Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

Error-Tolerant Anytime Approach to Plan Recognition Using a Particle Filter

Jean Massardi, Mathieu Gravel, Éric Beaudry
Department of Computer Science
Université du Québec à Montréal
{massardi.jean,gravel.mathieu.3}@courrier.uqam.ca, beaudry.eric@uqam.ca

Abstract

Classical plan recognition approaches require restrictive assumptions and are generally off-line. However, many real-world plan recognition applications must deal with real-time constraints, noisy information, temporal relations in actions, agent preferences, and so on. Many existing approaches have tried to relax assumptions, but none can deal with the above-cited needs. This paper proposes an extension of previous works on plan recognition based on plan tree grammar. Our anytime top-down approach uses a particle filter. This approach manages to give a quick reliable solution to the plan recognition problem while dealing with noisy observations and without reducing the expressiveness of plan libraries. Empirical results on simulated problems show the efficiency of our approach.

Introduction

Plan recognition consists of deducing goals and plans followed by an observed agent. It is an essential component for some intelligent systems interacting with other agents. Examples of plan recognition applications are found in video games (Albrecht, Zukerman, and Nicholson 1998)(Kabanza et al. 2010), detection of hostile behaviors (Geib and Goldman 2001), human-robot interactions (Kelley et al. 2008) and smart-home environments (Bouchard, Giroux, and Bouzouane 2006).

Many existing approaches to plan recognition require a plan library. A plan library is a collection of symbols and rules. Symbols represent goals, sub-goals and actions, while rules represent logical links between these symbols. Plan recognition techniques over plan libraries usually use abductive reasoning in order to infer the plan of an observed agent. In the case of hierarchical plans, abductive reasoning is sometimes called the bottom-up approach (Geib and Goldman 2009) (Raghavan and Mooney 2011).

These approaches suffer from several limitations. Firstly, there is a multiplicity of formalisms used to describe a plan library, such as Context Free Grammar (CFG) (Pynadath and Wellman 2000), the Abstract Hidden Markov Memory Model (AHMMM) (Bui 2003), Bayesian Logic Programs (Raghavan and Mooney 2011) or even a plan as fully ordered sequences of actions (Zhuo 2017). These formalisms have different expressivity and different solving algorithms. In the last decade, CFG describing Hierarchical Task Networks (HTN) from Geib and Goldman (2009) was one of the most used frameworks. Secondly, abductive reasoning over plan recognition problems are inherently heavy in calculus, being at least NP-Hard in a fully observable setting (Vilain 1991) and semi-decidable in general (Behnke, Höller, and Biundo 2015). This leads to difficulties in obtaining real-time results. Finally, although most of the library-based plan recognition algorithms could theoretically support noisy and missing observations, the calculus cost required makes them impractical for real-life applications (Bisson, Larochelle, and Kabanza 2015; Geib and Goldman 2009; Kabanza et al. 2013).

In this paper, we focus on plan recognition with noisy observations, where the observed agent is neither cooperative nor hostile with the observant. This case is known as *keyhole* context (Carberry 2001). To solve this family of plan recognition problems, we propose a new algorithm based on a particle filter.

In the next sections, we present a few previous works on plan recognition, especially on plan library based techniques. Then we discuss the definition of a plan library as well as a decision model and the different types of noisy observations encountered in plan recognition. We also present a general overview of discrete particle filters. We then describe our algorithm for plan recognition based on a particle filter. Finally, we present experimental results showing accuracy and noise tolerance of our approach.

Previous Work

Generally speaking, plan recognition is the opposite task of planning (Sukthankar et al. 2014). While a planning problem focuses on producing a plan – e.g., a partially or fully ordered sequence of actions – plan recognition aims to find the goals of an observed agent, based on its actions. In a typical plan recognition problem, the observations correspond to low-level actions performed by the observed agent. Plan recognition can be done in several contexts, single agent, multi-agent, with or without interaction between the observant and observed agent, with perfect observability or with noises, etc. We focus on single agent plan recognition and

suppose no interaction between the observant and observed agent.

We can divide plan recognition techniques into two families: inverse planning techniques and plan library techniques. Inverse planning techniques (Ramırez and Geffner 2009) use planners as a way to produce plans for the different possible goals and compare these plans with the actual actions of the observed agent. The closer a plan is to the sequence of observed actions, the more probable the underlying goal is to be the one the observed agent wants to perform. These techniques require a description of the problem as a complete planning domain. Plan library techniques rely on the plan description as a collection of behavior related to executing a goal. In this paper, we focus on the HTN plan definition of plan library as proposed by Geib and Goldman (2009). This formalism describes plan library as a plan tree grammar, which can be seen as a hierarchical, partially ordered AND/OR tree. It allows for expressing multiple interleaving goals, hierarchy in goals and actions, loops in execution and can handle several constraint definitions between actions.

