

INCREMENTAL CASE-BASED PLAN RECOGNITION WITH LOCAL PREDICTIONS

BORIS KERKEZ and MICHAEL T. COX

*Department of Computer Science and Engineering
Wright State University, Dayton, OH
{bkerkez, mcox}@cs.wright.edu*

We present a novel case-based plan recognition method that interprets observations of plan behavior using an incrementally constructed case library of past observations. The technique is novel in several ways. It combines plan recognition with case-based reasoning and leverages the strengths of both. The representation of a plan is a sequence of action-state pairs rather than only the actions. The technique compensates for the additional complexity with a unique abstraction scheme augmented by pseudo-isomorphic similarity relations to represent indices into the case base. Past cases are used to predict subsequent actions by adapting old actions and their arguments. Moreover, the technique makes predictions despite observations of unknown actions. This paper evaluates the algorithms and their implementation both analytically and empirically. The evaluation criteria include prediction accuracy at both an abstract and a concrete level and across multiple domains with and without case-adaptation. In each domain the system starts with an empty case base that grows to include thousands of past observations. Results demonstrate that this new method is accurate, robust, scalable, and general across domains.

Keywords: Plan recognition, case-based reasoning, incremental learning, local prediction.

1. Introduction

In trying to interpret the world, an agent must be able to explain the events that take place as the world changes. Most important is the explanation of volitional events caused by other actors in the environment. Other actors have goals and plans to achieve these goals, but the observer often has access to only a stream of actions and events that occur. Case-based reasoning attempts to use experience of past events to interpret current events, and moreover tries to infer the goals of the observed actor and to predict what the actor will do next. Plan recognition is another technique that attempts to match current actions to known plans in order to predict the actions and goals of an actor. The research presented here combines both methods in a very novel manner to perform the same inferential tasks (to perform goal and action prediction).

Inference in general is a very difficult problem, because it amounts to “mind reading”¹ or trying to understand the motivations, intent, and causal relations between what an agent thinks given what an agent does. In common sense reasoning with humans, however, this is a ubiquitous affair. Spouses often know what their counter-part wishes before an intention or goal is verbalized. Yet, actual intention may never be fully revealed. Deep inferential ability comes with extensive interactive experience over a long period of time and across numerous situations, but misunderstandings still occur frequently. However correct inferential expectations and predictions continually enable intelligent decision-making and behavior in an extremely complex world.

1.1. Plan recognition.

In plan recognition systems, the main performance task is recognition of an observed agent’s plans and goals, based on agent’s planning behavior. Recognition can be *intended*, as in natural language dialogue, where the observed agent is aware of the recognizer, and tries to assist or obstruct the plan recognition task.^{4,14} During *keyhole* plan recognition, the observed agent does not participate in the recognition process; the recognizer determines the goals of the planner based solely on available observations of the agent’s behavior. In order to infer the plans and goals of the observed agent, the recognizer typically compares the observations of the agent’s behavior with possible plans contained in its *plan library*, and tries to find the plan(s) from the library that would account for the observed behavior.

In a large number of plan recognition systems, the plan library is specified for the recognizer *a priori*, by some external agent who is often the system designer.^{8,31} This limits the plan recognition process because only the plans known in advance can be recognized. Novel plans cannot be considered. Furthermore, it is often required that the plan library be complete.^{8,56} That is, the plan library contains all of the possible plans an observed agent may pursue. This approach is suitable in situations where possible plans can be enumerated in advance. However, enumerating all of the plans in large domains is an exceedingly difficult knowledge acquisition task, because the number of possible plans may be very large. Furthermore, such an enumeration may include many plans that the agent never uses in practice. Lesh and Etzioni show how the presence of extraneous plans in the plan library can impact the efficiency of the recognizer.⁴¹ They also introduce techniques to automate the process of constructing the plan library by synthesizing possible plans and goals of the observed agent through plan and goal biases. Automatic plan-library construction algorithms suffer from the same problem as hand-coding a plan library. The plans that are automatically generated may be extraneous despite complex generation biases that minimize their occurrence.

1.2. Case-based reasoning

Case-Based Reasoning (CBR) is a technology that emphasizes retrieval of past episodes from a distributed knowledge base (memory) followed by adaptation in context to fit the current situation, rather than decomposition and re-composition of the problem and solution *ab initio*.³⁹ CBR systems have been successfully used in numerous problem-solving

¹ Paul Cohen called such problems of inference “the mind-reading problem” during an invited presentation at the AAAI-99 Workshop on Mixed-initiative intelligence.¹⁷

tasks and domains; however, the central tasks CBR addresses are those of large-scale interpretation, classification, diagnosis, and explanation of complex situations.

In very general terms, case-based interpretation is summarized in the following five steps:^{21,30,48}

1. Given a new observation, *select an index* from salient features in the input and *retrieve* a set of prior cases from the case base.⁴⁶
2. Given the retrieval set, *select the most similar* past case to provide a basis for interpretation of the observation.³⁸
3. *Adapt* the old interpretation based on the difference between the old observation and the new observation.^{27,28}
4. *Apply* and *evaluate* the new interpretation.^{28,40}
5. If warranted, *store* the new interpretation in the case base by selecting a set of indices from salient features in the observation.³⁷

One of the characteristics of the CBR approach is the emphasis upon similarity evaluation and partial matching. A past case is usually only close in its fit to the current problem. Thus the most similar example from past experience must be found in order to apply the outcome to current needs. Similarly, to perform plan recognition, a system has no guarantee for an exact match to the historical events when plan library is not complete.

We take a case-based approach to plan recognition. The approach is applicable to state-space planners where world states of the planner can be monitored and where states are represented as collections of logical predicates. We focus not only on the knowledge about the actions the planner performs but also on the states of the world in which the planner finds itself during planning. We show that the knowledge about the world states increases the efficiency of the recognizer in planning domains with wealth of the state information. Cases rich in state knowledge enable more informed predictions for recognition systems in which the planner's internal decision cycle (i.e. reasons for taking actions) is not observable. We propose a scheme intrinsic in the case-based approach, in which a plan recognition system is able to learn about novel plans and incorporate them in its plan library for future use, thus incrementally improving its recognition process. By employing a case-based approach to plan recognition along with the knowledge about planner's world states, our system is able to predict the agent's behavior even when the actions and states observed are not consistent with any plans in the plan library. This is because our system is able to partially match past cases that guide the recognition process when exact matches are not available. Such approach greatly improves the robustness and flexibility of a plan recognition system.

1.3. Incremental case-based plan recognition

Although the case-based approach to plan recognition is not new, the novelty of our approach primarily arises from the manner in which we represent a plan (i.e., a case) and from the representation of indices by which we store and retrieve plans.^{7,29} Unlike most plan recognition systems that represent a plan as a sequence of actions bracketed by an initial state and a goal state,³¹ we represent a plan as a sequence of action-state pairs such that the initial pair is as follows:

(<null-action>, <initial-state>)

The last action is also represented as follows:

`(<final-action>, <goal-state>)`.³⁴

Therefore unlike traditional representations, our plans contain intermediate state information. By saving this intermediate state information, we can find past cases that match a current observation at arbitrary points in a stream of observed actions by maintaining the states that result from the observed actions.

However, because we maintain intermediate state information, the structure of a plan is far more complex. Furthermore many more past cases will match a single observation, because given a current observed state, the state may match multiple intermediate states in multiple past cases. In order to compensate for this additional complexity, we take advantage of a smaller abstract state-space that corresponds to the much larger concrete state space existing in a given domain.

Consider for example the logistics (i.e., package delivery) domain that contains trucks that transport packages from one location to another.⁵² In this domain, packages are to be transported to particular destinations. Destinations can be either a post office or an airport. Airplanes fly between airports, and, trucks transport packages between an airport and a post office. The defined actions that exist in this domain are as follows: *load-truck* (*TRUCK*, *OBJECT*), *load-plane* (*PLANE*, *OBJECT*), *drive-truck* (*TRUCK*), *fly-plane* (*PLANE*), *unload-truck* (*TRUCK*, *OBJECT*), and *unload-plane* (*PLANE*, *OBJECT*). A single state in this domain might be the situation where one package (say *Package-1*) is located at a post office (perhaps *PostOffice-1*) and a truck (*Truck-1*) is located at the same post office. The concrete representation for this state is the set containing the following two ground literals.

`{(at-obj Package-1 PostOffice-1), (at-truck Truck-1 PostOffice-1)}`

An alternate state might be represented in the following set.

`{(at-obj Package-1 PostOffice-2), (at-truck Truck-1 PostOffice-1)}`

These are different states, because in the first example both the truck and package are at the same location and in the second state they are at separate locations. But note that the literal *(at-truck Truck-1 PostOffice-1)* is shared by both states. The difference between the two states comes from the *at-obj* literals. Now if we replace the ground literal with the corresponding generalized literal *(at-obj package location)*, the two states are the same.

A more complex state is shown in Figure 1. The objects in this state include three packages, three trucks, and two planes (see legend). The figure also shows two cities, each with an airport and a post office. The literals that exist in the domain include *at-truck*, *at-airplane*, *at-obj*, *inside-truck*, and *inside-airplane*.² The literals are shown in Table 1. In this domain an abstract state is represented as a feature vector having dimensionality equal to the number of generalized literals. If the dimensions are listed in the order that the literals are enumerated immediately above, then the abstract state for

² The literals in Table 1 are actually *dynamic* predicates. Three *static* predicates also exist. Static predicates never change, because no operators exist that can change them.

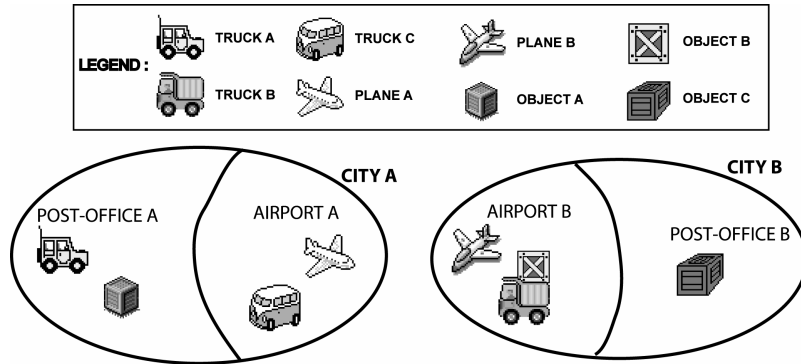


Figure 1. A simple example of the logistics planning scenario with two cities.

Table 1 is $[3\ 2\ 2\ 1\ 0]$. The reason for this is that the state contains three *at-truck* literals, two *at-airplane* literals, and so on for the remainder of the vector. If we encode the simple states from the previous paragraph likewise, they would both be represented by the vector $[1\ 0\ 1\ 0\ 0]$.³ A concrete plan can be represented in abstract form as a graph within an abstract space.³⁴ A particular plan contains a vertex for each abstract state and edges between vertices to represent actions that transition the plan from one abstract state to another (see Figure 2). Each abstract state then points to a bin containing all previously observed concrete states that share the same abstract representation. Furthermore, each concrete state in a given bin points to its location within all cases having that concrete state. Each case (plan P_i) is of length k_i . The benefit of using this abstract state representation is that a case library is thereby partitioned into a relatively small number of bins such that each bin is a unique abstract state.

Table 1. Ground literals representing the state in Figure 1.

PREDICATE	ARGUMENT 1	ARGUMENT 2
AT-TRUCK	TRUCK A	POST-OFFICE A
AT-TRUCK	TRUCK B	AIRPORT B
AT-TRUCK	TRUCK C	AIRPORT A
AT-AIRPLANE	PLANE A	AIRPORT A
AT-AIRPLANE	PLANE B	AIRPORT B
AT-OBJ	OBJECT A	POST-OFFICE A
AT-OBJ	OBJECT C	POST-OFFICE B
INSIDE-TRUCK	OBJECTB	TRUCKB

For example given the state $[3\ 2\ 2\ 1\ 0]$ illustrated in Figure 1 such that *PlaneB* and *TruckB* (loaded with *ObjectB*) is at *AirportB*, we might observe the actions of unloading *ObjectB* from *TruckB*, loading it into *PlaneB*, flying it to *AirportA*, and unloading it there. In the abstract space, the sequence of actions would be represented as the vectors and

³ Like the abstract vector formed from Table 1, this vector excludes static predicates.

