Dealing with Ambiguity in Plan Recognition under Time Constraints

Moser Silva Fagundes, Felipe Meneguzzi, Rafael H. Bordini, Renata Vieira
Postgraduate Programme in Computer Science – School of Informatics (FACIN)
Pontifical Catholic University of Rio Grande do Sul (PUCRS) – Porto Alegre – RS, Brazil
{moser.fagundes,felipe.meneguzzi,rafael.bordini,renata.vieira}@pucrs.br

ABSTRACT

Plan recognition has been widely used in agents that need to infer which plans are being executed or which activities are being performed by others. In many applications, reasoning and acting in response to plan recognition requires time. In such systems, plan recognition is expected to be made not only with precision, but also in a timely fashion. When recognition cannot be made in time, an agent attempting to recognize plans can interact with the observed agents to disambiguate multiple hypotheses. However, such an intrusive behavior is often not possible, very costly, or undesirable. In this paper, we focus on the problem of deciding when to interact with the observed agents in order to determine their plans under execution. To tackle this problem, we develop a plan recognizer that, on one hand is the least intrusive possible, and on the other hand attempts to recognize the plans of the observed agents with precision as soon as possible and no later than it is viable to respond to the recognized plan.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*intelligent agents, multiagent systems*

General Terms

Algorithms, Design, Theory

Keywords

Plan recognition, Plan disambiguation, Multiagent systems

1. INTRODUCTION

Plan recognition has been widely employed in a variety of applications in which an agent needs to infer which plans are being executed or which activities are being performed by others. Such recognition might be aimed at either coordinating with other agents, assisting human users in their daily activities, or even identifying potentially dangerous behavior from other agents. Examples of such applications include multigent teamwork coordination [19], assistant agents for norm violation prediction [16], detecting abnormal behavior for airport security [4], intrusion detection systems, [11] and a host of proactive interface agents [1]. Common to all of these applications is that plan recognition is always found as a part of a larger reasoning cycle whereby the output of the plan recognizer is used as input to some other decision-making process. Thus, any reasoning process depending on the output of plan recognition must deal with the uncertainty and inherent latency that occurs as the agent gathers observations about the subjects of plan recognition.

In many multiagent applications, reasoning and acting in response to plan recognition may require time. For instance, consider a mechanism that monitors tasks being executed by other agents and reallocates the tasks that are about to fail. Such a mechanism may take time to successfully predict failures in plans under execution and compute reallocation solutions. In such applications, plan recognition is expected to be made not only with precision, but also in a *timely fashion*. This means that plans under execution need to be recognized with *consistency* and within specific *time constraints* (i.e. while reallocation still can avert failure), so responses to plan recognition can be effectively taken (e.g. reallocation of tasks, suggestion of alternative plans, warning messages). In practice, time and precision requirements can be hard to fulfill since the recognizer may contend with multiple hypotheses for a long time due to ambiguity in the candidate plans. In this situation, plans under execution are recognized with precision only when all hypotheses but one have been ruled out by the plan recognizer.

If the time constraint in a plan recognition attempt cannot be met, the recognizer can interact with the *target agent*[1] in order to ask for information about its plans. The problem with relying on such interactions to recognize plans is that, in systems in which agents are *autonomous*, there is no guarantee that the target agents will behave in an entirely predictable way. Human users, for example, may take a long time to reply to requests from the plan recognizer, or even intentionally ignore the requests if they are made too often [12], if they provide wrong information, or if they are made in an intrusive way [20]. Furthermore, communication infrastructure may not be available, or interaction between the recognizer and target agents may not be allowed or recommended during the plan recognition process. For example, in applications where the agents communicate in inherently or potentially hostile environments, the messages exchanged between the agents might be used by adversaries for malicious purposes [17].

[1] *Target agents* (observed agents) execute the plans that the observer agent aims to recognize. Target agents can be software agents or human users interacting with the system.

Appears in: *Alessio Lomuscio, Paul Scerri, Ana Bazzan, and Michael Huhns (eds.), Proceedings of the 13th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014), May 5-9, 2014, Paris, France.*

On that basis, we develop a plan recognizer that, on one hand is the least intrusive possible, and on the other hand aims to recognize the plans under execution with precision as soon as possible and no later than it is viable for the agent system to respond to the recognized plan. In this paper, we focus on the plan recognition problem of deciding when to interact with the target agents in order to determine their plans under execution. To deal with this problem, we extend the *Symbolic Plan Recognition* (SBR) technique proposed by Avrahami-Zilberbrand and Kaminka [2] in order to support such a decision-making process during the plan recognition. First, we develop a method to estimate the expected time to recognize a plan from multiple hypotheses given a sequence of observations. Second, we propose a method to save data about how many times a plan is selected, taking into account the context where it has been executed. Using this method, a plan recognition agent is able to reason about and dynamically adjust the certainty it ascribes to the observed plans as they are recognized.

We assume the existence of a *recognition deadline* indicating the time limit for a response to the recognized plan to be effectively taken, and a *degree of certainty* (threshold) that determines the minimum probability to consider a plan hypothesis as correct. Our plan recognition agent then checks the following two conditions to decide whether or not to interact with the target agent:

(1) Does the expected time to recognize a plan from multiple hypotheses exceed the deadline to act in response to the plan recognition?

(2) Is there no plan recognition hypothesis with degree of certainty that exceeds the given threshold?

