Efficient Hybrid Algorithms for Plan Recognition and Detection of Suspicious and Anomalous Behavior

Dorit Avrahami-Zilberbrand
Computer Science Department

Ph.D. Thesis

Submitted to the Senate of Bar-Ilan University
Ramat Gan, Israel
March 2009

This work was carried out under the supervision of Prof. Gal A. Kaminka, Computer Science Department, Bar-Ilan University.

Dedication

This dissertation is lovingly dedicated to my parents, Gila and Magid Avrahami, who raised me to be the person I am today. Their unconditional love, guidance, encouragement and support have sustained me throughout my life. Thank you for everything. I love you!

Acknowledgments

I would like to express my deep gratitude to my thesis advisor, Prof. Gal A. Kaminka, for his encouragement, for thoughtful guidance, for his support and understanding during the course of research, the best advisor I could have asked for. This work could not have been carried out without him.

I would also like to thank all of the members of the Maverick research group, especially to Efi Merdler for his assistance.

Also, I wish to express my gratitude to my husband Nadav, who has been a great source of motivation and inspiration. I thank him for his love, patience and practical assistance in my research. I thank our daughter Noa, who has been a source of great happiness and joy.

I wish to thank my family, for providing a loving environment for me. To my parents, Magid and Gila, my sister Ronit and my brother Tomer, for always being there when I needed them.

Finally, I would like to express special gratitude to Batya and Menahem Zilberbrand for their support.

This research was carried out with partial support by the Israeli Ministry of Commerce AVNET Consortium, and the Israel Science Foundation Grant #1357/07.

Abstract

Plan recognition is the process of inferring other agents' plans and goals based on their observable actions. Modern applications of plan recognition, in particular in surveillance and security raise several challenges. First, a number of key capabilities are missing from all but a handful of plan recognizers: (a) handling complex multi-featured observations; (b) dealing with plan execution duration constraints; (c) handling lossy observations (where an observation is intermittently lost); and (d) handling interleaved plans. Second, essentially all previous work in plan recognition has focused on recognition accuracy itself, with no regard to the use of the information in the recognizing agent. As a result, low-likelihood recognition hypotheses that may imply significant meaning to the observer, are ignored in existing work. In this work we present set of efficient plan recognition algorithms that are capable of handling the variety of features required of realistic recognition tasks. We also present novel efficient algorithms that allow the observer to incorporate her own biases and preferences—in the form of a utility function—into the plan recognition process. This allows choosing recognition hypotheses based on their expected utility to the observer. We call this Utility-based Plan Recognition (UPR). We demonstrate the efficacy of the techniques described above, by applying them to the problem of detecting anomalous and suspicious behavior. The system contains the symbolic plan recognition algorithm, which detects anomalous behavior, and the utility-based plan recognizer which reasons about the expected cost of hypotheses. These two components form a highly efficient hybrid plan recognizer capable of recognizing abnormal and potentially dangerous activities. We evaluate the system with extensive experiments, using real-world and simulated activity data, from a variety of sources.

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

Plan recognition focuses on mechanisms for recognizing the unobservable state of an agent, given observations of its interaction with its environment. Plan recognition has been extensively investigated in the past two decades (e.g., [22,23,31,51,70,87]). When applied to observations of human activity, it is also referred to as *activity recognition* (e.g., [19,27,42,55,64]).

Plan- or activity- recognition is used in a wide range of applications, such as intrusion detection applications [31,32], virtual training environments [87], visual monitoring [18] and detection of suspicious or anomalous behavior [66,70,91].

Most approaches to plan recognition utilize a plan library, which encodes the behavioral repertoire of a typical observed agent. Observations are matched against this plan library in sequence, and resulting recognition hypotheses are often ranked according to their likelihood or via some other ranking method. In general, plan recognition libraries have a complex structure, that encode (explain) a large number of possible observation sequences. During run-time, an observation sequence is matched against the plan-library, and matching sequences within the plan library are treated as plan recognition hypotheses.

Modern applications of plan recognition challenge existing work in this area. In particular, recent applications of plan recognition to human *activity recognition* monitoring and surveillance have raised several challenges:

- First, a number of key capabilities are missing from all but a handful of plan recognizers. Most existing investigations, for instance, ignore the computational cost of matching complex multi-feature observations

against all possible plan-steps in the plan library (a non-trivial task). Moreover, most recognizers do not allow recognition based on duration of behaviors, nor do they address recognition despite intermittently lost observations. Finally, most existing work ignores recognition of interleaved goals. We survey previous work in detail in Chapter 2.

- Second, essentially all plan recognition algorithms are task-neutral. They generate a list of hypotheses—typically ranked by decreasing likelihood—and leave it to the decision-making component to examine the hypotheses and draw a conclusion leading to taking action. This is a classic generate-and-test design.

  But as generating the full list of hypotheses is expensive, many plan recognizers only return a handful of the top-ranked hypotheses (e.g., the top three likely hypotheses). This necessarily causes these plan recognizers to ignore the significance of hypotheses to the observer. As a result, low-likelihood recognition hypotheses that may imply significant danger or opportunity to the observer, are ignored.

  For instance, suppose we observe a rare sequence of unix commands that can be explained by for some plan I or for a more common plan L. Most plan recognition systems will prefer the most likely hypothesis L, and ignore I. Yet, if the expected utility (risk) of I for the observer is high (e.g., if I is a plan to take down the computer system), then that hypothesis should be preferred when trying to recognize suspicious behavior. If I has expected utility that is very low for a malicious user, then L may be better.

- Finally, there has been very little work on extending these single-agent plan recognition frameworks to multi-agent scenarios. Plan recognition with multi-agent settings can be performed, in principle, by treating agents as independent, and recognizing the plan of each agent separately. However, some scenarios require agents to participate in dynamic coordinated tasks, where team membership changes over time. Tracking individual agents independently fails to recognize a team joint goal and activities [50]; it will thus fail to capture recognizing the behavior of agents with respect to their group.

In the first part of this dissertation, we address the challenges listed above. In Chapter 3 we build on our previously published work [4, 5], to create a

new comprehensive symbolic plan recognizer that is capable of handling the variety of features required of realistic recognition tasks. To the best of our knowledge, this recognizer has the fastest plan-recognition algorithms today. We then use it as a basis for a hybrid recognizer which reasons about the expected utility of hypotheses (Chapter 4), and thus takes the expected costs (and gains) of an hypothesis into account. The symbolic recognizer also serves as the basis for a multi-agent plan-recognition framework (Chapter 5), which allows tracking the relationships between agents.

To demonstrate the efficacy of the techniques described above, in the second part of the work we apply them to the problem of detecting anomalous and suspicious behavior. The system we describe contains two key components: The symbolic plan recognizer (Chapter 3) is used to detect anomalous behavior (Chapter 6), and the utility-based plan recognizer (Chapter 4) which reasons about the expected cost of hypotheses is used to detect suspicious behavior (Chapter 7). Together, the two components form a highly efficient hybrid plan recognizer capable of recognizing abnormal and potentially dangerous activities.

We evaluate the two components in extensive experiments, using real-world and simulated activity data, from a variety of sources. We show that the techniques are able to detect both anomalous and suspicious behavior, while providing high levels of precision and recall (i.e., small levels of false positives and false negatives).

Below we discuss the contributions of the dissertation in greater detail. In Section 1.1 we describe our contributions to the problem of plan recognition. In Section 1.2 we introduce our work on detecting anomalous and suspicious behavior.

1.1 An Efficient Hybrid Model for Plan Recognition

This section discusses the contributions of the dissertation in the area of plan recognition. We begin by discussing the improvements for SBR, efficient symbolic plan recognition 1.1.1. Then we discuss the contributions of UPR, utility based plan recognition in Section 1.1.2. And then, our work in multi-agent framework 1.1.3.

1.1.1 SBR: Efficient Symbolic Plan Recognition

Symbolic plan recognizers generate hypotheses as to the unobservable state of an agent consistent with the observations, with no ranking (probabilistic or otherwise). Because they provide no ranking, symbolic plan recognizers are often viewed as inadequate for modern applications. Indeed, even early instantiations of symbolic recognizers have utilized some heuristics to rank hypotheses, e.g., [50, 51, 87].

We posit that highly efficient symbolic recognizers can in fact be very useful. First, in applications of detecting anomalous behavior, a symbolic recognizer can quickly discard all invalid hypotheses without wasting effort on computing their likelihood or value. Second, symbolic plan recognizers can be used as a basis for hybrid plan recognition systems (see [31,33,49,77]). Here, the symbolic recognizer is used to efficiently rule out zero-likelihood (or zero-value) hypotheses. Thus the more computationally-intense probabilistic or utility-based reasoning process focuses only on valid hypotheses.

However, to support modern applications of plan recognition, a number of capabilities are needed. These are not present in most —if not all— symbolic plan recognizers.

First, many applications have complex multi-feature observations, rather than a single atomic feature. Most existing investigations ignore the computational cost of matching complex multi-feature observations against all possible plan-steps in the plan library, this is non-trivial task, and should be taken into account. In our earlier work [4, 5], we presented specialized data-structure, a matching decision-tree called FDT (Feature Decision Tree). The FDT is generated automatically once prior to execution, and it efficiently matches multi-feature observations, to the plan library. Clearly, the use of the FDT leads to very significant improvements in the matching time compared to previous work. However, its space complexity is exponential in the number of features.

Second, previous investigations typically do not utilize information on the execution duration of plans. It is possible to explicitly reason about time, and thus for example demand minimum and maximum durations for each plan. This information can be used to rule out hypotheses that match instantaneous observations, but whose hypothesized duration does not match observations over time.

Third, most previous investigations are not capable of coping with agents that pursue multiple goals. The models consider multiple goals only in se-

quence, where the observed agent finishes a series of plan-steps in order to finish one goal, and only then moves on to pursuing another goal. While sequential goals are certainly common in some domains, in many others agent can start with one goal, then move to another goal, and finally return to accomplish the first goal, from the point it has paused. One simple example of this is where we attempt to recognize the goal of a web user, who stops in the middle of navigating a web page and jumps to a news site, only to return to her previous work afterwards.

An ideal plan recognition system would be able to address the deficiencies above, taking into account that observations might be lost, due to sensory failures.

Specifically, extending SBR, makes the following contributions: (a) reducing space complexity of matching complex multi-featured observations to the plan library; (b) dealing with plan execution duration constraints; (c) handling lossy observations (where an observation is intermittently lost); and (d) handling interleaved plans (where an agent interrupts a plan for another, only to return to the first later). This symbolic model is highly efficient and will be used later as a basis for a hybrid symbolic-probabilistic recognizer, and for the multi-agent framework.

The recognition algorithms we develop follow in the footsteps of [4, 5] in their focus on efficiency. They rely on lazy commitment to hypotheses, to avoid computation of hypotheses with every step (as other algorithms do, e.g., [33]). Instead, they use linear-time bookkeeping with every observation, which allows extraction of hypotheses only as needed.

In the second part of the dissertation (Part II), we demonstrate the capabilities of these algorithms in the application of detecting anomalous behavior. We evaluate the system with extensive experiments, using real-world data from machine vision trackers, which track movements of people, and report on their coordinate positions.

1.1.2 UPR: Efficient Utility-based Plan Recognition

Essentially all plan recognition techniques ignore the decision processes of the recognizing agent, and focus on probabilistic or heuristic ranking of recognition hypotheses, with no regard to the task for which knowledge of the plans of others is needed. As a result, low-likelihood recognition hypotheses that may carry significant gains or costs to the observer, might be ignored.

For instance, suppose we observe a sequence of Unix commands that can

be explained by some plan I or by a more common plan L. Probabilistic plan recognition systems will prefer the most likely hypothesis L, and ignore I. This, in fact would be a better hypothesis for general recognition. Yet, if the expected cost (risk) of I for the observer is high (e.g., if I is a plan to take down the computer system), then that hypothesis should be preferred when trying to recognize suspicious behavior; the task biases the ranking of hypotheses.

We propose a novel plan recognition approach, *utility-based plan recognition* (UPR), in which the observer folds its biases and preferences—in the form of a utility function—into the plan recognition process itself. Using UPR, the recognition process ranks recognition hypotheses based on their *expected utility to the observer*. This allows the observer, for instance, to select hypotheses based on their expected costs (e.g., in the case of a risk-averse observer), or expected gains.

We present a formal procedure for constructing Plan Recognition Influence Diagrams (PRID), based on Charniak and Goldman's [23] procedure for constructing Bayesian Networks. Unfortunately, while in principle UPR can be carried out via influence diagrams, such reasoning is intractable in the general case [41, 67], and specifically in the types of influence diagrams considered for UPR (see Section 4.1.3 for a detailed discussion).

We therefore present an efficient UPR recognizer, able to carry out plan recognition in worst-case complexity of $O(NDT)$, where N is the size of a hierarchical plan library, D is the depth of the library, and T is the number of observations. This complexity is achieved by using a hybrid approach that combines an efficient symbolic plan recognizer [5, 12], with a decision-theoretic inference built on top of a hierarchical Markov model. We restrict these algorithms to the case of *keyhole recognition*, where the observed agent does not modify its behavior based on the knowledge that it is being observed.

In the second part of the dissertation (Part II), we demonstrate the novel capabilities of UPR, and its efficient implementation as described above. We tested the capabilities of our algorithms in three different recognition tasks in the application of suspicious behavior. The domain for the first task consisted of recognizing passengers that leave articles unattended, as in the example above. In the second task we show how our algorithms can catch a dangerous driver that cuts between two lanes repeatedly. The last experiment intends to show how previous work, which has used costs heuristically [87], can now be recast in a principled manner. All of these examples show that we should not ignore the observer biases, since the most probable hypothesis sometimes

masks hypotheses that are important for the observer.

1.1.3 Detecting Multi-Agent Dynamic Groups

This work takes first steps to address the challenge of plan recognition for dynamic multi-agent teams. Most previous work has focused on recognizing specific (and limited) coordinated behaviors and does not deal with the problem of identifying interactions between groups of agents. In contrast, this work utilizes the information from group of agents, to identify the interactions between groups of agents, using a Dynamic Hierarchical Group Model (DHGM) that tracks the dynamic grouping and un-grouping of agents.

There are behaviors that can be captured only when tracking individuals with respect to the group and not as individuals. For example, identifying passenger in the airport that behaves differently from other passengers in the same group. While reasoning about individual agents in a multi-agent framework is expensive, we reduce this complexity by utilizing the DHGM that encapsulate shared data of agents in the same group.

Although multiple frameworks have been developed for single-agent plan recognition, there has been less work on extending these frameworks to multi-agent scenarios. Plan recognition with multi-agent settings can be performed, in principle, by treating agents as independent, and recognizing the plan of each agent separately. However, there are number of problems with this method in complex multi-agent scenarios. First, some scenarios require agents to participate in dynamic teams where team membership changes over time. Tracking individual agents independently fails to recognize a team joint goal and activities [86]; it will thus fail to capture recognizing the behavior of agents with respect to their group.

Previous work has shown that given a model of hierarchical relationship between agents, one can recognize team plans, involving multiple agents [44, 46,49]. However, these previous work focused specific social structures, where agents form teams based on a-priori agreements as to specific plans. In order to recognize team plans in these previous methods, the monitoring agent must first know which plans are ideally to be agreed upon. In contrast, in our work we do not have static social structure that is given in advanced, but instead use the plan library to identify dynamically changing structure of the groups. For example, a group of passengers in the airport may seem like one group when standing in the security check line, and afterward when splitting to two groups, the organizational structure need to be modified.

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

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

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

1.2 Detecting Anomalous and Suspicious Behavior

This part of the work discusses the application of detecting anomalous and suspicious behavior. We utilize the new model for plan recognition presented in the first part to detect anomalous and suspicious behavior.

There have been several attempts at utilizing plan recognition for recognition of suspicious, erroneous, or anomalous behavior, e.g., [18,25,28,32,56, 66,91]. These have mostly operated under the assumption that a plan library is available that covers this intended *negative behavior*, and thus recognition of hypotheses implying such behavior is treated no differently from recognition of hypotheses implying no suspicion. Recently, a different approach has been taken by several researchers in which the plan library is used in an inverse fashion. The plan library is limited to covering only *positive behavior*. When a plan-recognizer is unable to match observations against the library (or generates hypotheses with very low likelihood), an anomaly is announced [27,55].

Second, essentially all previous work on recognizing anomalous plans fo-

cuses on probabilistic ranking of hypotheses, and ignores the utility of actions to the observer. As a result, low-likelihood recognition hypotheses that may imply significant danger are ignored. For instance, a sequence of observations may indicate a person leaving her bag behind. This is rare—but not necessarily negative—behavior (some would say, especially among young children and adult professors), and should not always be treated as a suspicious behavior. It is the potential impact of this act that makes the act suspicious.

This work addresses these challenges. We use the efficient algorithms for plan recognition introduced in the first part (Part I) of the work, to consider the problem of detecting anomalous and suspicious behavior.

First, we use the symbolic plan recognitions algorithms for detecting abnormal activity (the plan library represent normal behavior; any activity which is not recognized is abnormal). For example, the normal activity model may include usual movements of people in the airport. The Symbolic algorithm will detect abnormal patterns like walking in the wrong direction, taking more than usual amount of time to get to the security check. The SBR algorithms introduced in the first part, handle the variety of features required for this domain: recognition based on duration, handling lossy observation streams, etc.

Second, we use the efficient hybrid utility based plan recognizer, to detect suspicious behavior in three different tasks. The domain for the first task consisted of recognizing passengers that leave articles unattended, as in the example above. In the second task we will show how our algorithms can catch a dangerous driver that cuts between two lanes repeatedly. The last experiment intends to show how previous work, which has used costs heuristically [87], can now be recast in a principled manner. All of these examples show that we should not ignore the observer biases, since the most probable hypothesis sometimes mask hypotheses that are important for the observer.

1.3 Dissertation Organization

This dissertation is constructed of 9 chapters, organized in two main parts (see Figure 1.1). This chapter constitutes the introduction to this thesis. The next chapter surveys the related work. Chapters 3–5 constitute Part I of the dissertation, which deals with new efficient algorithms for plan recognition. Chapters 6–7 constitute Part II of the dissertation, which deals with detecting anomalous and suspicious behavior. Chapter 8 concludes and discusses future

work.

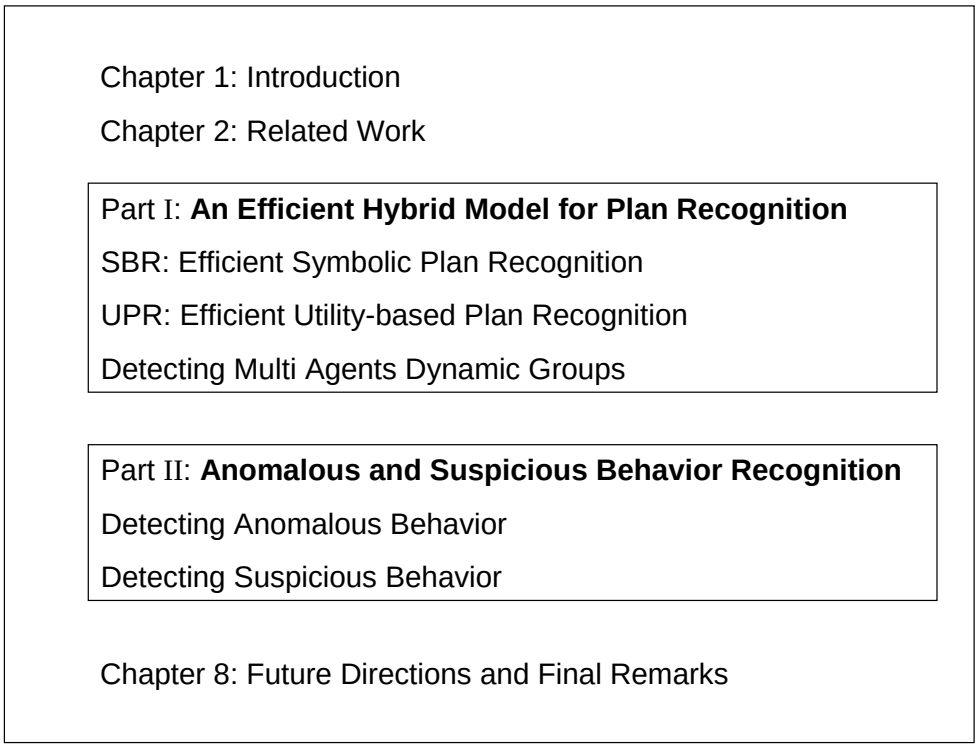

Figure 1.1: Thesis structure.

1.4 Publications

Subsets of the results that appear in this dissertation were published in the proceedings of the following refereed journals, conferences and workshops:

- Dorit Avrahami-Zilberbrand, Gal A. Kaminka and Hila Zarosim. Fast and complete plan recognition: Allowing for duration, interleaved execution, and lossy observations. In Proceedings of the IJCAI Workshop on Modeling Others from Observations (MOO-05), 2005. [12].

- Dorit Avrahami-Zilberbrand and Gal A. Kaminka. A Hybrid symbolic-probabilistic plan recognizer: Initial steps. In Proceedings of the AAAI Workshop on Modeling Others from Observations (MOO-06), 2006. [6].

- Dorit Avrahami-Zilberbrand and Gal A. Kaminka. Incorporating observer biases in keyhole plan recognition (efficiently!). In Proceedings of Twenty-Second National Conference on Artificial Intelligence (AAAI-07), Vancouver, British Columbia, 2007. [7].

- Dorit Avrahami-Zilberbrand and Gal A. Kaminka. Utility-based plan recognition: An extended abstract (short paper). In Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-07), 2007. [9].

- Dorit Avrahami-Zilberbrand and Gal A. Kaminka. Towards dynamic tracking of multi-agents teams: An initial report. In Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR-07), 2007. [8].

- Dorit Avrahami-Zilberbrand and Gal A. Kaminka. Fast symbolic plan recognition. Submitted to Artificial Intelligence (AIJ), 2009. [10].

- Dorit Avrahami-Zilberbrand and Gal A. Kaminka. Incorporating Observer Biases in Keyhole Plan Recognition (Efficiently!). Submitted to Artificial Intelligence (AIJ), 2009. [11].

Chapter 2

Related Work

2.1 Plan Recognition

Plan recognition is the task of inferring the intentions, plans, and/or goals of an agent based on observations of its actions. We begin by discussing symbolic and hybrid approaches to plan recognition (Section 2.1.1), then probabilistic approaches (Section 2.1.2), then utility-based approaches (Section 2.1.3) and then multi-agent plan recognition (Section 2.1.4). For a comprehensive survey, see [22].

2.1.1 Symbolic and Hybrid-Symbolic Approaches

Symbolic plan recognition algorithms find hypotheses as to the current intention, goal, or plan of the agent, without considering the relative likelihood of the hypotheses. Early work by Kautz [51] provided a formal theory of plan recognition, focusing on symbolic methods. In this work the problem is viewed as a deductive inference, and relies on a representation called *action taxonomy*, where every observed action is a part of one or more "top level plans". The task of the plan recognition is to find minimal set of top plans that explain the observations. However, this approach faces inherent difficulties when applied to the complex, dynamic settings in which agents are typically deployed. First, agents may take continuous (servo) actions, intended to maintain some interaction with their environment (e.g., velocity or heading). The actions in this case are not discrete and instantaneous, but instead are composed of multiple continuous changes to approximate some target function [65]. Modelling such actions using discrete operators is diffi-

cult at best. Second, since the input to classic plan recognition algorithms is a stream of actions, it is difficult to incorporate an additional context that may be observable, and may be affecting the internal decision-making of the observed robot. For instance, an observed person may be of a certain height, which is observable, but is certainly not an observed action—and is ignored by Kautz's method. Moreover, Kautz's method does not account for the duration of plan execution.

Several investigations have utilized multi-featured observations (also of continuous actions), but did not address the efficiency of matching observations to the plan library, in contrast to our work: RESC [87] and RESL [50] use a hierarchical representation (similar to what we use) to maintain a single hypothesis (RESC) or multiple hypotheses (RESL) as to the current state of an observed agent. Both algorithms ignore observation history in the current state hypotheses, in contrast to our algorithms. Moreover, these algorithms do not account for complex temporal reasoning (duration and interleaving).

Recently, there appeared a number of plan recognition techniques that use a hybrid approach: An underlying symbolic recognizer is used to efficiently filters inconsistent hypotheses, passing them to a probabilistic inference engine, which focuses on ranking recognition hypotheses. Indeed, the techniques described in this work have been utilized with a hybrid recognizer that focuses on recognizing the most costly hypotheses, in a decision-theoretical sense [7].

Geib et al. [33] developed PHATT, a hybrid recognizer, where a symbolic algorithm filters inconsistent hypotheses before they are considered probabilistically. PHATT assumes instantaneous, atomic actions, and takes a generate-and-test approach: With each observation, the symbolic algorithm generates a *pending set* of possible expected observations, which are matched against the next observation to maintain correct state history hypotheses. The size of the pending set may grow exponentially [31]. In contrast, our work decouples the current state and state history queries, and incrementally maintains hypotheses implicitly, without predicting impending observations. The hypotheses are thus computed only when needed (when hopefully, many of them have been ruled out). [33] describe a hybrid symbolic-probabilistic plan recognition system, allowing for interleaved plans as well as some partial observability. However, the system does not allow for taking durations into account, nor does it address efficient matching of (lossy) multi-feature observations.

YOYO* [49] is a hybrid probabilistic plan recognition algorithm for multi-agent overhearing. The plan-library used by YOYO* included information

about the average duration of plan steps, which is used to calculate the likelihood of an agent terminating one step and selecting another without being observed to do so. In this, YOYO* addressed missing observations (though their likelihood of becoming lost is assumed to be provided a-priori). However, in contrast to our work, YOYO* did not address matching multi-feature observations (where some features may be intermittently lost), nor did it allow for interleaved plans.

Retz-Schmidt [77] describes a system that takes the image sequences of the robotic soccer game, translates it to natural language, and translates it to a plan library. There is a top down process of matching preconditions to paths in the plan library. Then, the matching subgraphs are extracted and compared with the next observation. If there are more than one hypothesis there is a probabilistic inference. However, the approach does not allow for interruptions of plans, nor for complex temporal reasoning.