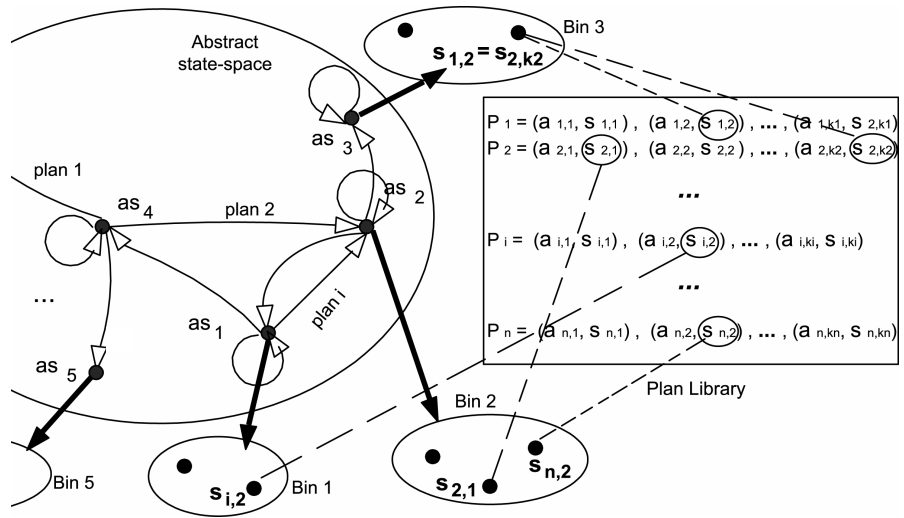


Figure 2. Indexing and storage structures. Abstract states (as_i) point to bins (bold lines), containing world states (s_j), which point (dashed lines) to past plans (P_j) in which they occurred.

transitions as solid lines (plan P_1) shown in Figure 3. After observing the first action, however, another possibility is that the truck is driven to *Post-OfficeB* where *ObjectC* is loaded instead. This sequence is shown as dashed lines (plan P_2). But which sequence is most likely is not clear simply from observing the unload action.

When *retrieving* old cases to interpret the initial situation, a system will transform the currently observed concrete state of Table 1 to the corresponding abstract state [3 2 2 1 0] (linear time computation), find the bin that is equivalent to the abstract state (constant time), and return those cases indexed by each concrete state in the bin (linear). Although we discuss retrieval in detail within Section 3, briefly consider Figure 2 again. If the current state maps to abstract state as_i shown in Figure 2, then retrieval is limited to bin one. Bin one contains the concrete state $s_{i,2}$ and points to plan P_i .

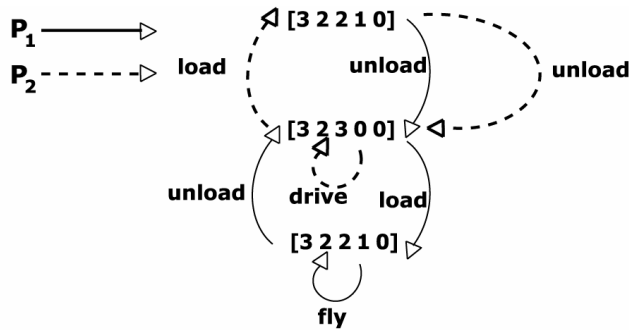


Figure 3. Plan representation in an abstract space.

If the retrieval set were composed of multiple past plans, then the system would *select the most similar case* among them relative to the current observation sequence.

Were the set empty, the system selects another index for further retrieval. However, our current example contains just P_i , so the system will use it for interpretation.

In this paper, interpretation (i.e., applying a case) means predicting the goals, actions and plans of the observed agent. Predicting the goals and overall plan of the agent is a *global* prediction concerning the entire input event-sequence. Predicting the next action that will occur in the sequence is *local* to the current state and the current event history. This paper restricts itself to discussion and evaluation of only local predictions, although global prediction will be briefly outlined in Section 7.

As will be explained further in Section 4, *adaptation* may or may not occur. Simply knowing the action that will occur next is valuable information, so the system may perform abstract prediction that does no adaptation. The system simply predicts that the action in the old case will be repeated. Thus given P_1 as the only past plan in the example in Figure 3 at the beginning of execution of the current plan P_2 , we predict that an unload action will occur that transitions from the abstract state [3 2 2 1 0] to [3 2 3 0 0].

A concrete prediction that determines what object will be unloaded from what truck includes more information, but it requires adaptation. Especially if the truck had more than one object within it, then a further choice exists as to which object will be unloaded. Given just one object in the truck, we know that it will be unloaded, because humans understand the semantics of the actions. But in our example, the past plan may have included some *ObjectD* in the truck, not *ObjectB* or *ObjectC*. In this case a substitution of variables must be performed to properly adapt the old observation to the new one. In Section 4 we discuss an efficient method to determine a clever substitution. The easiest substitution, however, is to use the object that was in the prior plan. In this case, the system would have successfully interpreted the observation at the abstract level but failed to do so at the concrete level.

As mentioned above, we apply a case (i.e., interpret an observation) by predicting the subsequent action given an observed state. We then evaluate the prediction by comparing it to the action in the following action-state pair observation. Most sections in this paper will include an evaluation result set. Although weighted evaluation schemes are discussed with future research, here we simply report percent accuracy based on whether the predictions were correct or not.

As first described in our previous work and discussed here in Section 5, our system does not operate with a complete plan library, but rather it incrementally builds its library from the observed planning behavior by the *storage* of cases.³² Such incremental construction further decreases the complexity of the state-space and minimizes the existence of extraneous plans.⁴¹

Incremental case base construction implies an interesting tradeoff. Because we start with an empty plan library, all actions in a domain will have an initial observation that cannot be predicted by previous cases because a system cannot predict something of which it knows nothing. For example the dashed past case in Figure 3 predicts a subsequent drive truck in the service of taking more objects to be loaded into planes at the airport. If the system has never seen an instance of a plane flying, then of course no case exists that can predict the fly. This fact decreases the absolute accuracy of overall predictions, yet it allows the system to be robust in the face of novel input.

Throughout this paper we illustrate the algorithms using examples from three common planning domains. These domains include the classic blocksworld, the logistics world, and the extended-STRIPS world. Although the domains are often considered toy domains (especially blocksworld), they are used to present simple examples from which

far more complex problem solutions can be derived. Moreover, we use the PRODIGY non-linear state-space planner to construct test observation sequences of up to 60,000 in length in which to obtain our empirical data.^{15,54,55} Section 6 introduces results from the Extended-STRIPS domain and compares these to the results from the logistics domain to draw some generalities of Incremental Case-Based Plan Recognition. First, however, we use the blocksworld domain to describe case representation and abstract state-space indexing on cases.

2. Case Representation

A major focus of our research is to incorporate the knowledge about states of the observed agent's world into the plan recognition processes. Because agent actions are assumed not to be random, they must be in the service of some set of goals. Therefore the agent's actions can be thought of as a plan, and the plan recognition process is to infer the plan and goals given these actions.

2.1. Concrete states, goals, and plans

Traditionally AI represents a plan as an ordered sequence of actions that transform an initial state into a distinguished goal state. Each action is represented by an operator that contains both preconditions and effects. Preconditions establish applicability criteria, and effects change the state of the world. Furthermore, each action starts in a certain state of the world and results in a change from the current state to another one.

States are represented as sets of instantiated logical predicates (or *ground literals*) that are true. All other states are considered false by the closed-world assumption. Consider the blocksworld state illustrated in Figure 4.⁴ It is represented by the following state.

```
{ (clear B), (on B A), (on A C), (on-table C), (arm-empty) }
```

Only in very limited tasks such as problem-solving for human examinations and game playing, however, is there a clearly defined initial state. Agent behavior is more often a continuous process with no initial state and goals that come and go and change.²² Here we treat all states the same.

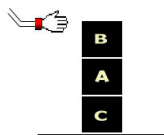


Figure 4. Example of a blocksworld world state.

Goals are traditionally represented as sets of *ground literals*. These are partial representations of the world states in which the truth values of only some of all of the

⁴ The simple blocksworld domain assumes that a single agent (illustrated by a hand, but not explicitly part of any problem) can reconfigure various blocks using the following actions: *pickup* (*OBJECT*), *putdown* (*OBJECT*), *stack* (*OBJECT*, *OBJECT*), and *unstuck* (*OBJECT*, *OBJECT*). Blocksworld problems contain various objects that are always blocks. The problem specifies an initial and a final desired arrangement of the blocks.

possible instantiated state predicates (literals) are specified. For example, a goal from the blocksworld domain $\{(on\ BlockB\ BlockA)\}$ does not include specifications for other state literals (e.g., positions of other blocks).

Both illustrations in Figure 5 depict states of the blocksworld domain in which the partial goal $\{(on\ B\ A)\}$ is satisfied. Contrastingly, the complete state specification for the state on the left of Figure 5 is as follows.

```
{(on B A), (on D C), (clear B), (clear D), (on-table A), (on-table C),
  (arm-empty)}
```

The state on the right is uniquely determined by the following set.

```
{(on B A), (clear B), (clear C), (clear D), (on-table A), (on-table C),
  (on-table D), (arm-empty)}
```

Eleven other world states, not depicted here, also satisfy goal $\{(on\ B\ A)\}$.



Figure 5. Two states that achieve the goal $\{(on\ B\ A)\}$.

We offer a novel alternative representation of the traditional view of a plan. A *plan* is a sequence of state changes, where the state changes are caused by the actions the agent performs.⁵ We represent the plan (i.e., a case) as an arbitrarily long sequence of action-state pairs. Each pair encodes both the action and the new state that results from an action. Note that using this representation, the initial pair in a plan always contains the *null* action that “results” in the first observed state, and, the final pair contains the action that results in the goal state.

A full state representation of a goal has another advantage in both case-based planning and plan recognition. A case-based planner may not be able to successfully retrieve a solution to the current problem when only partial goal state predicates are specified. For example, if we assume that the planner encounters a goal specification $\{(NOT\ (clear\ BlockA))\}$ for the first time, matching this goal state with $(on\ BlockA\ BlockB)$ would fail, even if plans whose goal states are shown in Figure 5 were already observed and stored in the case-base. However, both states in Figure 5 implicitly contain state predicates, which define a goal state for the newly seen problem; storing all of the state predicates for goal states would yield successful matches.

Although knowledge about every world state is beneficial for plan recognition, the number of possible states in the complete state-space may be quite large. The exact number depends on the domain theory as well as on the instances of objects a particular problem definition may entail. To gain understanding of how large the state-space may be, let s denote a state represented as a collection of m predicates, that is

⁵ Although world state changes in the real-world systems can be caused by exogenous events and multiple agents we do not deal with these conditions here.^[22]

$$s = \bigcup_{i=1}^m p_i$$

A predicate p_i is represented by a tuple

$$p_i = (n_i, a_{i,1}, \dots, a_{i,l_i})$$

where n_i is the name of the i -th predicate and $a_{i,j}$ are the arguments of the i -th predicate with a total of l_i arguments. Each predicate in the context of state representation essentially represents a function b that assigns a truth value to a predicate, that is

$$b: p_i \rightarrow \{TRUE, FALSE\}.$$

As we saw in the previous chapter, this research operates under a closed-world assumption, where if a predicate is not known to be *true*, then it is assumed that the truth-value of a predicate is *false*. Thus a state representation contains only predicates whose truth values are true given a world state. Therefore, we formally have the following state representation

$$s = \bigcup_{i=1}^m p_i \mid b(p_i) = TRUE.$$

Let $args(p_i)$ denote a set of all arguments of a predicate p_i , that is

$$args(p_i) = \bigcup_{j=1}^{l_i} \{a_{i,j}\},$$

and let $Card$ be a function that returns a cardinality of an input set. Given a state s , let

$$P_s^k = \{p_i \mid p_i \in s \wedge 1 \leq i \leq m \wedge Card(args(p_i)) = k\}$$

be a possibly empty set of all predicates in the given state that have arity k (i.e., predicate has k arguments). Each predicate p has arguments of certain types from the type hierarchy of the domain theory, and predicates instantiate into tokens of appropriate types at planning time.

