Activity Recognition for Agent Teams

Gita Reese Sukthankar

July 2007

CMU-RI-07-23

Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

Thesis Committee:
Dr. Katia Sycara (chair)
Dr. Illah Nourbakhsh
Dr. Paul Scerri
Dr. Milind Tambe, University of Southern California

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

Keywords: multi-agent plan recognition, activity recognition, teamwork

Abstract

Proficient teams can accomplish goals that would not otherwise be achievable by groups of uncoordinated individuals. This thesis addresses the problem of analyzing team activities from external observations and prior knowledge of the team's behavior patterns. There are three general classes of recognition cues that are potentially valuable for team activity/plan recognition: (1) spatial relationships between team members and/or physical landmarks that stay fixed over a period of time; (2) temporal dependencies between behaviors in a plan or between actions in a behavior; (3) coordination constraints between agents and the actions that they are performing. This thesis examines how to leverage available spatial, temporal, and coordination cues to perform offline multi-agent activity/plan recognition for teams with dynamic membership.

In physical domains (military, athletic, or robotic), team behaviors often have an observable spatio-temporal structure, defined by the relative physical positions of team members and their relation to static landmarks. We suggest that this structure, along with temporal dependencies and coordination constraints defined by a team plan library, can be exploited to perform behavior recognition on traces of agent activity over time, even in the presence of uninvolved agents. Unlike prior work in team plan recognition where it is assumed that team membership stays constant over time, this thesis addresses the novel problem of recovering *agent-to-team* assignment for team tasks where team composition, the mapping of agents into teams, changes over time; this allows the analysis of more complicated tasks in which agents must periodically divide into subteams.

This thesis makes four main contributions: (1) an efficient and robust technique for formation identification based on spatial relationships; (2) a new algorithm for simultaneously determining team membership and performing behavior recognition on spatio-temporal traces with dynamic team membership; (3) a general pruning technique based on coordination cues that improves the efficiency of plan recognition for dynamic teams; (4) methods for identifying player policies in team games that lack strong spatial, temporal, and coordination dependencies.

Acknowledgments

Friendship is very important. China has a proverb. "One chopstick is easily broken, but a dozen can hardly be bent."
–Matthew Polly, American Shaolin

First and foremost, I have to thank my parents, Sudha and William Reese, for inspiring me with their intelligence, idealism and dedication. My Princeton friends—Stan Yamane, Matt Mullin, Erich Greene, and Dan Bernstein—have known me since my "technopeasant" days. I treasure all the time that I've spent playing Dungeons and Dragons with them; they taught me the meaning of teamwork and in many ways contributed to my research thinking.

At Carnegie Mellon, there have been many people who have made my time at the university special. Due to my three year leave of absence at HP Labs, most of the women who entered the Ph.D. program with me—Ashley Stroupe, Bernardine Dias, Joelle Pineau, Rosemary Emery-Montemerlo, and Vandi Verma—have been gone for some years, but I fondly remember all the great potlucks we had together during my first two years in the program. Working with Women@SCS, in general, and Carol Frieze, in particular, has been a wonderful part of my CMU experience.

Through the years, the RETSINA group has hosted many excellent researchers (too numerous to list) who have helped hone my research ideas. My office mate, Mary Berna-Koes, was a great source of support throughout the program. I've also been blessed with three talented research collaborators, Joseph Giampapa, Michael Mandel, and Reid van Lehn, who contributed to parts of the work described in this document. I am grateful to my advisor, Katia Sycara, for guiding me on my thesis journey and to my committee, Paul Scerri, Illah Nourbakhsh, and Milind Tambe, for their valuable insights. Mike Lewis provided useful feedback on the presentation of this material.

In addition to having a world-class computer science program, Carnegie Mellon boasts some excellent martial arts training opportunities. This June, I received my black belt in Shotokan karate. I am grateful to all the people in the Pittsburgh Shotokan Karate club—my instructors and fellow students—for helping me in my journey as a martial artist. But the person who has indisputably made the most difference in my life is my husband, Rahul Sukthankar, who is my best friend, my karate sparring partner, and my research inspiration. Without him, this thesis would not be possible.

Contents

Chapter 1

Introduction

> *Teamwork is the ability to work together towards a common vision. The ability to direct individual accomplishments towards organizational objectives. It is the fuel that allows common people to attain uncommon results.*
> *-Andrew Carnegie*

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.

This thesis addresses the problem of analyzing team activities from external observations and prior knowledge of the team's behavior patterns. To analyze the performance of these tasks, we need to extend existing single-entity activity recognition algorithms to accommodate team behaviors, regardless of whether the observed entities are software

Figure 1.1: Applications for team activity recognition include: (a) Team MOUT training environments; (b) Sports commentator agents such as ISAAC (ISI Soccer Automated Assistant Coach) [74]; (c) Visual monitoring systems such as VSAM (Visual Surveillance and Monitoring) [25].

agents, humans, or even robots. Embedding activity recognition algorithms into multi-player virtual environments will enable:

- the development of smarter opponents that can recognize and respond to the tactics of human teams;
- the creation of better synthetic training partners that can predict their teammates' intentions and coordinate without unnecessary communication;
- automated annotation of multi-player game logs;
- intelligent user correction in team training environments.

Figure 1 shows some examples of applications of that could benefit from the inclusion of team behavior recognition: team training environments (Figure 1a), sports commentator agents (Figure 1b), and visual surveillance systems (Figure 1c). The objective of our teamwork analysis is to be able to identify the team's goal, the actions the team members are taking to achieve the goal, and the coordination patterns between team members. Specifically we want to be able to answer queries of the following type:

1. find all examples of a given team formation;
2. determine which players are cooperating;
3. identify the policy that a player is executing;
4. recognize which plan was used by the winning team.

1.1 Problem Formulation

This thesis addresses the general AI problem of keyhole plan recognition for multi-agent systems. In *intended* plan recognition, it is assumed that the agents structure their behavior to make their intentions clear to the observer, whereas in *keyhole* plan recognition the algorithm "eavesdrops" on the agents' actions [24].

We use the term **agent** to denote an independent entity capable of physical motion and action, such as humans, simulated entities in a virtual environment, or robots. A **team** is defined as "a set of agents having a shared objective and a shared mental state" [23]. **Subteams** are created when a subset of the team coordinates to independently pursue a separate objective; members of a subteam can rejoin their original team after finishing their objective or band with other agents to form different teams. Observations of the agents are in the form of **spatio-temporal traces**, a time series of the agents' physical positions and **actions**. Although the agents (e.g., human game players) can communicate verbally, their communications are not monitored nor included as part of the observed trace (see Appendix A for a preliminary analysis on the role of communication in human teams).

From a complete set of observations over all the agents, we recognize the following team characteristics:

formation: a static layout describing the relative positions between the agents and physical landmarks.

team behavior: a short, observable segment of coordinated movement and action executed by an agent team. Unlike actions, team behaviors are (1) non-atomic and (2) involve multiple agents.

plan: an ordered sequence of team behaviors describing a recipe used by an agent team to achieve a goal. During the course of a single plan, the original team can split or merge into different subteams to execute behaviors. Plans can be abandoned during execution, if necessary.

This thesis assumes that *a priori* domain knowledge is specified in the form of a **library** containing the set of possible formations, team behaviors, and plans; automatically learning these models from data is beyond the scope of this work.

In the literature, the term, *activity* recognition, is often used to describe the problem of segmenting and classifying low-level data gathered by cameras or wireless sensors into a description of the activity performed (e.g., walking or waving); this is analogous to the process of behavior recognition described in this thesis, during which spatio-temporal traces are mapped to sequences of behaviors. *Plan* recognition [2] is generally used to describe the mapping of sequences of atomic actions to high-level plans, whereas in this thesis the observed behavior sequences occur in a parallel, non-synchronous fashion.

1.2 Challenges

Multi-agent activity/plan recognition is a challenging problem for several reasons. First, multi-agent behaviors often exhibit greater variation in the observed spatio-temporal

traces than single-agent behaviors. For instance, when two soldiers execute the bounding overwatch team behavior, one legal variation is that soldier A moves ahead of soldier B; the other is that soldier B is the first to move. Checking all possible legal permutations of the behavior to identify the match can be time-consuming. In the best case, certain properties of the spatio-temporal trace (e.g., motion of the team centroid) are invariant over all possible legal executions of the behavior. Second, in dynamic domains, the composition of the team can change over time as teams split into subteams, subteams merge with one another, and individual agents join or leave teams. In a scenario where there are uninvolved agents, it can be difficult to determine which agents are engaged in team behaviors without prior knowledge of the agents' intentions (consider a convoy or funeral procession driving on a crowded city street). In these cases, it is impossible to treat the team observation as a single high-dimensional vector of concatenated single-agent observations, and identifying which agents share the team objective becomes a problem in its own right. Third, when agents split into subteams, each subteam creates a parallel execution trace that has to be analyzed separately; synchronizing the timestamps between behaviors can be a challenge when the behaviors have variable durations.

However, in one aspect, keyhole team plan recognition is easier than single-agent plan recognition since it shares some characteristics with intended plan recognition. Although the agents are not attempting to communicate their intentions to the recognizer, to coordinate with their fellow team members, they often "telegraph" their future actions through predictable behavior to facilitate coordination. In that sense, the observable component of their actions can be more revealing than that of a single independent agent.

1.3 Approach

There are three general classes of recognition cues that are potentially valuable for multi-agent activity/plan recognition:

- **spatial** relationships between team members and/or physical landmarks that remain fixed over a period of time;
- **temporal** dependencies between behaviors in a plan or between actions in a team behavior;
- **coordination** constraints between agents and the actions that they are performing.

Thesis Question: How can we leverage available spatial, temporal, and coordination cues to perform multi-agent activity/plan recognition for teams with dynamic membership?

The approach in this thesis is to explore the use of spatial, temporal, and coordination cues to perform offline multi-agent plan/activity recognition in physical domains. Although there has been recent progress in robustly tracking multiple moving objects in video data [111], in this thesis we focus on analyzing team behaviors in simulations and games, due to the relative ease of simultaneously acquiring unambiguous location data for multiple entities.

In physical domains (military, athletic, or robotic), team behaviors often have an observable spatio-temporal structure, defined by the relative physical positions of team members and their relation to static landmarks; we suggest that this structure, along with temporal dependencies and coordination constraints defined by a team plan library, can be exploited to perform behavior recognition on traces of agent activity over time, even in the presence of uninvolved agents.

Unlike prior work in team plan recognition where it is assumed that team membership stays constant over time, this thesis addresses the novel problem of recovering *agent-to-team* assignment for team tasks where team composition, the mapping of agents into teams, changes over time; this allows the analysis of more complicated tasks in which agents must periodically divide into subteams.

For this thesis, we examine the problem of recognizing activities and plans within a military setting . We analyze four types of data: (1) agent plans and activities in a MOUT (Military Operations in Urban Terrain) battlefield simulation; (2) teams of human players reproducing military maneuvers in a computer game environment (Unreal Tournament); (3) teams of human players playing tactical battlefield scenarios using a turn-based table-top game (Dungeons and Dragons); (4) human subjects performing physical movements in a motion capture lab.

1.4 Contributions

This thesis makes the following contributions:

- an efficient and robust technique for formation identification based on spatial relationships (Chapter 3);
- a new algorithm for simultaneously determining team membership and performing behavior recognition on spatio-temporal traces with dynamic team membership (Chapter 4);
- a general pruning technique based on coordination cues that improves the efficiency of plan recognition for dynamic teams (Chapter 5);
- an approach for identifying player policies in team games that lack strong spatial,

Table 1.1: Summary of data requirements for the algorithms discussed in the thesis

Method	**Input Data**	**Library**	**Chapter**
formation identification	single snapshot of positions	spatial templates	3
spatially-invariant HMMs	time series of positions	HMM models	3
STABR	time series of positions	spatial templates	4
multi-agent plan recognition	time series of actions	HTN plans	5
policy recognition (Dempster-Shafer)	time series of actions	m-values	6
policy recognition (SVM)	time series of actions	simulated game data	6
physical capability model	single snapshot of positions	cost maps	7
human activity recognition	motion capture data	behavior graphs	7

temporal, and coordination dependencies (Chapter 6).

- a physical capability model for human agents based on motion capture data that enables online prediction of opponent movements (Chapter 7)

Earlier versions of this research have appeared in several conference publications [90–95]. In the next chapter, we provide an overview of related work on activity/plan recognition.

Chapter 2

Related Work

In his seminal work on plan recognition, Kautz defined his framework as a process for determining "which conclusions are absolutely justified on the basis of the observations, the recognizer's knowledge, and a number of explicit *closed-world* assumptions" [2]. In general, this union of observations, prior knowledge, and closed-world assumptions characterizes the research efforts on plan and activity recognition. This chapter discusses the related work on general plan recognition, activity recognition, military domains, and multi-agent plan/activity recognition.

2.1 Plan Recognition

Much of the early work on plan recognition relied on logical methods, either viewing plan recognition as a specialized type of hypothetical [21] or unsound reasoning [1]. However, Kautz's event hierarchy framework [54] combined deductive reasoning with a specific set of assumptions. Two important assumptions that he introduced are exhaustiveness assumption and minimum cardinality. The exhaustiveness assumption specifies that the

world is limited to the known types of events (plans), hence it is possible to determine that a particular event has taken place by eliminating all other possibilities. The minimum cardinality assumption follows the principle of parsimony, only assuming the minimum number of events that explain the observations. These two assumptions are either explicitly or implicitly made by most current plan recognition frameworks and are also important in this thesis.

By introducing probabilistic reasoning techniques, plan recognition becomes a process of determining which plans are *likely* rather than which conclusions are *justified*. Charniak and Goldman [20] developed the first probabilistic model of plan recognition, using Bayesian belief nets for plan inference. Like Kautz's model, Charniak and Goldman's plan language relied on hierarchical action descriptions and did not support sequences of actions; later Huber et al. [46] demonstrated a method for translating general acyclic plan specifications, including action sequence dependencies, into belief nets, Alternatives to Bayesian inference, such as Dempster-Shafer theory, have also been explored [9, 17] and are used in portions of our work (see Chapter 6).

Kautz's plan recognition algorithm is exponential in the size of the knowledge base. By conceptualizing plan recognition as a variant of a context-free grammar parsing problem, some researchers [64, 106] have developed approaches to reduce the complexity of the problem. Pynadath demonstrated a hybrid method for combining probabilistic reasoning with a grammar parsing approach using probabilistic state-dependent grammars (PSDGs) for plan recognition [72]. The PSDG model can be represented as a dynamic Bayesian network, which is a commonly used model for activity recognition (see next section). Goldman et al. [40] noted that one problem with the use of parsing as a model of plan recognition is that it does not handle partially-ordered or interleaved plans and proposed an alternate representation based on probabilistic Horn abduction.

Specialized search techniques have been developed to reduce the time required for plan recognition; for instance, RESC (Real-Time Situated Commitments) is a real-time approach to tracking the operator hierarchies of a Soar agent [100]. RESC uses information from the current world state to determine the validity of the operator hierarchies, commits to a a single interpretation, and backtracks in case of mistaken interpretation; this approach has the advantage that it can be used in real-time opponent modeling and handles reactive behaviors well [100]. Rather than committing to a single interpretation, the RESL algorithm marks every plan whose observations matches expectations and maintains it as a possible hypothesis [52]. Avrahami-Zilberbrand and Kaminka later developed a single-agent plan recognition algorithm [5] which improves on RESL in several ways: (1) more efficient observation matching through the use of feature decision trees (FDTs); (2) the use of temporal structure to rule out inconsistent hypotheses for current state queries.

2.2 Activity Recognition

The term *activity recognition* is used to describe the problem of segmenting and classifying low-level movement data into a higher-level description of the activity performed. Whereas plan recognition algorithms typically deal with symbolic data and atomic actions, activity recognition algorithms begin with traces of the human's position over time, gleaned from sources such as cameras, GPS readings, wireless signal strength measurements, or a motion capture apparatus.

The dynamic Bayesian network model has been applied successfully to both plan recognition and activity recognition. Bui has demonstrated the use of different types of DBNs—the Abstract Hidden Markov Model (AHMM) [16] and the Abstract Hidden Markov Memory Model (AHMEM) [15]—for recognizing the future destinations of a sin-

gle moving human from camera data. The AHMM model is representationally quite similar to the PSDGs introduced by Pynadath although it can only be used to model "memoryless" policies. An AHMM has no mechanism for remembering the originating subplan and thus cannot be used to model behavior hierarchies in which lower-level behaviors can be followed by a return of control to a higher-level behavior. However the AHMEM (Abstract Hidden Markov MEmory Model) rectifies this omission; Bui demonstrates an approximate inference technique based on Rao-Blackwellized particle filters that can handle partial observability and deep behavior hierarchies better than exact inference methods. Often the specialized problem of recognizing the human subject's future location is known as "goal recognition". N-gram models have been used successfully for goal recognition in interactive narrative environments [66] and RF-based wireless localization [112].

2.3 Multi-Agent Activity/Plan Recognition

Previous work on team behavior recognition has been primarily evaluated within athletic domains, including American football [48], basketball [13, 50], and Robocup soccer simulations [58, 78, 79]. Recognition for military air-combat scenarios has been examined in the context of event tracking [99] and teammate monitoring [52], A general framework for multi-agent activity recognition (Hierarchical Multiagent Markov Processes) [83] has been demonstrated for a single pair of humans moving around a laboratory.

To recognize athletic behaviors, researchers have exploited simple region-based [48] or distance-based [79] heuristics to build accurate, but domain-specific classifiers. For instance, based on the premise that all behaviors always occur on the same playing field with a known number of entities, it is often possible to divide the playing field into grids or typed regions (e.g., goal, scrimmage line) that can be used to classify player actions.

The first part of Intille and Bobick's work [47] on football game annotation addresses the low-level problem of extracting accurate player trajectories from video sequences that include substantial camera movement (panning and zooming). Using knowledge of the football field geometry and markings, they warp the original distorted image sequence into a rectified one with the appearance of a stationary camera and background. Then, by exploiting closed-world knowledge of the objects in the scene, they create context-specific feature templates for tracking individual players. Play recognition [48] is performed on player trajectories using belief networks both to recognize agent actions from visual evidence (e.g., catching a pass) and to determine the temporal relations between actions (e.g. before, after, around). Jug et al. [50] used a similar framework for basketball analysis. These frameworks do not address the problem of dynamic team composition; all agents are committed to the execution of the single plan for the duration of the play.

Unlike Intille and Bobick, Riley does not attempt to produce annotated traces with sequences of Robocup team behaviors. Instead he extracts specific information, such as home areas [80], opponent positions during set-plays [79], and adversarial models [78], from logs of Robocup simulation league games to be used by a coach agent advising a team. For instance, information about opponent agent home areas are used as triggers for coaching advice and for doing "formation-based marking", in which different team members are assigned to track members of the opposing team.[1] Formations are generated from past logs in two phases: (1) fitting a minimal rectangle that explains the majority of each agent's past positions; (2) modeling pair-wise distance correlations between agents by shifting the minimal rectangles.

Commentating from post-game log analysis has been demonstrated in the Robocup domain by the ISAAC system [74] using a combination of data mining and inductive

[1]Riley uses the term "formation" to specify a grouping of home areas, regions of the field where an agent can generally be found; in this thesis, formations designate a spatial pattern of relative agent positions.

learning; unsupervised data mining and knowledge discovery approaches have also been applied to the problem of recognizing strategic patterns from NBA basketball data [13]. However, the emphasis of the ISAAC system was to produce a high-level summary of game action, suitable for natural language summaries or debugging general team problems, rather than detailed play recognition of each team's actions.

In the 2005 Robocup coach competition, Kuhlmann et al. [58] demonstrated a game analysis approach in which the team's movement patterns were fitted to a parametric model of agent behavior. Patterns were scored according to their similarity to models learned from the pre-game logs. This work is conceptually similar to our data-driven approach for policy recognition (described in Chapter 6). Recently, Beetz et al. [10] developed a system for matching soccer ball motions to different action models using decision-trees.

There has also been work on extending single-agent plan recognition frameworks [15, 100], both to create symbolic [99] and probabilistic [83] multi-agent plan recognition frameworks. These efforts have focused on the use of temporal behavior models and do not extensively utilize spatial information; such models have also been employed to detect teamwork failures [52] and agent-coordination termination [83]. Due to the difficulty of acquiring reliable location data for multiple entities, much of the research has been evaluated in simulation; however improvements in sensor technology such as the microwave position system described in [11] should make real-world deployments possible in the future.

Tambe [99] observed that the use of team models for multi-agent plan recognition can aid the process of tracking team activity by constraining the tracking search and eliminating the execution of large numbers of agent models. Unlike our algorithm, his system $\text{RESC}_{\text{team}}$ does not maintain multiple hypotheses of team behavior nor does it precom-

pute temporal and inter-agent constraints to prune the plan library. RESC$_{team}$ executes a team model and invokes a minimal cost repair mechanism if the operator hierarchy selection or the role assignment causes match failure. RESC$_{team}$ assumes that the subteams are either known in advance or detected solely based on agents' proximity to each other; although this assumption is often valid, it is violated when a subteam that shares a common purpose becomes spatially separated (e.g., perimeter guarding in the MOUT domain).

Saria and Mahadevan [83] demonstrate how a single-agent probabilistic plan recognition framework (the Abstract Hidden Markov Model) can be extended to reason about the joint policies of multiple cooperating agents. Their model, the Hierarchical Multiagent Markov Process, only represents two possible outcomes of agent coordination: (1) joint policies terminate when all of the participating agents complete their individual policies; (2) joint policies terminate when any of the agents completes its policy. This limited subset of execution outcomes is insufficient to model the common situation in which an agent abandons one plan in favor of a new plan, while the other agents continue executing the initial plan. Our framework (described in Chapter 5) correctly handles this situation because we do not assume that agent plan abandonment terminates the team plan. Also it is unclear whether this type of simple coordination mechanism can scale to handle the complexity of larger teams with multiple subteams.

Unlike other groups, Kaminka and Tambe's work [52] on plan recognition for socially-attentive monitoring does not treat teams as automatically sharing a single plan. Due to message failure and sensor error, team members' plans might diverge, in spite of sharing the same top level goal. The focus of their research on socially-attentive monitoring is the use of plan recognition by teammates to determine when other team members have inconsistent beliefs. Kaminka [51] developed the concept of *team coherence*, the ratio of total agents to the number of active plans, to represent the possibility of team coordi-

nation failures; he demonstrates that plan recognition can be used as part of scalable disagreement-detection system to detect the existence of incoherent team plans. In this thesis, we represent these 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.

2.4 MOUT Domain

Many of the experiments in this document use behaviors, maps, and scenarios derived from human soldiers performing MOUT (Military Operations in Urban Terrain) tasks as described in [76]. In MOUT scenarios, platoons of soldiers attempt to achieve strategic objectives, such as clearing buildings, escorting convoys, and attacking enemy positions, within a cluttered, hazardous urban environment. The commanding officers must react to new threats in the environment and changes in spatial layout (e.g., blocked roads, booby-trapped zones) without direct guidance from the chain of command. It is often unclear whether observed people in the combat zone are civilians or enemy snipers. Although deliberative planning is required to systematically clear large areas of urban terrain, soldiers must execute reactive behaviors in the face of unexpected threats such as snipers and booby-traps.

The MOUT domain poses several significant teamwork challenges. Adversarial team planning is required when soldiers coordinate to outmaneuver and flank an enemy in an covered position. Successful teams cooperate to assist team members in trouble, such as wounded soldiers or teammates pinned down by enemy fire. Some team tasks, such as moving in formation, demand tightly-coupled physical coordination whereas other team tasks (e.g., securing a defensive perimeter) only require loose coordination. Splitting into

sub-teams is commonly required to achieve goals; even formation movement is usually accomplished by dividing into smaller groups.

Spatial environmental features (buildings, intersections, doorways) are important features that influence MOUT planning [12], yet unlike team sports which are played on a field that can be simply characterized by regions (e.g., goal, yard lines, end zone), MOUT spatial landmarks are a complex combination of features. According to the cognitive task analysis of the building clearing task [71], spatial cues dominate all of the task-focused decision requirements of building clearing MOUT operations. The six *task-focused* decision requirements include: (1) securing the perimeter; (2) approaching the building; (3) entering the building; (4) clearing the building; (5) maintaining and evaluating security; (6) evacuating the building. For these tasks, features such as proximity to other buildings, opportunities for cover, open spaces, windows in the building, street layout, fortifications, height of buildings, locations of stairways, obstacles, potential booby-traps, and doors, are critical cues that human decision-makers use to determine how to achieve the building clearing task. Dynamic spatial features such as civilian activity and presumed enemy location also factor into several stages of the decision making process. Out of the five *task-independent* decision requirements, spatial cues are used for three of them: (1) maintain the enemy's perspective; (2) maintain big picture and situation awareness; (3) project into the future. For these three decision requirements, important spatial cues include: location of hallways, stairwells, general building layout, teammates' locations, and last known enemy positions.

Although some aspects of MOUT planning are highly structured, commanders are encouraged to be sufficiently unpredictable to frustrate potential enemies. The cognitive task analysis for building clearing [71] offers suggestions such as "Refrain from establishing patterns the enemy could learn" and "Use creative thinking to bypass obstacles". This

suggests that expert MOUT teams demonstrate variability in how they approach buildings with similar layouts, perhaps approaching from different vantage points or clearing rooms in different orders. However, expert teams exhibit coherent movement patterns. In Jirsa's study on MOUT team coordination dynamics [49], he hypothesizes that expert team performance is limited to a low-dimensional subspace. He suggests that the degree of confinement to the experts' subspace is a measure of team performance and coherence.

In summary, the MOUT domain offers the following interesting challenges to team behavior recognition:

1. The behavior recognition algorithm must be able to handle deliberative, hierarchical behaviors, as well as reactive behaviors.

2. The spatial landmarks that are important cues for MOUT behavior recognition cannot be simply characterized by static regions in a playing field, as has been done in other work [48, 78] on team behavior recognition.

3. Team tasks frequently require the splitting and merging into subteams, making the agent-to-team assignment problem more challenging.

4. Complete MOUT scenarios, including a soldier platoon, enemy snipers, and civilians, can feature large numbers of agents (50–100) cooperating in multiple team groupings; this is significantly greater than the number of agents addressed in multi-agent plan recognition frameworks [83] that only address coordination between pairs of agents.

Most of the work in the MOUT domain has tackled the problem of plan generation for computer-generated forces (CGFs) rather than the inverse problem of plan recognition. Pearson and Laird [70] created a diagrammatic tool for authoring exemplars of spatial plans which their system later generalized into SOAR productions to be incorporated

into MOUT bots; Best and Lebiere [12] created a spatial language for authoring plans to be executed in ACT-R. The primary goal of these systems was to create easily-authored and reusable spatial representations that MOUT subject matter experts could use to program cognitive models.

Several MOUT training systems have been developed to teach human soldiers the building clearing task. The Collaborative Warrior tutor, a prototype model-tracing tutoring system for MOUT described in [65], included both plan generation for CGFs and plan recognition of the trainee's actions. This system used very simple spatial models to evaluate the correctness of the trainee's plan (e.g. whether the trainee failed to clear rooms of enemy soldiers) and no recognition results are reported. Fowlkes et al. [37] developed a haptic-enhanced team training environment to train human subjects to move in a back-to-back formation with a simulated soldier; their study examined how haptics could improve acquisition of movement and communication skills. Recently, Hoover et al. [44] constructed a full-scale MOUT training environment at the 263rd Army National Guard, Air and Missile Defense Command site. Teams of trainees learn the room clearing task in a "shoothouse" consisting of 6 rooms and interconnecting hallways, instrumented with 36 cameras.

Chapter 3

Formation Identification for Agent Teams

To analyze performance of team tasks, we need to extend existing behavior recognition formalisms to accommodate group behaviors. Due to the increase in number of actions generated, assuming that each agent is simultaneously executing actions, team behaviors have a more complicated *temporal* structure than single agent behaviors. However team plans involving physical movement also possess a distinctive *spatial* structure, characterized by the relative positions of teammates and external landmarks, that can be exploited to classify team behaviors. This chapter describes a framework for constructing spatio-temporal models to robustly recognize physical team behaviors from position traces of the agents' movement.

We demonstrate our method in the domain of MOUT (Military Operations in Urban Terrain) team planning to recognize military team behaviors customarily performed by human soldiers while moving through an urban environment. Examples of team behaviors include building entry, perimeter guarding, opponent flanking, and formation movement, such as stacked and bounding overwatch.[1] Although we focus on MOUT team

[1]See Section 3.1 for more details on MOUT team behaviors

behaviors, our algorithms are also applicable for recognizing team behaviors in other military, robotic, and athletic domains.

Specifically, we present two classes of spatial models and illustrate how they can be used to robustly recognize team behaviors in the presence of spatial variations, noise, clutter, and human variability in behavior execution.

team templates: models encoding static spatial relationships between team members and external landmarks. These models are composed of the relative positions of map entities and can be generalized to different geographic layouts using similarity transforms. To efficiently identify team templates over large map areas without using exhaustive search, we employ an efficient randomized search technique, RANSAC (Random Sampling and Consensus) [35].

spatially-invariant HMMs: a set of Hidden Markov Model classifiers applied over short overlapping time windows to identify temporal patterns in agent team configurations. By representing the agents' positions in a canonical reference frame, the classifier is robust to certain spatial variations.

We evaluate our modeling and recognition techniques on two different types of data: (1) large urban simulation battle maps, and (2) position traces of two-person human teams performing designated sequences of MOUT behaviors in a customized version of Unreal Tournament (a commercially available first-person shooter game).

3.1 MOUT Formations

MOUT (Military Operations in Urban Terrain) scenarios involve moving teams of soldiers and vehicles through heavily-cluttered urban areas to accomplish high-level strate-

gic objectives. A common task performed by human soldiers is building clearing, during which a firing-team of soldiers must enter a building and clear it of enemy occupants and hazards; the challenge is to explore unknown areas while constantly remaining in a defensible position against enemy attack. To achieve this, soldiers move through the urban terrain using set procedures that maintain a defensive position while still propelling the team forward.

The cognitive task analysis of building clearing given in [71] emphasizes the importance of spatial cues in the soldiers' decision making process. Features such as proximity to other buildings, opportunities for cover, open spaces, windows in the building, street layout, fortifications, height of buildings, locations of stairways, obstacles, potential booby-traps, and doors, are critical cues that expert human decision-makers must take into account while performing the task. Certain team behaviors are triggered by the spatial characteristics of the terrain; for instance, firing-teams have special procedures for moving through L-shaped and T-shaped street intersections. In these cases, we hypothesize that static spatial configurations, such as the ones described in our team template models, are highly predictive of the teams' actions.

However, there are some cases, particularly for smaller two-soldier subteams, in which static spatial configurations, by themselves, lack predictive power. The second part of the chapter focuses on three such behaviors: stacked movement, bounding overwatch, and buttonhook entry commonly used during the building clearing task. These behaviors are difficult to identify solely on the basis of static snapshots due to their spatial similarities (see Figure 3.4).

During stacked movement the purpose is to move the team in such a way that their gun angles completely span all possible areas of approach; the team moves slowly and in synchrony. For moving through open areas or intersections, this approach is less fea-

Figure 3.1: MOUT scenario in customized Unreal Tournament environment from spectator viewpoint. A pair of human players control soldiers A and B as they execute the procedure for traversing a T-shaped intersection. The bot models and animations were modified to conform to the appearance of real human soldiers rather than the larger-than-life UT fantasy fighter models.

sible since it's difficult to cover all possible threatened areas. In this case, the bounding overwatch behavior is used; one soldier moves forward while the other remains stationary. The buttonhook entry is similar to bounding overwatch; one soldier moves through the doorway hugging the wall while the other soldier waits and guards. After the entry is clear, the second soldier moves through the doorway hugging the opposite wall. To recognize these behaviors, our classifier needs to exploit the temporal information in behavior sequences in conjunction with spatial information on the position and velocities of team members in a way that is robust to geographic variations that modify the execution of the behavior.

3.2 Methods

In this section, we present our methodology for developing spatial models and using them to classify instances of team behavior. Section 3.2.1 discusses the representation and authoring of team templates to encode static spatial relationships between team members and external landmarks. Section 3.2.2 provides an overview of the randomized search technique, RANSAC (Random Sampling and Consensus) that we use to efficiently and robustly identify these templates over large simulation map areas without resorting to exhaustive search. In Section 3.2.4 we describe the Unreal Tournament simulation environment that we use to collect data from pairs of human subjects performing the three types of statically-similar team behaviors (bounding overwatch, buttonhook entry, stacked movement). Finally, Section 3.2.5 presents our procedure for converting agent position traces into the canonical representation used by our Hidden Markov Model classifiers (Section 3.2.6).

3.2.1 Static Formation Representation

To model team behaviors, we developed a tool that enables the interactive specification of static formation models based on their characteristic spatial relationships. Once a library of spatial models has been constructed, they can be used to classify formations of MOUT entities on a 2D annotated urban map. If labeled training data of team behaviors exists, these team templates could potentially be learned from data using a supervised classifier; however all the experiments described here were created with the authoring interface shown in Figure 3.2.

Each formation template contains the following attributes:

behavior name: Behaviors are represented by collections of spatial models; however no particular temporal structure, or execution order, is attached to the collection. Each spatial model can only belong to a single behavior; we do not include models which appear in a large set of behaviors since they are unlikely to help in discriminating between multiple behaviors.

spatial position of entities: Spatially-compact entities such as soldiers and small terrain features (e.g., doors, obstacles providing cover), are treated as points and represented by the (x, y) locations of their centroids; larger entities, such as walls or roads, are represented by sets of points connected by visibility constraints.

entity type: For our library of MOUT behaviors, we designated the following eleven types of entities: teammate, opponent, civilian, person (unknown type), hard cover, soft cover, empty area, window, intersection, doorway, hazard, and spatial objective (e.g., assembly area).