2.1.2 Uncertainty in Plan Recognition Hypotheses

Typically, probabilistic plan recognition approaches, and other methods for reasoning under hypothesis uncertainty, focus on determining the most likely hypothesis, or set of hypotheses, as to the current and/or past states of the agent. Previous work in this area has focused on utilizing specialized structures and techniques that allows more efficient recognition, or more expressive forms of plan recognition.

In contrast to all of these, we present non-probabilistic plan-recognition algorithms: either symbolic, or decision-theoretic. In our presentation of utility-based plan recognition, we present both a general method based on influence diagram, as well as a highly efficient hybrid recognizer (which sacrifices some of the expressivity of the more general model).

The first principled approach for reasoning under uncertainty in recognition hypotheses was introduced by Carberry [21], who used Dempster-Shafer belief functions. A different recognition system, but based on the same underlying theory, was later used by Bauer et al. [13].

Charniak and Goldman [23] presented a different method for uncertainty reasoning in plan recognition, based on probability theory. They presented a formal procedure for constructing Plan Recognition Bayesian Networks, as part of a story understanding system. Huber et al. [43] also generate probabilistic belief networks from plan libraries encoding BDI procedures to solve plan recognition problems. Later work [3, 92] has utilized Dynamic Bayesian

Networks (DBN) for more complex plan recognition tasks. Applications to visual surveillance also utilized belief networks (e.g., [38, 39]).

In all of these, the goal of the probabilistic methods is to calculate the conditional probability of a set of variables (plans) given the values of another set of variables (the observations). There are two types of techniques: (a) exact inference (b) approximate inference. It is known that exact inference of BN is intractable with respect to the network size [24], as it is in DBN [16, 52]. Approximate inference, although scaling with the network size, is NP-Hard with respect to the hard-bound of the estimates.

As recognition tasks grew in complexity, more efficient models were needed. There have been much work that trades expressivity for runtime, e.g., by introducing a Markov assumption into the models. Hidden Markov Models (HMMs) and extensions have been explored extensively ([17, 20, 29, 35, 37, 62, 75]) for plan recognition.

An *HMM* is the most simple case of *DBN*, where in each time slice there is only a single state variable and an observation node. HMMs explicitly represent uncertainty in observations. This model has been widely used in traditional activity recognition for representing and learning characteristics in simple activities (e.g., [80, 94]. The complexity of the HMM is $O(TN^2)$ where N is the number of the states and T is the number of observations (time steps). When the activities become more complex, or has long-term temporal dependency, this model is not adequate.

Coupled HMMs [17] and *Factorial HMM* [34] are extensions of the HMM with multiple hidden interacting chains. In these models, the size of the belief state is exponential in the number of hidden chains. For this reason, approximate inference techniques are required. These models can handle complex activities, but are not sufficient to model hierarchical activities.

The *AHMM* [20], HHMM [29] and Layered HMM [69] are capable of handling activities that have hierarchical structure. These models are used in activity recognition applications such as learning and detecting activities from movement trajectories [64] and inferring from raw GPS sensor user's daily movements ([55]). These models are using an approximate inference algorithms for the recognition process. In our work, we follow in the footsteps of HHMM in representing probabilistic information in the plan library (see Section 4.2.1 for details).

Another class of extensions is exploring more complex formalisms for reasoning under uncertainty in plan recognition, in order to improve expressivity while maintaining efficiency. There has been recent work on using Hidden

Semi-Markov Models (HSMMs) in recognizing plans and activities [27, 59]. Hidden Semi-Markov Models allow for providing probabilistic constraints over the duration of plans, as well as the ability to detect anomalies. Recent work has moved beyond DBNs and Markov models to use Conditional Random Fields and similar techniques [88] which allow for greater expressivity. Blayblock and Allen [15] provided a novel HMM-based model that allows efficient exact reasoning about hierarchical goals. Hu and Yang [42] model interacting and concurrent goals (which we do not focus on in this work). Bui et al. [19] introduce a Hidden Permutation model that can learn the partial ordering constraints in location-based activity recognition.

There are also *probabilistic grammar approaches*, which allow for efficient exact inference: using a *PCFG* (Probabilistic Context-Free Grammar) to represent observed agent behavior turns plan recognition into a parsing task [73]. This method works by constructing a Bayesian network to represent the distribution of parses trees. The *PSDG* (Probabilistic State-Dependent Grammars [74]) extends the PCFG by allowing the probabilities production to depend on the state of the observed agent. The major problem with parsing as a model for plan recognition is that it does not address missing observations nor interleaved plans.

Our work differs significantly from probabilistic approaches, as UPR reasons about expected utility of hypotheses, rather than their likelihoods. All of the probabilistic approaches presented here ignore the use of utilities. Moreover, none of these probabilistic approaches consider combining symbolic inference to reduce complexity and to allow a richer class of plan recognition inference, as we do.

2.1.3 Utility-Based Approaches

There have been a few investigations of the use of utilities or costs in plan recognition. We distinguish previous work discussing the utility of others' actions to themselves, and previous work discussing the utility of others' actions to the observer.

Most existing work addresses utilities of the other agent's actions to itself, in contrast to our work. There exist numerous investigations of (sequential) multi-agent decision-making, which models the entire decision-making process of a rational agent reasoning about others (e.g., Recursive Modeling Method (RMM) [36, 67]; Multi-Agent Influence Diagrams [53, 83]; and POMDP variants [14, 76, 89]). Such works focus on the decision making, and

deemphasize the recognition process—which is complex in itself—necessary to establish the hypotheses underlying the decisions.

Mao and Gratch [57, 58] have explored explicit modeling of observed agents' utilities as part of ranking recognition hypotheses. Here, equally-likely hypotheses are ranked based on the preferences of the observed agent, as expressed in its own utilities, and under the assumption of rationality. Similarly, Suzic [84] proposes a generic framework for tactical plan recognition using Multi-Entity Bayesian Networks (MEBN). MEBN also take into account a-priori knowledge of the utility, given plans. Pynadath and Marsella [71, 72] have developed a model with decision-theoretic approach, including beliefs about the modeling agent's environment and recursive models of other agents. They modified the POMDPs algorithm to take into account psychological models instead of the assumption of perfect rationality. However, the agent is seeking to maximize the expected reward of the observed agent behavior as in a POMDP. Our work differs from theirs in that we consider the impact of recognition hypotheses on the observer, not on the observed.

Indeed, there exists a conceptual difficulty with this approach. It can be argued that the utility function of the observed agent is already accounted for by the a-priori probabilities of the hypotheses: The fact that most people do not consider planting a bomb to be an action with high expected utility means that the a-priori probability of this (in the plan-library) is very low. Thus, using the utility function of the observed may not even add meaningful information to the recognition process. However, it may add to the expressivity of the plan library.

Sukthankar and Sycara [81] present a cost minimization approach to behavior recognition, in which the recognizer uses behavior transition cost function to select the most parsimonious, minimal-cost, recognition hypothesis explaining the human's actions. However, the cost is to the observed agent, transitioning between behaviors, and is intended to increase recognition coherence.

More closely-related work examined reasoning about the utility of recognition hypotheses *for the observer*. Tambe and Rosenbloom [87] have examined the use of reactive plan recognition in simulated air-combat domains. Here, the observing agent may face ambiguous observations, where some hypotheses imply extreme danger (a missile being fired towards the observer), and other hypotheses imply gains (the opponent running away). RESC takes a heuristic approach that prefers hypotheses that imply significant costs to the observer (e.g., potential destruction). The relative likelihood of such hy-

potheses is ignored. While we are inspired by this work, we take a principled, decision-theoretic, approach. In the algorithms we present, the likelihood of hypotheses is combined with their utilities, to calculate the expected impact on the observer. We show in Section 7.3 that this subsumes the earlier, heuristic work.

2.1.4 Dynamic Tracking of Multi-Agent Teams

YOYO [46] is a symbolic approach for detecting disagreements among team members. This work exploits knowledge on the social structures of the team (called team hierarchy) to efficiently recognize splits in teams, where an agent is executing a different plan than the rest of its team. In order to detect disagreements, the monitoring agent must first know which plans are ideally to be agreed upon. In contrast, in our work we do not have a static social structure that is given in advanced, but a dynamically track structure of groups which changes over the time. However, while we track splits in the team, we cannot categorically determine that a split is an anomaly.

$RESC_{team}$ [85] is a symbolic multi-agent plan recognition scheme which represents only one coherent hypothesis. $RESC_{team}$ can reason about the assignment of agents to sub-teams, meaning that it does not require a static team hierarchy as YOYO. However, it still uses information about the plans expected to be agreed-upon.

Intille and Bobick [44] rely on coordination constraints among football players to recognize team-tactics. They thus similarly focus on the interactions between agents, rather than each agent as an individual. However, they do not address dynamic grouping and ungrouping: The recognized interactions are used to recognize the multi-agent plans that contain them.

Hongeng and Nevatia [40] recognizes multi agent events observed by a static camera. Multi-agent event is represented by a number of action threads, where each thread executed by a single actor. These action threads are related by temporal constraints generating a multi-agent event graph. They thus similarly track dynamic interactions between agents, but they are restricted to specific constraints between agents that are defined in the network. Moreover, they can not detect dynamic splitting and merging of groups.

STABR [82] is a Team Assignment and Behavior Recognition model , recovering agent-to-team assignments where the mapping of agents into teams, changes over time. This work thus also addresses the problem of behavior recognition for teams with dynamic team composition. However, this ap-

proach is based on matching agent positions to pre-specified geometric formation templates. In contrast, in our work we do not have static information about groups, but detect dynamic splitting and merging of groups.

2.2 Detecting Anomalous and Suspicious Behavior

Applications to surveillance began to appear shortly after the introduction of the basic method (e.g., [38]), and continue to this day (e.g., [18, 25, 27, 32, 55, 56, 66, 91]). There are two approaches: Detection based on anomalous behavior; and detection based on suspicious behavior. Our work presents efficient algorithms for both of these approaches.

The approach based on suspicious behavior is the most popular one. It operates under the assumption that a plan library is available that covers the intended *negative behavior*, and thus recognition of hypotheses implying such behavior is treated no differently from recognition of hypotheses implying normal activity. Examples include models that encode possible attacks in an intrusion detection system, trying to predict whether an attack is performed [32]; models that focus on recognizing specific suspicious behaviors in train stations [25], such as fighting; models with an a priori set of attack templates [45] that are compared against observations to infer whether a particular terrorist attack matches one or more templates; and models with representation of human activities based on tracked trajectories e.g., [66].

All the approaches in this group focus on probabilistic ranking of hypotheses utilizing various probabilistic plan recognition algorithms as described in Section 2.1.2. Our work also incorporates the observer biases to the suspicious behavior approach, allowing reduction in false alarms and more task-oriented ranking of hypotheses.

The second approach is based on anomalous behavior; this approach has been taken by several researchers in plan recognition. Here, the plan library is used in an inverse fashion. The plan library is limited to covering only *positive behavior*. When a plan-recognizer is unable to match observations against the library (or generates hypotheses with very low likelihood), an anomaly is announced [27, 55].

Several approaches have been proposed to tackle the abnormality detection problem. The literature on anomaly detection (e.g., for security, user

identification, etc.) is vast. Much of it is only superficially related, in the sense that the overall goals may be the same, but the application domains and the methods appropriate for them are very different. For instance, fingerprinting users from their behavior (e.g., [54]) is a related task, but the focus is on the machine learning technique (of characteristic users behavior, in a supervised manner). Similarly, we will not address here related work on anomaly detection in data signals (e.g., [2, 26]). We focus instead on the process of matching observations to intentions, encoded in a given plan library. We thus limit ourselves to related research within plan recognition.

Xiang and Gong [93] adopted dynamic Bayesian network to model each type of normal video patterns. An activity is identified as abnormal if the likelihood of being generated by normal models is less then a threshold. Duong et al. [27] use Switching Semi-Markov Models to represent user activities and perform abnormally detection in the context of durations and not in state. Yin et al. [95] present a two phase approach to detect abnormal behavior based on wireless sensors attached to the human body. This approach first uses Support Vector Machine (SVM) which filters out the activities with a high probability of being normal, then derives from this model abnormal activity model. Here also the model is learned on the normal activities.

However, these previous approaches leave several open challenges. First, a number of key capabilities implied by the application are often ignored in existing work: Recognition based on duration of behaviors, recognition despite intermittently lost observations, and recognition of interleaved goals. These capabilities are required both for recognition of anomalous and suspicious behavior model. We address these in this dissertation.

Part I

An Efficient Hybrid Model for Plan Recognition

In this part we present a set of efficient algorithms that tackle challenges in plan recognition. Plan recognition is the process of inferring other agents' plans and goals based on their observable actions. This process often takes the form of matching observations of an agent's actions to a plan library, a model of possible plans selected by the agent.

First, we show extensions to the basic symbolic plan recognition model (Chapter 3), dealing with several open challenges: (a) reducing space complexity of matching complex multi-featured observations to the plan library; (b) dealing with plan execution duration constraints; (c) handling lossy observations (where an observation is intermittently lost); and (d) handling interleaved plans (where an agent interrupts a plan for another, only to return to the first later). These algorithms are *symbolic* in that they provide hypotheses with no ranking (probabilistic or otherwise). This symbolic model is highly efficient and will be used later as a basis for a hybrid symbolic-probabilistic recognizer, and for the multi-agent framework.

We then turn to present novel efficient algorithms that allow the observer to incorporate her own biases and preferences—in the form of a utility function—into the plan recognition process (Chapter 4). This allows choosing recognition hypotheses based on their expected utility to the observer. We call this Utility-based Plan Recognition (UPR). We present a general model of UPR, extending Charniak and Goldman's [23] Bayesian Network model using Influence Diagrams. But, since reasoning about such expected utilities is intractable in the general case, we present a hybrid symbolic/decision-theoretic plan recognizer, whose runtime complexity in the worst case is $O(NDT)$, where N is the plan library size, D is the depth of the library and T is the number of observations.

Finally, we present initial steps towards efficient dynamic tracking of the organization of multi-agent teams (Chapter 5). This is done by using a combination of single-agent symbolic plan recognizer, and the DHGM data-structure. We maintain book-keeping information which allows us to dynamically track and hypothesize as to the organizational structure of a group of agents. To illustrate the use of DHGM, we present an application for detecting suspicious behavior in multi-agent framework (Section 5.3). This model is capable of catching suspicious behaviors that can be captured only when tracking agents with respect to their group and not only as individuals.

Chapter 3

SBR: Efficient Symbolic Plan Recognition

It is important for agents to model other agents' unobserved plans and goals, based on their observable actions. This process of modeling others based on observations is known as plan-recognition. Plan recognition has been studied for many years. It often takes the form of matching observations of an agent's actions to a plan-library, a model of possible plans selected by the agent. However, there are several open key challenges in modern plan recognition: (a) reducing space complexity of matching complex multi-featured observations to the plan library; (b) dealing with plan execution duration constraints; (c) handling lossy observations (where an observation is intermittently lost); and (d) handling interleaved plans (where an agent interrupts a plan for another, only to return to the first later).

In this chapter, we present efficient algorithms that address these challenges, in the context of symbolic plan recognition. The recognition algorithms we develop follow in the footsteps of [4, 5] in their focus on completeness and efficiency. They rely on lazy commitment to hypotheses, to avoid computation of hypotheses with every step (as other algorithms do, e.g., [33]). Instead, they use linear-time bookkeeping with every observation, which allows extraction of hypotheses only as needed. As commonly done in plan recognition work (e.g., [18]), we utilize a hierarchical representation of the plan library (Section 3.1.1).

Specifically, in Section 3.1 we describe the basics of SBR, giving definitions to plan library. In Section 3.2 we show how to reduce the space complexity of matching complex multi-featured observations to the plan library. In

Section 3.3 we present efficient algorithms for dealing with plan execution duration constraints, handling interleaved plans (where an agent interrupts a plan for another, only to return to the first later), and handling lossy observations (where an observation is intermittently lost). In Section 3.4 we summarize the improvements to the basic symbolic plan recognition model and their computational complexity results.

3.1 Fast Symbolic Plan Recognition: The Basics

This section describes the basics of SBR, a highly-efficient symbolic plan recognizer, introduced in [4, 5] (the reader is referred there for details). This brief description is presented for completeness, so as to put the contributions of this dissertation (Sections 3.2–3.3) in the right context.

In Section 3.1.1 we provide basic definitions of the plan library, a model of possible plans selected by the agent. In Section 3.1.2 we describe the efficient matching of multi-featured observations to the plan library, by using a specialized data-structure, a matching decision-tree called FDT (Feature Decision Tree). Section 3.1.3 presents efficient algorithms for answering the current state query, i.e., hypotheses as to the internal state of the observed agent when the observations were made.

3.1.1 The Plan Library

The plan-library is a model of possible plans selected by the agent. As commonly done in plan recognition work (e.g., [18]), we utilize a hierarchical representation of the plan library described bellow.

We represent the plan library as a single-root directed acyclic connected graph $G = \langle P, D, S \rangle$, where the set P of vertices represents *plan steps*, and edges can be of two types: Decomposition edges (D) that decompose plan steps into sub-steps, and sequential edges (S) that specify the expected temporal order of execution.

For the discussion, we refer to the children of the root node as *top-level plans*, and to all other nodes simply as *plans*. However, we use the term plan here in its general sense, inclusive of reaction plans [30], behaviors [60, 65], and recipes [63]. For our purposes, a plan step is defined as follows:

Definition 1. *Plan step.* A plan step P is set of actions that maintain or achieve a goal. A plan-step may be atomic, or non-atomic. If it is non-atomic, it can be broken down into sub-steps, each a plan-step in itself.

A totally-ordered sequence of plan sub-steps defines a plan-sequence, which the agent is supposed to execute in service of the parent plan.

Definition 2. *Plan sequence.* A totally-ordered sequence of plan steps that are linked via sequential edges, describing execution order of a given plan step by its sub-steps.

To represent plan steps that have duration (i.e., plan steps that execute over a number of time-steps), plan steps may have a sequential self-cycle, to represent their continuation over multiple time-steps.

However, no cycles are allowed hierarchically. At any given time, the observed agent is assumed to be executing a *plan decomposition path*, root-to-leaf through decomposition edges.

Definition 3. *Plan decomposition.* A set of plan steps that are linked via decomposition edges, elaborating (expanding) a given plan step.

Each plan has an associated set of conditions on observable features of the agent and its actions. When these conditions hold, the observations are said to match the plan. For example, a kick-ball-to-goal plan (of a robotic soccer player) may have the following observation features (conditions): The ball must be visible, the distance to the ball is within a given range, and the opponent goal is visible within shooting distance. If all the above conditions are satisfied, the plan matches the observation.

Definition 4. *Observation.* A tuple of observed feature values $\langle f_1, f_2, ..., f_n \rangle$, at in a given time stamp.

Definition 5. *Observation sequence.* A sequence of consecutive observations $\langle o_1, o_2, ..., o_t \rangle$ ordered by time-stamps.

Definition 6. *Matching.* Mapping between an observation to subset of plan steps. i.e., from the space of feature value vectors V^F to the powerset of plan steps.

Figure 3.1 shows an example portion of a plan library, inspired by the behavior hierarchies of RoboCup soccer teams (e.g. [50]). The figure shows

decomposition edges (solid arrows) and sequential edges (dashed arrows). The top level plans are *defend*, *attack*, and *score*. The figure does not show the observation conditions associated with plan steps. For presentation clarity, we show the decomposition edges only to the first (in temporal order) child plans. Thus in the figure, the path $root \rightarrow defend \rightarrow turn \rightarrow with\ ball$ can be an hypothesis as to the current plan of an observed player. In realistic settings, likely more than one path will match an observation tuple, and this may result in a set of such decomposition paths, i.e., a set of hypotheses as to the current state of the observed agent (answering a current state query).

An observed agent is assumed to change its internal state in two ways. First, it may follow a sequential edge to the next plan step. Second, it may reactively interrupt plan execution at any time, and select a new (first) plan. For instance, suppose the agent was executing $root \rightarrow defend \rightarrow turn \rightarrow with\ ball$, and then interrupted execution of this plan. It may now choose $root \rightarrow attack \rightarrow pass$, but not $root \rightarrow attack \rightarrow turn \rightarrow withball$.

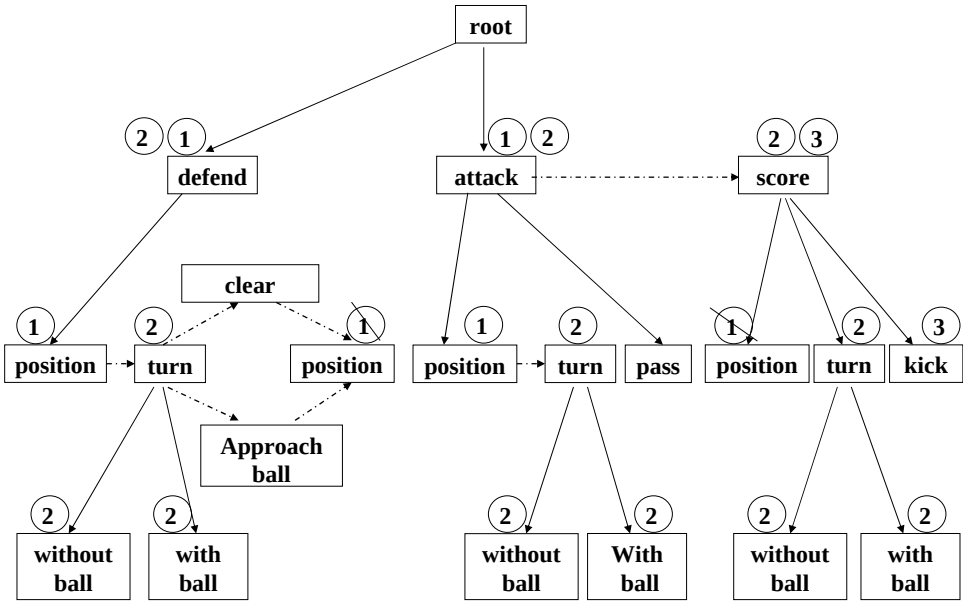

Figure 3.1: Example plan library. Circled numbers denote timestamps (Section 3.1.3).

3.1.2 Efficient Matching

The first phase of recognition, common to all recognition approaches, matches the observations made by the recognizer to plans in the plan library. The SBR consider complex observations, that consist of a tuple of observed features, including states of the world that pertain to the agent (e.g., a soccer player's uniform number), actions taken (e.g., *kick*), and execution conditions maintained (e.g., *speed* = 200). Matching such observations to plans can be expensive, if we go over all plans and for each plan check all observed features (e.g., [50]).

The SBR proposes a highly efficient matching step, augmenting the plan library with a *Feature Decision Tree* (FDT), which efficiently maps observations to matching nodes in the plan library. In this way, features are tested only once, no need to go over all the plan library and test all features. An FDT is a decision tree, where nodes correspond to features, and branches to conditions on their values. Determining the plans that match a set of observation features is efficiently achieved by traversing the FDT top-down, taking branches that correspond to the observed values of features, until a leaf node is reached. Each leaf node points at the plans that match the conjunctive set of observations along the top-down path. Ideally, each leaf nodes points to only one plan, though this may not be possible due to inherent ambiguity in the plan library.

Figure 3.2 shows the connection between the FDT and the plan library. It shows a portion of an FDT using features associated with plan-steps in Figure 3.1. Each plan-step executed by the agent, can be identified according to observed features, such as: Distance from other players, *have_ball*, opponent goal visibility, uniform number of agent. The FDT separates the plan-steps according to the values of these features. To determine matching plan-steps, the matching algorithm first checks the *have_ball* feature. Based on its value, it continues the appropriate branch to test in sequence other features, until it finally reaches a leaf node. This leaf node will have pointers to all instances of the plan-steps associated with it in the plan library. For instance the leaf-node for *position* will have four separate pointers into the plan library in Figure 3.1. Note that since the plan-step *turn* is applicable regardless of whether *have_ball* is true or false, a node associated with it will appear in both left and right subtrees of the *have_ball* root node.

To generate the FDT, we translate the plan library to set of instances, called training set. Each plan-step in the plan library will represent one

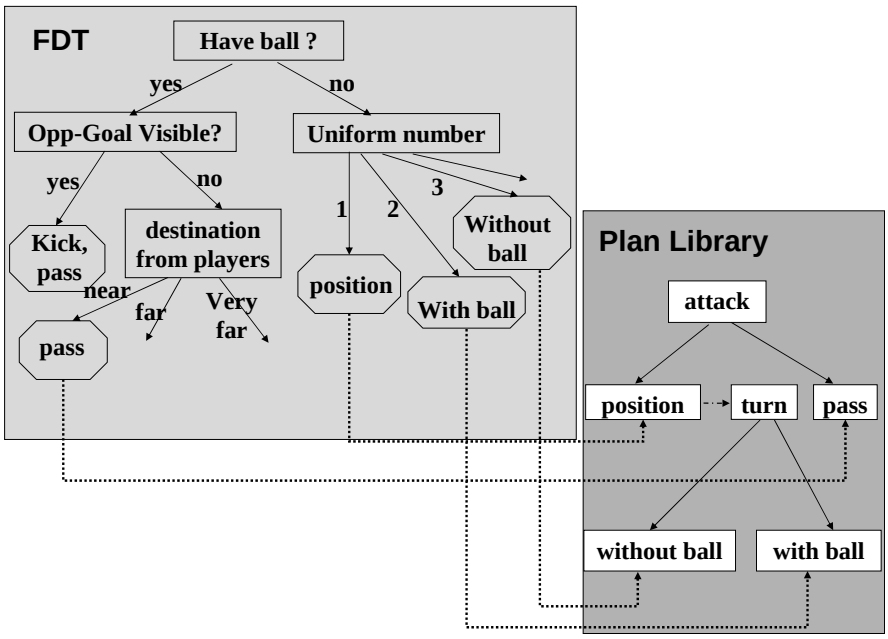

Figure 3.2: Example FDT with plan library.

instance in the training set. An instance is a fixed set of values of features (e.g, velocity) and the class of the instance, in our case the class is the plan-step itself. Note that each plan-step will appear just once in the training set, since the same plan-step has the same features. In case that a plan-step do not test a feature we put a asterisk to denote missing values (all values) for that feature, otherwise we put the value of the feature. For example, let us consider three plan-steps: $B1, B2, B3$ and three boolean features: $a1, a2, a3$. Suppose the following conditions on the plan-steps: $B1$ is possible if $a1 \wedge a3$, $B2$ if $\neg a2$, and $B3$ if $a1 \wedge a2 \wedge a3$. The resulting training set is shown in table 3.1.

classes \ features	a1	a2	a3
B1	T	$*$	T
B2	$*$	F	$*$
B3	T	T	T

Table 3.1: Example training set, generated from plan-steps in a plan library.