The representation of state-space that accounts for the complete states may be quite complex. Let n be the number of different object instances for a given problem. To account for the worst-case scenario, assume that all n instances are of the same type. Given a state s and

$$P_s^0, P_s^1, \dots, P_s^q, \text{ where } q = \max(Card(args(p))), \forall p \in s,$$

the length of a feature vector representing a state with respect to a given problem without any loss of state information in the state-space is

$$|\vec{F}| = |P^0| + n|P^1| + n^2|P^2| + \dots + n^q|P^q| = \sum_{i=0}^q n^i |P^i|$$

and the size of the state-space with respect to a given problem

$$|SS| = 2^{|\vec{F}|}.$$

In the blocksworld planning domain, the maximum number of arguments in any state predicate is two. The sets

$$P^0 = \{\text{(arm - empty)}\}, P^1 = \{\text{(clear obj)}, \text{(on - table obj)}, \text{(holding obj)}\}, P^2 = \{\text{(on obj obj)}\}$$

represent examples of predicate sets in the blocksworld domain. In this planning domain, given only $n=3$ different blocks with which to work, the feature vector size is $3^0(1) + 3^1(3) + 3^2(1) = 19$, and the state-space size is $2^{19} = 524,288$. Therefore, approaches based on pruning a fully constructed state-space are computationally expensive, because the state-space size prohibits efficient pruning.²⁵ Such approaches are promising only if they are able to prune large regions of the state-space.

In the next section, we describe a scheme that is able to transform the observed states into their abstracted representation, retaining the knowledge about the structure of a concrete world state. The abstracted representation of world states allows us to focus the plan recognition process on the relevant features of the states, within much smaller abstract state-space.

2.2. Abstract states and case indexing

To increase indexing efficiency, we employ a simple abstraction scheme in which observed states of the world correspond to type-generalized state predicate representations.^{33,35} The abstract state representation is a non-negative integer vector in which each dimension represents a number of instances of a single type-generalized state predicate.

For instance, consider the blocksworld state depicted in Figure 6. The picture shows two block towers of height two. They are represented in a concrete predicate form below

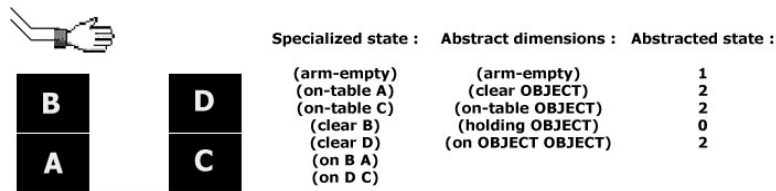


Figure 6. An example of a blocksworld state and its representations.

the label called specialized state. The abstract dimensions in the blocksworld are shown next. Given this, the concrete state is represented by the vector [1 2 2 0 1], because the

arm-empty predicate is true (the dimension is one), two objects are *clear*, two *on-table* predicates are true, the *holding* predicate is not true (the dimension is zero), and two *on* relations exist between two separate objects.

To understand the representation of a plan that leverages this representation consider Figure 7. It shows an initial (first observed) state, an intermediate state, and a goal state. To get from the first state to the last, an agent picks up *BlockA* and stacks it on *BlockB*. The concrete plan is therefore represented by the sequence of three pairs (*null*, $s_{1,1}$), (*pickup (BlockA)*, $s_{1,2}$), and (*stack (BlockA, BlockB)*, $s_{1,3}$).

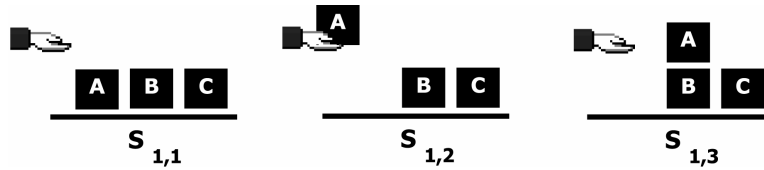


Figure 7. A sequence of three concrete blocksworld states, $s_{1,1}$, $s_{1,2}$, and $s_{1,3}$ for plan $P1$.

Abstract states constitute vertices of the abstracted state-space graph, in which an edge from one abstract state to another indicates the observation of an action that changed the former world state into the latter one. Each case can be abstracted to represent a path in the abstracted state-space graph. The three concrete states are represented in this abstract space by the vectors $[1\ 3\ 3\ 0\ 0]$, $[0\ 2\ 2\ 1\ 0]$, and $[1\ 2\ 2\ 0\ 1]$.

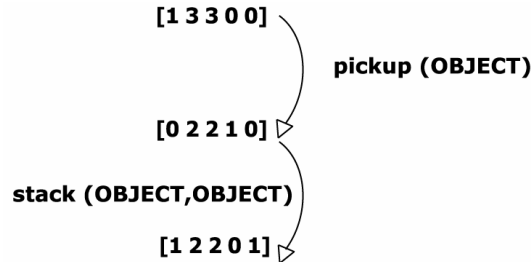


Figure 8. The abstract state-space graph for the plan illustrated above.

Using these vectors, the abstract plan can be represented by the following sequence of pairs.⁶

```
(null, [1 3 3 0 0])
(pickup (OBJECT), [0 2 2 1 0]),
(stack (OBJECT OBJECT), [1 2 2 0 1]).
```

The abstract-space graph for this path is then shown in Figure 8 above.

Now consider Figure 9. It introduces two additional concrete states from which the goal state $s_{1,3}$ in Figure 7 can be reached. From the initial state on the bottom right within Figure 9, an agent can unstack *BlockC*, put it down, then pickup *BlockA* and stack it on *BlockB*. Let us call this plan $P2$. Alternatively from the initial state on the top right in the

⁶ In the current implementation of the recognition system, the abstract plan is represented implicitly. An explicit representation will be necessary in future research.

figure, an agent can unstack *BlockA* from *BlockC* and then stack *BlockA* on *BlockB*. Let us call this plan *P3*. This now presents us with three plans that result in the same goal state, *P1*, *P2*, and *P3*. These concrete plans are shown pictorially in Figure 9. The state of a given plan P_i will be called $s_{i,j}$ where i indicates the plan number and j is the location in the plan the state occupies from $j = 1$ to $j = \text{length}(\text{plan}_i)$.

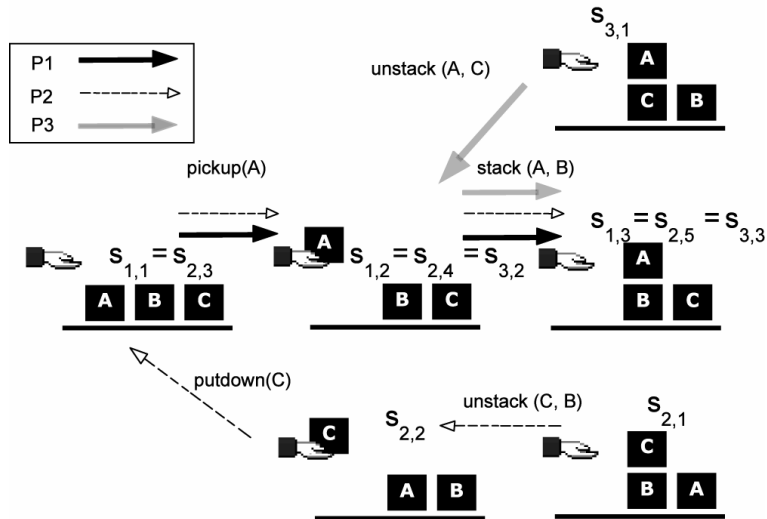


Figure 9. Plans $P1$, $P2$, and $P3$ in concrete representation.

Note that a number of similarities exist between the three plans. Of course all three share that same final state. Indeed, they all share the last plan step (i.e., to stack *BlockA* on *BlockB*) which transitions the next to the last state in all plans to the goal state. Plan *P2* shares the last two steps in the plan, so *P1* is a sub-plan within *P2*. Given these similarities, they differ in that all three plans start from different initial observations. Note also that each state above or below another state in Figure 9 is structurally the same, differing only by the labeling of the blocks.

Now let us consider the abstract representation of the three concrete plans. Plan *P1* is shown in the abstract space in Figure 8. Figure 10 is an abstract graph representation of plans *P2* and *P3* in addition to *P1*. *P1* is shown in solid arrows, *P2* with dotted arrows, and *P3* with dots and dashes. Boxed “I”s represent initial states of the depicted plans.

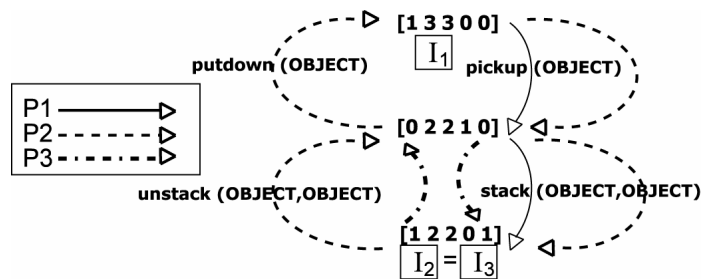


Figure 10. Plans $P1$, $P2$, and $P3$ in abstract representation.

Remember that the first pair in plans $P1$, $P2$ and $P3$ in the plan library starts with the null action. Note also that each state $s_{i,j}$ represents an actual concrete state from Figure 9. In the figure we see that a bin contains one “column” of states in Figure 9. A number of these concrete states are equal and a number are not. As mentioned above, the differences are a variation of the labeling of blocks in the state. However, as will be discussed in Section 4, Adaptation, it is not an easy computational task to determine which states are identical.

A world state is indexed by its abstracted state representation. Each abstract state points to a set of concrete world states (i.e., ground literals) indexed by it. The sets are called bins, and all states in them have the same abstract vector value. The states within the bins in turn point to the cases that contain them. To understand this key relationship among abstract and concrete plans and states, see Figure 11. This is a particular version of the more generalized figure (Figure 2) from the introduction.

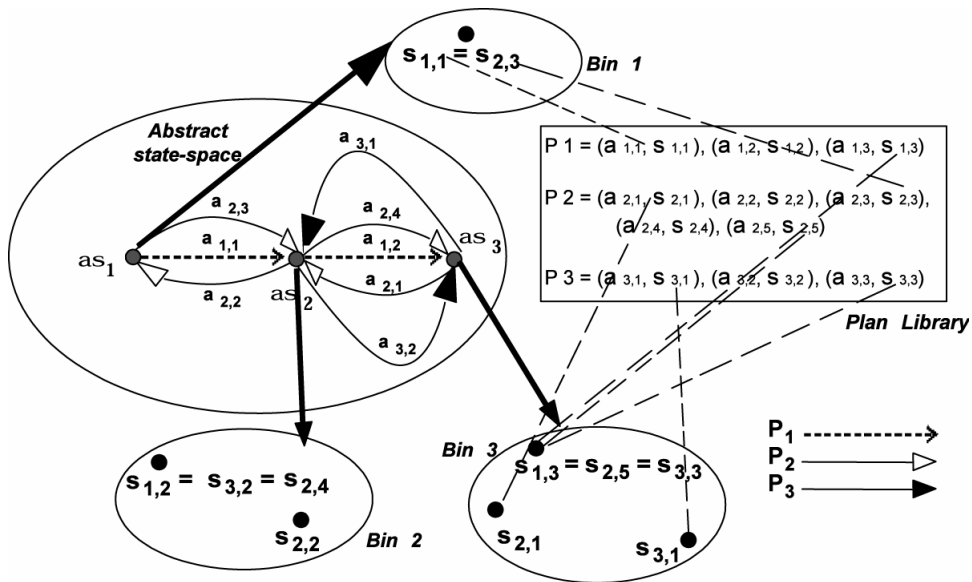


Figure 11. Indexing and storage structures for plans $P1$, $P2$, and $P3$.

