ABSTRACT

KUMARAN, VIKRAM. Plan Recognition as Candidate Space Search. (Under the direction of Michael Young).

Effective human computer interaction is enhanced by a machine’s ability to make educated guesses about the intention of its user. In our research, we have developed a novel plan recognition algorithm – based on plan space search planners – to recognize plans given a limited set of observed actions. Our focus in this research is towards accurately picking possible plans and not towards disambiguation or building plan libraries and therefore we complement other advances in this field, namely probability based recognition and other plan library based recognition systems. Along with the ability to recognize overall goal of an agent our algorithm also allows us to make local predictions, a feature absent in most of the other system.
PLAN RECOGNITION AS CANDIDATE SPACE SEARCH

by

VIKRAM KUMARAN

A thesis submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Master of Science

COMPUTER SCIENCE

Raleigh
2006

APPROVED BY:

__________________________________________
Dr. Dennis R. Bahler

__________________________________________
Dr. James C. Lester

__________________________________________
Dr. R. Michael Young
BIOGRAPHY

Vikram Kumaran spent his early life in Chennai India. After completing his undergraduate degree in Chemical Engineering from the Indian Institute of Technology, Chennai, he left tropical India for the land of snow, Rochester, New York to pursue his graduate studies. He trained to be a Chemical Engineer and completed his MS in that field from the University of Rochester. He met his wife there and followed her to North Carolina. Having spent many years exploring computational models in Chemical Engineering, he decided to pursue, software engineering as his career path.

After stepping away from academics for a few years, he realized that applied computer science in the area of information technology did not satisfy his curiosity for the subject. To satisfy his yearning for something more he decided to pursue an academic degree in computer science. He joined the masters program in the department of computer science, North Carolina State University.

As the masters chapter of his life comes to an end he hopes to continue his involvement in fundamental research by pursuing interesting problems in AI and collaborating with his mentors and peers.
TABLE OF CONTENTS

LIST OF TABLES ........................................................................................................ iv

LIST OF FIGURES ...................................................................................................... v

1 INTRODUCTION ................................................................................................... 1

2 PLAN RECOGNITION AS CANDIDATE SPACE SEARCH ................................. 4
   2.1 Motivation ........................................................................................................ 4
   2.2 Models of Plan Recognition ........................................................................... 6

3 COMPUTATIONAL MODEL OF PLAN RECOGNITION ..................................... 17
   3.1 Definitions ...................................................................................................... 17
   3.2 Assumptions .................................................................................................. 19
   3.3 Algorithm ....................................................................................................... 20
   3.4 Assertions ...................................................................................................... 24
   3.5 Analysis .......................................................................................................... 27
   3.6 Implementation ............................................................................................. 28

4 EXAMPLES ........................................................................................................... 30
   4.1 Cooking Domain ............................................................................................ 30
   4.2 Vacation Domain ........................................................................................... 33

5 ANALYSIS OF THE RECOGNIZER ................................................................. 41
   5.1 Comparison with Peers ................................................................................ 41
   5.2 Applications of the Recognizer .................................................................... 43

6 CONCLUSION ........................................................................................................ 46

REFERENCES ........................................................................................................... 48
LIST OF TABLES

Table 4.1: Cooking Domain Recognizer Results .................................................. 33
Table 4.2: Vacation Domain Recognizer Results.................................................. 38
Table 4.3: Vacation Domain Recognizer Results.................................................. 38
LIST OF FIGURES

Figure 1.1 : Schematic Representation of Plan Recognition systems............................. 2
Figure 2.1 : Sample Event Hierarchy Kautz [1987] ....................................................... 11
Figure 4.1: The Cooking Domain Actions...................................................................... 30
Figure 4.2: The Cooking Domain Plan Library .............................................................. 31
Figure 4.3: The Planning Graph...................................................................................... 31
Figure 4.4: The Vacation Domain Actions ..................................................................... 35
Figure 4.5: The Vacation Domain Plan Library .............................................................. 36
Figure 4.6: The Vacation Domain Planning Graph A ...................................................... 37
Figure 4.7: The Vacation Domain Planning Graph B ...................................................... 39
Figure 4.8: The Vacation Domain Planning Graph C ...................................................... 39
1 INTRODUCTION

Intelligence is not just reaction to stimuli but the ability to think ahead and decide on a series of steps and contingencies to reach a desired goal. This view of intelligence might explain why “planning” has been an active area of research in AI for many years. Similar to this argument for intelligence one can also argue that meaningful interaction between intelligent beings occurs when they are able to discern the plans of one another. Human conversation would get very tedious if people had to explain every action and describe all their goals and intentions. Human communication relies on the communication of minimal but sufficient information enough to determine the intentions without the need for step by step explanations. For example, if you see a person getting down from a taxi, with luggage at an airport it is usually safe to assume that they are going to a airline counter, check-in, pass security, board a plane and fly somewhere, even though you did not see all those steps being performed. This process of determining an agent’s plan and goals can be classified as plan recognition.

Plan recognition is an important aspect of AI and has many different areas of application, namely, natural language understanding, human computer interaction, medical diagnosis, user modeling, tutoring systems, story recognition, interactive games and intrusion detection.

Plan recognition is usually classified into two broad categories with unique challenges associated with each. One type of plan recognition is an interactive process where the recognized and recognizer engage in dialog to resolve any ambiguity. This type of recognition is called intended recognition [Cohen et al., 1981]. Intended recognition is common in natural language dialog systems. On the other hand, there are applications where it is necessary to discern the intention of an agent solely based on the observed actions of the agent. This type of recognition is called keyhole recognition [Cohen et al., 1981]. In some applications the intention is to generate unsolicited recommendations with the goal of complementing or hindering the user’s efforts. In these types of systems, keyhole recognition plays an important role. This is especially true in tutoring systems, intrusion detection systems etc.
At the highest level almost all recognition systems can be schematically represented using a small set of clearly identified roles, as show in the Figure 1.1. Recognition starts with a library of possible plans that an agent would be expected to execute in a given domain. The observing agent then tracks one or more actions performed by the observed agent. The observer then runs its plan recognition algorithm (recognizer). Based on the observed actions, some domain dependent heuristics and plan library the recognizer predicts one or more intended goals and plans of the observed agent [Carberry, 2001]. Usually the plan library and heuristics is domain dependent while the recognizer itself is domain independent.

![Schematic Representation of Plan Recognition systems](image)

**Figure 1.1 : Schematic Representation of Plan Recognition systems**

Research in plan recognition has focused on improving each of the different aspects described in the Figure 1.1. Research done by Kautz [1987] was primarily focused on strengthening the reasoning aspects of plan recognition. He developed a reasoning framework based on circumscription to determine a minimal set of agent’s intended goals/plans that explained the observed actions given a completely specified hierarchical description of all possible plans in the domain. His work is usually recognized as the first
attempt at formalizing plan recognition. His formal theory of plan recognition has been
the basis for subsequent plan recognition research [Vilain, 1990; Alexandersson, 1995;
Wobcke, 2002]. While having a sound and efficient reasoning algorithm is essential for
successful recognition, in realistic problems, the challenge is usually the domain size and
ability to define plan libraries. In most real life domains plan libraries are very large and
it is almost impossible to create them manually. Researchers have developed case based
plan recognition systems that learn novel plans by observation. This learning algorithm
builds and evolves the plan library over time [Kerkez et al., 2001]. Given a massive plan
library, recognition algorithms need to efficiently access plans from the library to
perform effectively in real time situations. Some researchers have tried to design more
efficient data structures using their expertise in knowledge representation [Weida, 1995].
A sound, complete, and efficient reasoning algorithm or a well-defined and easily
accessible plan library, are not sufficient to build a perfect plan recognition system. One
of the biggest stumbling blocks in recognition is choosing between multiple, equally
possible plans. Most systems use some kind of heuristic to arrive at a single plan. The
heuristics might either be domain dependent or user dependent. Some recognizers choose
to deemphasize the importance of formal theory and emphasize heuristics based on rules
of thumb [Allen et al., 1980; Allen, 1983]. Another solution to this problem is the use of
probabilistic approaches like Bayesian networks and Dempster-Shafer theory [Albrecht et
al., 1998; Charniak et al., 1993].

In the work presented here, we have developed a sound reasoning algorithm that
identifies sets of plans that explain a sequence of observed actions. The algorithm forms
these solution plan-sets by searching through a space of candidate plans. Most of the
earlier work in this area has focused on determining the high-level intention or end goal
of the agent rather than identifying specific plans that they are being executed [Kautz,
1987; Lesh, 1998]. Our approach is focused on improving two aspects of plan recognition
over other approaches – firstly, going beyond the recognition of intentions and goals to
identify actual plans being executed by the user and secondly arriving at all possible
explanations for an observation using the tools created for efficient planning.
2 PLAN RECOGNITION AS CANDIDATE SPACE SEARCH

2.1 Motivation

There are many aspects to the problem of plan recognition. One issue inherent in many
plan recognition contexts is multiple equally possible solutions. When there are multiple
explanations for a set of observed actions the only means of disambiguating between
these choices is the use of domain specific knowledge. An advantage of recognizers that
rely on heuristics or uncertain reasoning lies in their ability to tackle the ambiguity that is
inherent in plan recognition. Heuristics are useful in choosing between plan sets that are
deemed as equally valid by formal analysis. However, using them as the sole means to
choose between all possible plans would not be as effective. This conclusion is based on
the fact that heuristics requires extensive domain knowledge that might not be always
available. This leads us to believe that formal methods have a significant role to play in
identifying the candidate plan-sets. We have tried to develop a framework that does not
exclude the use of abductive reasoning and domain specific heuristics to explain
observations; these techniques can easily be used to aid our algorithm with a resulting
hybrid system that may be stronger than its parts.