The main inference mechanism for plan recognition over plan tree grammars is weighted Bayesian reasoning over a set of valid hypotheses. The main problem consists in finding a way to generate and maintain this set. Geib and Goldman (2009) proposed two types of solutions, top-down approaches and bottom-up approaches, both relying on Bayesian weighted models. Top-down approaches consist in generating all possible outcomes using the plan library, i.e., all possible plans, then pruning this set of hypotheses using the observations. Geib and Goldman argue that this method is not completely feasible due to the high number of hypotheses needed at the initial generation. Bottom-up approaches consist in navigating in plan trees from the leaf to the roots. Hypotheses are generated based on observations, therefore only valid hypotheses are maintained. Generating only valid hypotheses based on constraints is a non-trivial task. These two approaches are calculus-heavy because the generation of hypotheses leads to combinatorial explosion.

Several solutions have been proposed to reduce the calculus cost of bottom-up approaches. Yappr (Geib, Maraist, and Goldman 2008) uses Plan Frontier Fragment Grammars (PFFG) to maintain only plan tree frontiers and not complete plan trees. ELIXIR (Geib 2009) uses Combinatory Categorial Grammar (CCG) to delay the commitment to plan hypotheses. SLIM (Mirsky and Gal 2016) proposes a hybrid top-down/bottom-up approach by limiting the depth of goal recognition in the bottom-up part, thereby limiting the overall runtime. Doplar (Kabanza et al. 2013) proposes an anytime algorithm by controlling the exploration space using bounds on goal probabilities rather than focusing on computing the exact underlying probabilities.

Some recent works have focused on decision models for plan recognition. The decision model corresponds to the preferences of the observed agent over some executions rather than others. For example, if the observed agent wants to satisfy the goal *Make a beverage*, an adult might prefer to make tea and a child might prefer chocolate. With SLIM (Mirsky and Gal 2016), the authors proposed to describe the decision model as a probability repartition over production rules. Bisson, Larochelle, and Kabanza (2015) use a Recursive Neural Network (RNN) to learn this decision model and use the resulting network directly as a plan recognition method. With HARE, Zhuo proposes to represent both the plan library and the decision model as a matrix of rating scores (Zhuo 2017).

Most of the previous works presented in this paper can theoretically handle some kind of noises, but at a high calculus cost. In most cases, this is performed by maintaining non-valid hypotheses which could be ruled out in the case of perfect observability. In the case of non-perfect observability, the bottom-up approach loses its advantages compared to top-down approaches, since invalid hypotheses have to be generated. Another approach proposed by Vattam and Aha (2015) consists in using a similarity metric. In their work, the authors use Action Sequence Graphs to represent their plan library.

Problem Definition

We describe a plan library as defined by Geib and Goldman (2009).

Definition 1. *A plan library (PL) is a tuple* $PL = (A, NT, G, P)$ *where:*

- A *is a finite set of terminal symbols;*
- NT *is a finite set of non-terminal symbols;*
- $G \subset NT$ *is the set of goals;*
- P *is a set of production rules in the form* $\alpha \rightarrow S, \sigma$*, with* $\alpha \in NT$*,* S *a set of symbols from* $A \cup NT$ *and* σ *a partial order of* S*.*

A partial order σ is in the form (i, j), which means that the i^{th} action S_i has to be executed before the j^{th} action S_j in order to complete the task α.

This definition corresponds to a multiset partially ordered Context Free Grammar (CFG). It is similar to HTN planning formalism, without the concepts of preconditions and effects. Terminal symbols correspond to low-level actions. Using this formalism, plans can be described as plan trees, where the goal is the root of the tree and low-level actions are the leaves.

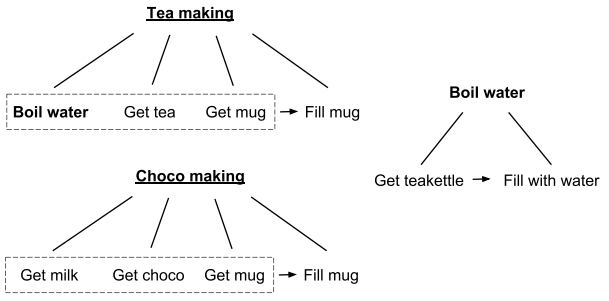

Figure 1: Example of a simple plan library

Figure 1 presents a simple plan library for two root goals and describes the actions needed to make a cup of tea or

a cold chocolate drink. Rectangles indicate permutable actions. In this example, for $Tea\ making$ the terminal actions $Get\ tea$, $Get\ mug$ and the non-terminal action $Boil\ water$ can be performed in any order, but all of them have to be performed before doing $Fill\ mug$. $Boil\ water$ can also be decomposed into terminal actions. $Choco\ making$ can be decomposed using the same logic.