Worse still, states that map to the same abstract state may not even be structurally the same. Given four blocks for example, one structural configuration is to have one tower of three blocks with a lone block on the table, whereas another configuration is to have two towers each consisting of two blocks, one on another. Noting that both have two clear blocks, two blocks on the table, and two instances of one block stacked on another, both of these arrangements map to the same abstract state. If we could distinguish between states that are structurally the same and those that are not, then a system could eliminate many cases in a bin that are irrelevant for the interpretation of observations. As it turns out, we can accomplish this by partitioning the bins further by utilizing equivalence classes formed according to the given equivalence relation. Although complete structural identity provides a useful equivalence relation for analyzing bin distributions, it is computationally intractable to compute. As we will see in Section

3, however, a weaker form of this equivalence relation exists that provides a practical means for reducing a case retrieval set. Instead of focusing on past situations in a single bin, the recognizer can focus on a specific equivalence class within the matched bin and eliminate form consideration situations that lay outside of the matched equivalence class.

2.3. State-space characteristics

The abstraction scheme presented here provides a means of rapid access to the concrete world states indexed by their abstract representation. It is a one-to-many relation, because an abstract state may index several concrete states. The result is a smaller set of first level indexes, which ensures similarity at the level of concrete states. The exact space savings of such an abstraction scheme depend on the domain theory. In general, the number of possible abstract states for a given domain theory is

$$\prod_{i=1}^L (m_i + 1)$$

where L is the number of abstract dimensions (i.e., length of the abstract vector), and m_i is the maximum possible value in abstract dimension i . Given n object instances, abstract dimensions corresponding to i -ary predicates may possibly have values from 0 to n^i . For example, in the case of the blocksworld domain, the abstracted state feature vector size is five (i.e., the number of abstract dimensions), and number of possible abstract states for ten object instances is 48,884. This represents a considerable savings when compared to the number of all possible concrete states (see Section 2.1). However, not all of these states are possible; in the blocksworld domain, there cannot be more than $n-1$ instances of the *on* state predicate, because no more than $n-1$ blocks can be stacked directly on top of other blocks. Also, the only possible values for the abstracted *holding* state predicate are zero and one, because the domain theory restricts resources to only one robot arm that can hold the blocks. Effectively, this reduces the possible abstract state-space to 4,840 possible states. Some of the remaining abstract states are still not possible. For example, abstract state that has a nine as a value of *on* state predicate dimension, cannot have more than a value of one in either *clear* or *on-table* dimensions (assuming ten object instances). The incremental construction of the case library (see Section 5) assures that the number of both abstract and concrete states actually stored is minimal.

To experimentally determine the characteristics of a state-space, we now present an empirical analysis of the case indexing structures (i.e., bins and equivalence classes) created during the plan recognition process. To do so we return to the logistics domain discussed in the introduction. For this domain, the recognizer observes several thousand plans consisting of tens of thousands of action-state pairs. We generate test observables by supplying multiple problems to the Prodigy 4.0 planning system using a nonlinear planning mode and using a search time bound of 16 seconds.⁷ The planning problems that served as the input into PRODIGY are generated randomly from a problem generator (see Section 6.1 for details). The problem generator produces two separate sets of domain observables using two seeds to the random number generator. Extended-STRIPS

⁷ If Prodigy 4.0 did not produce a solution to a particular problem in the allotted time, we discarded the problem. This limits the observations to relatively simple planning problems (i.e., those that can be solved with the search algorithm in Prodigy 4.0).

observables are generated using the same two seeds. All empirical results use this method. Results reported for any problem domain are the averages of the two separate problem sets.

As we can see from Figure 12, the number of abstract states observed is much smaller than the number of concrete states, and the rate at which new abstract states increase is also considerably lower. This indicates that the abstract state-space (represented by the bins) is much smaller than the corresponding concrete state-space. As discussed briefly in the previous section, and in more detail in the subsection 3.2, our system further partitions the elements in a bin according to the given equivalence relation. Bins are then partitioned into equivalence classes, in which all states are identical under the given equivalence relation. The number of equivalence classes falls between the bin and concrete numbers. This is as expected, because bins are partitioned by the equivalence classes. If all bins had just one equivalence class, then the dashed curve would be equal to solid curve.

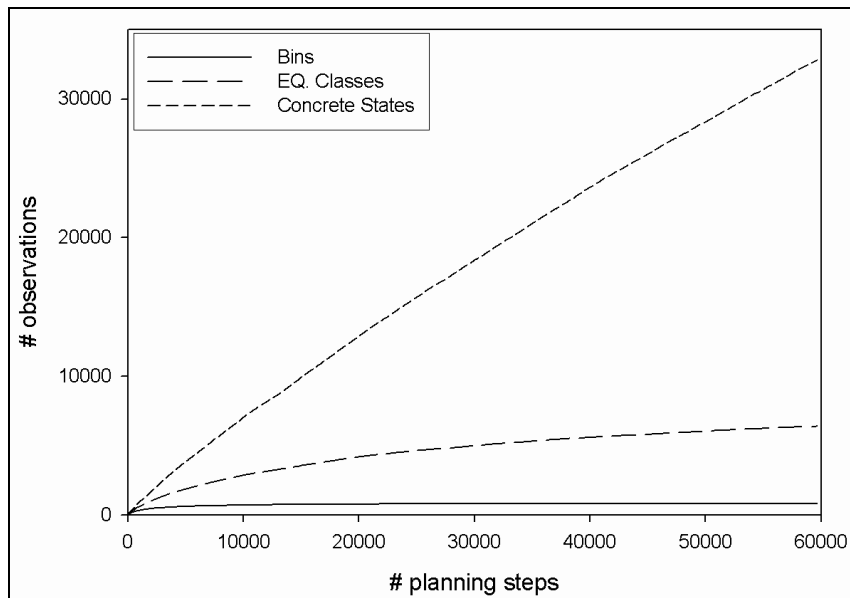


Figure 12. The number of abstract states (bins), equivalence classes and concrete states in the logistics planning domain

The logistics domain is characterized by a relatively small state-space, especially when dealing with a 3-city domain configuration. Contrastingly, the state-space for the extended-STRIPS domain discussed in later sections of this paper is quite large, even in the case of a 4-room domain configuration. Because logistics recognition problems arise from a relatively small state-space, the recognizer will be able to explore most of the 3-city logistics state-space after about 60,000 observed planning steps. These results are very encouraging. In Section 6, we experimentally determine the asymptotic abstract state-space behavior for the extended-STRIPS domain and compare it with the logistics domain data here.

3. Retrieval and Case Elimination

Given an input observation, the retrieval task of an incremental case-based plan-recognition system is to find a past observation that is the most likely candidate to provide an accurate prediction of the subsequent observation. To perform this task, a case-based interpreter will first retrieve a set of matching cases and second reduce the retrieval set to a small number of most similar cases with which to interpret the observation.⁸ Our algorithm accomplishes these two tasks by determining an abstract bin that serves as the retrieval set and then eliminating candidates to a potentially smaller set of past cases represented by a single class within a bin (see Figure 13).

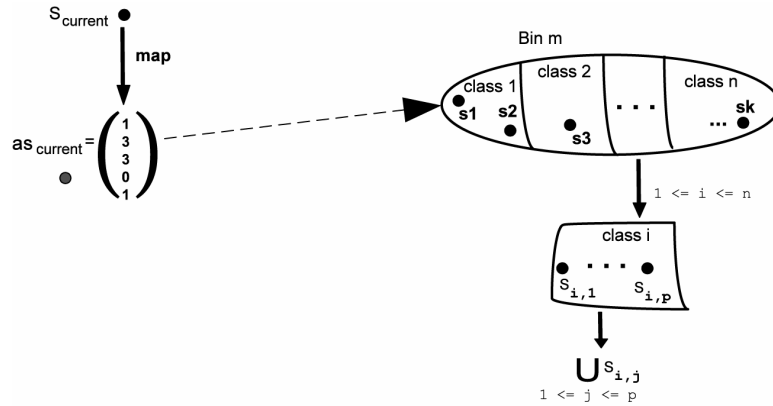


Figure 13. Index selection and past situation retrieval

3.1. Index selection and case retrieval set

To determine the abstract bin corresponding to a given observation, a concrete state must map to an abstract vector (index) such as those described in Section 2.2. This mapping is shown in the left side of Figure 13 and represents an index selection. The mapping is calculated in linear time that is proportional to the concrete state size. Figure 14 sketches the algorithm to perform the mapping.

Let $s = (p_1, \dots, p_q)$ be a collection of ground literals representing a concrete state s with $p_i = (n_i a_{i,1}, \dots, a_{i,q_i})$, where n_i is the name of i -th predicate p_i , q_i is the number of arguments of p_i , and $a_{i,j}$ are the arguments of predicate p_i . Furthermore, let L be the length of the abstract vector, and let $DIM = (dim_1, \dots, dim_L)$ be a list of predicate names corresponding to each dimension of the abstract vector. Let m, k be integers. Given a currently observed concrete state s , construction of abstract state A_s proceeds as illustrated in the left portion of Figure 13. Figure 14 shows the algorithm *MAP INTO ABSTRACT STATE* that maps concrete states into their abstract representations.

⁸ Most case-based reasoners choose a single case to use (as do we), however, some will use multiple cases to perform reasoning. For example see Veloso's case merging strategy.⁵² This method uses PRODIGY in case-based mode (i.e., Prodigy/Analogy) for illustrating the merge algorithm in planning tasks.⁵³

```

int[] MAP_INTO_ABSTRACT_STATE (State s)

  for  $1 \leq m \leq L$ 
     $A_s[m] = 0$ 
  end for

  for  $1 \leq k \leq q$ 
     $P_{current} = P_k = (n_k a_{k,1} \dots a_{k,qk})$ 
    int index = indexof( $n_k, DIM$ )
     $A_s[index] = A_s[index] + 1$ 
  end for
  return  $A_s$ 

```

Figure 14. Map a concrete state to an abstract state.

The function *indexof* simply returns the position of its first argument in a list represented by its second argument. The algorithm returns A_s , a non-negative integer vector that is the abstract representation of a given input concrete state.

Now given the abstract state as a result of the mapping process, it serves as an index into the case base. The index points to a bin that represents the *retrieval set* of candidate cases. The index is implemented by a hash table that maps vectors to bins. Figure 13 illustrates this by the dashed arrow connecting the left and right sides of the figure. Because we utilize hash tables for indexing, this scheme allows us to retrieve the initial set of matching cases very efficiently.

Some times no abstract state will exist for a given concrete state, because an example of that state has not been seen. That is, no situation similar to the current one has been observed in the past. If so, the recognizer will not be able to retrieve any cases, and therefore it will be unable to make a prediction. As will be explained in Section 5, the current observation will be stored in the case library so that prediction can be made in the future given a similar observation. It will be able to do this because the future state will map to the abstract state under which the current observation is indexed.

3.2. Retrieval set reduction

One of the main problems of the abstract indexing scheme introduced in the previous section, with respect to the retrieval efficiency, is the number of concrete states within the bins. Clearly there may exist some bins that contain a large number of concrete states, which in turn may point to a large number of past cases in which they are contained. Because the recognizer predicts the planner's intentions based on the intentions from the previous cases, the number of possible prediction candidates for a given situation may be quite large. Choosing the most frequently pursued prediction is an effective way to cope with a large number of possible predictions. However, there still may exist situations that have many actions that are pursued with equal frequency, resulting in the saturation of the bins with concrete states and decreased retrieval efficiency.