In our research we would like to tackle the following two questions

Can we leverage our understanding of planning algorithms to devise an efficient
plan recognizer?

Developing scalable and fast recognizers has been a challenge for the plan recognition
community since the time the problem was presented. Heuristic methods found a way to
be fast but they tend to use unsound reasoning techniques and their rules are usually
domain dependent [Kautz, 1987]. Kautz’s framework is the first attempt at using formal
methods and theory to describe the problem of plan recognition; he himself agrees that
his algorithms are not built for speed. Many subsequent attempts at modifying Kautz’s
framework had to do so at the expense of limiting either the type of plans that could be
recognized or the expressiveness of the representation [Vilain, 1990; Lesh, 1998]. Most
other attempts endeavor to solve the problem by devising optimal ways of representing and building plan libraries, with the focus on access speed and size. [Lesh, 1998; Kerkez et al., 2001] These approaches can coexist with our approach as these techniques only affects what the plan library contains and how it is built and not how the plan library is used.

Over the past few years one of the fastest planners for solving classical planning problems has been GRAPHPLAN, [Blum et al., 1997] created by Blum and Furst. Kambhampati [1997] in his analysis of GRAPHPLAN, considers GRAPHPLAN to be a member of the group of forward state-space planners. One of the main problems for planners that have to do forward state space search is that the search tree can grow large even for small problem sizes. According to Kambhampati the secret behind graph plan is that it uses mutex relations that block the propagation of illegal subsets of propositions as the search graph is built. Kambhampati speculates that, calculating and using 2-sized mutex relationship reduces the search space significantly, because it is not computationally expensive ($O(n^2)$) and action–action interaction is usually pair wise. Our plan recognition algorithm takes a similar approach. We attempt plan recognition by searching through candidate plan space. We believe that if you define and propagate mutex relationships as we expand the plan space we can bring about the same efficiency to plan recognition that graph plan brought to planning.

**How to develop a plan recognition algorithm that is effective in predicting the immediate future along with the overall intention of the agents?**

Most of the current techniques in plan recognition are mainly interested in recognizing the overall intention of the user. Kautz’s model [1987] tries to get at the “End” events from observations; Lesh’s algorithms [1998] are goal recognition algorithms and not plan recognition algorithms. Intention or goal recognition is a very useful aspect of plan recognition and is sufficient for several applications. However, in systems that do intrusion detection or game systems that want to proactively mediate user interaction, it is not enough if we know the user’s ultimate goal. It is just as important to guess how the
user is trying to reach that goal and respond accordingly. In such situations we still need to recognize global intentions, but to react to local context of the interaction we need to be able to predict the next action. We feel that our system that builds the search graph incrementally as each action is observed, and keeps track of possible plans after each action, will be able to perform well at both local and global recognition.

### 2.2 Models of Plan Recognition

The problem of plan recognition has been approached from many directions. Our strategy is focused on enhancing reasoning rather than efficient knowledge representation. Reasoning engines are good at efficiently pruning irrelevant plans from the plan library and returning a small set of plans that matches the observed actions. Researchers have made different choices on how they choose the best candidates from the available set of plans. One popular choice made by early plan recognition systems, was the use of common sense rules or heuristics to choose the correct plan in the library [Allen et al., 1980; Allen, 1983]. This approach is very successful, especially when dealing with small domains. As the area of research developed, formal methods were employed to determine the minimal set of plans that explains observations [Kautz, 1987; Vilain, 1990; Lesh, 1998]. Currently there is lot of work being done to identify agent plans based on probability [Albrecht et al., 1998; Charniak et al., 1993; Bauer, 1994]. In this section we will discuss important developments in these three areas of research. In subsequent sections we will explain our approach and compare it with the systems presented in this section.

#### 2.2.1 Heuristics Recognition Systems

One of the early attempts at plan recognition was made by Schmidt et al. [1978] who classified plan recognition as an intersection of psychology and artificial intelligence. They performed psychological tests on subjects to determine if humans use plan recognition in their day to day life. The subjects were given a detailed list of statements which was ordered as a sequence of events being observed by the subject. The subject was then asked to determine the goal of the actor in the statements, recall the set of actions or predict the next action. Based on their response, it was determined that the
subjects formed a small set of hypothesis to explain the actions and with each new action the hypothesis was either strengthened or rejected. The set of hypothesis was always small and subjects tended to form the hypothesis in a hierarchical manner forming more abstract hypothesis initially at first and filling in the details as more evidence was observed. Subjects were also more tolerant with hypothesis that did not exactly fit all observed actions, tending to explain away the offending actions.

Based on their experimental data Schmidt et al. developed a system called the BELIEVER. Schmidt’s system internally divided plan recognition into two parts one was knowledge and the other process. Knowledge was represented as three domains, the world, the person, and the plan domains. World domain represented facts about the world, person domain information regarding the agent’s belief, likes etc. and the plan domain information about plan structures and action relationships. This division into three was based on psychological models of human cognitive process. The plan recognition was done using a plan generator and plan critic. The plan generator was used to create plausible plans based primarily on the person model and the plan critic was used to weed out incorrect or improbable hypothesis.

The plan generator and plan critic worked on a hypothesis about the agent’s goals called the expectation structure to generate the appropriate plans. The system then tries to match observed actions on to acts in the expectation structure. The authors do not clearly explain how the initial hypothesis regarding the agent’s goals is made. This system also does not support interleaving plans. The main contribution from Schimdt et al. is not their recognizer but the experiments they did to show how humans use plan recognition as part of the communication process.

Plan recognition is an important aspect of natural language recognition and there has been a lot of work done in the area of plan recognition that was motivated by the need to build systems that are able to participate in a dialogue, with human users. Allen [1980; 1983] developed a system that converted user utterances into speech acts. Based on these acts the system reasoned about the user’s intentions. The end goal was to achieve natural language understanding. The system took single utterances like “What time does the Montreal train leave” and using inference rules and heuristics, it backward chained all the
way to the point where it could conclude about user plans. The heuristics were simple rules for example, if a person wants P, and P is a precondition of action A, then the person may want to perform A; or if a person wants to know if P is true, they may want P to be true (or false). This approach is not scalable to large domains as enumerating all the rules would become cumbersome. Also, the recognizer could only handle single utterances.

Carberry [1987; 1990] developed a system that would recognize multiple utterances. Carberry used a hierarchical structure called the context model. The context model represented the systems beliefs about the speakers intended plan and was built based on a dialogue in progress between the system and the speaker. The system did local analysis and based on some focusing heuristics chose a subplan that the user is most likely pursuing based on the current context. This subplan is then appended to the context model and the context model is expanded. The system then performs a global analysis to determine any higher-level plans that need to be included as a result of the current subplan being pursued. The context model is then expanded to include them. Each node in the context model is a goal that the information seeker is attempting to reach with an identified plan. Based on the context model the system is able to understand the dialogue at multiple levels and hence communicate effectively with the speaker.

Pollack [1990] criticized earlier plan recognition attempts for concentrating on specific recipes of action instead of the complex mental attitudes that represents a plan in an actors mind. The argument was that, by dealing with plan recognition at a higher level of abstraction from specific plans in the plan library it is possible to better serve/respond to the intention of the agent and develop a more robust recognizer and not get misled by incorrectly constructed plans, novel plans etc.

One of the primary motivations for early systems of plan recognition was to recognize human communication in the form of natural language dialogues and stories. Real time interactions involved in conversing with humans require recognizers to be fast rather than accurate. It is not necessary to know the next words of the speaker, but it is important to guess the intention or context of the conversation. These systems also have the luxury of engaging in clarification dialogues to resolve ambiguity. This need for speed encourages
these systems to engage in shortcuts, in terms of heuristic inference rules, to narrow the
search space. These inference rules however are not perfect and sacrifice soundness and
completeness to achieve speed. For example, in Allen’s system [1980; 1983] one of the
inference rules is that if an agent is checking about the validity of a statement P then he
would like P to be true. This however will fail in the case when an agent is just checking
the truthfulness of P because he is concerned that P might become true. The difference
can clearly be seen between the following two questions. a) Has the train left yet? b)
Was there an accident? One advantage of these systems is the emphasis on recognizing
the overall intention of the agent. This lets these systems work at a more abstract level
and hence a smaller search space. Intention or goal recognition compared to plan
recognition might be a good target for natural language dialogue systems but might not
be useful in other situations. For example in a tutoring situation, local recognition is
important as steps are as important if not more compared to overall intention or goal of
the student. Also all systems might not have the luxury of clarification dialogues,
intrusion detection systems being a good example.

