Hypothesis Pruning and Ranking for Large Plan Recognition Problems

Gita Sukthankar
School of EECS
University of Central Florida
Orlando, FL
gitars@eecs.ucf.edu

Katia Sycara
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA
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 to obtain the team state history. Naive pruning can be inadvisable when low-level observations are unreliable due to sensor noise and classification errors. In such conditions, we eschew pruning in favor of prioritization and show how our scheme can be extended to rank-order the hypotheses. Experiments show that this robust pre-processing approach ranks the correct plan within the top 10%, even under conditions of severe noise.

Introduction

Proficient teams can accomplish goals that would not otherwise be achievable by groups of uncoordinated individuals. Often when a task is too complicated to be performed by an individual agent, it can be achieved through the coordinated efforts of a team of agents over a period of time. In real life, human teams can be found everywhere performing a wide variety of endeavors, ranging from the fun (sports, computer games) to the serious (work, military). Moreover, teams exist in the virtual world as well—in simulations, training environments, and multi-player games.

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 approach 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. We further extend our method to robustly address the case where observations may be unreliable due to low-level sensor noise or recognition errors.

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; hence 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 changes through time as agents join and leave the team. However, each trace corresponds to the execution of some plan in the library.

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 through each plan. 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 (perceptual aliasing).

Multi-agent Plan Representation

Our work employs a multi-agent extension of the hierarchical task network plan libraries commonly 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. 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. Figure 1 shows a small example plan 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 special node types, **SPLIT** and **RECRUIT**, to represent the splitting and merging of agent teams. A **SPLIT** node denotes that the following portion of the plan can be decomposed into parallel subtasks, each of which is handled by its own subteam. The node specifies the composition of each subteam and their tasks (which are simply plan trees). Any agents not allocated to a subteam will continue to execute the original plan until released. 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 (released from other tasks) become available.

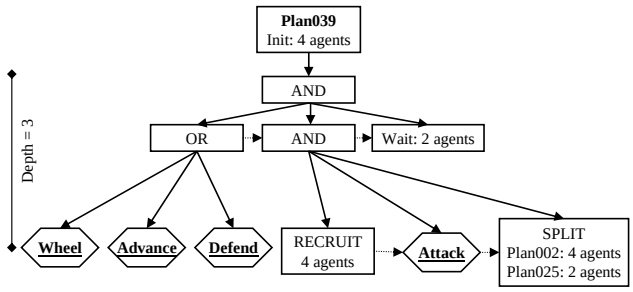


Figure 1: Example plan tree. The top node lists the plan library index and the number of agents required (4) to start execution of this team plan. Hexagonal nodes denote directly observable behaviors; square nodes are effectively invisible to an external observer and must be indirectly inferred. The **RECRUIT** node indicates when additional agents are needed to continue plan execution. The **SPLIT** node denotes where the plan requires multiple subteams to execute subplans (002 and 025) in parallel. At the end of the plan, any remaining agents are released for recruitment by new plans.

SPLIT and **RECRUIT** are not directly observable actions and must be inferred from changing team sizes in observable leaf nodes. Since different observed actions can vary in duration, we do not assume strong synchronization across plans based on atomic action duration.

Kaminka & Bowling (2002) developed the concept of *team coherence*, the ratio of total agents to the number of active plans, to represent the possibility of team coordination failures; they demonstrate that plan recognition can be used as part of a scalable disagreement-detection system to detect the existence of incoherent team plans. Here, we represent such teamwork failures as plan abandonment; if the agents reconcile their differences and resume coordination, it is detected as a new plan instance, rather than a continuation of a previous team plan.

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 the observed actions of 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 could 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.[1]

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 hash, 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. Intuitively, h is needed because the formation of a subteam (i.e., **SPLIT**) is an invisible event; one can indirectly hypothesize the existence of a split only by noting changes in agent behavior. The presence of a **SPLIT** node can also be detected by observing a drop in team size in the parent trace. Specifically, h captures relationships between pairs of plans of the form that an observable behavior in the first plan can be followed by an observable behavior in the second plan (i.e., a subset of agents executing the first plan can **SPLIT** off to execute the second plan). Given a pair of observations, h enables us to identify the set of candidate plans that qualify as subplans for the identified parent plan. This allows us to significantly restrict the plan library for the child trace. Figure 2 illustrates the construction of h for a highly-simplified plan library consisting of two plan trees.

