Towards Dynamic Tracking of Multi-Agents Teams: An Initial Report

Dorit Avrahami-Zilberbrand and **Gal A. Kaminka**
The MAVERICK Group
Computer Science Department
Bar Ilan University, Israel
{avrahad1,galk}@cs.biu.ac.il

Abstract

This paper takes first steps to address the challenge of plan recognition for dynamic multi-agents teams, in the context of suspicious behavior recognition. Plan recognition is the process of inferring other agents' plans and goals based on their observable actions. Team plan recognition poses the challenge of such inference, of a team's joint goals and plans. Most previous work have focused on recognizing specific (and limited) coordinated behaviors and do not deal with the problem of identifying interactions between groups of agents, and with identifying suspicious behavior from this information. In contrast, this paper utilizes the information from group of agents, to identify the interactions between groups of agents, using a Dynamic Hierarchical Group Model (DHGM) that tracks the dynamic grouping and ungrouping of agents. We show how such information can be used to identify potential suspicious behavior. These suspicious behaviors can be captured only when tracking individuals with respect to the group and not as individuals. For example, identifying passenger in the airport that behaves differently from other passengers in the same group. While reasoning about individual agents in a multi-agents framework is expensive, we reduce this complexity by utilizing the DHGM that encapsulate shared data of agents in the same group.

Introduction

Plan recognition (Charniak & Goldman 1993; Duong *et al.* 2005; Geib 2004) focuses on mechanisms for recognizing the unobservable state of an agent, given observations of its interaction with its environment. Most approaches to plan recognition utilize a plan library, which encodes the behavioral repertoire of observed agents. Observations are matched against this plan library in sequence.

Although multiple frameworks have been developed for single-agent plan recognition, there has been less work on extending these frameworks to multi-agent scenarios. Plan recognition with multi-agents can be performed, in principle, by treating agents as independent, and recognizing the plan of each agent separately. However, there are number of problems with this method in complex multi-agents scenarios. First, some scenarios require agents to participate in dynamic teams where team membership changes over time. Tracking individual agents independently fails to recognize a team joint goal and activities (Tambe 1997); it will thus fail to capture recognizing the behavior of agents with respect to their group.

Previous work has shown that given a model of hierarchical relationship between agents, one can recognize team plans, involving multiple agents (Intille & Bobick 1999; Kaminka, Pynadath, & Tambe 2002; Kaminka & Bowling 2002). However, these previous work focused specific social structures, where agents form teams based on a-priori agreements as to specific plans. In order to recognize team plans in these previous methods, the monitoring agent must first know which plans are ideally to be agreed upon. In contrast, in our work we do not have static social structure that is given in advanced, but instead use the plan library to identify dynamically changing structure of the groups. For example, a group of passengers in the airport may seem like one group when standing in the security check line, and afterward when splitting to two groups, the organizational structure need to be modified.

We propose a method for tracking the dynamically changed structures of groups of agents. This information can be used in several ways. First, identifying an agent that behave differently from other agents in the same group (which can serve as a basis for recognizing suspicious behavior). Second, we can understand better the agent actions by saving the history of its group.

For example consider the *queue-cutting problem*, where two friends are standing in specific position in the security line. One of them goes to the restrooms. When she returns, she joins her friend in the security line, rather than the end of the line. If we would not save the history of her group, we may consider this person to be a suspect of cutting the line. However, when knowing the history of the group, we can understand better its actions.

This paper proposes initial steps towards a method for tracking groups and changes in these groups (merging and splitting) by saving information on the common plan that each group executes. To do this, we would use a Dynamic Hierarchical Group Model (DHGM) that indicates the connection between agents. This will allow us to know the agent group history, and to reduce complexity by saving for each sub group the same plan library. We provide an analysis and suggest directions for future work.

Related Work and Motivation

There has been considerable research into plan recognition algorithms. Here we only address those efforts that are closely related.

YOYO (Kaminka & Bowling 2002) is a symbolic approach for detecting disagreements among team members. This work exploits knowledge on the social structures of the team (called team hierarchy) to efficiently recognize splits in teams, where an agent is executing a different plan than the rest of its team. In order to detect disagreements, the monitoring agent must first know which plans are ideally to be agreed upon. In contrast, in our work we do not have static social structure that is given in advanced, but dynamically track structure of groups which changes over the time. However, while we track splits in the team, we cannot categorically determine that a split is an anomaly.

