Efficient Plan Recognition for Dynamic Multi-agent Teams

Gita Sukthankar
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, FL 32816
gitars@eecs.ucf.edu

Katia Sycara
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
katia@cs.cmu.edu

Abstract

This paper addresses the problem of plan recognition for multi-agent teams. Complex multi-agent tasks typically require dynamic teams where the team membership changes over time. Teams split into subteams to work in parallel, merge with other teams to tackle more demanding tasks, and disband when plans are completed. We introduce a new multi-agent plan representation that explicitly encodes dynamic team membership and demonstrate the suitability of this formalism for plan recognition. From our multi-agent plan representation, we extract local temporal dependencies that dramatically prune the hypothesis set of potentially-valid team plans. The reduced plan library can be efficiently processed using existing tree search techniques to obtain the team state history. Although multi-agent plan recognition is theoretically more computationally expensive than single-agent plan recognition, we show that, in practice, the presence of agent resource dependencies significantly reduces the set of potentially-valid plans.

Introduction

In this paper, we address the problem of *multi-agent plan recognition*, the process of inferring actions and goals of multiple agents from a sequence of observations and a plan library. Although multiple frameworks have been developed for single-agent plan recognition, there has been less work on extending these frameworks to multi-agent scenarios. In the simplest case, where all of the agents are members of one team and executing a single team plan (e.g., players executing a single football play), plan recognition can be performed by concatenating individual agent observations and matching them against the team plan library (Intille & Bobick 1999). However, this is not possible for many complex multi-agent scenarios that require agents to participate in *dynamic teams* where team membership changes over time (Tambe 1997). In such scenarios, teams split into subteams to work in parallel, merge with other teams to tackle more demanding tasks, and disband when plans are completed. Although it is possible to model and recognize such tasks using single-agent plan recognition techniques, we demonstrate that the existence of agent resource dependencies in the plan library can be leveraged to make the plan recognition process more efficient, in the same way that plan libraries containing certain temporal ordering constraints can reduce the complexity of single-agent plan recognition (Geib 2004).

We present an algorithm for multi-agent plan recognition that leverages several types of agent resource dependencies and temporal ordering constraints in the plan library to prune the size of the plan library considered for each observation trace. Thus, our technique can be used as a preprocessing stage for a variety of single-agent plan recognition techniques to improve performance on multi-agent plan recognition problems. We also introduce a multi-agent planning formalism that explicitly encodes agent resource requirements and illustrate how temporal dependencies extracted from this formalism can be precompiled into an index to be maintained in conjunction with the plan library. We demonstrate the performance of our recognition techniques in the domain of military plan recognition for large scenarios containing 100 agents and 40 simultaneously-executing plans.

Problem Formulation

We formulate the multi-agent plan recognition problem as follows. Let $\mathcal{A} = \{a_0, a_1, \ldots, a_{N-1}\}$ be the set of agents in the scenario. A **team** consists of a subset of agents, and we require that an agent only participate in one team at any given time; thus a **team assignment** is a set partition on $\mathcal{A}$. During the course of a scenario, agents can assemble into new teams; similarly, teams can disband to enable their members to form new teams. Thus the team assignment is expected to change over time during the course of a scenario. The observable actions of a team are specified by a set of **behaviors**, $\mathcal{B}$. We assume that the sequence of observed behaviors is the result of an execution of a team plan, P_r drawn from a known library $\mathcal{P}$.

Let $\mathcal{T} = \{T_0, T_1, \ldots, T_{m-1}\}$ be the set of **agent traces**, where each trace T_i is a temporally-ordered sequence of tuples with observed behaviors and their corresponding agent assignment:

$$T_i = ((B_0, \mathcal{A}_{i,0}), (B_1, \mathcal{A}_{i,1}), \ldots (B_t, \mathcal{A}_{i,t})),$$

where $B_t \in \mathcal{B}$ is the observed behavior executed by a team of agents $\mathcal{A}_{i,t} \subset \mathcal{A}$ at time t. Note that the composition of the team can change through time as agents join and leave the team.