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 hash, 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. Figure 3 illustrates a simple example with a plan library consisting of two plan trees. Some observable sequences could only have been legally generated by one of those two trees (e.g., **C**,**A**), while others are ambiguous (e.g., **A**,**B**).

The size of these hash 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.

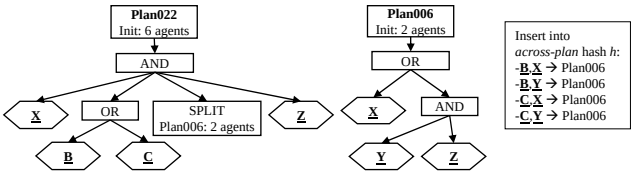


Figure 2: Example of across-plan relationships captured by hash h. h captures observable behaviors across a team split. In this case, the **SPLIT** node in the parent plan (022) could be preceded by observation **B** or **C**, while the first step in the subplan (006) will generate either **X** or **Y**. Therefore, h will contain four entries, all pointing to Plan006. Observing one of these four sequences is an indication that the system should consider the possibility of a **SPLIT**.

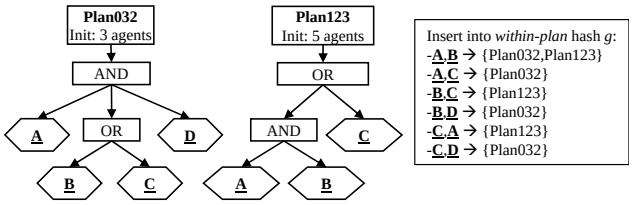


Figure 3: Example of within-plan relationships captured by hash g. g captures all plans where two observations can be observed in sequence. For instance, the observed sequence of three agents executing **A**,**B** could be generated by either of the plan trees whereas **A**,**C** could only be the result of Plan032. These temporal constraints can significantly prune the set of possible plan hypotheses.

Analyzing Scenarios with Dynamic Team Membership

The techniques described here rely on the availability of agent team assignments for multi-agent behavior traces. Recovering this information can be challenging in scenarios with dynamic team assignment, during which the composition of the team changes over the course of the plan. The pool of possible *agent-to-team assignments* grows very quickly with the number of agents and is equivalent to the number of partitions of a set.

Fortunately, for many applications involving physically-embodied agents it is possible to robustly infer team membership from spatio-temporal data. In cases where the agent teams are physically well-separated, clustering can be used

[1]For this constraint to hold if plan abandonment is possible, we must assume that abandonment cannot occur *during* subteam formation—it either occurs before subteam formation or after the execution of the subteam's initial observed behavior.

to recover team assignments. In more complicated scenarios, one can use algorithms such as STABR (Sukthankar & Sycara 2006). In many cases, one can assume that all the agents concurrently executing the same behavior are part of the same team as is done in (Avrahami-Zilberbrand & Kaminka 2007).

Robustness to Observation Noise

In some simulation environments, one can collect highly-accurate low-level behavior traces from multiple agents and humans acting in the virtual world. However most real-world activity recognition systems that monitor the activity of humans using cameras (Nguyen *et al.* 2005), GPS readings (Liao, Fox, & Kautz 2004), or wireless signal strength measurements (Yin, Chai, & Yang 2004), report error rates ranging from 5%–40% in accurately classifying behaviors from position data. These error rates pose a challenge for our algorithm since we rely on the existence of temporal dependencies between behavior observations, across and within traces, to prune the plan library. If these dependencies were corrupted by observation noise, then the pruning algorithm as described above could incorrectly prune correct plans because the noisy observation traces might contain observed transitions that would be "illegal" according to the correct plan. On the other hand, observation failures resulting in fewer behavior transitions being recorded would not adversely affect pruning accuracy since the absence of transitions cannot trigger the deletion of a plan from the hypothesis set.

To address this challenge, we extend the approach described above by shifting the focus from *pruning* to *prioritization*. Rather than eliminating from consideration those plans that could not legally generate the observed behavior transitions, we order plans based on their likelihood of generating the observed sequences. This likelihood is estimated according to the same criteria employed for pruning—temporal dependencies between observations, both within and across traces. We pre-process the plan library in the same manner, to construct the hashes g (within-trace constraints) and h (across-trace constraints). However, these hashes are employed in a different manner against the observed data. For pruning, the hashes were used to delete plans from the hypothesis set; here they are used to augment the likelihoods of plans that are consistent with the given observation. By assuming conditional independence of observed transitions, we can approximate the log-likelihood of matching a given observation to a particular plan as the sum of independent contributions from each transition. In the absence of additional information from the low-level recognizer, we can treat these contributions as equal. This leads to the following approach for plan ordering. For each observed trace, we accumulate a score that is a linear combination of contributions from observations that are consistent with g and h. The plan library is sorted according to this score (this ordering is specific to each trace), and the behavior recognizer is applied to the plans from most promising to least promising until a suitable match is found.