$RESC_{team}$ (Tambe 1996) is a symbolic multi-agent plan recognition scheme which represents only one coherent hypothesis. $RESC_{team}$ can reason about the assignment of agents to sub-teams, meaning that it does not require a static team hierarchy as YOYO. However, it still uses information about the plans expected to be agreed-upon.

Intille and Bobick (Intille & Bobick 1999) rely on coordination constraints among football players to recognize team-tactics. They thus similarly focus on the interactions between agents, rather than each agent as an individual. However, they do not address dynamic grouping and ungrouping: the recognized interactions are used to recognize the multi-agent plans that contain them.

(Hongeng & Nevatia 2001) recognizing multi agent events observed by static camera. Multi agent event is represented by number of action threads, where each thread executed by single actor. These action threads related by temporal constraints generating a multi-agents event graph. They thus similarly track dynamic interactions between agents, but they are restricted to specific constrains between agents that are defined in the network. Moreover, they can not detect dynamic splitting and merging of groups.

STABR (Sukthankar & Sycara 2006) is a Team Assignment and Behavior Recognition model , recovering agent-to-team assignments where the mapping of agents into teams, changes over time. This paper thus also addresses the problem of behavior recognition for teams with dynamic team composition. However, this approach is based on matching agent positions to pre-specified geometric formation templates. In contrast, in our work we do not have static information about groups, but detect dynamic splitting and merging of groups.

Detecting Multi Agents Dynamic Groups

This section introduces *Dynamic Hierarchy Group Model* (DHGM), a dynamically-maintained structure for tracking groups and sub-groups when the groups split into sub-groups and/or merge with other groups. We use a plan recognizer with each agent. For this purpose, we use here a highly efficient symbolic plan recognizer (SBR) (Avrahami-Zilberbrand & Kaminka 2005) that is used to filter through hypotheses, maintaining only those that are consistent with the observations. Note, however, that any plan recognizer can be used. Then, we introduce DHGM and provide its complexity analysis.

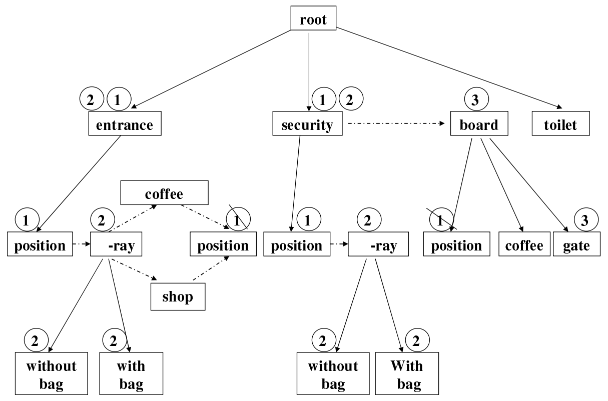

Figure 1: **An example of plan library.**

Efficient Symbolic Plan Recognition (SBR) : The Basics

We exploit SBR, a highly-efficient symbolic plan recognizer, briefly described below. The reader is referred to (Avrahami-Zilberbrand & Kaminka 2005) for details.

SBR's plan library is a single-root directed graph, where vertices denote *plan steps*, and edges can be of two types: Decomposition edges decompose plan steps into sub-steps, and sequential edges specify the temporal order of execution. The graph is acyclic along decomposition transitions.

Each plan has an associated set of conditions on observable features of the agent and its actions. When these conditions hold, the observations are said to match the plan. At any given time, the observed agent is assumed to be executing a *plan decomposition path*, root-to-leaf through decomposition edges. An observed agent is assumed to change its internal state in two ways. First, it may follow a sequential edge to the next plan step. Second, it may reactively interrupt plan execution at any time, and select a new (first) plan. Figure 1 shows an example portion of a plan library.

The recognizer operates as follow: First, it matches observations to specific plan steps in the library according to the plan step's conditions. Then, after matching plan steps are found, they are tagged by the time-stamp of the observation. These tags are then propagated up the plan library, so that complete plan-paths (root to leaf) are tagged to indicate they constitute hypotheses as to the internal state of the observed agent when the observations were made. The propagation process tags paths along decomposition edges. However, the propagation process is not a simple matter of following from child to parent. A plan may match the current observation, yet be *temporally inconsistent*, when a history of observations is considered. SBR is able to quickly determine the temporal consistency of a hypothesized recognized plan (Avrahami-Zilberbrand & Kaminka 2005).