An important consideration in matching formations is generalization—how well do models developed for one scenario match formations observed in a different spatial lay-

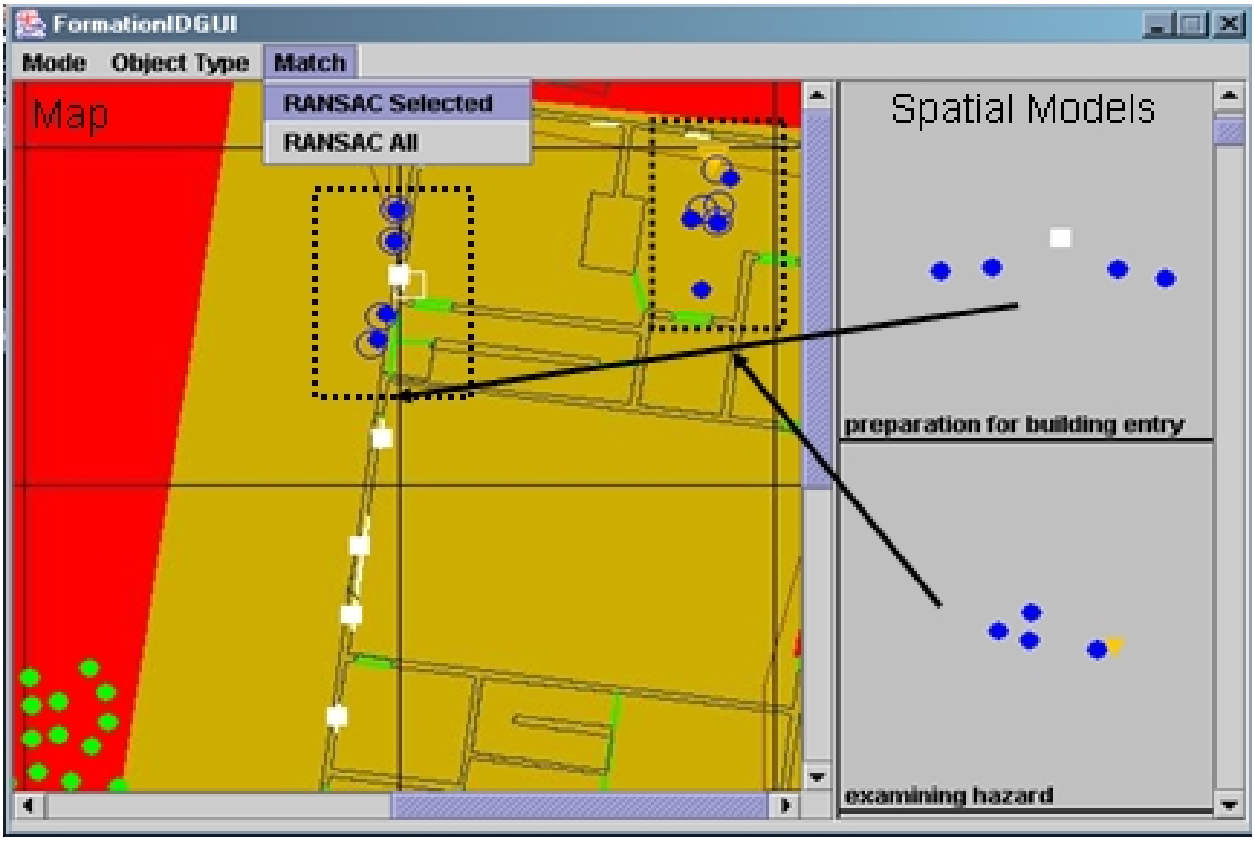


Figure 3.2: Spatial model authoring and matching system. The library of previously created spatial models are shown to the right of the screen; the left side of the GUI displays an annotated map to be analyzed. A fire team of soldiers (blue circles) examining a hazard (orange inverted triangle) are displayed on the map; a second fire team prepares to enter the building through the door (marked by the white rectangle). Our matching technique successfully associates both groupings of soldiers with the correct spatial models; hollow circles and rectangles show the locations of the entities as predicted by model projection.

out? Without generalization it is impractical to exhaustively enumerate all possible spatial relationships of interest that can occur across different maps. To address this problem, we explore generalization as defined by the space of similarity transforms. Specifically, we consider a model to match an observed formation if there exists a combination of rotations, translations and scalings that map the model to the observed entities. Such transforms can be parametrized in homogeneous coordinates as:

$$\mathbf{T} = \begin{pmatrix} s\cos(\theta) & s\sin(\theta) & t_x \\ -s\sin(\theta) & s\cos(\theta) & t_y \\ 0 & 0 & 1 \end{pmatrix},$$

where θ is the angle of rotation, s is the scale factor, and t_x and t_y denote the translation vector. We can extend this formulation to 3-D transforms by using a 4×4 matrix, but we focus on 2-D transforms suitable for the MOUT domain in this chapter. Given a model consisting of entities whose positions are denoted by $\{(x_i, y_i) : i \in 1 \ldots N\}$, one obtains their corresponding transformed positions $\{(X_i, Y_i)\}$ through

$$\begin{pmatrix} X_1 & \ldots & X_i & \ldots & X_N \\ Y_1 & \ldots & Y_i & \ldots & Y_N \\ 1 & \ldots & 1 & \ldots & 1 \end{pmatrix} = T \begin{pmatrix} x_1 & \ldots & x_i & \ldots & x_N \\ y_1 & \ldots & y_i & \ldots & y_N \\ 1 & \ldots & 1 & \ldots & 1 \end{pmatrix}.$$

The goal of matching a formation model to observed map entities is to determine whether there exists a similarity transform T that maps elements in the model to legal entities on the map. The next section describes a robust technique for efficiently searching the space of possible transforms.

3.2.2 Efficient Template Matching

Given a set of spatial models and valid transforms, the problem of determining which spatial models are applicable to the current map can be solved by searching the space

of potential transforms and models to find all the combinations of model plus transform that result in a match of sufficient quality. A standard approach to similar problems is exhaustive template matching [7]. The space of transforms is discretized to create a manageable set of maps that can be applied to each template. Each transformed instance of the model is evaluated against the map using a function that quantifies the quality of the given match, and those model instances that score better than a threshold are retained. This process is exhaustively repeated over a range of scales and rotations, over all models in the library. Clearly, such a process is time consuming, scales poorly to higher dimensional transforms, and can be sensitive to discretization effects, noise and occlusion.

Instead, we employ a statistically-robust technique, Random Sampling and Consensus (RANSAC) [35], to efficiently sample the space of transforms using hypotheses generated from minimal sample sets of point correspondences. The key difference is that, rather than enumerating over the unrestricted space of all similarity transforms, we only sample from the subset of transforms that are consistent for some part of the model. The algorithm can be summarized as follows:

hypothesis generation: entities are drawn uniformly and at random from the annotated map and associated with randomly-selected entities of the same type in the model. Two pairs of corresponding entities are sufficient to uniquely specify a transform hypothesis. This data-driven method of generating hypotheses is much more efficient than uniformly sampling the space of possible transforms or exhaustively searching a discretization of the transform space.

hypothesis testing: Given a transform hypothesis, we project all of the entities in the model to the coordinate frame of the map and assess the quality of the match based on both spatial similarity and type matching. This gives us the likelihood that the given hypothesis could have generated the observed data in the map.

For each spatial model, we use RANSAC to randomly generate and test a large number of plausible transforms and select those hypotheses (a combination of a model and a valid transform) with match quality better than a specified threshold.

Since our spatial transforms have four degrees of freedom, they can be fully specified by two pairs of point correspondences. First, we randomly select two entities from the model under consideration; then based on the types of the entities (e.g., civilians, hard cover, hazard) we randomly select candidate entities on the map with compatible object types. The positions of these entities is used as the minimal set to generate a transform hypothesis as follows.

Given the minimal set $\{(x_1, y_1), (x_2, y_2)\}$ from the model and the corresponding set of points $\{(X_1, Y_1), (X_2, Y_2)\}$ from the map, we generate a third virtual pair of correspondences $(x_3, y_3) \mapsto (X_3, Y_3)$ where

$$\begin{aligned} x_3 &= x_1 + y_2 - y_1 \\ y_3 &= y_1 + x_1 - x_2 \\ X_3 &= X_1 + Y_2 - Y_1 \\ Y_3 &= Y_1 + X_1 - X_2. \end{aligned}$$

The third point creates a right-angled isosceles triangle with clockwise vertex order in both model and map. Three point correspondences completely specify an affine transformation, and an affine transform that preserves both aspect ratio and chirality is an orientation-preserving similarity transform. From these three pairs of correspondences, we can directly recover $\mathbf{T}$ using matrix inversion,

$$T = \begin{pmatrix} X_1 & X_2 & X_3 \\ Y_1 & Y_2 & Y_3 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}^{-1}.$$

Although this is a solution to a general affine transform given three pairs of point correspondences [42], $\mathbf{T}$ is guaranteed to be a valid, orientation-preserving similarity.

To score a hypotheses, we transform the location of every entity in the model to the map using the transform $\mathbf{T}$. Each model entity contributes a positive vote for the given hypothesis if the distance from its predicted location to a nearby valid entity of compatible type falls within a specified threshold. The quality of a hypothesis is defined as the normalized sum of these individual votes.

Since RANSAC stochastically searches the space of possible transforms it is not guaranteed to find the best match. However the following formula can be used to determine how many iterations are necessary to achieve the best match with a specified probability of success [109]:

$$m = \left\lceil \frac{\log(1-P)}{\log[1-(1-\epsilon)^s]} \right\rceil .$$

P is the target probability (e.g., $P = 0.99$ means the best match is found 99% of the time). s is the number of elements required to define the minimal set ($s = 2$ since a similarity transform requires 2 pairs of point correspondences). ϵ is the expected fraction of outliers in the data set. In traditional RANSAC applications ϵ is typically only about 0.1 (10% of the points are expected to be invalid). For our application, the fraction of outliers refers to the number of map annotations that do not match a single model; since each map actually contains multiple models in addition to entities that do not match any model the fraction of expected outliers is approximately 0.95. From the formula above, this indicates that the number of RANSAC iterations required to reliably find the best match is 1840. Note that executing the given number of iterations does not guarantee a particular level of classification accuracy; it merely ensures that there is only a 0.01% probability that a similarity transform exists that would achieve a higher score than the one returned by RANSAC. In Section 3.3.1, we present classification results for identifying team behaviors

from 2D annotated maps of simulated urban layouts using our team template model and RANSAC.

3.2.3 Spatio-temporal Classification

There are some cases, particularly for smaller two-soldier subteams, in which static spatial configurations, by themselves, lack sufficient predictive power. To recognize these behaviors, our classifiers need to exploit the *temporal* information in behavior sequences in conjunction with spatial information on the position and velocities of team members. Since team behaviors can be executed in a variety of terrains, the classifiers must be robust to deviations in behavior execution caused by the team's response to local terrain features. However, arbitrarily introducing similarity transforms in the middle of a behavior sequence can destroy the spatio-temporal pattern created by the team's movements. To address the problem, we developed spatially-invariant classifiers, by transforming our position data into a canonical reference frame defined by the team's motion, and applying a set of Hidden Markov Model classifiers to recognize three statically similar team behaviors (bounding overwatch, buttonhook entry, stacked movement).

3.2.4 Human Data Collection

To evaluate our spatio-temporal classifiers, we collected data from a pair of human players using our modified Unreal Tournament game interface to manipulate "bots" through a small urban layout while performing a particular sequence of team behaviors (see Figure 3.1). Note that the subjects were not playing Unreal Tournament, but using Unreal Tournament to execute sequences of commonly used MOUT team maneuvers. To directly monitor the performance of human players, we customized Unreal Tournament (UT) us-

ing the game development language *Unrealscript*. Many of the original UT game classes were written in *Unrealscript* and thus can be directly subclassed to produce modified versions of the game (known as mods); for example, Gamebots [53] is an example of a mod that allows external programs to control game characters using network sockets.

We developed our own `TrainingBot` mod that allows us to save the state of all the bots in the scenario; currently we save each player's ID number, position $(x, y.z)$, and rotation $(\theta$,$\phi)$ every 0.15 seconds. This information is useful for both offline behavior analysis and for a separate replay mode that allows us to create bots that follow the paths recorded by the original players.

3.2.5 Canonical Representation

Due to the continuous nature of the domain, automatically determining the exact transition points between team behaviors is a difficult problem. While approaching and entering buildings, the players continue moving their bots, changing team behaviors as appropriate for the physical layout. We address this issue by dividing the traces into short, overlapping time windows during which we assume that a single behavior is dominant; these windows are classified independently as described in Section 3.2.6. To recognize team behaviors performed in different physical layouts, it is important for our classifier to be rotationally- and translationally-invariant; we achieve this by transforming the data in each window into a canonical coordinate frame as described below.

More formally, we define:

- $a \in 1, \ldots, A$ is an index over A agents;
- j is an index over W overlapping windows;
- $t \in 1, \ldots, T$ is an index over the T frames in a given window;

- $\mathbf{x}_{a,j,t}$ is the vector containing the (x, y) position of agent a at frame t in window j.

The centroid of the positions of the agents in any given frame can be calculated as:

$$\mathbf{C}_{j,t} = \frac{1}{A} \sum_{\forall a} \mathbf{x}_{a,j,t}.$$

We describe the configuration of the agent team at any given time relative to this centroid to achieve translation invariance. However, rather than rotating each frame independently we define a shared canonical orientation for all of the frames in a window. This is important because it allows us to distinguish between similar formations moving in different directions (e.g., agents moving line abreast vs. single file). One standard technique for defining a canonical orientation is to use the principal axis of the data points for that window, which can be calculated using principal component analysis (PCA). However for efficiency we have empirically determined that one can achieve similar results by defining the canonical orientation as the displacement of the team centroid over the course of this temporal window: $\mathbf{d}_j = \mathbf{C}_{j,T} - \mathbf{C}_{j,1}$.

We rotate all of the data in each window so as to align its canonical orientation $\mathbf{d}_j$ with the x-axis, using the rotation matrix $\mathbf{R}_j$. Thus the canonical coordinates, $\mathbf{x}'$, can be calculated as:

$$\mathbf{x}'_{a,j,t} = \mathbf{R}_j \mathbf{x}_{a,j,t} - \mathbf{c}_{j,t}.$$

Our recognition technique (described in Section 3.2.6) also employs an estimate of the agents' velocities, which we compute as:

$$\mathbf{v}_{a,j,t} = ||x'_{a,j,t+1} - x'_{a,j,t}||.$$

The intuition behind the canonical representation is that a team's behavior is defined not by the absolute positions and orientation of its agents, but rather by their relative motions. In the canonical representation, the team is oriented so that it is facing along

the positive x-axis, and motions are encoded with respect to the team's centroid. In this representation, instances of team behaviors such as the stacked formation are clearly distinguishable from bounding overwatch since the lead agent's x position in the former is consistently positive while the corresponding agent in the latter oscillates from positive to negative with each bound. The agents in the buttonhook behavior both approach each other (as they pass through the doorway) but then separate; this is clearly different from stacked formation but superficially resembles bounding overwatch.

3.2.6 HMM Classification

For each canonically-transformed window in our trace, our goal is to select the best behavior model. We perform this classification task by developing a set of hidden Markov models (HMMs), one for each behavior b, and selecting the model with the highest log-likelihood of generating the observed data. Our models ($\{\lambda_b\}$) are parameterized by the following:

- N, the number of hidden states for the behavior;
- $\mathbf{A} = \{a_{ij}\}$, the matrix of state transition probabilities, where $a_{ij} = Pr(q_{t+1} = j | q_t = i), \forall i, j$ and q_t denotes the state at frame t;
- $\mathbf{B} = \{b_i(o_t)\}$, where $b_i(o_t) = \mathcal{N}(\mu_i, \Sigma_i)$. The observation space is continuous and approximated by a single multivariate Gaussian distribution with mean, μ_i and a covariance matrix, Σ_i, for each state i;
- $\pi = \{\pi_i\}$, the initial state distribution.

For our problem, given A agents in a team, the observations at time t and window w are the tuple:

$$o_t = (\mathbf{x}'_{1,w,t}, v_{1,w,t}, \ldots, \mathbf{x}'_{A,w,t}, v_{A,w,t}).$$

We determine the structure for each behavior HMM based on our domain knowledge. For instance, the stacked behavior can be described using only two states ($N = 2$), whereas we represent the more complicated bounding overwatch behavior using six states connected in a ring. Each hidden state captures an idealized snapshot of the team formation at some point in time, where the observation tuple (in canonical coordinates) is well modeled by a single Gaussian. Rather than initializing the HMMs with random parameters, we use reasonable starting values. These can be polished using expectation-maximization (EM) [28] on labeled training data.

To determine the probability, $Pr(o_{1\ldots T}|\lambda_b)$, of generating the observed data with the model λ_b, we employ the forward algorithm [73] as implemented in the Hidden Markov Model toolbox [67]. We classify each window segment with the label of the model that generated the highest log-likelihood.

3.3 Formation Recognition Results

We evaluate our methods using two sets of experiments: (1) formation recognition in simulated 2-D overhead maps of urban areas annotated with the location of MOUT entities (see Section 3.2.1). (2) behavior identification from activity traces of two-person human teams performing sequences of MOUT behaviors (see Section 3.2.4). The former assesses the accuracy of the RANSAC-based method for static formation recognition while the latter examines the performance of spatially-invariant HMMs on spatio-temporal traces.

3.3.1 Static Formation Matching

To test the robustness of our approach for static formation matching, we add clutter to the maps and distort formations by perturbing the positions of MOUT entities. Figure 3.3 reports the precision (fraction of correctly-classified results) and recall (fraction of formations that were detected) of our classifier under different conditions of clutter and location perturbation. Note that our approach independently matches each template against the data; thus, all matches that exceed the threshold score are reported as detections. The precision/recall curves are generated by varying this threshold parameter. There is no intrinsic restriction against assigning the same map entity to different templates — this enables us to create templates corresponding to a team and its component sub-teams, and to simultaneously recognize both.

In each experiment, we randomly place fifty MOUT formations on the urban map and report the precision and recall averaged over ten RANSAC searches. The left panel of Figure 3.3 shows the effects of adding clutter (spurious MOUT entities of the appropriate type) to the map, without increasing the number of RANSAC iterations. The percentage of clutter is measured against the total number of MOUT entities in the formations on the map (thus +100% clutter denotes a 1:1 ratio between spurious and desired MOUT entities). The results show that, as expected, the RANSAC-based approach is very resistant to the presence of spurious entities on the map, and that precision/recall remain very high even at the extreme clutter levels. The right panel of Figure 3.3 shows precision/recall results for experiments where the locations of each of the MOUT entities were perturbed with iid Gaussian noise. We show results under three noise conditions, where the standard deviation of the noise is $\sigma \in \{0, 10, 20\}$ pixels. As expected, the performance degrades as noise is added since the spatial configuration of the formation ceases to resemble the formation represented by the idealized model. However, the technique

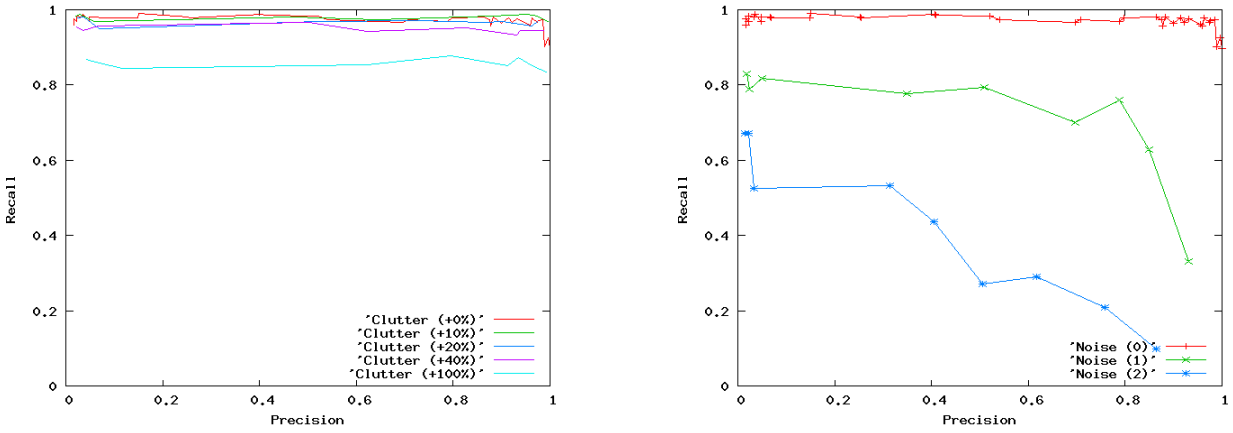


Figure 3.3: Precision and recall curves for matching team templates using the RANSAC-based method. Fifty formations were placed on an urban map using randomized similarity transforms. Each run employed 100,000 iterations and the precision/recall averaged over 10 trials is shown. The left panel shows the effect of adding spurious MOUT entities (clutter) while maintaining the same number of iterations. The right panel shows the results of distorting observed formations by perturbing the location of each MOUT entity on the map with iid Gaussian noise ($\sigma \in \{0, 10, 20\} pixels$). Our method is extremely robust to clutter and performs adequately under moderately noisy conditions.

correctly identifying an acceptable fraction of the formations under reasonable noise conditions (80% precision and 80% recall with noise at $\sigma = 10$).

3.3.2 Dynamic Formation Recognition

To evaluate our spatio-temporal classification method, we developed HMM models for three behaviors typically employed by two-person firing teams during the building clearing task: stacked movement, bounding overwatch, and buttonhook entry (see Figure 3.4). Note that these three behaviors look very similar in static formation snapshots and can only be robustly recognized by observing spatio-temporal traces. Position data was simultaneously recorded from two subjects at 0.15 second intervals using our `TrainingBot`

mod (see Section 3.2.4). Players executed team behaviors in predesignated sequences, transitioning smoothly from one behavior into the next, adapting each behavior as needed to the local physical layout (turning corridors, entering rooms). The traces were divided into overlapping 20 frame (3 second) windows, which were transformed into a canonical coordinate frame as described in Section 3.2.5 (illustrated in the inset of Figure 3.4). The window size was empirically selected base on the observed average speed of the MOUT soldiers.

By using real data collected from human players rather than simulated traces, we can evaluate the robustness of our approach to realistic deviations during behavior execution. Figure 3.4 (right) shows a raw trace for each behavior; note that consecutive portions of the same behavior do exhibit significant variation, as observed in the bounding overwatch behavior. Each behavior was also performed in a variety of local physical layouts. Table 3.1 presents the classification results (confusion matrix) for the three modeled behaviors; the accuracy of the HMM approach is good, particularly for the stacked formation. Buttonhook entry is sometimes confused with bounding overwatch, as may be expected from similarities in the canonical representation shown in Figure 3.4.

3.4 Discussion

The two approaches presented in this paper, RANSAC-based static formation matching and spatially-invariant HMMs, are complementary. Team templates are easy to author and can be rapidly executed over large, cluttered maps. Spatially-invariant HMMs are more labor-intensive to design, but are well-suited for classifying statically-indistinguishable behaviors. Formation templates are effective at determining the spatial context of an observed behavior, enabling improved discrimination between behaviors that display a

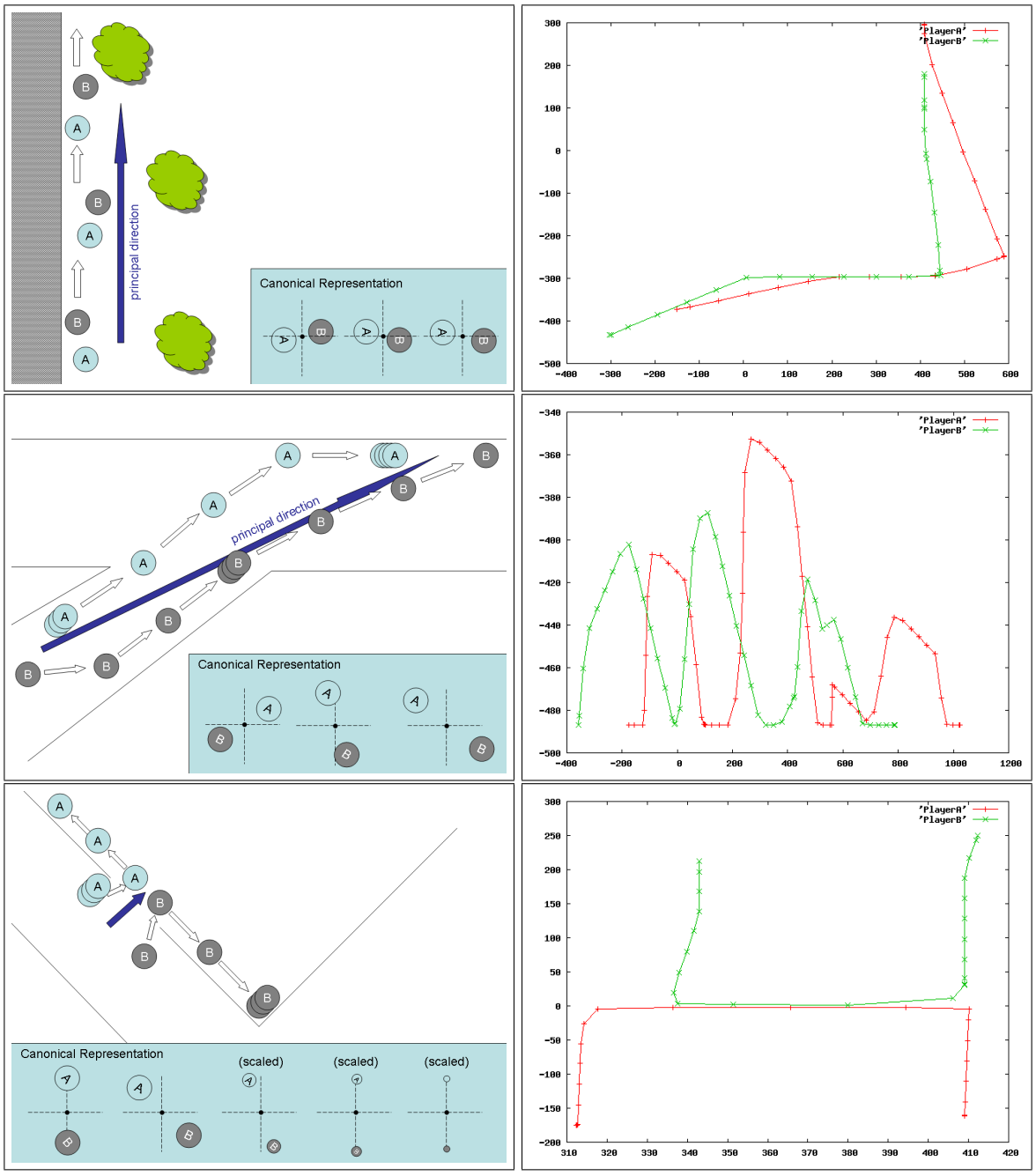


Figure 3.4: Team Behaviors: Stacked Formation (top), Bounding Overwatch (middle), Buttonhook Entry (bottom). Schematics for each behavior, along with the canonical representation for several frames, are depicted in the left column. A sample raw trace for each behavior is shown in the right column; the coordinates of the axes are in Unreal Tournament length units.

Table 3.1: Confusion matrix for HMM behavior classification. The ground truth is given in the left column; the classification result is given in the top row. The spatially-invariant Hidden Markov Model approach achieves good accuracy. Buttonhook entry is occasionally confused with bounding overwatch, since both are superficially similar in their initial stages, as seen in Figure 3.4.

	stacked	bounding	buttonhook
stacked	**90%**	10%	0%
bounding	14%	**67%**	19%
buttonhook	0%	33%	**67%**

similar temporal structure. For instance, although a buttonhook entry and a bounding overwatch often appear similar from the perspective of a spatially-invariant HMM, the presence of context can enable us to disambiguate them since the former behavior is typically performed to enter a door or window, while the latter is used to move the formation down a corridor or street.

The MOUT domain demands scalability, both in terms of larger team sizes and large maps. Our static formation matching approach scales well to larger team sizes. In fact, introducing more points into the team template is actually beneficial for two reasons: (1) the minimal set required to calculate the transform is independent of team size; (2) including additional model entities improves the robustness of the matching since a greater number of entities can contribute votes in support of the best hypothesis, and it is less probable that an accidental configuration of spurious entities will be a good match. Our approach also scales well to large map sizes since the complexity of the algorithm depends not on the map nor its discretization but rather on the number of entities.

In principle, the spatially-invariant HMM approach can also be applied to larger teams;

however constructing models for teams with more members can become difficult since additional team members create a greater variety of observable configurations (even in our canonical representation). For the MOUT domain, dividing larger teams (of 8 or 16) into small (e.g., two-person) subteams that can be be analyzed independently can be a practical approach. Larger team behaviors would be detected by combining the outputs of specialized classifiers (e.g., detecting invariants) rather than requiring a single, highly-complex classifier.

This chapter described two complementary methods for recognizing physical team behaviors from observed formations. We showed that our techniques are robust to the effects of local spatial variations, clutter, noise, and human variability during behavior execution. However, a fundamental assumption in both of our approaches is that the team composition is fixed and known *a priori*. The next chapter relaxes this assumption and presents approaches for recognizing behaviors in scenarios with dynamic team composition.

Chapter 4

Simultaneous Team Assignment and Behavior Recognition

4.1 Introduction

Although there has been considerable research on the problem of single-agent behavior recognition, there has been substantially less work on multi-agent behavior recognition. Most of the previous work makes one of two assumptions: (1) each agent is a decoupled entity that can be analyzed individually using a single-agent activity inferencing algorithm, or (2) the agents are always working together and can be analyzed as a single cohesive entity represented by a high-dimensional feature vector. In either case, team composition is generally assumed to be *static*; this chapter specifically addresses the problem of behavior recognition for teams with *dynamic* team composition. In scenarios with dynamic team composition, agents are assumed to be independent entities that coordinate with other agents to perform team behaviors; teams disband when the behavior is complete, merge with other teams to create larger formations, and occasionally split into

subteams to perform multiple behaviors in parallel. Relaxing the assumption of static team assignment is desirable because it enables the analysis of more complex team tasks. However this makes the team behavior recognition problem substantially more difficult since behaviors are characterized by the aggregate motion of the entire team and cannot generally be determined by observing the movements of a single agent in isolation.

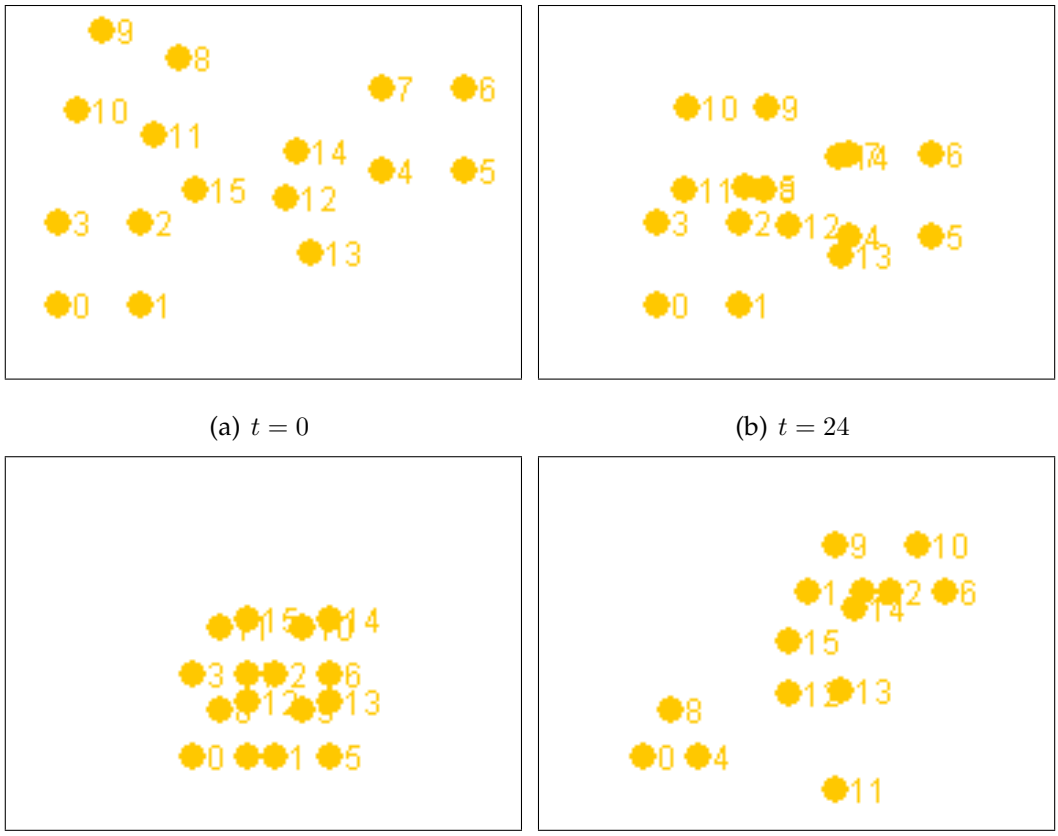
We introduce a new algorithm, Simultaneous Team Assignment and Behavior Recognition (STABR), that recovers both team assignments and behavior annotations from traces of agent position over time. STABR leverages information from the spatial relationships of the team members to create sets of potential team assignments at selected time-steps. These spatial relationships are efficiently identified using the formation recognition approach described in Chapter 3 to generate potential team assignment hypotheses. Sequences of team assignment hypotheses are evaluated using dynamic programming to derive a parsimonious explanation for the entire observed spatio-temporal trace. To prune the number of hypotheses, potential team assignments are fitted to a parameterized team behavior model; poorly-fitting hypotheses are eliminated before the dynamic programming phase. The proposed approach is able to perform accurate team behavior recognition without exhaustive search over the partition set of potential team assignments, as demonstrated on several scenarios of simulated military maneuvers.