After generating the training set, the construction of the FDT is done similar to that of a decision tree with missing values [78]. Similarly to a decision tree, the construction of the FDT can use information gain to determine the most important features to test first, thus hopefully testing fewer features. We briefly review this well-known process here. The reader is referred to [61, 78] for details.

The FDT construction algorithm is presented below (Algorithm 1). First, we check if the instances can not be divided, meaning that a node points at only a single plan, or there are no more features that can differentiate between the plans associated with the instances. In this case we create a leaf (Lines 1–2). Otherwise, we create a node, and associates it with the feature that provides the greatest information gain (Lines 3–4) (intuitively, that divides plan-steps that test it as uniformly as possible). We then create children FDT nodes for each of its values (Lines 5–9), and recursively repeat the process of selecting a feature that best divides the plans associated with the node.

To create the children FDT nodes, we follow the procedure for handling missing values in decision trees (See [78]). Briefly, the algorithm (in each step), divides the instances according to the tested features. Each instance gets a weight, that is initialized to one at the beginning of the algorithm. This weight represents the fraction of the instances having this value for

the feature, and is used for the purpose of computing the information gain. When there is *'*'* value for the tested feature in the instance, we divide this weight between all branches, therefore dividing its weight in the number of possible values that the feature can take. When having value for the feature, the weight of the instance remains the same.

The children are constructed as follows. For each possible value of the selected feature (line 5), we select all instances that correspond to this value or have *'*'* value (line 6). For each selected instance, we update its weight in the following manner: if there is a *'*'* value, then we divide its weight in the number of the values of this feature, otherwise the weight remains the same (line 7). We also update the $NotTested$ set of features by removing the new tested feature (line 8). Then we recursively repeat on this process of selecting a feature that best divides the plans associated with the node with the new instances, new weights and the new not tested features, and dividing accordingly.

Algorithm 1 FormTree($Instances, Weights, NotTested$)

1: **if** (NotTested=$\emptyset \vee$ Instances=1) **then**
2: return $CreateLeaf(Instances)$
3: $BestFeature \leftarrow argmax_{f \in NotTested} Gain(f)$
4: $CreateNode(BestFeature)$
5: **for all** possible values v of $BestFeature$ **do**
6: $newInstances \leftarrow$ instances with value v or $*$
7: $NewWeights \leftarrow CalculateWeights(NewInstances)$
8: $NotTested \leftarrow NotTested - \{BestFeature\}$
9: $FormTree(NewInstances, NewWeights, NotTested)$

3.1.3 Current-State Hypothesis

An important query in reactive plan recognition is with respect to the current plan step selected by the observed agent. In most hierarchical plan-libraries—as in ours—this query translates to determining the decomposition paths (root-to-leaves) that are consistent with the observations, and potentially are being executed by the observed agent. Each such path is a *current-state hypothesis*.

To answer such queries, the SBR recognizer operates as follow: First, it matches observations to specific plan steps in the library, e.g., using the

FDT (described earlier). Then, after matching plan steps are found, they are tagged by the time-stamp of the observation. These tags are then propagated up the plan library (see below for explanation of the *propagateUp* algorithm), so that complete plan-paths (root to leaf) are tagged to indicate they constitute hypotheses as to the internal state of the observed agent when the observations were made.

The inference process is carried out by the *CSQ* (Current State Query) algorithm. The CSQ algorithm tags matching plan steps with time-stamps, and tries to propagate these tags up along the plan library, so that complete paths (root to leaf) are tagged. If the propagation fails due to temporal constraints (see below), it deletes all tags it has generated in climbing up the graph.

The propagation up process done by the *propagateUp* algorithm (Algorithm 2), which tags paths in the plan library as consistent with the current observation. To do this, it must propagates these tags up along decomposition edges. However, the propagation process is not a simple matter of climbing from child to parent. A plan may match the current observation, yet be *temporally inconsistent*, when a history of observations is considered.

Figure 3.1 shows the process in action (the circled numbers in the figure denote the time-stamps). Assume that the matching algorithm matches at time $t = 1$ the multiple instances of the *position* plan. At time $t = 1$, Propagate begins with the four *position* instances. It immediately fails to tag the instance that follows *clear* and *approachball*, since these were not tagged at $t = 0$. The *position* instance under *score* is initially tagged, but in propagating the tag up, the parent *score* fails, because it follows *attack*, and *attack* is not tagged $t = 0$. Therefore, all tags $t = 1$ will be removed from *score* and its child *position*. The two remaining instances successfully tag up and down, and result in possible hypotheses $root \rightarrow defend \rightarrow position$ and $root \rightarrow attack \rightarrow position$.

To disqualify hypotheses that are inconsistent (e.g., given a history of observations), *CSQ* calls the algorithm *IsConsistent* (Algorithm 3) to make the decision of whether a proposed time-stamp should be applied to a given plan step, given information in the model. The *IsConsistent* (Algorithm 3) operates as follows: Line 2 checks whether time stamp t is temporally consistent, i.e., if one of three cases holds: (a) the node in question was tagged at time $t - 1$ (i.e., it is continuing in a self-cycle); or (b) the node follows a sequential edge from a plan that was successfully tagged at time $t - 1$; or (c) the node is a first child (there is no sequential edge leading

Algorithm 2 PropagateUp(Node v, Plan Library g, Time-stamp t)

```
1: Tagged ← ∅
2: PropagateUpSuccess ← true
3: v ← w
4: while v ≠ root(g) ∧ PropagateUpSuccess ∧ ¬tagged(v, t) do
5:    if IsConsistent(v, g, t) then
6:       Tagged ← tagged ∪ {v}
7:       v ← parent(v)
8:       PropagateUpSuccess ← true
9:    else
10:      PropagateUpSuccess ← false
11: if ¬propagateUpSuccess then
12:    for all a ∈ Tagged do
13:       delete_tag(a, t)
```

into it). A first child may be selected at any time (e.g., if another plan was interrupted)[1].

If neither of these cases is applicable, then the node is not part of a temporally-consistent hypothesis, and its tag should be deleted, along with all tags that it has generated in climbing up the graph. This final deletion of all failing tags takes place in the CSQ Algorithm.

The CSQ is meant to be called with every new observation. The tags made on the plan-library are used to save information from one run to the next. Note that it assumes that the calls to it have been made in order of increasing depth, from the parent to the children, to avoid the case that the children are tagged, but their parent is not.

Algorithm 3 IsConsistent(Node v, Plan Library g, Time-stamp t)

```
1: if tagged(parent(v), t) ∨ features(parent(v)) = ∅ then
2:    if tagged(v, t − 1) ∨ ∃PreviousSeqEdgeTaggedWith(v, t − 1) ∨
      NoSeqEdges(v) then
3:       return true
4: return false
```

[1] In Sections 3.3.1 and 3.3.2, we modify these consistency checks and extend them, so that they take additional constraints into account.

3.2 Reducing Matching Space Complexity: Compact FDT

The FDT presented in Section 3.1.2 efficiently maps the observation to the plan library. Clearly, the use of the FDT leads to very significant improvements in the matching time compared to previous work. However, its space complexity is exponential in the number of features: $O(V^F)$, where V is the maximum number of values for the features and F is the number of features. This section will introduce compact-FDT, which reduces the space complexity of the FDT to linear in the plan library size.

To analyze the worst case space complexity of the FDT, we differentiate between two cases: (a) Each behavior in the plan library contains conditions on all observed features; (b) Not all observed features are tested, therefore the training set that construct the FDT has many *'*'* values (wildcard which match all values). In case (a), the worst-case space complexity is bounded in the number of behaviors $O(L)$, i.e,. it is linear in the plan library size; each behavior will appear only once in the leaves, since each feature splits the behaviors into different branches according to their values.

However, in case (b) when we have *'*'* values, we usually get to the worst-case exponential space complexity of $O(V^F)$. The reason for this is that we split each behavior to appear in more than one branch. Instead, the behavior will appear in all branches. For example, assume that we have a plan-step $p1$ that has feature $f1$ with the value *'*'*, meaning that for $p1$ to be true, the observed value of $f1$ will not matter. In the FDT the node that tests $f1$, will have number of branches which plan-step $p1$ will appear in all of them, since $p1$ matches all values of feature $f1$.

To demonstrate the problem let us consider three plan-steps: $b1, b2, b3$ and three boolean features: $a1, a2, a3$. Suppose the following conditions on the plan-steps: $b1$ is possible if $a1$ is true, $b2$ if $a2$ is true, and $b3$ if $a3$ is true. The training set shown in table 3.1. The corresponding FDT is shown in figure 3.3. Solid arrows denote true values and dashed arrows denote false values. Here we got the maximum space $2^3 = 8$. Each behavior can appear in more than one leaf. For example, the behavior $b1$ appears in 4 leaves.

To overcome this problem, we suggest to use a new data structure, called *Compact-FDT*, that represents the mapping of observations to plans in a different manner. The idea is to add a new branch marked *'*'* for a feature x, if this feature cannot distinguish between classes. Classes that appears

classes \ features	x	y	z
b1	T	$*$	$*$
b2	$*$	T	$*$
b3	$*$	$*$	T

Table 3.2: Example training set, generated from plan-steps

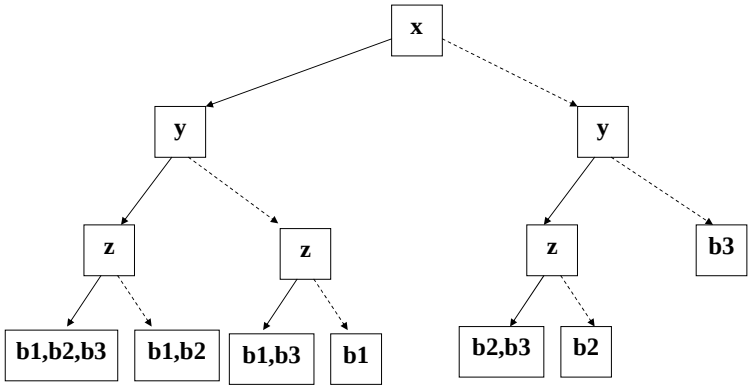

Figure 3.3: FDT with exponential size. Solid arrows denote true values and dashed arrows denote false values

both in the true branch and the false branch are put together in the *'*'* branch. Figure 3.4 shows the Compact-FDT corresponding to the previous example.

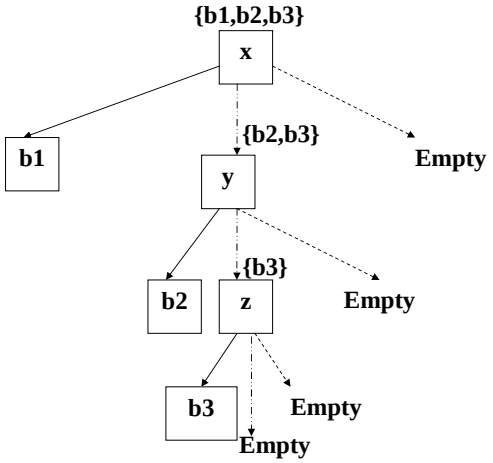

Figure 3.4: Compact FDT for the same plan library as in figure 3.3. Solid arrows denote true values, dashed arrows denote false values, and middle arrows denote *'*'* branch

The construction of the Compact FDT is done as in the FDT (Algorithm 1), but with a number of differences (Algorithm 4). To create an *'*'* value branch (Lines 9–11), we select only the plan steps (instances) with *'*'* value for the tested feature. Note that in line 13 we take only those instances with value v and not as in FDT, that we took also *'*'* value. The construction of the *'*'* branch is different from the construction of $MissValues$ (Lines 6–8), where all instances are copied from the parent, no matter what are their values.

When traversing the Compact FDT, *in addition* to the branch we would have gone in FDT, we will also go to the branch marked with (*'*'*). The result will be the union of all behaviors in the leaves both from the *'*'* branches and from the traversal of the tree according to the observed values of each

feature.

Algorithm 4 FormCompactTree($Instances, Weights, NotTested$)

```
1: if (NotTested=∅ ∨ Instances=1) then
2:    return CreateLeaf(Instances)
3: BestFeature ← argmax_{f∈NotTested} Gain(f)
4: CreateNode(BestFeature)
5: for all possible values v of BestFeature do
6:    if v = missing value then
7:       NewInstances ← Instances
8:       NewWeights ← Weights
9:    else if v = * then
10:      NewInstances ← Instances with value *
11:      NewWeights ← Weights of NewInstances
12:   else
13:      NewInstances ← all instances with value v
14:      NewWeights ← CalculateWeights(newInstances)
15:   NewNotTested ← NotTested − {BestFeature}
16:   FormCompactTree(NewInstances, NewWeights, NewNotTested)
```

The Compact FDT Matching algorithm (Algorithm 5) is as the matching algorithm of the FDT (see [5] for more details), except for Lines 6–7. The addition is in Lines 6–7, where in addition to traversing the FDT according to the values of the observed features we are also following the *'*'* branches until we get to the matching leaves. Then, after getting to the appropriate leaves in the Compact FDT, we return pointers to the relevant plan-steps in the plan library (Line 2). Note that as opposed to FDT, where we reach only one leaf in the end of the process, here we unify the results of reaching more than one leaf plan step.

Complexity Analysis We define k as the number of values of each feature plus one for *'*'* value. Now, we differentiate between three cases:

1. $k = 2$, meaning each feature has 2 values: true or false, plus *'*'* value.
2. $k > 2$, and there exists a single value for each feature; each behavior has one value for its feature (except *'*'* values).
3. $k > 2$, and there exist multiple values for each feature. For example, in behavior $b1$ the value of feature $f1$ is $v1$ or $v2$. For behavior $b2$

Algorithm 5 Matching(Feature Tree Node v, Observed Features List F, Plan Library L, Feature Tree fdt)

```
1: if v is a leaf then
2:     return all plan-steps in L that match plan-step in v
3: else
4:     i ← FeatureIndex(fdt, v)
5:     v1 ← child(fdt, v, F_i)
6:     v2 ← child(fdt, v, *)
7:     return MatchCompact(v1, F, L, fdt) ⋃ MatchCompact(v2, F, L, fdt)
```

the values are $v2$ or $v3$. Note that in this case, it might be an overlap between values.

In the first two cases, the worst case space complexity is $O(L)$, where L is the plan library size. The CompactFDT is a full $(k+1)$-ary tree; each node has $k+1$ or zero branching children. Each feature will divide the behaviors associated to it to $k+1$ branches. Some of these can be empty branches, but at least 2 branches have values, otherwise there is no point to use this feature since it does not distinguish between different behaviors. Each behavior will not appear in more than one branch: It will appear either in its value branch (there is no overlapping between branches values) or the '*' branch. The total number of all behaviors in the leaves will be $O(L)$, the number of behaviors in the plan library. The worst case space complexity of a $full$ $(k+1)$-ary tree is when it is *complete* (i.e., all leaves are in the same level all internal nodes have exactly $k+1$ branches). It is known that the number of leaves in a complete $k+1$-ary tree is equal to $(k+1)^h$, where h is the height of the tree. The number of internal nodes in the full $k+1$-ary tree is $\frac{(k+1)^h-1}{(k+1)-1}$. Recall that we said that the number of leaves are $L=(k+1)^h$. Therefore the space complexity of the Compact FDT is $O(L+\frac{L-1}{(k+1)-1}) = O(L)$.

In the third case, where each behavior has multiple values, the worst case space complexity is still exponential: $O(V^F)$. This will happen when for each feature we can not distinct between the different behavior, since the values overlap. We believe that this is a rare case in practical settings. In principle, it can be handled by different branching, but this lies outside the scope of this work.

The run-time complexity remain linear in the plan library size $O(L)$, in all cases. In the worst case we will go over the whole compact-FDT, where

selecting in each step the branch that matches the value of the observed feature and the *'*'* branch, visiting only once in each node. This will be done by traversing the compact-FDT top down recursively choosing the *'*'* branches and the branch according to the observed feature till getting to the leaves. We will not visit the same node more than once.

3.3 Accounting for Complex Temporal Behaviors

The basic model described in Section 3.1.3 may be used to recognize plan(s) that are ordered in time. It also allows for plan steps to have duration, by modelling self-cycles. However, it cannot recognize more complex forms of temporal execution of plans, such as maintaining the selection of a specific plan-step within some bounded interval, or interrupting a sequence of plan steps under one node, to execute another, only to return to it to the first sequence later (plan interleaving).

This section shows how the CSQ mechanisms introduced in Section 3.1.3 can be extended to account for these temporal plans. Section 3.3.1 extends the *propagateUp* algorithm (Algorithm 2) to take duration bounds (minimum and maximum time spent in a plan-step) into account. Section 3.3.2 independently explores modifications to *propagateUp* for interleaving. Section 3.3.3 independently explores modifications to *propagateUp* for handling missing observations (where all features are unobservable).

3.3.1 Managing Durations

Instances of the same plan step can vary in the duration of their execution. For example, depending on the distance to the ball, a soccer player may take a long time or short time to execute the *approach ball* plan in Figure 3.1. As a result, we may have multiple observation time-stamps $(t, t+1, \ldots t+k)$ that are all consistent with a single plan, and only reflect the duration that its execution requires between one and $k+1$ time-stamps.

However, often some bounds are known on execution duration. For instance, in an airport terminal, there exists a difference in the plans of a passenger who stands at the check-in area for a few minutes, and a security guard who stands there for a few hours. Or, in a different example, a bas-

ketball player is only allowed inside the basket zone for a limited amount of time.

We thus want to take into account constraints on the duration of plan-steps. We can allow such constrains in the approach we presented. We extend the tagging mechanism to allow two types of tags: (a) *Hard tags*, which signify that a plan step is (or is not) consistent with respect to previous plan-steps (the $IsConsistent$ algorithm, Algorithm 3), and also consistent with duration constraints (minimum, maximum time); and (b) *Soft tags*, which signify that a plan-step is (or is not) consistent with observations (and with prior plan-steps), but is not within its duration constraints (e.g., this plan-step has not been selected for sufficient amount of time, but may be in few time-stamps).

We make the following additions to the $propagateUp$ algorithm, and present the revised algorithm (Algorithm 6): (1) Line 4 checks if v is not tagged with *hard tag* (instead of checking if tagged); (2) Line 5: checks if the maximum duration holds, using $CalcDuration$ algorithm (Algorithm 7); (3) Line 6: in the $IsConsistent$ algorithm, all the occurrences of $tagged(v,t)$ should be replaced with $tagged(v,t,Soft)$, and the $ExistsPreviousSeqEdgeTaggedWith(v,t-1)$ should be replaced with $ExistsPreviousSeqEdgeTaggedWith(v,t-1,Hard)$; (4) Lines 7–12 if the minimum duration constrain holds, then the plan will be tagged in time-stamp t with $Hard$ tag, otherwise with $Soft$ tag.

The algorithm *calcDuration* (Algorithm 7), calculates the duration in which the plan was active. To calculate this, the algorithm goes over the time-stamps of the plan from time stamp t, and counts the number of consecutive tags (time-stamps with no temporal gaps). For example, if plan is tagged with time-stamps 1,4,5 and 6 then the duration will be 3.

3.3.2 Interleaved Plans

Many plan recognition algorithms cannot cope with modelling an agent that is pursuing multiple plans (i.e., for multiple goals), by interleaving plan steps. Here, the agent may begin with one plan, and interrupt its execution to execute another, only to return to the remaining plan steps in the first plan.

To handle interleaving, we add a $memoryFlag$ in each of the first children. This flag will hold the latest time-stamp tag, in the sequential link chain from this child. We will use this flag to disqualify plans that are in the middle of the chain and are not the ones that we paused at. For example, in Figure 3.1,

Algorithm 6 PropagateUp(Node v, Plan Library g, Time-stamp t)

```
1: Tagged ← ∅
2: PropagateUpSuccess ← true
3: v ← w
4: while v ≠ root(g) ∧ PropagateUpSuccess ∧ ¬tagged(v, t, Hard) do
5:    if CalcDuration(v, t) < maxDuration(v) then
6:       if IsConsistent(v, g, t) then
7:          if CalcDuration(v, t) < minDuration(v) − 1 then
8:             tag(v, t, Soft)
9:          else
10:            tag(v, t, Hard)
11:         Tagged ← tagged ∪ {v}
12:         v ← parent(v)
13:         propagateUpSuccess ← true
14:      else
15:         propagateUpSuccess ← false
16:   else
17:      propagateUpSuccess ← false
18: if ¬propagateUpSuccess then
19:   for all a ∈ Tagged do
20:      delete_tag(a, t)
```

Algorithm 7 calcDuration(Node v, Time-stamp t)

```
1: counter ← 0
2: time ← t
3: while time > 0 do
4:    if tagged(v, t − 1, Soft) then
5:       Counter ←counter+1
6:       time ←time-1
7:    else
8:       break
9: return(counter)
```

the position plan under attack will hold $memoryFlag$ that contain the time-stamp 2. Suppose that there is a sequential link also from turn to pass. Then we would like to return exactly to pass and not to the turn plan step.

The approach we present above can deal with these cases by making one change to the $isConsistent$ algorithm (algorithm 3). In line 2, the condition: $\exists IncomingSeqEdgeTagged(v, t-1)$ should be replaced with two conditions: First, $\exists IncomingSeqEdgeTagged(v, t-1 or smaller))$. Meaning that we check whether there is an incoming sequential edge from a plan-step tagged with a time-stamp that is smaller or equal to $t-1$. This will allow a plan-step to be considered consistent if it continues a previously interrupted sequence of plan-steps. Second, we add the condition $\exists IncomingSeqEdgeTagged(v, MemoryFlag)$, this condition forces a return to the interrupted plan-step.

Although we add recognition capabilities, we should remember that this change can influence the number of possible hypotheses. A partial solution is to tag those plans that can be interrupted and resumed with a $jump$ label. The $isConsistent$ algorithm will then check plans based on their $jump$ settings. For plans with a $jump$ label, it will check that the previous node had time equal to $t-1$. For plans without a $jump$ label, it will check equal or smaller than $t-1$ constraint. This allows greater control over the accuracy of the model, and facilitates increased efficiency.

Another issue to discuss is the question of how many ticks (time units) can pass from the time we interrupt a plan step, until returning to it. To limit this time, we can add a flag $MaximumInterruptTime$, and add also the constraint that the difference between a new observation's time-stamp and that of a node with an incoming sequential edge, must be smaller than $MaximumInterruptTime$. This can disqualify lingering hypotheses, and make the algorithm more accurate.

3.3.3 Missing Observations

An underlying assumption in previous investigations is that every change in internal state (in our terms, change in plan path) is somehow reflected in observations. However, in realistic settings, this assumption is sometimes violated (e.g., in Overhearing applications [49]. Some internal decision-making may be permanently or intermittently unobservable, for all of the plans along a specific plan decomposition path. In this case, an entire observation is essentially missing (all features are unobservable).

We propose some small changes in the propagation algorithms to allow them to address this difficulty. The idea is to mark potentially-unobservable plans with a *lossy* label in the plan library (though the first and the last plans in a plan-step sequence cannot have a *lossy* label).

We again modify the algorithms $IsConsistent$ (Algorithm 3) and $propagateUp$ (Algorithm 2). In the propagating process, nodes that are labelled as *lossy* and are part of a sequence will be skipped (if they are not tagged), when the sequence is checked for temporal consistency. This is done by replacing the method $tagged(X, t)$ with a new method.

The $AdvanceTagged(v, t)$ algorithm (Algorithm 8) exploits the sequential edges and the *lossy* labels. Plan v will be considered as tagged at time-stamp t if one of two cases holds: (a) the plan itself is tagged with time-stamp $t-1$; (b) the plan is not tagged with time-stamp $t-1$, but it has *lossy* label and the sequential edge that points to it is tagged with time-stamp $t-1$. In case that there are chains of *lossy* labels, we check if the first plan that is not labelled with *lossy* is tagged with time-stamp $t-1$. It is important to note here that the time referred to by the time-stamp is the observation time, not world time. Thus $t-1$ is the time of the previous observation, not a single tick ago.

This feature must be used carefully, since it can significantly influence runtime, and the number of possible hypotheses. Specifically, labelling many plan steps as *lossy* will result in long backtracks through previous plan-step nodes, until we arrive at node that was not labelled with *lossy* or that was tagged with time stamp $t-1$. This can be significantly more expensive to do than just checking whether the previous node on the sequence was tagged with time stamp $t-1$. The number of output hypotheses also increases when adding many *lossy* labels, because a plan-step labelled *lossy* can be a part of many hypotheses, without being observed.

3.4 Summary: SBR Improvements

Compared to earlier work [4, 5] on the SBR algorithms, this chapter introduced a number of significant improvements to symbolic plan recognition. Figure 3.5 summarize these improvements, and their computational complexity results introduced in this chapter.

Algorithm 8 AdvanceTagged(Node v,Timestamp t, Plan Library g)

```
1: if tagged(v, t) then
2:    return true
3: else
4:    while (v labelled as lossy) ∧ (∃X, s.t. (X, v) ∈ g) do
5:       if tagged(X, t) then
6:          return true
7:       v ← X
8: return false
```

	Basic SBR	Extended SBR
Matching best-case space complexity	Exponential: *O(V^F)*, where *V* is the maximum number of values for the features and *F* is the number of features.	Linear in the plan library size *O(L)*.
Matching worst-case space complexity	*O(V^F)*	*O(V^F).* We believe that this is a rare case in practical settings (each behavior has multiple values for each feature).
Matching run-time complexity	Linear in the plan library size *O(F+L)*.	Linear in the plan library size *O(F+L)*.
Simple CSQ run-time complexity	Linear in the plan library size *O(L)*.	Linear in the plan library size *O(L)*.
CSQ run-time with durations	-	Linear in the plan library size *O(L)*.
CSQ run-time with interleaved goals	-	Linear in the plan library size *O(L)*.
CSQ run-time with lossy observations	-	Linear in the plan library size *O(L)*.

