must abide by. Our focus here is on information management. In order to manage information efficiently, we construct an information-gathering plan that must consider the tradeoff between obtaining the high-priority information (of which the user is likely to make the most use) and satisfying temporal deadline constraints (indicat-

ing that information must be obtained before the actual time when the user needs it).

Interaction Module decides the *time* when to offer certain information to the user based on its belief about the relevance of information to the user's current state, and the *format* of information that is aligned with the user's cognitive workload. In order to accomplish this task, the interaction module is integrated with both the cognition module (which performs mental workload assessment and determine the timing for information presentation) and the assistance module (which retrieves the information that is to be presented to the user).

Note that the focus this paper is on the plan recognition module to identify the user's current plan and predict its future steps. Thus, we shall not go into further details about the other modules, except where necessary for the understanding of plan recognition.

4 PLAN RECOGNITION

Based on the assumption that a human user intends to act rationally, we use a decision-theoretic model to represent a human user's reasoning about consequences to maximize her long-term rewards. We first assume that the agent can fully observe the user's current state and action, and knows the user's starting state. These assumptions will later be relaxed as described in Section 4.4.

4.1 MDP-based user model

We take a Markov Decision Process (MDP) to represent the user's planning process. An MDP is a state-based model of a sequential (discrete time) decision-making process for a fully observable environment with a stochastic transition model, *i.e.*, there is no uncertainty regarding the user's current state, but transitioning from one state to another is nondeterministic (Bellman, 1957). The user's objective, modeled in an MDP, is to create a plan that maximizes her long-term cumulative reward.

Formally, an MDP is represented as a tuple $\langle S, A, r, T, \gamma \rangle$, where S denotes a set of states; A, a set of actions; $r : S \times A \rightarrow \mathbb{R}$, a function specifying a reward (from an environment) of taking an action in a state; $T : S \times A \times S \rightarrow \mathbb{R}$, a state transition function; and γ, a discount factor indicating that a reward received in the future is worth less than an immediate reward. Solving an MDP generally refers to a search for a *policy* that maps each state to an optimal action with respect to a discounted long-term expected reward.

4.2 Goal recognition

The first part of our algorithm recognizes the user's ultimate goals from a set of candidate goals (or rewarding states) from an observed trajectory of user actions. We define set G of possible goal states as all states with positive rewards such that $G \subseteq S$ and $r(g) > 0, \forall g \in G$.

Algorithm 1 An algorithm for plan recognition

```
1: function PREDICT-FUTURE-STEPS(goals G, observations O)
2:     t ← Tree()
3:     n ← Node()
4:     addNodeToTree(n,t)
5:     current-state s ← lastObservation(O)
6:     for all goal g ∈ G do
7:         w_g ← Equation (1)
8:         BLD-PLAN-TREE(t, n, π_g, s, w_g, 0)
```

Initialization. The algorithm initializes the probability distribution over the set G of possible goals, denoted by $p(g)$ for each goal g in G, proportionally to the reward $r(g)$: such that $\sum_{g \in G} p(g) = 1$ and $p(g) \propto r(g)$. The algorithm then computes an optimal policy π_g for every goal g in G, considering the positive reward only from the specified goal state g and zero rewards from any other states $s \in S \wedge s \neq g$. We use a variation of the *value iteration* algorithm (Bellman, 1957) for solving an MDP.

Goal estimation. Let $O_t = s_1, a_1, s_2, a_2, ..., s_t, a_t$ denote a sequence of observed states and actions from time steps 1 through t where $s_{t'} \in S, a_{t'} \in A, \forall t' \in \{1, ..., t\}$. Here, the assistant agent needs to estimate the user's targeted goals.

After observing a sequence of user states and actions, the assistant agent updates the conditional probability $p(g|O_t)$ of that the user is pursuing goal g given the sequence of observations O_t. The conditional probability $p(g|O_t)$ can be rewritten using the Bayes rule as:

$$p(g|O_t) = \frac{p(s_1, a_1, ..., s_t, a_t|g)p(g)}{\sum_{g' \in G} p(s_1, a_1, ..., s_t, a_t|g')p(g')}. \quad (1)$$

By applying the chain rule, we can write the conditional probability of observing the sequence of states and actions given a goal as:

$$\begin{aligned} p(s_1, a_1, ..., s_t, a_t|g) &= p(s_1|g)p(a_1|s_1, g)p(s_2|s_1, a_1, g) \\ &... \quad p(s_t|s_{t-1}, a_{t-1}, ..., s_1, g). \end{aligned}$$