4.2 Problem Formulation

We formulate our 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 $\mathcal{A}$. An agent that is not currently a member of any team is known as a **singleton**, and

is unrestricted in its motion. By contrast, the agents in a team are constrained to move according to a set of **team behaviors**, $\mathcal{B}$. The subset of behaviors available to a given team is specified by the domain and can depend on the number of agents in the formation and their relative configurations. For instance, the domain could specify that four agents in a square formation may execute a "wheel" (formation advances in an arc by rotating about a corner), but not a "pivot" (formation rotates about its center), which may be restricted to teams of three agents. In the course of a scenario, agents (either singletons or subsets of disbanding teams) can assemble into new teams; similarly, teams can disband to enable their members to form new teams or to operate as singletons. Thus the team assignment is expected to change over time during the course of a scenario. The team assignments over time and the behavior executed by each team are hidden from our system. We assume that our input consists only of a **spatio-temporal trace**, which is a sequence of noisy observations of the 2D position of each agent through time, $\mathbf{a}_i(t) \in \Re^2$. We illustrate this with an example: Figure 4.1 shows several frames from a scenario with 16 agents. In Figure 4.1(a), 12 of the agents are arrayed in three teams of four agents in a square formation, $(\{a_0, \ldots, a_3\}, \{a_4, \ldots, a_7\}, \{a_8, \ldots, a_{11}\})$, with the remaining four agents as singletons. In Figure 4.1(b), the squares are converging towards the central area and the formations are starting to interleave. In Figure 4.1(c), the squares are disbanding and those are regrouping into four groups of three, arrayed as triangles. Finally, in Figure 4.1(d), the triangles are moving away from the central area. For illustration purposes, observation noise is not shown in this figure.

Our goal is to recover a team and a behavior assignment for every agent $a_i \in \mathcal{A}$ at every time-step t. This can be succinctly expressed in the form of two tables: (1) an agent-to-team assignment $a_i(t) \mapsto \mathcal{S} \subset \mathcal{A}$; (2) a team-to-behavior assignment $S_j(t) \mapsto b \in \mathcal{B}$. It is important to note that one cannot, in general, infer the behavior of a team by examining the motion trace of any single agent. Similarly, one cannot assign an agent to a team



(a) $t = 0$ (b) $t = 24$

(c) $t = 50$ (d) $t = 80$

Figure 4.1: (a) An example scenario with three teams of 4 agents, $(\{a_0, \ldots, a_3\}, \{a_4, \ldots, a_7\}, \{a_8, \ldots, a_{11}\})$ and four singleton agents $(a_{12}, \ldots, a_{15})$; (b) teams maneuver while maintaining formation and converge to central area; (c) the three teams disband and regroup into four teams of 3 agents; (d) the various teams scatter as units. The interleaving of agent formations, the presence of singletons and observation noise (suppressed here) makes the team assignment and behavior recognition challenging.

without confirming that the behavior of the proposed team is legal.

Ideally, one may wish to consider every legal agent-to-team assignment and team-to-behavior assignment at every time-step and then select the sequence that best matches the observed data. However, a straightforward implementation of this idea is computationally infeasible. The pool of potential agent-to-team assignments grows very quickly with the number of agents; this is equivalent to the number of partitions of a set, and is given by the *Bell number* of the set [81]:

$$B_n = \sum_{0}^{n} S(k, n),$$

where $S(k, n)$ denotes a Stirling number of the second kind,

$$S(k, n) = S(k-1, n-1) + kS(k, n-1) \qquad 1 \leq k \leq n$$
$$S(n, n) = S(n, 1) = 1.$$

For instance, the number of team assignments in the 16-agent example shown in Figure 4.1, $B_{16} > 10^{10}$. Clearly, examining every potential team assignment at even a single time-step is infeasible. And naively evaluating all of the possible combinations of partitions over the entire spatio-temporal sequence further increases the complexity in an exponential manner.

Fortunately, a closer examination of the problem reveals structure that can be exploited to generate a computationally-feasible solution. The key observations behind our algorithm are summarized as follows. First, at each time-step, the relative positions of the agents in a team is constrained by the spatial configuration of the formation. Even though it may not be possible to unambiguously determine from a single time-step that an observed subset of agents is arrayed in a particular formation, one can profitably employ a static analysis of agent positions to generate hypotheses of valid team assignments and behaviors. Second, although an analysis of the motion of a single agent may not be suf-

ficient to infer its behavior, an examination of the aggregate movement of several agents in isolation (i.e., a hypothesized team) generates significant information about team behavior. Third, by defining appropriate cost functions for the sequence, one can employ dynamic programming to dramatically reduce the time needed to find good sequences of team and behavior assignments through time. The next section details each of these ideas and describes how they contribute to the design of the STABR algorithm.

4.3 The STABR Algorithm

STABR analyzes spatio-temporal traces in three stages. First, it performs a static analysis of agent positions at each time-step to identify potential agent configurations that may correspond to known formations; these are used as an initial set of agent-to-team assignment hypotheses in later stages. STABR maintains multiple potentially-conflicting assignments for an agent, if there is spatial support. Second, STABR examines hypothesized team assignments in isolation and determines whether they have sufficient local spatio-temporal support. Pruning unlikely hypotheses at this stage is crucial since it greatly affects the performance of the last stage. This analysis also enables STABR to determine plausible behavior assignments for each of the surviving hypotheses. Third, these agent-to-team hypotheses are used to generate complete partitions over the agents. In the worst case, this state space could be exponential in the number of surviving hypotheses, underscoring the benefits of pruning. STABR then organizes the states (partitions) over the spatio-temporal sequence in the form of a lattice and employs dynamic programming to identify minimal cost solutions. These correspond to agent-to-team and team-to-behavior assignments that are a good fit to the observed sequence.

4.3.1 Static identification of agent formations

The first stage of the recognition process is to identify potential team assignments, based on static spatial cues. We do this by matching agent positions to pre-specified geometric formation templates; this enables the recovery of more complicated team relationships than the standard approach of clustering agents into teams based solely on proximity. STABR employs the static formation recognition approach described in Chapter 3 to generate and test potential team assignment hypotheses at selected time-steps. For each formation template, agents are drawn uniformly and at random from both the template and the scenario. These point correspondences are used to generate a transform hypothesis to project the remaining template points into the scenario coordinate frame. If the predicted positions are sufficiently close to the observations, the template is accepted as valid and these agents are assigned to a team. In the example scenario, where 75% of the points are effectively outliers for the square formation, a reliable detection (P=99%) only requires 71 iterations, whereas an exhaustive search through the space would need ${}^{16}C_4 = 1820$ iterations.

4.3.2 Spatio-temporal analysis of individual teams

The first stage of the algorithm identifies, independently for each time-step, a set of hypothetical team assignments. The second stage identifies those team assignments that have significant temporal support, and generates behavior hypotheses for each such team that are consistent with the observed positions. The inability to find a plausible behavior to explain the motion of a hypothesized team is a strong indicator that the hypothesis does not correspond to a real team, but is rather a visual illusion caused by a coincidental configuration of agents.

The behavior recognition proceeds on a team-by-team basis. Each team is independently evaluated over the temporal intervals during which it was detected against a set of parameterized team behavior models. Although STABR could employ arbitrary motion models, here, we model team behaviors as a set of constrained rigid-body transforms, such as "advance" (pure translation in the forward direction), "wheel" (rotation about a forward corner), and "pivot" (rotation about team centroid). If the team's movement does not fit any known behavior, the assignment is discarded since it is likely to have been the result of a spurious detection. This is best illustrated by an example. Consider a team hypothesis that assigns three agents, (a_1, a_2, a_3) to a "triangle" formation. Domain knowledge specifies the set of behaviors that are available to each formation, and this imposes constraints on their observed motion. For instance, a triangle formation may only be allowed to pivot about its centroid or advance as a wedge. The algorithm evaluates the likelihood of explaining the observed team positions $\{\mathbf{a}_1(t), \mathbf{a}_2(t), \mathbf{a}_3(t)\}$, using each behavior, over a sliding time window. Each behavior is characterized by a small number of parameters; given these parameters and an initial agent configuration, the behavior specifies the position of each agent in the formation over the time interval. For example, the pivot behavior for a triangle formation is parameterized by the angular velocity, ω. $\forall i \in \{1, 2, 3\}$:

$$\mathbf{a}_i(t) = \begin{bmatrix} \cos\omega(t-t_0) & \sin\omega(t-t_0) \\ -\sin\omega(t-t_0) & \cos\omega(t-t_0) \end{bmatrix} [\mathbf{a}_i(t_0) - \bar{\mathbf{a}}] + \bar{\mathbf{a}}$$

where $\bar{\mathbf{a}}$ denotes the (stationary) centroid of the triangle formation. Our algorithm attempts to fit the model to the observed data. In this example, if a_1, a_2 and a_3 were really executing a pivot, the algorithm should recover an ω that matches the observations well (in a least-squares sense). On the other hand, if a_1, a_2 and a_3 were actually translating, then the pivot behavior would fail to match (for any choice of ω). Thus, we iterate through each behavior and prune those behaviors that fail to match and also prune those

team hypotheses that cannot be explained by any legal team behavior. The computational benefits of pruning team hypotheses are discussed in the next section.

4.3.3 Explaining sequences of hypotheses

The final stage of STABR searches the space of team assignment and behavior recognition hypotheses generated by earlier stages for a consistent explanation over the entire spatio-temporal trace. In general, there may be several consistent explanations for the given observed agent movements; for instance, it is always possible to explain any trace as a coincidental convergence of uncoordinated singleton movement (though this would be highly improbable). We employ a cost function that formalizes the intuition that an explanation that requires fewer changes in team assignment and behavior is preferable [91]. Given this cost function, we apply dynamic programming over the sequence to efficiently find the minimal-cost solution.

For every time slice, STABR generates a list of potential set partitions from the team assignment labels returned by the RANSAC-based formation matching and validated by spatio-temporal behavior analysis. This list of set partitions represents a potential world state for that time slice; each world state contains a team assignment for every agent such that no agent is assigned to multiple teams (see Section 4.5 for ramifications of relaxing this requirement). Generating a list of consistent world states is still (worst-case) exponential in the number of team assignment hypotheses but this is a dramatic improvement over considering the Bell number of total set partitions at each time step. Thus, effective pruning of team assignment hypotheses using spatio-temporal behavior analysis in earlier stages can greatly reduce running time.

Any sequence through this set of partitions is both consistent (all agents are assigned to teams and no agent is assigned to multiple teams) and supported by local spatio-

temporal evidence. We use a cost function to select the solution that most parsimoniously explains the scenario. This be formulated as a shortest path problem through the space of consistent team and behavior assignments and solved efficiently using dynamic programming. The cost, C_{T-1} of explaining the entire sequence is computed using:

$$C_{t+1}^{q} = \min_{\forall p}\{C_t^p + D_{p,q}\} + \gamma|q|,$$

where p and q are potential world states at time-steps t and $t+1$ respectively, $|q|$ denotes the number of teams in the partition, and γ is a domain-specific parameter that controls the degree to which STABR favors sequences with large teams. D is a "distance" between states that captures differences in both team membership and behavior:

$$D_{p,q} = \sum_{\forall a_i \in \mathcal{A}} \beta I[\mathsf{b}_p(a_i), \mathsf{b}_q(a_i)] + \tau I[\mathsf{t}_p(a_i), \mathsf{t}_q(a_i)],$$

where $I[.,.]$ is an indicator variable; $b_p(a_i)$ and $b_q(a_i)$ return the behavior label for agent a_i under states p and q respectively; $t_p(a_i)$ and $t_q(a_i)$ return the team assignment for agent a_i under states p and q respectively; β and τ are domain-specific parameters that can be tuned to improve recognition accuracy.

4.4 Experiments

We evaluate STABR on a set of scenarios of simulated military formations. The simulator generates traces for the position of each agent, corrupted with iid Gaussian observation noise and emits ground-truth data of the correct team assignments and behavior for the scenario. STABR processes this data and generates a team-assignment and a behavior for each agent, at every time-step. Our evaluation metrics are summarized below:

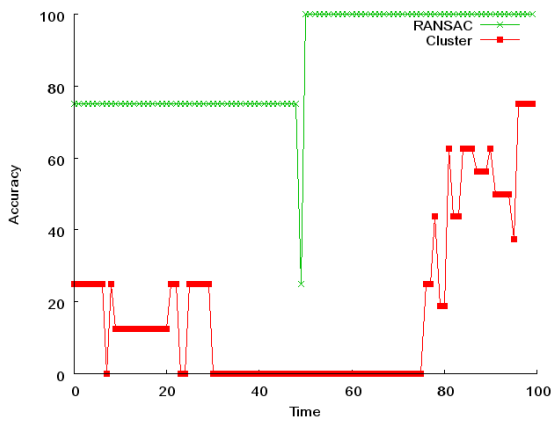
1. **team assignment accuracy**: We score, at each time-step, whether the team assignment for each agent is correct. We employ a conservative metric and require the

team memberships to match exactly; e.g., the absence of a single agent in a k-member team counts as k errors — one for each of the incorrectly-labeled agents — rather than a single assignment error. Team assignment accuracy is plotted over time (Figure 4.2(a)) to show results on a particular scenario and averaged over the scenario to generate aggregate results (Table 4.1).

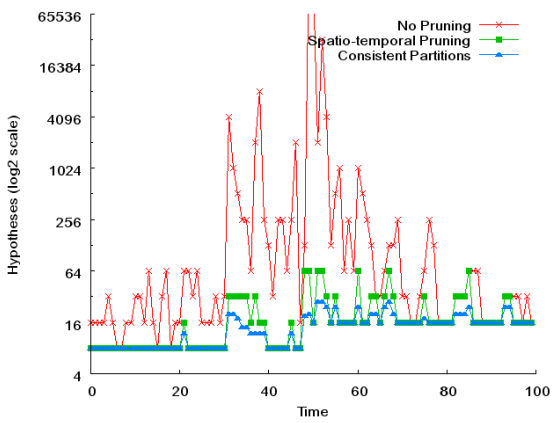
2. **behavior recognition accuracy**: This measures the quality of behavior recognition and is computed in an analogous manner as team assignment accuracy, using the same conservative metric.
3. **hypothesis set size**: We examine the number of hypotheses that are considered by STABR during various stages. This enables us to assess the contribution of spatio-temporal pruning.

Each of the following experiments examines a particular aspect of STABR to better understand its contributions. In all of these experiments, the STABR parameters were set to $\beta = \tau = 1$ and γ=0.1, unless otherwise specified.

The first experiment evaluates the benefits of employing our static formation matching approach against standard proximity-based clustering. K-means and agglomerative clustering are two popular unsupervised clustering methods [28] that are frequently employed to group agents into teams. Since the former requires that the number of clusters be externally-specified, we chose to compare STABR against the latter. In this experiment, the first stage of STABR is replaced with agglomerative clustering, where groups of proximal agents were aggregated into teams. Figure 4.2(a) presents the team assignment accuracy for both algorithms on the scenario shown in Figure 4.1. Agglomerative clustering and RANSAC both perform well when the agent teams are well-separated. However, as the formations begin to interleave, the accuracy of agglomerative clustering deteriorates rapidly. This is because agents that are proximal should frequently be assigned to differ-



(a) team assignment accuracy



(b) #hypotheses

Figure 4.2: (a) Team assignment accuracy for STABR comparing agglomerative clustering with RANSAC on the scenario shown in Figure 4.1. Clearly proximity-based clustering is ineffective when agent formations are in close proximity. (b) Pruning team assignment hypotheses based on spatio-temporal behavior recognition drastically reduces the number of hypotheses that STABR considers. The number of hypotheses that remain after pruning closely follows the actual size of the consistent partitions.

ent teams. The transient drop in accuracy near $t = 50$ corresponds to frames where 12 agents simultaneously transition from three groups of 4 agents to four groups of 3 agents over the span of a few frames; although either assignment would be correct during this interval, the ground truth file arbitrarily selects a single transition point, and STABR's explanation is marked as incorrect. Results on behavior recognition mirror those for team assignment, since correctly identifying an agent's behavior generally requires the algorithm to also group it into the correct team.

Table 4.1 summarizes the agent team assignment accuracy for STABR over several scenarios. While proximity-based clustering can handle the simplest scenario, it copes poorly with the interleaved formations in more complex scenarios.

The second experiment studies the contribution of the spatio-temporal behavior recognition, not in terms of accuracy but rather in terms of reducing the number of hypotheses

Table 4.1: Agent team assignment accuracy, averaged over all agents and time for a variety of scenarios. The benefits of RANSAC over proximity-based clustering are clearly evident. The scenario illustrated in Figure 4.1 is Scenario D.

	Cluster	RANSAC
Scenario A	95.8%	97.8%
Scenario B	57.0%	99.3%
Scenario C	36.0%	99.5%
Scenario D	18.3%	98.5%
Scenario E	0.0%	95.0%

from which world-states need to be generated. Since the execution time of STABR's last stage can grow exponentially with the size of this hypothesis set, it is important to reduce the set of team assignment hypotheses (without jeopardizing accuracy). Figure 4.2(b) shows (in semi-log scale) the size of the hypothesis set before and after spatio-temporal pruning along with the actual size of the consistent set (which is not actually known until stage 3). As can be seen, the spatio-temporal behavior recognition dramatically reduces the number of hypotheses that need to be considered by the third stage — without adversely affecting accuracy.

4.5 Discussion

In military scenarios, group assignment can be quite challenging because modern forces often split into multiple disconnected parts (e.g., far-ranging scouts, small diversion groups, and flanking elements). Recognizing what the force is doing is often possible once it is clear which units are involved. STABR correctly recovers team assignments even in cases

of non-spatially contiguous divisions that foil standard clustering approaches to team assignment.

The STABR approach is based on the following intuitions:

1. Initial agent-to-team assignments can be made on the basis of static spatial cues.
2. The aggregate agent movement for an incorrect team assignment will generally fail to match any behavior model; this can be exploited to prune poor team assignments thus speeding computation.
3. Desirable team and behavior assignments "explain" the activities of a large number of agents over long chunks of the spatio-temporal sequence.

The scenarios presented in this chapter illustrate the operation of STABR in environments that lack the external cues used by other multi-agent plan recognition approaches, such as landmarks, cleanly-clustered agent teams, and extensive domain knowledge. When such cues are available, they can be directly incorporated into STABR, both to improve accuracy and to prune hypotheses. Thus, STABR provides a principled framework for reasoning about dynamic team assignments in spatial domains.

The chicken-and-egg problem of simultaneous team assignment and behavior recognition is conceptually similar to other AI problems, such as image segmentation/object recognition in computer vision. During the image segmentation phase, pixels are assigned to objects that are then classified by an object recognition algorithm. The choices made by segmentation affect the quality of the object recognition; thus one can favor segmentations that generate recognizable objects. In the same way, STABR favors team assignments that produce recognizable behaviors.

Although STABR was designed specifically for the analysis of spatio-temporal traces, we believe that STABR can also be applied to a broader class of problems, where spatial information does not govern team structure. For instance, agents could be assigned to

teams based on observed inter-agent communication patterns in conjunction with role templates that represent functional relationship between agents. In such domains, it may be necessary to relax the restriction on team membership to allow an agent to simultaneously belong to multiple teams. This change would simplify the process of generating valid world states since it removes the need for consistency checking at the expense of increasing the number of potential hypotheses that need to be considered.

This chapter introduces a new algorithm, STABR, that generates both behavior annotations and team assignments from spatio-temporal agent traces. The proposed approach performs accurate team behavior recognition without an exhaustive search over the combinatorial space of potential team assignments. Experiments on several simulated military maneuvers demonstrate that STABR is accurate at both team assignment and behavior recognition.

In the next chapter, we extend these ideas to cases where the sequence of actions performed by each team are generated according to a specified set of plan libraries. We show how this domain knowledge can be exploited to efficiently constrain the plan recognition search space.

Chapter 5

Efficient Plan Recognition for Dynamic Teams

5.1 Introduction

In the previous chapter, we described STABR, an approach for recognizing behaviors in scenarios where team membership is dynamic. STABR focuses on exploiting spatial constraints between agents in a team and validating that the spatio-temporal behavior of the hypothesized team (agent subset) matches valid movement patterns for the team. However, STABR still operates at a lower level than classical plan recognition, where the sequence of actions performed by an agent corresponds to an execution trace through a plan tree. In this chapter, we address the problem of *plan recognition for dynamic teams*, where the actions of the team are derived from an explicit plan and where the composition of agent teams changes according to the specified plan. Although it is possible to apply single-agent plan recognition techniques in such scenarios, 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 [39].

Specifically, 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.

5.2 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 changes 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 through each plan (see Figure 5.1). 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.

5.3 Multi-agent Plan Representation

In this section, we describe our extensions to the hierarchical task network plan libraries generally used for single-agent planning [30]. 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

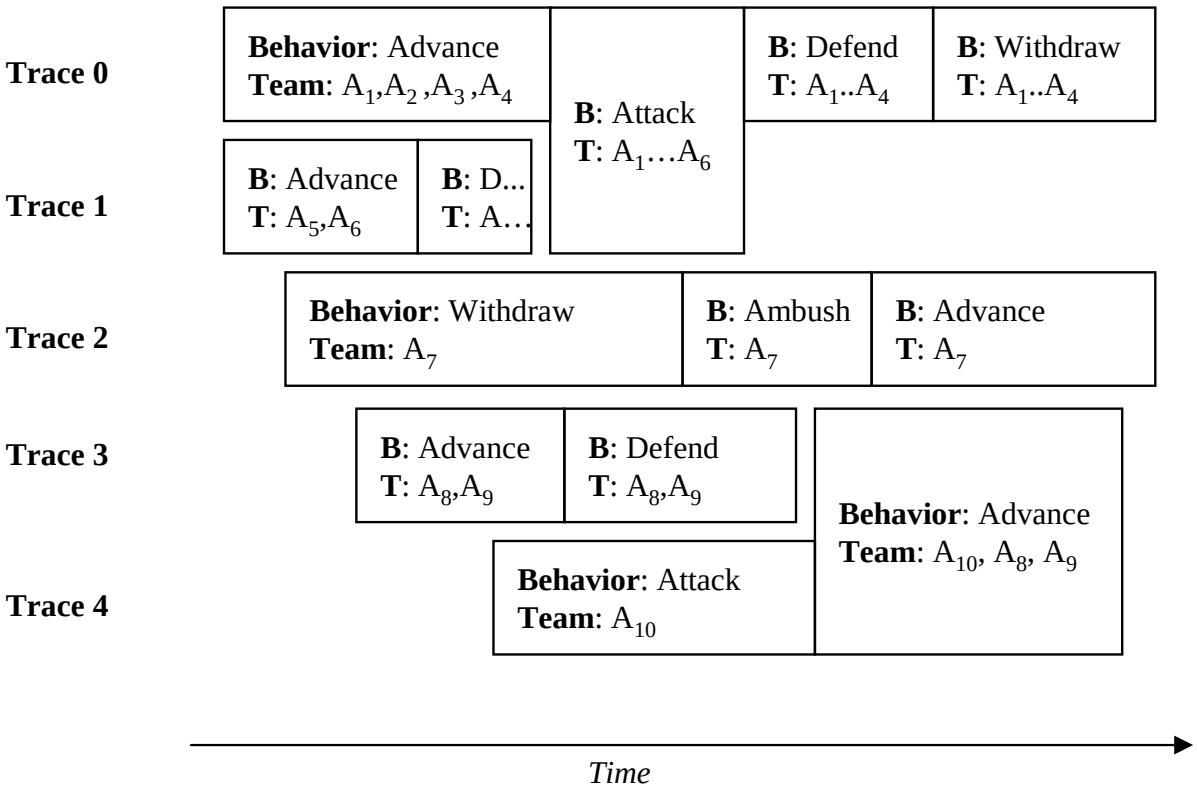


Figure 5.1: Example execution traces, $\mathcal{T} = \{T_0, T_1, \ldots, T_4\}$. Each trace T_i is a temporally-ordered sequence of tuples with observed behaviors, B, and their corresponding agent assignments, $\mathcal{A}$. Note that the composition of the team can change through time as agents join and leave the team. The goal of multi-agent plan recognition is to identify the plan trees that generated these execution traces.

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 5.2 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 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)

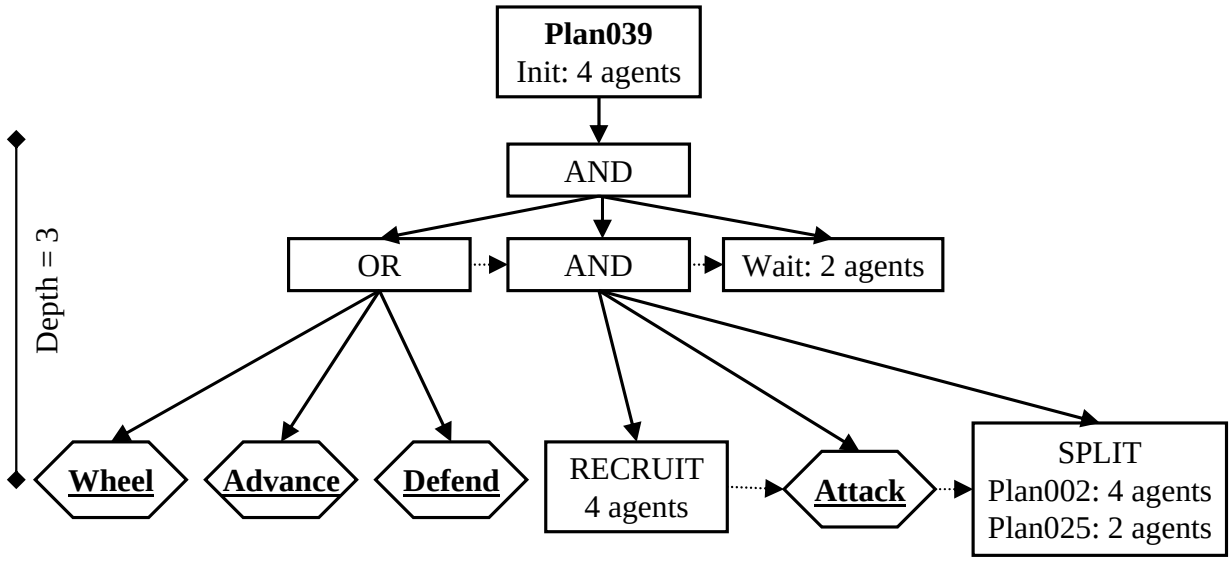


Figure 5.2: Example plan tree. The top node lists the plan library index and the number of agents required (4) to start execution of this team plan. Hexagonal nodes denote directly observable behaviors; square nodes are effectively invisible to an external observer and must be indirectly inferred. The **RECRUIT** node indicates a point where additional agents are needed to continue plan execution (i.e., to execute the **Attack** behavior). The **SPLIT** node denotes a point where the plan requires multiple subteams to execute subplans (002 and 025) in parallel. At the end of the plan, any remaining agents enter the **WAIT** state where they become available for recruitment by new plans.

records the order in which each trace commenced execution but there is no timestamp synchronization across traces.

Kaminka [51] 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; he demonstrates that plan recognition can be used as part of 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.

5.4 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.

5.4.1 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.[1]

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 as described above.

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.

5.4.2 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

[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 behavior.

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 5.3 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 5.4 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.

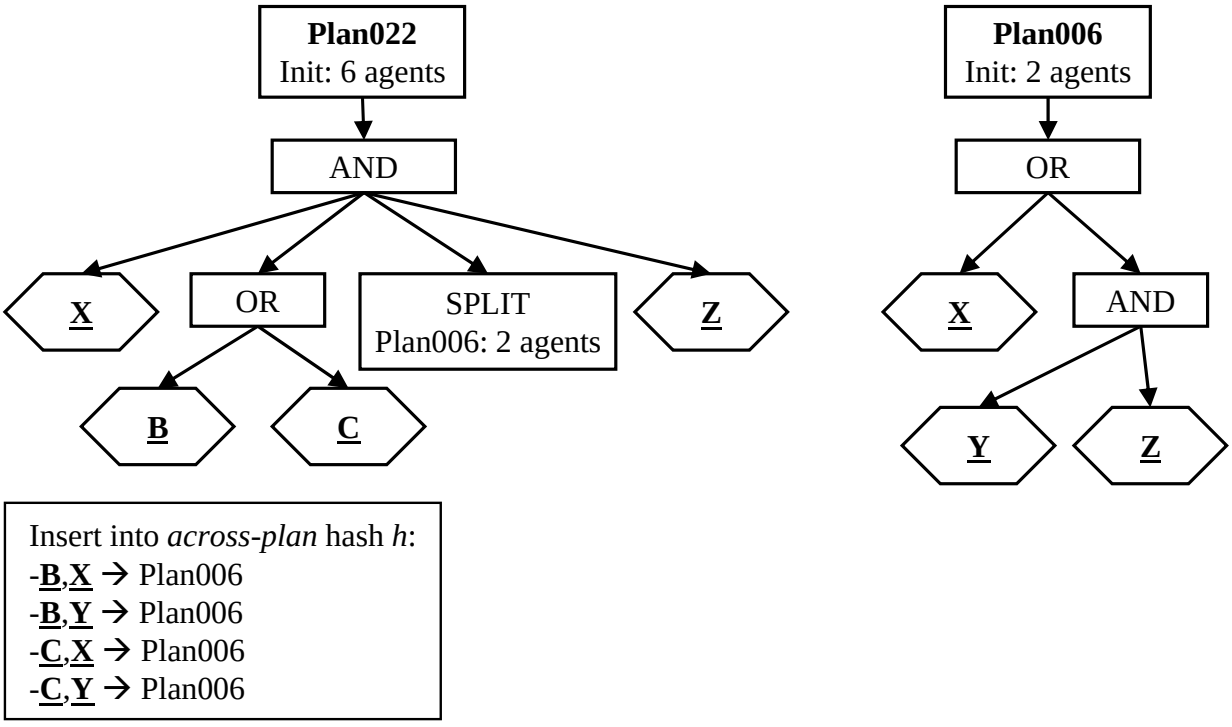


Figure 5.3: Example of across-plan relationships captured by hash h. h captures observable behaviors across a team split. In this case, the **SPLIT** node in the parent plan (022) could be preceded by observation **B** or **C**, while the first step in the subplan (006) will generate either **X** or **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**.

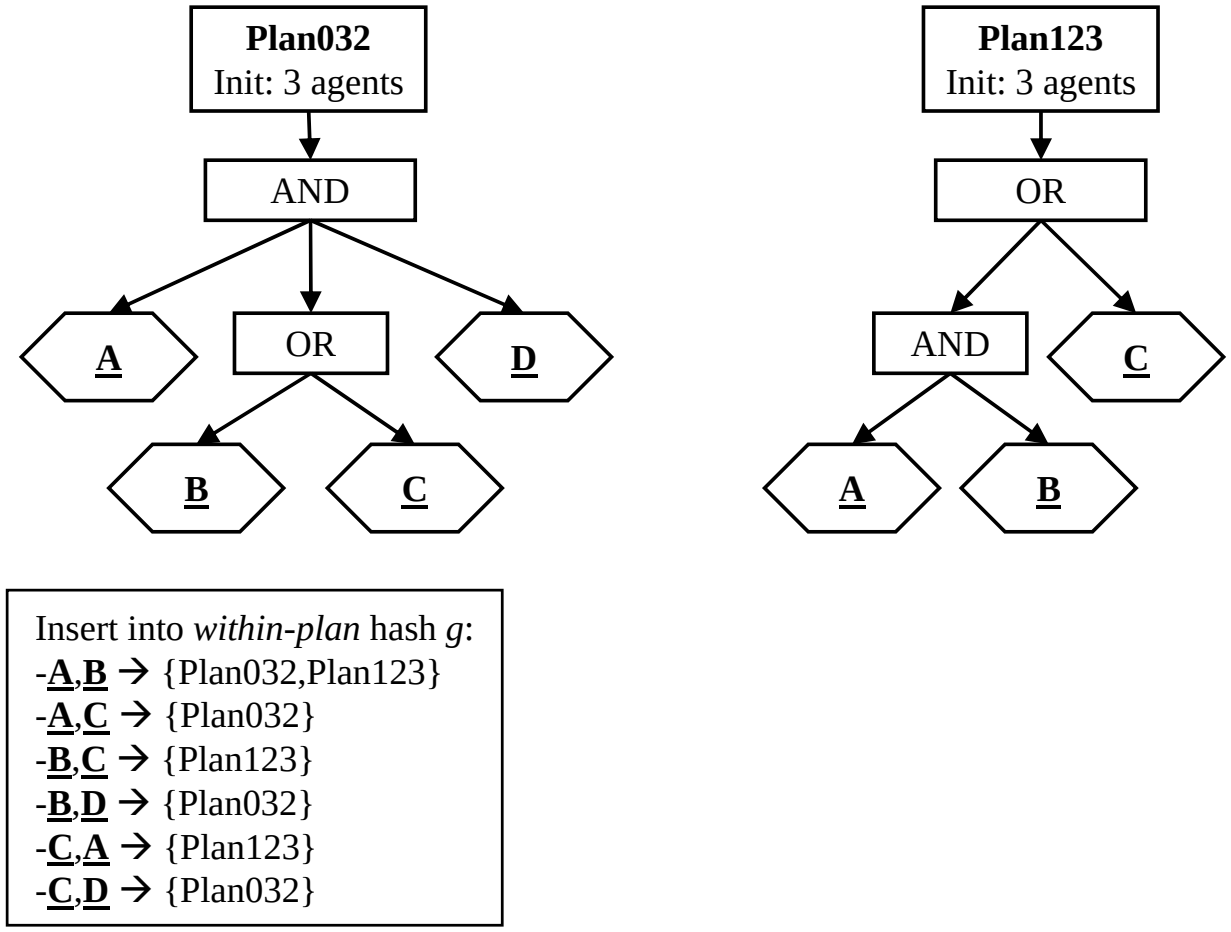


Figure 5.4: Example of within-plan relationships captured by hash g. g captures all plans where two observations can be observed in sequence. For instance, the observed sequence of three agents executing **A**,**B** could be generated by either of the plan trees whereas **A**,**C** could only be the result of Plan032. These temporal constraints can significantly prune the set of possible plan hypotheses.