A technique that can be used to cope with a large number of concrete instances in a single instance class is to create new hierarchical levels of representation by extracting the common features between instances and then indexing on the differences. For each differing attribute, the instances are then split into subsets depending on the values of the attribute. Then for each subset the process is recursively repeated. After reaching a size

threshold that constitutes a halt condition, subsets of instances that share common features will exist in a new structure.³⁷

The resultant hierarchical structure is a discrimination net. In the context of the plan recognition system presented in this work, bins represent classes and world states are the concrete instances. The discrimination net could then be used with the current state to find smaller sets of cases within a bin. This technique has been utilized in an early case-based interpreter called CYRUS.³⁷

In this paper we investigate a slightly different approach from the indexing techniques of CYRUS. Instead of creating a hierarchical net that contains overlapping sets of states (i.e., many states may appear in multiple leaves of the net), we partition the bins into mutually exclusive containers of concrete states. These containers thus represent a second level of indexing. Any equivalence relation would serve this purpose, as it naturally partitions the set of all of its elements into disjoint subsets. One obvious equivalence relation we could utilize is an exact semantic equivalence of concrete states. In this case, however, each equivalence class would contain a single state, and each bin would contain as many equivalence classes as the number of states within that bin.

3.2.1 Equivalence classes

The process of the equivalence class creation utilizes a change of representation.^{5,24} During representational change, world states represented by ground literals are transformed into corresponding *state graphs*.^{32,34} This representation is graph-based. The states are represented as directed graphs with loops. These graphs are called *directed nets*. As before, let state s be a collection of m predicates, that is

$$S = \bigcup_{i=1}^m p_i \cdot$$

A predicate p_i is represented by a l_i+1 tuple

$$p_i = (n_i, a_{i,1}, \dots, a_{i,l_i})$$

where n_i is the name of the i -th predicate and $a_{i,j}$ are the arguments of the i -th predicate with a total of l_i predicate arguments. Let $args(p_i)$ denote a set of all arguments of a predicate p_i , that is

$$args(p_i) = \bigcup_{j=1}^{l_i} \{a_{i,j}\} \cdot$$

Furthermore, let p^0 denote a set of names of all predicates that have no arguments,

$$p^0 = \left\{ \bigcup_{i=1}^m n_i \mid args(p_i) = \emptyset \right\}$$

The vertices of a graph representing a world state S can now be defined as

$$V(G_S) = \left\{ \bigcup_{i=1}^m \text{args}(p_i) \cup p^0 \right\}.$$

A directed edge from one vertex to another one indicates that the former appears before the latter in the ordered list of arguments of the same predicate. Formally,

$$e_{v_j, v_k} \Leftrightarrow \exists p_i \mid v_j \in \text{args}(p_i) \wedge v_k \in \text{args}(p_i) \wedge j \leq k$$

Figure 15 shows a pictorial example of a representational change from a predicate representation consisting of ground literals into the state graph. Notice that edges can be loops when a state predicate has only one argument. The notion of edges can be extended to the cases where state predicates have more than two arguments. By applying a hypergraph representation an edge can be adjacent to more than two vertices. The current

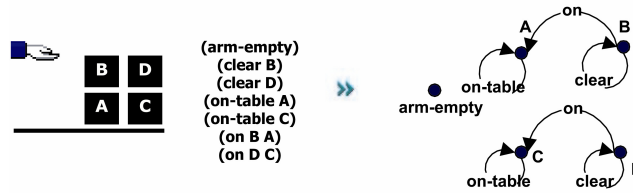


Figure 15. An example state graph representation from the blocksworld planning domain.

implementation of the recognition system supports the state predicates with more than two arguments and state graphs that are hypergraphs with three or more dimensions.

In Section 4 we will see that the state graphs help in the determination of the argument substitutions during adaptation. The change of representation is also important in determining whether a single indexing bin contains *structurally* different states. To illustrate this, consider Figure 16. It describes two structurally different world states having the same abstract representation and thus belonging to the same bin. State abstraction alone is not able to capture the differences among structurally different world states in the same bin. However, once the states in the same bin are transformed into their corresponding state graphs, the structural differences among them can be captured.

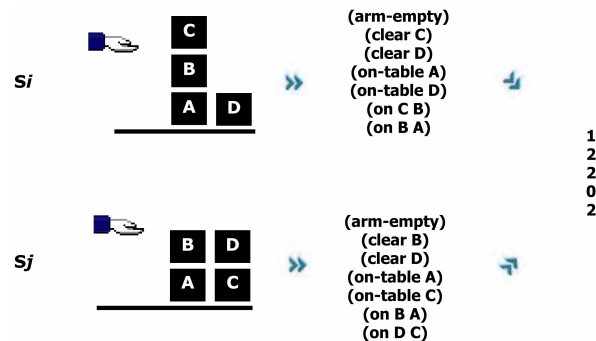


Figure 16. An example of two structurally different world states s_i and s_j with identical abstract representations.

Essentially, two states will be structurally identical if and only if their corresponding state graphs are isomorphic.³² The graph isomorphism is a one-to-one and onto mapping that maps the vertices of one graph onto the vertices of another graph while preserving the edges. Figure 17 shows the two states from Figure 16 in the state-graph representation; it can easily be seen that the two graphs are not isomorphic.

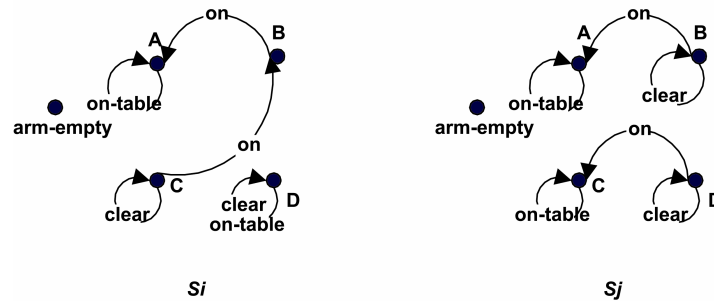


Figure 17. State graphs for states s_i and s_j from Figure 16 with identical abstract representations.

Transformation of a world state in the predicate form into its state graph is linear in the number of vertices and edges of the state graph. Given such little overhead, the benefits introduced by this representational change have a potential to focus the retrieval of past situations to a smaller subset of relevant world states. This is because world states within the same bin can be further separated into groups of structurally equivalent states through an equivalence relation. Graph isomorphism is an equivalence relation that naturally partitions the bins into equivalence classes containing only structurally equivalent states. By comparing the two state graphs at retrieval and storage time, the recognizer can determine whether they are isomorphic and thus structurally equivalent. Moreover, the number of graph comparisons needed to locate an equivalence class may be reduced, because all members in a single equivalence class are considered to be identical under the given equivalence relation. Comparing the current instance to a single member of an equivalence class (called *class representative*) is sufficient to determine equivalence class membership.

3.2.2 Pseudo-isomorphism and equivalence classes

The main problem with equivalence class groupings based on the state graph isomorphism equivalence relation is that the complexity of the graph isomorphism problem may be too large for the recognizer to handle. The computational complexity of the graph isomorphism problem is unfortunately not known. It is not proven to be an *NP*-complete problem, and it is also not known to be in the class *P*. This problem is often speculated to be in the intermediate complexity class *NPI*, particularly because both the counting and decision versions of this problem are equally hard, a property not shared by *NP*-complete problems. However, polynomial-time graph isomorphism algorithms exist for several classes of graphs. Examples are graphs with bounded degree, trees, and partial *k*-trees.^{43,1,11} Most of these polynomial isomorphism algorithms are, however, not applicable in practice because of large constant multipliers. The exceptions are planar graphs and circular-arc graphs for which efficient isomorphism algorithms exist.¹⁸

It is possible to utilize knowledge about the domain theory to assure that the equivalence relation based on graph isomorphism is feasible. When all state graphs in a given planning domain belong to a certain class of graphs for which efficient isomorphism algorithms exist, then equivalence classes can be efficiently used to group similar past situations inside the bins. If we consider the logistics planning domain without the static state predicates, then the state graphs are always planar. Finding isomorphisms for planar graphs is efficient, and it may help focus the retrieval of past situations in the logistics domain. However, this approach does not generalize to other planning domains, because the state graphs of these domains may not all be planar.

To formulate a practical equivalence relation for partitioning the bins, we introduce a pseudo-isomorphism scheme based on the isomorphism mapping of the state graphs. This process utilizes the information contained in the transformed state graphs, much like the adaptation process described in Section 4. To explain the process that determines the pseudo-isomorphic equivalence of two concrete states, consider an example with two states s_1 and s_2 and their corresponding state graphs G_1 and G_2 , respectively. Let

$$G_1 = (V_{G_1}, E_{G_1}), G_2 = (V_{G_2}, E_{G_2}), \text{ with } V(G_1) = \{v_1, \dots, v_{k_1}\}, V(G_2) = \{u_1, \dots, u_{k_2}\}.$$

Let us assume that state s_1 is the class representative of the equivalence class in question, while state s_2 is the observed state that needs to be put into an appropriate equivalence class. Given two state graphs G_1 and G_2 , Algorithm for the method *EQUAL_UNDER_PSEUDO-ISOMORPHISM* that decides whether or not two state graphs are pseudo-isomorphic is specified in Figure 18.

```

Boolean EQUAL_UNDER_PSEUDO-ISOMORPHISM (G1, G2)

  if k1 ≠ k2
    return FALSE
  else
    k = k1 = k2
  Set G1Connections = ∅, G2Connections = ∅
  for each vertex ui ∈ VG1, vi ∈ VG2, 1 ≤ i ≤ k
    String connectionsG1 = getSortedConnections(G1, ui)
    String connectionsG2 = getSortedConnections(G2, vi)
    G1Connections = G1Connections ∪ connectionsG1
    G2Connections = G2Connections ∪ connectionsG2
  end for

  for all G1Conn ∈ G1Connections
    if G1Conn ∈ G2Connections
      G2Connections = G2Connections - {G1Conn}
    else
      return FALSE // match not found!
  end for

  if G2Connections = ∅
    return TRUE
  else
    return FALSE

```

Figure 18. The algorithm for *EQUAL_UNDER_PSEUDO-ISOMORPHISM* method.

The algorithm *EQUAL_UNDER_PSEUDO-ISOMORPHISM* starts by checking if the two input graphs have vertex sets of equal sizes. If not, then they cannot be pseudo-

isomorphic. Otherwise, we call method *getSortedConnections* for each vertex in the two input vertex sets. Strings representing resulting connections are then placed in sets *G2Connections* and *G1Connections* for vertex sets of state graphs G_2 and G_1 , respectively. The second *for* loop considers each produced string in *G1Connections* set and attempts to find a matching string in *G2Connections* set. If such match cannot be found, then no pseudo-isomorphism exists between the two input graphs. Otherwise, we remove the found matched string from *G2Connections* set and continue until we have considered every string in set *G1Connections*. At the end, we need to check if *G2Connections* set is empty. If this set is not empty, then there exist edges in the input graph G_2 that are not present in the input graph G_1 . Otherwise, the input graphs are pseudo-isomorphic and thus in the identical equivalence class under pseudo-isomorphism relation.

The algorithm for the method *getSortedConnections* is depicted in Figure 19. The method *getSortedConnections* starts by initializing the return *connections* string to an empty string. Next, we create the state graph G' , which is a subgraph of the input state graph G induced by the input vertex v . For each edge e_i' in the induced subgraph G' , we produce an incidence set *incidencySet* that contains all vertices from the subgraph G' incident to the edge e_i' . Because state graphs contain directed edges, the vertices in the *incidencySet* are ordered by the order of their adjacency to the edge e_i' . Note that such ordering allows state graphs to be hypergraphs, where an edge can be adjacent to more than two vertices. We then take the label of the edge e_i' , concatenate it with the index of the vertex in the *incidencySet* produced by the method *getVertexPosition*, and add it to the connection string.

```
String getSortedConnections (G, v)

String connections = ""
StateGraph G' = inducedSubgraph(G, v)
V(G') = {v1', ..., vt'}, E(G') = {e1', ..., es'}
for each edge ei', 1 ≤ i ≤ s
    String edgeLabel = getEdgeLabel(ei')
    incidenceSet {v1', ..., vt'} = getIncidencySet(ei')
    int j = getVertexPosition(incidencySet, v)
    connections = connections + edgeLabel + (String)j + " "
end for
connections = Sort(connections)
return connections
```

Figure 19. The algorithm for *getSortedConnections* method.