2.2.2 Formal Recognition Methods

Initial attempts at plan recognition tended to rely on heuristics and domain dependent
inference rules to narrow the search space. Kautz [1987] made significant progress
formalizing the process of plan recognition. Kautz’s complaint about earlier attempts at
plan recognition was their basis on unsound rules of inference. The unsound inference
rules were created by reversing normally sound implications. If a particular plan entails
an action it is unsound to assume that the action “may” imply the plan [Allen et al.,
1980]. Kautz also argues that the decision about which rules to apply and when to stop
applying them was not based on formal theory. In contrast Katuz’s framework decides
on the agent’s plans solely on the observations, recognizer’s knowledge and a limited set
of explicit “closed world” assumptions. Kautz’s publication has provided the foundation
for new theories on plan recognition [Wobcke, 2002; Vilain, 1990].

In Kautz’s framework all actions are uniformly referred to as events. The recognizer
organizes all its knowledge in a hierarchical structure relating events to one another.
Events can be related to other events by either a specialization/abstraction link or a
decomposition link. For example an event “Boil Fettuccini” is a specialization of a “Boil Pasta” event, and the event “Make Fettuccini Alfredo” is composed of several events including “Boil Fettuccini” and “Make Alfredo Sauce”. The event hierarchy also defines a special type of event called the “End” event. These “End” events are not components of any other event. The whole goal of plan recognition is to identify these “End” events that would generate the set of observed events. Kautz’s framework is limited to recognize only plans in the event hierarchy. Kautz’s system recognizes stereotypical action rather than unique or idiosyncratic behavior. Kautz argues that abandoning the assumption that the system has a complete set of plans would make the size of the problem too big as the number of possible plans becomes infinite and it would be almost impossible to get to the solution in reasonable time. An example of Kautz’s event hierarchy is given in figure 3.1. Building on his assumption that the event hierarchy encodes the complete knowledge available to the agent, Kautz makes a set of assumptions that form the basis for his formal plan recognition theory.

- **Exhaustiveness Assumption**: Based on the assumption that the event hierarchy is complete Kautz assumes that the ways of specialization as represented in the event hierarchy are the only known ways of specializations. For example if a cook can make three types of “Make Sauce”, namely “Make Alfredo Sauce”, “Make Pesto Sauce” and “Make Marinara Sauce” and we know that the cook is not making Pesto or Alfredo we can conclude that the cook is making Marinara.

- **Disjointness Assumption**: The assumption is that two events are disjoint and cannot both coexist, if they neither abstract one another nor abstract a common type. For example in a hierarchy let us say there are two events “Eat in Restaurant” and “Cook Pasta”. These two events would be disjoint as they do not abstract one another or abstract a common type.

- **Component/Use Assumption**: This assumption states that if an event is observed then it implies that a disjunction of all events that has it as a component is true. For example in the event hierarchy in figure 2.2. If the “Get Gun” event is noticed then either “Hunt” or “Rob Bank” is true.
- **Minimum Cardinality Assumptions:** All the assumptions listed above are applicable for all events in the event hierarchy and these assumptions are independent of observations. They mainly describe the relationship between events in the hierarchy. The Minimal Cardinality Assumptions combines the information from several observations and the event hierarchy. Given a set of observations, using the three assumptions described above it is possible to arrive at a set of “End” events that would explain the observations. This assumption states that the recognizer only needs to consider a minimum set of “End” events that explains the observations.

![Figure 2.1: Sample Event Hierarchy Kautz [1987]](image)

It can be seen that these assumptions are intuitively reasonable and can be taken as is, but Kautz goes one step further and uses circumscription to develop a model theory for his plan recognition framework. He describes the first three assumptions above as particular circumscriptions of the event hierarchy. He uses the minimum cardinality assumption to narrow the set of models generated by circumscription. Kautz’s algorithm takes the following path: for each observation the component/use assumption and abstraction assumptions are applied till an “End” event is reached. Constraints are checked at each step and “End” events inferred from different observations are equated and resulting constraints propagated. If combining two observations results in all alternatives being eliminated then it is assumed that the
observations are from distinct “End” events. Multiple simultaneous “End” events are obtained by grouping observations in all possible combinations.

The plan recognition framework developed by Kautz is built using the expressive power of first order logic. The framework allows for multiple plans and does not put any arbitrary constraints on the ordering or temporal relations between steps. One aspect of this theory is the emphasis on “End” events to the detriment of local context. It is not straightforward to determine how the “End” event affects local events in real time incremental recognition. The plan recognition framework is similar to many plan recognition systems that came before and also some that came after [Vilain, 1990; Allen et al., 1980; Allen, 1983; Lesh, 1998] in being unable to recognize erroneous plans. Erroneous plans usually occur when the agent makes planning errors by possibly performing irrelevant actions or by changing their mind in the middle of a plan. One of the main assumptions of this framework is that the event hierarchy available to the recognizer is a complete representation of all plans in the domain. It is not always possible to generate a complete event hierarchy in realistic systems due to the number of possible plans, which tends to infinity, especially when you include erroneous plans. The worst case time for recognition of end events in large domains is exponential in the size of the knowledge base. The very conservative nature of its recognition algorithm has been raised as a concern by Lesh and others [Lesh, 1998; Blaylock, 2002]. If there are multiple possible “End” events that explain the set of observations this framework lists all of these “End” events as possibilities and does not prefer one over the other. It does not choose based on user preferences and beliefs, like heuristic recognizers [Allen et al., 1980; Allen, 1983] or probability based recognizers [Charniak et al., 1993; Bauer, 1994]. On the other hand this recognizer never makes a wrong guess, given a complete event hierarchy.

There has been work done by other researchers to improve on Kautz’s framework. Villian postulated that if you consider plan hierarchies as grammars, Kautz’s plan recognition assumptions are very similar to assumptions underlying parsing. One could then bring to bear the power of efficient parsing algorithms to do plan recognition. To use his methodology, Villian had to make some simplifying assumptions about the problem,
which limited its applicability like disallowing interleaved plans [Vilain, 1990]. Wobcke tried to adapt ideas from belief revision and conditional logic to improve the theory and incorporate the observer’s plausibility ordering of possible plans [Wobcke, 2002]. The other significant effort to use strict consistency based approach to recognize an agent goal from observation was done by Lesh [1998].

Lesh identified a set of drawbacks that, according to him, had kept formal methods from making significant inroads in the area of plan recognition. He claimed that the obstacles that Kautz’s and others faced were a) the generation of the plan library, b) fast recognition on huge plan libraries [Lesh et al., 1995; Lesh et al., 1996], and c) consideration of user preferences to prioritize goals [Lesh, 1997]. Instead of taking up the more general problem of plan recognition, Lesh, chose to take on a subset of that problem, goal recognition.

Lesh used a two pronged approach to solve the problem of massive plan libraries. Lesh formulated a novel way to create and organize plan libraries. The first idea was to limit the library to only have relevant plans and second to be able to access the knowledge-base efficiently. The construction of the plan library that encodes all the information available to the recognizer is usually left out in most frameworks, and is assumed to be automatically available to the plan recognizer [Kautz, 1987]. Lesh defines a system that would automate the construction of the library. A plan is included or discarded from the library based on a notion of bias borrowed from concept learning [Mitchell, 1982]. Bias bridges the gap between a completely generic specification of a problem domain as a set of allowed actions and a set of predicates, and a complete listing of all allowed plans. Using assumptions about type of plans people execute, called plan bias, and assumptions about the type of goals that people have, called goal bias, the system decides if a plan needs to be admitted into the library. One example of plan bias that Lesh uses is a directed-weak plan bias. This bias states that a plan can be anything as long as every action in the plan is relevant. An example of a goal bias would be conjunctive-search goal bias. In this bias the assumption is that some of the variables in the goals predicates should coincide. These biases help in reducing the size of the plan library while not being overly restrictive. To provide a compact representation of plan library Lesh borrows from
research on plan-operator graphs [Smith et al., 1993]. Large sets of plans are compactly represented as graphs called consistency graphs. The idea behind consistency graphs is to keep track of only those goals that are consistent with the observed action sequence. Informally, a goal G is consistent with a sequence of actions A if there is a plan P, which when executed has A as its prefix and results in goal G. In broad terms a consistency graph is made up of vertices that are actions and these actions are connected by edges when they support one another. The recognition proceeds by removing inconsistent edges and unsupported actions from the graph as new observations are made. Lesh provides a sound consistency checker (RIGS) that requires quadratic space and polynomial time with respect to the number of actions and goals in the system. He also provides a goal recognizer (BOCE) which he claims takes linear time with respect to the number of goals, for certain types of goals. Along with fast, sound and complete goal recognizers Lesh also developed an unsupervised learning algorithm (ADAPT) to fine tune the plan library to user preferences.

Although Lesh provided a framework to improve the efficiency of strict consistency based recognizers his work concentrates on special case of plan recognition, namely goal recognition. Goal recognition answers the question, ‘what is the user trying to do?’ rather than the question, ‘how is the user trying to do it?’ Even though Lesh’s recognizer is able to perform very fast in the test cases, some researchers [Blaylock, 2002] observe that this might be because of the type of goals represented. In the examples considered by Lesh, ruling out some goals prunes away huge sections of the search space. One observation we make when we look at Lesh’s consistency graph is that his graph works at the level of actions and is not concerned with plans or plan level interactions. Lesh’s framework makes significant improvement over Kautz’s framework in terms of scalability and speed while sacrificing on expressiveness [Blaylock et al., 2003].