A plan is temporally consistent in time stamp t if one of three cases holds: (a) the node in question was tagged at time $t-1$ (i.e., it is continuing in a self-cycle); or (b) the node follows a sequential edge from a plan that was successfully tagged at time $t-1$; or (c) the node is a first child (there is no sequential edge leading into it). A first child may be selected at any time (e.g., if another plan was interrupted). If neither of these cases is applicable, then the node is not part of a temporally-consistent hypothesis, and its tag should be deleted, along with all tags that it has generated in climbing up the graph. The tags made on the plan-library are used to save information from one run to the next.

Figure 1 shows the process in action (the circled numbers in the figure denote the time-stamps). Assume that the matching algorithm matches at time $t = 1$ the multiple instances of the $position$ plan. At time $t = 1$, Propagate begins with the four $position$ instances. It immediately fails to tag the instance that follows $coffee$ and $shop$, since these were not tagged at $t = 0$. The $position$ instance under $board$ is initially tagged, but in propagating the tag up, the parent $board$ fails, because it follows $security$, and $security$ is not tagged $t = 0$. Therefore, all tags $t = 1$ will be removed from $board$ and its child $position$. The two remaining instances successfully tag up and down, and result in possible hypotheses $root \rightarrow entrance \rightarrow position$ and $root \rightarrow security \rightarrow position$.

At the end of the SBR process we are left with a set of *current-state hypotheses*, i.e., a set of paths through the hierarchy, that the observed agent may have executed at the time of the last observation. The overall worst-case run-time complexity of this process is $O(lD)$ (Avrahami-Zilberbrand & Kaminka 2005). Here, l is the number of plan-steps that directly match the observations; D is depth of a degenerate plan-library (i.e., a linked list). Extensions to this model address interleaved plans and limits on durations (Avrahami-Zilberbrand, Kaminka, & Zarosim 2005).

Dynamic Hierarchy Group Model

After getting all *current state hypotheses* from the symbolic recognizer, the next step is to determine the agent's group. This is done by using the *Dynamic Hierarchy Group Model* (DHGM), described in this section.

The DHGM is a dynamically-maintained structure that reflects the current groups of the agents, and the history of these groups. This structure is built dynamically with every observation. Each node in the DHGM represents a group with one or more agents that are executing the same behavior (i.e,. the plan recognizer identified the same plan-steps in the plan library for this group). Each node in the DHGM points to leaves in the plan library that the agent is assumed to be executing. Each node also holds a time-stamp counter that counts the number of consecutive time-stamps that the agent the group were in this branch. From each node there are branches to sub-groups, meaning that the group had been split and executing now different plan-steps in the plan library. When the agent returns to its group, or join other sub-group, the branches are merged.

Figure 2 shows portion of a DHGM (noted with G), that points to plan library P. On top there is an array of agents A, that points to their place in the Group Hierarchy. In this figure, agents 1,2 and 4 belong to one sub-group and agents 3,5 belongs to a second sub-group.

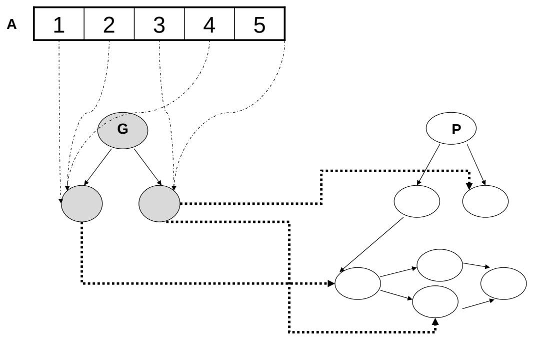

Figure 2: **Example Dynamic Hierarchy Group Model noted with G.**

The DHGM maintenance process is described in Algorithms 1–2. The GroupDetection algorithm (Algorithm 1) initializes the DHGM with single root node that all agents belongs to this node, and it points to the given plan library. With each new observation it calls $UpdateGroup$ algorithm (Algorithm 2).