By the MDP problem definition, the state transition probability is independent of the goals. By the Markov assumption, the state transition probability is also independent of any past states except the current state, and the user's action selection depends only on the current state and the specific goal. By using these conditional independence relationships, we get:

$$\begin{aligned} p(s_1, a_1, ..., s_t, a_t|g) &= p(s_1)p(a_1|s_1, g)p(s_2|s_1, a_1) \\ &... \quad p(s_t|s_{t-1}, a_{t-1}), \quad (2) \end{aligned}$$

where the probability $p(a|s,g)$ represents the user's stochastic policy $\pi_g(s,a)$ for selecting action a from state s given goal g that has been computed at the initialization step.

By combining Equation 1 and 2, the conditional probability of a goal given a series of observations can be obtained. We use this conditional probability to assign weights when constructing a tree of predicted plan steps. That is, a set of likely plan steps towards a goal is weighted by the conditional probability of the user pursuing the goal.

Handling changing goals. The user may change a goal during execution, or the user may interleave plans for multiple goals at the same time. Our algorithm for handling changing goals is to discount the values of old observations as follows. The likelihood of a sequence of observations given a goal is expressed in a product form such that $p(O_t|g) = p(o_t|O_{t-1},g) \times ... \times p(o_2|O_1,g) \times p(o_1|g)$. In order to discount the mass from each observation $p(o_t|O_{t-1},g)$ separately, we first take the logarithm to transform the equation to a sum of products, and then discount each term as follows:

$$\begin{aligned} log[p(O_t|g)] &= \gamma^0 log[p(o_t|O_{t-1},g)] + \\ &... \quad +\gamma^{t-1} log[p(o_1|g)], \end{aligned}$$

where γ is a discount factor such that the most recent observation is not discounted and the older observations are discounted exponentially. Since we are only interested in relative likelihood of observing the given sequence of states and actions given a goal, such a monotonic transformation is valid (although this value no longer represents a probability).

Our approach of using decayed observations is similar to the $M2$ model in (Baker et al., 2009) in a high level, but a detailed comparison is not available because their algorithm description is abstract.

4.3 Plan prediction

The second half of the algorithm is designed to predict the most likely sequence of actions that the user will take in the future. Here, we describe an algorithm for predicting plan steps for one goal. Using the goal weights that have been computed earlier using Equation 1, the algorithm combines the predicted plan steps for all goals as shown in Algorithm 1.

Initialization. The algorithm computes an optimal stochastic policy π for the MDP problem with one specific goal state. This policy can be computed by solving the MDP to maximize the long-term expected rewards. Instead of a deterministic policy that specifies only the best action that results in the maximum reward, we compute a stochastic policy such that probability $p(a|s,g)$ of taking action a given state a when pursuing goal g is proportional to its long-term expected value $v(s,a,g)$:

$$p(a|s,g) \propto \beta \; v(s,a,g),$$

where β is a normalizing constant. The intuition for using a stochastic policy is to allow the agent to explore multiple likely plan paths in parallel, relaxing the assumption that the user always acts to maximize her expected reward.

Algorithm 2 Recursive building of a plan tree

```
function BLD-PLAN-TREE(plan-tree t, node n,
policy π, state s, weight w, deadline d)
    for all action a ∈ A do
        w' ← π(s,a)w
        if w' > threshold θ then
            n' ← Node(action a, priority w', deadline d)
            addChildNode(parent n, child n')
            s' ← sampleNextState(state s, action a)
            BLD-PLAN-TREE(t, n', π, s', w', d+1)
```

Plan-tree construction. From the most recently observed user state, the algorithm constructs the most likely future plans from the state. Thus, the resulting output is a tree-like plan segment, known here as a *plan-tree*, in which a *node* contains a predicted user-action associated with the following two features: *priority* and *deadline*. We compute the *priority* of a node from the probability representing the agent's belief that the user will select the action in the future; that is, the agent assigns higher priorities to assist those actions that are more likely to be taken by the user. On the other hand, the *deadline* indicates the predicted time step when the user will execute the action; that is, the agent must prepare assistance before the deadline by which the user will need help.