Figure 3.5: Summary of the improvements to the basic symbolic plan recognition model introduced in Chapter 3.

Chapter 4

UPR: Efficient Utility-based Plan Recognition

Essentially all previous work in plan recognition has focused on recognition accuracy itself, with no regard to the use of the information in the recognizing agent. As a result, low-likelihood recognition hypotheses that may imply significant meaning to the observer are ignored in existing work. In this work, we present novel efficient algorithms that allow the observer to incorporate her own biases and preferences—in the form of a utility function—into the plan recognition process. This allows choosing recognition hypotheses based on their expected utility to the observer. We call this Utility-based Plan Recognition (UPR). We present a general model of UPR, extending Charniak and Goldman's [23] Bayesian Network model using Influence Diagrams. But, since reasoning about such expected utilities is intractable in the general case, we present a hybrid symbolic/decision-theoretic plan recognizer, whose runtime complexity in the worst case is $O(NDT)$, where N is the plan library size, D is the depth of the library and T is the number of observations.

Specifically, in Section 4.1 we present a procedure to create a Plan Recognition Influence Diagrams. In Section 4.2 we present efficient hybrid symbolic/decision-theoretic plan recognizer.

4.1 UPR in Influence Diagrams

This section presents a procedure to create a Plan Recognition Influence Diagrams (PRID). We first introduce the seminal Bayesian Model of Plan

Recognition introduced in [23] (Section 4.1.1). Then, in Section 4.1.2 we extend this procedure to create an influence diagram from the Bayesian Network. Finally, we show the complexity of this approach in Section 4.1.3.

4.1.1 A Bayesian Model of Plan Recognition

The first probabilistic plan recognition system was described by Charniak and Goldman [23], presenting a procedure for constructing Plan Recognition Bayesian Networks. We build upon this basic model, and show how to add utilities to this model in Section 4.1.2. Note that there are other models (e.g, [43]) that also generate belief networks from plan libraries to solve plan recognition problems. We believe our approach can be applied for other types of belief networks in the same manner.

We briefly describe the Charniak and Goldman construction method; The reader is referred to [23] for a detailed description. We use an illustrative example (Figure 4.1), in which there are two competing hypotheses: Robbing and shopping (taken from [23]). This plan library is constructed based on the story: "Jack went to the liquor store. He pointed a gun at the owner". After processing the first line, the constructed Bayesian network will infer with high probability that Jack is shopping at a liquor store, however after reading the second line, it will infer that a robbery is more probable.

Figure 4.1 shows four types of nodes, introduced in [23]:

1. Hypothesis plans nodes denoted by *(inst object type)*. There are two hypotheses nodes from this type in Figure 4.1: *(inst iss3 liquor-shopping)*, meaning $iss3$ is some sub-type of the hypothesis liquor-shopping, and *(inst rob4 rob)* which mean that $rob4$ is some sub-type of the hypothesis rob.

2. Plan-step nodes which explain the observed action. For example, *(inst go1 go)* explains the going event (observed action).

3. Slot-filler nodes denoted by *(=(stp1 p) a1)*, indicating that the event $a1$ is a step in plan p. The slot fillers connect the observed action (the event) to the plan. For example, *(=(go-stp iss3) go1)* is a slot filler that connects the observed action *(inst go1 go)* to the plan *(inst iss3 liquor-shopping)*.

4. Other evidence nodes, with general evidence information. For example: the node *(=(destination go1) is2)* connects between *(=(go-stp iss3) is2)*

and *(=(store-of iss3) is2)*, to show that the destination of going is a liquor store.

In this example it is clear that the robbing hypothesis matches, since we had the observation that Jack points a gun at the owner. But, there are cases that it is unclear whether we still want the most probable hypothesis or the hypothesis that costs more to the observer.

For example, if we replace the second line in the previous example with: "Jack holds a gun in his pocket". In this case the Bayes Net still infers that this is the shopping event with high probability. The reason is that the a-priori probability of shopping is higher than robbery, and holding a gun in the pocket does not necessary means that the person is planning a robbery. However, this case can be dangerous for the observer (i.e,. the owner of the liquor-store) and she may want to take into consideration the matching hypothesis of robbing that can harm her.

This is an example where the observer may want to combine into the plan recognition model a utility function that model the risk she would take for different cases, and take into consideration also hypotheses with the high cost and not with the high probability. In Section 4.1.2 we present procedures that take into account the utility function of the observer.

4.1.2 Plan Recognition Influence Diagram

We now show how to include reasoning about the utility of hypotheses to the observer. We use a general mechanism for making decisions called influence diagrams ([41]), which extends belief networks with additional node types for actions and utilities. Influence Diagrams have three types of nodes: (a) Chance nodes (ovals), which represent random variables as in belief nets (the agent's beliefs about the world); (b) Decision nodes (rectangles), which represent points where the decision maker has a choice of actions; and (c) Utility nodes or Value nodes (diamonds), which represent the agent utility function.

We extend the procedure described in Section 4.1.1 to influence diagrams. The procedure to create a Plan Recognition Influence Diagrams (*PRID*) is as follows:

1. We add a utility node u_i for each hypothesized plan p_i.
2. We connect all slot-fillers of plan p_i to the utility node u_i via utility arcs. In this way we can have a different cost for different actions of

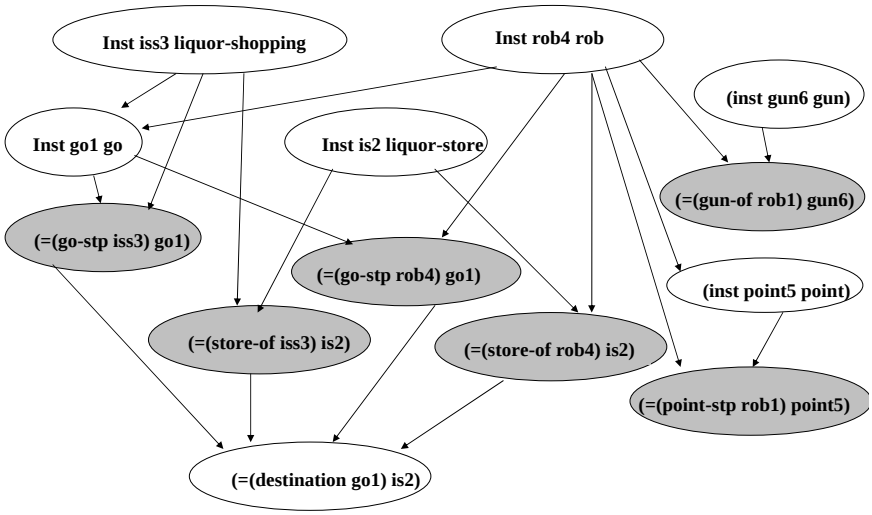

Figure 4.1: An example of Bayesian Network, Competition between shopping and Robbing (taken from [23]). Slot-fillers are in Bold.

the observed agent (note that we add an arc from the slot-filler and not from the observed action, since the slot fillers connects between the observed action and the plan).

3. We add arcs from *other evidence* node to u_i(for example we want to give different costs for different destinations – higher cost for being at a jewelry store then in a liquor-store).

Note that, alternatively, we can add arcs only from the hypothesized plan p_i to u_i, but we will not be able to give different costs for different events under the hypothesized plan p_i. For example, we will not be able to give higher cost for the event of holding a big gun, and lower cost for holding a small gun. Note also that there is no decision node, since we do not have a decision to make here. The *PRID* only represents the information, and does not decide on the actions.

Figure 4.2 shows the resulting *PRID* (plan recognition influence diagram) for the example in Section 4.1.1. We added two utilities nodes (denoted with diamonds): One for the shopping plan and one for the robbing plan. We use these here to reason about cost of the observer (high utility, means high cost for the observer). Then, from the slot fillers of these plans and from the *other evidence* node, we added arcs to the appropriate utilities nodes (denoted with dashed arrows). For presentation clarity, nodes that are not related to utility were omitted from the figure.

Evaluating the influence diagram is done quite similar to the algorithm described in [79]: (a) Set evidence variables for the current state; (b) Calculate posterior probabilities for the parent nodes of each of the utility nodes; (c) Calculate the resulting utility; (d) Return the plan with the highest utility (highest cost for the observer, in the example).

A Simple Running Example. Assume that we had an observation like: "Jack holds a gun in his pocket in the liquor-store". The observer set up the utilities information as to it subjective preferences and biases to balance between the probabilities and costs. We assume the following probabilities (the probabilities table were built based on [23]):

1. The prior probabilities of the Hypotheses plans *(inst iss3 liquor-shopping)* and *(inst rob4 rob)*, are 0.8 and 0.02, respectively.

2. The plan-step which explains the observed action probability is true given its parents are true, when both parents are false we give a uniform

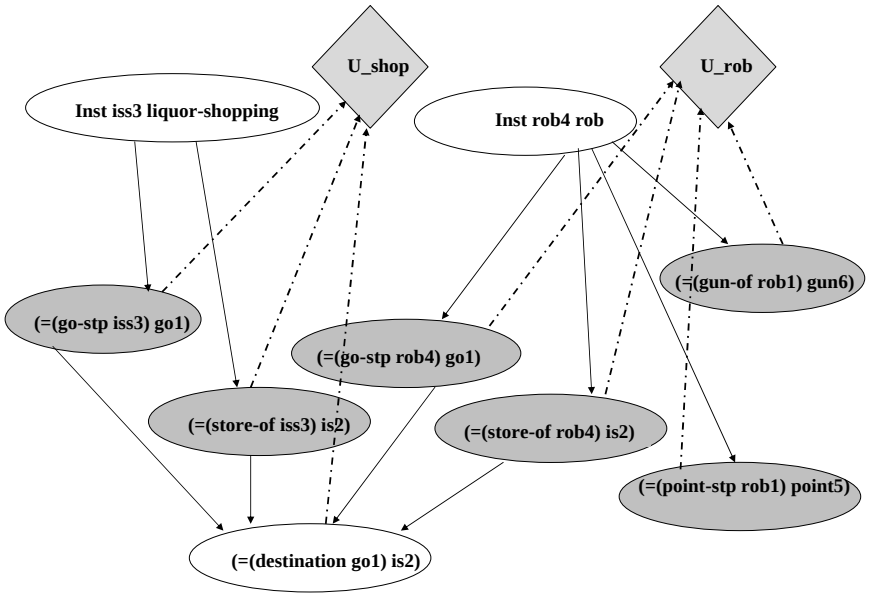

Figure 4.2: An example of Influence Diagram for the Competition between shopping and Robbing (dashed arrows denote utilities arcs)

Parents Nodes		Inst go1 go	
Inst iss3 liquor-shopping	Inst rob4 rob	True	False
True	True	1	0
	False	1	0
False	True	1	0
	False	0.5	0.5

Table 4.1: The Conditional Probability Table of *Inst go1 go*.

distribution (0.5 for true and 0.5 for false). For example, see *(inst go1 go)* conditional probabilities in Table 4.1.

3. The slot-filler is true when both its parents are true. For example, see the Conditional probabilities of *(=(go-stp iss3)go1)* in Table 4.2.

4. The *other evidence* is true when the context it explains is true. For example, (=(destination go1)is2) is true when both (=(go-stp iss3)go1) and (=(store-of iss3)is2) are true or (=(go-stp rob4)is2) and (=(store-of rob4)is2) are true.

We now add utilities (costs for the observer). Assume that we want to give high cost to someone enters the store with a gun. In this case we put on the arc from *(=(gun-of rob1) gun6)* to the utility node *U_Rob* cost of 100. Figure 4.3 shows the calculated probabilities and utilities of this example. We see that in this simple example the probability of hypothesis of robbing is 0.022, while the shopping hypothesis is about 0.88. Thus, the Bayesian Network described at [23] will infer that this is a shopping event with high probability. However, with the influence diagram we can see that there is a risk with Jack and closely track his moves to see whether he points a gun later on.

4.1.3 Complexity Analysis

We have shown here how a UPR recognizer could be implemented by extending the use of plan-recognition Bayesian Networks [23] to influence diagrams [41]. However, the run-time complexity of inference in such representations may be inhibitory for real-world cases.

Parents Nodes		(=(go-stp iss3)go1)	
Inst iss3 liquor-shopping	Inst go1 go	True	False
True	True	1	0
	False	0	1
False	True	0	1
	False	0	1

Table 4.2: The Conditional Probability Table of *(=(go-stp iss3)go1)*.

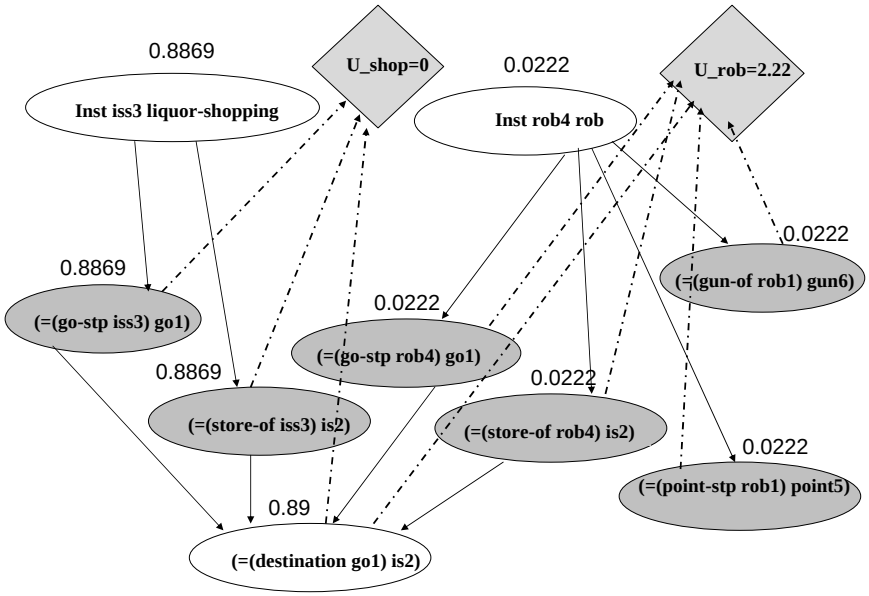

Figure 4.3: A running example of Influence Diagram for the Competition between shopping and Robbing (dashed arrows denote utilities arcs)

In general, probabilistic inference is NP-Hard [24]. Even approximate inference is NP-Hard; although scales well with the network size, is NP-Hard with respect to the hard-bound of the estimates. In Section 4.2 we will show a hybrid UPR technique which reduces the complexity to worst-case runtime complexity $O(NDT)$, where D is the depth of the plan library, N is the plan library size and T is the number of observations. However, to achieve this, the hybrid UPR sacrifices expressivity.

4.2 A Hybrid UPR Technique

This section presents an efficient hybrid UPR technique. Here, a highly efficient symbolic plan recognizer presented in Chapter 3 is used to filter through hypotheses, maintaining only those that are consistent with the observations (but not ranking the hypotheses in any way). We then add an expected utility aggregation layer, which is run on top of the symbolic recognizer (Section 4.2.1). In Section 4.2.2 we reduce the complexity. In Section 4.2.3 we discuss the expressivity sacrifices the hybrid UPR does for reducing the complexity.

4.2.1 Computing the Expected Utility of an Hypothesis

After getting all *current state hypotheses* from the symbolic recognizer, the next step is to compute the expected utility of each hypothesis. This is done by multiplying the posterior probability of a hypothesis, by its utility to the observer.

We follow in the footsteps of *Hierarchical Hidden Markov Model* (HHMM) [29] in representing probabilistic information in the plan library. We denote plan-steps in the plan library by q_i^d, where i is the plan-step index and d is its hierarchy depth, $1 \leq d \leq D$. For each plan step, there are three probabilities:

Sequential transition. For each internal state q_i^d, there is a state transition probability matrix denoted by $A^{q^d} = (a_{i,j}^{q^d})$, where $a_{i,j}^{q^d} = P(q_j^d | q_i^d)$ is the probability of making a sequential transition from the i^{th} plan-step to the j^{th} plan-step. Note that self-cycle transitions are also included in A^{q^d}.

Interruption. We denote by $a_{i,end}^{q^d}$ a transition to a special plan step end^d which signifies an interruption of the sequence of current plan step q_i^d, and

immediate return of control to its parent, q^{d-1}.

Decomposition transition. When the observed agent first selects a decomposable plan step q_i^d, it must select between its (first) children for execution. The decomposition transition probability is denoted $\Pi^{q^d} = \pi^{q^d}(q^{d+1}) = P(q_k^{d+1}|q_i^d)$, the probability that plan-step q_i^d will initially activate the plan-step q_k^{d+1}.

Observation Probabilities. Each leaf has an output emission probability vector $B^{q^d} = (b^{q^d}(o))$. This is the probability of observing o when the observed agent is in plan-step q^d. For presentation clarity, we treat observations as children of leaves, and use the decomposition transition Π^{q^d} for the leaves as B^{q^d}.

In addition to transition and interruption probabilities, we add utility information on the edges in the plan library. The utilities on the edges represent the cost or gains to the observer, given that the observed agent selects the edge. For the remainder of the work, we use the term cost to refer to a positive value associated with an edge or node. As in the probabilistic reasoning process, for each node we have three kinds of utilities: (a) E^{q^d} is the sequential transition utility (cost) to the observer, conditioned on the observed agent transitioning to the next plan-step, paralleling A^{q^d}; (b) $e_{i,end}^{q^d}$ is the interruption utility; and (c) Ψ^{q^d} is the decomposition utility to the observer, paralleling Π^{q^d}.

Figure 4.4 shows portion of the plan library of an agent walking with or without a suitcase in the airport, occasionally putting it up and picking it up again, an example discussed below. Note the *end* plan step at each level, and the transition from each plan-step to this end plan step. This edge represent the probability to interrupt. The utilities are shown in diamonds (we omitted zero utilities, for clarity). The transitions allowing an agent to leave a suitcase without picking it up are associated with large positive costs, since they signify danger to the observer.

We use these probabilities and utilities to rank the hypotheses selected by the SBR. First, we determine all paths from each hypothesized leaf in time-stamp $t-1$, to the leaf of each of the current state hypotheses in time stamp t. Then, we traverse these paths multiplying the transition probabilities on edges by the transition utilities, and accumulating the utilities along the paths. If there is more than one way to get from the leaf of the previous hypothesis to the leaf of the current hypothesis, then it should be accounted for in the accumulation. Finally, we can determine the *most costly* current

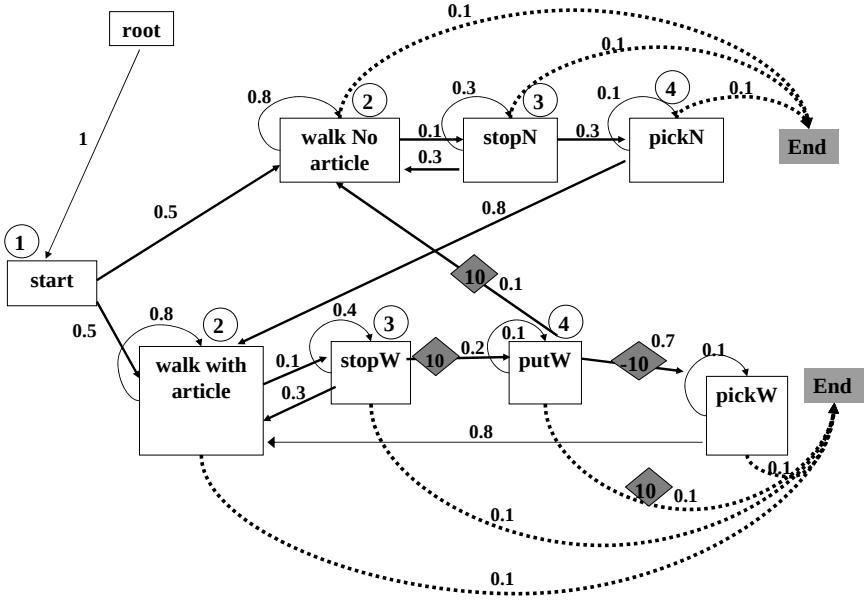

Figure 4.4: An example plan library. Recognition time-stamps (example in text) appear in circles. Costs appear in diamonds.

plan-step (the current-state hypothesis with maximum expected cost). Identically, we can also find the *most likely* current plan-step, for comparison.

Formally, let us denote hypotheses at time $t-1$ (each a path from root to leaf) as $W = \{W_1, W_2, ..., W_r\}$, and the hypotheses at time t as $X = \{X_1, X_2, ..., X_l\}$. To calculate the maximum expected-utility (most costly) hypothesis, we need to calculate for each current hypothesis X_i its expected cost to the observer, $U(X_i|O)$, where O is the sequence of observations thus far. Due to the use of SBR to filter hypotheses, we know that the $t-1$ observations in O have resulted in hypotheses W, and that observation t results in new hypotheses X. Therefore, under assumption of Markovian plan-step selection, $U(X_i|O) = U(X_i|W)$.

The most costly hypothesis is computed in Equation 4.1. We use $P(W_k)$, calculated in the previous time-stamp, and multiply it by the probability and the cost to the observer of taking this step from W_k to X_i. This is done for all i, k.

$$\hat{X}_i = \underset{X_i}{\operatorname{argmax}} \sum_{W_k \in W} P(W_k) \cdot P(X_i|W_k) \cdot U(X_i|W_k) \tag{4.1}$$

To calculate the expected utility $E(X_i|W_k) = P(X_i|W_k) \cdot U(X_i|W_k)$, let X_i be composed of plan steps $\{x_i^1, ..., x_i^m\}$ and W_k be composed of $\{w_k^1, ..., w_k^n\}$ (the upper index denotes depth). There are two ways in which the observed agent could have gone from executing the leaf $w^n \in W_k$ to executing the leaf $x^m \in X_i$: First, there may exist $w \in W_k$, $x \in X_i$ such that x and w have a common parent, and x is a direct decomposition of this common parent. Then, the expected utility is accumulated by climbing up vertices in W_k (by taking *interrupt* edges) until we hit the common parent, and then climbing down (by taking first child decomposition edges) to x^m. Or, in the second case, x^m is reached by following a sequential edge from a vertex w to a vertex x.

In both cases, the probability of climbing up from a leaf w^n at depth n, to a parent w^j (where $j < n$) is given by

$$\alpha_{w^n}^j = \prod_{d=n}^{j} a_{w,\text{end}}^d \tag{4.2}$$

and the utility is given by

$$\gamma^j_{w^n} = \sum_{d=n}^{j} e^d_{w,\text{end}}. \tag{4.3}$$

The probability of climbing down from a parent x^j to a leaf x^m is given by

$$\beta^j_{x^m} = \prod_{d=j}^{m} \pi^{x^d}(x^{d+1}) \tag{4.4}$$

and the utility is given by

$$\delta^j_{x^m} = \sum_{d=j}^{m} \psi^{x^d}(x^{d+1}). \tag{4.5}$$

Note that we omitted the plan-step index, and left only the depth index, for presentation clarity.

Using α^j_w, β^j_x, γ^j_w and δ^j_x, and summing over all possible j's, we can calculate the expected utility (Equation 4.6) for the two cases in which a move from w_n to x_m is possible .

$$\begin{aligned} E(X_i|W_k) &= P(X_i|W_k) \times U(X_i|W_k) \\ &= \sum_{j=n-1}^{1} [(\alpha^j_w \cdot \beta^j_x) \times (\gamma^j_w + \delta^j_x) \times \text{Eq}(x^j, w^j)] \\ &+ \sum_{j=n-1}^{1} [\alpha^j_w \cdot a^j_{w,x} \cdot \beta^j_x] \times (\gamma^j_w + e^j_{w,x} + \delta^j_x) \end{aligned} \tag{4.6}$$

The first term covers the first case (transition via interruption to a common parent). Let $\text{Eq}(x^j, w^j)$ return 1 if $x^j = w^j$, and 0 otherwise. The summation over j accumulates the probability multiplying the utility of all ways of interrupting a plan w^n, climbing up from w^n to the common parent $x^j = w^j$, and following decompositions down to x^m.

The second term covers the second case, where a sequential transition is taken. $a^j_{w,x}$ is the probability of taking a sequential edge from w^j to x^j, given that such an edge exists ($a^j_{w,x} > 0$), and that the observed agent is done in w_j. To calculate the expected utility, we first multiply the probability of

climbing up to a plan-step that has a sequential transition to a parent of x^m, then we multiply in the probability of taking the transition, and then we multiply in the probability of climbing down again to x^m. Then, we multiply in the utility summation along this path.

A naive algorithm for computing the expected costs of hypotheses at time t can be expensive to run. It would go over all leaves of the paths in $t-1$ and for each of these, traverse the plan library until getting to all leaves of paths we got in time-stamp t. The worst-case complexity of this process is $O(N^2T)$, where N is the plan library size, and T is the number of observations.

4.2.2 Efficient UPR Hybrid Algorithms

We developed a set of algorithms that calculates the expected utilities of hypotheses (Equation 4.1) in worst-case runtime complexity $O(NDT)$, where D is the depth of the plan library (N,T are as above). The algorithms are based on the observation that the structural constraints on the plan library are such, that all the paths from any path (hypothesis) true at time $t-1$, to a given hypothesis X_i, true at time t, must necessarily go through a single node S that is a part of X_i. Moreover, S is necessarily a common node to X_i and one or more paths at time $t-1$. If we can compute α_S and γ_S up to this node S, then we could propagate from it to all paths X in which it is a part, that are true at time t. In other words, we can reuse the summation α_S and γ_S for all hypotheses in which S participates.

This translates into the following procedure. We begin with the leaves of all $t-1$ hypotheses W_k ($1 \leq k \leq n$). We sum the utilities and multiply the probabilities while climbing up from the leaves along the hierarchy, all the way to the root, storing intermediate results in the internal nodes w^j (plan-steps) of the hierarchy. We then look for internal nodes (i) that have a child marked at time t (i.e., w^j is a common parent, $\mathrm{Eq}(w^j, x^j)$ is true); or (ii) that have a sequential transition to an internal node marked at time t (i.e., $a^j_{w,x} > 0$). A node (marked time t) that satisfies either of these cases is one through which one or more time t hypotheses X_i pass, i.e., a node S as above. We then propagate down the calculated probability and utility downward (β_S and δ_S).