This paper is organized as follows. In Section 2, we briefly survey related work about plan recognition and we comment on their limitations. We follow with the development of our contribution in Section 3, describing the underlying SBR approach and our extensions. In Section 4, we validate our algorithms. Finally, we conclude the paper pointing towards future developments in Section 5.

2. RELATED WORK

Algorithms to recognize the intentions and plans executed by autonomous agents (including human agents) have long been studied in the Artificial Intelligence field under the general term of *plan recognition*. Such work has yielded a number of approaches to plan recognition [6, 18] and models that use them in specific applications [19, 16, 4, 11]. The reader is referred to [1, 8] for a survey on plan recognition.

The contributions of this paper build on SBR [2], an existing symbolic plan recognition model detailed in Section 3.1. SBR has been extended in [5] to deal with constraints on plan duration. YOYO* [14] is a probabilistic plan recognition algorithm that uses information about the average plan steps duration. Duong et al. [10] provide probabilistic constraints over duration of plans using a hidden semi-Markov model. Reasoning about duration constraints is useful to improve the hypotheses elimination process, but it does not tackle our problem of estimating the expected time to plan recognition given a sequence of observations (i.e. context).

The work in [3] relies on SBR to develop a hybrid symbolic-probabilistic plan recognizer which disambiguates hypotheses using probabilistic information in the plan library. In this work, the probabilities are hard-coded into the plan library, and the context at which plan steps are executed is not taken into account to learn or assess the probabilities. Bui et al. [7] propose a framework for online probabilistic plan recognition using Markov models, but no technique to learn the model parameters is proposed. In our approach, we count plan selections and we infer the chances of the current hypotheses based on how many times they have been selected in previous episodes.

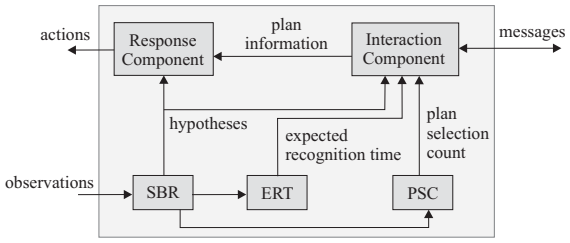


Figure 1: Plan recognizer architecture.

3. PLAN RECOGNITION AGENT

This section describes our model for a plan recognition agent, outlined in Figure 1. The agent model is made up of five components (grey boxes). SBR [2] is a symbolic plan recognizer which generates hypotheses based on a sequence of observations. Our contribution relies on the SBR (*Symbolic Plan Recognition*) technique, which is briefly presented in Section 3.1. The ERT (*Expected Recognition Time*) component, which uses the hypotheses generated by SBR to assess the expected time to recognize a plan, is described in Section 3.2. The PSC (*Plan Selection Counter*) component is used to save data about plan selections made by the target agents, taking into consideration their context of execution. PSC is detailed in Section 3.3. Finally, the *Interaction Component*, which makes the decision about whether or not to ask the target agents for further information, is detailed in Section 3.4. The *Response Component*, which acts in response to the plans that have been recognized, is not described in detail because its implementation is application-specific.

3.1 Symbolic Plan Recognition

Symbolic Plan Recognition (SBR) is a method for complete and symbolic plan recognition that uses a *Feature Decision Tree* (FDT) to efficiently match multi-featured observations to plan steps in a plan library. In what follows, we briefly describe this method (for details, see [2]).

The plan library is hierarchically represented by a single-root directed acyclic connected graph where the children of the root node are *top-level plans* and all other nodes are simply *plan steps*. In the library, plan steps can have sequential edges specifying a totally-ordered plan execution sequence. A plan step can be decomposed into sub-steps by vertical edges. No hierarchical cycles are allowed in the library. The only kind of cycle permitted is the self-cycle which represents a plan step being executed during multiple subsequent time stamps. Figure 2 shows part of a plan library provided as an example in [2], inspired by the behavior hierarchies of Robocup soccer teams [15]. In this library, solid lines represent decomposition links and dashed lines represent sequential links. For instance, there is a decomposition link between *defend* and *position*, and a sequential link between

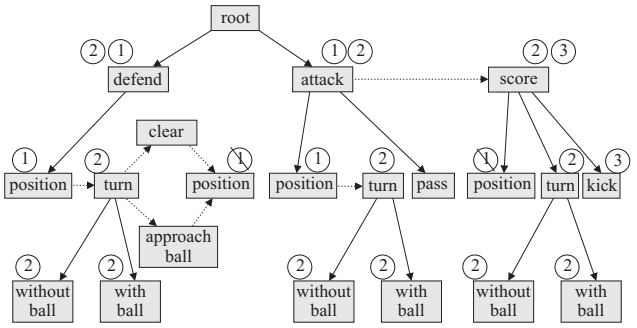


Figure 2: Plan library example [2].

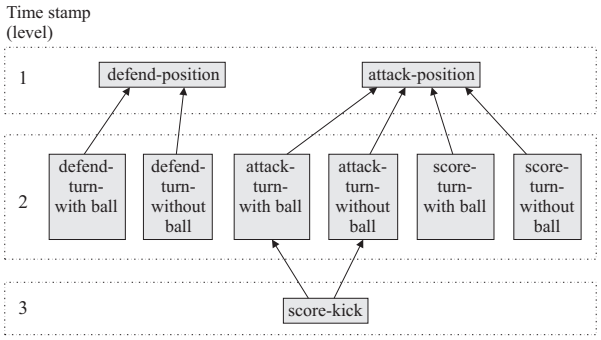


Figure 3: Hypotheses graph example [2].

this *position* and *turn*; in this example, the path *root* → *defend* → *turn* → *with-ball* can be a hypothesis as to the plan being executed by the observed player. In the plan library, numbers denote time stamps (e.g. *position* has been considered a hypothesis at time stamp 1).