There has been some subsequent work in goal recognition that is based on graph construction and analysis, by Hong [Hong, 2001]. Hong’s recognizer tries to recognize goals without a plan library. In their research they define a set of goal schemas that represent templates of possible goals for the agent. They then build a goal graph that incorporates each observed actions as it comes in. They then determine if any of the
predicates achieved by the observed actions match the goal schema. The advantage of this approach is that it eliminates the need for plan libraries. As the emphasis is on recognizing partially or fully achieved goals rather than trying to predict future actions, this framework does not make early predictions and needs to have seen significant portion of the observations before making end predictions [Blaylock et al., 2003]. While there is still work being done in developing formal methods for plan recognition most of the recent advancements have been in using uncertain reasoning or probability theory to recognize plans from observations, as we will discuss in the next section.

2.2.3 Probabilistic Recognition Methods

One of the main complaints about using deductive techniques for plan recognition is that these techniques are incapable of choosing between two plans in spite of the fact that one of the plans is more likely than the other. To give an example from the work done by Charniak and Goldman [Charniak et al., 1993], let us say that the plan library consist of two plans one that says “an agent packs his bags, goes to the airport, boards a plane and goes to Hawaii” and the other that says that “an agent packs his bags, goes to the airport and sets up camp at the airport”. One of the plans given above is very likely to happen and the other rarely happens. Formal methods of plan recognition would give both these plans equal weight when it sees the agent pack bags and go to the airport. Proponents of the probability theory based recognition argue that most formal methods ignore prior probabilities associated with plans and goals.

One of the early and strong champions of a Bayesian model of plan recognition has been Charniak and Goldman [Charniak et al., 1993; Goldman, 1990]. Their system called “wimp” does plan recognition as a two stage process. First the system retrieves a set of candidate explanations from the plan knowledge base which is indexed as a semantic network, or plan network. In the second stage the selected candidate hypotheses are added to the plan recognition Bayesian network, which represents a probability distribution over the set of possible explanation. Then by performing Bayesian updating the most likely interpretation for the given observation is chosen. Following Chainiak’s work there have been many improvements and extensions of his work that uses Bayesian
nets but as our approach is based more on formal models we will not discuss them further here [Huber et al., 1994; Pynadath et al., 2000; Bui, 2003; Albrecht et al., 1998;]

Dempster-Shafer theory has been used by some to reason about plans [Bauer, 1994; Carberry, 1990; Bauer, 1994; Bauer, 1996]. One of the early attempts at using Dempster-Shafer theory for plan recognition was done by Carberry [Carberry, 1990]. Based on psychological studies on human inference and decision making she devised a strategy that could prefer rational default inference but defer unwarranted conclusions until new evidence was collected that supported them. Bauer [Bauer, 1994; Bauer, 1994; Bauer, 1996] chose to develop his own framework for plan recognition based on DFT. While Carberry’s work was limited to making default inferences between plausible and implausible plans Bauer’s theory helps making finely tuned selection between different hypotheses. Bauer’s research contends that DST can deal with partial or total ignorance unlike Bayesian methods [Bauer, 1994]. Most of his later work was on incorporating user preferences in plan recognition [Bauer, 1996].

It is very clear that ambiguity is unavoidable when dealing with plan recognition which makes it inevitable that theories like Bayesian probability and DST will be necessary to make the right choice. However these theories if applied by brute force will not scale well. Probability calculations are very expensive and the size of Bayes nets can grow exponentially with the size of the domain [Lesh, 1998]. In new domains these methods require too many numbers that are not readily available.

In our view formal methods of recognition and method based on uncertain reasoning are complementary to each other and can coexist. Probability theory can work on smaller domains pruned using Formal methods of plan recognition. On the other hand probability theory can help with the disambiguating among multiple plausible plans considered equally valid by the formal methods.
3 \hspace{1em} \textbf{COMPUTATIONAL MODEL OF PLAN RECOGNITION}

In this section we describe our plan recognition algorithm in detail and frame it in the context of other similar work. In the first part of this chapter we define the terms that are relevant to the problem space. These terms will be used through the rest of the thesis. In the later parts of the chapter we describe our algorithm and analyze the power and scope of the same.

3.1 Definitions

\textbf{Definition 1:} An \textbf{action} describes the changes that an agent can make in the world. Our Action representation is based on the PDDL 2.1 action definition without the inclusion of fluents [Fox et al., 2003]. An action consists of 1) parameters – a list of variables on which the action rule operates, 2) precondition – a first order statement that must be satisfied before the action can be applied, 3) effects – the changes the action imposes on the state of the world. The changes are also represented in first order logic but without Skolem functions and disjunctions. The \textbf{action library (A)} lists all the actions that can be executed in the domain

\textbf{Definition 2:} An \textbf{observed action} is an action that the user executes in the world and is observed by the recognizer. An observed action contains no universal or existential quantifiers and all variables are bound.

\textbf{Definition 3:} The \textbf{observation (A)} is a totally ordered set of \textit{observed} actions. This is usually the input to a plan recognizer and corresponds to the set of actions an agent has performed.

\textbf{Definition 4:} A \textbf{partial plan} is a five-tuple: \( P = < T, O, B, ST, L >, \) where:

- \( T \) is the set of steps in the plan. \( t_0 \) and \( t_\infty \) corresponds to the first and last steps respectively.
- \( ST \) is a symbol table that maps for each step the actions that are executed in that time step. Each action is assigned to the earliest time step it can be executed without breaking any of the plan constraints.
- $O$ is the ordering relationships between the steps in $T$. The first step in the plan is always $t_0$ and the last step is $t_{\infty}$. The ordering relationship is a binary relationship between each time step pair.

- $B$ is the co-designation and non-co-designation constraints on the variables that occur in the plan actions.

- $L$ is the list of causal links. A causal link is of the form, $a_i \rightarrow a_j$, and denotes a commitment by the planner that the precondition $P$ of action $a_j$ will be supported by an effect of action $a_i$.

**Definition 5:** A **plan library** is a collection of plans that represents the complete set of plans the agent can execute in the domain.

**Definition 6:** An **instantiated plan** is a plan that is being executed by an agent. An instantiated plan can be thought of consisting of two parts: the instantiated and the expected. The instantiated part of the plan contains actions that are assumed to have already been executed and are called the instantiated actions. The rest of the actions in the plan are called the expected actions of the plan, as they are expected to be executed in the future to complete the plan.

**Definition 7:** The initial state ($I_0$) of the world is the state at which the first observed action was executed. The current state ($I_n$) of the world is the state of the system after the last observed action was executed.

**Definition 8:** The actions in a plan can be topologically sorted based on the ordering relationships in the plan into a discrete time line. Each point on the discrete time line is called a time step. Each action in the plan occupies the earliest time step it can be executed.

**Definition 9:** A ground action $a$ is said to occur next in an instantiated plan $p$ if it is possible to substitute $a$ for any of the expected actions in $p$ in the first time step after the instantiated actions in $p$, without violating any of the constraints in $p$.

**Definition 10:** The intermediate state of an instantiated plan is the state of the world after the instantiated actions in a plan. The state only consists of conditions from the causal links that have been established by the instantiated actions. It does not contain side effects of the instantiated actions. Given a plan all its intermediate states can be identified.
by going through actions in each time step and identifying all the ways in which the causal links in the plan can be established.

**Definition 11:** A **planning graph** is defined in the research by Blum and Furst [1997]. It is repeated here for completeness. *Planning graph* is a directed, leveled graph with two kinds of nodes and three kinds of edges. The levels in the *planning graph* alternate between *preposition levels* containing *preposition nodes* and *action levels* containing *action nodes*. The graph starts with a preposition level containing all the prepositions in the initial state of the system. Action nodes in action level $i$ are connected by *precondition-edges* to their preconditions in the preposition level $i$, by *add-edges* to the add effects of the actions in preposition level $i+1$, by *delete-edges* to the delete effects in preposition level $i+1$.

**Definition 12:** A **no-op** action node in an action level of a planning graph is added to the graph to ensure that every preposition in level $i$ appears in preposition level $i+1$. The no-op action is connected by a precondition-edge to a preposition in level $i$ and by an add edge to the same preposition in level $i+1$.

**Definition 13:** Two action nodes $a$ and $b$ in a planning graph are considered **mutually exclusive** if either of the two actions deletes a precondition or add-effect of the other, or if there is a precondition of action $a$ and a precondition of action $b$ that are marked as mutually exclusive of each other in the previous preposition level.

Two preposition nodes $p$ and $q$ in a planning graph are considered **mutually exclusive** of each other if all ways of creating $p$ are mutually exclusive of all ways of creating $q$.

This property of **mutual exclusion** between nodes is defined in the research by Blum and Furst [1997]. It is repeated here for completeness.