This process is described in Algorithms 1–3. The *CalcProbAndUtils* algorithm (Algorithm 9) calculates the probabilities and utilities of all valid hypotheses that the symbolic algorithm had returned (Lines 1–2), normalizes these probabilities (Line 3) and find the hypothesis with the maximum

probability and hypothesis with the maximum utility (Line 4). To calculate these probabilities and utilities, it goes over all matching paths M in time stamp $t-1$, we got from SBR (which we did not visit yet) and calls the *PropagateUpAndDown* algorithm (Algorithm 10).

The *PropagateUpAndDown* algorithm (Algorithm 10) calculates P, which holds the probability from paths in $t-1$, and calculates E, which holds the expected utility from paths in $t-1$. The algorithm first initialize P and E with the probability and expected utility of B that was calculated in $t-1$, were B is a leaf of one of the paths from $t-1$. It also mark B as visited, not to calculated it twice. Second, it propagate P and E to behaviors that have sequential edges from B with tag t and to B itself if it has self cycle with tag t (Lines 5–8). This propagation is done by calling *CalcDown* algorithm (Algorithm 11), that will be described later. Then, the *PropagateUpAndDown* algorithm goes up along the hierarchy from the leaf B, accumulating the probabilities and utilities (P and U) from time $t-1$ by recursive call to *PropagateUpAndDown* algorithm (Lines 11–17) and propagating them down to leaves that marked with time-stamp t (Lines 18–23) using the *CalcDown* algorithm (Algorithm 11).

The *CalcDown* algorithm (Algorithm 11) goes down along the hierarchy and multiplies probabilities and accumulate utilities (Lines 1–2) until reaching the leaf of path at time t. When reaching to leaf node (Line 3) it adds the probability to the probability of this node and adds the utility to the utility of the node (Lines 4–5). Therefore, the probabilities and utilities of the leaves are saved in the tree in its leaves, and can be collected, normalized and analyzed later by algorithm 9.

Algorithm 9 CalcProbAndUtils(SBR $t-1$ results W,SBR t results X, Plan Library g, Time-stamp t)

1: **for all** $v \in W$ not visited **do**
2: $\quad PropagateUpAndDown(v, g, t, root(g))$
3: $Normalize(X, g, t)$
4: $TopU \leftarrow$ FindMaxU(X,g,t)

To illustrate these UPR algorithms in action, let us examine a portion of a plan library in Figure 4.5. Suppose that in time $t-1$ the SBR had returned that the two plan-steps C and D are matching, and G in time-stamp t. To calculate the expected utility with the naive algorithm, we would traverse the plan library in the following manner: $E(G|C,D) = (\alpha_C^A \cdot$

Algorithm 10 PropagateUpAndDown(SBR $t-1$ path v, Plan Library g, Time-stamp t, End Plan r)

1: $B \leftarrow leaf(v)$
2: $P \leftarrow P(B, t-1)$
3: $E \leftarrow E(B, t-1)$
4: Mark B as visited
5: $S \leftarrow$ All behaviors tagged with t with sequential edge from B
6: **for all** $s \in S$ **do**
7: $\quad CalcDown(P, a_{B,s}, E, e_{B,s}, B)$
8: **if** $\exists$ Self cycle on B tagged with t **then**
9: $\quad CalcDown(P, a_{B,B}, E, e_{B,B}, B)$
10: **while** $B \neq r$ **do**
11: $\quad P \leftarrow P \times a_{B,end}$
12: $\quad E \leftarrow E + P \times e_{B,end}$
13: $\quad B \leftarrow Parent(B)$
14: $\quad C \leftarrow$ All children of B tagged with $t-1$ and not visited
15: $\quad$ **for all** $c \in C$ **do**
16: $\quad\quad P \leftarrow P + PropagateProbUpAndDown(leaf(c), g, t, B)$
17: $\quad\quad E \leftarrow E + PropagateProbUpAndDown(leaf(c), g, t, B)$
18: $\quad C \leftarrow$ All children of B tagged with t
19: $\quad$ **for all** $c \in C$ **do**
20: $\quad\quad CalcDown(P, a_{c,end} \times \Pi_{B,c}, E, e_{c,end} + \Psi_{B,c}, c)$
21: $\quad S \leftarrow sequentialEdges(B, t)$
22: $\quad$ **for all** $s \in S$ **do**
23: $\quad\quad CalcDown(P, a_{B,s}, E, e_{B,s}, B)$
24: $return(P, E)$

Algorithm 11 CalcDown(probability P, probability p, expected utility E,utility u,Plan B, Plan Library g, Time-stamp t)

1: $P \leftarrow P \times p$
2: $E \leftarrow E \times p + u \times p$
3: **if** $isLeaf(B)$ **then**
4: $\quad P(B, t) \leftarrow P(B, t) + P$
5: $\quad E(B, t) \leftarrow E(B, t) + E$
6: **else**
7: $\quad C \leftarrow$ children of B that tagged with t
8: $\quad$ **for all** $c \in C$ **do**
9: $\quad\quad CalcDown(P, \Pi_{B,c} \times a_{B,B}, E, \psi_{B,c} + e_{B,B}, B)$

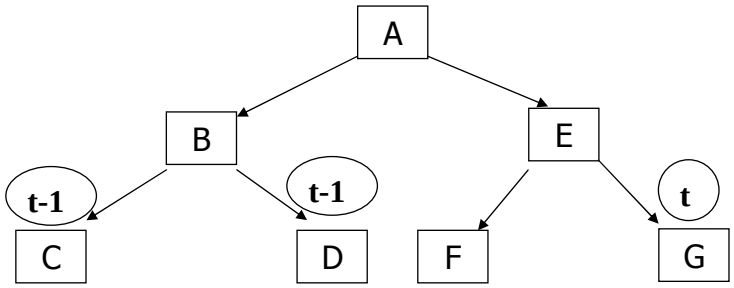

Figure 4.5: Efficient UPR Example.

$\beta_A^G + \alpha_D^A \cdot \beta_A^G) \times (\gamma_C^A + \delta_A^G + \gamma_D^A + \delta_A^G)$. With the efficient UPR: $E(G|C,D) = [(\alpha_C^B + \alpha_D^B) \times \alpha_B^A \times \beta_A^G] \times (\gamma_C^B + \gamma_D^B + \gamma_B^A + \delta_A^G)$. Meaning that the probabilities and utilities of C and D are stored in B, so we are not traversing the plan library more than necessary.

Complexity Analysis. The run-time complexity of the algorithm is $O(NDT)$: We first propagate the $t-1$ expected utilities up the hierarchy, not visiting plans that already been visited, in worst-case time $O(N)$. Then, calculating β for different depths, for paths tagged with t, is $O(ND)$. We do this for every observation, of which there are T, thus the overall complexity is $O(NDT)$.

Note the reliance on the underlying SBR: Since the symbolic recognizer provides the possible paths at times $t-1, t$, we do not need to consider all possible paths, and can begin the propagation process directly at the leaves of paths. Hopefully, many paths are disqualified by the symbolic algorithm, due to temporal coherence; in that case, we expect performance in practice to improve significantly over the worst case complexity.

4.2.3 Discussion

We showed here a hybrid UPR technique that reduces the complexity of influence diagrams. However, the technique sacrifices expressivity:

- *Multiple goals.* The hybrid UPR can not handle multiple goals, where the agent pursuing number of goals in parallel. The agent may begin with one plan, and interrupt its execution to execute another, only to return to the remaining plan steps in the first plan. The Hybrid-UPR can not handle this capability, though this is possible in Influence

Diagram.

- *Other evidence.* The hybrid UPR does not support inserting context (like destination in the previous example in Section 4.1.2). The use of context is important for adding different costs for different contexts. For example costs are different for liquor-store or jewelry store.
- *Markovian assumption.* The Hybrid UPR is based on Hierarchical HMM which assumes the Markov property, therefore is less general from the influence diagram that are based on Bayesian Networks.

Chapter 5

Recognizing Multi-Agent Dynamic Groups

This Chapter introduces *Dynamic Hierarchy Group Model* (DHGM), a dynamically-maintained structure for tracking groups and sub-groups when the groups split into sub-groups and/or merge with other groups. We use a plan recognizer with each agent. For this purpose, we use here a highly efficient symbolic plan recognizer (SBR) introduced in Chapter 3 that is used to filter through hypotheses, maintaining only those that are consistent with the observations. Note, however, that any plan recognizer can be used. Then, we provide its complexity analysis in Section 5.2. Section 5.3 shows an illustrative application for detecting suspicious behavior using DHGM.

5.1 Dynamic Hierarchy Group Model

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

The DHGM is a dynamically-maintained structure that reflects the current groups of the agents, and the history of these groups. This structure is built dynamically with every observation. Each node in the DHGM represents a group with one or more agents that are executing the same behavior (i.e,. the plan recognizer identified the same plan-steps in the plan library for this group). Each node in the DHGM points to leaves in the plan library that the agent is assumed to be executing. Each node also holds a time-stamp

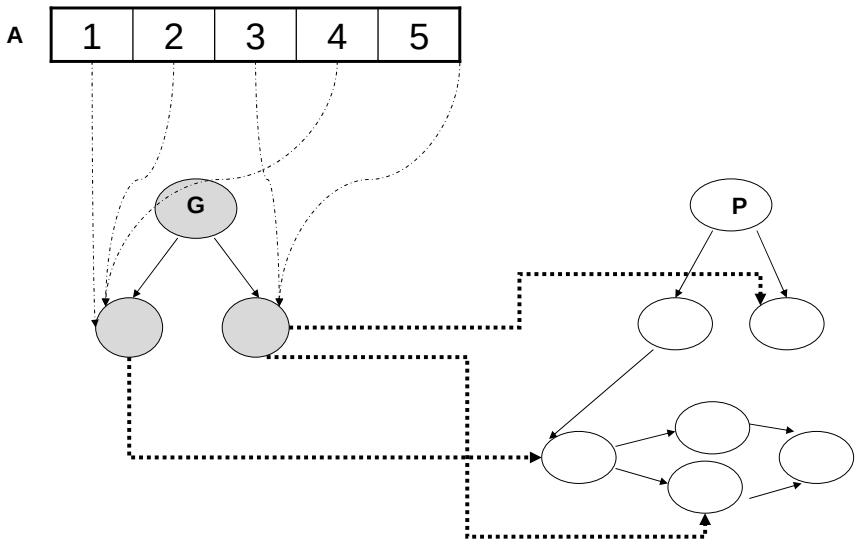

Figure 5.1: Example Dynamic Hierarchy Group Model noted with G.

counter that counts the number of consecutive time-stamps that the agent the group were in this branch. From each node there are branches to sub-groups, meaning that the group had been split and executing now different plan-steps in the plan library. When the agent returns to its group, or join other sub-group, the branches are merged.

Figure 5.1 shows portion of a DHGM (noted with G) that points to plan library P. On top there is an array of agents A, that points to their place in the Group Hierarchy. In this figure, agents 1,2 and 4 belong to one sub-group and agents 3,5 belong to a second sub-group.

The DHGM maintenance process is described in Algorithms 1–2. The GroupDetection algorithm (Algorithm 12) initializes the DHGM with single root node that all agents belongs to this node, and it points to the given plan library. With each new observation it calls *UpdateGroup* algorithm (Algorithm 13).

The UpdateGroup algorithm (Algorithm 13), traverses the DHGM bottom up. For each leaf node it executes the SBR algorithm on the current observation for all agents that belong to this leaf (Algorithm 13 line 7), with

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

Algorithm 12 GroupDetection(Plan Library p)

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

An example: We will demonstrate the process in action with a simple example shown in Figure 5.2 and the plan library in 5.3. Assume that there are 100 agents in the airport that execute the *position* plan-step in time-stamp $t = 1$. The DHGM here has one root node with 100 agents and points to all *position* instances in the plan library (note that for presentation clarity, Figure 5.2 points to the plan-step name and not to all instances in the plan library).

Now, in time-stamp $t = 2$ there are 50 agents that continue executing the *position* and 50 other agents execute the X-Ray plan-step, 20 without bags and 30 with bags. The DHGM will have the following structure a root node with 3 leaves: one for position, one for x-Ray with bag and one for X-Ray without bag. Now assume that in time stamp $t = 3$ the 50 agents that were in the security with or without bags, now execute the gate plan-step, and the 50 agents moved from the *position* to execute the X-Ray with or without bag (25 in each group). Now the DHGM will have under root two nodes, one

Algorithm 13 UpdateGroup(Group Hierarchy Node *groupNode*, Time-stamp t, Observation *obsrvArr*)

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

that points to the gate instances in the plan library and one that splits into 2 nodes: X-ray with bag and X-Ray without bag. In time stamp $t = 4$ all agents in the X-Ray without bag execute the gate plan-step, and under X-Ray with bag 24 agents execute the gate and one agent execute position. The DHGM will have under root two nodes, one that points to the gate instances in the plan library and one that splits into 2 nodes, one node with 25 agents that execute the *gate* plan-step and another node that splits to two nodes: one with 24 agents that execute *gate* and 1 agent that executes *position*. Note that time-stamps counters were omitted, for presentation clarity.

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

5.2 Complexity Analysis

Using the Dynamic Hierarchy Group Model for plan recognition in multi-agent scenarios is less expensive in space terms than operating individual plan

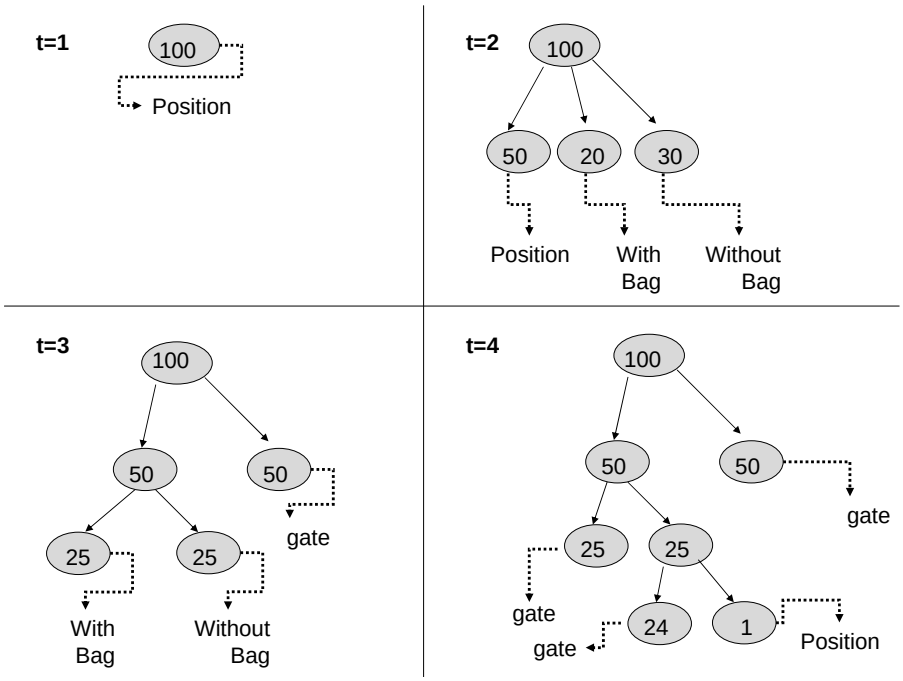

Figure 5.2: Example of the process of creating Dynamic Hierarchy Group Model. The Number on the node denotes number of agents belong to this group.

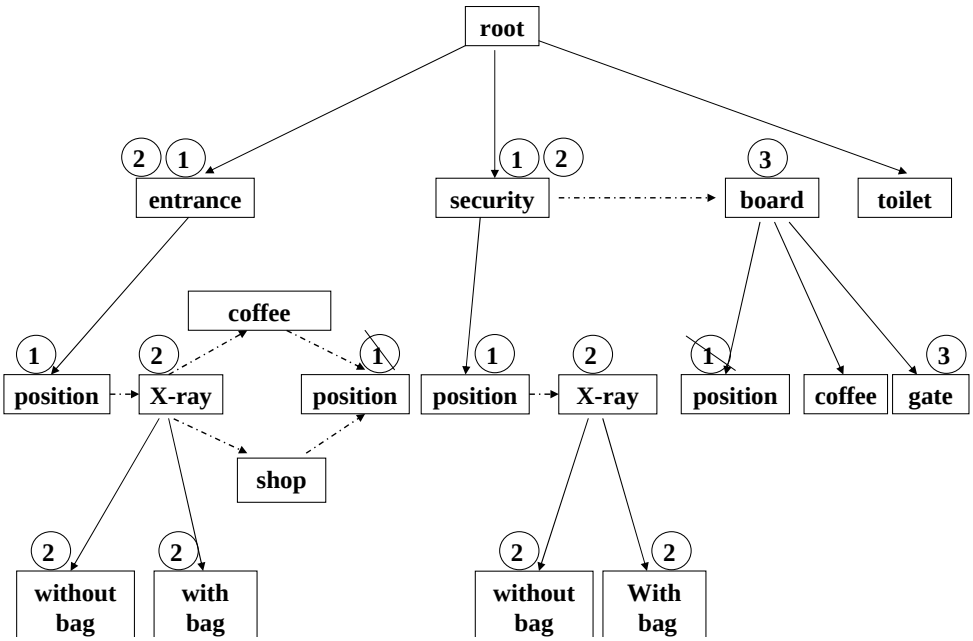

Figure 5.3: An example of plan library.

recognition method for each agent. We do not need to hold a plan library for each agent, but to use the same plan library with different notations for each group. Therefore the space complexity is $O(Lg)$, where L is the plan library size and g is the maximum number of groups. This should be compared to $O(Ln)$, where n is the number of agents.

The run-time complexity is larger than running individual plan recognition, since we not only run the SBR algorithm, but also check the connections between agents. we go over the group hierarchy $O(g)$ bottom up, where g is the number of groups. For each node: we execute SBR (Algorithm 13 line 7): $O(LD)$, where L is the plan library size, and D is the depth of the plan library. This is done for all agents in this node $O(LDn)$ (Algorithm 13 line 5), where n is the number of agents in the group. Then, compare all agents results $O(nlogn)$ with sort (Algorithm 13 line 9). Therefore, $O(gnLD + gnlogn)$. The gn (number of groups multiply the size of the group) is actually the total number of agents N. Therefore, it is $O(NLD + Nlogn)$, this is for one observation. It will be $O(TNLD + TNlogn)$, where T is the number of observations.

5.3 An Illustrative Application

In this section we present an initial method for detecting suspicious behavior using DHGM. There are suspicious behaviors that can be captured only when tracking agents with respect to their group and not only as individuals. For this purpose, we propose a number of heuristics for detecting suspicious behavior, using the information from the *Dynamic Hierarchy Group Model* that was introduced in Section 5.1.

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

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

ditions given a DHGM is described in Algorithm *CheckSuspiciousBehavior* (algorithm 14).

Using the DHGM we can identify agents that behave differently from other agents in the same group. For example, let's use the DHGM in the example in 5.2, we can see from the information on the DHGM that there is only one agent out of 24 that executes the *position* plan-step, and did not get to the *gate* like other agents. We would like to keep an open eye on this agent, since she might be dangerous.

Algorithm 14 CheckSuspiciousBehavior(Group Hierarchy Node *groupNode*)

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

Previous work has shown that given a model of hierarchical relationship between agents, one can identify deviations from the model [46] (which indicate anomalies), and may increase efficiency of recognition [49]. However, this previous work is limited to very specific social structures, where agents form teams based on agreements as to specific plans. In order to detect disagreements, the monitoring agent must first know which plans are ideally to be agreed upon. In contrast, in our work we do not have static social structure that is given in advance, but observations on dynamically changing structure of groups. For example, a group of passengers in the airport may seem like one group when standing in the security check line, and afterward when splitting into two groups, the structure needs to be modified.

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

The *queue-cutting problem* can be solved using the DHGM. In this case the SBR algorithm will return that there are no results for this agent according to the plan library. This will happen since the propagation phase has failed; therefore, we can check whether the matching phase has resulted with

matching plan-steps and compare them to the agent's history group. If the agent is executing the same plan-step as some of its group members, then we consider it as a legal behavior, and do not consider her a suspect as the single agent SBR would do.

Part II

Detecting Anomalous and Suspicious Behavior

In this part we apply the plan recognition algorithms described in Part I, to realistic problems in surveillance and activity recognition. In particular, we consider challenges in detecting anomalous and suspicious behaviors.

We begin by considering detection of anomalous behavior (Chapter 6). Here, the plan library represents normal behavior; any activity which does not match the plan library is considered abnormal. This approach can be effective in applications where we have few or no examples of suspicious behavior (which the system is to detect), but many examples for normal behavior (which the system should ignore). This is the case, for instance, in many vision-based surveillance systems, in public places. A symbolic plan recognition system is useful in recognition of abnormal patterns, such as walking in the wrong direction, taking more than usual amount of time to get to the security check, etc. The symbolic recognizer can efficiently match activities to the plan library and rule out hypotheses that do not match. When the resulting set of matching hypotheses is empty, the sequence of observations is flagged as anomalous. The symbolic algorithm is very fast, since it rejects or passes hypotheses without ranking them.

However, detection of abnormal behavior is not sufficient. There are cases where a normal behavior should be treated as suspicious. In these cases, we cannot remove the behavior from the plan-library (so as to make its detection possible using the anomalous behavior detection scheme outlined above), and yet we expect the system to detect it and flag it.

In Chapter 7, we turn to examine the use of UPR for recognizing suspicious behavior. Here the plan library explicitly encodes behavior to be recognized, along side any costs associated with the recognition of this behavior. This allows the UPR system to rank hypotheses based on their expected cost to the observing agent. As we shall see, this leads to being able to recognize potentially dangerous situations, despite their low likelihood.

Chapter 6

Detecting Anomalous Behavior

There have been several attempts at utilizing plan recognition for recognition of suspicious, erroneous, or anomalous behavior, e.g., [18,25,28,32,56,66,91]. These have mostly operated under the assumption that a plan library is available that covers this intended *negative behavior*, and thus recognition of hypotheses implying such behavior is treated no differently from recognition of hypotheses implying no suspicion.

Recently, a different approach has been taken by several researchers in which the plan library is used in an inverse fashion. The plan library is limited to covering only *positive behavior*. When a plan-recognizer is unable to match observations against the library (or generates hypotheses with very low likelihood), an anomaly is announced [27, 55].

We propose an anomalous behavior recognition system, presented in Figure 6.1. The input of the system is an observation sequence. The system is composed of an SBR (Symbolic Plan Recognition) module, which extracts coherent hypotheses from the observation sequence. If the set of hypotheses is empty, it declares the observation sequence as anomalous.

The symbolic plan-recognition algorithm that was introduced in Chapter 3 is useful in recognition of abnormal behavior, since it is very fast (no need in ranking hypotheses), and can handle key capabilities required by modern surveillance applications.

We evaluate the use of SBR for anomalous behavior recognition, using real-world data from machine vision trackers, which track movements of people, and report on their coordinate positions. In Section 6.1 we describe the experiments setup. Section 6.2 describes in details the experiments we conducted, along with their results on video clips and data from the CAVIAR

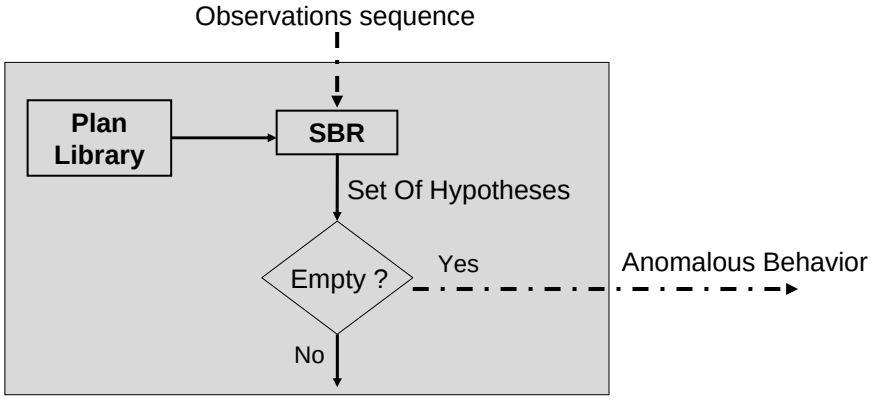

Figure 6.1: Anomalous Recognition System. Inputs and outputs for the system are in dashed arrows.

project [1]. Section 6.3 presents results from data sets gathered as part of our participation in the AVNET Consortium, which developed technologies for detection of criminal or otherwise suspicious objects and suspects.

6.1 Experiments Setup

We conduct our experiments in the context of a vision-based surveillance application. The plan library consists of discretized trajectories which correspond to known-to-be-valid trajectories. We use a learning algorithm, described briefly in Section 6.1.1, to construct this plan library. To evaluate the results of the SBR algorithms we use precision and recall measurements that are widely used in statistical classification, as described in Section 6.1.2.

6.1.1 The Learning Algorithm

We use a simple learning algorithm that was developed for the purpose of building plan recognition libraries based on examples of positive (valid) trajectories only. The learning algorithm is not part of this dissertation, and is fully described in [47, 48]. We provide a description below of its inputs and

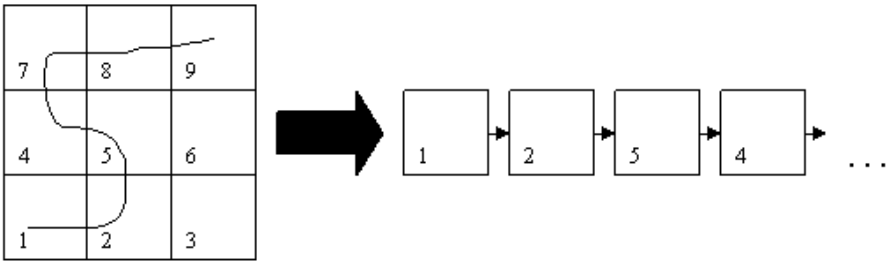

Figure 6.2: Result of running naive learning algorithm on one trajectory

outputs, as those are of interest here.