In this example, the underlying plan library is:

A = $\{Get\ teakettle, Fill\ with\ water, Get\ tea, Get\ mug, Fill\ mug, Get\ milk, Get\ choco\ \}$

NT = $\{Boil\ water, Tea\ making, Choco\ making\}$

G = $\{Tea\ making, Choco\ making\}$

P = $\{Tea\ making \rightarrow Boil\ water, Get\ tea, Get\ mug, Fill\ mug, \sigma = \{(1,4),(2,4),(3,4)\}$

$\{Choco\ making \rightarrow Get\ milk, Get\ choco, Get\ mug, Fill\ mug, \sigma = \{(1,4),(2,4),(3,4)\}$

$\{Boil\ water \rightarrow Get\ teakettle, Fill\ with\ water, \sigma = \{(1,2)\}\}$

For our plan library to handle noisy information and preferences over plan execution, we propose to add two more elements, the decision model and the noise function.

Decision Model

In SLIM, Mirsky and Gal declare the decision model (DM) as a part of the production rules. Since a plan library represents a method to achieve a goal, regardless of the operator, and a decision model (DM) represents the preferences for a single agent, we propose to separate the two definitions.

Definition 2. *A Decision Model DM over a Plan Library PL is a probability distribution in the form* $DM : NT \times P \rightarrow [0,1]$*, with the constraint* $\forall nt \in NT, \sum_{p \in P} DM(nt,p) = 1$.

Here we describe a Decision Model as a probability distribution over rules.

For simplicity of usage, we also add prior probability distribution over goals in the decision model. To do so, we add a primitive symbol in NT and corresponding production rules to generate goal and multi-goal setting, as well as the probability distribution of these rules in the function DM.

Taking Figure 1 as an example, we have two distinct goals $Tea\ making$ and $Choco\ making$ in G, we add a symbol in $NT\ Goals$ and two production rules in P in the form $Goals \rightarrow Tea\ making$ and $Goals \rightarrow Choco\ making$. The symbol $Goals$ is not an element of G. The probability distribution of these two rules describes the prior probability of each goal.

Noise Function

Generally speaking, noise in plan recognition can be classified by three types: (1) missing observation, when the observant misses actions done by the observed agent, (2) mislabeled observation, when one observation is mixed with another, (3) extraneous actions, when actions that are not part of the plan are observed.

We propose to describe noise using a single expression as the following.

Definition 3. *A noise function N over a Plan Library is a probability distribution in the form* $N : A \times (A + \emptyset)^p \rightarrow [0,1]$*, with the constraint* $\forall a \in A, \sum_{o \in (A+\emptyset)^p} N(a,o) = 1$ *and* $p \in \mathbb{N}$.

This function can produce four different kinds of behavior, corresponding to the three kinds of noise described earlier and the case where the action is correctly observed:

- $a \rightarrow a \mid a \in A$, case where the action is correctly observed.
- $a \rightarrow \emptyset \mid a \in A$, missing observation.
- $a \rightarrow b \mid a, b \in A$, mislabeled action.
- $a \rightarrow B \mid B \in A^p$ extraneous action. In this case, the correct observation can be part of B, but it is not necessary.

While this noise function can describe complex situations and is a probability distribution rather easy to use, the learning of this function can be a real challenge. As interesting as this problem is, it is not within the scope of this article. We show later that with an estimated noise function we can still obtain exploitable results.

With definitions 2 and 3, we introduce the extended plan library.

Definition 4. *An Extended Plan Library is a tuple* $EPL = (PL, DM, N)$ *where:*

- PL *is a plan library;*
- DM *is a decision model;*
- N *is a noise function.*

Finally, we define a plan recognition problem as the following:

Definition 5. *A plan recognition problem is a tuple* $P = (EPL, O)$ *where:*

- EPL *is an extended plan library.*
- O *is a sequence of observations* $O = \langle o_1, o_2, ..., o_n \rangle$ *where* $o_i \in A$ *.*

The objective of plan recognition is, using these definitions, to find $P(G|O)$ for goal recognition, or $P(Plan|O)$ for plan recognition. We define by $Plan$ the sequence of symbols representing observed and non-observed actions of an agent.

Particle Filter

A particle filter, also known as Sequential Monte-Carlo (Cappé, Godsill, and Moulines 2007), is a class of sampling algorithms used to compute probabilities for Markov processes. They are non-complete and anytime, capable of producing fast results, but with no guarantee of validity. This class of algorithm relies on a genetic approach using a population of simulation called particles. At each step in the process, particles are filtered using incoming observations. To maintain the quality of prediction, the total number of particles is maintained by resampling.