The algorithm builds a plan-tree by traversing the most likely actions (to be selected by the user) from the current user state according to the policy generated from the MDP user model. First, the algorithm creates a root node with probability 1 with no action attached. Then, according to the MDP policy, likely actions are sampled such that the algorithm assigns higher priorities to those actions that lead to a better state with respect to the user's planning objective. Note that the algorithm adds a new node for an action only if the agent's belief about the user's selecting the action is higher than some threshold θ; actions are pruned otherwise. The recursive process of predicting and constructing a plan tree from a state is described in Algorithm 2. The resulting plan-tree represents a horizon of sampled actions for which the agent can prepare appropriate assistance. When the agent determines a set of information-dependent actions to as-

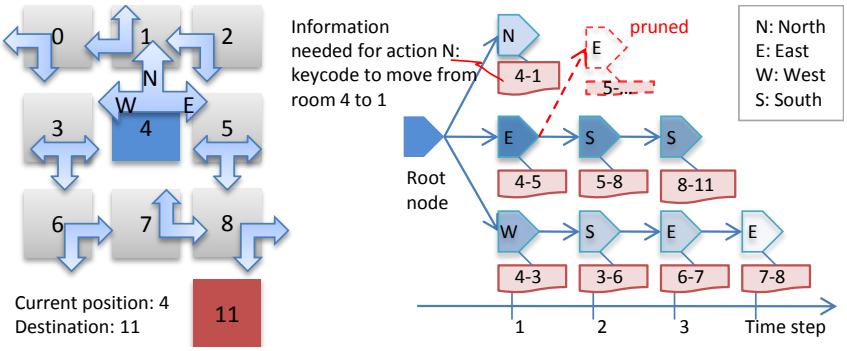

Figure 2: An example of predicted user plan.

sist from the predicted plan, the priorities are merged for redundant nodes such that the priority values in the later time steps are augmented to that of the same node in the earliest time step.

Illustrative example. Figure 2 shows an example where the user is navigating a grid to reach a destination (left). All available actions in a room are drawn as boxed arrows. A stochastic state transition is omitted here but we assume each action fails with some probability, *e.g.*, the turning to the east action may fail, resulting in the user's current position unchanged. In this problem, the agent generates a plan-tree of possible future user actions and a set of relevant information (right). A node is shaded to reflect the predicted probability of the user taking the associated action (*i.e.*, the darker, the more likely), and the time step represents the time constraint of information gathering.

4.4 Handling partial observability

Hitherto we have described algorithms based on the agent's full observability on user states. We extend our approach to handle a partially observable model for the case when the assistant agent cannot directly observe the user states and actions. Instead of observing the user's states and actions directly, the agent first infers the user's current state from indirect observations, *e.g.*, keyboard and mouse inputs from the user's computing environment or sensory inputs from various devices. The agent maintains a probability distribution over the set of user states, known as a *belief state*, that represents the agent's belief regarding the user's current state. For instance, if no prior knowledge is available the initial belief state can be a uniform distribution, indicating that the agent believes that the user can be in any state. The fully observable case can also be represented as a special case of belief state where the whole probability mass is concentrated in one state. To update a belief state, we use the *forward* algorithm (Rabiner, 1989) that estimates the probability of being in a state given a sequence of observations. We omit the details due to space limit.

5 EXPERIMENTS

As a proof of concept evaluation, we designed a game, known here as Open-Sesame, that succinctly represents an information-dependent planning problem. We note that Open-Sesame is not meant to fully represent a real-world scenario, but rather to evaluate the ability of *ANTIPA* to predict information needs in a controlled environment.

The Open-Sesame Game. The game consists of a grid-like maze where the four sides of a room in the grid can either be a wall or a door to an adjacent room; the user must enter a specific key code to open each door. Example in Figure 2 (left) shows a simplified version of the problem. The key codes are stored in a set of information sources; a catalog of information sources specifies which keys are stored in each source as well as the statistical properties of the source. The user can search for a needed keycode using a browser-like interface. Here, depending on the user's planned path to the goal, the user needs a different set of key codes. Thus, the key codes to unlock the doors represent the user's information needs. In this context, the agent aims to predict the user's future path and prefetch the keycodes that the user will need shortly.

Settings. We created three Open-Sesame games: one 6×6 and two 7×7 mazes with varying degrees of difficulty. The key codes were distributed over 7 information sources with varying source properties. The only type of observations for the agent was the room color which had been randomly selected from 7 colors (here, we purposely limited the agent's observation capability to simulate a partially observable setting). The agent was given the map of a maze, the user's starting position, and the catalog of information sources. During the experiments, each human subject was given 5 minutes of time to solve a game either *with* or *without* the agent assistance. In the experiments, total 13 games were played by 7 subjects.