Applying h requires one lookup per execution trace while g requires a linear scan through the observations.

5.4.3 Analyzing Scenarios with Dynamic Team Membership

This 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 to recover team assignments. In more complicated scenarios, one can use algorithms such as STABR (Simultaneous Team Assignment and Behavior Recognition), described in Chapter 4.

5.5 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.

5.5.1 Plan Library Generation

We follow the experimental protocol prescribed by Avrahami-Zildebrand and Kaminka [5], where simulated plan libraries of varying depths and complexity are randomly con-

structed. 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**.

5.5.2 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.

5.5.3 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 5.1. To reduce stochastic variation, the following graphs show results averaged over 100 experiments.

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

Table 5.1: 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
Observation label size $\|\mathcal{B}\|$	10
Parallel execution traces (average)	12

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 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, we do not claim better precision in general over baseline methods since these cases are infrequent; rather, the primary focus of this chapter 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: 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.

Figure 5.5 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 5.6 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, the pruning approaches significantly reduce 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 5.7 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.

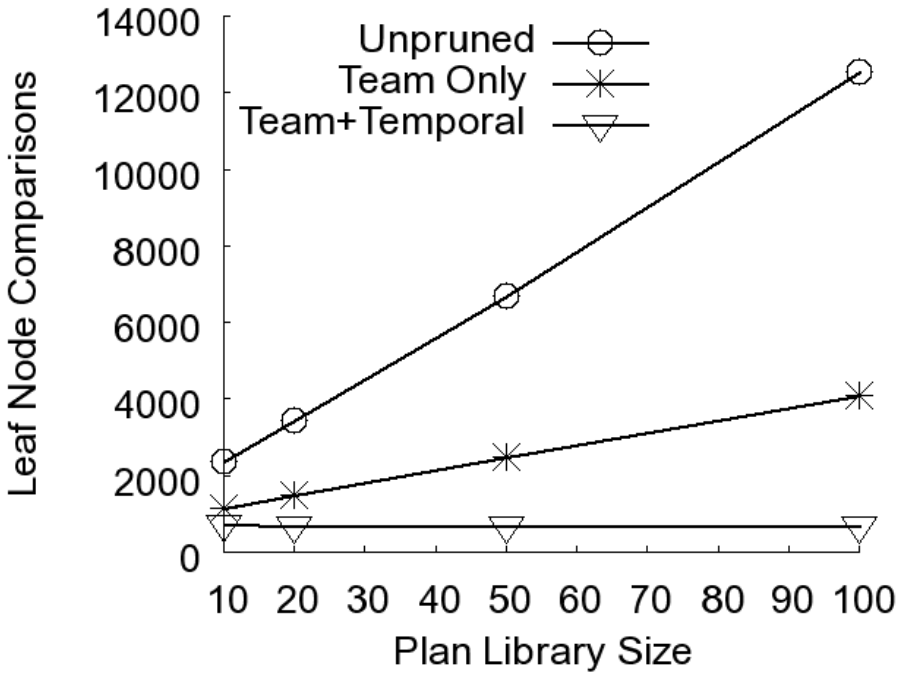


Figure 5.5: Cost of plan recognition time (as measured by leaf node comparisons) with plan library size, $|\mathcal{P}|$.

Figure 5.8 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.

5.6 Discussion

Geib [39] 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

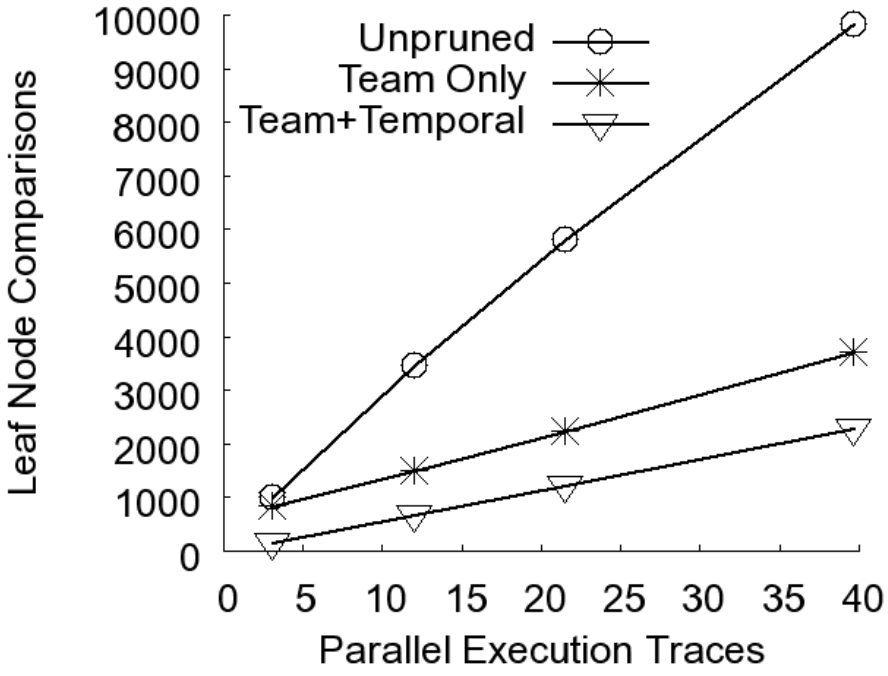


Figure 5.6: Cost of plan recognition time (as measured by leaf node comparisons) with increasing number of execution traces.

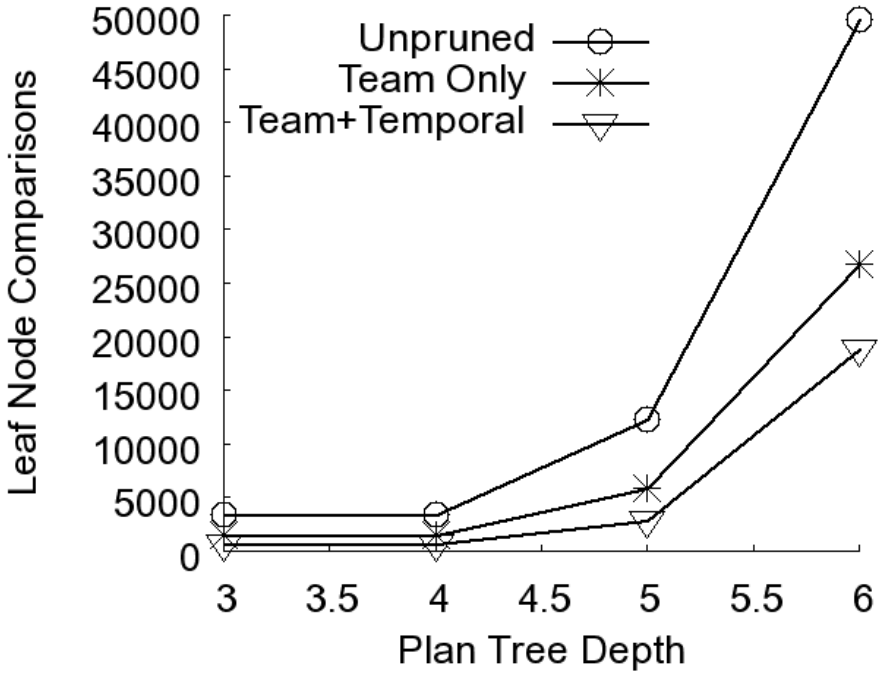


Figure 5.7: Cost of plan recognition time (as measured by leaf node comparisons) with average plan tree depth.

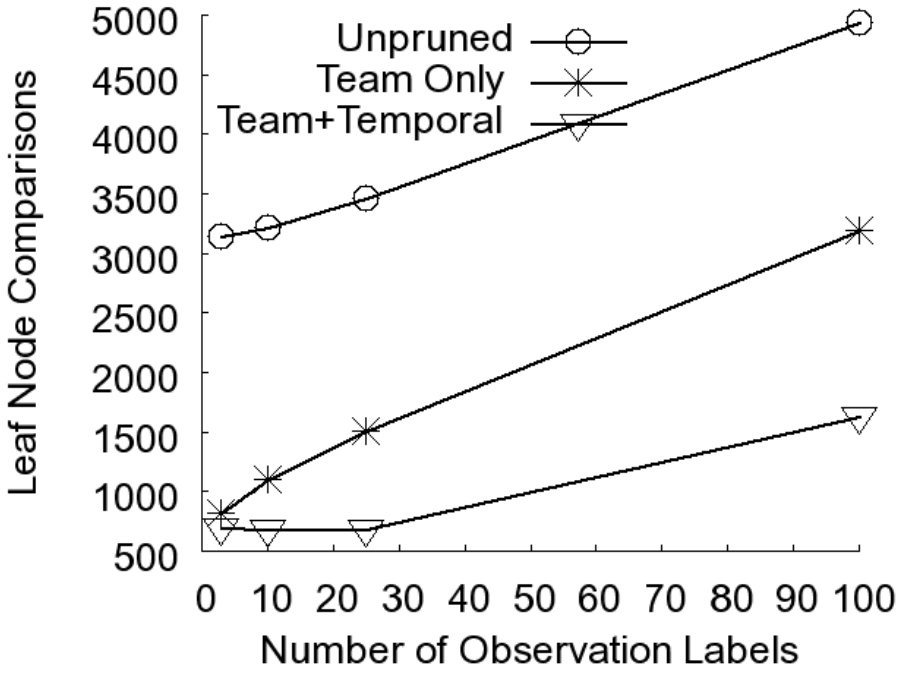


Figure 5.8: Cost of plan recognition time (as measured by leaf node comparisons) with potential observation labels, $|\mathcal{B}|$.

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.

We present 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.

Chapter 6

Analyzing Scenarios with Weak Spatio-Temporal Cues

In previous chapters, we have presented methods that rely on spatio-temporal cues to recognize the behavior of agent teams. However, agent teams in some domains may fail to exhibit strong spatio-temporal cues, or may perform sequences of behaviors that do not map well to traditional plan libraries. This chapter examines some of those issues in the context of multi-player tactical games.

We focus on the problem of analyzing multi-player tactical battle scenarios from game logs. The ability to identify individual and team plans from such observations is important for a wide range of applications including constructing opponent models, automated commentary, coaching applications, and surveillance systems. However, the military adage "no plan, no matter how well conceived, survives contact with the enemy intact" reveals that in many cases team plan execution halts early in the course of battle due to unexpected enemy actions. Of course, an ideal plan would include courses of action for all possible contingencies, but typical battle plans only include options for a small

set of expected outcomes. If the enemy's actions cause the world state to deviate from this expected set, the team is often forced abandon the plan. After multiple plans have been initiated and abandoned, matching the observation trace becomes a difficult proposition, even for an omniscient observer. Moreover, it is unclear whether expert human teams create deep plan trees in situations where enemy actions may force plan abandonment after a few time steps.

Even in cases when the pre-battle plan has been abandoned, we hypothesize that successful teams continue to follow a *policy* through the course of battle and that this policy can be recovered from the observed data. We define a *policy* as a preference model over possible actions, based on the current game state. A *team policy* is a collection of individual policies along with an assignment of players to policies (roles). Policies are typically broad but shallow, covering all possible game states without extending far through time, whereas *plans* are deep recipes for goal completion, extending many time steps into the future, but narrow, lacking contingencies for all but a small set of expected outcomes. The same player intentions can be expressed as either a plan or a policy for game play.

In this chapter, we present two techniques for recovering individual and team policies from multi-player tactical scenarios. Our scenarios are described and played using the Open Gaming Foundation d20 Game System (v3.5) [108]. The d20 System is a set of rules governing combat, negotiation, and resource management employed by popular turn-based tabletop games including Dungeons and Dragons and the Star Wars role-playing game.

The remainder of the chapter is organized as follows. Section 6.1 defines the policy recognition problem, describes the d20 rule system, and presents the battle scenarios that are used by our human players. The game logs from these battles provided the data on which our policy recognition approaches are evaluated. Sections 6.2 and 6.3 present

two complementary methods for policy recognition: an evidential reasoning system for scoring data from game logs and a discriminative classifier trained on simulated game logs. Section 6.4 discusses results in the context of plan recognition, and concludes the chapter.

6.1 Domain: d20 Gaming System

To simulate the process of enemy engagement, we adapted the combat section of Open Gaming Foundation's d20 System (v3.5) to create a set of multi-player tactical scenarios. The d20 System has several useful properties that make it a promising domain:

1. D20 is a turn-based rather than a real-time game system. A game can thus be logged as a sequence of discrete actions performed by each player and policy recognition can be directly executed on these streams of actions and observed game states.

2. The outcome of actions is stochastic, governed by rolling dice (typically an icosahedral die, abbreviated as *d20*). The rules define the difficulty levels for a broad range of combat tasks; to determine whether a particular action succeeds, the player rolls a d20 and attempts to score a number that is greater than or equal to the difficulty level of the task.

3. Spatial arrangements affect the outcome of many of the combat actions, making the difficulty level easier or harder; for instance, two allies on opposite sides of a target gain a mutual benefit for *flanking* the enemy. Thus, the tactical arrangements of units on the grid can significantly influence the course of a battle. Experienced players coordinate such actions to maximize their chance of success.

4. Teamwork between players is of paramount importance. The opposing forces are

> designed to be impossible for a single player to defeat. However, certain game rewards are occasionally awarded to the first player to achieve an objective. To succeed, players must simultaneously contribute to team goals in battle while pursuing their own competitive objectives.

A typical d20 Game is played by a group of 3–6 players with one referee. Each player controls one character within the virtual gaming world that the referee brings to life through verbal descriptions and miniatures on a dry-erase tabletop gaming map (Figure 6.1). Characters have a well-defined set of capabilities that determine their competence at various tasks in the virtual world. The referee controls the actions of all other entities in the virtual world (known as *non-player characters*). During a typical four hour session, the referee poses a series of challenges—diplomatic negotiations, battles, and puzzles—that the players must cooperatively solve. Success for the players enhances the capabilities of their characters, giving them more abilities and resources. Failure can result in the death of characters or the loss of resources Thus, characters' capabilities persist over multiple gaming sessions which makes it different from other iterative games where state is reset at the conclusion of each session.

Although tabletop gaming lacks the audio and visual special effects of its more popular computerized cousins, tabletop games typically have a stronger emphasis on team tactics and character optimization. In battle, time becomes a limited resource and players typically have to coordinate to overcome their foes before the foes defeat them. Since most computer games are real-time rather than turn-based, they disproportionally reward fast keyboard reflexes and manual dexterity over tactical and strategic battle planning.

Figure 6.1: A tactical battle in the d20 System. Players control characters represented by plastic miniatures in a discretized grid environment. Actions with stochastic outcomes are resolved using dice, with probabilities of a successful outcome listed in rulebooks. A human referee (not shown) adjudicates the legality of player actions and controls opponents, such as the dragon.

6.1.1 Multi-Player Tactical Scenarios

Our experiments in policy recognition focus on the subset of the d20 system actions that deal with tactical battles (as opposed to diplomatic negotiation or interpersonal interaction). Each scenario features a single battle between a group of player characters and a set of referee-controlled opponents. The players have limited knowledge of the world state and limited knowledge about their opponents' capabilities and current status; the referee has complete knowledge of the entire game state. Players are allowed to communicate in order to facilitate coordination but the referee will usually disallow excessive communication during battle and limit players to short verbal utterances.

Figure 6.2 shows a typical multi-player tactical scenario from Dungeons & Dragons. The three players must cooperate to achieve one of two objectives: defeat the dragon, or distract the dragon to rescue the prisoner in the corner of the room. Based on the capabilities of their characters, the players select a goal and allocate functional roles for the upcoming battle. Each of the three players was assigned a character originally developed for the 2006 Dungeons and Dragons Open Championships [27]; their capabilities are summarized in Table 6.1. Each character is capable of fulfilling multiple roles in a team but each is poorly suited to at least one role. The roles are:

1. **slayer** who fights the dragon in close-combat;
2. **blocker**, a defensive character who shields more vulnerable characters;
3. **sniper**, who skirmishes the dragon at range;
4. **medic** who restores health to other characters;
5. **scout**, a stealthy character who can rescue the prisoner without being noticed by the distracted dragon.

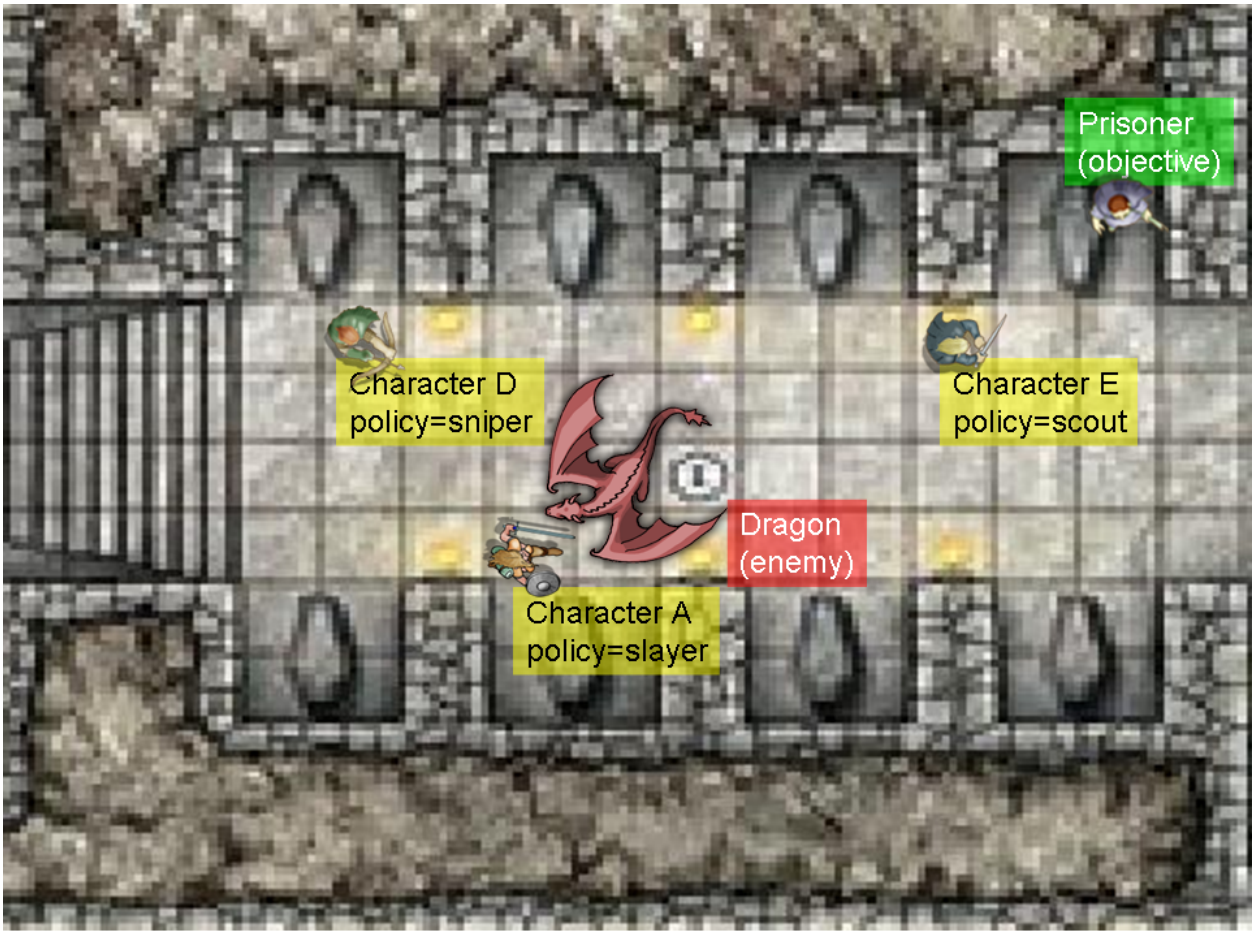


Figure 6.2: Primary tactical scenario. The three human players select roles that enable them to achieve one of two goals: (1) defeat the dragon in battle; (2) distract the dragon while one character frees the prisoner.

Table 6.1: Characters and summarized capabilities

Name	Offensive	Defensive	Magic	Stealth
A	high	medium	low	low
B	medium	high	low	low
C	low	medium	medium	low
D	high	low	medium	medium
E	medium	low	low	high
F	medium	low	high	medium

6.1.2 Game Mechanics

The battle sequence in each scenario is as follows:

1. The referee draws the terrain and places the opponents on the grid at the beginning of the scenario. Then the players place their miniatures on the grid to indicate their starting location.

2. At the start of the battle, each entity (players and opponents) rolls an initiative to determine the order of action each round (time unit). This initiative is maintained through the battle (with a few possible exceptions).

3. Each round is a complete cycle through the initiatives in which each entity can move and take an action from a set of standard actions.

4. If a character's health total goes below 0, it is dead or dying and can no longer take any actions in the battle unless revived by another player's actions.

Each character has the following attributes that affect combat actions: (1) **hit points**: current health total; (2) **armor class**: a measure of how difficult a character is to hit with

physical attacks; (3) **attack bonus**: a measure of how capable a character is at handling weaponry. When a character attacks an opponent, the attack is resolved using the following formula to determine whether the attack is successful:

$$d(20) + \text{attack bonus} + \text{modifiers} \geq \text{armor class} + \text{modifiers}$$

where $d(20)$ is the outcome of a twenty-sided die roll. If the expression is true, the attack succeeds and the opponent's hit points are reduced.[1] Situational modifiers include the effect of the spatial positioning of the characters. For instance, the defender's armor class improves if the attacker is partially occluded by an object that provides cover.

6.1.3 Problem Formulation

We define the problem of policy recognition in this domain as follows. Given a sequence of input observations $\mathcal{O}$ (including observable game state and player actions) and a set of player policies $\mathcal{P}$ and team policies T, the goal is to identify the policies $\rho \in \mathcal{P}$ and $\tau \in \mathcal{T}$ that were employed during the scenario. A player policy is an individual's preference model for available actions given a particular game state. For instance, in the scenario shown in Figure 6.1, the archer's policy might be to preferentially shoot the dragon from a distance rather than engaging in close combat with a sword, but he/she might do the latter to protect a fallen teammate. In a team situation, these individual policies can be used to describe a player's role in the team (e.g., combat medic). A team policy is an allocation of players to these tactical roles and is typically arranged prior to the scenario as a locker-room agreement [89]. However, circumstances during the battle (such as the elimination of a teammate or unexpected enemy reinforcements) can frequently force players to take actions that were *a priori* lower in their individual preference model.

[1]For Dungeons and Dragons, which is based in a fantasy setting, a similar system details how different characters can employ magic to attack opponents and how resistant characters are to the effects of magic.

In particular, one difference between policy recognition in a tactical battle and typical plan recognition is that agents rarely have the luxury of performing a pre-planned series of actions in the face of enemy threat. This means that methods that rely on temporal structure, such as Dynamic Bayesian Networks (DBNs) and Hidden Markov Models are not necessarily be well-suited to this task. An additional challenge is that, over the course of a single scenario, one only observes a small fraction of the possible game states, which makes policy learning difficult. Similarly, some games involve situations where the goal has failed and the most common actions for a policy are in fact rarely observed (e.g., an enemy creates a smokescreen early in the battle, forcing the archer to pursue lower-ranked options). The following sections present two complementary approaches to policy recognition: (1) a model-based method for combining evidence from observed events using Dempster-Shafer theory, and (2) a data-driven discriminative classifier using support vector machines (SVMs).

6.2 Model-Based Policy Recognition

The model-based method assigns evidence from observed game events to sets of hypothesized policies. These beliefs are aggregated using the Dempster-Shafer theory of evidential reasoning [88]. The primary benefit of this approach is that the model generalizes easily to different initial starting states (scenario goals, agent capabilities, number and composition of the team).

6.2.1 Dempster-Shafer Theory

This section presents a brief overview of the Dempster-Shafer theory of evidential reasoning [88]. Unlike traditional probability theory where evidence is associated with mutually-

exclusive outcomes, the Dempster-Shafer theory quantifies belief over *sets* of events. The three key notions of Dempster-Shafer theory are: (1) basic probability assignment functions (m); (2) belief functions (**Bel**); (3) plausibility functions (**Pl**). We describe these below.

The basic probability assignment function assigns a number between 0 and 1 to every combination of outcomes (the power set). Intuitively this represents the belief allocated to this subset and to no smaller subset. For example, after observing an agent's actions over some time, one may assert that it is following either policy ρ_1 or ρ_2, without further committing belief as to which of the two is more likely. This contrasts with the standard Bayesian approach that would typically impose a symmetric, non-informative prior over ρ_1 and ρ_2 (asserting that they were equally likely). More formally, given a finite set of outcomes Θ whose power set is denoted by 2^Θ, the basic probability assignment function, $m : 2^\Theta \mapsto [0, 1]$ satisfies:

$$\begin{aligned} m(\emptyset) &= 0 \\ \sum_{A \subseteq \Theta} m(A) &= 1. \end{aligned}$$

Following Shafer [88], the quantity $m(A)$ measures the belief committed *exactly* to the subset A, not the total belief committed to A. To obtain the measure of the total belief committed to A, one must also include the belief assigned to all proper subsets of A. Thus, we define the belief function Bel : $2^\Theta \mapsto [0, 1]$ as

$$\text{Bel}(A) = \sum_{B \subseteq A} m(B).$$

Intuitively the belief function quantifies the evidence that directly supports a subset of outcomes. The non-informative belief function (initial state for our system) is obtained by setting: $m(\Theta) = 1$ and $m(A) = 0 \quad \forall A \neq \Theta$.

The plausibility function quantifies the evidence that does not directly contradict the

outcomes of interest. We define the plausibility function $\text{Pl} : 2^{\Theta} \mapsto [0, 1]$ as

$$\text{Pl}(A) = \sum_{B \cap A \neq \emptyset} m(B).$$

The precise probability of an event is lower-bounded by its belief and upper-bounded by its plausibility functions, respectively.

We employ Dempster-Shafer theory to model how observed evidence affects our beliefs about a character's current policy. For instance, seeing a character moving on the battlefield could indicate that the agent's role is that of a **sniper**, a **medic** or a **scout** (rather than a **slayer** or **blocker**). This can be expressed as:

$$m(\{\text{sniper,medic,scout}\}) = 0.7 \text{ and } m(\Theta) = 0.3.$$

The **belief** that the agent is adopting one of these roles is 0.7, yet the belief that the agent is specifically a sniper is 0 (although the **plausibility** for either of these is 1). Conversely, while the **belief** that the agent is adopting a slayer policy is also 0, the plausibility is only 0.3.

Dempster-Shafer theory also prescribes how multiple, independent sources of evidence should be combined. Dempster's rule of combination [88] is a generalization of Bayes' rule and aggregates two basic probability assignments m_1 and m_2 using the following formula:

$$\begin{aligned} m_{12}(\emptyset) &= 0 \\ m_{12}(C \neq \emptyset) &= \frac{\sum_{A \cap B = C} m_1(A) m_2(B)}{1 - \sum_{A \cap B = \emptyset} m_1(A) m_2(B)}. \end{aligned}$$

One potential issue with this rule is its treatment of conflicting evidence. The normalizing term in the denominator redistributes the probability mass associated with conflicting evidence among the surviving hypotheses. Under certain conditions, this has the undesirable property of generating counterintuitive beliefs. In the pathological case, an

outcome judged as unlikely by both m_1 and m_2 can have a value of 1 in m_{12} if all other subsets conflict. To address this problem, several other rules of combination have been proposed [86].

Yager's rule [110] is very similar to Dempster's rule with the single exception that conflicting evidence is assigned to the universal set, $m(\Theta)$, rather than used as a normalizer. The rule can be stated as follows:

$$\begin{aligned} m_{12}(\emptyset) &= 0 \\ m_{12}(C \neq \emptyset) &= \sum_{A \cap B = C} m_1(A) m_2(B) \\ m_{12}(\Theta) &\leftarrow m_{12}(\Theta) + \sum_{A \cap B = \emptyset} m_1(A) m_2(B). \end{aligned}$$

Although this formulation is not associative, we implement Yager's rule in a quasi-associative manner to enable efficient online update [86]. In the absence of conflict, Yager's rule gives the same results as Dempster's rule of combination.

Finally, we consider another quasi-associative rule: the intuitive idea of averaging corresponding basic probability assignments [86]:

$$m_{1\ldots n}(A) = \frac{1}{n} \sum_{i=1}^{n} m_i(A).$$

Unfortunately it is impossible to know *a priori* which rule will perform well in a given domain since no single rule has all of the desirable properties.

6.2.2 Empirical Evaluation

Based on domain knowledge, we identified a general set of observable events that occur during a Dungeons and Dragons battle. Each of these events was associated with a basic probability assignment function to assign beliefs over sets of individual policies. For example, the observed event of a character being attacked by an opponent is associated

with: $m(\Theta) = 0.1, m(\{\overline{\text{scout}}\}) = 0.4, m(\{\text{blocker}\}) = 0.5$. This rule assigns a large belief (0.5) to the blocker policy, while reducing the plausibility of the scout policy to 0.1. Note that the set $\{\overline{\text{scout}}\}$ also includes the **blocker** policy, thus the plausibility of **blocker** is 1. The plausibility of the remaining three policies is 0.5.

Data from a series of Dungeons and Dragons games using the tactical scenario shown in Figure 6.2 was recorded and annotated according to our list of observable events. The m-functions for the set of events observed for each character was aggregated using the three rules of combination described in Section 6.2.1.

We computed the average accuracy over the set of battles for each of the three rules of combination. At the conclusion of each battle, the system made a forced choice, for each player, among the set of policies (roles). Each player was classified into the singleton policy with the highest belief. Comparing this against the ground truth and averaging over battles produces the confusion matrices given in Tables 6.2, 6.3 and 6.4. We note that, according to this forced-choice metric, all of the combination rules perform reasonably well, with Dempster's Rule scoring the best. The largest source of confusion is that the **slayer** policy is occasionally misclassified as **blocker**. This motivates the data-driven method described in Section 6.3.1 where we specifically learn classifiers to discriminate between these two similar policies.[2]

To illustrate how belief changes as evidence is aggregated using the different combination rules, we plot the belief for each policy for one battle from our dataset (Figure 6.3). Since neither Yager nor averaging employ normalization, we plot their beliefs on a semi-log scale. We note the following. Yager's rule makes conflicting evidence explicit by allocating significant mass to the **unknown** policy. In particular, this reveals the difficulty

[2]The **blocker** and **slayer** policies can generate very similar observable state since intelligent opponents often target the **slayer** preferentially in order to eliminate their biggest threat.