A step common to several plan recognition approaches is the matching phase where the recognizer matches observations to plans stored in the plan library. SBR assumes that there is a set of conditions on observable features associated with each plan step. When these conditions hold, the observation is said to match the plan. In order to efficiently match observations to plans, SBR augments the plan library with a decision tree (the FDT) that indexes each node in the plan library by the observations consistent with their execution. As a decision tree, nodes in the FDT represent observable features and the branches represent conditions on their values. Thus, from a set of observable features, finding all matching plans is just a matter of traversing the FDT top-down until a leaf node is reached (the leaf nodes are pointers to plan steps in the plan library).

To determine what are the possible paths in the plan library that the target is pursuing, SBR admits all recognition hypotheses that are consistent with the observation history, with no hypothesis ranking. The algorithm that answers the *current state hypothesis* is divided into two phases. First, the plan steps that match the observations are found by crossing the FDT top-down in the *matching phase*. Each matched plan step is tagged with the observation time stamp that is propagated up to ensure that complete paths (root-to-leaf) are also tagged. During this *propagation phase*, the tag consistency is checked. A tag t is considered consistent only if one of the following three cases hold: (a) the tag constitutes a self-cycle (the same plan step was tagged in the time stamp t–1); (b) the tag follows a sequential edge that was tagged at time stamp t–1; or (c) the tag is attached to a first child plan step (leftmost) in the library and, therefore, there is no sequential edge to be followed back in time t–1. If the path is not consistent, all tags in the path are deleted.

The time stamps can also be used to answer queries about sequences of plan steps that have been observed. Such queries are called *state-history queries* and they are answered by means of a *hypotheses graph*, an incrementally-maintained structure which holds hypotheses in levels according to time stamps. A node in the graph represents a complete tagged path and an edge connects a hypothesis tagged at a time stamp t with a consistent hypothesis tagged at t+1. Each graph level represents a time stamp and the nodes inside a level are all hypotheses for that time stamp. To determine consistent sequences of plan paths, it is necessary to traverse the graph bottom-up. Paths that connect the last level to the first level denote valid *state histories*. In Figure 3, we show an example of hypotheses graph, also from [2], in which the sequence of plan steps *attack* → *position* (at t=1), *attack* → *turn-with ball* (at t=2), and *score* → *kick* (at t=3) is a valid state history.

3.2 Assessing the Expected Time to Recognize Plans

Although SBR may be used to recognize plans, it cannot estimate the expected time to recognize these plans. That is, during the plan recognition process, the agent can entertain several hypotheses due to the ambiguity inherent to the plan library, but it cannot estimate the expected amount of time needed to confirm one of these hypotheses. In what follows, we describe the ERT (*Expected Recognition Time*) method, our proposal to make this assessment on the basis of previous plan recognition episodes.

To develop ERT, we have extended SBR, though we have made an additional assumption on the way observations are made. We assume that observations in SBR are made in fixed intervals of time (e.g. a new observation is made every n milliseconds), which allows us to assess time amounts using the time stamps in the plan step tags. Previous work, such as [5, 14], has already used the notion of step duration, but not for estimating expected disambiguation time.

In SBR, each plan representation has an associated set of conditions on observable features of the target agent and its actions. In one possible implementation (which we use as an example for this paper) we express conditions and observations by means of propositions. For instance, in Figure 4 we defined the following conditions for the plan step *position*:

$\{location(X, Y) \mid (0 < X \leq 20) \wedge (0 < Y \leq 10)\}$,
$\{uniform\_number(X) \mid (1 \leq X \leq 3)\}$,
$\neg have\_ball$.

The following observation satisfies this condition:

$location(2, 1), uniform\_number(3), \neg have\_ball$.

In order to provide a *context*[2] to the plan steps matching an observation, we tag the concrete actions (leaf nodes in the plan library) not only with time-stamps as in the original SBR, but also with the observation that led to it being

[2]A context is given by propositions and determines the circumstances in which plan steps have been observed.

position

time stamps	observations
34	(*location*(2,3),*uniform_number* (2),¬*have_ball*)
35	(*location*(2,3),*uniform_number* (2),¬*have_ball*)
36	(*location*(2,3),*uniform_number* (2),¬*have_ball*)
37	(*location*(2,4),*uniform_number* (2),¬*have_ball*)
38	(*location*(2,4),*uniform_number* (2),¬*have_ball*)
37	(*location*(3,4),*uniform_number* (2),¬*have_ball*)
38	(*location*(3,4),*uniform_number* (2),¬*have_ball*)

Figure 4: Example of CE table, showing the columns time stamps and observations.

tagged. In this way, we keep record that this plan step (concrete action) *may* have been executed in a given context at a given moment. In Figure 4, we illustrate a plan step *position* with seven time stamps and the observation made at each of these time stamps. The time stamps and their respective observation are kept in a temporary table discarded when the current plan recognition episode ends. This table is hereafter referred to as CE (*Current Episode*) table. Note that CE tables exist only in leaf nodes in the plan library; non-leaf nodes are tagged only with time stamps as in the original SBR.