Formaly speaking, we consider the following Markov Process $X_t|X_{t-1} = x_t \sim P(x_t|x_{t-1})$ with the observation $Y_t|X_t = y_t \sim P(y_t|x_t)$. The objective is to estimate

the probability distribution $P(x_t|y_0, ..., y_t)$. If we generate N random variables p from X_0, an estimator of P is given by Equation (1).

$$\hat{P}(x_t|y_0, ..., y_t) = \frac{1}{N} \sum_{i=1}^{n} \delta_{p_k^i}(x_k) \quad (1)$$

In this equation, δ is the Dirac measure.

Generally speaking, a particle filter approach has 4 steps:

1. **Initialization.** Create a population of simulations at state $t = 0$.
2. **Transition.** Advance the particle population at state $t = t + 1$ and compute their new output.
3. **Filtering.** At observation, remove particles having outputs not corresponding with the observations.
4. **Resampling.** Generate a new particle population by randomly choosing surviving particles from the filtering step. Return to step 2.

Like many sampling algorithms, particle filters depend on the number of samples. The more particles, the better the accuracy will be. On many problems, this accuracy will reach a plateau over a certain number of particles. This number depends on the problem.

Particle Filter Algorithm over Plan Trees

We propose to use a particle filter because, in plan recognition, even though there is a large number of plans hypotheses, only a small subset has a significant probability of being true. Therefore, the objective is to generate only a small number of hypotheses while limiting the computational cost of each one of them.

To do so, we propose to use partial plan trees as particles. Partial plan trees can be generated using a top-down approach rather than a bottom-up one. Top-down sequence generations are comparatively lighter in calculus than bottom-up approaches (Geib and Goldman 2009).

We represent a particle p as a tuple $p = (G, T, ENO)$ with:

- G is a goal or a set of goals;
- T is a partial plan tree for G;
- ENO corresponds to the Expected Next Observation. It is the last computed low-level action using a top-down approach to expand T.

Expected Next Observation Generation

As previously stated, we use plan trees as particles. To represent the elements in these plan trees, we use $Symbol\ Nodes$. A $Symbol\ Node$ is a tuple $SN = (s, C, r, f)$ with s the underlying symbol from $NT \cup A$, C the list of child $Symbol\ Nodes$ of SN, r a production rule of s, and f a boolean representing the current status of the $Symbol\ Node$, finished or not. A node is finished if (1) every symbol S in r exist in C; and (2) every c in C is finished. The root of a plan tree is a $Symbol\ Node$ with $s \in G$.

We will use the 2 following functions, described as follows:

1. **RNC**(P, S). This function describes a random selection using the probability distribution P over a set of elements S.
2. **VALIDSYM**(r, C). This function uses the constraints of the production rule r over the set C of $Symbol\ Nodes$ to return a valid symbol, or $\perp$ if no such symbol exists. A valid symbol has to satisfy two conditions: (1) it is not finished ($f = false$); and (2) all constraints have to be satisfied. This symbol is chosen non-deterministically over all possible valid symbols.

Algorithm 1 Expand Tree

```
1: function Expand(SN = (s, C, r, f), EPL)
2:     if s ∈ A then
3:         f ← True
4:         return RNC(N, s)
5:     end if
6:     c ← VALIDSYM(R, C)
7:     if c ∈ C then
8:         return Expand(C(c))
9:     end if
10:    if c = ⊥ then
11:        return ∅
12:    end if
13:    C(c) ←(c, ∅, RNC(DM, c), False)
14:    result ← Expand(C(c))
15:    if For all s in r, C(s).f = True then
16:        f = True
17:    end if
18:    return result
19: end function
```

Algorithm 1 is the recursive algorithm used to expand a plan tree and emit a symbol. There are two cases. Firstly, if the $Symbol\ Node$ is terminal, the function returns its underlying symbol with noise, and its status changes to finished ($F = True$). Secondly, the function `Expand` is applied to the tree and returns a valid symbol from its children. If all the symbols from R exist and are finished in C, the $Symbol\ Node$ changes its status to finished. If there is no valid symbol, the algorithm returns $\emptyset$.

This algorithm is similar to the one used for top-down approaches and the one used to generate test sequences for plan library based approach using this formalism. Here, we use it as a particle generator. However, there are two fundamental differences in the top-down approach from Geib and Goldman: (1) we develop the plan tree one node at a time in order to limit both the size of the tree and the commitment of each particle; and (2) we include the noise directly in this expansion algorithm. We thus avoid dealing with noise outside of the main inference algorithm. Dealing with noise inside tree expansion means we can avoid maintaining a large number of invalid hypotheses without sacrificing expressivity of the model.