The learning algorithm L receives a *training set* that contains observation sequences S. Each observation sequence $s \in S$ is one trajectory of an individual target that composed of all observations (samples) o_i, where i is an ordering index within s. Each observation o_i is a tuple $\langle x_i, y_i, t_i \rangle$, where x_i, y_i are Cartesian coordinates of points within W, and t is a time index.

The learning algorithm divides the work area W using a regular square-based grid. Each observation $\langle x_i, y_i, t_i \rangle$ is assigned a square that contains this points. For each trajectory of an individual target (s) in the *training set*, it creates a sequence of squares that represents that trajectory.

The output of the algorithm is a set K of discretized-trajectories, each a sequence of grid cells, that are used together as the plan library for the recognition algorithm. Figure 6.2 shows a work area that is divided into nine squares, with one trajectory. The resulting output of the algorithm is a sequence of squares that defines that trajectory (on the right of the figure).

The grid's square cell size is an input for the learning algorithm. By adjusting the size of the square we can influence the relaxation of the model. By decreasing the size of the square the learned library is more strict (there is less generalization); and in contrast, too large a value would cause over-generalization. Small square size may result in over-fitting; a trajectory that is very similar to an already seen trajectory, but differs slightly will not fit the model. By increasing the size of the square, the model is more general and it less sensitive to noise, but trajectories that are not similar might fit.

People are rarely twice in the exact same position. As a result, the training set may contain many sample trajectories that differ by very small distance. Part of the challenge in addressing this lies in adjusting the square

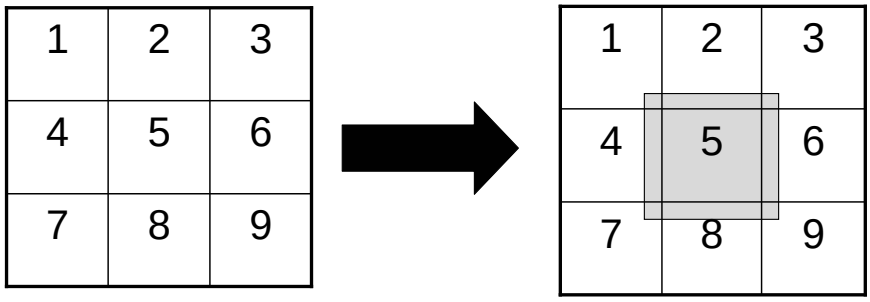

Figure 6.3: Demonstrating position overlap. Added position overlap for square number 5.

size, as described above. However, often a part of a trajectory would fall just outside the cell that contains the other examples of the same trajectories.

To solve this, the learning algorithm has another input parameter, called *Position Overlap Size*. The position overlap prevents over-fitting to the training data, by expanding each square such that it overlaps with those around it (see Figure 6.3, where the position overlap is shown for square number 5). Any point in a trajectory that lies in an overlapping area is defined to match both the overlapping square, as well as the square within which it falls. Thus, for instance, a point within cell 3 in Figure 6.3, at its bottom left corner (within the darkened overlapping area of cell 5) would match cell 3 and 5, as well as 6 and 2 (since these cells also have their own overlapping areas). Essentially, this is the analogous to having non-zero observation emitting probabilities for the same observation, from different states in a Hidden Markov Model.

6.1.2 Performance Measurements

To evaluate the results we use two widely used measurements in statistical classification precision and recall [90], defined below. These are always used in reference to a *testing set*, which contains the trajectories for which we want to measure the performance of the system.

Definition 7. *True Positives.* Number of trajectories correctly labelled as anomalous.

Definition 8. *False Positives.* Number of items incorrectly labelled as anomalous.

Definition 9. *False Negatives.* Number of items which were not labelled as anomalous but should have been.

Using these definitions, we can define the precision and recall measures.

Definition 10. *Precision.* Number of true positives divided by the total number of elements labelled as belonging to the class (sum of true positives and false positives).

Definition 11. *Recall.* Number of true positives divided by the total number of elements that actually belong to the class (sum of true positives and false negatives).

In our case, we have two classes: anomalous and non-anomalous. Therefore, the true positives are the number of anomalous trajectories that were classified correctly as such. The false positives are the number of non-anomalous trajectories that were mistakenly classified as anomalous. The false negatives are the number of anomalous trajectories that were not classified as anomalous (but should have been).

A perfect precision score of 1 means that every anomalous trajectory that was labelled as such was indeed anomalous (but this says nothing on anomalous trajectories that were classified as non-anomalous). A perfect recall score of 1 means that all anomalous trajectories were found (but says nothing about how many non-anomalous trajectories were also classified as anomalous).

Figure 6.4: A typical frame of image sequence in CAVIAR Project

6.2 CAVIAR Data

We utilize two sets of real-world data to evaluate SBR's performance as an anomalous behavior recognizer. Experiments with the first set are described in this section. The second set is described in Section 6.3.

The first set of experiments were conducted on video clips and data from the CAVIAR Project[1] [1]. The CAVIAR project contains a number of video clips with different scenarios of interest: People walking alone, standing in one place and browsing, etc. The videos are 384×288 pixels, 25 frames per second. Figure 6.4 shows a typical frame. The ground truth tracking data for these sequences, in pixel coordinates, is available as well, and was determined by hand-labelling the images (this was done by the CAVIAR data maintainers).

[1]EC Funded CAVIAR project/IST 2001 37540, found at URL: http://homepages.inf.ed.ac.uk/rbf/CAVIAR/

We use the ground-truth data to simulate the output of realistic trackers, at different levels of accuracy. To do this, each ground-truth point is converted by homography to position, given in centimeters, in the 2D plane on which the subjects move. We add then noise with normal distribution to simulate tracking errors. Higher variance simulates less accurate trackers, and low variance simulates more accurate trackers. In the experiments reported on below, we use a standard deviation of 11cm diagonal (8cm vertical and horizontal). The choice of noise model and parameters is based on information about state-of-the-art trackers (e.g., [68]).

To create a large set of data for the experiments (representing different tracking instances, i.e., the tracking results from many different video clips), we simulated multiple trajectories of different scenarios, and trained the learning system on them, to construct a plan library. This plan library was then used with different trajectories to test the ability of the algorithms to detect abnormal behavior.

6.2.1 Simple Abnormal Behavior

In the first experiment we tested simple abnormal behavior. We simulated three kinds of trajectories:

1. *Curved path A.* Taken from the first set *One person walking straight line and return* scenario in CAVIAR, the walking straight (without the return path). This path was defined by us as the basis for normal behavior.

2. *Curved path B.* Taken from the first set *One person walking straight* scenario in CAVIAR, which is similar to the above, but curves differently towards the end. We use this to evaluate the system's ability to detect abnormal behavior.

3. *U-Turn.* Taken from the first set *One person walking straight line and return* scenario in CAVIAR. Identical to Curve path A as above, but then at the end all subjects turn around and go back.

Figure 6.5 shows the three kind of trajectories. The arrow shows the starting position of the trajectories. The end-points lie at the other end (movement right to left).

We created 100 simulated trajectories of each type, for a total of 300 trajectories. We trained a model on 100 noisy trajectories from *Curved path*

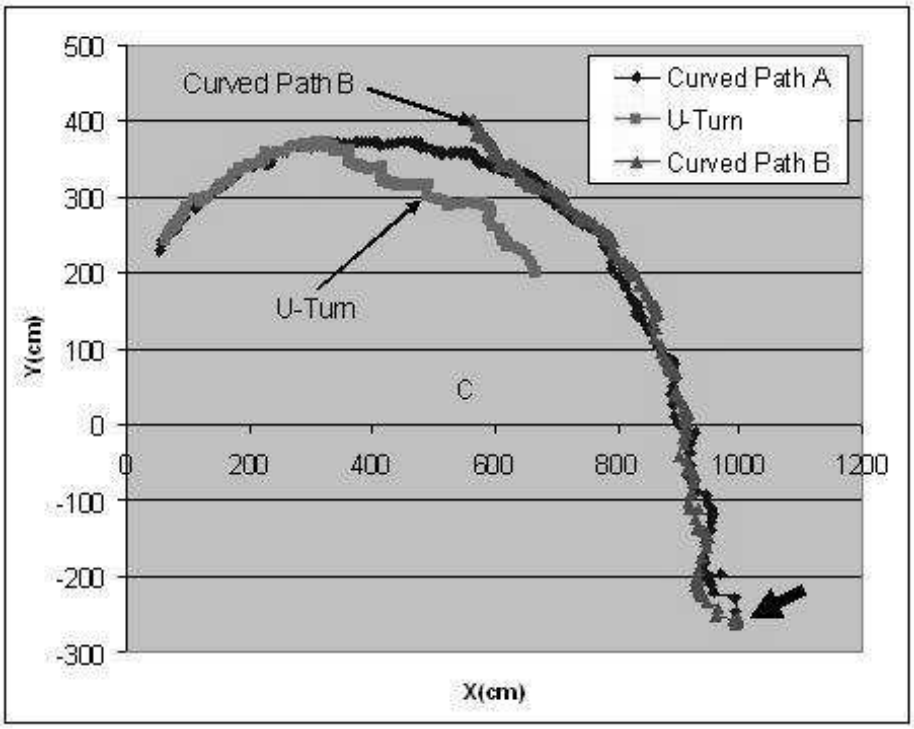

Figure 6.5: Three Trajectories: Legal path (Curved Path A), suspicious path (Curved Path B), and return path (U-Turn) from CAVIAR data. The arrow points at the starting point.

A, using the learning system described in Section 6.1.1. In this experiment we fixed the grid cell size to 55cm, and we vary the plan library relaxation parameter (called *Position Overlap Size* in Section 6.1.1). The cell size was chosen such that it covered the largest distance between two consecutive points in the training set trajectories.

Figure 6.6 shows the true positives versus false positives. The x axis is the plan library relaxation, and the y axis is the number of trajectories (subjects). We can see that the system starts stabilizing in plan library relaxation of 11 (the number of false positives is zero), and after a plan library relaxation of 15 the system is too relaxed; the number of abnormal trajectories that are *not* detected slowly increases.

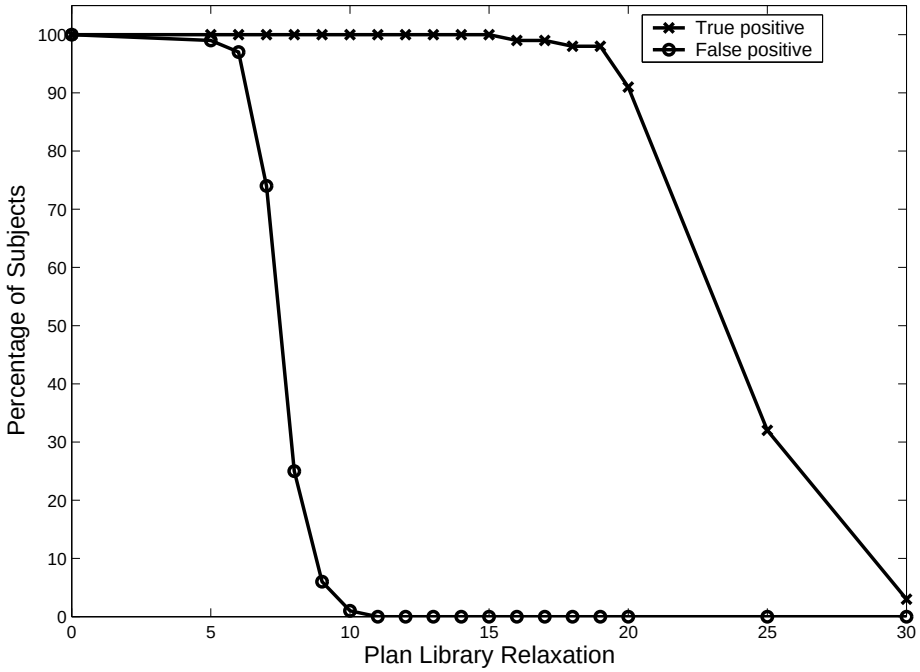

Figure 6.6: True positive vs. false positives on CAVIAR data.

Figure 6.7 shows the precision and recall of the system. The x axis is the plan library relaxation, and the y axis is the number of trajectories (subjects). The precision increases till perfect score of 1 from relaxation 11. The recall starts with perfect score of 1 and decreases slowly from relaxation 16. As in figure 6.6, we can see that for values of plan library relaxation of 11–15, we get perfect precision and perfect recall of 1.

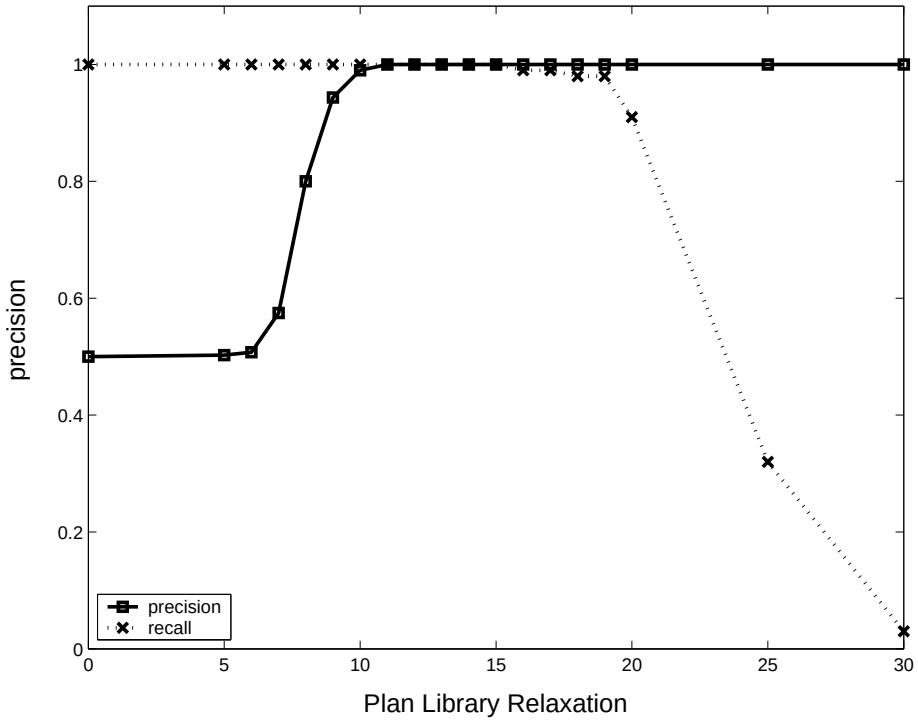

Figure 6.7: Precision and recall on CAVIAR data.

Despite the optimistic picture that the results above portray, it is important to remember that these results ignore the time for detection. However, in practice the time for detection matters.

Figure 6.8 shows the time for detecting the abnormal behavior with standard deviation bars. The x axis is the plan library relaxation, and the y axis is the time (sec) passed until detecting abnormal behavior. In this figure we can see that until plan library relaxation of 12, the time for detection is negative, since we detect too early before the abnormal behavior is taking

place. Two seconds is the maximum of the graph, since this is the time that the scene was over, therefore detecting at 2 seconds is too late.

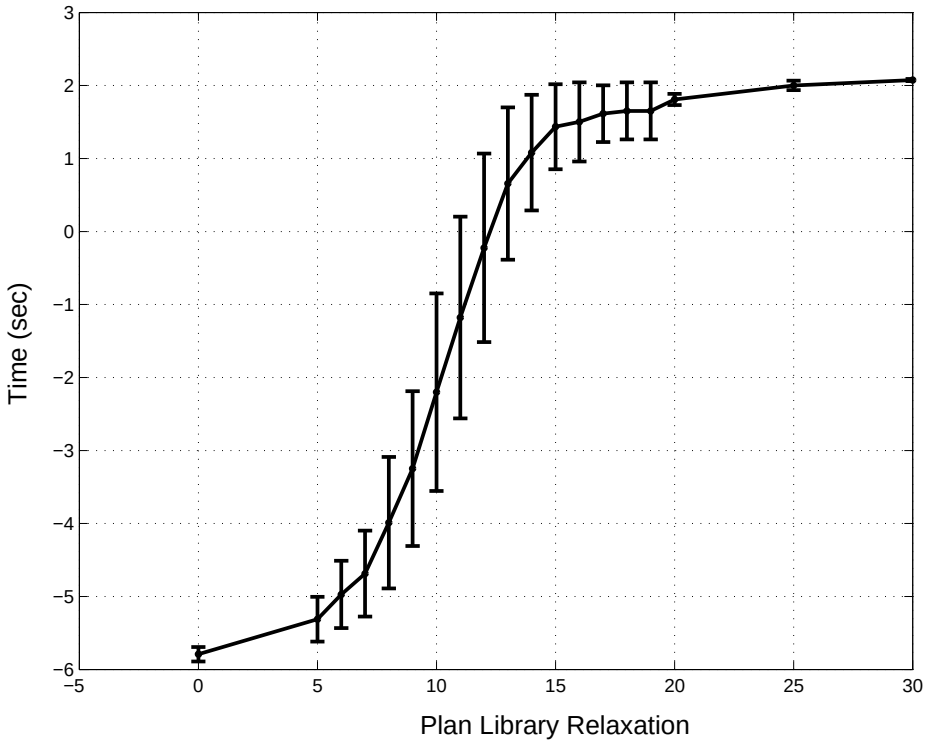

Figure 6.8: Time for detection suspicious path on CAVIAR data.

Figure 6.9 shows the trade-off between detecting too late and detecting too early as a function of the plan library relaxation (where too early is before the split and too late is the end of the abnormal path). The x axis is the plan library relaxation, and the y axis is the percentage of subjects that were detected too early or too late. We can see that relaxation of 16 gives the best results of 1% too early and 1% too late. After relaxation of 16, the percentage of trajectories that we detect too late is slowly increases.

The recognition algorithm also capable of detecting anomaly behavior in the direction, and not only in the position. The following experiment demonstrate this capability; here, we evaluate the use of the system in identifying the u-turn trajectories in the data-set. We sampled 100 instances of the *U-Turn* trajectory with gaussian noise and checked the time for detection.

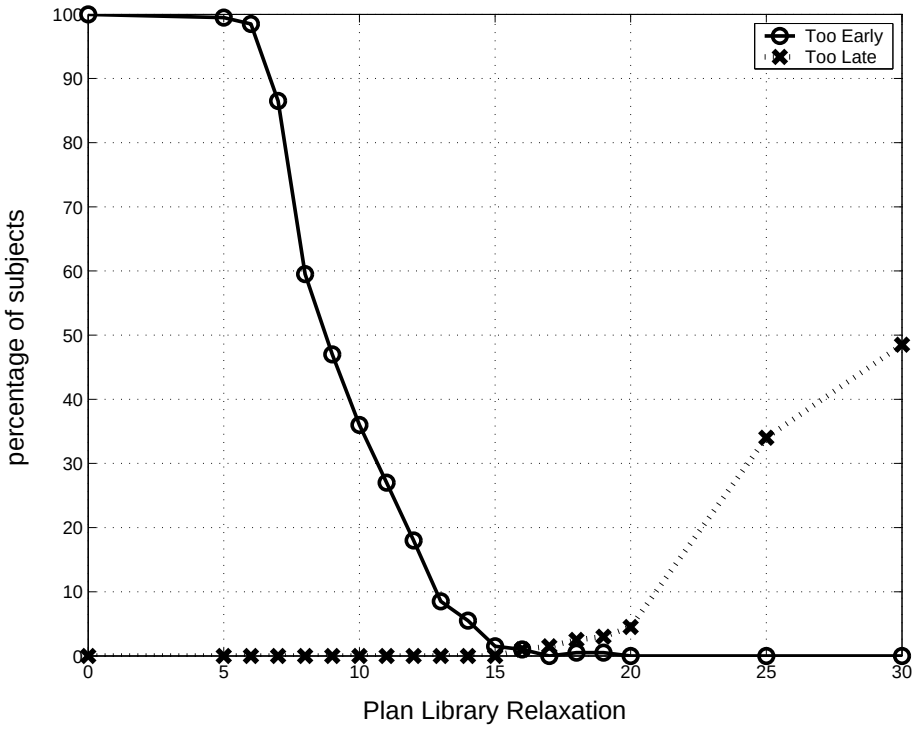

Figure 6.9: Too early detection and too late detection on CAVIAR data.

We trained a model on the same 100 noisy trajectories from *Curved path A* that we used in the first experiment. We first checked the time for detecting abnormal-behavior as u-turn.

Figure 6.10 shows the time for detecting abnormal behavior versus the plan library relaxation. The x axis is the plan library relaxation, and the y axis is the time (sec) passed until detecting the u-turn. We can see that until plan library relaxation of about 10, the time for detection is negative, since we detect too early before the u-turn behavior is taking place, and the standard deviation is high. The maximum detection time is 2.5 seconds after the turn is taking place. Figure 6.11 demonstrates the position on the trajectory 1 second after the turn, 2 second after the turn and 10 seconds after the turn.

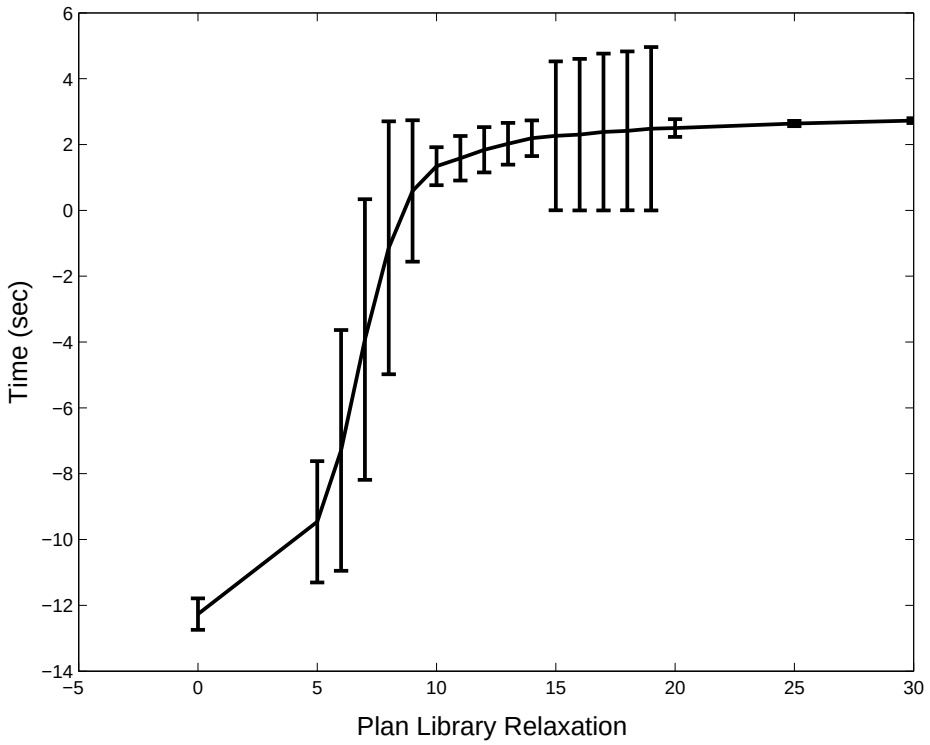

Figure 6.10: Time for detecting U Turn on CAVIAR data.

Figure 6.12 shows the precision and recall for the u-turn experiment as function of the time. The plan library relaxation was set to 15, which is

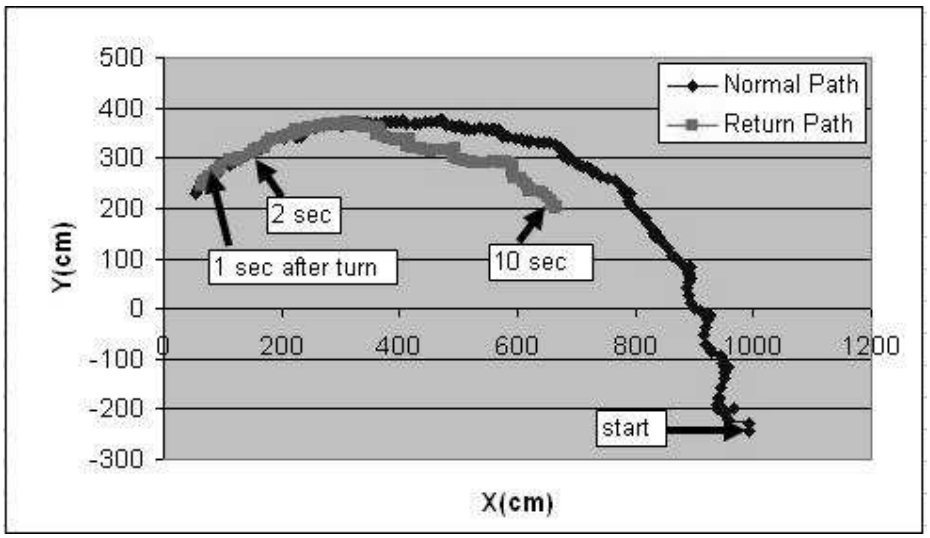

Figure 6.11: U Turn on CAVIAR data.

the best relaxation according to the first experiment. The precision has the perfect score of 1, for plan relaxation of 15 (every suspect that was labelled as suspect was indeed a suspect). The recall starts from score zero and gradually increases, and after about 2.5 seconds it gets the perfect score of 1 (all suspects were found 2.5 seconds after the turn).

6.2.2 Duration

Detecting abnormal behavior based on spatial motion is not enough. There is a need also to recognize abnormal behavior in time. For instance, we would like to recognize as abnormal someone that stays in one place an exaggerated amount of time, or that moves too slowly or too quickly.

Figure 6.13 shows a trajectory taken from the first set *Person browsing and reading for a while* scenario in CAVIAR. In this scenario there is a person that stands in one place for 7.8 seconds. For this experiment, this is defined by us as normal. We sampled 100 instances of this trajectory with gaussian noise and created a model with the learning system (described in Section 6.1.1). Since the learning system does not have the capability of learning durations, we learned on the trajectory without the waiting part, then manually added the maximum duration in the plan library.