Figure 4 shows only two columns of the table, *"time stamps"* and *"observations"*, which are relevant at this point; the other columns are shown later in this section. In this example, for the sake of clarity, we specify complete observations. In practice, propositions with no variable parameters, such as *have_ball*, are not saved in the table. We could implicitly assume that, for each entry in the CE table in Figure 4, ¬*have_ball* has been observed since this proposition is in the condition for *position*. This assumption cannot be made with *location* and $uniform\_number$, given that these propositions have variable parameters and the value of these variables is part of the *context*.

The SBR algorithm is executed as specified in [2] until we eliminate all hypotheses but one (the remaining hypothesis is assumed to be the plan under execution). At this moment (in recognition time stamp t), the recognizer triggers Algorithm 1, which updates the expected times to recognize the plans based on the concluded episode. The algorithm has two inputs: the plan library l and the recognition time stamp t. For each leaf plan step in l, the algorithm checks if there is a CE table associated to the step. If there is a table, the algorithm creates the set $Observ$ that stores only the distinct observations made in the plan step. For instance, in the CE table in Figure 4, there are seven entries, but only three distinct observations, which are:

$(location(2,3), uniform\_number(2), \neg have\_ball)$,
$(location(2,4), uniform\_number(2), \neg have\_ball)$,
$(location(3,4), uniform\_number(2), \neg have\_ball)$.

For each *entry* of the table (Algorithm 1, Lines 4–6) in which t' is the time stamp at which the observation *obs* has been made, the algorithm calculates the number of time steps taken to recognize the plan from this step and inserts this value into the entry (that is, it took $t - t'$ time steps to eliminate all hypotheses but one from the moment t' when the observation *obs* was made). In practice, we *append* in the CE table a cell with the value $t - t'$ (Line 5). In Figure 5, we show the *same* table as Figure 4 but with two extra columns: the time needed to recognize the plan (*"$t - t'$"*) and the average time to recognize (*"avg"*), as computed by Algorithm 1. In this table, the plan has been recognized at $t = 50$. For example, in the first entry, the observation is $(location(2,3),\ldots)$ and t'=*34*, so it took 16 time steps to recognize the plan. To update $Observ$, we use the *union* operator, so no duplicate observation is added to this set.

For all observations in $Observ$ (Lines 7–9), the algorithm calculates the average number of time steps necessary to recognize the plan under execution. This value means that, on average, it takes *avg* steps to recognize the plan when the given plan step is an hypothesis in the context given in *obs*. In the fourth column of the table in Figure 5 (*"avg"*), we can see the averages 15.0, 12.5, and 10.5 for the three distinct observations, respectively. These values have been inserted into the table by the function *average* (Line 8). For example, in the current plan recognition episode, when the observation $(location(2,3),\ldots)$ was made, it took on average 15 time steps to recognize the plan of the target agent. Finally, the algorithm calls ERT-UPDATE (Algorithm 2), which employs the values in the CE table to update the expected times learned from past plan recognition episodes.

Algorithm 1
ERT(Plan Library l, Recognition Time Stamp t)

```
1. for all step ∈ leafPlanSteps(l) do
2.     if CE(step) ≠ null then
3.         Observ ← ∅
4.         for all entry = ⟨t', obs⟩ ∈ CE(step) do
5.             append(entry, t − t')
6.             Observ ← Observ ∪ {obs}
7.         for all obs ∈ Observ do
8.             avg ← average(obs,CE(step))
9.             ERT-UPDATE(obs, avg, step)
```

Up to this moment, we have shown how to calculate the average time taken to recognize a plan based on observations made during a *single recognition episode*. In what follows, we show how to calculate/update the expected times to recognize plans based on past episodes. We save these expected

position

t=50 (*recognition time stamp*)

time stamps (*t'*)	observations (*obs*)	*t - t'*	*avg*
34	(*location*(2,3), ...)	16	15.0
35	(*location*(2,3), ...)	15	
36	(*location*(2,3), ...)	14	
37	(*location*(2,4), ...)	13	12.5
38	(*location*(2,4), ...)	12	
39	(*location*(3,4), ...)	11	10.5
40	(*location*(3,4), ...)	10	

Figure 5: Example of CE table with time stamps (*t'*), observations (*obs*), time to recognize the plan (*t-t'*) and average time (*avg*), with plan recognition time stamp *t=50*.

times in tables called ERT tables, which, unlike CE tables, are not discarded once the executions of the SBR and ERT algorithms finish. That is, ERT tables are stored throughout the lifetime of the plan recognizer agent. There is an ERT table for each plan step in the library. The entries in these tables are composed of a unique *"observation"*, the expected time to recognize the plan (*"ert"*) from the plan step given the observation, and the number of times that the *"ert"* value has been updated (*"nupd"*). In Figure 6, we show: (a) the same CE table with the plan step *position* in Figure 5, except that we show only the second and fourth columns; (b) the ERT table of *position* before executing the ERT-UPDATE algorithm; and (c) the resulting ERT table, highlighting the entries that have been added or updated on the basis of the three observations in the CE table.