The UpdateGroup algorithm (Algorithm 2), traverses the DHGM bottom up. For each leaf node it executes the SBR algorithm on the current observation for all agents that belong to this leaf (Algorithm 2 line 7), with the appropriate plan library and time-stamp tag. The SBR algorithm returns a set of paths through the hierarchy, that the observed agent may have executed according to the observation. The Algorithm then updates the $temporaryAgentArray$ that holds all SBR results for the current time-stamp (Algorithm 2 line 8). Then, it updates the DHGM according to the $temporaryAgentArray$: it creates a new branch for agents that have the same SBR result and does not have an appropriate branch yet (Algorithm 2 line 9). The algorithm also update the time-stamp counter of the group, to know how long the group exists. If new branches were created, then the time-stamp counter is initialized to 1. If the branches are the same we add one to the counter (Algorithm 2 lines 10–13). Finally, it merges all leaves that have the same results (points to same leaves in the plan library), meaning that after we updated them they have the same results as other branches in the same level.

Algorithm 1 GroupDetection(Plan Library p)

```
1: Create Group Hierarchy G with a single node
2: Initialize G with all agents belongs to its single node
3: Initialize G with pointer to root of plan library p
4: while $Observations \neq Empty$ do
5:     $t \leftarrow t + 1$
6:     $UpdateGroup(root(G), t, Observations)$
```

An example: We will demonstrate the process in action with a simple example shown in Figure 3 and the plan library

Algorithm 2 UpdateGroup(Group Hierarchy Node $groupNode$, Time-stamp t, Observation $obsrvArr$)

```
 1: for all child c that is not a leaf of groupNode do
 2:     $UpdateGroup(c, t, obsrvArr)$
 3: for all child c that is a leaf of groupNode do
 4:     create empty temporaryAgentArray
 5:     for all agent a$\in$ $groupNode$ do
 6:         $p \leftarrow$ plan library that groupNode points on
 7:         $ExecuteSBR(t, obsrvArr[a], p)$
 8:         update temporaryAgentArray[a] to point on new plan-
            steps
 9:     create branches according to the temporaryAgentArray
10:     if if no branches were created then
11:         increase time-stamp counter of this leaf
12:     else
13:         initialize time-stamp counter of this leaf
14:     for all child c that is a leaf$of groupNode$ do
15:         merge nodes if points to same plan steps
16:         initialize time-stamp counter of this leaf
```

in 1. Assume that there are 100 agents in the airport that execute the $position$ plan-step in time-stamp $t = 1$. The DHGM here has one root node with 100 agents and points to all $position$ instances in the plan library (note that for presentation clarity, Figure 3 points to the plan-step name and not to all instances in the plan library).

Now, in time-stamp $t = 2$ there are 50 agents that continue executing the $position$ and 50 other agents execute the X-Ray plan-step, 20 without bags and 30 with bags. The DHGM will have the following structure a root node with 3 leaves: one for position, one for x-Ray with bag and one for X-Ray without bag. Now assume that in time stamp $t = 3$ the 50 agents that were in the security with or without bags, now execute the gate plan-step, and the 50 agents moved from the $position$ to execute the X-Ray with or without bag (25 in each group). Now the DHGM will have under root two nodes, one that points to the gate instances in the plan library and one that splits into 2 nodes: X-ray with bag and X-Ray without bag. In time stamp $t = 4$ all agents in the X-Ray without bag execute the gate plan-step, and under X-Ray with bag 24 agents execute the gate and one agent execute position. The DHGM will have under root two nodes, one that points to the gate instances in the plan library and one that splits into 2 nodes, one node with 25 agents that execute the $gate$ plan-step and another node that splits to two nodes: one with 24 agents that execute $gate$ and 1 agent that executes $position$. Note that time-stamps counters were omitted, for presentation clarity.

Note that the Dynamic Hierarchy Group Model can either hold multiple instances of the plan library (for each group), or one plan library with different time-stamp tags for each group. See also time and space complexity in the next section.