Initialization

We use a structure called Pop which contains all the particles. We declare Pop as a map using symbols $s \in A$ as the

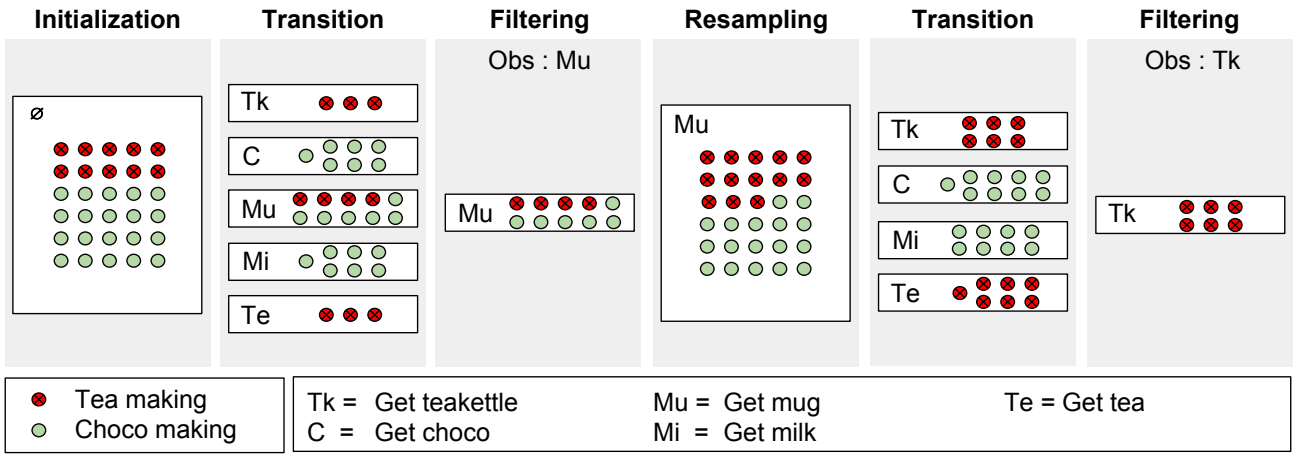

Figure 2: Plan recognition process

key to a set of particles p. s correspond to the last expected next observation of algorithm 1 produced. Algorithm 2 is used to generate the initial population.

Algorithm 2 Initialization

```
1: function Init(size)
2:     while Size(Pop) < size do
3:         s ← RNC(DM, goal)
4:         T ←(s, ∅, RNC(DM, s), False)
5:         ENO ← Expand(T)
6:         p ← (s, T, ENO)
7:         Pop[ENO].insert(p)
8:     end while
9:     return Pop
10: end function
```

$RNC(DM, goal)$ represents the random choice of goal for the root of a plan tree using the prior probability in the decision model, as described earlier.

Like every approach using a particle filter, the objective is to maintain a population large enough to obtain acceptable results and small enough to get these results quickly. The larger the population, the better the results.

Filtering and Resampling

We use each new observation to filter the population of particles using the following algorithm.

In this algorithm, we do not present how we treat the particles when multiple symbols have been emited, in case of extraneous action. To do so, ENO has to be changed to a stack of symbol rather than a singleton.

To solve the goal recognition problem, i.e., find the most probable goal, we count the number of corresponding particles present in the population map for each possible goal. The most probable goal is the one with the highest number of particles.

Algorithm 3 Filtering algorithm

```
1: function Filter(Pop, o ∈ A)
2:     size ← Size(Pop)
3:     FilteredPop ← Pop[o]
4:     Clear(Pop)
5:     while Size(newPopulation) < size do
6:         p ← random select(FilteredPop)
7:         p.ENO ← Expand(p.T)
8:         Pop[ENO] ← insert(p)
9:     end while
10:    return ePopulation
11: end function
```

$$P(G|O) = \frac{1}{|Pop|} \sum_{p \in Pop} \delta_p(p.G = G) \qquad (2)$$

In this version of the algorithm, we expand particles before filtering. Placing the expansion step before the filtering step enabled us to use the same algorithm to anticipate the next action of an observed agent. The probability of an agent's next action can be expressed as $P(o_{t+1}|o_t)$. This approach can be easily modified to expand and filter at the same time.

Using algorithm 1 and algorithm 3, we avoid generating an exponential number of hypotheses which depend on the number of observations. Observations are only used as a filter and not directly in plan tree generation.