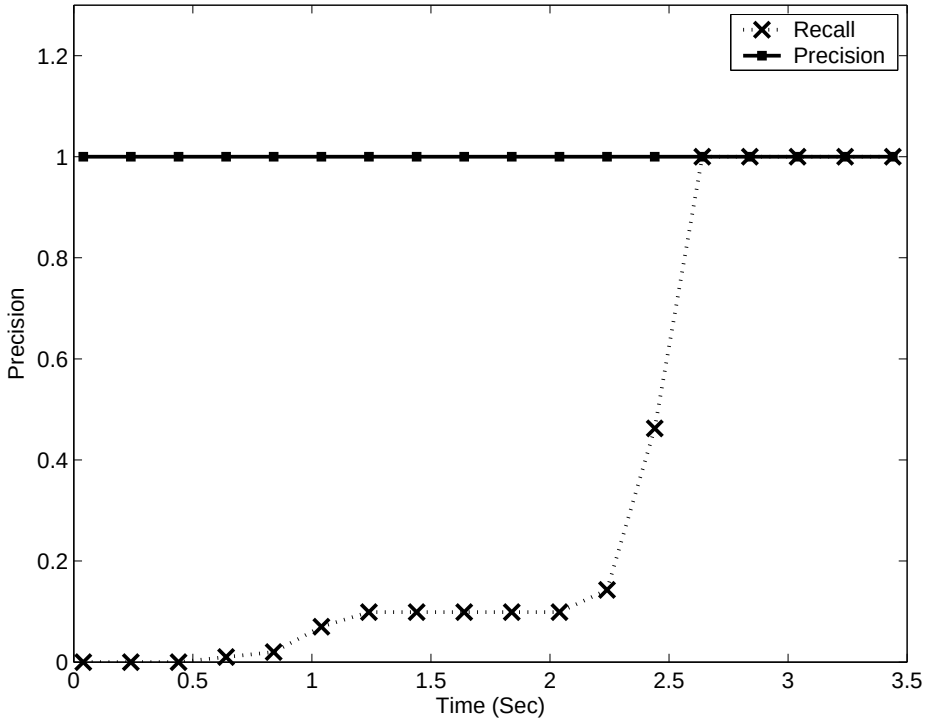

Figure 6.12: Precision and recall for U Turn versus time on CAVIAR data.

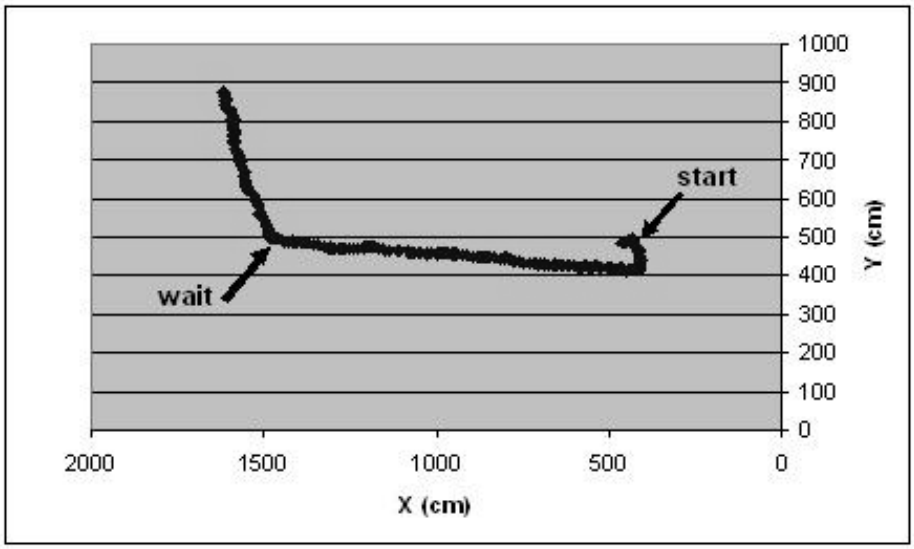

Figure 6.13: Trajectory with waiting time.

We also add *duration relaxation* parameter to the model. We add this parameter to the maximum duration. This is done for two reasons: (a) To avoid over-fitting, noisy trajectory might not fit the model, since the duration is slightly different; (b) an observation can fit more than one plan step (we have position overlap). Therefore we count it as matching in more than one plan step and increase the duration counter. Trajectory might not fit, since we increased its counter more than necessary.

In this experiment we fixed the grid cell size at 55cm, and the plan library relaxation parameter to 15 (the best relaxation according to the first experiment). We vary the *duration relaxation* parameter.

Figure 6.14 shows the time for detecting standing in one place behavior versus the duration relaxation (sec). The x axis is the duration relaxation in seconds, and the y axis is the time (sec) passed until detecting standing in one place more than the normal wait of 7.8 seconds. We can see that until duration relaxation of about 0.6 seconds, the time for detection is negative, since we detect too early before the abnormal waiting behavior is taking place, and the standard deviation is high. After duration relaxation of about 0.6 seconds, the recognition system detects abnormal waiting after about one second.

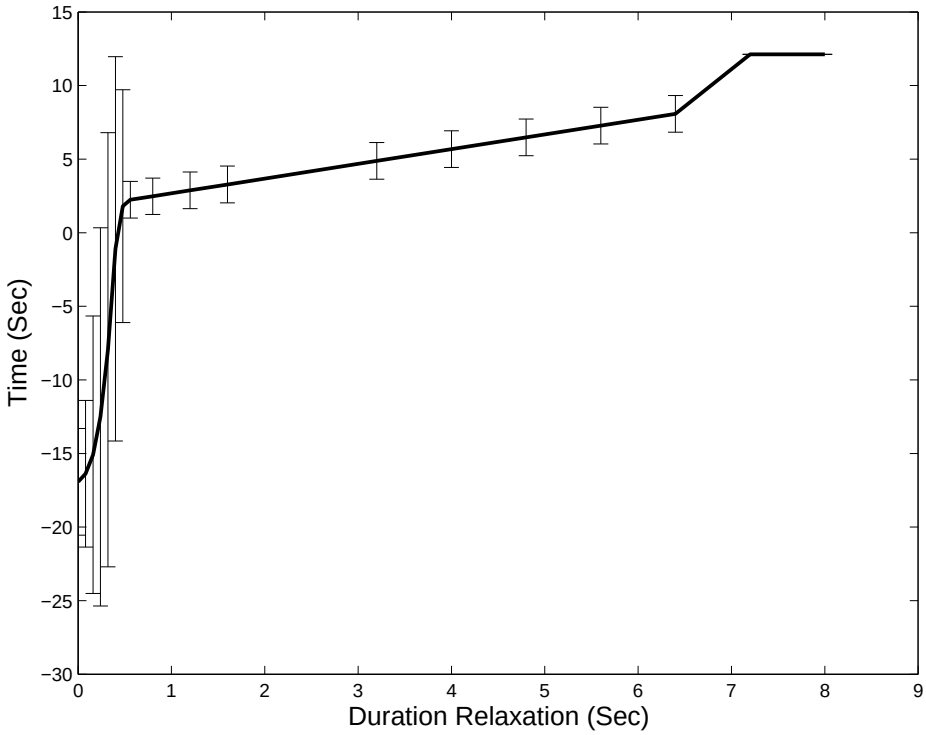

Figure 6.14: Time to detect standing in one place versus Duration Relaxation.

Figure 6.15 shows the precision and recall for the duration experiment as function of the duration relaxation(sec). The precision increases and gets perfect score of 1 starting from duration relaxation of 0.6 seconds (every trajectory that was labelled as such was indeed an anomalous trajectory, after relaxation of 0.6 seconds). The recall starts from a perfect score 1.0 (all anomalous trajectories were found), and decreases after 6 seconds to zero, since the model is too relaxed and anomalous trajectories are classified incorrectly as normal.

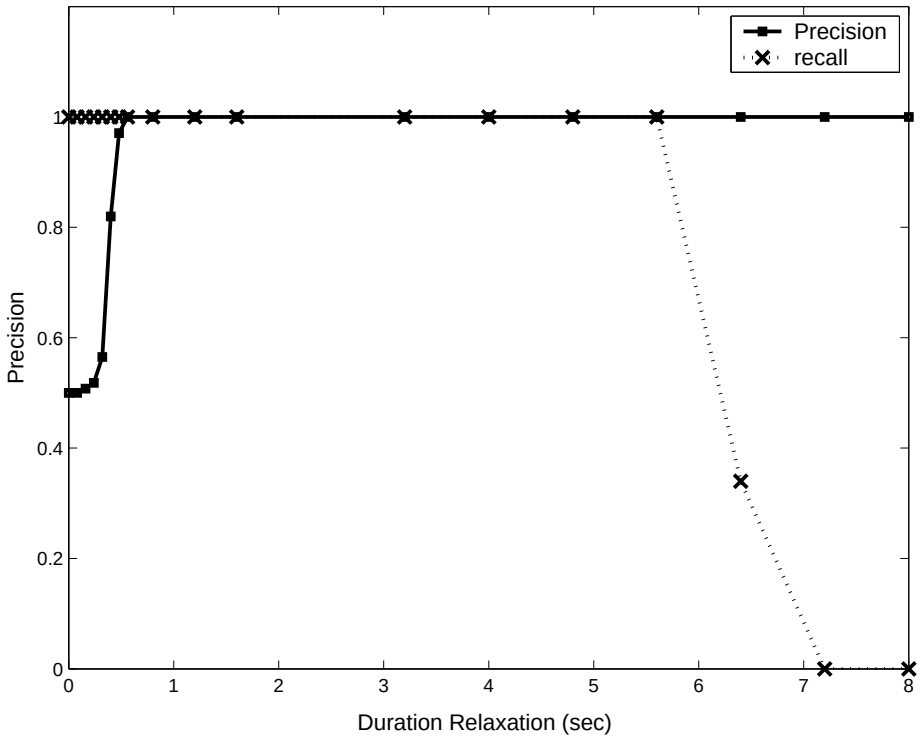

Figure 6.15: Number of suspects in the duration experiment.

6.3 RAFAEL Data

Parts of our work were funded through the AVNET consortium, a government-funded project including multiple industrial and academic part-

ners, for development of suspicious activity detection capabilities. As part of this project, we were given the tracking results from a commercial vision-based tracker, developed by RAFAEL.

We used the RAFAEL data sets in order to evaluate our algorithm. In the first experiment, we got 164 trajectories, all with normal behavior. We ran a 10-fold cross validation test on the data set in order to test the performance. We divided the whole set to 10 data sets, each contained 148 trajectories for train, and 16 trajectories for test (except the last test that contained 20 trajectories for test and 144 for train). We learned a model with square size fixed to be the size of the maximum step in the data (31), and the *position overlap* to be 10.

We checked the number of false positives (the number of non-suspects that mistakenly classified as suspects). Table 6.3 shows the results. We can see that the number of false positives are maximum 1 out of 16 (6.25%). On average (across the trials) the percentage of the false positives is 2.375%.

Test Number	Percentage of False Positives
1	0
2	0
3	0
4	0
5	6.25%
6	6.25%
7	0
8	0
9	6.25%
10	5%

Table 6.1: Percentage of false positives in AVNET data.

The next two experiments evaluate the recognition of anomalous behavior using the RAFAEL data-set. In the first experiment, we were given data set consisted of 18 trajectories (432 single points). We learned on this data a model, with grid size of 31 and position overlap of 10 (as in the first experiment). We tested it against a single trajectory with u-turn pattern. Figure 6.16 shows all the 18 trajectories, the turn pattern which was found as suspicious marked in bold by the recognition system. The arrows point on the start position and the turn position.

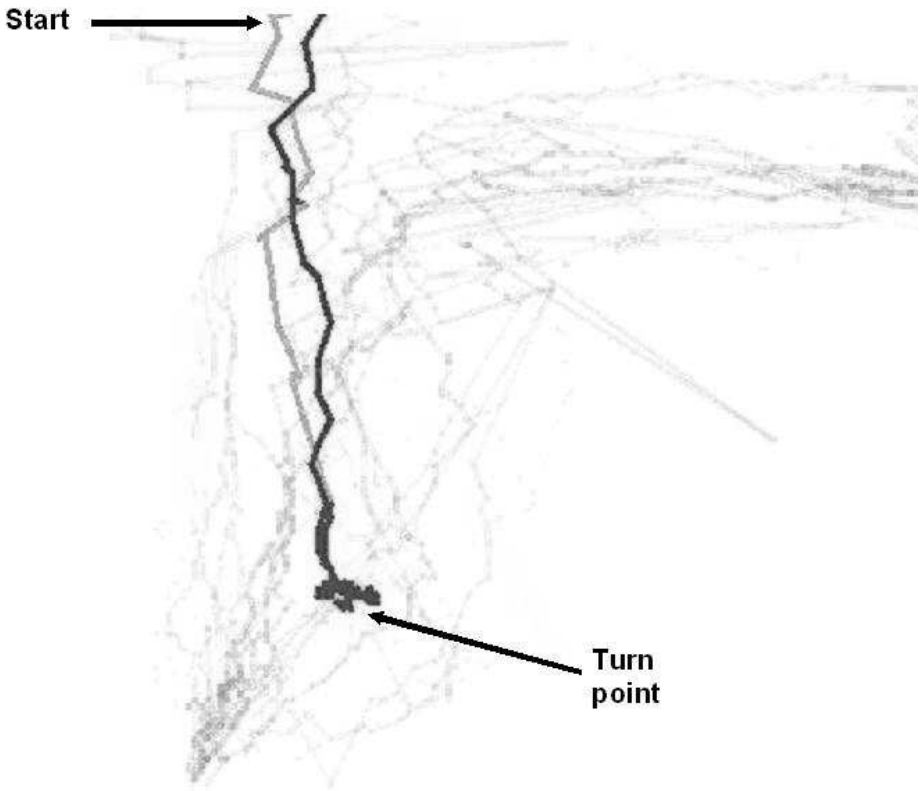

Figure 6.16: Detecting U-Turn on AVNET data.

In the second experiment of evaluating anomalous behavior, we were given data set consisted of 151 trajectories (4908 single points). We learned on this data a model, with grid size of 31 and position overlap of 10 (as in the first experiment). We tested it against a single trajectory of standing in place for a long duration. Figure 6.17 shows all the 151 trajectories, the trajectory that was detected as suspicious by the recognition system marked is in bold. The arrows point at the start position and the standing position.

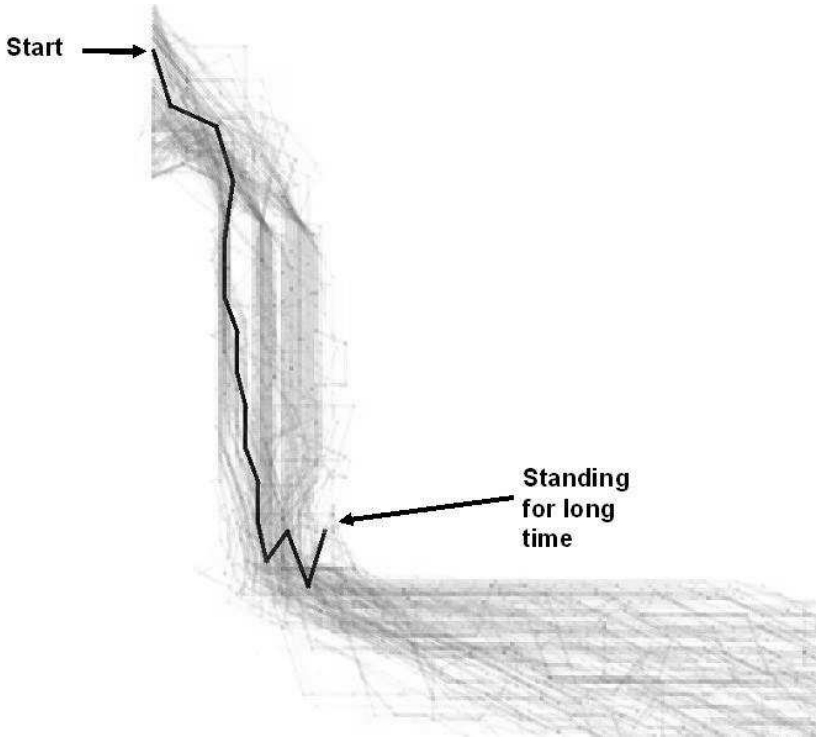

Figure 6.17: Detecting standing for long time on AVNET data

Chapter 7

Detecting Suspicious Behavior

To demonstrate the novel use of UPR, and its efficient implementation, as described in Chapter 4, we use hybrid anomalous and suspicious behavior recognition system. Figure 7.1 presents the hybrid system (this is an extension to the anomalous system showed in Chapter 6). The input to the system is an observation sequence. The system is composed from two modules: SBR (Symbolic Plan Recognition) and UPR (Utility Based Plan Recognition). The SBR module extracts coherent hypotheses from the observation sequence. If the set of hypotheses is empty, it declares the observation sequence as anomalous. If the set is not empty, then the hypotheses are passed to the UPR module that compute the most expected-costly hypotheses. When the expected cost of the top-ranked hypothesis reaches a given threshold, the system declares that the observation sequence is suspicious.

We tested the capabilities of our system in three different recognition tasks. The domain for the first task consisted of recognizing passengers that leave articles unattended, as in the example above. In the second task we will show how our algorithms can catch a dangerous driver that cuts between two lanes repeatedly. The last experiment intends to show how previous work, which has used costs heuristically [87], can now be recast in a principled manner. All of these examples show that we should not ignore the observer biases, since the most probable hypothesis sometimes mask hypotheses that are important for the observer.

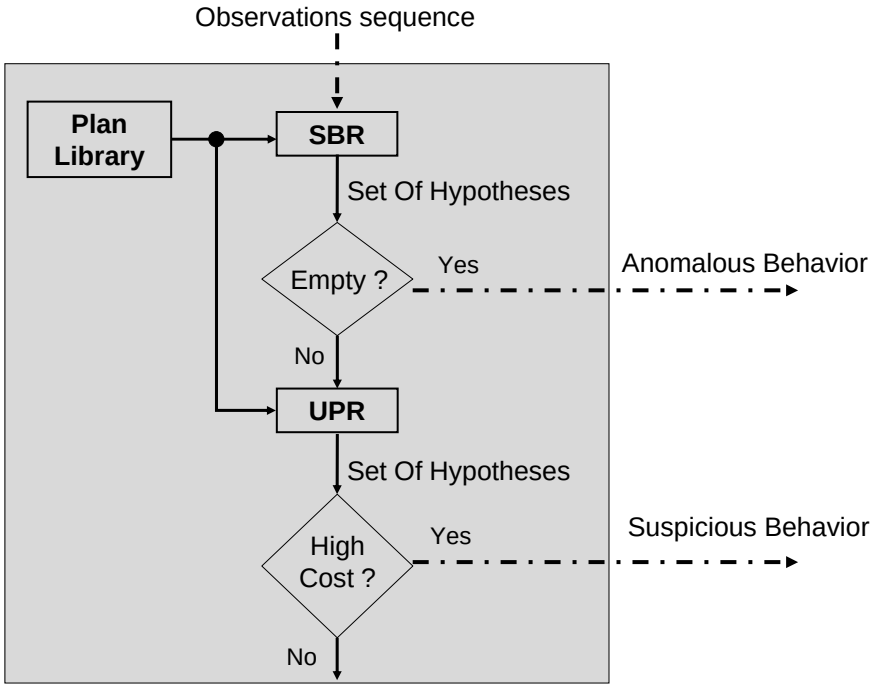

Figure 7.1: Suspicious Recognition System. Inputs and outputs for the system are in dashed arrows.

7.1 Leaving Unattended Articles

It is important to track a person that leaves her articles unattended in the airport. It is difficult, if not impossible, to catch this behavior using only probabilistic information. We examine the instantaneous recognition of costly hypotheses.

We demonstrate the process using the plan library in Figure 4.4. This plan library is used to track simulated passengers in an airport that walk about carrying articles, which they may put down and pick up again. The recognizer's task is to recognize passengers that put something down, and then continue to walk without it. Note that the task is difficult because the plan-steps are hidden (e.g., we see a passenger bending, but cannot decide whether it pick something up, put something down, or neither; we cannot decide whether a person has an article when they walk).

For the purposes of a short example, suppose that in time $t = 2$ (Figure 4.4), the SBR had returned that the two plan-steps marked *walk* match the observations (*walkN* means walking with no article, *walkW* signifies walking with an article); in time $t = 3$ the two *stop* plan steps match (*stopN* and *stopW*), and in time $t = 4$ the plan step *pickN* and plan step *putW*, match (e.g., we saw that the observed agent was bending).

The probability in $t = 4$ will be $P(putW|stopW) = 0.5 \times 0.2 = 0.1$ (the probability of *stopW* in previous time-stamp is 0.5, then following sequential link to *putW*), and in the same way $P(pickN|stopN) = 0.5 \times 0.3 = 0.15$. Normalizing the probabilities for the current time $t = 4$, $P(putW|stopW) = 0.4$ and $P(pickN|stopN) = 0.6$. The expected utility in time $t = 4$ is $U(putW|stopW) = P(putW|stopW) \times E(putW|stopW) = 0.4 \times 10 = 4$. The expected utility of *pickN* is zero. The expected costs, rather than likelihoods, raise suspicions of a passenger putting down an article (perhaps not picking it up).

Let us examine a more detailed example. We generated the following observations based on the plan library shown in Figure 4.4: Suppose that in time stamps $t = \{1 - 5\}$ the passenger walks in an airport, but we cannot tell whether she has an dangerous article in her possession. In time-stamps $t = \{6-7\}$ she stops, then at time $t = \{8\}$ we see her bending but can not tell whether to put down or to pick up something. In time-stamps $t = \{10-12\}$, she walks again.

Figure 7.2 shows the results from the recognition process for these observations. The X-axis measures the sequence of observations in time. The

probability of different leaves (corresponding to hypotheses) is shown on the Y-axis in the upper graph. The expected costs are shown in the lower graph. In both, the top-ranking hypothesis (after each observation), is the one whose value on the Y-axis is maximal for the observation.

In the probabilistic version (upper graph), we can see that the probabilities, in time $t = \{1 - 5\}$, are 0.5 since we have two possible hypotheses of walking. with or without an article ($walkW$ and $walkN$). Later when the person stops there are again two hypotheses $stopW$ and $stopN$. Then, in $t = \{7\}$ two plan steps match the observations: $pickW$ and $putN$, where the prior probability of $pickN$ is greater than $putN$ (after all, most passengers do not leave items unattended). As a result, the most likely hypothesis for the remainder of the sequence is that the passenger is currently walking with her article in hand $walkW$.

In the lower graph we can see a plot of the hypotheses, ranked by expected cost. At time $t = 8$ when the agent pick or put something, the cost is high (equal to 5), then in time stamp $t = \{9 - 12\}$ the top-ranking hypothesis is $walkN$, signifying that the passenger might have left an article unattended. Note that the prior probabilities on the behavior of the passenger have not changed. What is different here is the importance (cost) we attribute to observed actions.

7.2 Catching a Dangerous Driver

Some behavior becomes increasingly costly, or increasingly gainful, if repeated. For example, a driver switching a lane once or twice is not necessarily acting suspiciously. But a driver zigzagging across two lanes is dangerous. We demonstrate here the ability to accumulate costs of the most costly hypotheses, in order to capture behavior whose expected costs are prohibitive *over time*.

Figure 7.2 shows two lanes left and right in a continuous area, divided by a gird. There are 2 straight trajectories and one zigzag trajectory from left to right lane. From each position, the driver can begin moving to the next cell in the row (straight), or to one of the diagonal cells. We emphasize that the area and movements are continuous—the grid is only used to create a discrete state-space for the plan library. Moreover, the state-space is hidden: A car in the left lane may be mistakenly observed (with small probability) to be in the right lane, and vice versa.

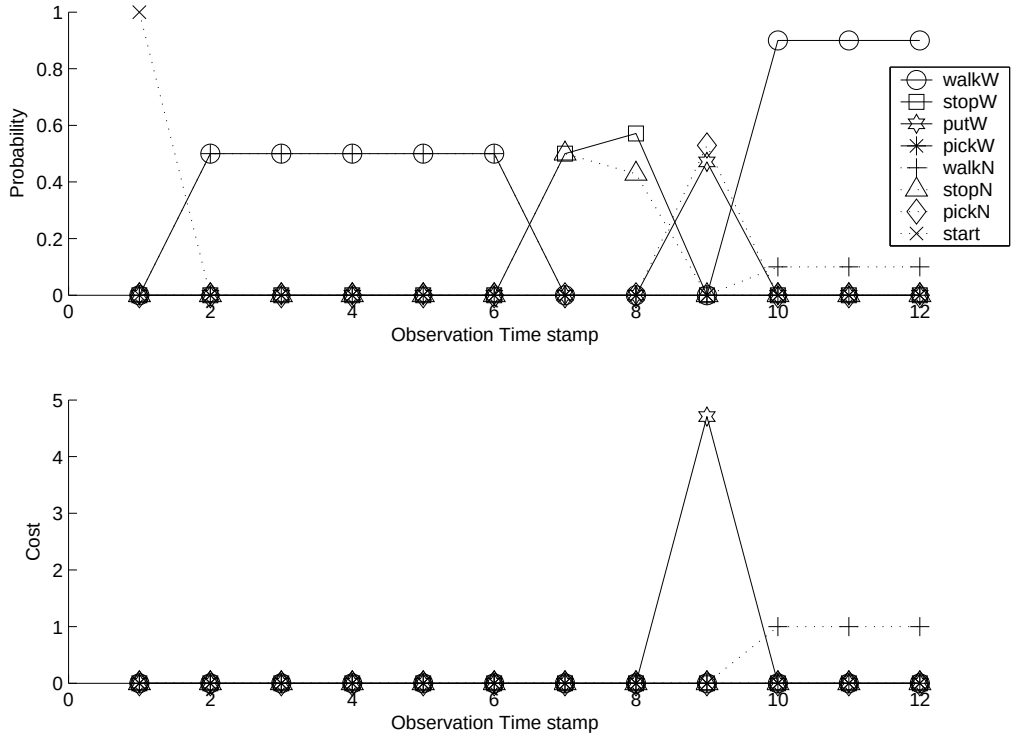

Figure 7.2: Leaving unattended articles: Probabilities and Costs

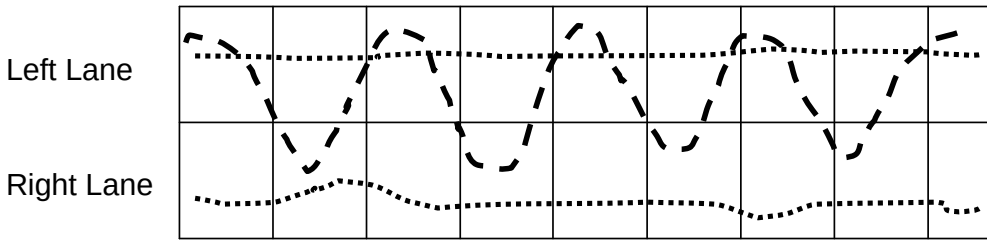

Figure 7.3: Simulated trajectories for drivers.