Table 6.2: Confusion matrix for model-based policy recognition (Dempster)

	sniper	slayer	medic	blocker	scout
sniper	100%	0%	0%	0%	0%
slayer	0%	83%	0%	17%	0%
medic	0%	0%	100%	0%	0%
blocker	0%	0%	0%	100%	0%
scout	0%	0%	0%	0%	100%

Table 6.3: Confusion matrix for model-based policy recognition (Yager)

	sniper	slayer	medic	blocker	scout
sniper	100%	0%	0%	0%	0%
slayer	0%	83%	0%	17%	0%
medic	0%	0%	100%	0%	0%
blocker	0%	0%	0%	100%	0%
scout	0%	0%	0%	0%	100%

Table 6.4: Confusion matrix for model-based policy recognition (Averaging)

	sniper	slayer	medic	blocker	scout
sniper	88%	0%	0%	12%	0%
slayer	7%	40%	7%	39%	7%
medic	0%	0%	0%	100%	0%
blocker	0%	0%	0%	100%	0%
scout	0%	0%	0%	0%	100%

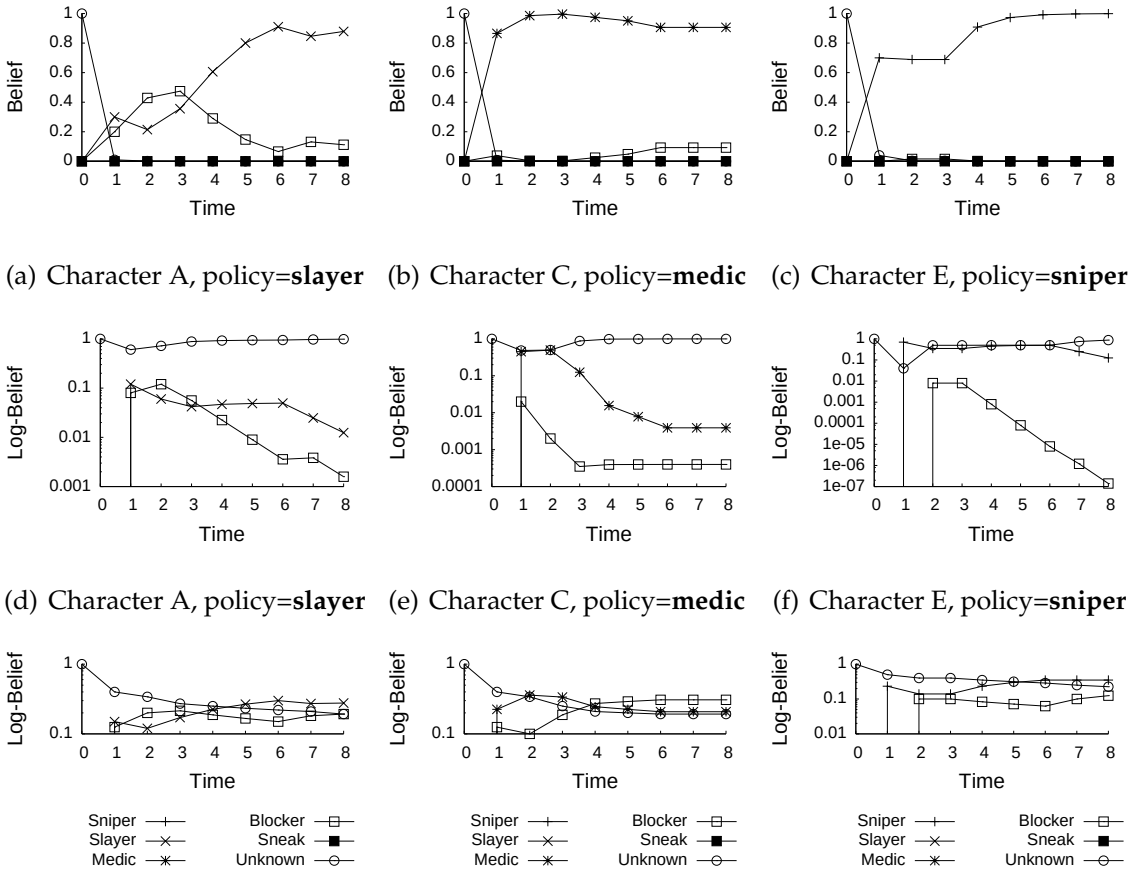
of distinguishing between the **blocker** and **slayer**, even late in the battle. A concern with Yager's rule is that the belief for a policy decays over time, despite increasing evidence because all of the rules leak some mass to the **unknown** set. Averaging corresponding m-values performs the least well in our domain.

6.3 Data-Driven Policy Recognition

To discriminate between similar policies, we propose a data-driven classification method that is trained using simulated battle data. By training the method on a specific scenario, it can exploit subtle statistical differences between the observed outcomes of similar policies. For instance, characters following the **slayer** policy should both inflict more damage on their opponents and receive more damage in return, whereas the more defensive **blocker** policy occupies the enemy without resulting in substantial losses on either side. This section describes the classifier that we employ, Support Vector Machines (SVM), and evaluates the method on a second scenario (Figure 6.4).

6.3.1 Support Vector Machines

The goal of policy classification is to label an observed action sequence as a member of one of k categories (e.g., **blocker** vs. **slayer**). We perform this classification using support vector machines [105]. Support vector machines (SVM) are a supervised binary classification algorithm that have been demonstrated to perform well on a variety of pattern classification tasks. Intuitively the support vector machine projects data points into a higher dimensional space, specified by a kernel function, and computes a maximum-margin hyperplane decision surface that separates the two classes. Support vectors are those data points that lie closest to this decision surface; if these data points were removed



(a) Character A, policy=**slayer** (b) Character C, policy=**medic** (c) Character E, policy=**sniper**

(d) Character A, policy=**slayer** (e) Character C, policy=**medic** (f) Character E, policy=**sniper**

(g) Character A, policy=**slayer** (h) Character C, policy=**medic** (i) Character E, policy=**sniper**

Figure 6.3: Evolution of beliefs over the course of one battle from the scenario shown in Figure 6.2. The beliefs for policies adopted by each of the three characters, according to the three rules of combination, are shown. The three rows correspond to Dempster's Rule, Yager's Rule and m-function averaging, respectively. The columns correspond to three characters A, C, E from Table 6.1, adopting the ground-truth policies, **slayer**, **medic** and **sniper**, respectively.

from the training data, the decision surface would change. Given a labeled training set $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_l, y_l)\}$, where $\mathbf{x}_i \in \Re^N$ is a feature vector and $y_i \in \{-1, +1\}$ is its binary class label, an SVM requires solving the following optimization problem:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i$$

constrained by:

$$\begin{aligned} y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + \mathbf{b}) &\geq 1 - \xi_i, \\ \xi_i &\geq 0. \end{aligned}$$

The function $\phi(.)$ that maps data points into the higher dimensional space is not explicitly represented; rather, a *kernel* function, $K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)\phi(\mathbf{x}_j)$, is used to implicitly specify this mapping. In our application, we use the popular radial basis function (RBF) kernel:

$$K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2), \quad \gamma > 0.$$

Many efficient implementations of SVMs are publicly available; we use LIBSVM [19] because it includes good routines for automatic data scaling and model selection (appropriate choice of C and γ using cross-validation). To use SVMs for k-class classification, we train ${}^kC_2$ pair-wise binary classifiers and assign the most popular label.

6.3.2 Empirical Evaluation

The data-driven method takes as input a feature vector summarizing the observed information about the battle and performs a forced classification into policies. We investigated three choices for feature sets: (1) a histogram of the observed character actions over the battle — this is similar to a bag of words model in information retrieval; (2) a vector with character and enemy status at every time step; (3) a concatenation of these two vectors.

Table 6.5: Confusion matrix for data-driven policy recognition on the scenario shown in Figure 6.4 using the three different feature sets.

	action		state		combined	
	blocker	slayer	blocker	slayer	blocker	slayer
blocker	96%	4%	82%	18%	98%	2%
slayer	0%	100%	16%	84%	0%	100%

To train the SVM, we generated training data by simulating a set of single player battles in a simplified scenario, using the policies of interest (**blocker** and **slayer**). The trained SVM was then evaluated on several other scenarios. Table 6.5 shows the confusion matrices for battles on one of these scenarios. This scenario, shown in Figure 6.4, is a two-player battle where the characters defend a bridge against multiple opponents; the goal is to correctly classify the policy employed by the front-line character. We observe that the data-driven method using the combined set of features can reliably discriminate between the **blocker** and **slayer** policies. The other two matrices indicate that, in this case, the classifier relies mainly on the histogram of observed actions. However, we note that such data-driven methods require sufficient quantities of training data to avoid overfitting, and that they generalize poorly to novel scenarios when such data has not been provided.

6.4 Discussion

One interesting aspect about battles in the d20 System is that a player's action choices are constrained by a complex interaction between the character's capabilities, its location on the map and its equipment (including consumable resources). Hence, different players have different attack options, and each player has different choices over time, depending

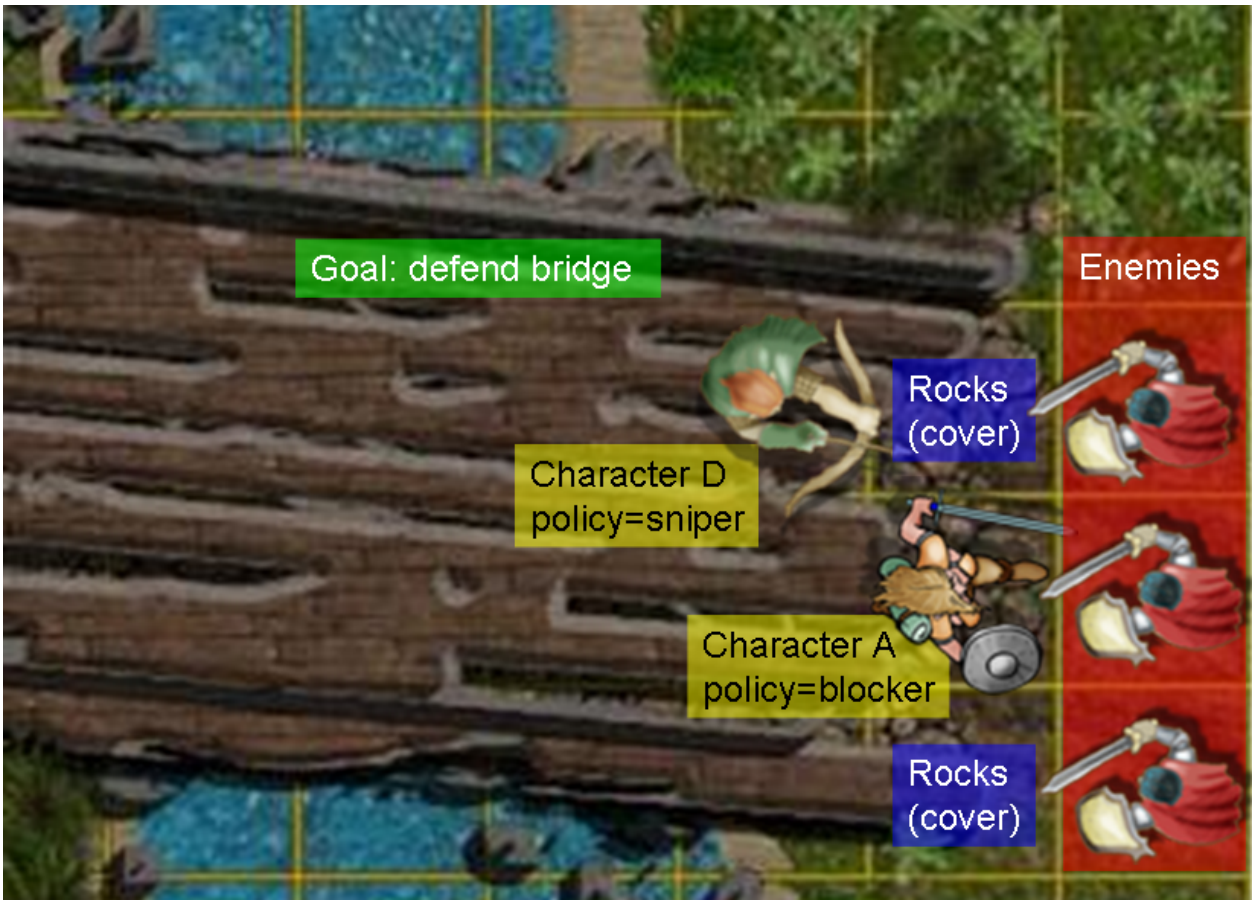


Figure 6.4: Tactical scenario where two players defend a bridge against multiple opponents.

on the current state of the battle. Despite the stochastic nature of the domain, players typically follow battle tactics that are identifiable to other humans. As the human referee controlling the opponents recognizes the players' tactics, he/she will often intelligently adapt to the players' tactics better than computer game engines that are relatively insensitive to player actions. An application of our work would be the development of computerized opponents that react realistically to player tactics to enhance both computer games and military simulations. Despite its fantasy setting with dragons and magic, the d20 tabletop system exercises many of the tactical concepts that current military commanders employ in decision-making games and situational awareness exercises [71].

Teamwork is an important aspect of tactics in the d20 system since the actions of other players can significantly affect the difficulty level of various combat actions. Although building a fully-specified team plan is typically impractical given the complexity of the domain, players generally enter battle with a "locker-room agreement" specifying the policy that the character will seek to use in the upcoming confrontation. There is not a simple mapping between a character capabilities and this policy; an effective team role must consider the capabilities of team members, the expected abilities of opponents and the overall team goal. We believe that identifying each character's policy is an important first step towards predicting the character's future actions, identifying the set of team goals, and generating an automated higher-level commentary on the scenario.

The model-based and data-driven approaches are very complementary approaches to battle analysis. The model-based approach generalizes well to other sets of characters, different opponent types, and variations in scenario. The data-driven classifier is able to detect subtle statistical differences in action and game state sequences to correctly classify externally-similar policies. The data-driven approach does not generalize as well to different character capabilities since a character's capabilities are implicitly incorporated

into the training set; thus a testing set that is statistically-different cannot be classified accurately. We believe that the two approaches should be combined into a hybrid system where the model-based recognizer identifies high-belief policy sets and the data-driven classifier discriminates between those specific policies. Such an approach is similar in spirit to Carberry's work on incorporating Dempster-Shafer beliefs to focus heuristics for plan recognition [17].

This chapter explores two promising approaches for policy recognition: (1) a model-based system for combining evidence from observed events using Dempster-Shafer theory, and (2) a data-driven classification using support vector machines (SVMs). Evaluation of our techniques on logs of real and simulated games demonstrate that we can recognize player policies with a high degree of accuracy.

Using our game logging methodology and domain-generated m-functions, the model-based approach performs extremely well over a broad range of initial conditions. Dempster's rule slightly outperforms the other two rules on the forced classification task. The majority of errors involve confusion between the **blocker** and **slayer** policies, which appear similar at a coarse level. To address this issues, we trained a set of discriminative classifiers using simulated battle logs and evaluated the effects of different feature vectors. The resulting classifiers are highly accurate at classifying these policies, although they do not generalize to characters' with different capabilities. Thus, our two approaches are complementary and could be combined into a hybrid policy recognition system to provide detailed automated battle commentary of multi-player tactical scenarios.

Chapter 7

Activity Recognition from Human Motion Capture

In this chapter, we examine the use of an alternate source of data, human motion capture, for activity recognition. We describe our approach for predicting movements based on the construction of a model of the subject's physical capabilities from motion capture data. Using the physical capability model and the assumption that the human is performing to his/her maximum ability, we can predict opponents' movements, as well as create agents with more realistic motion models. We also discuss an offline approach for recognizing sequences of the subject's actions using cost minimization to select the most parsimonious explanation for the subject's movements. The two approaches are complimentary and suited for different types of applications. In cooperative and competitive domains, it is often important for agents to be able to reason about the future behavior of their fellow agents in an online fashion; this knowledge can be incorporated into the agent's planning to guide its future actions. Offline recognition algorithms are better suited for training environments where they can be utilized for logging and correcting user errors.

7.1 Online Planning and Prediction using a Physical Capability Model

In many domains, our reasoning is constrained and influenced by an internal model of our physical capabilities. For instance, when guarding an opponent in a basketball game, our choice of action depends not only on tactical and strategic concerns, but also on what we believe our abilities to be relative to our opponents' physical speed, agility, and endurance. Here, we address the problem of modeling human physical skills and incorporating this model into the planning process of a humanoid agent, enabling it to predict the effect of its actions and respond realistically to those of its teammates and opponents. We believe that fine-grained modeling of an agent's physical capabilities is an important aspect of planning in sports domains (e.g., basketball, football, hockey) as well as in certain , such as small-unit urban operations. We focus on the following issues:

- modeling humanoid physical capabilities;
- incorporating knowledge of the agent's physical aptitude into planning and opponent modeling;
- constructing agent simulations that enforce a realistic physical capability model.

We believe that these questions are as vital to creating human-like agents in active physical domains as cognitive modeling is for creating agents in more "cerebral" domains.

When selecting actions for agents, the following important question often arises: how much time does it actually take the agent to accomplish each action? For instance, is it faster for an agent to move to a point behind its current position by turning around and running forward, or by moving backward without changing its direction? Either sequence of actions will ultimately get the agent to the same (x, y) location so there is no

reason for a naïve planner lacking a model of human behavior to prefer one plan over the other. *A priori*, the planner might assume that the plan requiring fewer discrete actions (moving directly backward) should be preferred over the slightly "longer" plan (turning, moving forward, turning again), even though for a real human that "shorter plan" would take longer to execute. Human behavior is often asymmetric in a way that computer generated plans are not. Humans have a dominant side (eye, hand, foot) that leads them to perform actions such as manipulating objects, jumping, and kicking in different ways. Similarly, in path planning, humans often exhibit the trait of taking one path from A to B and a different (possibly longer) path from B to A, violating the predictions of typical robot path planning algorithms.

We propose the following general framework for building a cost model of human actions:

1. gather exemplars of domain-specific behavior (e.g., running, dribbling, sneaking) using a human motion capture system;

2. construct a motion graph that enables rapid generation of animation sequences for each behavior;

3. identify an appropriate cost function for scoring candidate animation sequences based on elapsed time and distance criteria;

4. precompute a cost map that expresses the variation in cost for executing a particular behavior;

5. given a set of equivalent goal-achieving behaviors, select the one that minimizes the total cost.

The basic assumption underlying our approach is that motion sequences of behaviors that are implausible or difficult for humans to execute cannot be constructed without incurring a substantial penalty in the cost function. Our method requires a fairly complete basis set of data for every behavior that the agent is allowed to execute, otherwise the cost function will falsely penalize behaviors possible for a human to perform but not well represented in the data set. The remainder of this section presents details about each aspect of the model construction.

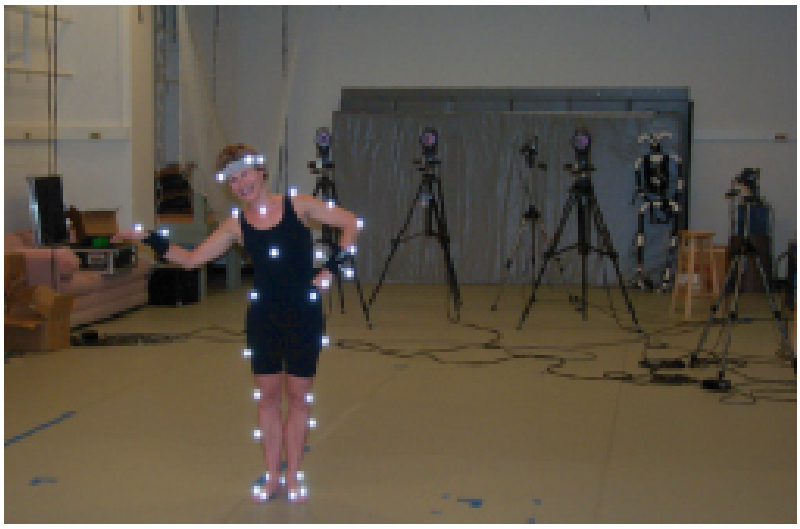


Figure 7.1: A subject wearing a retro-reflective marker set in the CMU Motion Capture Laboratory.

7.1.1 Data Collection

We record human activity using a Vicon optical motion capture system with twelve cameras, each capable of recording at 120Hz, with images of 1000×1000 resolution. We use a marker set with 43 14mm markers that is an adaptation of a standard biomechanical

marker set with additional markers to facilitate distinguishing the left side of the body from the right side in an automatic fashion. The motions are captured in a working volume for the subject of approximately $8' \times 24'$. A subject is shown in the motion capture laboratory in Figure 7.1. The motion is generally captured in long clips (over a minute) to allow the subjects to perform natural transitions between behaviors and is represented by a skeleton that includes the subject's limb lengths and joint range of motion (computed automatically during a calibration phase). Each motion sequence contains trajectories for the position and orientation of the root node (pelvis) as well as relative joint angles for each body part.

We manually annotate the data to label individual domain-specific behaviors, such as walking, probing, inspecting and covering. Because we aim to capture a full spanning set of motion for a particular behavior, our manual labelling is made tractable with long sessions where each take is aimed at capturing the full range of a single behavior. Often capturing multiple behaviors in one take is desirable for future synthesis of natural transitions between behaviors or when a particular domain's fine-grained behaviors are difficult to capture individually. In these cases, semi-automated techniques have been proposed [4] to assist in annotation that require a relatively small portion of the database to be manually labelled.

During a motion capture session, the subject is instructed to perform a set of physical behaviors such as moving around the room, performing a physical skill or manipulating objects. It is important to have the subject perform any variations on a behavior (e.g., picking up an object with the right hand and the left hand) that need to be incorporated in the physical capability model. Directly capturing transitions between behaviors produces higher quality motion data than interpolating motion segments between behaviors. Assessing the final quality of a motion dataset is a difficult problem; Reitsma and Pollard [75]

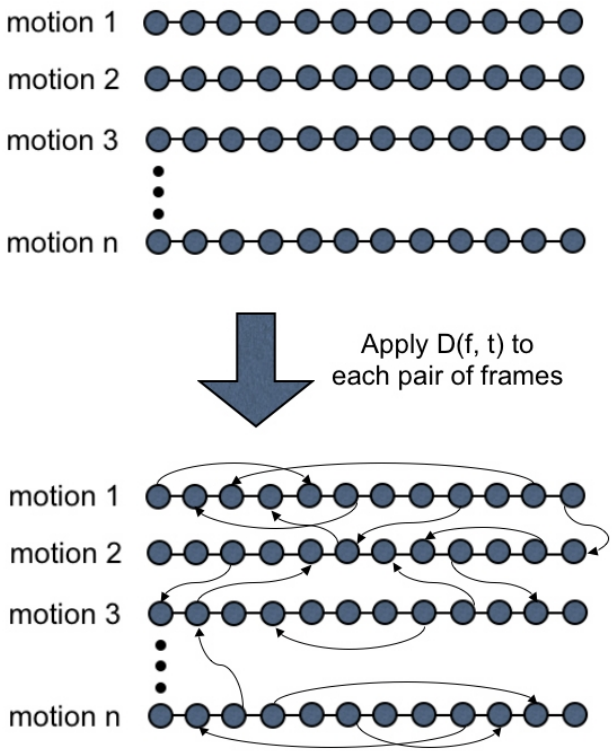


Figure 7.2: The motion graph is constructed by finding the distance between each frame in the database. Edges are added to the graph when $\mathbf{D}(\mathbf{f}, \mathbf{t})$ is below a specified threshold. An edge between two nodes indicates that a transition may be made to smoothly splice the corresponding streams of data together.

describe an algorithm for embedding motion capture data into a target environment and calculating path quality and environment coverage. In practice, it is possible for an experienced technician to evaluate the dataset by visually observing a synthetic character animated using the data and noting dis continuities in the character's motion.

7.1.2 Motion Capture Graphs

To explore the full physical capabilities of a human for a particular behavior in a domain-appropriate, simulated environment, we must move beyond the raw motion data. The

data alone merely allows playback of the subject's performance from the capture session in an environment that is fixed in size and layout. It would be desirable to adapt the captured behaviors to new, more complex environments, in which an animator or computer simulation could intuitively direct the character's actions.

Motion graphs were introduced [3, 56, 59] to provide a solution to this need for control by automatically discovering the ways in which the original data could be reassembled to produce visually smooth motion. Instead of being restricted to a linear playback of the motion clips, the algorithm automatically produces choice points where streams of motion can be smoothly spliced to create novel motion sequences. Individual animation frames act as nodes in the graph, and the choice points act as directed arcs indicating a possible transition between two frames. Next, we discuss (1) how to construct a motion graph and (2) how searching the graph structure enables us to compute a cost for navigating between two points in a simulated environment.

Computing a Distance Metric

The key to building a motion graph is defining an appropriate distance metric between pairs of frames in the database. The metric must ensure that the position and velocity of each body part be sufficiently similar for two pieces of motion to be joined. Since the data is captured in the global coordinate system of the capture area, some care needs to be taken when comparing motion captured in different regions of the space. For many behaviors, the data in the global coordinate system may be translated along the ground and rotated about the human's vertical axis without affecting any important qualities of the motion. An alignment transformation can be computed to create a canonical frame of reference for comparison of pose sequences.

Our distance metric is modeled most closely after the one introduced by [56]. The dis-

tance metric, $D(f,t)$, is computed between frame f and t in the database using the joint positions of the poses in a small transition time window starting at f and t. The purpose of computing the metric over a window of frames is to ensure that velocity mismatches between the two frames are penalized in the calculated metric. A coordinate transformation, $T(f,t)$, is computed to align f and t in the first frame of the window so each pose has matching translation in the ground plane and vertical axis rotation. The metric is computed as follows:

$$D(f,t) = \sum_{i=0}^{\text{WS}} \sum_{j=0}^{J} w_j \left\| p(f+i,j) - (T(f,t)p(t+i,j)) \right\|^2, \tag{7.1}$$

where WS is the size of the transition window, J is the number of joints in the character, w_j allows for weighting the importance of each joint, $p(f,j)$ is the global position of joint j at frame f in x,y,z coordinates, and $T(f,t)$ is the coordinate transformation that maps frame t to frame f.

An edge connecting two nodes (frames) in the motion graph is added whenever the distance between the two frames is below a specified threshold. This threshold may be varied to balance the transition smoothness with the size and versatility of the graph. Typically, it is empirically set so that transitions exhibit no visual discontinuities nor physical anomalies. For rendering at runtime, transitions are blended over a small time window using a sinusoidal "ease-in/ease-out" function to smooth the discontinuity between the two motion streams.

Once the distance metric has been calculated between all frames, we employ a pruning strategy adapted from [56, 59] to remove troublesome edges from the graph; this operation is to avoid the problem of generating paths through the motion graph that cause the character to get stuck in a small subset of the motion database. We remove sinks and dead-ends in the graph by keeping only the largest strongly-connected component (SCC) of the graph, and this can be performed efficiently [101].

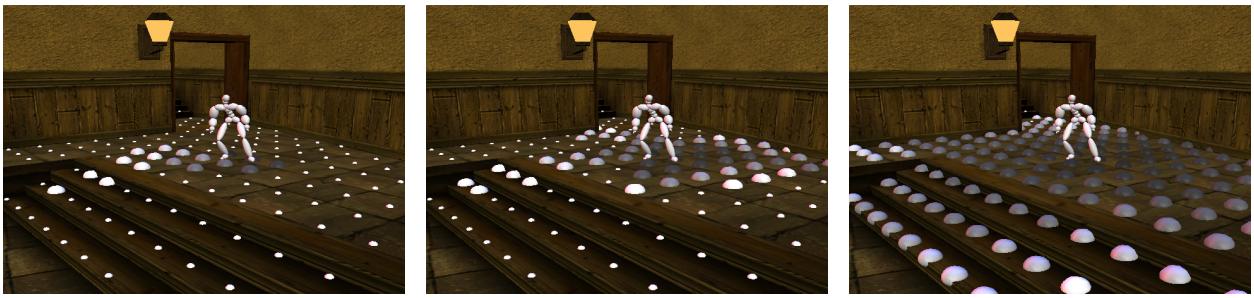


Figure 7.3: The Monte Carlo sampling of paths through the motion graph iteratively approximates the cost map. Light and dark color spheres indicate high and low physical cost according to our metric. The first panel shows a single path through a motion graph while the subsequent panels show how additional iterations contribute to convergence. As yet unexplored regions of the world are indicated by smaller spheres.

7.1.3 Evaluating Actions

We now present a metric to score the cost for a human to move through an environment while performing a particular behavior, where each path is generated by a motion graph. The full set of paths is sampled using a stochastic algorithm that converges on a final cost map for the space. An extension to the algorithm is also presented to handle obstacles that may be present in a real environment.

Scoring Animation Sequences

Given a motion capture graph, we can generate candidate sequences of the character performing each behavior that are visually smooth and lifelike. These sequences, or paths, consist of a series of frames and transitions created by traversing the motion graph. Each sequence represents a possible chain of motions that the human could have executed to reach the goal position. To compute the cost of executing a sequence, we examine two criteria: (1) time cost, which is directly proportional to the path length in frames; (2) goal

achievement—how well the path achieves the desired goal state. The cost metric is

$$C = F + \alpha \|r(x, y) - g(x, y)\|^2, \tag{7.2}$$

where C is cost, F is path frame length, $r(x, y)$ is the character's position, and $g(x, y)$ is desired goal position. The path cost is dominated by F which represents the time cost required for the character to reach a given location; the second term penalizes paths that terminate farther away from the desired goal. Increasing the discretization of the environment reduces the magnitude of the penalty term since all paths that fall within a grid cell are very close to the goal but increases the time required to generate the cost map. The experiments described in this paper were run with $\alpha = 0$. Note that this cost is complementary to the distance metric that we use when assembling the motion graph; because we know that every sequence is guaranteed to be smooth and human-like within a certain threshold, we omit smoothness from our path scoring criterion.

Calculating the Cost Map

To extract the cost of performing a behavior for a given set of constraints, we "unroll" the motion graph to create a cost map over the environment for a given behavior. The map size should be large enough to accommodate the region over which the planner may require cost estimates, and sampled at sufficient resolution to ensure that discretization errors do not eliminate solutions. For example, a map covering a 50'×50' area at a resolution of 10×10 corresponds to a grid with 100 equally-spaced cells, each 5'×5' in size.

The algorithm stochastically samples the set of valid paths to move through the environment using the selected behavior and annotates each cell with the cost of the best path through that cell. Edges in the graph may optionally be disabled if the planner wishes to enforce constraints such as a maximum velocity for the character. The basic steps of the algorithm are as follows:

- Disable edges in the graph according to constraints provided by the planner, eliminating illegal actions.
- Estimate a reasonable maximum search depth of the graph (dependent on desired map size) to bound the search. Paths greater than the estimated maximum depth are not considered.
- Perform a Monte Carlo sampling of the motion path space, updating each cost map cell's score (according to the metric described in Section 7.1.3) along the candidate paths. Random paths are repeatedly generated until the cost map converges to a stable configuration.

We adopted this strategy due to the high branching factor of the motion graph and the necessity to simultaneously evaluate every potential cell in the cost map's space. Since real human movement is highly variable, making early pruning decisions is difficult in general. Exhaustively searching the graph using breadth-first search is prohibitively expensive, and more directed search strategies (e.g., A*) are inappropriate since a search would need to be initiated for each cost map cell. If computation time is limited, the Monte Carlo search also has the desirable property that it may be terminated early to produce an approximation of the final cost map. Figure 7.3 shows a visualization of the search process from within our simulator; Figure 7.4 shows the number of iterations required for cost value convergence.

Our previously computed cost maps are invariant to an agent's position and orientation because they can be embedded with the agent anywhere in an environment. However, they do not reflect the true physical cost for the agent in the presence of obstacles. In our solution, if there are obstacles in the environment, candidate paths that enter obstructed regions are pruned to eliminate physically impossible paths. For a complex environment, the simulator must enforce behavioral constraints to ensure that candidate

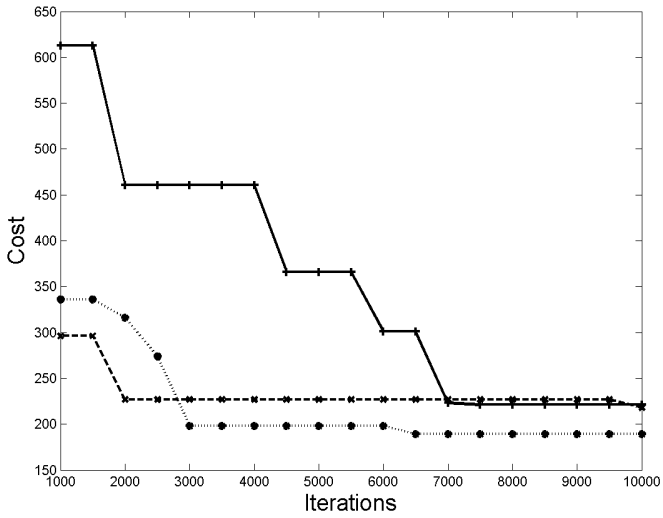


Figure 7.4: Convergence of physical capability model. This graph shows the cost value from three different cells in the cost map generated using the sneak motion graph. We see that the values have converged after 8000 iterations.

paths do not violate the terrain requirements (e.g., chasms must be crossed with jumping or crawlways traversed with crawling). These constraints are checked during the search to eliminate invalid paths.

To perform the Monte Carlo sampling, we selected a reasonable maximum search depth based on the average velocity of the subject. The cost maps used in this paper were constructed with search depths of 500–2000 frames. The algorithm executes a random walk of the motion graph, starting from the character's current pose and orientation. At every choice point in the motion graph, the algorithm stochastically selects a choice point to follow after pruning physically impossible paths. Each grid cell in the cost map is initialized as having cost greater than the maximum search depth; if a cheaper path to that location is discovered the cost of that cell is set to the cost of the path and that path is saved for future use.

7.1.4 Comparative Cost Models for Planning

An important application area for these models is the MOUT (Military Operations in Urban Terrain) domain, where soldiers perform small-unit tactical maneuvers. For our preliminary cost model of the MOUT soldier, we captured data from three human subjects performing various behaviors required for the execution of team tactical maneuvers: walking, running, sneaking, taking cover, rising from the ground, using communications equipment, hand signaling team members, inspecting areas, and probing for booby-traps. Using the human data and the algorithms described in Sections 7.1.3 and 7.1.3, we built a motion capture graph that generates smooth animations of the MOUT soldier performing various behaviors.

In the first scenario, we model the cost of the MOUT soldier running around an obstacle, to determine the tradeoffs between having the soldier run around a hazardous area vs. other plans of action (sneaking or probing for booby-traps). We restrict the animation system to using behaviors labeled with the "running" descriptor and represent the hazardous area as an obstacle to prevent the system from simulating paths that cross the area.

We compare our motion-capture based cost model to the popular distance-based cost model used by grassfire robotic path planners. Figure 7.5 shows the distribution of costs (time) as a function of distance traversed by the agent for the physical capability model. Unlike grassfire, which would always predict that cost is proportional to traversed distance, we see that our model predicts a range of possible times, dependent on the relative position of the goal with respect to the starting pose of the agent.