To illustrate the above algorithms on an example, consider Figure 20 that shows two blocksworld states. In order to provide the input into the algorithm *EQUAL_UNDER_PSEUDO-ISOMORPHISM*, the observer first needs to transform the states into corresponding state graphs. The state graphs are shown in Figure 21.

The algorithm first ensures that both graphs have vertex sets of the same size and then it initializes sets *G1Connections* and *G2Connections* to empty sets. Next the algorithm obtains the *connectionsG1* and *connectionsG2* strings by calling the *getSortedConnections* method for each vertex in each of the two graphs. The *connection* strings produced for the given vertices are shown in Table 2.

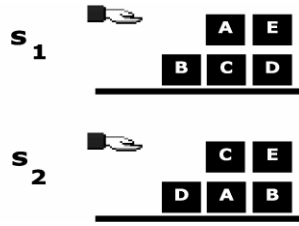


Figure 20. Two blockworld states s_1 and s_2 .

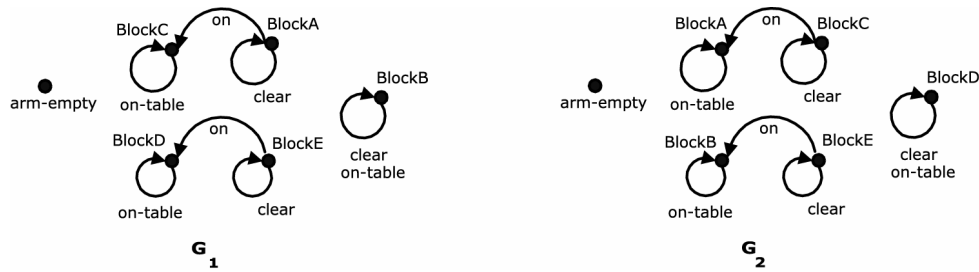


Figure 21. State graphs G_1 and G_2 for the states s_1 and s_2 from Figure 20, respectively.

Table 2. The *connection* strings produced for each vertex in G_1 and G_2 via *getSortedConnections* method.

G_1		G_2	
vertex	connections	vertex	connections
arm-empty	null	arm-empty	null
BlockA	clear1 on1	BlockA	on2 on-table1
BlockB	clear1 on-table1	BlockB	on2 on-table1
BlockC	on2 on-table1	BlockC	clear1 on1
BlockD	on2 on-table1	BlockD	clear1 on-table1
BlockE	clear1 on1	BlockE	clear1 on1

At this point in the algorithm, the observer has constructed the following two sets:

$$G1Connections = \{\text{null}, \text{clear1 on1}, \text{clear1 on-table1}, \text{on2 on-table 1}, \text{on2 on-table 1}, \text{clear1 on1}\}$$

$$G2Connections = \{\text{null}, \text{on2 on-table 1}, \text{on2 on-table 1}, \text{clear1 on1}, \text{clear1 on-table1}, \text{clear1 on1}\}$$

The last step in the *EQUAL_UNDER_PSEUDO-ISOMORPHISM* algorithm is to attempt to match elements in the above two sets. We can see that clearly there exists a match in the set $G1Connections$ for every element of the set $G2Connections$. Therefore, the algorithm returns *true* answer for this particular example, indicating that the two input state graphs are indeed pseudo-isomorphic.

The procedure described above allows the recognizer to efficiently determine whether two concrete states are identical under the pseudo-isomorphic equivalence relation. The process is linear in the number of edges and vertices in the state graphs. As soon as the edge incidence set of some vertex in G_2 does not match with any such set in the graph G_1 , the process returns with a negative answer. It also may be the case that the answer returned by the above process is positive when in reality two state graphs are not isomorphic. The simplicity of the proposed equivalence determination process, however, provides the recognizer with an efficient way to further partition the bins into containers of mutually exclusive elements under the given equivalence relation.

Using the proposed equivalence relation, the recognizer can now focus its attention to a subset of the bin represented by an equivalence class. Because all of the states in a single equivalence class are identical under the pseudo-isomorphic relation, the worst-case number of comparisons that the recognizer must consider to determine the appropriate equivalence class is equal to the number of equivalence classes within the matched bin. In situations where every equivalence class contains a single element (e.g., complete semantic equality equivalence), the recognizer still must consider all states within the bin in the worst-case scenario. However, we saw in Section 2 that such situations are rare and that the equivalence classes further decrease the size of the subset of the state-space on which the recognizer focuses its reasoning.

The equivalence classes serve as the second-level indices into the case-base. Once the appropriate bin is identified, the recognizer transforms the current state into a corresponding state graph and then compares the graph to all class representatives in the matched bin. If a match is found, the recognizer retrieves the matched equivalence class along with all concrete states it contains for further processing. Otherwise, the recognizer will fail to retrieve any concrete states and will have to wait for future occurrences of such a situation in order to be able to form predictions of the planner's future intent.

3.3. Empirical evaluations

Once a mapping to an abstract state occurs, and an equivalence class in the matched bin is identified, some method must be used to reduce the number of concrete states in the matched equivalence class within a bin that comprise the candidates for interpretation. Once a single state s_i is produced, the local prediction is simply a_{i+1} . The simplest method to produce s_i is to randomly eliminate all but one state from the bin and output that one. We will call this heuristic the *random elimination strategy* (RE). Another solution is to choose the state in the bin that occurs in the highest number of cases. We will call this method the *most frequent strategy* (F).

To compare these two strategies empirically, we accumulate successful predictions across 60,000 observations generated by random 3-city logistics problems. All differences among the prediction accuracies for all subsequent empirical results were analyzed using one-way ANOVA test followed by the Scheffe's test with significance level $\alpha=0.05$. A prediction can be successful in two ways as discussed in the introduction. An abstract prediction occurs if the algorithm can predict next action at an abstract level. To do this it must only specify the action itself. The abstract arguments add no information and are given by the domain definition. As argued in the introduction, knowing just the subsequent action type is useful even if the arguments to the action remain unknown. An example is that a system may predict a truck will load two objects without knowing which objects these will be.

Figure 22 shows the results of both RE and F strategies with respect to a *baseline* (B) performance. The baseline is simply formed by picking an action at random from the pool of all previously observed actions and taking the action name (type) as the abstract prediction. No retrieval or cases are necessary for this calculation. The results show that over time the recognizer will perform successfully about 35% of the time in the logistics domain. That is, over 17,000 predictions out of 60,000 will be correct, despite the fact that in early stages the case base is extremely sparse. This is more than three times the rate of the baseline performance.

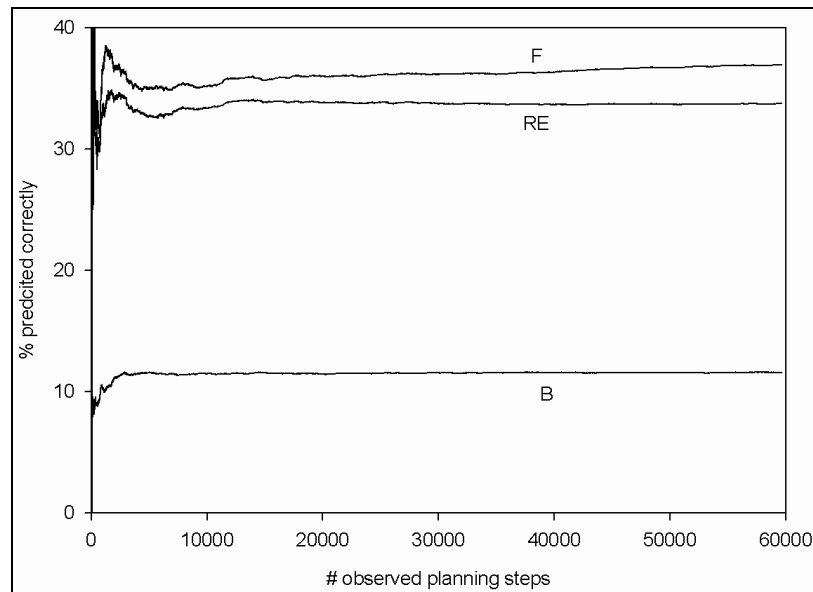


Figure 22. Percentages of correctly predicted *abstract* actions in the *logistics* domain under the baseline (B), random elimination (RE) and the most frequent (F) recognition strategies.

A second measure of performance is to predict both an action and the arguments to the action. That is, rather than just predicting what action will occur next (e.g., load some truck), the recognizer must predict a concrete action instead (e.g., loading a particular truck with a specific object). Results using this measure for the same observations included in the Figure 22 are shown in Figure 23. Here the baseline is computed by randomly choosing an action from the pool of all previously observed actions. Such a selection is biased, because the actions that occur with the highest frequency have a higher probability of being chosen.

As with abstract predictions, we use both RE and F strategies to generate a concrete action. However to specify the arguments for the action, we simply reuse the arguments of action a_{i+1} from the past case. Figure 23 graphs the resulting accuracy using RE and F strategies and lays them atop the base line performance. Note that the level of accuracy is significantly below that of abstract prediction in Figure 22. This is to be expected, because we must predict more information. For example the base line performances themselves are significantly different.

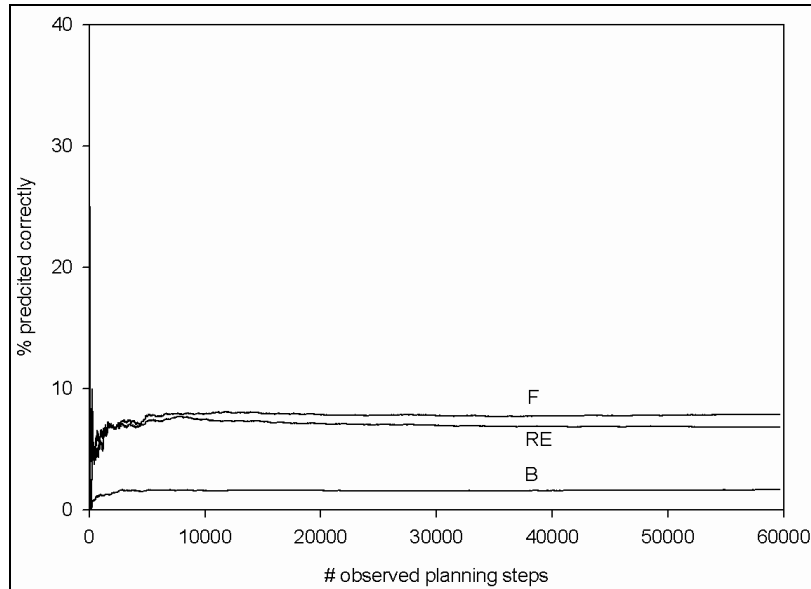


Figure 23. Percentages of correctly predicted *concrete* actions in the *logistics* domain under the baseline (*B*), random elimination (*RE*) and the most frequent (*F*) recognition strategies.

In summary, the use of abstract world states is an efficient indexing scheme, because the number of abstract states is much smaller than the number of concrete world states. The distribution of concrete states into bins indexed by their abstracted representation provides means to eliminate a large number of possible hypotheses (predictions), thereby focusing the recognition process on the relevant cases. The accuracies of these hypotheses can be improved greatly if we select arguments for the predicted action in a more intelligent manner. To do this requires a substitution of arguments through case adaptation.

4. Adaptation

4.1. Predicate argument substitutions

In the previous section, we described prediction results in the context of the case retrieval without adaptation. Instead of reusing a matched action selected from a retrieved case as is, adaptation involves calculating a set of new action arguments that can be substituted for the old arguments. The substitution process is based on determination of similarity between the current planning situation and retrieved past situation in which the matched action was pursued. Consider Figure 24a and Figure 24b with states s_a and s_b , respectively. The two states in the figure have the same abstract representation and the

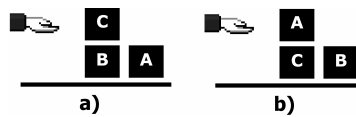


Figure 24. Two alternate initially observed states.

same structure, but the positions of the blocks are not identical. Substitution of arguments provides a means of relating the two structurally identical world states in predicate representation. In case of Figure 24, the substitution σ defined by

$$\sigma = \{A/C, B/A, C/B\}$$

represents the unique substitution of arguments of state s_b into arguments of state s_a . If state s_b is the currently observed state and state s_a is the matching state from an old case, then substitutions can provide more accurate predictions than simply using the corresponding old action. If the planner performed a “pick-up *BlockC*” action in the previous situation represented by state s_a , then the substitution mechanism enables the recognizer to adapt the argument of the predicted pick-up action from *BlockC* to *BlockA*, because *BlockA* from state s_b substitutes for *BlockC* in state s_a .