### 3.2 Assumptions

**Assumption 1:** The plan library is finite and complete. Any plan that the agent executes in the domain matches one or more plans in the plan library. The plans in the plan library might be executed in parallel. This assumption is common to most formal plan recognition frameworks [Lesh, 1998; Kautz, 1987].
**Assumption 2:** All executed actions are observed. This assumption is true in domains where the recognizer is an integral part of the environment.

**Assumption 3:** The plans executed by the agent in the system are all valid plans. This means that the plan can be represented as a series of totally ordered actions which can be executed in sequence to reach from the initial state to the goal state and for every action at the time step of its execution all its preconditions have been met.

**Assumption 4:** If the agent is executing multiple plans in parallel the agent will reach the goals state of each plan either at the end or as an intermediate state of the combined plan.

### 3.3 Algorithm

The input to our recognizer is the set of observed actions, the current state of the domain, and the library of plans in the chosen domain. The output of the recognizer is the set of plans the agent is executing and the expected actions.

The algorithm is an incremental recognizer and is based on building a planning graph similar to GRAPHPLAN [Blum et al., 1997]. The current state, \(I_n\), is taken to represent the starting state of the planning graph and exactly like in GRAPHPLAN, a planning graph is created with alternating action and preposition levels. Also similar to GRAPHPLAN, mutual exclusivity is tracked between actions and between prepositions at each level. The focus of the algorithm will be to predict the next actions based on the planning graph. The planning graph represents the possible plans in progress given the observations, this implies that a subset of prepositions \(I_s\) from a given proposition level should represent the *intermediate state* of an instantiated partial plan that the agent is executing. Each partial plan in the plan library consists of a finite set of intermediate states. As the plan library is available and assumed to be complete for a domain, the intermediate states can be determined pre-runtime. Given the proposition level and the intermediate states of a partial plan, a subset \(I_s\) can be found in the preposition level, that matches one or more intermediate states in the partial plan. \(I_s\) should not contain mutually exclusive prepositions. If there exists a valid plan that can establish \(I_s\) from \(I_n\) it can be identified by backtracking through the planning graph (similar to GRAPHPLAN’s backward-chaining algorithm) with help from the identified plan’s own structure. If such
a plan exists it is a candidate solution and can be used to predict future actions of the agent.

This algorithm identifies multiple plans that could explain the observed actions, and one or more of these plans could be the plan the agent is executing. It would be unreasonable to expect the recognizer to identify the exact plan the agent is going to execute. The recognizer has access to partial information about the intention of the agent (the observed actions). To make an exact prediction based on partial information, the recognizer would have to resort to unsound reasoning. The recognizer however does make the assumption that the plan library is complete and the agent is executing plans from the plan library. If this assumption was not made every valid plan contained in the planning graph would be a candidate plan, which could be a large set. The assumption about the complete plan library helps the recognizer limit the number of plans in the planning graph it considers as candidate. Our plan recognizer does not jump to conclusions by making an unsound choice on the candidate plan.

Create_Planning_Graph

\( I_n \) - The current state of the domain (input)

\( \Lambda \) - Action library (input)

\( PG \) – Planning graph expanded up to \( k \) levels. (output)

\( k \) – Number of preposition levels in \( PG \) (input)

This procedure creates a planning graph as described in the GRAPHPLAN algorithm [Blum et al., 1997]. The planning graph is expanded to \( k \) levels stopping at a preposition level.

1. Initialize

   1.1. Set \( PG \) to represent \( I_n \) with none of the preposition pair marked as mutual exclusive.

   1.2. Set counter = 0.

2. If the last level was preposition level create the next action level.

   2.1. For each action in \( \Lambda \) add it to the action level if
2.1.1. Its preconditions can be supported by prepositions in the previous level
2.1.2. None of the preconditions have been marked mutually exclusive
2.2. Add no-op actions that have same preconditions and effects.
2.3. Add edges connecting preconditions to actions.
2.4. For each action in action level track all mutual exclusivity with other actions in
the level
3. If the last level was action level create the next preposition level
3.1. Add effects of the actions in the previous level
3.2. Connect the effects with the actions using add or delete edges
3.3. Mark two prepositions as exclusive if all ways of generating one is exclusive of
ways of generating the other
3.4. Increment counter
4. if (counter \( \geq k \)) return \( PG \) else go back to step 2.

**IdentifyIntermediateState**

\[
(PL \rightarrow \text{Propositions at a proposition level in a planning graph (input)}\\
p \rightarrow \text{Partial Plan (input)}\\
P_{\text{inst}} \rightarrow \{(p_{\text{inst}}, \mathcal{I}_{\text{bs}}) \mid p_{\text{inst}} \text{ is an instantiated plan obtained by instantiating the input}\\
\text{partial plan } p \text{ with a intermediate state of } \mathcal{I}_{\text{bs}} \text{ where } \mathcal{I}_{\text{bs}} \in PL \text{ (output)}\\
)
\]

This procedure given a partial plan and a proposition level identifies all the intermediate
states that are established by a subset of the preposition level.

1. Initialize \( P_{\text{inst}} \) to a null set.
2. Loop through each intermediate state in \( p \)
   2.1. If the intermediate state \( \mathcal{I}_{\text{bs}} \) is a subset of the prepositions in \( PL \).
      2.1.1. Add bindings to \( p \) and create an instantiated plan \( p_{\text{inst}} \) for the intermediate
state \( \mathcal{I}_{\text{bs}} \).
      2.1.2. Add \( (p_{\text{inst}}, \mathcal{I}_{\text{bs}}) \) to \( P_{\text{inst}} \).
3. Return \( P_{\text{inst}} \).
Check Intermediate State

(  
  $I_{bs}$ - A intermediate state (input) 
  $PG$ – a planning graph (input) 
  $p_{inst}$ - the instantiated partial plan for which $I_{bs}$ is the intermediate state. (input) 
  $q$ – either a valid partial plan starting at $I_n$ that establishes $I_{hs}$ or null. (output)
)

This procedure checks to see if a given set of prepositions $(I_{bs})$ can be established starting from the current state $I_n$ using actions from the action library. This is done using a backward chaining algorithm similar to GRAPHPLAN. However in this case we have the instantiated partial plan as a guide.

1. Initialize
   1.1. goal $G = I_{bs}$
   1.2. time-step = t, planning graph level = t
2. In $p_{inst}$ find the actions in last time-step that established the prepositions in G.
3. In the planning graph $PG$ find the actions in the last action level (no-op included) that have prepositions in $I_{hs}$ as add effects. Choose actions that are not exclusive of any action that has already been selected. Use actions identified in step 2 as guide.
4. If cannot find action to satisfy $G$ go back up one time-step and $PG$ level and try another set of actions starting at step 2. If time step is t and all possible actions have been checked return null.
5. If not at initial state
   5.1. reset $G$ = precondition of identified actions
   5.2. decrement time-step and $PG$ level
   5.3. go back to step 2.
6. The path of actions is identified from initial state to goal state then,
   6.1. $q = p_{inst}$
   6.2. substitute instantiated action in $q$ with the identified action path.
6.3. return $q$

**Recognizer**

( $\Pi$ - Library of partially ordered plans, (input)
  $I_n$ - The current state of the domain (input)
  $\Lambda$ - Action library (input)
  $k$ - Number of preposition levels in $PG$ (input)
  $P$ - recognized candidate plans (output)
)

This procedure identifies the partial plans in the plan library that can explain the current state of the planning graph as an intermediate state within the plan.

1. **Initialize**
   1.1. $P$ to an empty set

2. **Graph Generation**
   2.1. Create planning graph $PG = \text{create_planning_graph}(I_n, \Lambda, k)$
   2.2. Get final preposition $PL$ level from $PG$

3. **Graph Analysis**
   3.1. For each $p \in \Pi$ do the following steps
      3.1.1. Get the set of instantiated plans $PI = \text{identify_intermediate_state}(PL, p)$
         3.1.1.1. For each $(p_{inst}, I_{hs}) \in PI$ loop
            3.1.1.1.1. Find $q = \text{check_intermediate_state}(I_{hs}, p_{inst}, PG)$.
            3.1.1.1.2. If $q$ not null add $(q, I_{hs})$ to $P$.

3.4 **Assertions**

**Claim 1:** The plan recognizer is sound.

**Reasoning:**
By definition we know the following facts about our problem
1. All the plans in the plan library are valid plans.
2. An intermediate state is a state in a partial plan that can be reached by executing some of the actions in the partial plan without violating any of its constraints.
3. GRAPHPLAN uses a backward-chaining search algorithm to identify a path of actions that lead from a goal-set in the final preposition level to the initial preposition level. In their paper Blum and Furst [1997] have shown that the backward chaining search algorithm returns valid plans. In our recognizer we have implemented the check_intermediate_state procedure that takes a subset of the current preposition level and uses the same algorithm as the GRAPHPLAN search to find a valid plan to support the preposition set.
4. With the facts given above, when our plan recognizer comes up with a partial plan $p_{inst}$ at an intermediate state $I_{hx}$ as a possible explanation (Step 3.1.1 above), it has to be verified by the check_intermediate_state procedure.
5. Being verified by the check_intermediate_state procedure implies a valid plan exists starting at the initial state $I_0$ to the state $I_{hx}$ (from fact 3 listed above).
6. Given that the intermediate state $I_{hx}$ has can established using the plan returned by the check_intermediate_state procedure, the rest of the instantiated plan, $p_{inst}$, can be executed to complete the plan. This is possible as all the causal links to execute the expected actions of the instantiated plan has been established (see definition of intermediate state).
7. Statement (6) above implies that the identified partial plan is a possible plan the agent is executing, in other words our plan recognizer is sound in its inference.