Our goal is to identify the set of plans, $\mathcal{P}_i$ that is consistent with each trace, T_i, and the corresponding execution path

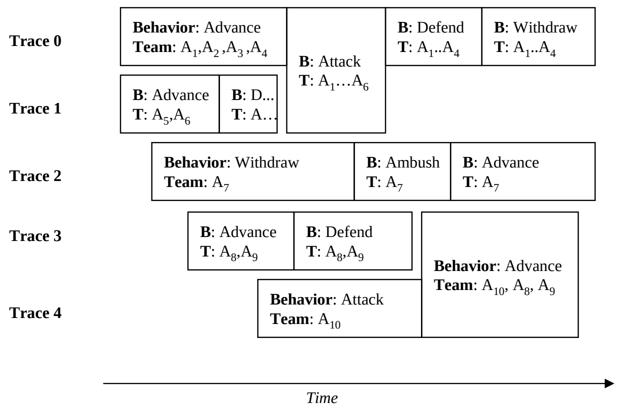


Figure 1: Example execution traces, $\mathcal{T} = \{T_0, T_1, \ldots, T_4\}$. Each trace T_i is a temporally-ordered sequence of tuples with observed behaviors, B, and their corresponding agent assignments, $\mathcal{A}$. Note that the composition of the team can change through time as agents join and leave the team. The goal of multi-agent plan recognition is to identify the plan trees that generated these execution traces.

through each plan (see Figure 1). This can be challenging since most of the nodes in a plan tree do not generate observable behaviors and multiple nodes in a single plan tree can generate the same observation.

Multi-agent Plan Representation

In this section, we describe our extensions to the hierarchical task network plan libraries generally used for single-agent planning (Erol, Hendler, & Nau 1994). The principal purpose of our multi-agent representation is to correctly model dependencies in parallel execution of plans with dynamic team membership. Although our simulator uses this plan representation to generate synthetic team plans, we do not suggest that the use of this formalism alone guarantees good multi-agent coordination. Since our simulator operates in a centralized fashion we deliberately omit considerations of communication cost and privacy from our decision-making process. Instead, we focus on modeling possible sequences of externally observable changes that can occur during the multi-agent execution process.

Each plan is modeled as a separate AND/OR tree, with additional directed arcs that represent ordering constraints between internal tree nodes. Observable actions are represented as leaf nodes. These nodes are the only nodes permitted to have sequential self-cycles; no other cycles are permitted in the tree.

Additionally all plans are marked with an *agent resource requirement*, the number of agents required for the plan to commence execution (additional agents can be recruited during subsequent stages of a plan). For our military team planning domain, most leaf nodes represent observable multi-agent behaviors (e.g., movement in formation) and thus require multiple agents to execute. Note that the agent resource requirement specified in the top level node does not represent the maximum number of agents required to execute all branches of the plan, merely the number of agents required to commence plan execution.

We use two node types, **SPLIT** and **RECRUIT**, to represent the splitting and merging of agent teams. When a plan requires multiple subteams to execute subplans in parallel, the **SPLIT** node is used. The children of the **SPLIT** node will commence execution in parallel; optionally the parent plan can continue if there are still nodes in the original plan that require execution. Merging teams are represented by **RECRUIT** nodes. **RECRUIT** nodes are a mechanism for teams to acquire more members to meet an agent resource requirement; if no agents can be found, plan execution blocks at the **RECRUIT** node until sufficient agents become available. Agents that are not currently performing an action are marked as executing a **WAIT**. All plan trees terminate with a **WAIT** node that frees agents for recruitment by other plan trees. **SPLIT** and **RECRUIT** are not directly observable actions and must be inferred from changing team sizes in observable leaf nodes. Note that parallel agent traces are not assumed to be synchronized with strong timestamps, nor are observations assumed to have atomic duration. Our simulator (described below) records the order in which each trace commenced execution but there is no timestamp synchronization across traces.

Method

In this section, we discuss our method of automatically recovering and utilizing hidden structure embedded in user-defined multi-agent plan libraries. This hidden structure can be efficiently discovered when the plan library is created, indexed in tables that are stored and updated along with the plan library, and used as part of a pre-processing pruning step before invoking plan recognition to significantly reduce the number of plan libraries considered for each observation trace.

Implicit Temporal Dependencies

Traditional plan recognition would examine each trace T_i independently, and test each plan from the library $P_r \in \mathcal{P}$ against the trace to determine whether P_r can explain the observations in T_i. We propose uncovering the structure between related traces T_i and T_j to mutually constrain the set of plans that need to be considered for each trace.

Note that we cannot determine which traces are related simply by tracking a single agent through time as that agent may be involved in a series of unconnected team plans. However, by monitoring team agent memberships for traces T_i and T_j, we can hypothesize whether a subset of agents $\mathcal{A}_j$ from T_i may have left as a group to form T_j. In that case the candidate plans P_r and P_s for traces T_i and T_j, respectively, must be able to generate observations that explain both the final observation of $\mathcal{A}_j$ in T_i (not necessarily the final observation in T_i) and the initial observation of $\mathcal{A}_j$ in T_j.