Complexity Analysis

Using the Dynamic Hierarchy Group Model for plan recognition in multi agents scenarios is less expensive in space terms, than operating individual plan recognition method for each agent. We do not need to hold a plan library for each

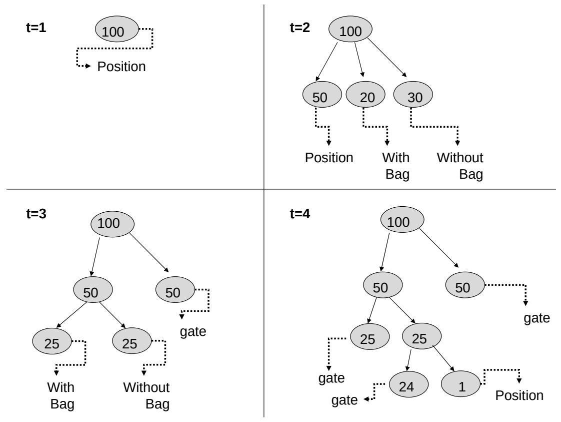

Figure 3: **Example of the process of creating Dynamic Hierarchy Group Model. The Number on the node denotes number of agents belong to this group.**

agent, but to use the same plan library with different notations for each group. Therefore the space complexity is $O(Lg)$, where L is the plan library size and g is the maximum number of groups. This should be compared to $O(Ln)$, where n is the number of agents.

The run-time complexity is larger than running individual plan recognition, since we not only run the SBR algorithm, but also check the connections between agents. we go over the group hierarchy $O(g)$ bottom up, where g is the number of groups. For each node: we execute SBR (Algorithm 2 line 7): $O(LD)$, where L is the plan library size, and D is the depth of the plan library. This is done for all agents in this node $O(LDn)$ (Algorithm 2 line 5), where n is the number of agents in the group. Then, compare all agents results $O(nlogn)$ with sort (Algorithm 2 line 9). Therefore, $O(gnLD + gnlogn)$. The gn (number of groups multiply the size of the group) is actually the total number of agents N. Therefore, it is $O(NLD + Nlogn)$, this is for one observation. It will be $O(TNLD + TNlogn)$, where T is the number of observations.

Detecting Suspicious Behavior

This section presents a method for detecting suspicious behavior using DHGM. There are suspicious behaviors that can be captured only when tracking agents with respect to their group and not only as individuals. For this purpose, we utilize the *Dynamic Hierarchy Group Model* that was introduced in the previous section to detect suspicious behavior.

We differentiate between two types of suspicious behavior recognition models: (i) explicit recognition; and (ii) implicit recognition. In the explicit recognition model, the plan library represents suspicious behavior; any activity that matches the model will be assumed to be suspicious. This is the model used in (Geib & Goldman 2001). It can be challenging to learn plan libraries for explicit recognition, in domains where we have fewer recordings of suspicious behavior. Indeed, in many domains, we normally have more recordings of normal behavior (for instance, in vision-based surveillance in public places). Here, we can utilize implicit recognition, where the plan library represents normal behavior; any activity which is not recognized is abnormal. For example, the normal activity model may include usual movements of people in the airport.

A symbolic plan-recognition algorithm, such as SBR, is useful in implicit recognition of abnormal patterns, such as walking in the wrong direction, taking more then usual amount of time to get to the security check, etc. However, without a way to rank hypotheses (e.g., probabilistically (Geib 2004), or based on expected costs (Avrahami-Zilberbrand & Kaminka 2007)), it may be less useful for explicit recognition.

Recognizing the plans of multiple agents can help in detecting suspicious behavior. There are suspicious behaviors that can be captured only when tracking agents with respect to their group and not only as individuals. We propose a number of heuristics for detecting suspicious behavior, using the information from the DHGM showed in the previous section.

The first heuristic we propose is that agents that behave differently from other agents in the same group will be considered suspicious. For example, passengers that behave differently from other passengers in the airport (not standing with all other passengers in the line).

We consider an agent or group of agents as behaving differently from other agents in the same group, and therefore suspicious, if the following conditions hold: (a) The agent did not return to his group k consecutive time-stamps (where k is a constant that is application-dependent); (b) The rest of the group is m times bigger from the group of suspects (where m is a constant that is application-dependent). These conditions can be extended as needed for specific application and conditions. The process of checking for these conditions given a DHGM is described in Algorithm $CheckSuspiciousBehavior$ (algorithm 3).

Using the DHGM we can identify agents that behave differently from other agents in the same group. For example, lets use the DHGM in the example in 3, we can see from the information on the DHGM that there is only one agent out of 24 that executes the $position$ plan-step, and did not get to the $gate$ like other agents. We would like to keep an open eye on this agent, since she might be dangerous.