**Claim 2:** If an agent is executing a valid plan, even in the case where the execution is in parallel with other plans at least one of the intermediate states of the plan will be reached by the agent.

**Reasoning:**
1. According to assumption 4 the goal state of the every plan will be reached as either as the final state or as an intermediate state.
2. The goal state of a plan is one of its intermediate states as per the definition of intermediate states.

3. From statements (1) & (2) we can say that, if the agent is executing a valid plan from the plan library at least one of the intermediate states of the plan will be reached by the agent.

**Claim 3:** If an agent is executing a valid plan, even in the case where the execution is in parallel with other plans, and an intermediate state is reached as part of the execution, this intermediate state will be a subset of the prepositions at one of the preposition level in the planning graph. No preposition in the subset will be mutually exclusive.

**Reasoning:**

1. An agent executing one or more plans from the plan library does so by performing a series of actions in an order. Let us call this the *execution path* of the plan.
2. The intermediate state reached by the agent is between two actions in the execution path.
3. We know from Blum and Furst [1997] that the planning graph has a weaker constraint than the constraints on valid plans so if there is a valid plan it will be part of the planning graph.
4. From statements (1) and (3) and assumption 3 described in the previous section, we know that execution path is part of the plan graph.
5. From the definition of the planning graph we know that the states between the actions in the execution path are subsets of the corresponding preposition levels.
6. Since there exists a plan (the execution path) that supports the preposition subset, no preposition in the subset is mutually exclusive.

**Claim 4:** Our plan recognizer is complete with respect to the plan library. In other words we can say that, if any plan in the plan library is a possible candidate it will be identified by our plan recognizer.

**Reasoning:**

1. Let us say an agent is executing a plan $q$ from the plan library.
2. From claim 2 we can say that at least one of its intermediate-state will be reached. Let us call that state $I_s$.

3. From claim 3 above we can say that $I_s$ would be a subset of a preposition level. The intermediate $I_s$ will be identified as a subset of $PL$ by the procedure $\text{identify}_\text{intermediate}_\text{state}(PL, q)$, where $PL$ is the preposition level.

4. When we run the procedure $\text{check}_\text{intermediate}_\text{state}$ on the intermediate state $I_s$ it will return success. This we can say for sure because $I_s$ is the intermediate state of a valid plan being executed by the agent. In other words it is part of the execution path. Also, Blum and Furst [1997] show that their backward search algorithm, that we have implemented, is sound and complete. If there is a valid plan it will be found.

5. From statement (4) we can say that any plan that the agent is executing from the plan library will be identified.

3.5 Analysis

Having described the algorithm in the earlier section and having explored some claims about our algorithm in the previous section, in this section we try to describe the asymptotic behavior of our algorithm and compare it with other algorithms in the literature.

Our algorithm consist of three main procedures namely $\text{create}_\text{planning}_\text{graph}$, $\text{identify}_\text{intermediate}_\text{state}$ and $\text{check}_\text{intermediate}_\text{state}$. The $\text{create}_\text{planning}_\text{graph}$ procedure is based on similar graph creation algorithm used in the GRAPHPLAN algorithm [Blum et al., 1997]. Based on the theorem proved by Blum and Furst we can say that the size of the planning graph and the time taken to create it would be polynomial in the length of longest add list of any action, number of objects, number of prepositions in the initial conditions, the number of possible actions and the number of levels the graph is extended.

The number of intermediate states in a plan could be exponential with respect to the number of actions in the plan. If we assume that the size of the plans in the plan library is bounded and finite, then we can also state that the number of intermediate states for all
the plans in the plan library is a finite quantity. The plan library contains the same plans for every run of the recognizer. Which means the intermediate states for all the plans in the plan library could be computed ahead at design time and does not have to be determined at runtime. We will talk about the asymptotic behavior of our algorithm in terms of the total number of intermediate states and not the total number of plans in the plan library. In the `identify_intermediate_state` procedure of our algorithm, every intermediate state in the plan library needs to be compared to the prepositions in a preposition level. The comparison proceeds by checking if every preposition that is true in the intermediate state is true in the preposition level, and to also checking that no two prepositions in the intermediate state are marked as mutually exclusive in the preposition level. This check would be a polynomial function of the number of prepositions in the preposition level and the number of intermediate states in the plan library.

The third procedure in our algorithm `check_intermediate_state` does a backward chaining search through the planning graph to see if there exists a valid plan that could establish the intermediate state from the current state after the observation. The complexity of a backward chaining search in the worst case will be exponential. At every action level this search needs to consider all the possible sets of actions that establishes the goal state and so the complexity will be exponential in the number of action at each level of the planning graph. However as Kambhampati [1997] has shown the power of GRAPHPLAN is in its usage of the planning graph to represent a pruned state-space search tree, using mutex propagation. The efficiency of GRAPHPLAN is based on the propagation of mutex relationships. The backward chaining search in our algorithm also has the advantage of having an identified partial plan. The search can use the partial plan as a heuristic to guide the choice of actions at each action level. This heuristic might significantly reduce the number of paths that need to be checked before we identify a valid plan.

### 3.6 Implementation

The algorithm is implemented in Java and was tested on a 2 GHz intel Windows machine with 2 GB of RAM. The implementation was tested with multiple example domains.
namely a cooking domain and a vacation domain. The data structures used for this implementation were standard data structures provided by JDK 1.4.2. No optimization was done to implement more efficient data structures. The vacation domain consists of 5 types of objects and 14 objects. There are 9 actions in the action library and 5 plans in the plan library. The longest plan consists of 9 actions and the shortest 5 actions. When the planning graph was expanded forward to 3 preposition levels, the recognizer took about a 1 second to list all the possible plans for the given initial state.
4 EXAMPLES

In this chapter we use some example domains to describe the algorithm presented in the previous chapter. The first example we will use to highlight the algorithm will be from the cooking domain. This example will be followed by an example about vacation planning.

4.1 Cooking Domain

The cooking domain plan library consist of four possible plans, making a meatless pasta dish, pasta with meat, a meatless salad and a salad with meat.

![Image of cooking domain actions]

Figure 4.1: The Cooking Domain Actions
Figure 4.2: The Cooking Domain Plan Library

Figure 4.3: The Planning Graph
Let us assume that three actions were observed by the plan recognizer and they were namely

boil (FETTUCINE) $\rightarrow$ makeSauce(ALFREDO) $\rightarrow$ grill(CHICKEN)

The state of the system after the observed actions is

$\text{Prepared}(\text{FETTUCINE})^\wedge\text{empty}(\text{PLATE})^\wedge\text{hasSauce}(\text{ALFREDO})$

$\wedge\text{prepared}(\text{CHICKEN})^\wedge\neg\text{prepared}(\text{LETTUCE})$

The figure 4.3 shows the expansion of the planning graph by one prepositional-level using the create_planning_graph procedure. The highlighted actions correspond to the observed actions. The next level shows predicted actions based on the current state $I_n$.

Given the preposition level $PL$ and the plan library ($\Pi$), doing the graph analysis as explained in the recognizer procedure above, we can identify the intermediate state for each $p \in \Pi$. Each plan instance, $(p_{\text{inst}}, I_{hs}) \in PI$, identified in the graph analysis step is given below in the following table (Table 4.1). The table shows what the next expected action is for the identified instantiated plan and also the valid plan returned from check_intermediate_state procedure. Extending the planning graph based on the expected actions listed Table 4.1 we can see that if the Plate(FETTUCINE) is executed then there can be no salad on the menu as there will be no empty plate remaining. However if the next action is Chop(LETTUCE) then all four plans are still possible. The mutual exclusivity between the plate_pasta action and plate_salad action splits the number of possible plans into two groups. One set of these plans would be eliminated based on the next observed action.
Table 4.1: Cooking Domain Recognizer Results

<table>
<thead>
<tr>
<th>$I_{bs}$</th>
<th>$P_{inst}$</th>
<th>Observed Actions</th>
<th>Expected Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasSauce(ALFREDO), prepared(FETTUCINI), empty(PLATE)</td>
<td>Make Pasta Dish</td>
<td>Boil(FETTUCINI), MakeSauce(ALFREDO)</td>
<td>PlatePasta(FETTUCINE)</td>
</tr>
<tr>
<td>hasSauce(ALFREDO), prepared(FETTUCINI), prepared(CHICKEN), empty(PLATE)</td>
<td>Make Meat Pasta Dish</td>
<td>Boil(FETTUCINI), MakeSauce(ALFREDO), Grill(CHICKEN)</td>
<td>PlatePasta(FETTUCINE) → AddMeat(CHICKEN)</td>
</tr>
<tr>
<td>¬prepared(LETTUCE), empty(PLATE)</td>
<td>Make Plain Salad</td>
<td></td>
<td>Chop(LETTUCE) → PlateSalad(LETTUCE)</td>
</tr>
<tr>
<td>¬prepared(LETTUCE), Prepared(CHICKEN) empty(PLATE)</td>
<td>Make Meat Salad</td>
<td>Grill(CHICKEN)</td>
<td>Chop(LETTUCE) → PlateSalad(LETTUCE) → AddMeat(CHICKEN)</td>
</tr>
</tbody>
</table>

The cooking domain is a simple domain with not much variety in the number of possible paths that an agent can take. However it is useful to show the workings of the algorithm and also highlight the power of mutual exclusivity in narrowing the possible explanations produced by the algorithm. We will now move on to another example that has a little bit more variety.