Figure 2 presents an example of population filtering over the plan library introduced by Figure 1. The problem here is to identify what beverage an observed agent is making, tea or chocolate. We focus only on the filtering aspect, and not on the generation of the expected next observation, for each particle. We suppose no noise and a prior probability of 1/3 for $Tea\ making$ and 2/3 for $Choco\ making$. At the Initialisation stage, there is no observation made yet, we randomly choose goals and create particles corresponding to these goals. In the first transition, we expand the trees and

generate the first expected next observations (in this example, there are five possible next observations: Tk, C, Mu, Mi and Te). These particles are sorted depending on their expected next observation. These two stages correspond to Algorithm 2. When an observation is made (here, it is Mu), the particles are filtered. We keep only particles where the expected next observation correspond to the observation. At the resampling stage, the surviving particles are duplicated to maintain the population size. The particles are then mutated again at the second transition stage and sorted again by their new expected next observation. These three stages, Filtering, Resampling and Transition, correspond to Algortihm 3. At the last filtering stage, we observe Tk, the only surviving particles are from $Tea\ making$, therefore, it is the goal behind these two observations.

Experiments

We implemented our proposed approach in C++. For PHATT and SLIM, we use the public Python implementation from Mirsky. Experiments are made on a PC with an Intel Core i7-6600U 2.6 GHz CPU and 2 GB of RAM. The tests aimed to: (1) verify the accuracy of this new algorithm; (2) compare its speed to other state-of-the-art approaches; and (3) assess its tolerance to noisy observations.

We use a simulated plan library similar to DOPLAR (Kabanza et al. 2013) and SLIM (Mirsky and Gal 2016). This simulated domain is a randomly generated domain with 100 low-level actions, 5 goals, a depth of 4, a branching factor of 3 for AND rules and a branching factor of 2 for OR rules. We consider equi-probability for every rule and the identity function for the noise function. With these settings, the length of a plan is nine actions. We add a 33% probability of ordering constraint between any pair of symbols in all the production rules.

Accuracy with Full Observability

The first set of experiments aims to assess the accuracy of the particle filter approach. We ran our algorithm on 100 instances of simulated plan libraries. For each instance, we performed the tests using different population sizes.

Figure 3 presents the average accuracy of the goal recognition algorithm as a percentage (i.e, how often the algorithm returns the true goal) over the percentage of plan completion for 50, 100, 250 and 500 particles. We also experimented with 750 and 1,000 particles, and the results were nearly identical to 500 particles. This demonstrates that increasing the number of particles beyond 500 does not improve precision in goal recognition nor decrease the percentage of plan completion needed to identify the goal. At any percentage of plan completion, the average accuracy of goal recognition depends on the number of particles, the higher the better, up to 500 particles. As the plan completion progresses, the average accuracy of goal recognition increases from below 20% to nearly 60% for 50 particles and from 20% to 100% for 500 particles. For every particles number chosen, the average percentage accuracy of goal recognition reaches to a plateau at 30% of plan completion, meaning the algorithm reaches a definitive conclusion about the recognized goal before the plan is achieved, about a third way

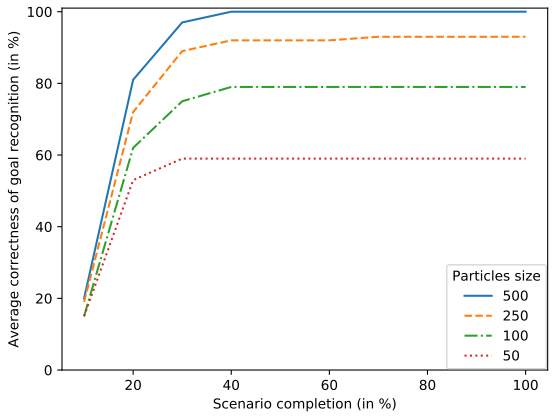

Figure 3: Average rank of the correct goal by scenario completion with full observability

through its execution. We compared the performances with SLIM and PHATT and we observed less than 1% difference using our particle filter with 500 particles. These results show that this approach produces reliable results, even if it is not a complete approach that can guarantee a result.

These results vary depending on several characteristics in the plan libraries. For instance, with more goals, 10 instead of 5, the convergence to a solution is reached between 10% and 20% rather than 30%. Likewise, the fewer the number of actions, the more ambiguous the problem. With 10 low-level actions rather than 100, convergence is reached around 60% where maximum accuracy is at 80% rather than 100%.

Performances

We compare our approach to PHATT and SLIM approaches, well-known complete approaches with a public distribution.