Algorithm 3 CheckSuspiciousBehavior(Group Hierarchy Node $groupNode$)

```
1: for all child c1 that is a leaf of groupNode do
2:     for all child c2 that is a leaf of groupNode do
3:         if |number of agents in c1| $<$ $m \times$ |number of agents in c2| then
4:             if number of time-stamps counter in c1 $>$ $k$ then
5:                 c1 group is suspicious
```

Previous work has shown that given a model of hierarchical relationship between agents, one can identify deviations from the model (Kaminka & Bowling 2002) (which indicate anomalies), and may increase efficiency of recognition (Kaminka, Pynadath, & Tambe 2002). However, this previous work is limited to very specific social structures, where agents form teams based on agreements as to specific plans. In order to detect disagreements, the monitoring agent must first know which plans are ideally to be agreed upon. In contrast, in our work we do not have static social structure that is given in advanced, but observations on dynamically changing structure of groups. For example group of passengers in the airport may seem like one group when standing in the security check line, and afterward when splitting to two groups, the structure need to be modified.

The second heuristic we propose is to explicitly clear an agent, if it behaves normally with respect to its group's history. For example consider the *queue-cutting problem*, where two friends are standing in specific position in the security line, when one goes out to the restrooms, and returns to join her friend. If we would not save the history of the group, we may consider this person to be a suspect of cutting in line. However, when knowing the history of the group, we can explicitly clear her.

The *queue-cutting problem* can be solved using the DHGM. In this case the SBR algorithm will return that there are no results for this agent according to the plan library. This will happen since the propagation phase has failed, therefore we can check whether the matching phase has resulted with matching plan-steps and compare them to the agent's history group. If the agent is executing the same plan-step that some of its group members, then we consider it as a legal behavior, and not consider her a suspect as the single agent SBR would do.

Summary and Future Work

This paper presents initial steps towards efficient dynamic tracking of the organization of multi-agent teams. Using a combination of single-agent plan recognizer, and the DHGM data-structure, we maintain book-keeping information which allows us to dynamically track and hypothesize as to the organizational structure of a group of agents. We also discuss two heuristics that use this information to recognize suspicious behavior.

Much remains for future work, of course. We have only begun to address issues of uncertainty, for instance. We would also like to more tightly integrate SBR and DHGM, and more formally define the recognition queries answerable by multi-agent plan recognizers in general, and DHGM in particular.

Acknowledgments. This research was supported in part by ISF grant #1211/04. Special thanks to Nadav Zilberbrand and K.Ushi.

References

Avrahami-Zilberbrand, D., and Kaminka, G. A. 2005. Fast and complete symbolic plan recognition. In *IJCAI-05*.

Avrahami-Zilberbrand, D., and Kaminka, G. A. 2007. Incorporating observer biases in keyhole plan recognition (efficiently!). In *Proceedings of Twenty-Second National Conference on Artificial Intelligence (AAAI-07)*.

Avrahami-Zilberbrand, D.; Kaminka, G. A.; and Zarosim, H. 2005. Fast and complete plan recognition: Allowing for duration, interleaved execution, and lossy observations. In *Proceedings of the MOO-05 Workshop*.

Charniak, E., and Goldman, R. P. 1993. A Bayesian model of plan recognition. *AIJ* 64(1):53–79.

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

Geib, C. W., and Goldman, R. P. 2001. Plan recognition in intrusion detection systems. In *In DARPA Information Survivability Conference and Exposition (DISCEX)*.

Geib, C. W. 2004. Assessing the complexity of plan recognition. In *AAAI-04*.

Hongeng, S., and Nevatia, R. 2001. Multi-agent event recognition. In *ICCV*, 84–93.

Intille, S. S., and Bobick, A. F. 1999. A framework for recognizing multi-agent action from visual evidence. In *AAAI-99*, 518–525.

Kaminka, G. A., and Bowling, M. 2002. Towards robust teams with many agents. In *AAMAS-02*.

Kaminka, G. A.; Pynadath, D. V.; and Tambe, M. 2002. Monitoring teams by overhearing: A multi-agent plan recognition approach. *Journal of Artificial Intelligence Research* 17.

Sukthankar, G., and Sycara, K. 2006. Simultaneous team assignment and behavior recognition from spatio-temporal agent traces. In *Proceedings of Twenty-First National Conference on Artificial Intelligence (AAAI-06)*.

Tambe, M. 1996. Tracking dynamic team activity. In *AAAI-96*.

Tambe, M. 1997. Towards flexible teamwork. *JAIR* 7:83–124.