### 4.2 Vacation Domain

In this example we describe a vacation scenario, where the plan library consists of the five types of vacation plans. Two of them involve going to the beach, one involves going on a flight, one taking a cruise and one vacation plan is about going to the theater.

Let us assume that two actions were observed by the plan recognizer and they were namely,

**pack (LUGGAGE) → buy_tickets(THEATER).**

Let us say the initial state before the two observed actions was
~packed(LUGGAGE)\textsuperscript{at}(HOME, LUGGAGE)\textsuperscript{at}(HOME, AGENT)

Using these two pieces of information we can draw the planning graph as described in our plan recognition algorithm. The figure 4.6, below, shows the planning graph generated for this domain. The graph in the figure has been generated up to three action levels and four preposition levels. Even though the graph can be extended to multiple levels, as the number of levels increases the number of states that can be reached also increases. As the number of possible predicates increases, the number of possible plans that could explain the observation also increases, which means the predictive power of the algorithm decreases as the depth of the graph increases. On the other hand as the plan graph is extended the number of mutual conflicts among prepositions could also possibly increase which would reduce the number of valid preposition subsets and hence the number of candidate plans.

The state of the system after the observed actions are executed is

packed(LUGGAGE)\textsuperscript{at}(HOME, LUGGAGE)\textsuperscript{at}(HOME, AGENT)\textsuperscript{has_tickets}(THEATER)

The only actions that are possible at this state are hailing for a cab or buying tickets. Having hailed a cab the agent has transport to travel. The details of the planning graph are summarized in the figure 4.6 below.

For brevity multiple actions are represented as one at the action levels. For example the drive action represents driving from a location ?D to a location ?E, where ?D and ?E could be any two locations. These drive actions would actually be classified as mutually exclusive according to the definition set by Blum and Furst [1997] in graph plan. For example if the agent drives from home to the theater, the drive action that has the agent driving from home to the airport would be considered mutually exclusive. The effect of one (~\textsuperscript{at}(HOME, AGENT) ) would negate the precondition of the other (\textsuperscript{at}(HOME, AGENT) )
Figure 4.4: The Vacation Domain Actions

At the action level 2 the possible actions are driving from home to somewhere else and loading the luggage into the cab. At next proposition level (level 3) multiple intermediate states can be identified for each plan in the plan library. The Table 4.2 below gives some of the identified plan instances. Note that only some of the plans explain all the observed actions. This is not a requirement for a plan to be a candidate for solution. As the observed actions might result from multiple plans being executed in parallel, observed actions might be white knights that reestablish some broken causal link needed to execute plans in parallel. Note that in the expected actions column the last action is in italics, this is to emphasize the difference between instantiated and expected actions.
Figure 4.5: The Vacation Domain Plan Library
Figure 4.6: The Vacation Domain Planning Graph A

Table 4.2 gives instantiated plans that could be in progress given the observed actions returned from step 3.1.1.1.2 (recognizer), some intermediate states and instantiated plans were eliminated in the previous steps of the recognizer. Examples of a subsets of the preposition level that does not match any intermediate plan state are at(AIRPORT, CAR) ^ at(HOME, AGENT) or at(HOME, AGENT) ^ at(AIRPORT, LUGGAGE). These preposition set will be filtered out by the identify_intermediate_state procedure. On the other hand not all the instantiated plans identified in step 3.1.1 (recognizer) will return a valid plan from the procedure check_intermediate_state in the step 3.1.1.1.1 (recognizer). For example the instantiated plan (rent-car-and-go-to-beach, in(CAR, LUGGAGE) ^ at(HOME, CAR) ^ at(HOME, AGENT)) does not match any valid subset at the given preposition level and so cannot generate a valid plan.

Now let us say the next observed action is Hail(HOME, CAB). This changes the original planning graph and the change is shown in the planning graph, figure 4.7.
Table 4.2: Vacation Domain Recognizer Results

<table>
<thead>
<tr>
<th>$I_{bs}$</th>
<th>$P_{inst}$</th>
<th>Observed Actions</th>
<th>Expected Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packed(LUGGAGE) ^at(AIRPORT, AGENT) ^at(HOME, LUGGAGE)</td>
<td>Rent car and go to beach</td>
<td>Pack(LUGGAGE)</td>
<td>Hail(HOME, CAB) → Drive(HOME, AIRPORT) → Rent_car()</td>
</tr>
<tr>
<td>Has_tickets(AIRPLANE) ^in(CAB, LUGGAGE) ^at(HOME, AGENT) ^at(HOME, CAB)</td>
<td>Fly away from home</td>
<td>Pack(LUGGAGE)</td>
<td>Hail(HOME, CAB) → Buy_tickets(AIRPLANE) → Load_luggage() → Drive(HOME, AIRPORT)</td>
</tr>
<tr>
<td>Has_tickets(CRUISE) ^in(CAB, LUGGAGE) ^at(HOME, AGENT) ^at(HOME, CAB)</td>
<td>Go on a cruise</td>
<td>Pack(LUGGAGE)</td>
<td>Hail(HOME, CAB) → Buy_tickets(CRUISE) → Load_luggage() → Drive(HOME, PORT)</td>
</tr>
<tr>
<td>at(THEATER, AGENT) ^has_tickets(THEATER)</td>
<td>Go to theater</td>
<td>Pack(LUGGAGE) → Buy_tickets(THEATER)</td>
<td>Hail(HOME, CAB) → Drive(HOME, THEATER) → Watch_Show()</td>
</tr>
<tr>
<td>in(CAB, LUGGAGE) ^at(HOME, CAB) ^at(HOME, AGENT)</td>
<td>Drive to beach</td>
<td>Pack(LUGGAGE)</td>
<td>Hail(HOME, CAB) → Load_luggage() → Drive(HOME, BEACH)</td>
</tr>
</tbody>
</table>

Given the proposition level as shown in the planning graph we can see that the intermediate states identified in Table 4.2 are still true. However if the next action observed is Drive(HOME, THEATER) the graph changes as shown in figure 4.8

Table 4.3: Vacation Domain Recognizer Results

<table>
<thead>
<tr>
<th>$I_{bs}$</th>
<th>$P_{inst}$</th>
<th>Observed Actions</th>
<th>Expected Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>at(THEATER, AGENT) ^has_tickets(THEATER)</td>
<td>Go to theater</td>
<td>Pack(LUGGAGE) → Buy_tickets(THEATER)</td>
<td>Hail(HOME, CAB) → Drive(HOME, THEATER) → Watch_Show()</td>
</tr>
</tbody>
</table>

The only possible plan being executed based on the last preposition level is given above in Table 4.3
Looking at the planning graph, the only plan that is possible seems to be the plan of watching a show. However if the agent decides to drive back home the agent can get back
to the initial state and all other plans in the plan library will again become possible. This example showed how as observed actions come in one by one; the recognizer incrementally updates its internal model and gives the explanations for the observed actions using our algorithm.

The two recognition problems described in this chapter shed light on the mechanism of the algorithm described in the previous chapter. The examples show that in problems where plans are built on similar sequence of actions, it is usually hard to guess the intention of the agent based on limited set of observations. It will always be possible from the planning graph to identify multiple plans that could explain the observed actions. If the plan space is segmented, in the sense that, going down one set of steps prevents the possibility of another set of plans and mutual exclusion between actions is the dominant source of conflict between plans, then this approach clearly has an advantage.

Considering mutual exclusion while identifying explanations allows the recognizer to work in segregated segments of the plan space. This groups the plans in the plan library and allows for a smaller set of candidate plans to be identified.
5 ANALYSIS OF THE RECOGNIZER

The previous chapter describes the plan recognition algorithm and explores its reach through multiple examples. This chapter tries to place the algorithm in the context of current literature and compares the similarities and differences with other recognizers.