Table 1: Average runtime in seconds for PHATT, SLIM and for a particle filter approach with 500 particles

Obs.	1	2	3	4	5	6	7	8	9
PHATT	0.25	0.35	0.57	0.87	1.23	1.71	2.58	4.61	9.88
SLIM	0.17	0.17	0.19	0.26	0.46	0.78	1.28	2.02	4.60
PF-500	0.011	0.019	0.027	0.035	0.043	0.051	0.059	0.067	0.075

Table 1 presents the average runtime for PHATT, SLIM and our approach (PF-500). Since the number of hypotheses generated by our approach is constant (500 particles), and the approach is online, the runtime difference between two observations is constant at 0.008 seconds, except for the first observation at 0.011 seconds. We suppose this increase is due to data structure initializations. This represents a vast improvement in the average runtime compared to PHATT and SLIM as these two approaches are complete bottom-up approaches, meaning the number of hypotheses generated, and thus the average runtime, increases exponentially over the number of observations. We can see that for the first observation our approach already has a lower average runtime by order of magnitude (0.011 seconds) than PHATT and SLIM (0.25 seconds and 0.17 seconds, respectively).

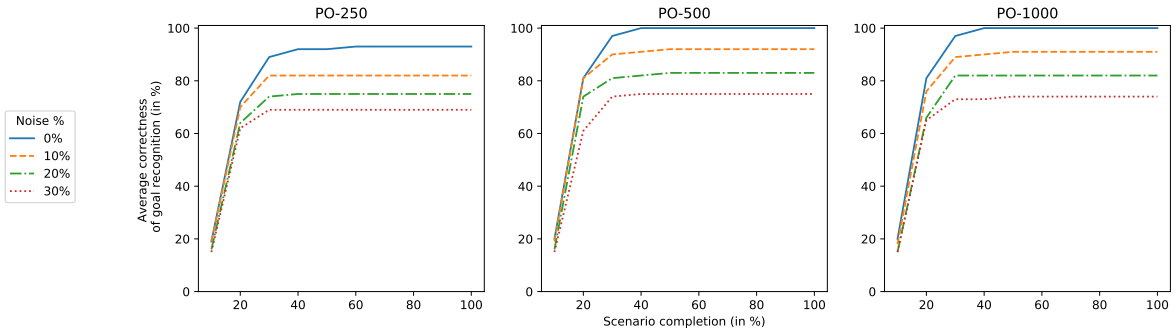

Figure 4: Average rank of the correct goal by scenario completion for noisy information

The fact that our approach has a constant average runtime regardless of the number of observations makes it very interesting in the case of long plans (i.e, more than five actions) and useful in real-world applications.

Accuracy with Noise

We then tested the robustness of our approach against noisy information. To do so, we add a noise function to our simulated plan libraries. We set three different levels of noise at 10%, 20%, and 30%. These percentages correspond to the number of erroneous observations, regardless of the kind of noise (missing, mislabeled, or extraneous). For this test, the three types of noises have the same probability of occurrence.

We set the particle population to 250, 500, and 1,000 to see if the population would impact the results the same way as in a full observability context. We ran 100 tests for each population size.

Figure 4 presents the average accuracy of goal recognition as a percentage over the plan completion for all the three population sizes. For 250 particles in a full observability setting, the average rank of correctly recognized goals reaches a plateau at 30% of scenario completion, around 95% accuracy. This plateau is reached at the same percentage of scenario completion regardless of the noise addition. The average rank of correctly recognized goals decreases linearly with noise increasing to the lowest point of 65% of goals recognized correctly for 30% of noisy observations. Increasing the number of particles to 500 did not lower the percentage of scenario completion required to reach the plateau for the average rank of correctly recognized goals, however, the value of this plateau is close to 100% of goals recognized correctly in a full observability setting and decreases linearly with noise introduction up to the lowest value of 70% with 30% of noisy observations. Increasing the number of particles to 1,000 only marginally improves the results. According to these results, a number of 500 particles is sufficient to get optimal results in terms of percentage of scenario completion required to reach the plateau of correctly recognized goals and in terms of average rank of correct goals recognized with or without noise. Our approach is robust to noisy observations as the performances decrease linearly with the percentage of noise.

We also tried to account for difference in performance based on the type of noise. We ran our approach at 20% noise with four different settings: (1) only missing observations (M); (2) only mislabeled (ML); (3) only extraneous (E); and (4) all of the previous ones equally distributed (A).

Table 2: Average correct goal recognition in percent at the end of a plan for a particle filter with 500 particles

Noise Type	20% M	20% ML	20% E	20% A
PF-500	83	79	83	81

Table 2 shows the results. As we can see, the noise type does not significantly impact the accuracy of our approach.

Conclusion and Discussion

We proposed a new approach to plan recognition over plan libraries using a particle filter with a population of plan trees. This is an anytime approach and does not limit the expressivity of plan libraries as contextual grammar. This approach also deals with noisy observations. We provide evidence of its efficiency on simulated plan libraries.

As all plan library based techniques, this approach requires complementary learning techniques to create and maintain the plan library, as well as the decision model and noise function.