Algorithm 2 describes how an ERT table is updated. This algorithm has three inputs: an observation (*obs*), the average time to recognize the plan (*avg*), and the plan step (*step*). Note that the first two input parameters correspond to the second and fourth attributes of a line in the CE table associated to the given plan step, respectively (see Figure 5). First, Algorithm 2 checks if *obs* has already been inserted into the ERT table. If the observation is not in the table, the algorithm adds a new entry (Lines 2–3) containing: (i) the observation *obs*; (ii) the average time *avg* (in this case, the expected time to recognize the plan is the only time known, estimated during the current plan recognition episode); and (iii) the integer 1 which initializes the update counter *"nupd"*. If the observation already exists in the table, its corresponding entry e is retrieved (Line 5), and the value in the field *"nupd"* of the ERT table is increased by one (Line 6). In the notation used in the algorithm, e[*"nupd"*] denotes the column *"nupd"* in the entry e in the ERT table of *step* (Figure 6(b) illustrates an ERT table). Finally, the expected recognition time in the column *"ert"* of the ERT table is updated in Line 7, where function α weights new and previous expected recognition times, and *avg* is the average time to plan recognition in the current episode (input).

Algorithm 2
ERT-UPDATE
(Observation *obs*, Average Time *avg*, Plan Step *step*)

1. **if** *obs* $\notin$ ERT(*step*) **then**
2. $\quad e \leftarrow \langle obs, avg, 1\rangle$
3. $\quad add(e, \text{ERT}(step))$
4. **else**
5. $\quad e \leftarrow entry(obs, \text{ERT}(step))$
6. $\quad e[\text{"nupd"}] \leftarrow e[\text{"nupd"}] + 1$
7. $\quad e[\text{"ert"}] \leftarrow (1-\alpha(e[\text{"nupd"}]))e[\text{"ert"}]+\alpha(e[\text{"nupd"}])avg$

In order to avoid fluctuations in the value of the expected recognition time due to variations in user behavior, the update of the expected time to recognition (Line 7) leverages the notion of *learning rate* from reinforcement learning in the form of an α multiplier. Introducing α prevents *ert* from being affected by unusual behavior by multiplying the update value by a number in the $(0, 1]$ range. In our approach, the learning rate is a function of the number of times a certain observation has been made by the agent during recognition, and we want the estimation of expected time to recognition to gradually stabilize as the agent has gathered enough data.

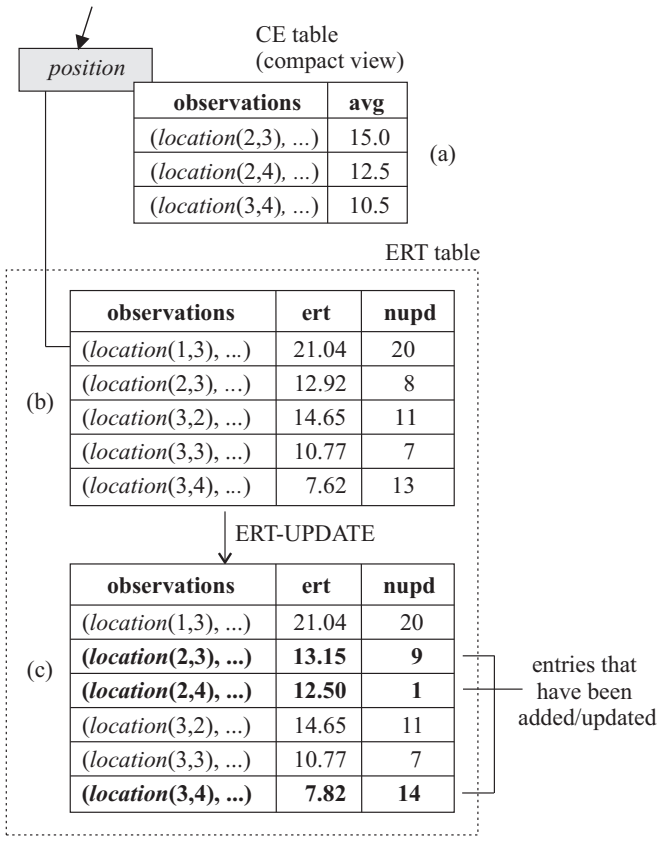


Figure 6: Plan step *position* with (a) the CE table (compact view), (b) ERT table before executing the ERT-UPDATE algorithm, and (c) after executing the algorithm.

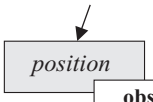
Hence,the function α used in Algorithm 2 takes the following form:

$$\alpha(n) = \frac{1}{n}$$

In Figure 6(b), we show an example of ERT table with five observations and their respective *"ert"* and *"nupd" before* calling ERT-UPDATE. There are three distinct observations within the CE table of the plan step *position*, so three entries of the ERT table will be added or updated. In Figure 6(c), we show the ERT table *after* the execution of the algorithm, with one new (the third) and two updated (the second and the last) entries.

3.3 Plan Selection Count

While other plan recognizer approaches, such as [9, 13, 5], assume that probabilities for plan selection are provided at design-time, we aim to determine these probabilities based on past successful recognition episodes. In Section 3.2, we have shown that SBR can be extended to allow the assessment of the time needed to recognize a plan, and this section shows how SBR can be extended to allow us to count how many times each plan has been selected by the target agent, taking into account the context where it has been executed (e.g. how many times *position* has been selected by the target agent when $location(2,3), uniform\_number(2)$ and $\neg have\_ball$ have been observed).



augmented ERT table

observations	ert	nupd	nps
(*location*(1,3), ...)	21.04	20	17
(*location*(2,3), ...)	13.15	9	8
(*location*(2,4), ...)	12.50	1	0
(*location*(3,2), ...)	14.65	11	10
(*location*(3,3), ...)	10.77	7	5
(*location*(3,4), ...)	7.82	14	1

Figure 7: ERT table with plan selection count (*nps*).