5.1 Comparison with Peers

From the extensive literature on plan recognition the three approaches that are the most similar to our recognizer would be the algorithms of Kautz [1987], Lesh [1998] and Hong [2001]. While Kautz and Lesh expect a plan library to be defined before the recognizer can be run, Hong only expects goal schemas which are parameterized goal prepositions. Kautz’s work could be divided into two parts, a formal theory to describe the problem of plan recognition and some proposed algorithms to recognize plans. Our approach follows Kautz and Lesh, to the extent that, we expect the recognizer to have access to a plan library which is assumed to contain a complete set of all partially ordered plans available to the agent. The algorithms proposed by Kautz’s focuses on recognizing intentions and end actions. Kautz’s algorithm tries to find the minimum set of end events that would explain the observation. This is based on the minimum cardinality assumption. In our approach we have focused more on identifying all possible plans that could explain the observation and we show how much of each possible plan has been completed and how the agent is likely to complete the plan. By focusing on finding the minimum set of plans that explains all the observed actions, the recognizer could possibly ignore possible plan-sets that explain the actions but require more than a minimum set of plans. For example, let us say that an agent is seen going to the bank after picking up a shotgun. Let us say that the plan library contains three plans one of robbing the bank, one of going to the woods hunting and one of going to the bank to withdraw cash. By focusing on getting a minimum set of end events we would, after reviewing the observation, assume that the agent is going to rob the bank. However there still exists a possibility that the agent is just going to the bank to withdraw some cash before he goes hunting in the woods. In our algorithm the planning graph would show all the possible plans and the analysis would consider both the cases. The candidate set returned by our recognizer is not specific and
does not identify a single plan but it is more complete with respect to considering all possible intentions of the agent.

Unlike Kautz’s recognizer and our recognizer, Lesh is interested in goal recognition and not plan recognition. In other words, Lesh is interested in identifying a goal set that is consistent with the observed actions, while we are interested in identifying plans that are consistent to the observed actions. Lesh also concentrates on automated construction of plan libraries. These plan libraries are built, based on some simple assumptions called biases, regarding the chain of actions forming the plans. Lesh’s week-bias plan library would consider all possible action sequences built from a give action schema set as long as all the actions in the plan are in service of another action or the final goal of the plan. One of the assumptions made by Lesh is that there are no irrelevant actions, and that the observed actions form a prefix to every plan that the agent is executing. We do not make that assumption. We do not care if irrelevant actions were executed by the agent, as long as it is possible for our algorithm to identify a valid plan that gets to the intermediate state of a plan in the plan library starting from the initial state of the system. The current state is the state reached by the agent after the observed actions have be executed. Our algorithm makes sure that the intermediate state is a subset to the current state.. In our algorithm we could modify the check_intermediate_state procedure to only consider plans that have the observed actions as prefix and where the observed actions are in the service of some other action in the plan. This restriction would reduce the number of candidate plans returned by our algorithm but it would match the behavior of Lesh’s recognizer.

One of the criticisms of Lesh’s algorithm is that it works fast only because in the examples that were used the goals were represented in a subsumption hierarchy. This type of domain would also help our recognizer as many plan paths in the planning graph could be pruned based on the mutual exclusion relationships maintained as part of the planning graph. One of the features provided in Lesh’s recognizer is the idea of making the plan library dynamic instead of static. Our assertion has been that building the plan library is separate from plan recognition and is an on going process. Plans in the plan library could be added or deleted based on information from the domain as more
observations are gathered. We accept that the plan library does change. Our contention is that the changes occur at a much larger time scale when compared to the time scale of individual recognition problems.

Hong’s approach [2001] is based on expanding the planning graph as in GRAPHPLAN [Blum et al., 1997], which is similar to our approach. Like Lesh, Hong is more interested in goal recognition. Hong’s goal recognizer does not need complete plan libraries but it uses goal schemas that are parameterized prepositions. These parameterized prepositions are the intended goals. We can say that the similarity between our approaches is restricted to the use of planning graphs. While Hong stops at identifying groups of prepositions in the planning graph that match the goal schema, we go a step further and try to identify intermediate states of plans, states that are before the end goal state of the plan. We are able to do this because we assume that the plan space available to the agent is finite and is restricted to the plan library. The approach taken by Hong could be implemented by our algorithm if a predefined preposition set is returned from the identify_intermediate_state procedure, every time the procedure is invoked. This would mean that given a fixed intermediate state check_intermediate_state would have to find a valid plan.

Our approach has many positives when compared to what is available in the literature as explained above. However one possible limitation of our approach is that it does not attempt to call out a specific plan or a plan-set as being the explanation of the observed actions. In other words we favor completeness over specificity. The probability based approaches are better at disambiguating between the candidate plans and should be used for that purpose but strict consistency based approaches provide sound recognizers and should be valued accordingly. As always probabilistic and heuristic methods could be used to pick a specific plan from the plan library to improve specificity.

5.2 Applications of the Recognizer

In the current format the algorithm returns all possible plans from the plan library that the agent could be executing for an observation. The question could be flipped and the recognizer could be asked to determine if a specific plan is a possible candidate. In the graph analysis phase of the recognizer instead of going through all the plans in the plan
library and calling \texttt{identify\_intermediate\_state} procedure for each of those plans, the recognizer would call the procedure on the partial plan of interest and if a possible intermediate state is identified it would be checked by calling \texttt{check\_intermediate\_state} and the recognizer would clearly identify if the partial plan could be a candidate plan or not. Since in this scenario the recognizer does not have to consider all the plans in the plan library, the recognizer would be faster than if all possible plans need to be considered. This feature could be useful in many real life scenarios. For example in the case of intrusion detection, we might want to know if a specific intrusion technique is being used by a malicious agent. In the case of a tutoring domain the system might be interested in identifying the specific pattern being used by the student in solving the problem.

Plan recognition is used for mediation, the goal is to steer the agent away from certain states that could possibly disrupt a pre-decided plan. The idea is to check if the agent’s actions could lead to a state that disrupts some critical causal link in the pre-decided plan. So if the disruptive set of prepositions is a subset of a preposition level in the planning graph, the graph analysis step of the plan recognition can be modified to return all partial plans from the plan library that have the offending preposition subset as an intermediate state through the \texttt{identify\_intermediate\_state} procedure. These intermediate states would be checked by calling \texttt{check\_intermediate\_state} and the recognizer would clearly identify the partial plans that could break the pre-decided plan. It might be possible, using the planning graph, to identify alternate actions and interventions to thwart the agent’s disruptive plan. The roadblocks to the disruptive plan can be selected from the mutual exclusions identified in the planning graph. By executing these mediations it would be possible to avoid getting the system into an untenable state that causes disruptions to the pre-decided plan.

The two scenarios discussed above shows the flexibility of the algorithm in adapting to different plan recognition needs. When the question is identification of plans being executed, the recognizer is biased towards completeness. Sometimes the goal of the system is not identifying the exact plan of the agent. Alternate questions could be asked, like in the scenarios described above, which would lead to more specific answers from
the plan recognizer. This however would require making simple modifications to our algorithm to accommodate the different needs of the new problem.

As can be seen from the discussion above there are some problems that are out of reach for our algorithm, but there are many domains and questions for which our algorithm can be a very good fit.
6 Conclusion

The first chapter in this thesis describes a schematic representation of a plan recognizer. The primary components of that diagram are the reasoning-engine, plan-library, observer, observed, and heuristics-engine. In this research we have focused on developing a flexible reasoning-engine that can identify the overall plans and intentions of the observed and at the same time predicting its immediate next actions. The overall intention is gleaned through the identification of plan-sets from the plan-library that can explain the observed actions. The local predictions are made based on the expected actions in the identified candidate plans.

The reasoning-engine is built using the important concepts that lead to the development of efficient planning systems. The goal of a planner is to identify that one plan that will take the agent from an initial state to a goal state. In the case of a plan recognizer the intention is to identify the possible plans and the goal state of the agent based on some preliminary actions starting from the initial state. To make the problem tractable we do need to make the assumption that we know all the plans that the agent could be executing. GRAPHPLAN, an efficient planner, uses mutual exclusivity to partition the plan space into plans that cannot coexist. This feature serves us well in tackling the problem of plan recognition. Plan recognition in our case is the search for plans from the plan library in the plan graph. By eliminating parts of the plan space using mutual exclusion we make our search more efficient.

In Chapter 2 we described the motivation behind this research and in subsequent chapters we have shown our progress towards addressing the needs described there. But, as always, with any good research, there are a lot of questions that have arisen out of our research as there are answers provided. As described in detail in earlier chapters, one of the caveats of our approach is the avoidance of abductive reasoning. In many real life situations our approach would give a large number of possible solutions without providing any help with narrowing down these solutions to one explanation. This we have argued would have to be based on abductive reasoning which has been the forte of systems built using probability. One of the areas we would like to explore in the future.
would be the creation of a hybrid system that uses deductive reasoning to create a shortlist of explanations and then uses probability to narrow the selection down to one. The success of our recognizer depends on the availability of a complete plan library. In most realistic systems the creation of the plan library would be a very big task. Lesh [1998] addressed this concern by building these plan libraries based on domain independent biases. A domain independent approach might give a comprehensive plan library but the library might be cluttered with improbable, trivial and unusable plans. It would be more effective to use the recognizer itself to catalog new plans that fall outside the library and incorporate it into its repository for future use. This approach would make the plan library a more dynamic entity rather than a static entity. The plan library could also be tied to specific user preferences or user models to make the recognition more effective.

In summary, we have attempted to address specific aspects in the area of plan recognition. As computers and humans interact in more sophisticated ways there is a need to develop systems that make the interface more intuitive and less formal. Building systems that understand the user at a higher level of abstraction than specific key strokes or button clicks is vision for the future and hopefully this research has moved the boundaries forward, in that direction.
REFERENCES