When the state-space for a planning domain is more complex, calculating an argument substitution can face computational complexity problems. Consider the state shown in Figure 25 in which all fifty different blocks are laying unstacked on the table. When two such states are considered for the argument substitutions, it is clear that any block from one state can be substituted for any block from the other state, resulting in exponential number of possible substitutions. These possible substitutions are essentially all possible permutations of fifty objects, the size of which is not practical for a computer implementation. Another example is shown in Figure 26 where two different states that have an identical abstract representation are considered for the argument substitutions. Because the two states are structurally different, no exact argument substitutions are possible. In the worst case situation, the reasoner will need to check all possible combinations of substitutions in order to determine that no substitutions are actually possible. Again, the overhead associated with such a failed search is very costly. In the case of the example in Figure 26, we may simply check the stacked blocks first and realize that no substitutions are possible. This particular heuristic, however, does not generalize across all planning domains.



Figure 25. An example of a state from the blocksworld domain where the complexity of the substitution scheme is large.



Figure 26. Two structurally different blocksworld states with identical abstract representations.

Computational complexity involved in the determination of substitutions for a pair of states can be viewed in the context of the state graphs introduced in Section 3.2. We showed that the determination of whether two states are structurally identical may be accomplished by answering a decision question about the existence of an isomorphic mapping between the corresponding state graphs. As it turns out, the problem of finding the argument substitutions is reducible to the problem of finding the isomorphism mappings between the state graphs. If such isomorphic mapping f exists, then for every pair of vertices v_1 and v_2 representing domain objects a_1 and a_2 , respectively,

$$f(v_1) = v_2 \Leftrightarrow \sigma(a_1) = a_2.$$

Therefore, the complexity of the substitution problem is a consequence of the complexity of the graph isomorphism problem. When all state graphs in a given planning domain belong to a class of graphs for which efficient isomorphism algorithms exist, determining possible substitutions is straightforward. In general, graph isomorphism is computationally intractable. For practical purposes, we utilize a sub-optimal substitution scheme whose running time is linear. The substitution scheme is an extension of the pseudo-isomorphism relation used in the creation of equivalence classes during indexing and retrieval. If two state graphs are pseudo-isomorphic, then a possibility exists that they are also isomorphic. The pseudo-isomorphic substitution process is neither correct nor complete, because it is not guaranteed to find any correct substitutions when they exist. On the other hand, the process guarantees that two state graphs are not isomorphic if they are not pseudo-isomorphic, because of their different incidence sets.

4.2. Substitution based on pseudo-isomorphism

In the context of local plan recognition, we can limit the substitutions to a subset of all domain objects. A natural choice is to limit the substitutions to include only the arguments of the considered planning action. Because the number of arguments of a typical planning action is small, substitutions for the action arguments may be found much easier. In terms of the state graph representation, we attempt to find the pseudo-isomorphism mapping of two graphs induced by the vertices that are the arguments of the considered action. Induced graphs can be much smaller than the complete state graphs and finding isomorphism among induced graphs in general could thus be simpler. In extreme cases, however, a vertex representing an action argument can be adjacent to a large number of vertices or even to every other vertex. Even in such situations, the complexity associated with the pseudo-isomorphism scheme is linear in the number of vertices and edges of a graph.

The substitution process that we chose uses the information contained in the transformed state graphs, concentrating on those vertices that represent the currently matched action arguments. The process is similar to the equivalence class comparisons introduced in Section 3. To explain the substitution algorithm, consider two states s_{new} and s_{old} with their corresponding state graphs G_{new} and G_{old} , respectively. Let

$$G_{new} = (V_{G_{new}}, E_{G_{new}}), G_{old} = (V_{G_{old}}, E_{G_{old}}), \text{ with } V_{G_{new}} = \{v_1, \dots, v_{k_1}\}, V_{G_{old}} = \{u_1, \dots, u_{k_2}\}.$$

Let us assume that state s_{new} is the currently observed state, while state s_{old} is the previously observed state that needs to be adapted for the current situation. Because we are adapting action arguments, let a_{old} be the action taken in the state s_{old} that needs

adaptation. Because arguments of the chosen action are domain objects, the set consisting of the chosen action's arguments is

$$\text{args}(a_{new}) = \{u_{a_1}, u_{a_2}, \dots, u_{a_q}\} \subseteq V_{G_{old}}.$$

Assuming that the action a_{old} has arguments

$$u_{a_1}, \dots, u_{a_q} \text{ and that } k_1 = k_2,$$

the algorithm for finding substitutions for action arguments is illustrated in Figure 27.

```

Set FIND_ACTION_SUBSTITUTIONS ( $G_{old}, G_{new}, \{u_{a_1}, \dots, u_{a_q}\}$ )

Set substitutions =  $\emptyset$ 
for each action argument  $u_i, a_1 \leq i \leq a_q$ 
  String connections = getSortedConnections( $G_{old}, u_i$ )
  Set matchedVertices = findVerticesWithConnections( $G_{new}, connections$ )
  Vertex  $v_i$  = chooseSingleSubstitution(matchedVertices)
  while  $v_j \in substitutions$  AND matchedVertices  $\neq \emptyset$ 
    matchedVertices = matchedVertices -  $\{v_j\}$ 
     $v_j$  = chooseSingleSubstitution(matchedVertices)
  end while
  if matchedVertices  $\neq \emptyset$ 
    substitutions = substitutions  $\cup v_j$ 
  else
    return  $\emptyset$ 
  end if
end for
return substitutions

```

Figure 27. The algorithm for finding substitutions for action arguments.

The algorithm *FIND_ACTION_SUBSTITUTIONS* in Figure 27 is quite similar to the *EQUAL_UNDER_PSEUDO-ISOMORPHISM* algorithm illustrated earlier in Figure 18. *FIND_ACTION_SUBSTITUTIONS* method first initializes the return set *substitutions* to an empty set. The *substitutions* set will contain vertices that are substitutions for the input vertex set. The *for* loop sequentially considers vertices that correspond to the arguments of the action that is being adapted. For each such vertex, the algorithm attempts to find a unique *connections* string in the state graph G_{old} by calling *getSortedConnections* method. We then attempt to find *matchedVertices*, a set of vertices in the state graph G_{new} that have the identical *connections* string. Subsequently, we choose a single vertex v_j in the *matchedVertices* set as a substitution for the vertex u_i by calling the *chooseSingleSubstitution* method. At present time, *chooseSingleSubstitution* method simply returns the first vertex in the input vertex set. If such vertex v_j is found, we add it to the return set *substitutions*. Otherwise, no substitution for the input vertex u_i exists and the algorithm returns an empty set. After all input vertices have been considered in the *for* loop, the algorithm returns the set *substitutions* which contains vertices that substitute for the vertices in the input vertex set.

The algorithm for the method *getSortedConnections* was shown earlier in Figure 19, while the algorithm for the method *findVerticesWithConnections* is depicted in Figure 28.

The algorithm in Figure 28 starts by initializing the return set *matchedVertices* to an

```

Set findVerticesWithConnections(G, connections)

Set matchedVertices = ∅
V(G) = {v1, ..., vm}
for each vertex vi, 1 ≤ i ≤ m
  String conn = getSortedConnections(G, vi)
  if conn = connections
    matchedVertices = matchedVertices ∪ vi
end for
return matchedVertices

```

Figure 28 The algorithm for *findVerticesWithConnections* method.

empty set. Then for each vertex v_i in the vertex set for the input state graph G , we find the *conn* string by calling the method *getSortedConnections*. If the *conn* string is identical to the input *connections* string, a match has been found and we add the vertex v_i to the *matchedVertices* return set. After considering each vertex in the input graph G , the algorithm returns the *matchedVertices* set that contains all vertices in the input graphs whose connections string is identical to the input *connections* string.

The algorithm for finding the substitutions for the action arguments is linear in the number of edges and vertices in the subgraph induced by the vertices acting as the action arguments. Thus this algorithm enables the observer to quickly determine that no substitutions exist between the two given states, because the edge incidencey set of some vertex u in G_{old} may not have a match in the graph G_{new} . The process may also find some substitutions that are not applicable outside the subgraph and therefore are not feasible substitutions between the two states.⁹ This sub-optimal behavior is compensated by the efficiency and the simplicity of the proposed substitution process.

To illustrate the *FIND_ACTION_SUBSTITUTIONS* algorithm, let us look at an example involving the plans P_{old} and P_{new} illustrated in Figure 29. In this example, we are trying to adapt the arguments of the action *pickup BlockB* from plan P_{old} to match the current situation s_{new} in the currently observed plan P_{new} . Figure 30 shows the state graphs G_{old} and G_{new} for the states s_{old} and s_{new} from Figure 29, respectively.

Because the past action *pickup BlockB* contains a single argument *BlockB*, the *FIND_ACTION_SUBSTITUTIONS* algorithm will consider a single vertex, namely

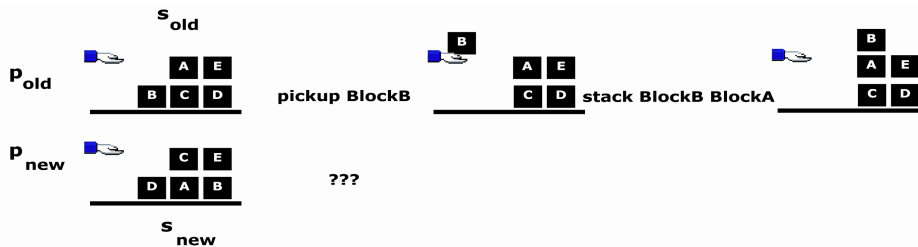


Figure 29. Past plan P_{old} and the initial state of the current plan P_{new} .

⁹ Infeasible substitutions should ideally be eliminated from further consideration.

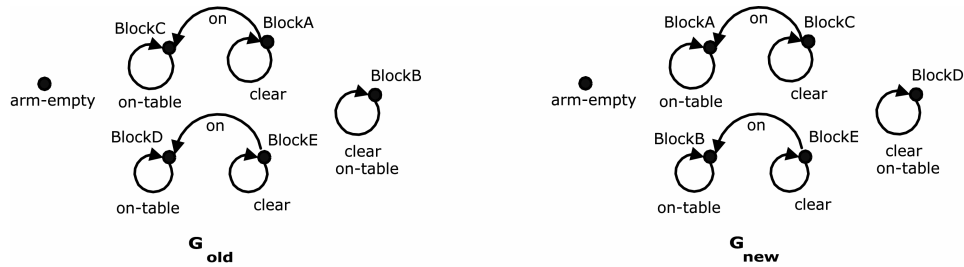


Figure 30. State graphs G_{old} and G_{new} for the states s_{old} and s_{new} from Figure 29, respectively.

BlockB. The *connections* string returned by the method *getSortedConnections* is the string

clear1 on-table1.

The reason for this is that the vertex *BlockB* in the state graph G_{old} induces the subgraph G'_{old} shown in Figure 31. There are two edges in this subgraph labeled *clear* and *on-table*. The incidencey set for these edges includes a single vertex *BlockB*, whose position in the set is obviously one.

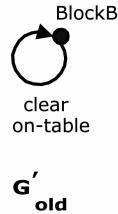


Figure 31. Subgraph G'_{old} induced by the vertex *BlockB* in the state graph G_{old} .

Table 3 shows the *connections* strings produced by calling the method *getSortedConnections* for the vertices in the state graph G_{new} . We can see that only one vertex, *BlockD*, has the required connections string *clear1 on-table1*. Therefore, the method *findVerticesWithConnections* returns a substitution set containing a single vertex *BlockD*.