Our method does not require synchronization across traces since it is difficult in practice to predict execution for a plan tree: (1) when behaviors have stochastic durations; (2) many execution paths exist for a given plan tree; (3) plan execution can block pending agent resource requirements. Although we eschew timestamps, we can still determine temporal causality relationships between traces.

Similar temporal dependencies also exist between consecutive observations during a single execution trace. For instance, the observation sequence (B_p, B_q) can typically not be generated by every plan in the library, particularly if $|\mathcal{B}|$ is large or when plans exhibit distinctive behavior sequences. These dependencies are implicitly employed by typical plan recognition algorithms; our work generalizes this concept across related execution traces.

Plan Library Pruning

Our method exploits the implicit temporal dependencies between observations, across and within traces, to prune the plan library and to dramatically reduce the execution time of multi-agent plan recognition. Our algorithm for recovering hidden dependencies from the plan library proceeds as follows. First, we construct a hashtable, h that maps pairs of observations to sets of plans. Specifically, $h : B_p \times B_q \rightarrow \{P_j\}$ iff some parent plan P_i could emit observation B_p immediately before subteam formation and its subplan P_j could emit observation B_q immediately after execution. h can be efficiently constructed in a single traversal of the plan library prior to plan execution.

Given the pair of observations, we can identify the set of candidate plans that qualify as subplans for the identified parent. This enables us to significantly restrict the plan library for the child trace. The temporal dependencies that exist between consecutive observations in a single execution trace can be exploited to further prune the set of potential plans. This is also implemented using a hashtable, g, that maps pairs of potentially-consecutive observations within a plan tree to sets of plans, which we also precompute using a single traversal of the plan library.

The size of these hashtable can be $O(|\mathcal{B}|^2|\mathcal{P}|)$ in the worst case since each entry could include the entire set of plans. In practice h and g are sparse both in entries and values. Applying h requires one lookup per execution trace while g requires a linear scan through the observations.

Results

Before describing the results of our experiments, we first present our methodology for creating a plan library and simulating execution traces that respect both temporal and resource constraints.

Plan Library Generation

We follow the experimental protocol described in (Avrahami-Zilberbrand & Kaminka 2005), where simulated plan libraries of varying depths and complexity are randomly constructed. Randomly-generated plans do not reflect the distinctive structure of real-world plans and are therefore a pessimistic evaluation of our method since it relies so heavily on regularities between consecutive observations (both within and between plans). The plan trees are randomly assembled from **OR**, **AND**, **SPLIT**, **RECRUIT** nodes, and leaf (behavior) nodes. Adding a higher percentage of **SPLIT** nodes into the tree implicitly increases the number of execution traces since our simulator (described below) creates a new execution trace for each subplan generated by a **SPLIT**.

Table 1: Default plan generation parameters

Parameter	Default
Number of agents $|\mathcal{A}|$	100
Plan library size $|\mathcal{L}|$	20
Plan tree depth (avg)	4
Plan tree branching factor (avg)	3
Observation label size $|\mathcal{B}|$	10
Parallel execution traces (average)	12

Execution Trace Generation

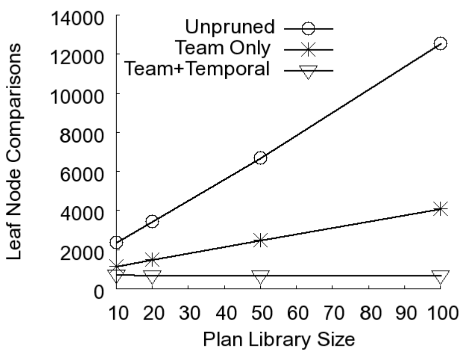
Given a plan library and a pool of agents, the execution trace generator simulates plan execution by allocating agents from the pool to plans as they commence execution and blocking plans at **RECRUIT** nodes while agent resource constraints remain unfulfilled. Note that a given plan tree can generate many node sequences; the same node sequence will execute differently based on which other plans are drawing from the agent pool.

Evaluation

To evaluate the efficacy of our method, we examine three pruning strategies over a range of conditions. The default settings for each parameter are shown in Table 1. To reduce stochastic variation, the following graphs show results averaged over 100 experiments.

On average, the across-trace (h) and within-trace (g) hashtables are at 19% and 70% occupancy, respectively. The average number of plans hashed under each key is 1.14 and 2.87, respectively. The average wall-clock execution time for the default scenario, on a 3.6 GHz Intel Pentium 4, is only 0.14s, showing that multi-agent plan recognition for a group of 100 agents is feasible.