PSC (*Plan Selection Count*) is a simple method that utilizes an augmented ERT table to keep track of how many times the given plan step has been taken when certain observations were made. To this end, we have included a new column, named *"nps"* (see Figure 7). Note that *"nupd"* and *"nps"* count events of different types: the first indicates how many times the respective *"ert"* has been updated, and is computed by Algorithm 1, while the second indicates how many times the plan has been *in fact* executed in a given context, and is computed by Algorithm 3.

In our plan recognizer model, Algorithm 3 is used to update the plan selection count every time a plan execution ends. First, the algorithm initializes three variables (Lines 1–3): the empty set $VisitedSteps$, used to store the plan steps of the plan library l visited during the algorithm execution; *level*, initialized with the last level of the hypotheses graph g; and the set *Nodes*, initialized with all nodes in the last *level* of g. This algorithm iterates over the levels in the graph g (Lines 4–20), starting in the last level at which the plan execution ended, traversing to the *parent nodes* in the level t–1, and so on until the nodes in the first level have been visited. Note that the algorithm travels from child to parent nodes, so some of the nodes in the level may not be visited since they correspond to hypotheses that have been dismissed by negative evidence along the plan recognition process. For example, in the hypotheses graph shown in Figure 3, the last level is 3, and there is only one node in this level, $score \rightarrow kick$. Traversing the graph to the parent nodes would lead us to $attack \rightarrow turn \rightarrow withoutball$ and $attack \rightarrow turn \rightarrow withball$, which in the next iteration would lead us to $attack \rightarrow position$. Only these four graph nodes would be visited by Algorithm 3 since they constitute the confirmed hypothesis.

For each *node* in the current *level*, the algorithm gets the plan decomposition path[3] represented within the node. Then, the algorithm gets the leaf plan step (recall that only leaf plan steps save CE tables) in the decomposition path, and verifies if this plan step has not been visited. In that case, the algorithm adds the plan step to the set of visited steps and creates the set *Observ* that will store the distinct observations made in the plan step. These distinct observations are stored in the CE table which, at this point, has not been discarded yet. To select only distinct observations, the algorithm iterates over the CE table and checks if the observation o within the current entry ($e1$ is an entry of the CE table) is not in the set *Observ* (Lines 11–14). If o is not in the set, the algorithm adds this observation to *Observ*, and increases by one the value of *nps* in the corresponding entry ($e2$) in the ERT table (Lines 15–16). In summary, we travel along the confirmed hypothesis in the hypotheses graph, updating the values that indicate how many times the selected plan has been executed for each observed context.

[3]Recall that a node in a hypotheses graph stores plan steps as a plan decomposition path, from root to leaf. For example, in the node of level 3 in Figure 3, the decomposition path is $attack \rightarrow kick$.

Algorithm 3
PSC(Hypotheses Graph g, Plan Library l)

```
1.  VisitedSteps ← ∅
2.  level ← max level (g)
3.  Nodes ← nodes(g, level)
4.  repeat
5.      for all node ∈ Nodes do
6.          path ← decompostionPath(node, g)
7.          step ← leafPlanStep(path)
8.          if step ∉ VisitedSteps then
9.              VisitedSteps ← VisitedSteps ∪ {step}
10.             Observ ← ∅
11.             for all e1 ∈ CE (step, l) do
12.                 o ← e1["obs"]
13.                 if o ∉ Observ then
14.                     Observ ← Observ ∪ {o}
15.                     e2 ← entry(o, ERT(step))
16.                     e2["nps"] ← e2["nps"] + 1
17.     if level > 1 then
18.         Nodes ← parents(g, Nodes)
19.     level ← level − 1
20. until level ≤ 0
```

3.4 Interaction Component

The interaction component is employed to decide whether or not to interact with the target agent in order to determine which plan is being executed by that agent. Such a decision is supported by the information computed by ERT and PSC.

Unlike the ERT algorithm which is called only when a plan recognition episode ends, the interaction component is called every time new observations arrive. This way, the recognizer is able to decide whether to interact or not with the target agent at any time step. First, this component calculates $\overline{ert}(t)$, the *average of the expected recognition times* (saved in the ERT tables) of all valid hypotheses at the current time stamp t, which are indicated in the last level of the hypotheses graph. To calculate this average, we use the last observation as a key to the ERT table of each plan hypothesis. For example, if we have observed

$$location(1,3), uniform\_number(2), \neg have\_ball$$

at time stamp t and $defend \rightarrow position$ is a valid hypothesis at t, we get the *ert* for this observation, which would be 21.04 according to the ERT table in Figure 7. The average $\overline{ert}(t)$ is calculated with *ert* of each valid hypothesis.

On this basis, we expect to recognize the plan of the target agent at time $t+\overline{ert}(t)$. We assume that there is a *recognition deadline*, denoted as $\rho(t)$, which gives the period (in number of time steps from t) in which a response to plan recognition is effective. The specification of the deadline function ρ is application-specific and lies out of the scope of this paper.

To support the decision making process, this component calculates the chance of each hypothesis by employing the

selection count (*"nps"* attribute in the ERT table) of the current valid hypotheses in the hypotheses graph. For example, consider two hypotheses: one has been confirmed 20 times, while the other has been confirmed only 5 times in past recognition episodes. Based on these counts, we assess that the first hypothesis has chance 0.8, while the second 0.2. The function *maxChance*(t) returns the maximum chance between all current hypotheses. In our example, this function would return 0.8. We assume that there is a threshold φ which indicates the minimum chance to consider a hypothesis as correct (although it may turn out to be incorrect).