Results. The results are summarized in Table 1 that

	−agent	+agent
Total time (sec)	300	262.2
Total query time (sec)	48.1	10.7
Query time ratio	0.16	0.04
# of moves	13.2	14.6
# of steps away from goal	6.3	3

Table 1: User study results for with (+) and without (−) agent assistance

compares the user performance on two conditions: with and without agent assistance. In the table, the total time measured the duration of a game; the game ended when the subject either has reached the goal or has used up the given time. The results indicate that the subjects without agent assistance (−agent in Table 1) were not able to reach a goal within the given time, whereas the subjects with the agent assistance (+agent) achieved a goal within the time limit in 6 out of 13 games.

The total query time refers to the time that a human subject has spent for information gathering, averaged over all the subjects under the same condition (*i.e.*, with or without agent assistance), and the query time ratio represents how much time a subject spent for information gathering relative to the total time. The agent assistance reduced the user's information-gathering time to less than $\frac{1}{4}$.

In this experiment, the number of moves that the user has made during the game (# of moves) can be viewed as the user's search space in an effort to find a solution. On the other hand, the length of the shortest path to the goal from the user's ending state (# of steps away from goal) can be considered as the quality of solution. The size of test subjects is too small to draw a statistical conclusion. These initial results are, however, promising since they indicate that intelligent information management generally increased the user's search space and improved the user's performance with respect to the quality of solution.

6 CONCLUSION

The main contributions of this paper are the followings. We presented an intelligent information agent, *ANTIPA*, that anticipates the user's information needs using probabilistic plan recognition and performs information gathering prioritized by the predicted user constraints. In contrast to reactive assistive agent models, *ANTIPA* is designed to provide proactive assistance by predicting the user's time-constrained information needs. The ANTIPA architecture allows the agent to reason about time constraints of its information-gathering actions; accomplishing equivalent behavior using a POMDP would take an exponentially larger state space since the state space must include the retrieval status of all information needs in the problem domain. We empirically evaluated *ANTIPA* through a proof of concept experiment in an information-intensive game setting and showed promising preliminary results that the proactive agent assistance significantly reduced the information-gathering time and enhanced the user performance during the games.

In this paper, we have not considered the case where the agent has to explore and learn about an unknown (or previously incorrectly estimated) state space. We made a specific assumption that the agent knows the complete state space from which the user may explore only some subset. In real-life scenarios, users generally work in a dynamic environment where they must constantly collect new information regarding the changes in the environment, sharing resources and information with other users. In order to address such special issues that arise in the dynamic settings, in our future work we will investigate techniques for detecting environmental changes, incorporating new information, and alerting the user of changes in the environment.

REFERENCES

Armentano, M. G. and Amandi, A. (2007). Plan recognition for interface agents. *Artif. Intell. Rev.*, 28(2):131–162.

Baker, C., Saxe, R., and Tenenbaum, J. (2009). Action understanding as inverse planning. *Cognition*, 31:329–349.

Bellman, R. (1957). A markov decision process. *Journal of Mathematical Mechanics*, 6:679–684.

Boger, J., Poupart, P., Hoey, J., Boutilier, C., Fernie, G., and Mihailidis, A. (2005). A decision-theoretic approach to task assistance for persons with dementia. In *Proc. IJCAI*, pages 1293–1299.

Chalupsky, H., Gil, Y., Knoblock, C., Lerman, K., Oh, J., Pynadath, D., Russ, T., and Tambe, M. (2002). Electric Elves: Agent technology for supporting human organizations. *AI Magazine*, 23(2):11.

Freed, M., Carbonell, J., Gordon, G., Hayes, J., Myers, B., Siewiorek, D., Smith, S., Steinfeld, A., and Tomasic, A. (2008). Radar: A personal assistant that learns to reduce email overload. In *Proc. AAAI*.

Oh, J., Meneguzzi, F., and Sycara, K. P. (2010). ANTIPA: an architecture for intelligent information assistance. In *Proc. ECAI*, pages 1055–1056. IOS Press.

Rabiner, L. (1989). A tutorial on HMM and selected applications in speech recognition. *Proc. of IEEE*, 77(2):257–286.

Ramírez, M. and Geffner, H. (2009). Plan recognition as planning. In *Proc. IJCAI*, pages 1778–1783.

Yorke-Smith, N., Saadati, S., Myers, K. L., and Morley, D. N. (2009). Like an intuitive and courteous butler: a proactive personal agent for task management. In *Proc. AAMAS*, pages 337–344.