Figure 7.6 shows a cost map generated using grassfire path-planning [14] on a discretized version of the space, along with the "running" cost map created with the human motion capture data and the cost map of the "sneaking" behavior. The cost map gener-

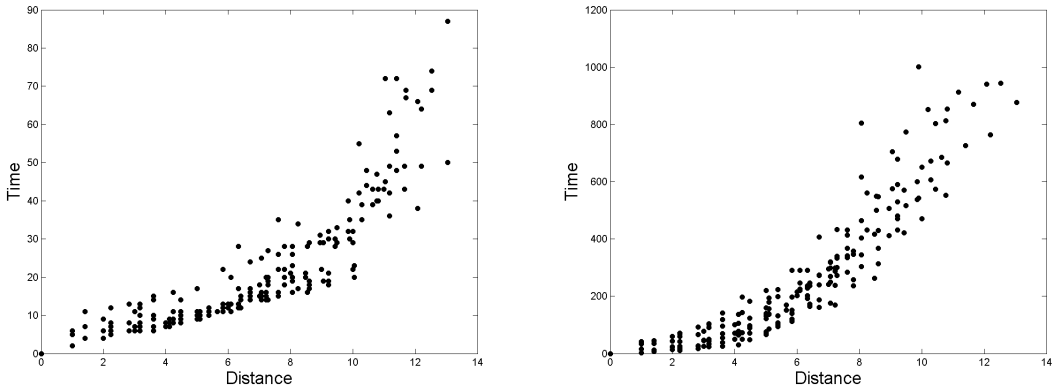


Figure 7.5: Predictions of the physical capability model for running (left) and sneaking (right). The graphs are displayed as distance (feet) vs. time (frames, 1/30 sec). Unlike grassfire, which would predict that the time required to cover a particular distance is constant, we observe that the time predicted by our physical capability model is path-dependent and highly variable.

ated by grassfire is shaded uniformly dark to light with distance from starting position. This reflects a belief that moving in any direction is equally feasible. Unfortunately this approximation is not consistent with human behavior, and agents that use this model are easily anticipated by human opponents.

7.1.5 Opponent Prediction using Physical Capability Models

Not only can the agent improve its own plans with a more refined cost model, but it can also anticipate its opponents' actions using the cost model. We discuss how our model could be applied in the basketball domain for a single agent attempting to bypass a blocker and score. To generate our motion graphs, we captured data from a human subject (male college student) performing basketball maneuvers such as dribbling, pivoting, spinning, and running. The offensive agent is required to use the dribbling behavior to move the ball down the court, whereas the defender can use any form of non-dribbling

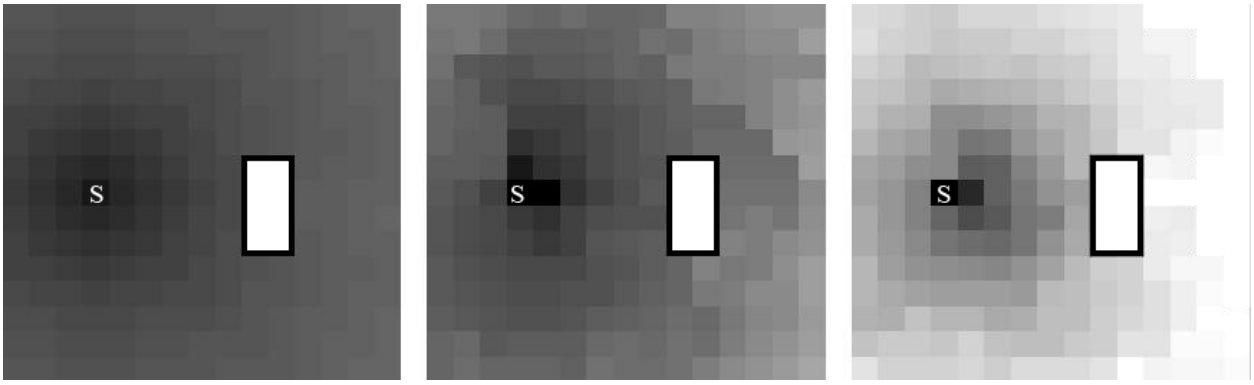


Figure 7.6: Comparative predictions of different cost models in a region with one obstacle. S denotes the character's starting square; the obstructed region is white outlined in black. Costs increase as squares change from dark to light. The left panel shows the simple distance-based model of running generated with grassfire path-planning. The middle panel shows the cost model of the running MOUT soldier generated with human motion capture data. The right panel shows the cost map generated for the sneaking behavior using the motion capture data; the sneaking behavior produces a lighter cost map due to the greater time cost associated from sneaking and is not related to the running cost map in a strictly linear manner. Note the cost asymmetries in the human data, compared the grassfire transform which predicts that moving the same absolute distance in any direction should cost the same.

movement. The offensive agent's goal pose is the shooting stance; it can shoot from any square with a varying chance of success, which we model as being inversely proportional to the distance to the basket. Which location should the offensive agent shoot from? Clearly shooting from its current position minimizes the chance of being blocked by the opponent agent but also has the lowest chance of shooting success since the character's location is far away from the basket. The closer spots raise the chance of shooting success but also increase the risk of being intercepted by the blocker. How can the shooting agent balance the tradeoffs?

One solution to this problem is to have the offensive agent model the time cost for the opponent to reach different potential shooting positions. By comparing this cost to its own cost to reach the shooting position, the offensive agent can balance the risk and reward of various locations. Combined with an estimate of the time required to make the shot, the shooting agent should choose a square such that:

$$C_o > C_d + T_s + \epsilon, \tag{7.3}$$

where C_o is cost required for an opponent to reach the square, C_d is the agent's dribbling cost, T_s is the agent's shooting time (constant for all locations), and ϵ is an error margin. Among the squares that fit this criterion, the offensive agent selects the square that maximizes its shooting success. Cost maps for the offensive and defensive agents are precomputed; the decision can be performed at execution time in $O(n)$ time, where n is the number of squares under consideration.

7.1.6 Simulating Agents with Physical Capability Models

To demonstrate how physical capability modeling can be incorporated within a simulation environment, we extended the Java-based TEAMBOTS [6] framework to enable

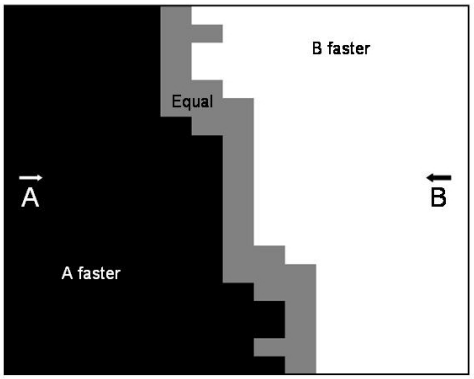


Figure 7.7: By examining the difference between its cost map and the opponent's, the agent can determine which squares it can reach before the opponent. Without prior knowledge, the agent simply models the opponent using the same cost map. This cost map was generated in 200,000 iterations considering paths of 50 frames or less (approximately 2 seconds).

agents to use cost maps in their movement planning process and precomputed motion data paths to generate their locomotion. TEAMBOTS was originally developed as a simulator and real-time robotic execution environment for simulating teams of soccer-playing robots for the Robocup competition. In our experiments, we modified TEAMBOTS to simulate a basketball game and implemented a two-player agent team.

Movement Generation

When an agent initiates a move to a new location, the modified TEAMBOTS simulator uses precomputed paths from the human motion capture data to move the agent between points, instead of the simple kinematic model included in the default TEAMBOTS implementation. Agents plan their moves by checking the precomputed cost maps to find the least-cost goal achieving location. For each location in the cost map, there is a corresponding clip of motion data that represents the shortest plausible motion that a human would use to reach the given location.

Some care is necessary to embed the motion data in the TEAMBOTS simulator for 2D visualization. The original data provides a three-dimensional global position, global orientation, and then relative orientations for the hierarchy of joints that represent human's skeleton. The position and orientations (represented as unit quaternions) are also relative to the coordinate frame of the original motion capture studio. Since the TEAMBOTS simulator is 2D and allocates each agent its own coordinate system, the data must be interpreted in the right way to get proper results. The data is exported from the 3D simulator with all positions relative to the origin, but rotations are left in their original coordinate frame;. To generate 2D motion paths, the data is projected into the ground plane. Joint angles for the skeleton are ignored because they aren't displayed in the TEAMBOTS 2D visualization. In order to render and execute the paths at runtime, the positions must be scaled to fit the game units in TEAMBOTS and a coordinate transformation must be applied to align the orientation with the steering angle of the agents. Each time an agent selects a move action, the coordinate transformation is computed to rotate the original motion data into the coordinate system of the moving agent. For playback, the Gamebots global timer is used to enforce a playback speed of 30 frames per second to match the original data. Figure 7.8 shows an example of a "spin" maneuver being simulated within our modified TEAMBOTS implementation.

Agent Strategies

Using our modified TEAMBOTS simulator, we developed a simple basketball team consisting of two offensive and one defensive players to illustrate the application of our physical capability model to planning and opponent modeling. Focus was given to providing strategies for the offensive players; the defensive agent simply tries to move toward the ball. The offensive agents can select from the following four actions:

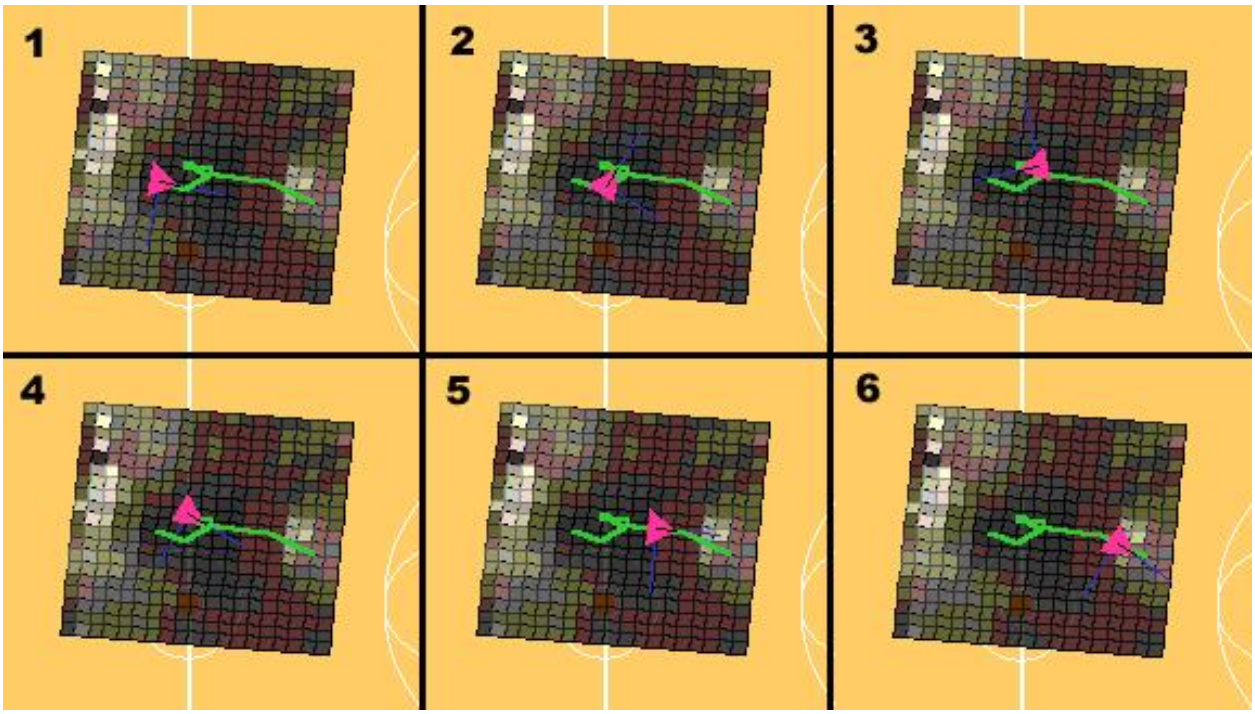


Figure 7.8: A “spin” maneuver executed within the TEAMBOTS simulator. The green pixels represent the full path taken by the agent. The blue vectors represent the agent’s current 2D coordinate frame. A coordinate transformation is applied to the motion data to align it with the agent’s initial steering angle.

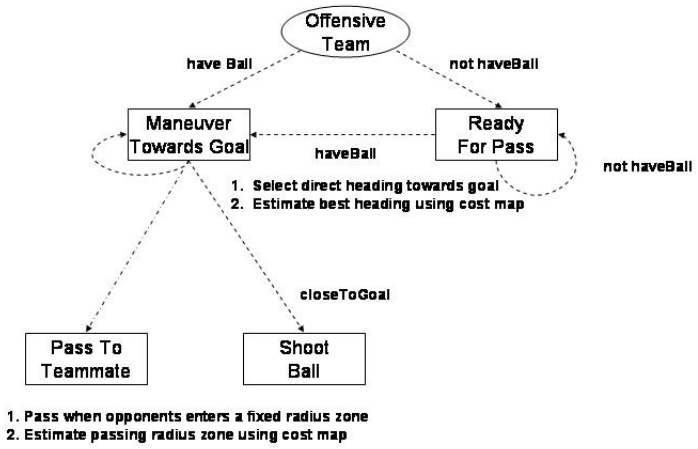


Figure 7.9: Conditional plan used by offensive agents. **Maneuver Toward Goal** and **Pass To Teammate** have two options, depending on whether the agent is utilizing its physical capability model.

Maneuver Toward Goal: The simpler model simply moves straight toward the goal with a constant velocity while the cost map version uses cost map to calculate the cheapest heading and uses motion data to move.

Ready For Pass: Also moves toward goal but also actively tries to avoid the teammate to remain open for a pass.

Shoot Ball: Shoots the ball at the basket.

Pass To Teammate: The simpler model passes when the defender is within a fixed radius while the cost map version estimates the opponent's arrival time using the technique described in Section 7.1.5.

Figure 7.9 displays the conditional plan followed by the offensive players.

Discussion

We compare our framework to related approaches that have emerged from different research communities:

- Many simulation agents use motion planning algorithms derived from the robotics community to produce rapid, optimal paths. Unfortunately, the resulting motion is not very human-like, and opponent agents created with this approach can be predictably outwitted in certain domains. Our method creates asymmetric cost models that are less easily anticipated by human trainees.

- The computer game industry has developed simple methods for parameterizing the behavior of AI bots using a small set of qualities such as speed and aggressiveness. Although this is a useful way to quickly create a large population of bots, the bots lack variability, since there are only a small number of attributes that can be tuned to adjust behaviors.

- Biomechanical data has been collected for a limited set of endeavors but is not easily incorporated into decision-making algorithms without a model that predicts how changing low-level parameters affects high-level behaviors (dribbling, sneaking). Our model is easily incorporated into planning or learning algorithms since we directly compute the effects of using the high-level behavior on the world.

Our technique requires collecting a complete set of basis behaviors for each domain to be modeled, since data omissions can be interpreted as higher cost regions in the cost map. Also our method does not model phenomena such as fatigue or injury unless the modeler explicitly introduces them as separate behavior states. Like many exemplar-based, non-parametric models, the data collection costs could be quite high; we envision our framework only being used to model a small set of behaviors of high importance.

The cost function used in this research is primarily based on minimizing behavior times; other types of cost functions (e.g., smoothness) have also been used to model human motor functions [103]. We believe that our algorithms could be adapted to different cost functions; since our primary concern was to model time-stressed human behaviors, the use of a minimum time criterion was appropriate for our simulation.

7.2 Cost Minimization Approach to Human Activity Recognition

This section presents a offline approach to human activity recognition—instead of modeling the physical capability of the subject, we attempt to classify the behaviors that a human subject is performing from a set of known military maneuvers. Low-level motion classification is performed using a support vector machine (SVM); output from the classifier is used as an input feature for the behavior recognizer. Given the dynamic and highly reactive nature of the domain, our system must handle behavior sequences that are frequently interrupted and often interleaved. To recognize such behavior sequences, we employ dynamic programming in conjunction with a maneuver transition cost function to efficiently select the most parsimonious explanation for the human's actions.

We focus on the problem of recognizing the behavior of individual MOUT soldiers in the context of physical actions and spatial environment features. MOUT soldier behaviors include scouting, ambushing, retreating, searching for hazards, and clearing rooms; we also model less structured behaviors appropriate for civilians in combat zones such as fleeing and hiding. Our representation for single soldier behaviors in the MOUT domain includes physical actions, environmental features, states, and state transitions. Behaviors are described as directed acyclic graphs as shown in Figure 7.10.

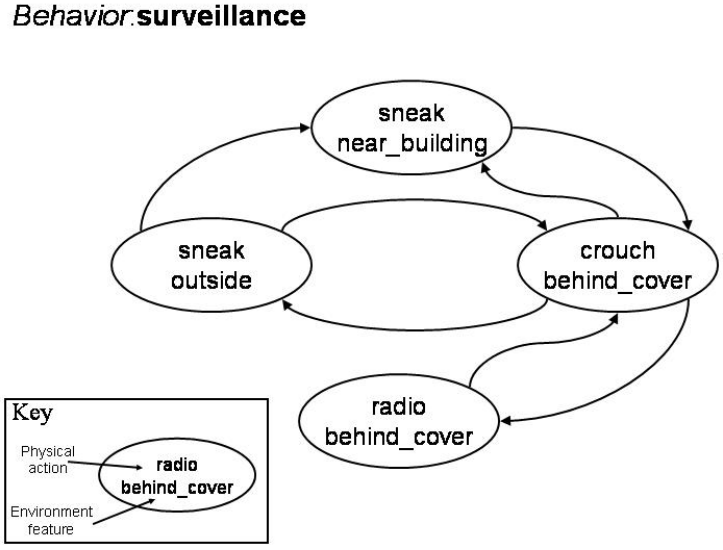


Figure 7.10: MOUT Behavior Representation: Surveillance. The *surveillance* behavior is used when a soldier wants to examine a building from the outside in preparation for entering the building, either as part of an attack or a building clearing operation. Any of the states listed in the diagram are valid starting points for the behavior; there is no single state transition that must always occur at the start of every surveillance behavior.

- *Physical actions* { walk, run, sneak, probe, wounded_movement, rise, crouch } are physical movements (Figure 7.12) classified from human motion capture traces using a support vector machine (SVM) action classifier as described in Section 7.2.3. Shoot and radio are special physical actions with auditory effects that are assumed to be reliably detected without relying on the action classifier.

- *Environmental features* { NEAR_HAZARD, BEHIND_COVER_INT, BEHIND_COVER_EXT, NEAR_CROSSING, NEAR_INTERSECTION, IN_CORRIDOR, IN_STREET, IN_ROOM, NEAR_INT_DOOR, NEAR_EXT_DOOR, IN_BUILDING, OUTSIDE } are derived directly from the simulator based on the human's (x, y) location as described in Section 7.2.6 and are assumed to be reliable.

- *States* are represented as a combination of the 9 recognized human actions with the 12 environment features (108 possible states).

- *Observation traces* are sequences of observed state transitions; since all of our states are self-connecting, only transitions between different states are recorded in this trace.

Human *behaviors* are represented as directed acyclic graphs of states, similar to the representation commonly used for robotic behaviors. For constant time retrieval efficiency, this is implemented as a hashtable that maps state transitions to the set of behaviors consistent with the given observation. For the MOUT soldier domain, we created a library of 20 behaviors including ambush, bounding, scouting, and guarding. The behavior author need not explicitly describe every state transition in the graph; the system will automatically expand general feature descriptions into a complete set of legal state transitions for the specification. For instance, the surveillance behavior (Figure 7.10) includes the description *sneak* OUTSIDE which could refer to many possible states (such as *sneak* IN_STREET or

sneak IN_BUILDING). All of the transitions between these expanded states are automatically generated when the behavior library is compiled.

7.2.1 Method

System Architecture

The purpose of our system is to recognize and produce an accurate description of physical *behaviors* performed by a single human subject engaged in a MOUT scenario. The human's *physical actions* are recorded using a motion capture system as described in Section 7.1.1 and classified by our classifier (Section 7.2.3). These actions are used by an environment simulator (Section 7.2.6) to generate *state* transitions; these sequences of state transitions, or *observation traces*, are used as the input to the behavior recognition system. To better understand the effectiveness of the individual components we decompose the process into two separate tasks, physical action classification and behavior recognition, and evaluate them separately.

Procedure

The human subject is instructed to perform a sequence of physical actions in the motion capture lab while wearing a retro-reflective marker set (see Figure 7.1). This produces a stream of high dimensional data describing the human's trajectory over time while performing specified physical actions. This data is collected and processed offline to produce training and test set for our action classifiers; to improve the classification performance we reduce the dimensionality (Section 7.2.2). Pairs of reduced-dimension motion capture frames are used as input for our action classifiers (Section 7.2.3).

To test our behavior recognition, we use a simulator (Section 7.2.6) that generates envi-

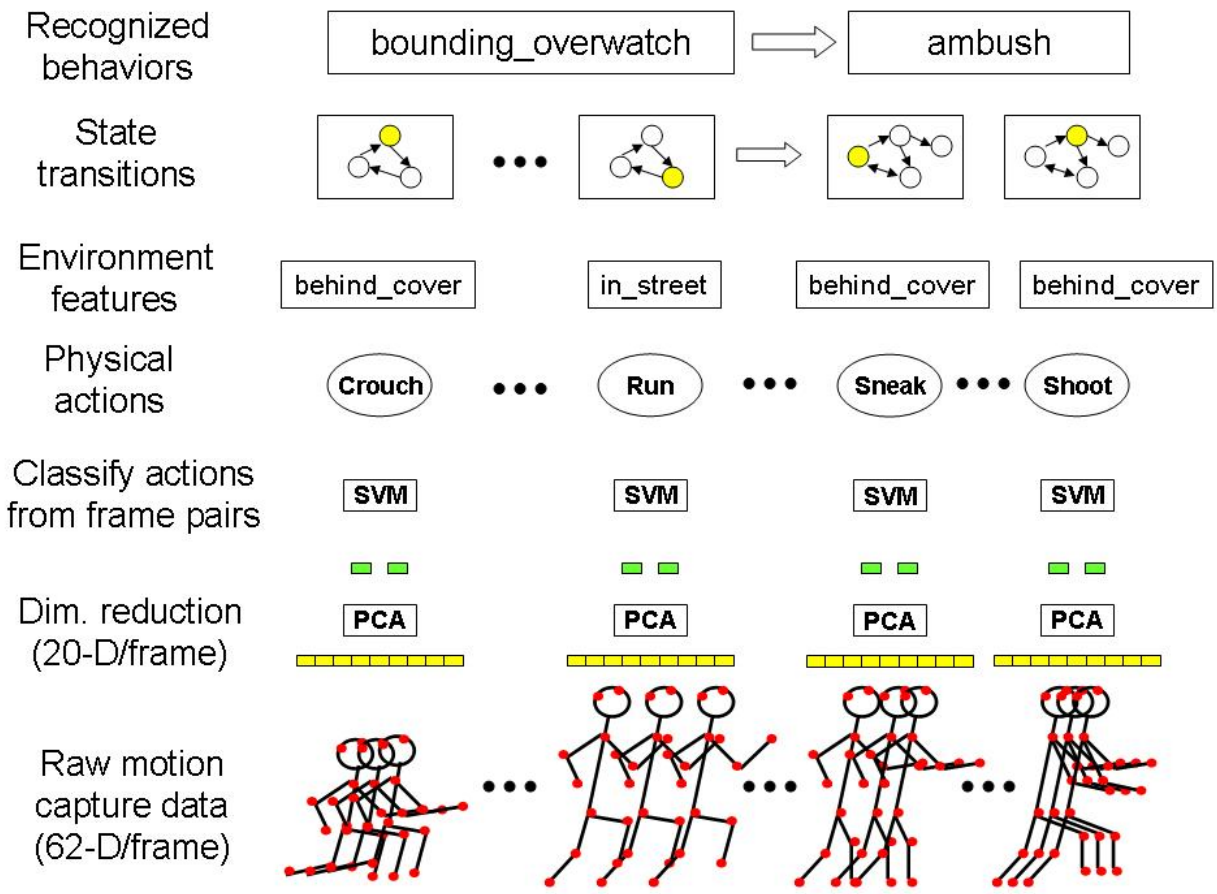


Figure 7.11: System diagram. The purpose of our system is to recognize and produce an accurate description of physical *behaviors* performed by a single human subject engaged in a MOUT scenario. The human's *physical actions* are recorded using a motion capture system as described in Section 7.1.1 and classified by our classifier (Section 7.2.3). These actions are used by an environment simulator (Section 7.2.6) to generate *state* transitions; these sequences of state transitions, or *observation traces*, are used as the input to the behavior recognition system.

ronmental features such as NEAR_BUILDING, NEAR_DOORWAY to supplement the motion capture data. Our assumption is that such features can be generated trivially from knowledge of the subject's (x, y) location (given by motion capture) and the simulated scenario environment. We synthesize observation traces in the simulator which serve as input to the behavior recognition (Section 7.2.5).

7.2.2 Dimensionality Reduction

To improve the robustness of action classification and prevent overfitting, we preprocess the motion capture data as follows. First, in order to make our action classifier invariant to global position, we transform the motion capture data to position the root node at the origin. Second, we eliminate trajectories of the minor appendages (fingers, thumbs, and toes) which are noisy and unimportant for the MOUT domain. Finally, we use Principal Components Analysis (PCA) [28] to reduce the pose vector to a manageable size, as described below. PCA has been employed in many applications, including the segmentation of motion capture data sequences [8].

PCA, also known as the discrete Karhunen-Loève transform, is an optimal linear method for reducing data redundancy in the least mean-squared reconstruction error sense. Using PCA, points in $\Re^d$ are projected into $\Re^m$ (where $m < d$, typically $m \ll d$). The intuition is that many real-world high-dimensional data sets can be well-approximated by lower dimensional manifolds embedded in the original space. We believe that the intrinsic dimensionality of poses relevant to our domain is much lower than the raw pose vector generated by the motion capture system. Each raw motion capture frame can be expressed as a pose vector, $\mathbf{x} \in \Re^d$, where $d = 56$. This high-dimensional vector can be approximated by the low-dimensional feature vector, $\theta \in \Re^m$, using the linear projection:

$$\theta = \mathbf{W}^T(\mathbf{x} - \mu), \tag{7.4}$$

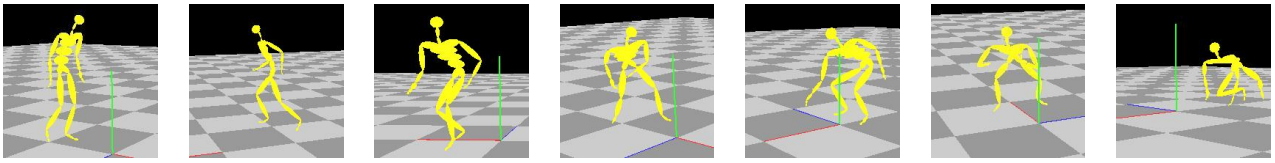


Figure 7.12: Representative poses for the subject's physical actions.

where $\mathbf{W}$ is the principal components basis and μ is the average pose vector, $\mu = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$. The projection matrix, $\mathbf{W}$, is learned from a training set of $N = 16,055$ frames of motion capture data, spanning the set of physical actions in our domain. $\mathbf{W}$ consists of the eigenvectors corresponding to the m largest eigenvalues of the training data covariance matrix, which are extracted using singular value decomposition (SVD). This reconstruction is theoretically perfect only when $m = d$; however, in our application, $m = 20$ produces reconstructions that are visually indistinguishable from the raw data (these components account for more than 95% of the energy in the data). The principal components are only computed once, in an off-line phase; dimensionality reduction of incoming motion capture frames is achieved by the efficient linear projection described by Equation 7.4.

7.2.3 Physical Action Classification

The goal of physical action classification is to label a short sequence of frames as a member of one of k categories (e.g., run, sneak, radio). We perform this classification using support vector machines [105] (see Section 6.3.1 for a description of SVMs). Many efficient implementations of SVMs are publicly available; we use LIBSVM [19] because it includes good routines for automatic data scaling and model selection (appropriate choice of C and γ using cross-validation). To use SVMs for k-class classification, we train ${}^{k}C_2$ pairwise binary classifiers and assign the most popular label.

After applying PCA to the raw motion capture data, the human's pose in each frame is represented by a 20-dimensional vector. To train our action classifier, we form 40-dimensional vectors by concatenating two pose vectors separated by $1/3$ second (10 frames). Our accuracy using a single frame without concatenation is about 20% worse; intuitively certain action classifications (e.g., walk vs. sneak) are much more difficult without information about how the pose evolves over time.

To train and evaluate our action classifier, we collected motion capture data of the subject performing the domain physical actions (Figure 7.12). We acquired 17 minutes of motion capture data stored in 25 AMC files. From the 32,111 total data frames, we divided the first half of the frames in each file to use as the training set and used the second half for testing. Using LIBSVM, we ran cross-validation on the training set to determine the best parameters C and γ for the RBF kernel. The confusion matrix for our action classification on the test data is shown in Table 7.1. The left column shows the label of the correct behavior, and the top row shows the assigned label. The average classification accuracy over the testing dataset was 76.9%. For the locomotion actions (walk, run, etc.) we achieved higher accuracies than on the non-locomotion actions (crouch, rise, probe). To improve performance on these physical actions, we believe that we need to increase our training set size since our initial motion clips of those actions were short compared to the other actions (only about 2000 frames per action class rather than 6000).

7.2.4 Classification

For each canonically-transformed window in our trace, our goal is to select the best behavior model. We perform this classification task by developing a set of hidden Markov models (HMMs), one for each behavior b, and selecting the model with the highest log-likelihood of generating the observed data. Our models ($\{\lambda_b\}$) are parameterized by the

Table 7.1: Confusion matrix for SVM action classification using pairs of concatenated frames with a $1/3$ second (10 frame) separation (40-dimensional vectors). The left column shows the label of the correct behavior, and the top row shows the assigned label. High results down the diagonal indicate good performance. The average classification accuracy over the testing dataset was 76.9%.

	walk	run	sneak	wound	probe	crouch	rise
walk	85.4%	3.9%	7.8%	0.6%	1.8%	0.5%	0.0%
run	12.5%	75.8%	7.9%	0.5%	2.8%	0.5%	0.0%
sneak	6.4%	0.8%	81.9%	0.0%	1.7%	9.2%	0.0%
wound	1.3%	3.4%	0.8%	93.9%	0.4%	0.3%	0.0%
probe	8.8%	3.6%	8.7%	19.3%	56.5%	0.6%	2.5%
crouch	2.9%	18.0%	10.3%	20.0%	11.2%	35.7%	1.9%
rise	12.1%	3.7%	15.1%	32.2%	6.4%	16.3%	14.2%

following:

- N, the number of hidden states for the behavior;
- $\mathbf{A} = \{a_{ij}\}$, the matrix of state transition probabilities, where $a_{ij} = Pr(q_{t+1} = j | q_t = i), \forall i, j$ and q_t denotes the state at frame t;
- $\mathbf{B} = \{b_i(o_t)\}$, where $b_i(o_t) = \mathcal{N}(\mu_i, \Sigma_i)$. The observation space is continuous and approximated by a single multivariate Gaussian distribution with mean, μ_i and a covariance matrix, Σ_i, for each state i.
- $\pi = \{\pi_i\}$, the initial state distribution.

For our problem, given A agents in a team, the observations at time t and window w are the tuple: $o_t = (\mathbf{x}'_{1,w,t}, v_{1,w,t}, \ldots, \mathbf{x}'_{A,w,t}, v_{A,w,t})$. We determine the structure for each behavior HMM based on our domain knowledge. For instance, the stacked behavior can be described using only two states ($N = 2$), whereas we represent the more complicated bounding overwatch behavior using six states connected in a ring. Each hidden state captures an idealized snapshot of the team formation at some point in time, where the observation tuple (in canonical coordinates) is well modeled by a single Gaussian. Rather than initializing the HMMs with random parameters, we use reasonable starting values. These can be polished using expectation-maximization (EM) [28] on labeled training data.

To determine the probability, $Pr(o_{1\ldots T} | \lambda_b)$, of generating the observed data with the model λ_b, we employ the forward algorithm [73] as implemented in the Hidden Markov Model toolbox [67]. We classify each window segment with the label of the model that generated the highest log-likelihood.

7.2.5 Behavior Recognition

During the final behavior recognition phase, we assign behavior labels to observation traces (sequences of state transitions). Observation traces are generated using the simulator as described in Section 7.2.6. These state transitions are of the form *(physical_action1* LOCATION_1 *physical_action2* LOCATION_2*)* where some change has occurred such that *physical_action1* $\neq$ *physical_action2* and LOCATION1 $\neq$ LOCATION2.

First we initialize the behavior recognition with a table that hashes state transitions (about 11,000 entries) to sets of legal behaviors (hypothesis sets). The hash is compiled directly from the behavior specifications to enable efficient searching; behaviors that include states with general features (e.g., outside) are expanded to include legal transitions for more specific cases (e.g., NEAR_BUILDING, NEAR_INTERSECTION).

The domain author designates a cost function to be applied to each potential *behavior transition*. A simple parsimonious cost function is to penalize any behavior transition by a fixed amount; self-transitions (explaining the subsequent behavior with the same label) are not penalized. This type of cost function delays as long as possible before assigning a new behavior label to state transitions.