In what follows, we analyze the four possible situations in which a plan recognition agent can find itself when deciding whether or not to interact with the target agent:

(1) $\overline{ert}(t) \leq \rho(t)$
The average expected recognition time $\overline{ert}(t)$ is smaller than or equal to the recognition deadline $\rho(t)$. In this case, the recognition deadline is expected to be met. Still, there is no guarantee that this will happen since the average time is assessed on past experiences that may not repeat themselves exactly before.

(1.1) $maxChance(t) \geq \varphi$
There is a hypothesis with chance greater than or equal to the threshold φ. In this case, an interaction with the target agent is not recommended since recognition is likely to happen with success. If the recognition does not happen by the deadline $\rho(t)$, the recognizer can assume that the most likely hypothesis is the correct one.

(1.2) $maxChance(t) < \varphi$
There is no hypothesis with chance greater than φ. In this case, the decision is supported only by the expected recognition time. If the plan recognition agent decides not to interact with the target agent and the recognition does not happen by the deadline, there will be no information to support the decision about which response to trigger. Recall that responses are normally highly dependent on the plan under execution.

(2) $\overline{ert}(t) > \rho(t)$
It is not expected that the deadline will be met, and in this case, it is not recommended to rely on the plan recognition algorithm to determine the plan under execution in time.

(2.1) $maxChance(t) \geq \varphi$
There is a hypothesis with chance greater than or equal to the threshold φ. In this case, the agent can trigger the response based on the chances, but it will be unlikely to confirm the hypothesis in time given that the plan recognition is expected to be made after the deadline $\rho(t)$.

(2.2) $maxChance(t) < \varphi$
In this case, interaction with the target agent is recommended since meeting the deadline is not expected, and there is no hypothesis with chance greater than φ.

3.5 Complexity Analysis

In this section, we analyze the complexity of the main algorithms presented in this paper. We are aware, of course, that our complete method relies on extensions to SBR that might affect the overall complexity (see [2] for a complexity analysis of SBR). Detailed complexity analysis of our method remains future work.

Let n be the number of leaf plan steps in a plan library and let m be the number of entries in the CE tables. Regarding the complexity of the algorithms specified in this section, the order of growth of Algorithm 1 is $O(n \cdot m)$ as this algorithm iterates over the set of leaf plan steps (Lines 1–9), and in the nested level, over the set of entries of the CE table of the plan step (Lines 4–6). Note that in this analysis we do not consider the loop over the set *Observ* (Lines 7–9) given that the size of this set is smaller or equal to the size of the CE table. Therefore, Algorithm 1 is polynomial in the number of leaf plan steps and number of entries in the CE tables. In practice, the time complexity is determined primarily by two factors that affect how big the CE tables can be: the size of the observation space and how often such observations are made.

Let l be the number of level of the hypotheses graph, let k be the number of nodes in a level of the graph, and let m be the the number of entries in the CE tables of the leaf plan steps. Algorithm 3 is performed in $O(l \cdot k \cdot m)$. Again, time complexity is affected by how often such observations are made – for each observation, we generate a new level in the hypotheses graph. In the worst case, all plan steps could be a hypothesis, which would result in the maximum size of nodes per level. In practice, this worst case is not likely to occur: the number of hypotheses tend to reduce as new observations are made.

4. VALIDATION

In this section, we empirically validate ERT and PSC, the components that provide the basis on which decisions about interactions with target agents are made. To do so, we have used the application domain specified in the plan library in Figure 2, and we have simulated a sequence of observations (the observations in the CE table depicted in Figure 4 have been made in the last plan recognition episode). Our validation has been carried out using the results of this simulation.

Once the simulation has ended, we have validated the *ert* values computed with Algorithm 1. The graph of Figure 8 shows the evolution of the *ert* values (y-axis) for the plan step *position* in Figure 7 as updates are made (*nupd*, x-axis). Each line in the graph indicates the expected recognition time *ert* for one of the six distinct observations made during the recognition episodes. As we can see in the graph, *ert* values tend to converge as updates are executed. For instance, the *ert* value for "*location*$(1,3),\ldots$" oscillates in the first 6 episodes and, after this, it stabilizes around 21. It means that if *position* is a valid hypothesis when we observe "*location*$(1,3),\ldots$", we expect to eliminate ambiguity (recognize the plan under execution) in $\sim$21 time steps. Note that the length of the lines can be different since a certain observation can be made more often than another (e.g. "*location*$(1,3),\ldots$" has been observed 20 times within *position*, while "*location*$(2,3),\ldots$" has been observed 9 times.

The plan selection count (PSC) made in Algorithm 3 has also been validated. Each value in the "*nps*" column of the ERT table in Figure 7 corresponds to the number of times that the plan step *position* has been executed by the target agent when the respective observation has been made.