Since plan recognition methods can return multiple hypotheses for each trace, the natural metrics for accuracy are precision and recall. The former measures the fraction of correctly-identified traces over the number of returned results while the latter is the ratio between the number of correctly-identified traces to the total number of traces. Since all of the methods evaluated in this paper are complete, it is unsurprising that they achieve perfect recall on all of our experiments. Precision drops only when multiple plan trees match the observed trace. In these experiments, precision was near-perfect for all methods, indicating that there was little ambiguity in the generated traces. In a small number of cases (where the observable action vocabulary was small), our method achieved higher precision than the baseline because it was able to disambiguate otherwise identical traces based on parent-child dependencies. However, given the infrequency of these cases, we cannot claim significant im-



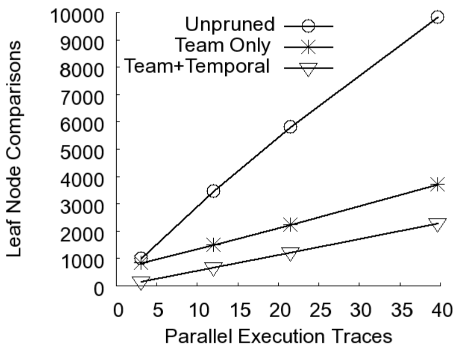


Figure 2: Cost of plan recognition time (as measured by leaf node comparisons) with (a) plan library size, $|\mathcal{P}|$, and (b) increasing number of execution traces.

provements in increasing the precision of multi-agent plan recognition over baseline complete methods.

Figure 2 shows the relative matching cost (execution time) for three recognition strategies:

Unpruned: standard depth-first matching of every observation trace against each plan in the library.

Team Only: prune plan libraries for each observation trace using across-trace dependencies from h before running depth-first matching.

Team+Temporal: prune plan libraries using both within-trace dependencies stored in g, and across-trace dependencies from h, before running depth-first matching.

We observe that:

1. Our combined pruning strategy (Team+Temporal) performs extremely well as the size of the plan library grows. While the cost of the baseline algorithm increases linearly, the cost of our method remains almost constant.
2. Although all of the methods scale linearly with the number of execution traces, it is clear that the cost of our method grows much more slowly. This enables us to tackle large multi-agent scenarios without difficulty.
3. Most plan recognition algorithms scale poorly with increasing average depth of the plan trees. While our scaling behavior is similar, the pruning dramatically reduces the matching cost enabling us to tackle more complex team plans.
4. As the number of observation labels grows, the within-agent pruning shows clear benefits since transitions become more distinctive.

Discussion

Geib (2004) discusses the problem of plan library authoring and suggests that users should refrain from including plans that share a common unordered prefix of actions in their libraries due to the enormous increase in potential explanations for a given observation sequence. In our algorithm, we discover characteristics of the plan library that *compress* the number of potential explanations. The benefits of implementing this process as an automatic preprocessing step are:

1. By automatically recovering this hidden structure, we remove some of the burden of plan library authorship from the user.
2. Preprocessing the plan library allows us to remain agnostic in our actual choice of plan recognition algorithm.

Although there is some amount of hidden temporal structure in single-agent plan libraries, when plans involve the formation of teams, additional structure is created by the enforcement of agent resource requirements.

Conclusion

This paper presents a method for efficiently performing plan recognition on multi-agent traces. We automatically recover hidden structure in the form of within-trace and across-trace observation dependencies embedded in multi-agent plan libraries. Our plan library pruning technique is compatible with existing single-agent plan recognition algorithms and enables these to scale to large real-world plan libraries. We are currently applying our symbolic plan recognition method to activity recognition for physically-embodied agent teams, such as squads of military operations in urban terrain (MOUT).

References

Avrahami-Zilberbrand, D., and Kaminka, G. 2005. Fast and complete symbolic plan recognition. In *Proceedings of International Joint Conference on Artificial Intelligence*.

Erol, K.; Hendler, J.; and Nau, D. 1994. HTN planning: Complexity and expressivity. In *Proceedings of National Conference on Artificial Intelligence*.

Geib, C. 2004. Assessing the complexity of plan recognition. In *Proceedings of National Conference on Artificial Intelligence*.

Intille, S., and Bobick, A. 1999. A framework for recognizing multi-agent action from visual evidence. In *Proceedings of National Conference on Artificial Intelligence*.

Tambe, M. 1997. Towards flexible teamwork. *Journal of Artificial Intelligence Research* 7:83–124.