Using the behavior library and the cost function, we search for the minimum cost explanation for the sequence of state-transitions; this is a shortest path problem which we solve efficiently using dynamic programming.

$$B_{t+1}^{q} = \min_{\forall p}\{B_{t}^{p} + T_{p,q}\}$$

$$T_{p,q} = \begin{cases} 0 & \text{if } p = q \\ 1 & \text{if } p \neq q \end{cases}$$

B_{t+1}^{q} is the cost of explaining a state-transition with behavior label q at time $t+1$; this cost can be calculated by finding the minimum of over all previous behavior labels p of

explaining a set with B_t^p combined with the cost transition function, $T_{p,q}$.

7.2.6 Environmental Simulator

To test new plan libraries and cost functions without the prohibitive expense of acquiring human data in the motion capture lab, we implemented an environmental simulator system capable of generating valid observation traces that correspond to a sequence of known behaviors performed in a specific MOUT environment, composed of typical urban terrain features.

The simulator is equipped with behavior descriptions, either known ones taken from the library used by the behavior recognizer or new ones written to represent cases in which the human deviates from the correct military procedure. The input to the simulator is a list of behaviors from which the simulator stochastically generates one run of valid observation traces that could have occurred if the human performed those behaviors in that MOUT environment layout. Currently our stochastic model is very simple; all possible state transitions exiting a state are equally likely.

Environmental feature descriptors (NEAR_DOORWAY, NEAR_INTERSECTION) are generated by selecting the location nearest to the soldier's (x, y) location; there is a preference order (e.g., NEAR_HAZARD dominates features such as NEAR_INTERSECTION) that dictates which environmental feature is reported if the human is equally close to multiple annotated map regions.

In the simple case, we assume that all state transitions are accurately detected; to model the effects of imperfect state transition detection we use the confusion matrix (Table 7.1) to stochastically model the likely output of our action classifier. For instance, the true state transition might be (*walk* NEAR_DOORWAY *walk* NEAR_DOORWAY); consulting

the top row of the confusion matrix we see that the walk physical action has an 85.4% chance of being detected correctly as walk, 3.9% chance of being classified as a run, 7.8% of being classified as a sneak, and a negligible chance of being detected as anything else. Using the confusion matrix as our simulator noise model we can systematically generate faulty data that realistically models the effects of imperfectly classified the motion capture data. Also, as we improve our classification procedure, we can quickly assess the impact on the behavior recognition.

7.2.7 Results

We examine our cost minimization approach to behavior recognition in the context of a common MOUT scenario to assess the impact of the following factors:

behavior transition cost function: how does changing the transition cost function affect the behavior explanation generated? Is there a good method for using domain knowledge to author a function $T_{p,q}$ that produces good recognition results?

noisy state transitions: how do noisy state transitions affect the behavior explanation output? How can we minimize the effect of incorrectly detected state transitions?

MOUT Scenario: Building Clearing

Building clearing, the process of investigating a building and eliminating hostile occupants and hazards within, is a common goal in MOUT operations. The standard followed military procedure used to clear the building shown in Figure 7.13 would be to use the following sequence of behaviors: **traverse_street**, **scout**, **enter_building**, **enter_room**, **traverse_corridor**, **enter_room**. Since the room marked by the star appears cluttered and

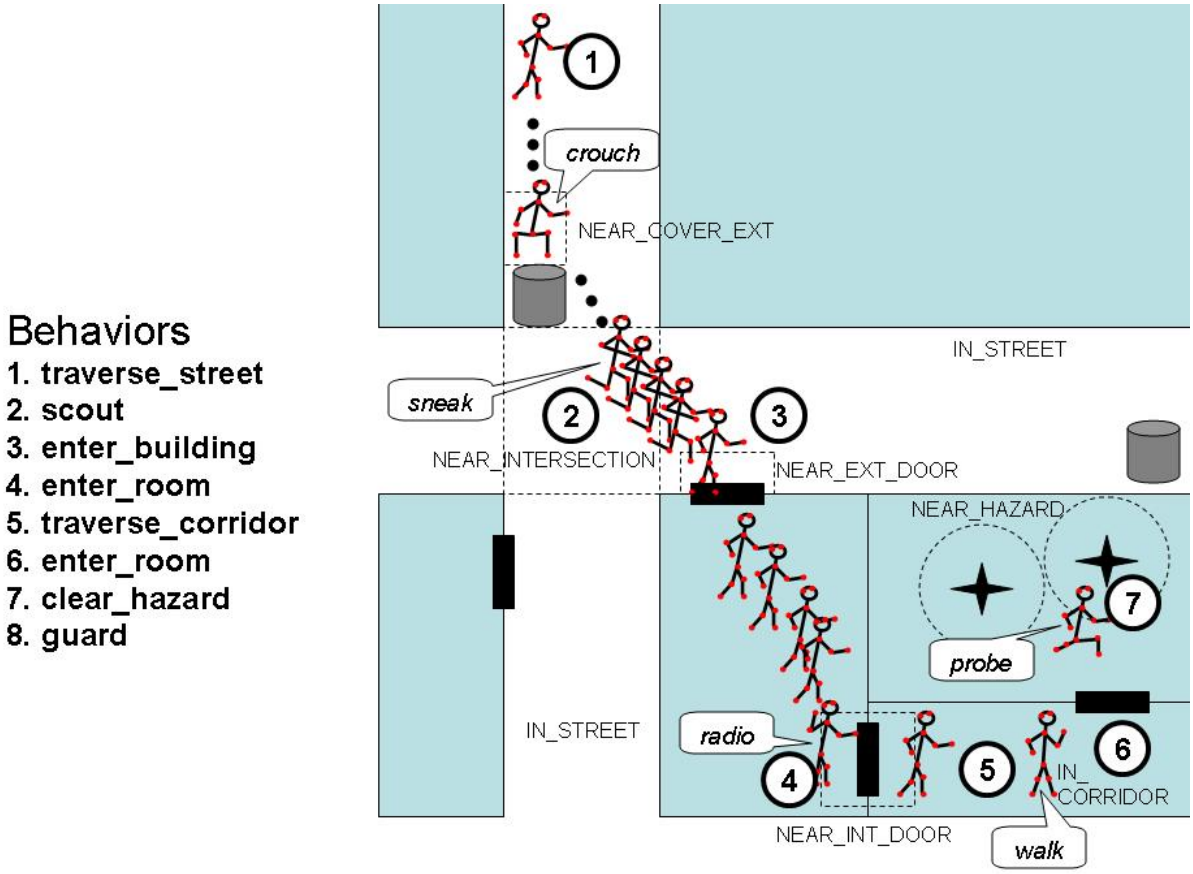


Figure 7.13: MOUT Scenario: Building Clearing. An overhead view of the schematic used by our simulator to generate observation traces for an example building clearing scenario. The standard followed military procedure would be to use the following sequence of behaviors: **traverse_street**, **scout**, **enter_building**, **enter_room**, **traverse_corridor**, **enter_room**. Since the room marked by the star appears cluttered and might potentially contain booby-traps, the soldier should choose to check the area for hazards (**clear_hazard**).

might potentially contain booby-traps, the soldier should choose to check the area for hazards (**clear_hazard**). In the final state, the soldier remains to guard the building to ensure that no one enters the building. If there are enemy forces in the area that fire on the soldier during the **traverse_street** behavior, the soldier should make a strategic withdrawal and counterattack; this can be accomplished by executing the behavior sequence, **retaliate retreat fast_bound** and **ambush**, before returning to the original building clearing operation. In the following subsections, we use the behavior sequences described above to illustrate the effects of behavior transition cost functions, noisy state transitions, and human behavior deviations on the behavior recognition process.

Impact of Cost on Behavior Recognition

The behavior transition cost function, $T_{p,q}$, directly affects the explanation generated by the dynamic programming search process. Using the parsimonious cost function described in Section 7.2.5, we analyze stochastically-generated state transition sequences for the building clearing operation (**traverse_street**, **scout**, **enter_building**, **enter_room**, **traverse_corridor**, **enter_room**, **clear_hazard**, **guard**). Typically, this behavior sequence generates about 50 state transitions; in the absence of classification noise, the recognizer with parsimonious cost function correctly labels between 90–100% of the state transitions. Often it mislabels the final guard behavior as being part of a **enter_room** behavior; because many of the same state transitions appear in both. However using our domain knowledge we know that the soldier should guard an area after clearing it; to represent that domain knowledge we decrease the cost of **clear_hazard** followed by **guard**.

We believe that the cost function is closely related to the behavior application as well as the domain. For instance if the agent is attempting to model an opponent, using a paranoid cost function that is sensitive to the ambush behavior (allowing cheaper transitions

from common behaviors to ambush) might be more useful, even at the expense of a slight decrease in overall behavior recognition accuracy.

Handling Noisy State Transitions

Another question is whether this approach is robust in the face of noisy state transitions. Based on frame-pair-level SVM classification results, the naive approach of proposing action transitions whenever the SVM label changed would not be sufficiently accurate (if each frame-pair were only classified with 76% accuracy, such a method would hallucinate false action transitions at unacceptable rates). Fortunately, our motion capture stream produces data at a rate (30 frames per second) that is much faster than the typical human action transition frequency. This enables us to apply a median-based outlier rejection scheme to the SVM output data that eliminates the majority of incorrect labels resulting in far fewer incorrect action transitions.

7.3 Discussion

This chapter examined the use of an alternate source of data, human motion capture, for activity recognition. We demonstrate a general framework for building a non-parametric model of an agent's physical capabilities using human motion capture data. By precomputing cost maps for actions and areas of interest, the algorithm provides the high-level decision-making algorithm with a sound basis for selecting one behavior from a set of equivalently goal-achieving actions. We also introduce an offline approach to the problem of activity recognition and demonstrates how it can be used to recognize physical behaviors even with imperfect action classification. Human motion capture is very promising as a data source for acquiring traces of single agent behaviors but is less useful as a means of

capturing traces of team activities due to the increased likelihood of occlusion and marker disambiguation problems.

Chapter 8

Conclusion and Future Directions

If I could solve all the problems myself, I would.

-Thomas Edison, when asked why he had a team of twenty-one assistants.

Developing software agents for virtual environments is an important research direction for the multi-agent systems community. Over the past ten years, the emergence of the WWW created an enormous repository of text information in easily accessible electronic form and became an important global testbed for information agents. In the same way, massively multi-user virtual environments will become a testbed for global communities of embodied agents that interact with humans for entertainment, education, and telepresence applications. Currently, worldwide revenues for massively multi-player online role-playing games (MMORPGs) exceed 1 billion dollars [69]. Second Life, a popular Internet-based virtual world, is inhabited by over 7.8 million people, performing $600,000 of business transactions per day [55].

Education is an growing application area for virtual environments; Second Life hosts virtual distance-learning classrooms for over 70 colleges and universities [85] and allows users to access library materials at virtual reference desks. By merging cognitive tutor-

ing programs and virtual environments, researchers have created innovative and fun programs for individual training in language [98] and decision-making [96] skills; additionally there has been work on designing pedagogical agents for team tasks, such as naval casualty response, in virtual environments [77]. Tasks requiring multiple participants should particularly benefit from the increased practice opportunities created by the easy availability of simulation training. For instance, training one football quarterback requires an entire team of supporting players and opponents; typically second and third string quarterbacks only receive limited practice time because of the training expense. Having software agent teammates ("virtual humans") available to practice at the trainee's convenience could greatly reduce the cost of team training.

Research advances in displays, haptics, and user interfaces will make future virtual environments more immersive than their current counterparts and enable the training of real physical skills in virtual environments. Some current games, such as Dance, Dance, Revolution [26], score the players on the speed and accuracy of gross physical motions like foot tapping and hopping while they dance to music on a pressure-sensing pad. The Wii remote for the Nintendo console uses accelerometers and infrared detectors to track the user's hand motions and allow the miming of game actions, such as swinging a sword [107]. Recent advances in marker-less motion capture technology could potentially be incorporated in games to provide full body sensing without a physical controller [68].

8.1 Contributions

Developing agents that intelligently cohabit this rich virtual world with humans will require effective activity/plan recognition. Creating shared understanding between human and agent teammates is the biggest challenge facing developers of mixed-initiative hu-

man/agent organizations. The limiting factor in most human-agent interactions is the users ability and willingness to spend time communicating with the agent in a manner that both humans and agents understand, rather than the agents computational power and bandwidth [97]. Embedding activity recognition algorithms into multi-player virtual environments will enable:

- the development of smarter opponents that can recognize and respond to the tactics of human teams;
- the creation of better synthetic training partners that can predict their teammates' intentions and coordinate without unnecessary communication;
- automated annotation of multi-player game logs;
- intelligent user correction in team training environments.

This thesis makes the following contributions towards the problem of activity recognition for agent teams:

Formation identification: Many team behaviors in physical domains exhibit distinctive spatial configurations among agents, and between agents and static objects. We present an efficient method for recognizing such team formations. Our method is robust to noise in agent position and high degree of clutter (other agents). Unlike template matching methods that must search exhaustively over a discretized parameter space, our approach only considers those hypotheses that are consistent with a minimal subset of agents, enabling it to scale easily to large agent teams in arbitrary spatial layouts.

Behavior recognition for dynamic agent teams: In complex scenarios, the membership of agent teams changes over time: teams assemble to accomplish specific tasks, cre-

ate subteams as needed, and disband into individuals. We present an approach for simultaneously recovering agent-to-team assignment and team behavior for such tasks. We efficiently hypothesize team assignments where the spatio-temporal traces for those agents (over a limited time window) match models of known team behavior.

Efficient plan recognition for dynamic agent teams: When team behaviors are generated by higher-level plans, our goal is to match observed agent activity to specific trees in a given plan library. Naive approaches to this problem can be extremely expensive since the number of hypotheses grows combinatorially. A particular challenge is that the actions of splitting and merging in dynamic teams are unobservable and can only be inferred through indirect means. We present efficient pruning techniques based on across-plan and within-plan action constraints generated by static analysis of plan libraries. This enables us to perform plan recognition for large dynamic teams.

We examine the following specific applications:

Spatio-temporal analysis of team formations in MOUT: We apply formation recognition in conjunction with HMM-based activity recognition to recognize behavior of small-team maneuvers in the Military Operations in Urban Terrain (MOUT) domain. We demonstrate our work on human subject traces collected using a modified version of the Unreal Tournament game engine.

Policy recognition in domains with weak spatio-temporal cues: We explore both model-based (Dempster-Shafer) and data-driven (SVM) approaches for recognizing player policies in Dungeons & Dragons skirmish scenarios, a domain with weak spatio-temporal cues.

Recognition of human activity from motion capture traces: We present an approach for constructing physical capability models of human activity from motion capture data. At the lowest level, these models enable us to recognize physical actions such as running and sneaking (in a MOUT context). At a higher level, we can employ physical capability models to problems of opponent modeling in sports.

8.2 Future Work

The area of activity recognition for agent teams is relatively young and remains a fertile research area. There are four areas that could clearly benefit from future work.

methods for acquiring plan/template libraries: One important question left unresolved by this thesis is the origin of template and plan libraries. In this thesis we demonstrate that it is possible to do some analysis without libraries (Chapter 6), but much of the work in this thesis depends on access to plan and template libraries. An important area of future research is the creation of these libraries, either through eliciting the information from subject matter experts [70] or through learning templates and plan libraries from data.

enhancing teamwork with online behavior recognition Human-agent teams have been demonstrated to be useful for diverse applications such as personal assistant agents [18], military joint mission planning [31], and virtual training environments [77]. Moreover, human-robot teams are becoming increasingly important as robots become more capable of functioning as equal partners. Mutual predictability between team members has been shown to be an important predictor of human team performance [97]. We believe that behavior recognition is one possible solution towards the problem of creating mutual predictability in human-agent (or human-robot)

teams. For human-agent teamwork applications, the two most important research challenges are (1) making online predictions in a timely fashion, and (2) incorporating predictions into the agent's decision-making process.

migrating from the virtual world to the real world The proliferation of wireless sensors, cameras, and handheld devices in home and office environments will make the same types of localization and tracking currently available for players within gaming environments feasible in real-world environments. However, the problem of combining this tracking information with high-level information about people's goals, schedules, and preferences remains; moreover the problem of human behavior recognition for large groups of humans remains largely unexplored. The same algorithms for multi-agent behavior recognition algorithms presented in this thesis for simulation and gaming environments could also be applied to analyzing multi-person activity in the real world.

combining communication and physical movement For some tasks, communication between team members is an important predictive cue of future action. By combining information about team members' communication patterns with physical movement, it is quite likely that we could improve recognition performance. Appendix A describes some preliminary results at analyzing the communication patterns of human teams performing collaborative search tasks in a virtual environment.

Appendix A

An Analysis of Communication and Coverage Patterns in Human Teams Performing Collaborative Search Tasks

Team decision making is a bundle of interdependent activities that involve gathering, interpreting and exchanging information; creating and identifying alternative courses of action; choosing among alternatives by integrating the often different perspectives of team members; implementing a choice and monitoring its consequences. Software agents can fill a critical need for (1) supporting human team members in accessing, filtering, and synthesizing information from disparate sources; (2) increasing team situation awareness; (3) aiding the formation of shared mental models; (4) supporting team coordination in making decisions related to resources, tactics, and goals to meet the overall planning objectives. Building effective human-agent teams requires overcoming several important scientific challenges that to date have not been addressed: (1) the creation of mutual understandability between humans and agents; (2) the development of coherent team

interactions; (3) establishing human trust in agent judgments.

It is well-recognized that proficient teams achieve goals and accomplish tasks that otherwise would not be achievable by groups of uncoordinated individuals. While previous work in teamwork theory [82] has focused on describing ways in which humans coordinate their activities, there has been little focus on which of those specific activities and information flows can be enhanced by being performed by software agents. The focus of our initial human team experimentation is to (a) establish a baseline of human-only teamwork for a given task domain and (b) ascertain the relative importance of different information flows for the team task in order to derive "insertion points" for agent assistance of human teams. These insertion points are not merely limited to coordination and information flows, but potentially include teamwork maintenance and task completion. In this chapter, we describe our analysis of a collaborative search task, a team scavenger hunt, performed by human subjects in a multi-player gaming environment. The results of this analysis will inform the future development of software agents to assist human teams performing search tasks.

A.1 Supporting Human Teamwork

Research in human team performance suggests that experienced teams develop a shared understanding or *shared mental model* to coordinate behaviors by anticipating each other's needs and adapting to task demands [34]. Furthermore, for such teams, both tacit and explicit coordination strategies are important in facilitating teamwork processes. Explicit coordination occurs through external verbal and non-verbal communications, whereas tacit coordination is thought to occur through the meta-cognitive activities of team members who have shared mental models of what should be done, when, and by whom [29,

33, 43]. A team's shared mental model thus allows the team members to coordinate their behavior and better communicate depending on situational demands. Initial theorizing on training shared mental models suggests that for teams to successfully coordinate their actions, they must possess commonly held knowledge structures, such as knowledge of teammates' roles and responsibilities along with team tasks and procedures.

Creating this shared cognition between human and agent teammates is the biggest challenge facing developers of mixed-initiative human/agent organizations. The limiting factor in most human-agent interactions is the human's ability and willingness to spend time communicating with agents in a manner that both humans and agents understand [97]. Horvitz [45] formulates this problem of mixed-initiative interaction as a process of managing uncertainties: (1) managing uncertainties that agents may have about the human's goals and focus of attention, and (2) uncertainty that humans have about agent plans and status. Creating agent understanding of human intent and making agents' results intelligible to a human are problems that must be addressed by any mixed-initiative system, whether the agents reduce uncertainty through communication, inference, or a mixture of the two.

A.1.1 Agent Roles in Human Teams

Sycara and Lewis [97] identify three primary roles played by agents interacting with human teams.

- **Agents support individual team members in completion of their own tasks.** These agents often function as personal assistant agents and are assigned to specific team members [18]. Task-specific agents utilized by multiple team members (e.g., [22]) also belong in this category.

- **Agents support the team as a whole.** Rather than focusing on task-completion activities, these agents directly facilitate teamwork by aiding communication and coordination among humans and agents, as well as focus of attention. The experimental results summarized in [97] indicate that this can be a very effective aiding strategy for agents in hybrid teams.

- **Agents assume the role of an equal team member.** These agents are expected to function as "virtual humans" within the organization, capable of the same reasoning and tasks as their human teammates [104]. This is the hardest role for a software agent to assume, since it is difficult to create a software agent that is as effective as a human at both task performance and teamwork skills.

There are additional research challenges, specific to the team role assumed by the agent. Agents that support individual human team members face the following challenges: (1) modeling user preferences; (2) determining optimal transfer-of-control policies [84]; (3) considering the status of user's attention in timing services [45]. Agents aiding teams [60–63], face a different set of problems: (1) identifying information that needs to be passed to other team members before being asked; (2) automatically prioritizing tasks for the human team members; (3) maintaining shared task information in a way that is useful for the human users. Agents assuming the role of equal team members [31, 32, 104] must additionally be able to: (1) competently execute their role in the team; (2) critique team errors; (3) independently suggest alternate courses of action. Perhaps because of these challenges, there are very few prior results on human-agent team aiding and teamwork. Examples of tasks that were investigated include target identification [62, 63], achievement of a military rendezvous plan [60, 61] and delivery of supplies to troops [31, 32]. All of this prior work has uniformly found that human-agent teams exhibited superior performance over human-only teams not only in achievement of task

objectives but also in performance stability.

A.1.2 Improving the Performance of Human Teams

We hypothesize that to improve the performance of human teams, agents must do some combination of the following:

- reduce information processing costs;
- decrease uncertainty in the task;
- improve coordination between team members;
- directly accomplish part of the team task.

Galbraith observed that "the more uncertainty in a task, the more information processing necessary to achieve a given level of performance" [38]. Hence, having the agents assist either in information processing or decreasing uncertainty should improve the team's performance. Moreover, in cases where the task is time-stressed, having the agents simply perform part of the task for the humans has the potential to improve team performance as well, particularly in cases where the task reward is an increasing function rather than a thresholded one. Based on experiments of student project teams, Kraut suggests that a human team's resultant state of coordination, defined as the degree to which interdependencies are managed well, is an important predictor of team performance [57]. This state of coordination can be created by mechanisms such as communication, shared cognition, and team history. If agents can improve the state of coordination between team members or reduce the cost of achieving a good state of coordination, the team performance should improve.

A.2 Collaborative Search

For our initial set of experiments, we monitored teams of human subjects performing a collaborative search task in simulation. Search and rescue is a challenging, time-stressed team task with a potentially high payoff since inadequate team performance can result in fatalities. By developing software agents capable of improving human team performance on collaborative search tasks, we can positively impact coalition search and rescue operations.

A.2.1 Wilderness Search and Rescue Operations

In this section we provide a task analysis of how civilian human teams perform wilderness search and rescue operations summarized from [41, 87]. We assume that many aspects of the task analysis are also applicable to military search and rescue teams, although military teams have access to different equipment and also often face the additional problem of rescuing victims from enemy territory. A goal-directed task analysis of wilderness search and rescue operations identified the following list of operational goals and subgoals [41]. The italicized task elements are also applicable to our simulated collaborative search task.

1. Stage preparation
    (a) Reporting party call
    (b) Activation call
    (c) Assemble (prepare for search)
2. Acquire missing person description
    (a) Gather missing person information

   (b) Determine missing person's intent

3. *Develop search plan*

   (a) Create a perimeter

   (b) *Assign priority to clues*

   (c) *Update map information*

   (d) *Create a priority pattern*

   (e) *Organize resources for search execution*

   (f) *Communicate search plan*

4. *Execute search plan*

   (a) *Follow plan*

   (b) *Find signs (or absence of)*

   (c) Keep searchers safe

   (d) *Communicate acquired information*

5. Recover victims

   (a) First aid for victims

   (b) Rescue, extract, or recover the missing person

6. *Debrief search team*

   (a) *Determine what happened*

   (b) *Evaluate how the team can improve*

When executing a wilderness search plan, the teams employ four distinct types of search: hasty, constraining, high probability region, and exhaustive. During hasty search, the searchers rapidly check high probability areas to determine the missing person's location or direction of travel. This type of search is often used in the initial part of the search plan. During constraining search, the searchers attempt to build a perimeter bounding the victim's location; an example of constraining search would be having searchers check a large snowy field for tracks to localize the victim to one side of the field. Hasty search and constraining search are used by the incident commander to find clues and establish search priorities. After search priorities have been established, the incident commander divides the search area into regions and deploys search teams to search high probability regions. Exhaustive search is done by having the searchers form a line and walk abreast through an area; this type of search is used to find clues such as clothing or wrappers after other forms of search have failed.

Wilderness search and rescue operations pose the following challenges to expert human teams: (1) information overload of the incident commander while assimilating information collected by the field teams; (2) the creation of accidental holes in the search pattern due to poor execution of the search plan by the field teams; (3) poor priority assignments in the search plan due to false clues and hunches. We believe that software agent assistance can potentially reduce the information overload of the incident commander and minimize errors during the execution of the search plan. In the next section, we describe our experimental version of the collaborative search task, the team scavenger hunt, which tests the ability of human subjects to collaborate to develop and execute a team search plan in a simulated environment.

A.2.2 Experimental Task: Team Scavenger Hunt

The collaborative search task that we designed for our experiments, the team scavenger hunt, recreates some of the challenges faced by expert human teams during search and rescue operations. To implement the task, we reconfigured a scenario in the multi-player game and battlefield simulator, Operation Flashpoint (OFP version 1.96) [36], by customizing the pre-game briefing, map, object triggers, and scoring mechanism.

In the team scavenger hunt, human subjects have to read a map, navigate a 3D simulated environment and recover a collection of objects (bottles) within a bounded amount of time (Figure A.1). The task is designed to evaluate the team's ability to develop and execute a search plan under time-stress. As an experimental task, the team scavenger hunt offers several advantages: (1) it can be learned and executed within a short period of time by novice subjects; (2) it can be simulated within a variety of testbeds; (3) it offers a simple team performance metric: number of objects collected.

The team scavenger hunt task can be made arbitrarily complicated by adjusting the following parameters: (1) task uncertainty, (2) reward function, (3) adversaries. Task uncertainty is increased if subjects are not provided with maps and have to simultaneously explore the area while searching for objects. Another way to increase task uncertainty is to have subjects locate objects based on clues or probability distributions, rather than precise locations. Varying reward functions can be used to elicit different types of team behavior. Individual players can be awarded incentives for high performance vs. having the rewards split equally among team members. A simple reward function is to have the reward be a linear function of items acquired across all team members; another option is to award points for portfolios of objects. A portfolio of objects is a collection that contains a specified number of unique objects with desired characteristics, e.g., a portfolio consisting of a table, a chair and a telephone, all of the same color. Having a portfolio

based reward system makes a subject's optimization problem harder because it penalizes locally greedy acquisition strategies. Adding adversaries to the task forces the players to replan to overcome unexpected obstacles. The game can be made more dynamic by adding mobile objects, automated adversaries to hinder the searchers, or having teams compete against each other.

In our initial version of the experimental task, the subjects have some uncertainty—they are provided with a terrain map, but only objects within a certain visibility range are revealed on the map. The current version of the game requires having the searchers collect static objects; subjects are rewarded based on their total team score, rather than their individual score, at collecting objects within an adversary-free environment.

A.2.3 Testbed

The experiments focused on the activity of three human players acting through virtual characters in the Operation Flashpoint (OFP version 1.96) simulated physical environment to find and crush liquor bottles in a twenty minute period. OFP is distributed with a simple but versatile scenario editor that greatly facilitates the creation of multi-player military and civilian scenarios and missions.

Terrain around and including the village of *Flers* on the island of *Normandie* was chosen as the focal point for the one practice and two experimental scenarios (Figure A.2). The area is a tract of land that is 512 meters long in a north–south direction (N/S), and 768 meters long in an east–west direction (E/W); in all, 393,216 square meters. On the 2-dimensional (2D) Operation Flashpoint map, this area corresponds to 4 map squares N/S, 6 map squares E/W, where each map square corresponds to 128 meters by 128 meters. Exploratory benchmarks determined that, depending on search technique and ability, it could take a single OFP civilian virtual character from sixty to ninety minutes

Figure A.1: Subject world view during bottle collection. This is a zoomed-in view that has an increased density of bottles for illustrative purposes; the actual 3D environment is much larger and contains a much lower bottle density.

Figure A.2: The 2D terrain map available to players in the Operation Flashpoint simulation environment. In addition to the terrain map, the subjects are provided with simulated versions of binoculars, compass, and watch.

to explore all 24 map squares of this scenario. In twenty minutes, a civilian character can thoroughly explore roughly ten map squares of the surrounding countryside. The village of Flers occupies four map squares; part of the village is organized in a radial street plan and another part has a N/S, E/W grid of streets and buildings. Given the area and layout, we have observed that it requires from ten to twenty minutes for the virtual civilian character to search the area.

A.3 Pilot Experiments

A.3.1 Procedure

Eight teams of three persons, each, were recruited to participate in the pilot study. Human subjects self-assessed and reported their abilities to play first person video games in terms of the following classification: novice, medium expertise, or expert. Combined expertise of the teams varied from "two novices and a medium expert" to a team of "three experts" (see Table A.2).

Each team member played the game through an assigned and dedicated laptop. All three members of a team sat at the same large table arranged in such a way that they could not look at each other's screen. The human subjects were forbidden to share computer screens, note sheets or other such aids — they could only describe their locations, intentions and actions in the game by using verbal communications and the 2D OFP map of Flers. All verbal communications, though face-to-face, were logged using TeamSpeak [102].

Time was taken during a practice session to instruct the players on the key and mouse commands for the game. Players were instructed on how to move their characters, find

and crush bottles, query bottle counts, and how to use additional aids that are available to their avatars. After sighting a bottle, a player must move their avatar to within a couple of meters of it in order to crush it and get credit for the crush. When they are close enough to crush the bottle, the command to crush that type of bottle, e.g. **Crush Martini Bottle**, will appear in the player's command menu at the bottom right corner of their screen. Feedback to the player is given in multiple ways: (1) the sound of a vehicle crashing into a wall, (2) puffs of oily black smoke emanating from the morphing bottle, (3) the morphing of the bottle into a crumpled form. If the player queries their bottle count, they will see that it has increased by one.

Once a player has crushed a bottle, the command to crush it is removed from their menu, never to appear again for that bottle, even if they happen upon its crushed remains at a later time. If a player encounters the remains of a bottle that was crushed by a teammate, they can choose to invoke the command to crush it in order to avoid false detection of that crushed bottle at a later time. No penalty was assessed for attempting to crush an already crushed bottle.

The five ways of detecting a bottle are:

1. *visual detection*, in which the human player "sees" a bottle via the unmagnified vision of their avatar,

2. *magnified visual detection*, in which the human player slightly magnifies (roughly, 3X) their avatar's field of vision,

3. *visual detection via binoculars*, in which the avatar uses binoculars for a narrower but more distant field of view,

4. *non-visual proximity sensing*, in which the player is notified of a bottle's presence whenever their avatar comes within "sensing range" of the bottle. A bottle is sensed

based on the expertise of the OFP avatar and if the player is proximate to it. This game effect is useful if the bottle is on the other side of a hedge or if the player accidentally passes the bottle. It does not work if the bottle is in a terrain depression, or more than a few meters away from the player. The notification consists of the player's command menu appearing in the bottom right corner of their screen, with the added command, "Crush X Bottle", where X indicates the type of bottle.

5. *tool tip sensing* of the bottles from the 2D map view of the world. OFP avatars can navigate the environment in a 2-dimensional map view. When in 2D map view, the player's avatar is represented as two concentric red circles with a radial line indicating the avatar's bearing. If the human user moves the mouse cursor over the area of the map in the vicinity of the avatar, they can detect any objects that they could normally see in the visual detect mode. When an object is detected, a "tool tip" label appears next to it, indicating the object's type.

Each experimental session was composed of a twenty minute practice period and two twenty minute search tasks. We evaluated the two experimental conditions: (1) **# Bottles Known** in which the subjects knew how many total bottles they were trying to recover; and (2) **# Bottles Unknown** in which they did not know how many bottles were hidden in the search area. By knowing the total bottle count, we hypothesize that subjects have a better sense of task progress and can assess their individual search performance. Comparing the team performance of subjects with the bottle count information vs. no bottle count information might predict the benefits of introducing agents to teams for search tasks.

A.3.2 Analysis of Team Communication

To analyze the coordination demands of the collaborative search task, we logged all communication between team members. We looked at the following categories of communication:

- **increasing situation awareness (SA)** This category includes all communications that increase the team members' situation awareness. Examples include communicating one's location, querying teammates for their positions, and discussions about terrain features or object locations.
- **sharing hints (Hints)** Occasionally subjects shared personal search techniques with their teammates, such as scanning large regions in a 2D map view or using binoculars while standing on high terrain features.
- **team planning** This category includes any discussion proposing, accepting or declining team search strategies; for example, team members often took responsibility for covering a certain region or suggested that other team members should redirect their search to a different area. We separated team planning into two categories: (1) planning before execution (**Pre Plan**) and (2) planning during execution (**In Plan**). Within these categories we examined two types of communications: (1) role allocation and (2) division of execution space.
- **monitoring task progress (Monitor)** Often subjects exchanged information about object counts, coverage progress, or time left remaining in the session.
- **sharing world beliefs (Beliefs)** Sometimes the subjects discussed their hypotheses about the relative frequency distributions of the bottles in different regions and speculated about the existence of bottle caches.

A: okay do we want someone to stay in the courtyard and do those bottles?

B: I'll do that and then head east.

C: I'll do the same area that I did before.

B: I'll clear the courtyard and then clear the road to the south.

A: I'll work on the northern part and 64.

C: I think I'm going to stay closer to the town and circle around.