In future works, we plan to work on concurrency as well as temporal reasoning of low-level actions. Temporal information can be used for two purposes: (1) provide complementary information to characterise low-level actions, which can be used in order to improve the plan recognition performances, (2) provide temporal information about when a probable next observation will occur. The second point is critical for temporal planning tasks.

Acknowledgments

This work was supported by Aging Gracefully across Environments using Technology to Support Wellness, Engagement and Long Life NCE Inc. (AGE-WELL NCE), a canadian federal program. We would like to thank Amandine Laffitte and Guillaume Gosset for their help.

References

Albrecht, D. W.; Zukerman, I.; and Nicholson, A. E. 1998. Bayesian models for keyhole plan recognition in an adventure game. *User Modeling and User-Adapted Interaction* 8(1-2):5–47.

Behnke, G.; Höller, D.; and Biundo, S. 2015. On the complexity of htn plan verification and its implications for plan recognition. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 25–33.

Bisson, F.; Larochelle, H.; and Kabanza, F. 2015. Using a recursive neural network to learn an agent's decision model for plan recognition. In *International Joint Conference on Artifical Intelligence (IJCAI)*, 918–924.

Bouchard, B.; Giroux, S.; and Bouzouane, A. 2006. A smart home agent for plan recognition of cognitively-impaired patients. *Journal of Computers* 1(5):53–62.

Bui, H. H. 2003. A general model for online probabilistic plan recognition. In *International Joint Conference on Artifical Intelligence (IJCAI)*, 1309–1315.

Cappé, O.; Godsill, S. J.; and Moulines, E. 2007. An overview of existing methods and recent advances in sequential monte carlo. *Institute of Electrical and Electronics Engineers (IEEE)* 95(5):899–924.

Carberry, S. 2001. Techniques for plan recognition. *User Modeling and User-Adapted Interaction* 11(1-2):31–48.

Geib, C. W., and Goldman, R. P. 2001. Plan recognition in intrusion detection systems. In *DARPA Information Survivability Conference (ISC)*, volume 1, 46–55. IEEE.

Geib, C. W., and Goldman, R. P. 2009. A probabilistic plan recognition algorithm based on plan tree grammars. *Artificial Intelligence* 173(11):1101 – 1132.

Geib, C. W.; Maraist, J.; and Goldman, R. P. 2008. A new probabilistic plan recognition algorithm based on string rewriting. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 91–98.

Geib, C. W. 2009. Delaying commitment in plan recognition using combinatory categorial grammars. In *International Joint Conference on Artifical Intelligence (IJCAI)*, 1702–1707.

Kabanza, F.; Bellefeuille, P.; Bisson, F.; Benaskeur, A. R.; and Irandoust, H. 2010. Opponent behaviour recognition for real-time strategy games. *AAAI workshop Plan, Activity, and Intent Recognition (PAIR)* 10(5).

Kabanza, F.; Filion, J.; Benaskeur, A. R.; and Irandoust, H. 2013. Controlling the hypothesis space in probabilistic plan recognition. In *International Joint Conference on Artifical Intelligence (IJCAI)*, 2306–2312.

Kelley, R.; Tavakkoli, A.; King, C.; Nicolescu, M.; Nicolescu, M.; and Bebis, G. 2008. Understanding human intentions via hidden markov models in autonomous mobile robots. In *ACM/IEEE International Conference on Human Robot Interaction (HRI)*, 367–374.

Mirsky, R., and Gal, Y. 2016. Slim: Semi-lazy inference mechanism for plan recognition. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 394–400.

Pynadath, D. V., and Wellman, M. P. 2000. Probabilistic state-dependent grammars for plan recognition. In *Uncertainty in Artificial Intelligence (UAI)*, 507–514. Morgan Kaufmann Publishers Inc.

Raghavan, S., and Mooney, R. J. 2011. Abductive plan recognition by extending bayesian logic programs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 629–644. Springer.

Ramırez, M., and Geffner, H. 2009. Plan recognition as planning. In *International Joint Conference on Artifical Intelligence (IJCAI)*, 1778–1783.

Sukthankar, G.; Geib, C.; Bui, H. H.; Pynadath, D.; and Goldman, R. P. 2014. *Plan, Activity, and Intent Recognition: Theory and practice*. Newnes.

Vattam, S. S., and Aha, D. W. 2015. Case-based plan recognition under imperfect observability. In *International Conference on Case-Based Reasoning (ICCBR)*, 381–395. Springer.

Vilain, M. B. 1991. Deduction as parsing: Tractable classification in the kl-one framework. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 464–470.

Zhuo, H. H. 2017. Human-aware plan recognition. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 3686–3693.