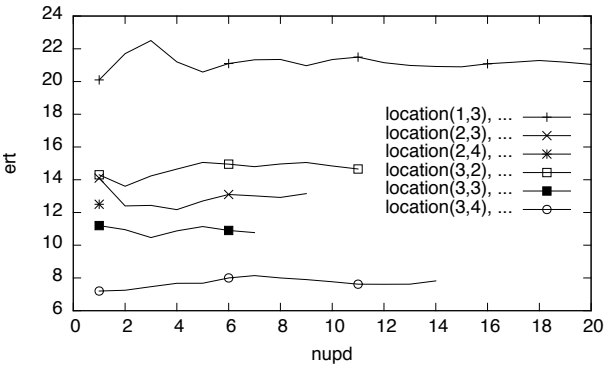


Figure 8: ERT evolution as updates are made. The y-axis indicates the expected recognition time (*ert*), and the x-axis indicates the number of updates (*nupd*) on the *ert* for each distinct observation.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we have developed an extension of the SBR algorithm [2] that not only provides plan hypotheses matching observations, but also provides an anytime expectation of time to eliminate ambiguity (and thus conclude the recognition process) before the algorithm settles for a single plan hypothesis. We also extended SBR with a method to assess the frequency with which particular plans are selected by a user in a given context. This algorithm allows an agent that employs the plan recognizer to make decisions not only on the actual plans recognized, but also on the assumption that further observations will increase certainty over time. Such extensions can be very useful to agents that incorporate plan recognition within a larger reasoning process, providing additional information that can help the agent decide its own courses of action in response to the recognized plan. To this effect, we provide an analysis of the various situations which an agent may face, given information provided by our recognizer. Given our analysis we provide insights into how an agent may respond to various situations with regards to the expected time to recognition and the certainty about the hypotheses. Our current algorithm has a limitation in that we do not allow for interleaved execution and lossy observations as in [5], but we believe that implementing these techniques with our approach should be straightforward, as extensions of SBR with this capability show [5, 3]. As future work, we aim to incorporate an aging mechanism for the observations to cope with changes to the observed agent behavior, as well as integrate the algorithm into a larger application in order to perform experiments in more complex scenarios.

Acknowledgements

Part of the results presented in this paper were obtained through research on a project titled "Semantic and Multi-Agent Technologies for Group Interaction", sponsored by Samsung Eletrônica da Amazônia Ltda. under the terms of Brazilian federal law No. 8.248/91.

6. REFERENCES

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

[2] D. Avrahami-Zilberbrand and G. A. Kaminka. Fast and complete symbolic plan recognition. In *IJCAI*, pages 653–658. Professional Book Center, 2005.

[3] D. Avrahami-Zilberbrand and G. A. Kaminka. Hybrid symbolic-probabilistic plan recognizer: Initial steps. In *Proceedings of the AAAI Workshop on Modeling Others from Observations*, pages 1–7, 2006.

[4] D. Avrahami-Zilberbrand and G. A. Kaminka. Incorporating observer biases in keyhole plan recognition (efficiently!). In *AAAI*, pages 944–949. AAAI Press, 2007.

[5] D. Avrahami-Zilberbrand, G. A. Kaminka, and H. Zarosim. Fast and complete plan recognition: Allowing for duration, interleaved execution, and lossy observations. In *Proceedings of the IJCAI Workshop on Modeling Others from Observations*, 2005.

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

[7] H. H. Bui, S. Venkatesh, and G. A. W. West. Policy recognition in the abstract hidden markov model. *J. Artif. Intell. Res. (JAIR)*, 17:451–499, 2002.

[8] S. Carberry. Techniques for plan recognition. *User Model. User-Adapt. Interact.*, 11(1-2):31–48, 2001.

[9] E. Charniak and R. P. Goldman. A bayesian model of plan recognition. *Artif. Intell.*, 64(1):53–79, 1993.

[10] T. V. Duong, H. H. Bui, D. Q. Phung, and S. Venkatesh. Activity recognition and abnormality detection with the switching hidden semi-markov model. In *CVPR (1)*, pages 838–845, 2005.

[11] C. W. Geib and R. P. Goldman. Plan recognition in intrusion detection systems. In *Proceedings of the Second DARPA Information Survivability Conference and Exposition (DISCEX II)*, pages 329–342, 2001.

[12] E. Horvitz. Principles of mixed-initiative user interfaces. In *CHI*, pages 159–166. ACM, 1999.

[13] M. J. Huber, E. H. Durfee, and M. P. Wellman. The automated mapping of plans for plan recognition. In *AAAI*, page 1460. AAAI Press / The MIT Press, 1994.

[14] G. A. Kaminka, D. V. Pynadath, and M. Tambe. Monitoring teams by overhearing: A multi-agent plan-recognition approach. *J. Artif. Intell. Res. (JAIR)*, 17:83–135, 2002.

[15] G. A. Kaminka and M. Tambe. Robust agent teams via socially-attentive monitoring. *J. Artif. Intell. Res. (JAIR)*, 12:105–147, 2000.

[16] J. Oh, F. Meneguzzi, K. P. Sycara, and T. J. Norman. Prognostic normative reasoning. *Eng. Appl. of AI*, 26(2):863–872, 2013.

[17] S. Okamoto, P. Paruchuri, Y. Wang, K. P. Sycara, J. Marecki, and M. Srivatsa. Multiagent communication security in adversarial settings. In *IAT*, pages 296–303. IEEE Computer Society, 2011.

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

[19] G. Sukthankar and K. Sycara. Activity recognition for dynamic multi-agent teams. *ACM TIST*, 3(1):18, 2011.

[20] J. Voskamp and B. Urban. Measuring cognitive workload in non-military scenarios criteria for sensor technologies. In *Foundations of Augmented Cognition. Neuroergonomics and Operational Neuroscience*, volume 5638 of *LNCS*, pages 304–310. Springer, 2009.