As with the pruning method, the prioritization approach is agnostic to the choice of behavior recognizer. Although all of the plans in the library can be sent to the recognizer for detailed analysis, in practice we apply the recognizer only to the most promising plans (i.e., the top 10%). This decision is supported by the experimental results shown below.

Table 1: Default plan generation parameters

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

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 prescribed by 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**.

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 simultaneously drawing from the limited pool of agents.

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. All of the strategies employed the same depth-first search with backtracking to match execution traces against plan hypotheses.

On average, the across-trace (h) and within-trace (g) hashes 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 here are complete, it is unremarkable 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, we do not claim better precision in general over baseline methods since these cases are infrequent; rather, the primary focus of this paper is to present a more efficient scheme for team plan recognition that exploits inter-plan constraints.

We perform a set of experiments to evaluate the efficiency of three approaches to team plan recognition:

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

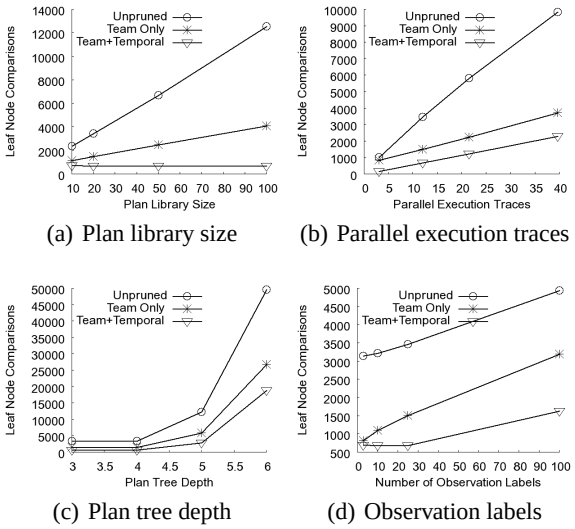
Team Only: prune plan libraries for each observation trace using across-trace dependencies from h before running depth-first matching.

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

Figure 4(a) shows how plan recognition time (as measured by the number of leaf node comparisons) scales with growth in library size (number of plan trees). We see that the Unpruned and Team Only approaches scale approximately linearly with library size while the cost for combined Team+Temporal pruning remains almost constant. This is because the set of plan trees that could explain a given *set* of observed traces remains small.

Figure 4(b) examines how the performance of the three methods scales with the number of observed execution traces. It is unsurprising that the time for all of the methods grows linearly. However, pruning significantly reduces cost. In this case, Team+Temporal achieves a consistent but less impressive improvement over Team Only. We see that the pruning strategies enable us to run plan recognition on much larger scenarios.

Figure 4(c) presents the cost of plan recognition against the average depth size of plan trees in the library. Since the number of nodes in a plan tree increases exponentially with depth, we expect to see a similar curve for each of the three approaches. However, we do see a dramatic reduction in cost due to pruning.



(a) Plan library size (b) Parallel execution traces

(c) Plan tree depth (d) Observation labels

Figure 4: Cost of plan recognition, as measured by leaf node comparisons, for different pruning strategies under varying conditions: (a) size of plan library; (b) average number of plans executing in parallel; (c) average depth of plan tree; (d) number of observable behaviors. Pruning using h and g enables dramatic improvements for large plan libraries.

Figure 4(d) shows how increasing the number of distinctly-recognizable low-level behaviors (number of observation labels) impacts the cost of team plan recognition. As the number of potential labels grows, it becomes easier to disambiguate sequences of observed actions. Consequently, the benefits of pruning within-trace (using hash g) become increasingly important. This is evident in our results, where Team+Temporal pruning shows clear benefits.

Robustness to Observation Noise

To evaluate the efficacy of our prioritization method, we examine the robustness of the ranking with respect to observation noise. These experiments were conducted with a library with 100 plans (average depth 4). The observation traces were generated as above and then corrupted by iid noise (conditions ranging from 0% to 50% probability of misidentification). A corrupted observation was replaced by a random observation drawn with uniform probability from the set of 10 observable actions.