Each grid-cell is a plan-step in the plan library. The associated probabilities and utilities are as follows: The probability for remaining in a plan-step (for all nodes) is 0.4. The probability of continuing in the same lane is 0.4. The probability of moving to either diagonal is 0.2. All costs are zero, except when moving diagonally, where the cost is 10. Observations are uncertain; with 0.1 probability, an observation would incorrectly report on the driver being in a given lane.

We generated 100 observation sequences (each of 20 observations) of a zigzagging driver, and 100 sequences of a safe driver. The observations were sampled (with noise) from the trajectories (i.e., with observation uncertainty). For each sequence of observations we accumulated the cost of the most costly hypothesis, along the 20 observations. We now have 100 samples of the accumulated costs for a dangerous driver, and 100 samples of the costs for a safe driver. Depending on a chosen threshold value, a safe driver may be declared dangerous (if its accumulated cost is greater than the threshold), and a dangerous driver might be declared safe (if its accumulated cost is smaller than the threshold).

Figure 7.4 shows the confusion error rate as a function of the threshold. The error rate measures the percentage of cases (out of 100) incorrectly identified. The figure shows that a trade-off exists in setting the threshold, in order to improve accuracy. Choosing a cost threshold at 50 will result in high accuracy, in this particular case: All dangerous drivers will be identified as dangerous, and yet 99 percent of safe drivers will be correctly identified as safe.

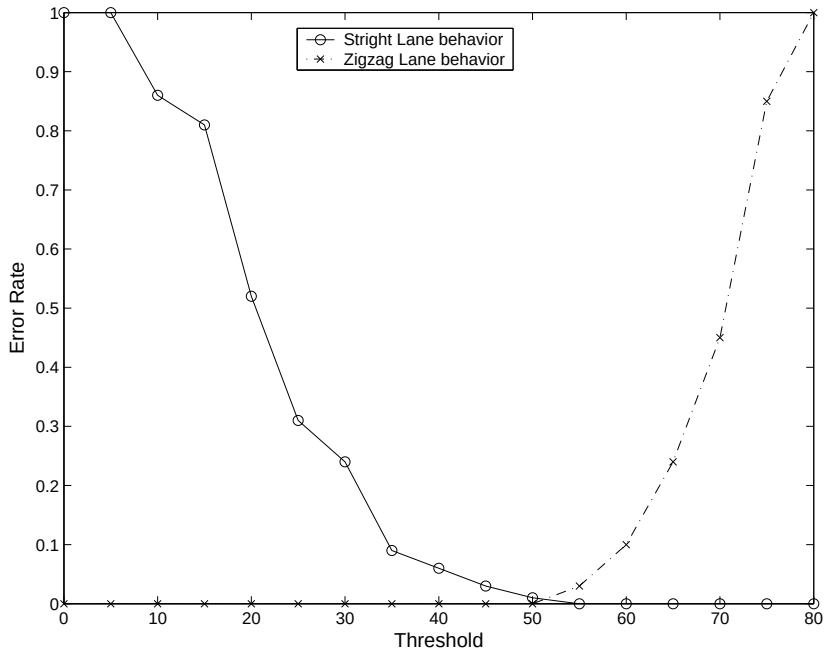

Figure 7.4: Confusion error rates for different thresholds for dangerous and safe drivers.

7.3 Air-Combat Environment

Tambe and Rosenbloom [87] used an example of agents in a simulated air-combat environment to demonstrate the RESC plan recognition algorithm. RESC heuristically prefers a single worst-case hypothesis, since an opponent is likely to engage in the most harmful maneuver in an hostile environment. The example used was of a air-combat maneuver, in [87] showed this heuristic in action in a simulated air-combat, where the turning actions of the opponent could be interpreted as either leading to it running away, or to its shooting a missile. RESC prefers the hypothesis that the opponent is shooting. However, unlike UPR, RESC will *always* prefer this hypothesis, regardless of its likelihood, and this has proven problematic [87]. Moreover, given several worst-case hypotheses, RESC will choose arbitrarily a single hypothesis to commit to, again regardless of its likelihood. Additional heuristics were therefore devised to control RESC's worst-case strategy [87].

We generalize this example to show UPR subsumes RESC's heuristic in a principled manner. Suppose instead of shooting a missile (which has infinite cost) vs. running away, we consider hypotheses of *invading air-space* vs. *runaway*, where invading the observer's air-space is costly for it, but not fatal. Figure 7.5 shows models of two types of opponents: An aggressive opponent (left sub-figure) that is more likely to shoot (0.8 a-priori probability) than to run away (0.2) , and a cowardly opponent (right sub-figure) that is more likely to run away (complement likelihoods). Note that these models are structurally the same; the assigned probabilities reflect the a-priori preferences of the different opponent types. Thus an observation matching both hypotheses will simply lead to both of them being possible, with different likelihoods. The maximum posterior hypothesis in the aggressive case will be that the opponent is trying to invade our airspace. In the cowardly case, it would be that the opponent is running away. RESC's heuristic would lead it to always selecting the aggressive case, regardless of the likelihood.

In contrast, UPR incorporates the biases of an *observing* pilot much more cleanly. Because it takes the likelihood of hypotheses into account in computing the expected cost, it can ignore sufficiently improbable (but still possible) worst-case hypotheses, in a principled manner. Moreover, UPR also allows modeling optimistic observers, who prefer best-case hypotheses.

Table 7.1 presents three cost models. In the first case, the *runaway* plan-step will get zero cost, and *invade* a high cost (10). This is an observer who is worried that its airspace being invaded, but not gaining anything from

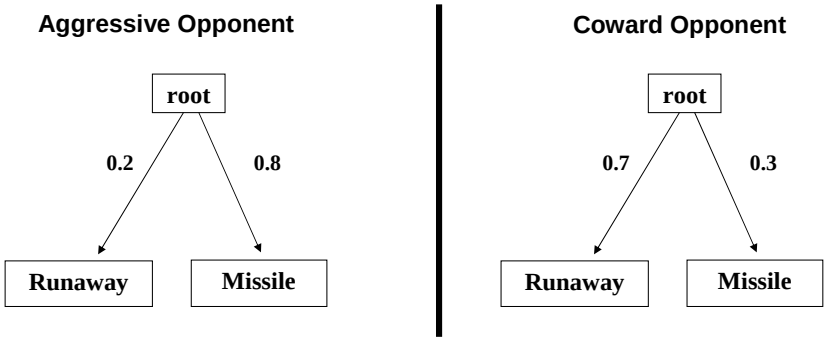

Figure 7.5: Air-Combat Environment. Two types of opponents.

	Runaway	Missile
Case A	0	10
Case B	−10	10
Case C	10	10

Table 7.1: Three cases of utilities for Figure 7.5.

scaring the opponent away. In the second case the *runaway* plan-step will get negative cost (i.e., a gain for the observer). In the third case there are the same costs. Tables 7.2 and 7.3 show the recognition results. The first row shows the results of following only the probabilistic reasoning in each model. The next three rows show the hypothesis costs for each hypothesis, in each of the three cases in Table 7.1.

In the cases of the aggressive opponent, both the most costly or the most probable hypothesis is the *invade* hypothesis. However, in the cowardly opponent case, the answer depends on the utility model. In cases A and B, where we gave high cost for missile, the most probable hypothesis stays runaway but the costly hypothesis is missile. In the third case, C, since we gave neutral costs (same for the two plan-steps), we got a result as in the probability model, meaning *runaway*. The conclusion is that the probabilistic model is not enough in case we want to incorporate biases of the observer, in this case that the missile plan-step is harmful for the observer.

This generalization of the original example in [87] demonstrates that the heuristic worst-case preference of RESC is subsumed by the principled use

	Runaway	Missile
Probabilistic	0.2	**0.8**
Cost A	0	**8**
Cost B	−2	**8**
Cost C	2	**8**

Table 7.2: Aggressive opponent: the result utilities for Figure 7.5.

	Runaway	Missile
Probabilistic	0.3	**0.7**
Cost A	0	**3**
Cost B	−7	**3**
Cost C	7	**3**

Table 7.3: Coward opponent: the result utilities for figure 7.5.

of decision-theoretic reasoning in our algorithms. And the complexity analysis in earlier sections shows that such reasoning does not necessarily entail significant computational costs. RESC's run-time complexity is linear in the plan-library. UPR's is polynomial.

Chapter 8

Future Directions and Final Remarks

We summarize the key contributions of this dissertation in Section 8.1. We discuss future directions for this research in Section 8.2.

8.1 Summary of Key Contributions

In the first part of this dissertation we concentrate on efficient hybrid plan recognition algorithms. Our contributions in this part of the work are as follows.

- We presented an efficient symbolic plan recognizer (Chapter 3) that is capable of handling open challenges in plan recognition. This recognizer adds significant capabilities to SBR, a previously-published symbolic plan recognizer [4, 5].
- We presented Utility based Plan Recognition (UPR), a general novel model of plan-recognition based on influence diagram. UPR allows the observer to incorporate her own biases and preferences—in the form of a utility function—into the plan recognition process (Chapter 4). We presented a hybrid algorithm which sacrifices some UPR expressivity, but gains much in execution time.
- We presented initial steps towards efficient dynamic tracking of the organization of multi-agent teams (Chapter 5). This is done by using a

combination of single-agent symbolic plan recognizer, and the DHGM data-structure, we maintain book-keeping information which allows us to dynamically track and hypothesize as to the organizational structure of a group of agents. This allows recognition based on interactions between agents.

In the second part of the dissertation, we considered the domain of detecting anomalous and suspicious behavior. We use the efficient plan recognition algorithms described in Part I to create efficient models for anomalous and suspicious behavior. A summary of this chapter's contributions are as follows.

- We present an anomalous behavior recognition model (chapter 6) using SBR, where the plan library represents normal behavior; any activity which is not matching the plan library is abnormal. We evaluate its use in two sets of experiments with trajectories created by machine vision systems.
- We present a suspicious behavior recognition model (Chapter 7) using UPR, where the plan library explicitly represents suspicious behavior; any activity that matches the model will be assumed to be suspicious. We demonstrate the capabilities of the suspicious behavior recognition model in three different domains: (a) Recognizing passengers that leave articles unattended; (b) Catching a dangerous driver that cuts between two lanes repeatedly; and (c) Air-Combat Environment.

8.2 Future Directions

In the context of plan recognition in multi-agent settings, much remains for future work. There are various points we would like to investigate further.

Formally define multi-agent plan recognition queries: In our work, we showed how to detect dynamic splitting and merging of groups with no reliance on static information on the groups. We did not give formal definitions as to the possible queries answerable by multi-agent plan recognizers in general, and DHGM in particular.

Apply UPR to multi-agent plan recognition: In our work, we relied on the symbolic plan recognition model to find groups that execute the same plan-step. This model does not address uncertainty in plan recognition hypotheses, nor incorporating a utility of the observer in the model. Future work should consider incorporating observer biases in the multi-agent plan recognition by for example adding costs to DHGM for splitting, etc.

Handle teams that execute different set of actions: The presented work on multi-agent plan recognition assumes that every member of a group executes the same or similar actions, as part of their team effort. Therefore, this approach can not be applied to teams of agents that are cooperating to achieve a common goal by each executing a different set of actions (such as a sports team). Future work should consider this issue.

Complete treatment for queue-cutting problem: In this work we introduced the queue cutting problem, where two friends are standing in specific position in the security line. One of them goes to the restrooms. When she returns, she joins her friend in the security line, rather than at the end of the line. Future work should consider providing a clear and complete treatment of this problem in the context of this model.

Bibliography

[1] EC funded CAVIAR project/IST 2001 37540. Found at URL: http://homepages.inf.ed.ac.uk/rbf/CAVIAR/.

[2] J. A. Adams. *Human Management of a Hierarchical System for the Control of Multiple Mobile Robots*. PhD thesis, University of Pennsylvania, 1995.

[3] D. Albrecht, I. Zukerman, and A. Nicholson. Bayesian models for keyhole plan recognition in adventure game. *User Modeling and User-Adapted Interaction*, 8(1–2):5–47, 1997.

[4] D. Avrahami. Symbolic behavior recognition. Master's thesis, Bar Ilan University, 2004.

[5] D. Avrahami-Zilberbrand and G. A. Kaminka. Fast and complete symbolic plan recognition. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-05)*, 2005.

[6] D. Avrahami-Zilberbrand and G. A. Kaminka. Hybrid symbolic-probabilistic plan recognizer: Initial steps. In *Proceedings of the AAAI Workshop on Modeling Others from Observations (MOO-06)*, 2006.

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

[8] D. Avrahami-Zilberbrand and G. A. Kaminka. Towards dynamic tracking of multi-agents teams: An initial report. In *Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR-07)*, 2007.

[9] D. Avrahami-Zilberbrand and G. A. Kaminka. Utility-based plan recognition: An extended abstract (short paper). In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-07)*, 2007.

[10] D. Avrahami-Zilberbrand and G. A. Kaminka. Fast symbolic plan recognition. *Submitted to Artificial Intelligence (AIJ)*, 2009.

[11] D. Avrahami-Zilberbrand and G. A. Kaminka. Incorporating observer biases in keyhole plan recognition (efficiently!). *Submitted to Artificial Intelligence (AIJ)*, 2009.

[12] D. Avrahami-Zilberbrand, G. A. Kaminka, and H. Zarosim. Fast and complete plan recognition: Allowing for duration, interleaved execution, and lossy observations. In *Proceedings of the IJCAI Workshop on Modeling Others from Observations (MOO-05)*, 2005.

[13] M. Bauer. A dempster-shafer approach to modeling agent preferences for plan recognition. *User Modeling and User-Adapted Interaction*, 5(4):317–348, 1995.

[14] D. S. Bernstein, E. A. Hansen, and S. Zilberstein. Bounded policy iteration for decentralized pomdps. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1287–1292, 2005.

[15] N. Blaylock and J. Allen. Fast hierarchical goal schema recognition. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI-06)*, pages 796–801, 2006.

[16] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In *Proc. Fourteenth Annual Conference on Uncertainty in AI (UAI)*, pages 33–42, 1998.

[17] M. Brand, N. Oliver, and A. Pentland. Coupled hidden markov models for complex action recognition. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, page 994, Washington, DC, USA, 1997. IEEE Computer Society.

[18] H. Bui. A general model for online probabilistic plan recognition. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-03)*, 2003.

[19] H. Bui, D. Phung, S. Venkatesh, and H. Phan. The hidden permutation model and location-based activity recognition. In *Proceedings of Twenty-Third National Conference on Artificial Intelligence (AAAI-08)*, 2008.

[20] H. Bui, S. Venkatesh, and G. West. Policy recognition in the abstract hidden markov models. *Journal of Artificial Intelligence Research*, 17:451–499, 2002.

[21] S. Carberry. Incorporating default inferences into plan recognition. In *Proceedings of the Eigth National Conference on Artificial Intelligence (AAAI-90)*, pages 471–478, 1990.

[22] S. Carrbery. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11:31–48, 2001.

[23] E. Charniak and R. P. Goldman. A Bayesian model of plan recognition. *Artificial Intelligence*, 64(1):53–79, Nov. 1993.

[24] G. F. Cooper. The computational complexity of probabilistic inference using bayesian belief networks (research note). *Artificial Intelligence*, 42(2-3):393–405, 1990.

[25] F. Cupillard, Alberto.Avanzi, F. Bremond, and M. Thonnat. Video understanding for metro surveillance, 2004.

[26] R. J. Doyle. Determining the loci of anomalies using minimal causal models. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1821–1827, Montreal, Quebec, Canada, 1995.

[27] T. Duong, H. Bui, D. Phung, and S. Venkatesh. Activity recognition and abnormality detection with the switching hidden semi-markov models. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR-2005)*, San Diego, CA, June 2005.

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

[29] S. Fine, Y. Singer, and N. Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

[30] R. J. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, 1987.

[31] C. W. Geib. Assessing the complexity of plan recognition. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, 2004.

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

[33] C. W. Geib and S. A. Harp. Empirical analysis of a probalistic task tracking algorithm. In *AAMAS workshop on Modeling Other agents from Observations (MOO-04)*, 2004.

[34] Z. Ghahramani and M. I. Jordan. Factorial hidden Markov models. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Proc. Conf. Advances in Neural Information Processing Systems, NIPS*, volume 8, pages 472–478. MIT Press, 1995.

[35] Z. Ghahramani and M. I. Jordan. Factorial hidden Markov models. *Machine Learning*, 29:245–275, 1997.

[36] P. J. Gmytrasiewicz and E. Durfee. A rigious operational formalization of recursive modeling. In *IJCAI '95: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 492–499, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[37] K. Han and M. Veloso. Automated robot behavior recognition applied to robotic soccer. In *Proceedings of the IJCAI-99 Workshop on Team Behavior and Plan-Recognition*, 1999. Also appears in Proceedings of the 9th International Symposium of Robotics Research (ISSR-99).

[38] B. Hilary and G. Shaogang. Advanced visual surveillance using bayesian networks. In *International Conference on Computer Vision*, June 1995.

[39] S. Hongeng, F. Brémond, and R. Nevatia. Bayesian framework for video surveillance application. In *15th International Conference on Pattern Recognition (ICPR'00)*, volume 1, pages 164–170, 2000.

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

[41] R. Howard and J. Matheson. Influence diagrams. In R. Howard and J. Matheson, editors, *Readings on the Principles and Applications of Decision Analysis*. Strategic Decisions Group, 1984.

[42] D. H. Hu and Q. Yang. CIGAR: Concurrent and interleaving goal and activity recognition. In *Proceedings of Twenty-Third National Conference on Artificial Intelligence (AAAI-08)*, pages 1363–1368, 2008.

[43] M. J. Huber, E. H. Durfee, and M. P. Wellman. The automated mapping of plans for plan recognition. In *AAAI'94: Proceedings of the twelfth national conference on Artificial intelligence (vol. 2)*, page 1460, Menlo Park, CA, USA, 1994. American Association for Artificial Intelligence.

[44] S. S. Intille and A. F. Bobick. A framework for recognizing multi-agent action from visual evidence. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 518–525. AAAI Press, July 1999.

[45] P. Jarvis, T. Lunt, and K. Myers. Identifying terrorist activity with ai plan recognition technology. In *The Sixteenth Innovative Applications of Artificial Intelligence Conference (IAAI 04)*, 2004.

[46] G. A. Kaminka and M. Bowling. Towards robust teams with many agents. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, 2002.

[47] G. A. Kaminka, E. Merdler, and D. Avrahami. Advanced unsupervised spatial learning algorithm for the avnet37 consortium: Final report (in hebrew). Technical Report MAVERICK 2006/01, Bar Ilan University, Computer Science Department, MAVERICK Group, 2006.

[48] G. A. Kaminka, E. Merdler, and D. Avrahami. Advanced unsupervised spatial learning algorithm for the avnet37 consortium: Interim report (in

hebrew). Technical Report MAVERICK 2006/01, Bar Ilan University, Computer Science Department, MAVERICK Group, 2006.

[49] G. A. Kaminka, D. V. Pynadath, and M. Tambe. Monitoring teams by overhearing: A multi-agent plan recognition approach. *Journal of Artificial Intelligence Research*, 17:83–135, 2002.

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

[51] H. A. Kautz and J. F. Allen. Generalized plan recognition. In *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, pages 32–37. AAAI press, 1986.

[52] U. Kjærulff. A computational scheme for reasoning in dynamic probabilistic networks. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI-1992)*, pages 121–129, San Mateo, CA, 1992. Morgan Kaufmann.

[53] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. *Games and Economic Behavior*, 45(1):181–221, 2003.

[54] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. *ACM Transactions on Information and System Security*, 2(3):295–331, Aug 1999.

[55] L. Liao, D. Fox, and H. A. Kautz. Learning and inferring transportation routines. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*, pages 348–353, 2004.

[56] D. Mahajan, N. Kwatra, S. Jain, P. Kalra, and S. Banerjee. A framework for activity recognition and detection of unusual activities. In *ICVGIP*, pages 15–21, 2004.

[57] W. Mao and J. Gratch. Decision-theoretic approaches to plan recognition. In *USC/ICT Technical Report*, 2004.

[58] W. Mao and J. Gratch. A utility-based approach to intention recognition. In *Proceedings of the AAMAS Workshop on Modeling Other Agents from Observations (MOO-04)*, NY City, NY, USA, July 2004.

[59] E. Marhasev, M. Hadad, G. A. Kaminka, and U. Feintuch. The use of hidden semi-markov models in clinical diagnosis maze tasks. *Intelligent Data Analysis*, 13(6):To Appear, 2009.

[60] M. J. Mataric. *Interaction and Intelligent Behavior*. PhD thesis, Massachusetts Institute of Technology, 1994.

[61] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[62] K. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, 2002.

[63] A. Newell. *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts, 1990.

[64] N. Nguyen, D. Phung, S. Venkatesh, and H. Bui. Learning and detecting activities from movement trajectories using the hierarchical hidden markov model. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2005.

[65] M. Nicolescu and M. J. Mataric. A hierarchical architecture for behavior-based robots. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-02)*, pages 227–233, Bologna, Italy, July 15–19 2002.

[66] W. Niu, J. Long, D. Han, and Y.-F. Wang. Human activity detection and recognition for video surveillance. In *Proceedings of the IEEE Multimedia and Expo Conference*, pages 719–722, 2004.

[67] S. Noh and P. Gmytrasiewicz. Flexible multi-agent decision-making under time pressure. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 35(5):697–707, 2005.

[68] P. E. of a Vision Based Lane Tracker Designed for Driver Assistance Systems. Ajoel c. mccall and mohan m. trivedi. *IEEE Intellient Vehicles Symposium*, pages 153–158, 2005.

[69] N. Oliver, E. Horvitz, and A. Garg. Layered representations for human activity recognition. *In Fourth IEEE International Conference on Multimodal Interfaces*, 8:831–843, 2002.

[70] D. Q. Phung, T. V. Duong, S. Venkatesh, and H. H. Bui. Topic transition detection using hierarchical hidden markov and semi-markov models. In *ACM Multimedia*, pages 11–20, 2005.

[71] D. V. Pynadath and S. Marsella. Psychsim: Modeling theory of mind with decision-theoretic agents. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 1181–1186, 2005.

[72] D. V. Pynadath and S. Marsella. Minimal mental models. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI-07)*, pages 1038–1044, 2007.

[73] D. V. Pynadath and M. P. Wellman. Generalized queries on probabilistic context-free grammars. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):65–77, 1998.

[74] D. V. Pynadath and M. P. Wellman. Probabilistic state-dependent grammars for plan recognition. In *Proceedings of the 16th Annual Conference on Uncertainty in Artificial Intelligence*, pages 507–514, 2000.

[75] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb. 1989.

[76] B. Rathnasabapathy, P. Doshi, and P. J. Gmytrasiewicz. Exact solutions of interactive pomdps using behavioral equivalence. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-06)*, pages 1025–1032, 2006.

[77] G. Retz-Schmidt. Recognizing intentions, interactions, and causes of plan failures. *User Modeling and User-Adapted Interaction*, 2:173–202, 1991.

[78] Q. J. Ross. *C4.5 Programs for machine learning.* Morgan Kaufmann Publishers,Inc, 1992.

[79] S. Russel and P. Norvig. *Artificial Intelligence, a Modern Approach.* Prentice Hall, 1995.

[80] T. Starner and A. Pentland. Real-time american sign language recognition from video using hidden markov models. In *In proceedings of SCV95*, 1995.

[81] G. Sukthankar and K. Sycara. A cost minimization approach to human behavior recognition. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05)*, 2005.

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

[83] D. Suryadi and P. J. Gmytrasiewicz. Learning models of other agents using influence diagrams. In *UM '99: Proceedings of the seventh international conference on User modeling*, pages 223–232, Secaucus, NJ, USA, 1999. Springer-Verlag New York, Inc.

[84] R. Suzic. A generic model of tactical plan recognition for threat assesment. In B. V. Dasarathy, editor, *Proceedings of SPIE Multisensor*, volume 5813, pages 105–116, March 2005.

[85] M. Tambe. Tracking dynamic team activity. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, August 1996.

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

[87] M. Tambe and P. S. Rosenbloom. RESC: An approach to agent tracking in a real-time, dynamic environment. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-95)*, August 1995.

[88] D. V. Vail, M. M. Veloso, and J. D. Lafferty. Conditional random fields for activity recognition. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-07)*, pages 1331–1338, 2007.

[89] P. Varakantham, R. T. Maheswaran, and M. Tambe. Exploiting belief bounds: Practical pomdps for personal assistant agents. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-05)*, pages 978–985, 2005.

[90] Wikipedia.

[91] G. Wu, Y. Wu, L. Jiao, Y.-F. Wang, and E. Y. Chang. Multi-camera spatio-temporal fusion and biased sequence-data learning for security surveillance. In *MULTIMEDIA '03: Proceedings of the eleventh ACM international conference on Multimedia*, pages 528–538, New York, NY, USA, 2003. ACM Press.

[92] T. Xiang and S. Gong. On the structure of dynamic bayesian networks for complex scene modelling. In *Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, October 2003.

[93] T. Xiang and S. Gong. Video behavior profiling for anomaly detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):893–908, 2008.

[94] J. Yamato, J. Ohya, and K. Ishii. Recognizing human action in time-sequential images using hidden markov model. In *In Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR-92)*, pages 379–385, 1992.

[95] J. Yin, Q. Yang, and J. J. Pan. Sensor-based abnormal human-activity detection. *IEEE Transactions on Knowledge and Data Engineering*, 20(8):1082–1090, 2008.