Figure A.3: Transcript of communication between subjects at the beginning of the search. This group of utterances was categorized as an example of team planning before execution. The "64" refers to a row on the map. There are significant pauses between the utterances; during one such pause, one of the subjects changes their mind and decides to cover a different area.

- **miscellaneous** Some of the communication between team members was not directly related to the experiment, such as social interaction or complaints over system issues (e.g., unexpected key lockups or display slowdowns).

Most of the team planning discussions were related to the division of the execution space: how to allocate the efforts of the team members to cover the entire map within the 20 minute time period. Although some teams agreed on a division of labor at the beginning of the task period, many teams modified their search strategies during execution based on their perceived task progress or their assessment of which areas contained a higher bottle density. The transcript shown in Figure A.3 is a typical example of team planning communication at the beginning of a search session. The three subjects quickly develop a search strategy in which each subject assumes responsibility for covering a certain region.

To assess the communication demands of the collaborative search task, we compiled frequency counts of the different types of team communication (Figure A.4). We believe

that the categories of increasing team situation awareness and monitoring task progress are amenable to agent assistance. Our model of human-agent teamwork predicts improved team performance if we can reduce the cost of information processing for the team.

A.3.3 Team Search Patterns

During the experiments, we had the subjects report their search patterns and self-assess their coverage of the area by annotating a hardcopy printout of the map; many subjects used these notes to track their coverage progress. We observed a variety of search strategies among the subjects: (1) scanning the map by quadrants; (2) following terrain features such as roads or hedges; (3) focusing effort in regions with higher bottle counts. Figure A.5 shows an example of one subject's search strategy. For each group of subjects, we measured (1) number of quadrants covered per subject; (2) number of quadrants covered per team (the union of each subject's coverage areas). The number of quadrants covered per team is a good measure of team coordination. On average each subject was able to cover 41.3% (mean of both conditions in Table A.1) of the region within 20 minutes; therefore, all three team members were required to perfectly cover the region. Figure A.5 shows an example search pattern that was reported by a subject in Team 4. This annotation was used to estimate the number of quadrants that the subject was able to cover; team coverage was determined by examining the union of all team members' individual coverages at the quadrant level. Table A.1 contains the individual (Columns A, B, and C) and team terrain coverage performance for all eight teams. Team scores are lower than the sum of individual scores due to coverage area overlap. On average, team coverage was 88% of the map for both the **# Bottles Known** and **# Bottles Unknown** conditions.

To be successful at covering the entire region, teams had to effectively divide the ex-

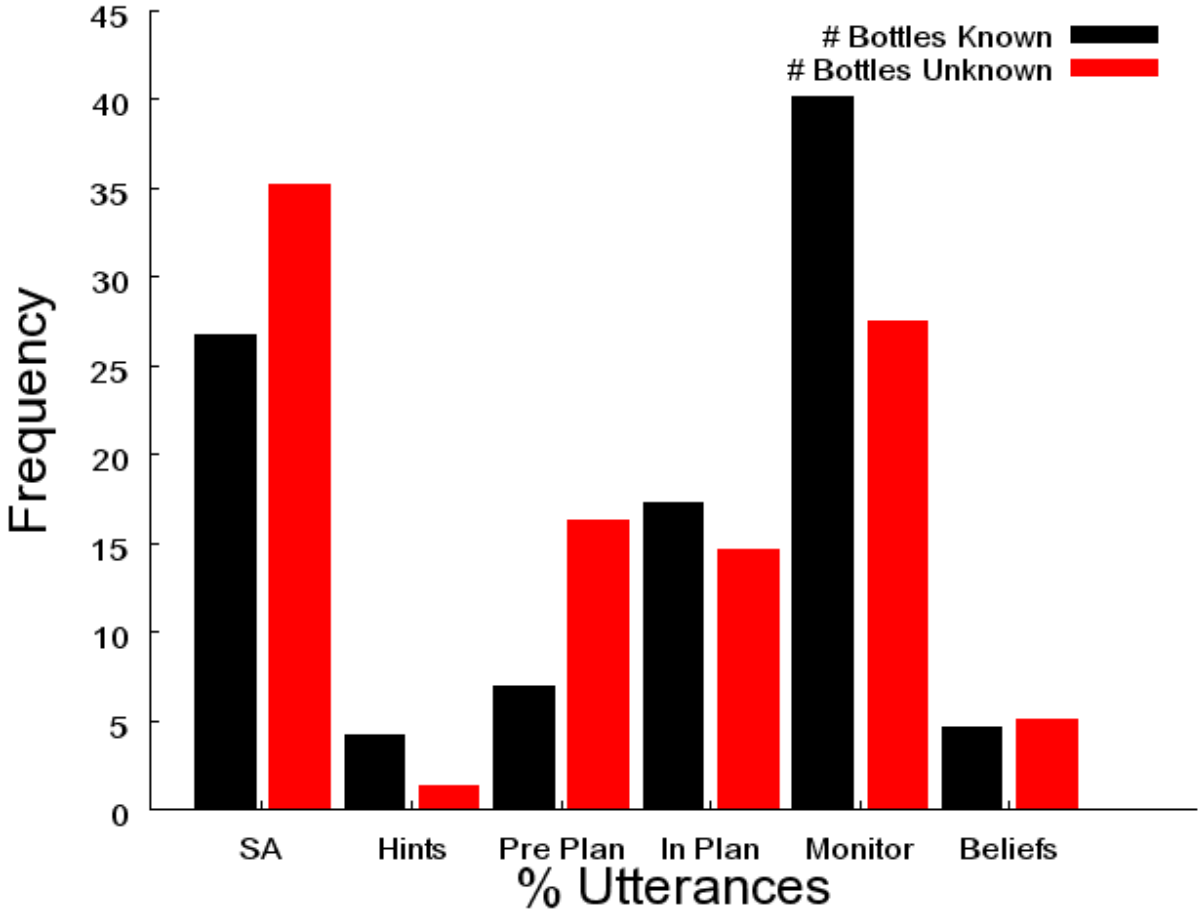


Figure A.4: Communication frequency averaged across four teams of subjects. Subjects appear to spend more time monitoring their task progress in **# Bottles Known** condition, whereas in the **# Bottles Unknown** condition more communications are devoted to increasing the team's situation awareness. Also, the subjects appear to be spending more time planning prior to execution in the **# Bottles Unknown** condition.

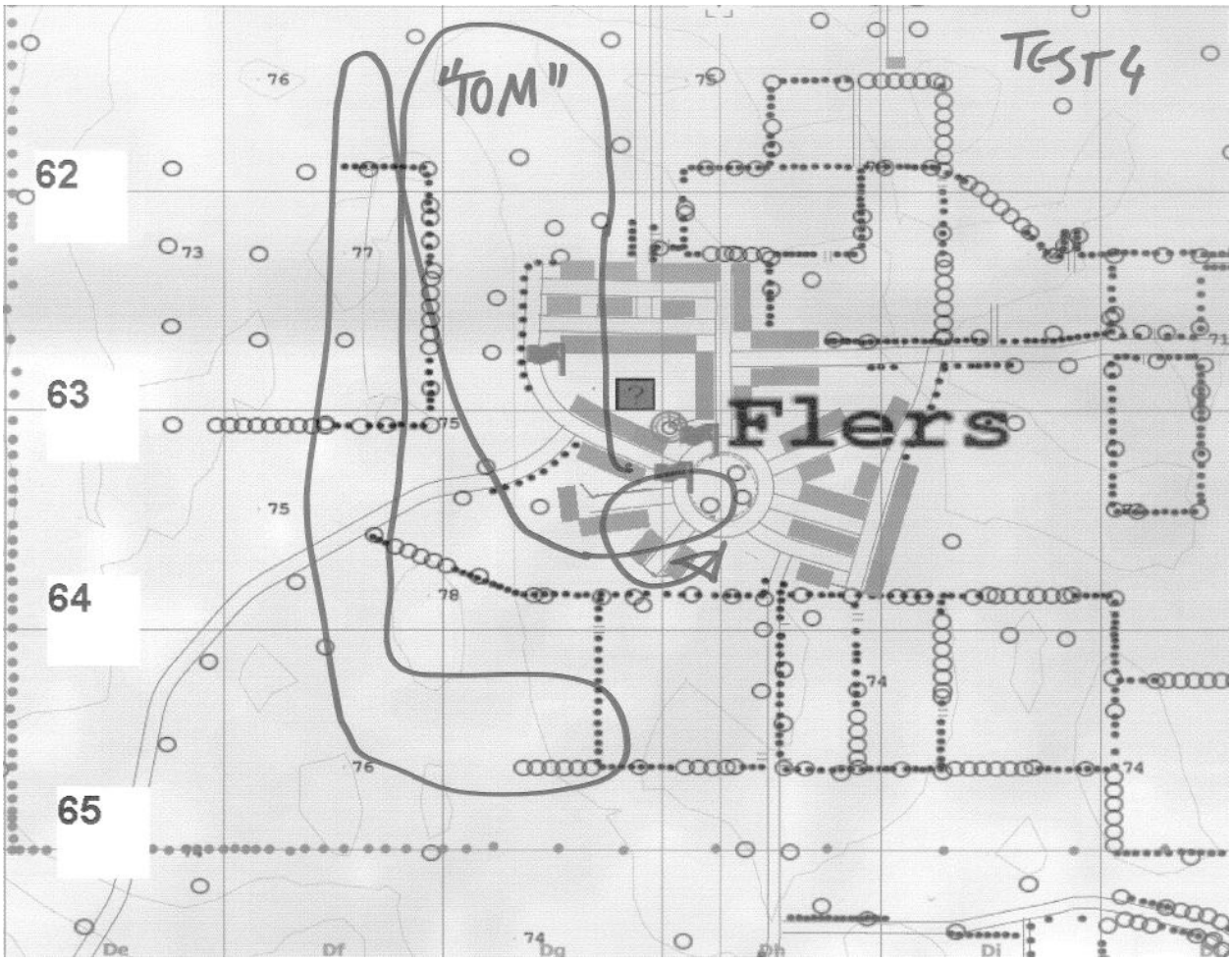


Figure A.5: Map of search area annotated by one of the subjects in Team 4. The numbers and letters at the edge of the map are the coordinates for quadrants. After each search, the subjects were asked to report their search pattern by drawing on a printout. These annotations were used to calculate the individual and team coverages (see Table A.1).

Table A.1: Terrain Coverage

	# Bottles Known				# Bottles Unknown			
Team #	A	B	C	Team	A	B	C	Team
1	25.0%	21.1%	50.0%	70.83%	41.6%	33.3%	41.6%	79.16%
2	25.0%	45.8%	20.8%	95.83%	58.3%	45.8%	20.8%	100.00%
3	20.8%	29.1%	50.0%	66.67%	33.3%	37.5%	37.5%	79.16%
4	33.3%	37.5%	37.5%	100.00%	33.3%	45.8%	41.6%	100.00%
5	37.5%	41.6%	54.1%	87.50%	45.8%	62.5%	37.5%	95.83%
6	45.8%	66.7%	37.5%	95.83%	66.7%	58.3%	37.5%	83.33%
7	75.0%	37.5%	25.0%	87.50%	58.3%	29.1%	33.3%	75.00%
8	75.0%	33.3%	45.8%	100.00%	25.0%	20.8%	66.7%	95.83%
mean		40.44%		88.01%		42.16%		88.54%

ecution space and be proactive at diagnosing and repairing accidental gaps in the search pattern. Instead of attempting to cover the entire region, some teams hypothesized that certain quadrants had a high bottle density and focused on thoroughly searching those quadrants at the expense of less promising areas. The search patterns demonstrated by the experimental subjects exhibited similar problems to the behavior of actual search and rescue teams: (1) the creation of accidental holes in the search pattern due to poor execution of the search plan, and (2) poor priority assignments in the search plan due to false clues and hunches. This is a promising area for agent assistance; by having agents track individual team members' coverage, gaps in the team coverage are exposed earlier in the search process allowing repairs to be made in a more timely fashion. Our model predicts that aiding the state of coordination between team members will result in task performance improvement. Another potential assistance strategy would be to have agents help the subjects form better priority assignments by noting the number of bottles found in each quadrant and informing team members about quadrants with higher bottle densities.

Table A.2: Bottles Retrieved

Team #	Subject Expertise	# Bottles Known	# Bottles Unknown
1	novice, medium, expert	84.61%	78.57%
2	novice, expert, expert	73.33%	54.76%
3	novice, novice, expert	60.52%	66.66%
4	novice, novice, medium	77.27%	35.71%
5	medium, expert, expert	86.36%	73.80%
6	novice, medium, medium	59.52%	52.38%
7	expert, expert, expert	81.81%	76.19%
8	medium, expert, expert	94.00%	80.95%
mean	—	77.17±12.24%	64.87 ± 15.88%

A.3.4 Team Performance

Table A.2 reports the performance of all the teams in our initial set of experiments, measured by percentage of bottles crushed by each team. We had each subject self-assess their expertise at computer games; this information is reported in the second column. During each session, we evaluated the performance of the teams on three search tasks: (1) an initial practice session during which the subjects were learning the user interface (results not shown), (2) a session in which the subjects knew the total number of bottles hidden on the map (labeled in the table as **# Bottles Known**), (3) a session in which the subjects did not know how many bottles they were trying to recover (**# Bottles Unknown**). Based on these preliminary results, it appears that knowing the total bottle count improved the performance, which indicates that this might be a promising area for agent assistance.

A.4 Discussion

This initial phase of experiments was designed to (1) create a baseline of expected team performance and (2) determine where agent aiding is likely to have the greatest impact.

From our preliminary results we noted a few trends:

- The categories with the highest communication traffic were situational awareness (e.g., communicating one's location to the team) and task monitoring (communicating bottle counts, time, and coverage). In the **# Bottles Known** condition, subjects actually had fewer task monitoring communications, but more communications relating to situational awareness. Pre-planning seemed to increase in the **# Bottles Unknown** condition.

- The team coverage did not differ in the **# Bottles Known** or **# Bottles Unknown** conditions; subjects reported that they were searching about the same amount of the map, although their bottle retrieval performance was lower.

- Team performance, measured by number of bottles retrieved, was poorer in **# Bottles Unknown** condition.

- Gaming expertise was predictive of individual bottle collecting performance, but not of an individual's terrain coverage.

Based on these results, we plan to focus our agent aiding on these areas:

- **reducing the cost of communication between teammates by having agents assist the subjects at increasing situational awareness and monitoring task progress**. We believe that this will free the humans' time to communicate about other aspects of the task, such as sharing search hints and team planning.

- **improving the coordination between team members at dividing the execution space** Subjects were not accurately able to self-assess their team coverage and often left holes in their search patterns. Helping teams accurately monitor team coverage is a very promising future area for agent assistance.

Listening to the recordings of the players was very valuable and gave us some insights. All teams adopted the common sense strategy of forming a team plan and dividing the search space. Most teams also replanned during execution when the following events occurred: (a) subjects finished their assigned coverage areas, (b) when new bottles were discovered, (c) as the deadline approached. Some teams evaluated themselves on terrain coverage whereas others focused on total bottle count. Often subjects formed hypotheses about areas with high bottle counts, similar to following false hunches in search and rescue operations, and speculated about the existence of hidden bottle caches. Although teams were allowed to self-organize, none of them elected a commander. Some players voluntarily assumed roles such as timekeeping or tallying bottle counts.

In the future, we plan to evaluate agent aiding in a version of the task that requires tighter coordination. By examining a task with more interdependencies, we believe that we will observe more planning, especially during execution. In the new version of the task, each subject has to retrieve a portfolio of seven bottles, one of each type (whiskey, martini, etc.). We hypothesize that coordination confers a huge benefit to the subjects if they pool information about bottle types that they have already acquired, the location of bottles that they do not need, and their portfolio requirements. Without team coordination, it is hard for even expert gamers to collect a portfolio of bottles, since it is more likely that they will collect a larger number of bottles without being able to find one of each type.

A.5 Conclusion

Our ongoing research objectives include understanding how agent-based team support affects team performance in critical, time-stressed situations and the impact of agents on

the adaptive decision-action cycle of the team. To understand the effects of agent aiding on team support, we designed a collaborative search task, the team scavenger hunt, that recreates some of the challenges faced by expert human teams during search and rescue operations. As an experimental task, the team scavenger hunt offers several advantages: (1) it can be learned and executed within a short period of time by novice subjects; (2) it can be simulated within a variety of testbeds; (3) it offers a simple team performance metric (4) it can be extended in different ways to evaluate concepts like team trust and adversarial reasoning. The team scavenger hunt problem touches on some interesting problems in artificial intelligence such as the multi-agent traveling salesman problem and preference satisfaction over sets of objects. We believe that it is a useful benchmark problem for other groups studying team behavior and agent assistance. This initial set of pilot experiments has allowed us to create a baseline of non-assisted team performance and also has given us some valuable clues as to where agent aiding is likely to have the greatest impact.

Bibliography

[1] J. Allen. Recognizing intentions from natural language utterances. In M. Brady and R. Berwick, editors, *Computational Models of Discourse*. MIT Press, 1983.

[2] J. Allen, H. Kautz, R. Pelavin, and J. Tennenberg. *Reasoning About Plans*, chapter 2, pages pp.121–148. Morgan Kaufmann Publishers, 1991.

[3] O. Arikan and D. Forsyth. Interactive motion generation from examples. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 483–490, 2002.

[4] O. Arikan, D. Forsyth, and J. O'Brien. Motion synthesis from annotations. *ACM Transactions in Graphics*, 22(3):402–408, 2003.

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

[6] T. Balch. `http://www.teambots.org/`, 2000. IIC, 85 Fifth Street Atlanta, Georgia, 30308.

[7] D. Ballard and C. Brown. *Computer Vision*. Prentice-Hall, 1982.

[8] J. Barbic, A. Safonova, J-Y. Pan, C. Faloutsos, J. Hodgins, and N. Pollard. Segmenting motion capture data into distinct behaviors. In *Proceedings of Graphics Interface 2004 (GI'04)*, 2004.

[9] M. Bauer. A Dempster-Shafer approach to modeling agent preferences for plan recognition. *User Modeling and User Adapted Interaction*, 5(4):pp.317–148, 1995.

[10] M. Beetz, J. Bandouch, S. Gedili, N v. Hoyningen-Huene, B. Kirchlechner, and A. Maldonado. Camera-based observation of football games for analyzing multi-agent activities. In *Proceedings of International Conference on Autonomous Agents and Multi-agent Systems*, 2006.

[11] M. Beetz, B. Kirchlechner, and M. Lames. Computerized real-time analysis of football games. *Pervasive Computing*, 4, 2005.

[12] B. Best and C. Lebiere. Spatial plans, communication, and teamwork in synthetic MOUT agents. In *Proceedings of Behavior Representation in Modeling and Simulation Conference (BRIMS)*, 2003.

[13] I. Bhandari, E. Colet, J. Parker, Z. Pines, R. Pratap, and K. Ramanujam. Advanced Scout: Data mining and knowledge discovery in NBA data. *Data Mining and Knowledge Discovery*, 1(1):121–125, 1997.

[14] H. Blum. A transformation for extracting new descriptors of shape. In *Proceedings of Symposium Models Perception Speech Visual Form*, 1964.

[15] H. Bui. A general model for online probabilistic plan recognition. In *Proceedings of International Joint Conference on Artficial Intelligence*, 2002.

[16] H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden markov model. *Journal of Artificial Intelligence Research*, 17, 2005.

[17] S. Carberry. Incorporating default inferences into plan recognition. In *Proceedings of National Conference on Artificial Intelligence*, 1990.

[18] H. Chalupsky et al. Electric Elves: Applying agent technology to support human organizations. In *Proceedings of the Innovative Applications of Artificial Intelligence Conference*, 2001.

[19] C.-C. Chang and C.-J. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[20] E. Charniak and R. Goldman. A Bayesian model of plan recognition. In *Proceedings of National Conference on Artificial Intelligence*, 1993.

[21] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence*. Addison Wesley, 1985.

[22] L. Chen and K. Sycara. Webmate: a personal agent for browsing and searching. In *Proceedings of the Second International Conference on Autonomous Agents*, 1998.

[23] P. Cohen and H. Levesque. Teamwork. *Nous*, 25(4), 1991.

[24] P. Cohen, C. Perrault, and J. Allen. Beyond question answering. In W. Lehnert and M. Ringle, editors, *Strategies for Natural Language Processing*. Lawrence Erlbaum Associates, 1981.

[25] R. Collins, A. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, and O. Hasegawa. A system for video surveillance and monitoring. Technical Report CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, May 2000.

[26] Dance Dance Revolution. Retrieved July 2007 `http://en.wikipedia.org/wiki/Dance_Dance_Revolution`.

[27] D&D Open Characters, 2006. `http://www.wizards.com/default.asp?x=rpga/conventions/genconindy-opench%aracters`.

[28] R. Duda, P. Hart, and D. Stork. *Pattern Classification*. Wiley & Sons, Inc, 2001.

[29] E. Entin and D. Serfaty. Adaptive team coordination. *Human Factors*, 41, 1999.

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

[31] X. Fan, B. Sun, S. Sun, M. McNeese, and J. Yen. RPD-Enabled agents teaming with humans for multi-context decision making. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2006.

[32] X. Fan, S. Sun, M. McNeese, and J. Yen. Extending the recognition-primed decision model to support human-agent collaboration. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2005.

[33] S. Fiore, E. Salas, and J. Cannon-Bowers. Group dynamics and shared mental model development. In M. London, editor, *How people evaluate others in organizations: Person perception and interpersonal judgment in industrial/organizational psychology*. Lawrence Erlbaum Associates, 2001.

[34] S. Fiore and J. Schooler. Process mapping and shared cognition: Teamwork and the development of shared problem models. In E. Salas and S. Fiore, editors, *Team Cognition: Understanding the Factors that Drive Process and Performance*. American Psychological Association, 2004.

[35] M. Fischler and R. Bolles. Random sample consensus: A paradigm for model fitting with application to image analysis and automated cartography. *Communications of the ACM*, 24(6), 1981.

[36] Operation Flashpoint: Cold War Crisis. `http://en.wikipedia.org/wiki/Operation_Flashpoint`, 2001. Accessed May 2007.

[37] J. Fowlkes, D. Washburn, S. Eitelman, J. Daly, and J. Cohn. Use of haptic displays to enhance training in a virtual environment. In *Proceedings of HCI International 2005*, 2005.

[38] J. Galbraith. *Organization Design*. Addison–Wesley Publication Company, 1977.

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

[40] R. Goldman, C. Geib, and C. Miller. A new model of plan recognition. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, 1999.

[41] M. Goodrich, M. Quigley, C. Humphrey, J. Adams, D. Gerhardt, J. Cooper, B. Buss, and B. Morse. Mini-UAVs for visual search in wilderness search: Tasks, autonomy, and interfaces. Technical Report BYUHCMI 2007-1, Brigham Young University, 2007.

[42] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.

[43] R. Hoeft, J. Kochan, and F. Jentsch. Automated team members in the cockpit: Myth or reality. In A. Schulz and L. Parker, editors, *Series: Advances in Human Performance and Cognitive Engineering Research*. Elsevier Science, 2006.

[44] A. Hoover, E. Muth, and F. Switzer. Clemson University MOUT Project, 2005.

[45] E. Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of SIGCHI*, 1999.

[46] M. Huber, E. Durfee, and M. Wellman. The automated mapping of plans for plan recognition. In *Procceedings of Uncertainty in Artifcial Intelligence (UAI)*, 1994.

[47] S. Intille and A. Bobick. Visual tracking using closed-worlds. Technical Report 294, MIT Media Lab, 1994.

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

[49] V. Jirsa. Context-independent team analysis. Presentation to VIRTE meeting, 2005.

[50] M. Jug, J. Pers, B. Dezman, and S. Kovacic. Trajectory based assessment of coordinated human activity. In *Proceedings of the International Conference on Computer Vision Systems (ICVS)*, 2003.

[51] G. Kaminka and M. Bowling. Towards robust teams with many agents. In *Proceedings of International Conference on Autonomous Agents and Multi-agent Systems*, 2002.

[52] G. Kaminka and M. Tambe. Robust agent teams via socially attentive monitoring. *Journal of Artificial Intelligence Research*, 12:pp.105–147, 2000.

[53] G. Kaminka, M. Veloso, S. Schaffer, C. Sollitto, R. Adobbati, A. Marshall, A. Scholer, and S. Tejada. Gamebots: A flexible test bed for multiagent team research. *Communications of the ACM*, 45(1), 2002.

[54] H. Kautz. *A Formal Theory of Plan Recognition*. PhD thesis, University of Rochester, 1987.

[55] Dave Kirkpatrick. It's not a game, 2007. Retrieved July 2007 `http://money.cnn.com/magazines/fortune/fortune_archive/2007/02/05/83991%20/`.

[56] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 473–482, 2002.

[57] B. Kraut, S. Fussell, F. Lerch, and A. Espinosa. Coordination in teams: Evidence from a simulated management game. *Journal of Organizational Behavior*, 2005.

[58] G. Kuhlmann, W. Knox, and P. Stone. Know thine enemy: A champion RoboCup coach agent. In *Proceedings of National Conference on Artificial Intelligence*, 2006.

[59] J. Lee, J. Chai, P. Reitsma, J. Hodgins, and N. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 491–500, 2002.

[60] T. Lenox. *Supporting teamwork using software agents in human-agent teams*. PhD thesis, Westminster College, 2000.

[61] T. Lenox, S. Hahn, M. Lewis, T. Payne, and K. Sycara. Agent-based aiding for individual and team planning tasks. In *Proceedings of IEA 2000/HFES 2000 Congress*, 2000.

[62] T. Lenox, M. Lewis, E. Roth, R. Shern, L. Roberts, T. Rafalski, and J. Jacobson. Support of teamwork in human-agent teams. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, 1998.

[63] T. Lenox, L. Roberts, and M. Lewis. Human-agent interaction in a target identification task. In *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics*, 1997.

[64] D. Lin and R. Goebel. A message passage algorithm for plan recognition. In *Proceedings of National Conference on Artificial Intelligence*, 1991.

[65] T. Livak. Collaborative warrior tutoring. Master's thesis, Worcester Polytechnic Institute, 2004.

[66] B. Mott, S. Lee, and J. Lester. Probabilistic goal recognition in interactive narrative environments. In *Proceedings of National Conference on Artificial Intelligence*, 2006.

[67] K. Murphy. The Bayes Net Toolbox for Matlab. *Computing Science and Statistics*, 33, 2001.

[68] Organic Motion's marker-less motion capture system makes its debut at GDC, 2007. Retrieved July 2007 `http://ncroal.talk.newsweek.com/default.asp?item=517804`.

[69] Parks Associaties. Online gaming revenues to triple by 2009, 2005. Retrieved July 2007 from `http://www.parksassociates.com/press/press_releases/2005/gaming-1.html`.

[70] D. Pearson and J. Laird. Redux: Example-drive diagrammatic tools for rapid knowledge acquisition. In *Proceedings of Behavior Representation in Modeling and Simulation Conference (BRIMS)*, 2004.

[71] J. Phillips, M. McCloskey, P. McDermott, S. Wiggins, and D. Battaglia. Decision-centered MOUT training for small unit leaders. Technical Report 1776, U.S. Army Research Institute for Behavioral and Social Sciences, 2001.

[72] D. Pynadath. *Probabilistic Grammars for Plan Recognition*. PhD thesis, University of Michigan, 1999.

[73] L. Rabiner. A tutorial on Hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77, 1989.

[74] T. Raines, M. Tambe, and S. Marsella. Automated assistants to aid humans in understanding team behaviors. In *Proceedings of the 4th International Conference on Autonmous Agents*, 2000.

[75] P. Reitsma and N. Pollard. Evaluating motion graphs for character navigation. In *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004.

[76] J. Rhodes. *Military Operations on Urbanized Terrain (MOUT)*, 1980.

[77] J. Rickel and W. Lewis Johnson. Virtual humans for team training in virtual reality. In *Proceedings of the Ninth International Conference on AI in Education*, pages 578–585. IOS Press, 1999.

[78] P. Riley and M. Veloso. On behavior classification in adversarial environments. In L. Parker, G. Bekey, and J. Barhen, editors, *Distributed Autonomous Robotic Systems 4*. Springer-Verlag, 2000.

[79] P. Riley and M. Veloso. Recognizing probabilistic opponent movement models. In A. Birk, S. Coradeschi, and S. Tadorokoro, editors, *RoboCup-2001: Robot Soccer World Cup V*. Springer Verlag, 2002.

[80] P. Riley, M. Veloso, and G. Kaminka. An empirical study of coaching. In H. Asama, T. Arai, T. Fukuda, and T. Hasegawa, editors, *Distributed Autonomous Robotic Systems 5*. Springer-Verlag, 2002.

[81] G. Rota. The number of partitions of a set. *American Mathematical Monthly*, 71, 1964.

[82] E. Salas and S. Fiore, editors. *Team Cognition: Understanding the Factors that Drive Process and Performance*. American Psychological Association, 2004.

[83] S. Saria and S. Mahadevan. Probabilistic plan recognition in multiagent systems. In *Proceedings of International Conference on AI and Planning Systems*, 2004.

[84] P. Scerri, D. Pynadath, L. Johnson, P. Rosenbloom, N. Schurr, and M. Tambe. A prototype infrastructure for distributed robot-agent-person teams. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2003.

[85] Second Life. Retrieved July 2007 `http://en.wikipedia.org/wiki/Second_Life`.

[86] K. Sentz and S. Ferson. Combination of evidence in Dempster-Shafer theory. Technical Report SAND2002-0835, Sandia National Labs, 2002.

[87] T. Setnicka. *Wilderness Search and Rescue*. Appalachian Mountian Club, 1980.

[88] G. Shafer. *A Mathemtical Theory of Evidence*. Princeton University Press, 1976.

[89] P. Stone and M. Veloso. Task decomposition, dynamic role assignment, and low-bandwidth communication for real-time strategic teamwork. *Artificial Intelligence*, 12:pp.241–273, 1999.

[90] G. Sukthankar, M. Mandel, K. Sycara, and J. K. Hodgins. Modeling physical capabilities of humanoid agents using motion capture data. In *Proceedings of International Conference on Autonomous Agents and Multi-agent Systems*, July 2004.

[91] G. Sukthankar and K. Sycara. A cost minimization approach to human behavior recognition. In *Proceedings of International Conference on Autonomous Agents and Multi-agent Systems*, 2005.

[92] G. Sukthankar and K. Sycara. Robust recognition of physical team behaviors using spatio-temporal models. In *Proceedings of International Conference on Autonomous Agents and Multi-agent Systems*, 2006.

[93] G. Sukthankar and K. Sycara. Simultaneous team assignment and behavior recognition from spatio-temporal agent traces. In *Proceedings of National Conference on Artificial Intelligence*, July 2006.

[94] G. Sukthankar and K. Sycara. Policy recognition for multi-player tactical scenarios. In *Proceedings of International Conference on Autonomous Agents and Multi-agent Systems*, 2007.

[95] G. Sukthankar, K. Sycara, J. Giampapa, C. Burnett, and A. Preece. Towards a model of agent-assisted team search. In *Conference of the International Technology Alliance in Network and Information Sciences*, 2007.

[96] W. Swartout, J. Gratch, R. Hill, E. Hovy, S. Marsella, J. Rickel, and D. Traum. Towards virtual humans. *AI Magazine*, 27(1), 2006.

[97] K. Sycara and M. Lewis. Integrating agents into human teams. In E. Salas and S. Fiore, editors, *Team Cognition: Understanding the Factors that Drive Process and Performance*. American Psychological Association, 2004.

[98] Tactical Language and Culture Training System, 2007. Retrieved July 2007 `http://www.tacticallanguage.com/tacticaliraqi/aboutus.htm`.

[99] M. Tambe. Tracking dynamic team activity. In *Proceedings of National Conference on Artificial Intelligence*, 1996.

[100] M. Tambe and P. Rosenbloom. RESC: An approach to agent tracking in a real-time dynamic environment. In *Proceedings of International Joint Conference on Artficial Intelligence*, 1995.

[101] R. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal of Computing 1*, pages 146–160, 1972.

[102] Teamspeak communication system. `http://www.goteamspeak.com`, 2001. Accessed May 2007.

[103] E. Todorov. Optimality principles in sensorimotor control. *Nature Neuroscience*, 7(9), 2004.

[104] D. Traum, J. Rickel, J. Gratch, and S. Marsella. Negotiation over tasks in hybrid human-agent teams for simulation-based training. In *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2003.

[105] V. Vapnik. *Statistical Learning Theory*. Wiley & Sons, Inc, 1998.

[106] M. Vilain. Getting serious about parsing plans: A grammatical analysis of plan recognition. In *Proceedings of National Conference on Artificial Intelligence*, 1990.

[107] Wii Remote. Retrieved July 2007 `http://en.wikipedia.org/wiki/Wii_Remote`.

[108] d20 v3.5 System Reference Document, 2003. `http://www.wizards.com/default.asp?x=d20/article/srd35`.

[109] G. Xu and Z. Zhang. *Epipolar Geometry in Stereo, Motion, and Object Recognition*. Kluwer, 1996.

[110] R. Yager. On the Dempster-Shafer framework and new combination rules. *Information Sciences*, 41:pp.93–137, 2000.

[111] C. Yang, R. Duraiswami, and L. Davis. Fast multiple object tracking via a hierarchical particle filter. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2005.

[112] J. Yin, X. Chai, and Q. Yang. High-level goal recognition in a wireless LAN. In *Proceedings of National Conference on Artificial Intelligence*, 2004.