The observed transitions were used to generate likelihood estimates for each of the 100 plans. The rank of the correct plan (known from ground truth) serves as a measure of the quality of the prioritization. Ideally, one would like the correct plan to be at rank 1; in practice, we would be satisfied if the correct plan appears reliably in the top 10%, since this gives us an order of magnitude improvement over a brute-force matching approach.

Figure 5 summarizes the average results from 100 independent trials for prioritization over a range of noise conditions. We make several observations. First, we note that the prioritization is very effective at scoring the correct plan within the first few ranks (average rank is only 5.2 out of

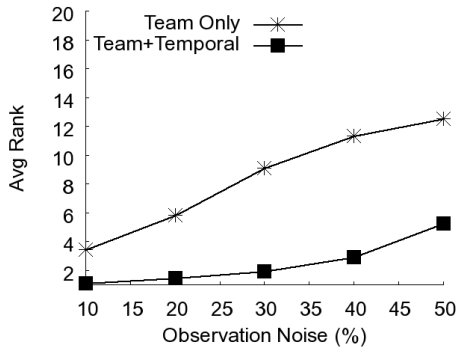


Figure 5: Average rank of correct plan in conditions of increasing observation noise. The prioritization scheme is effective at ordering plans such that the correct one is within the top 10%.

100 even in extremely noisy conditions). The standard deviations for these results ranged from 1.2 (for 10% noise) to 12.4 (for 50% noise). Thus, in moderately noisy conditions, it is reasonable to expect that the correct plan will fall within the top 10%. Second, we can see that although the across-team constraints alone are fairly effective at ordering the plan library, one can achieve significant improvements by also incorporating within-trace information. This is particularly valuable in high-noise conditions where the chance of corrupting a key observation spanning sub-team formation is non-negligible. Finally, we note that these experiments exploited no additional domain knowledge, such as better sensor models (e.g., confusion matrices for which observations are likely to appear similar) nor indications about which observations might be outliers based on higher-level plan knowledge. These additional sources of domain information can complement our prioritization strategy and further improve performance. This validates our belief that a prioritization-based strategy could significantly improve the efficiency of multi-agent team behavior recognition.

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. Our approach identifies characteristics of the plan library that *compress* the number of potential explanations. The benefits of implementing this as an automatic preprocessing step include the following:

1. By automatically recovering this hidden structure, we remove some of the burden of plan library authorship from the user.
2. Pruning and prioritization of the plan library works with a variety of plan recognition algorithms.
3. Prioritization of plans improves efficiency of plan recognition in the presence of observation noise.

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 extend our pruning approach to robustly handle scenarios with significant observation noise by generating an ordering over the plans in the library. An effective estimation of a given plan's likelihood of generating a particular observation trace enables the correct plan to reliably appear within the top 10%, allowing efficient recognition. We are currently applying this method to activity recognition for physically-embodied agent teams, such as squads of military operations in urban terrain (MOUT).

Acknowledgments

This work was supported by AFOSR grant F49620-01-1-0542, as well as the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-06-3-0001.

References

Avrahami-Zilberbrand, D., and Kaminka, G. 2005. Fast and complete symbolic plan recognition. In *Proceedings of IJCAI*.

Avrahami-Zilberbrand, D., and Kaminka, G. 2007. Towards dynamic tracking of multi-agents teams: An initial report. In *Proceedings of Workshop on Plan, Activity, and Intent Recognition (PAIR 2007)*.

Erol, K.; Hendler, J.; and Nau, D. 1994. HTN planning: Complexity and expressivity. In *Proceedings of AAAI*.

Geib, C. 2004. Assessing the complexity of plan recognition. In *Proceedings of AAAI*.

Intille, S., and Bobick, A. 1999. A framework for recognizing multi-agent action from visual evidence. In *Proceedings of AAAI*.

Kaminka, G., and Bowling, M. 2002. Towards robust teams with many agents. In *Proceedings of AAMAS*.

Liao, L.; Fox, D.; and Kautz, H. 2004. Learning and inferring transportation routines. In *Proceedings of AAAI*.

Nguyen, N.; Phun, D.; Venkatesh, S.; and Bui, H. 2005. Learning and detecting activities from movement trajectories using Hierarchical Hidden Markov Models. In *Proceedings of CVPR*.

Sukthankar, G., and Sycara, K. 2006. Simultaneous team assignment and behavior recognition from spatio-temporal agent traces. In *Proceedings of AAAI*.

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

Yin, J.; Chai, X.; and Yang, Q. 2004. High-level goal recognition in a wireless LAN. In *Proceedings of AAAI*.