The *FIND_ACTION_SUBSTITUTIONS* algorithm in this particular example returns

Table 3. The *connections* strings for the vertices in the graph G_{new} .

vertex	connections
arm-empty	null
BlockA	on2 on-table1
BlockB	on2 on-table1
BlockC	clear1 on1
BlockD	clear1 on-table1
BlockE	clear1 on1

a set containing the vertex *BlockD*. This means that the only possible substitution for *BlockB* in the past state s_{old} from the past problem P_{old} is *BlockD*. Therefore, the observing agent's prediction for the planner's next action, given the initial state s_{new} in the current plan P_{new} , is that the planner will execute *pickup BlockD* action.

The argument substitution scheme can enable the recognizer to make predictions that are more informed than predictions without the adaptation. The following subsection experimentally evaluates the effects of action argument adaptation.

4.3. Empirical evaluations

In order to evaluate the effectiveness of the pseudo-isomorphic argument substitution scheme, we performed recognition experiments on the same randomly seeded logistics domain problem sets used for the experiments in the previous section. Figure 32 shows the effects of the action argument adaptation scheme on the prediction accuracy compared to re-using the previously matched action without adaptation. We can see from Figure 32 that the percentage of correctly predicted actions for the random candidate elimination strategy with argument substitutions (*RE_W_S*) is twice the percentage of the corresponding scheme without the adaptation. The accuracy of the most frequent action selection strategy with adaptation more than doubles when compared to the most frequent strategy with no adaptation. Therefore, the adaptation process enables the recognizer to significantly increase the recognition accuracy on the experimental

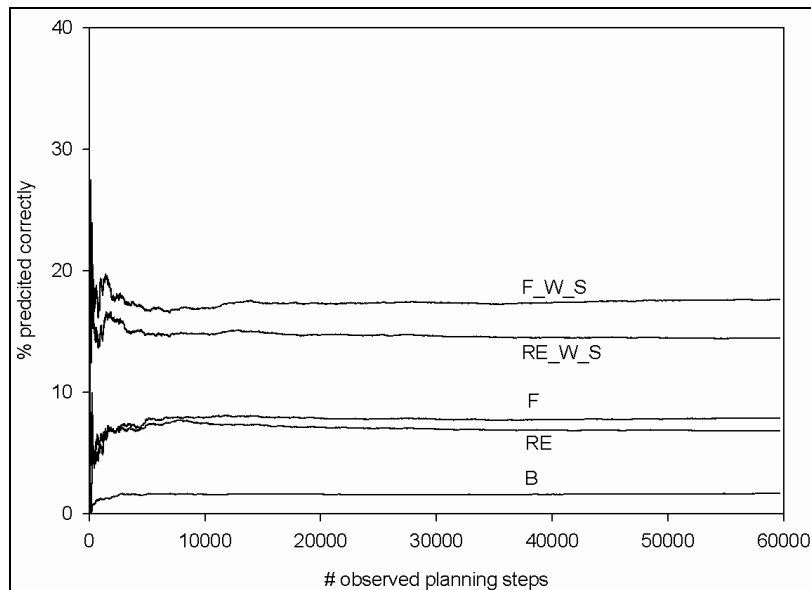


Figure 32. Percentages of correctly predicted *concrete* actions in the *logistics* domain, under the baseline (*B*), random elimination (*RE*) the most frequent (*F*), random elimination with substitutions (*RE_W_S*) and the most frequent with substitutions (*F_W_S*) recognition strategies.

problem sets in the logistics planning domain. The most frequent action selection strategy slightly outperforms the random candidate action elimination strategy as expected. The

statistical analysis of the differences reveals that the differences in performances among all recognition strategies are significant, determined by the Scheffe's test with $\alpha=0.05$.

Figure 33 shows the prediction accuracies that combines all of the previously shown graphs for the logistics planning domain. Curves whose names end in a suffix “_A” represent the next action predictions at the abstract level. Judging by ending intervals of the depicted percentage curves, it can be concluded that the recognizer is able to correctly predict about one out of three actions at the abstract level. At the concrete level, however, the accuracy drops to about one correct prediction in about five and a half attempts. Note that the prediction accuracy at the concrete level is bounded above by the accuracy at the abstract level, because the adaptation follows the selection of an action at the abstract level. Subsequently, we can conclude that the recognizer is able to correctly predict concrete actions about one half of the time given the upper prediction bound. Change in the concrete prediction accuracy can be accomplished by changing the adaptation criteria. The adaptation process, however, has no effect on the abstract prediction accuracy, as it is only concerned with the concrete level of abstraction.

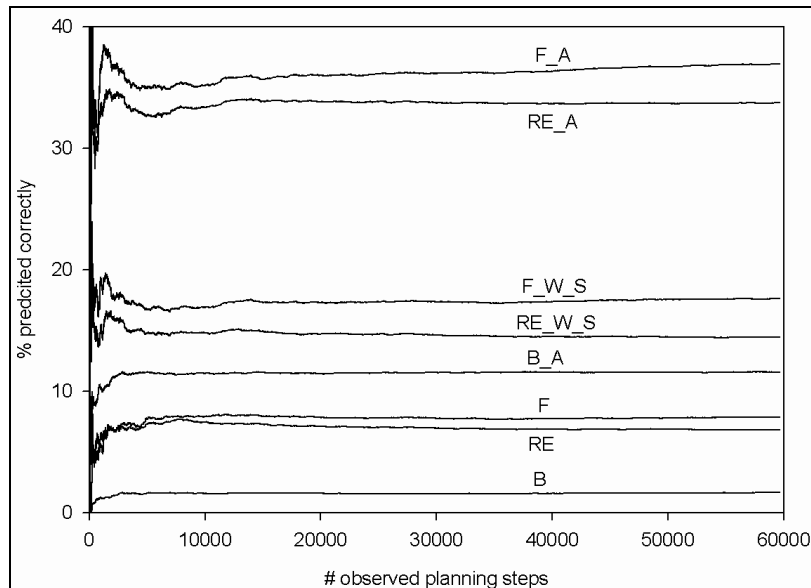


Figure 33. Percentages of correctly predicted next actions, abstract and concrete, for the logistics domain.

Evidence of near steady-state behavior for the problem sets in the logistics domain can be found in Figure 34, which shows the number of recognition steps at which no predictions could be made as a percentage of the overall number of observations. These steps involve situations where a completely new abstract state is observed and a new bin is created. The slope of the curve in Figure 34 falls throughout the life cycle of the system, reaching a small positive percentage by the end of observations. This indicates that most of the state-space is fully explored and is consistent with the findings illustrated earlier in Figure 12.

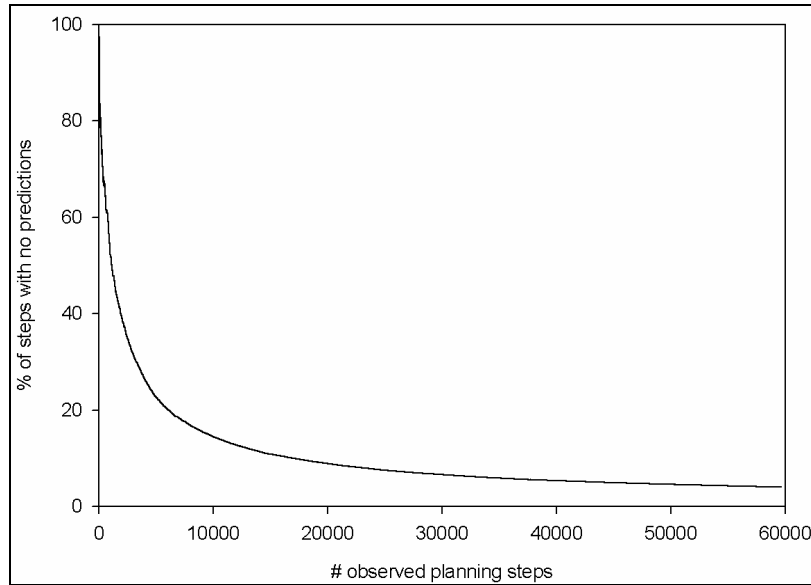


Figure 34. Percentages of planning steps at which no predictions could be made, for the logistics domain.

5. Case Storage (and planning with incomplete plan libraries)

After the recognizer observes the goal state of the current plan, it stores the observed case in the plan library. As discussed earlier, the library consists of plans represented as sequences of action-state pairs. The stored plans contain intermediate states that are also present within the pseudo-isomorphic equivalence classes. Each concrete state also contains pointers to the cases in the library in which it is contained, including its position in the sequence within those cases. This allows the recognizer to retrieve those cases that contain the situations similar to the currently observed situation.

Most plan recognition systems operate with *complete* plan libraries consisting of all possible plans the planner may potentially pursue. Therefore, a system with such a library guarantees that any plan observed by the recognizer will be present in its library. The recognition process involves matching the observed partial plan with the suffixes of plans from the library, and predicting the future planner's behavior from the suffixes of the matched plans. Such systems never encounter a novel planning action and have no mechanism to reason in such situations. In effect, the completeness of the plan library helps traditional recognizers avoid issues involved in novel observations of the planner's behavior.

To our knowledge, no other plan recognition system utilizes *incomplete* plan libraries. Yet there certainly exist situations where complete plan libraries introduce additional overheads. Lesh and Etzioni show that plans in the library that are never pursued (extraneous plans) impact the efficiency of the recognizer.⁴¹ However, except for the simplest domains with a small number of possible states, planners tend not to explore all of the state-space during the planning. In fact, the number of states a planner encounters is domain dependent and often constitutes only a fraction of the number of all

possible states it may explore, as our experimental results show. Because planners tend not to explore the complete state-space, completeness of plan libraries results in a number of extraneous plans. Moreover, enumeration of all planning episodes is often a tedious knowledge-engineering task. Furthermore, enumeration of all possible plans may be impossible to achieve, especially in adversarial planning domains such as in military planning domains.

The recognizer presented in this paper operates with incomplete plan libraries. In fact, the recognizer starts out with an empty plan library that is built incrementally from the observations of the planner's behavior. The intermediate planning states serve as indices into the memory that enable reasoning with incomplete plan libraries. While most traditional recognizers reason in terms of matching the observed planning actions with actions in the library, our system is also capable of reasoning in terms of planning situations. This in turn enables the recognizer to make predictions in light of novel planning actions.

The storage phase utilizes the same indexing structures already described in Section 3 concerning the retrieval phase. Observed cases are stored in the library in their extended representation (action-state pairs). Individual situations encountered during the observation of the current plan are placed in their corresponding equivalence classes. For each concrete state observed, the recognizer first forms the abstract vector representation according to the procedure outlined in Figure 14. The abstract states represent bin identifiers that are easily and efficiently accessible by the means of the bin hash table.

In situations when a bin corresponding to the observed abstract state does not exist, the recognizer creates a new bin and places it into the bin hash table. The recognizer also forms a single equivalence class within that bin and places the observed concrete state in the class as the class representative. The equivalence relation utilized for the creation of the equivalence classes is the pseudo-isomorphic relation discussed in Section 2.

On the other hand, when the bin corresponding to the abstract representation of the currently observed state already exists, the recognizer focuses its attention to the equivalence classes within the matched bin. The currently observed concrete state is transformed into the corresponding state graph, which is then compared to the state graphs corresponding to the equivalence class representatives. In situations when state graph comparisons fail to match any of the class representatives, a new equivalence class is created with the current state as the class representative. Otherwise, the current state is placed in the equivalence class represented by the state whose state graph was identical to the current state's graph under the pseudo-isomorphic equivalence relation.

Incremental construction of the plan library ensures that the only states present in the library are those states that were explored by the planner. This in turn minimizes the occurrence of extraneous plans and decreases the size of the explored portion of the state-space. Recognizers with incomplete plan libraries may run into the problem of incorrectly predicting the planner's course of action, especially in earlier stages when the indexing structures are being created at a higher rate. However, the flexibility of such an approach enables the recognizer to reason about the planner's future intent in cases when traditional recognizers utilizing complete plan libraries would falter (e.g., when a novel planning action is observed).

After the indexing phase of the recognition cycle is complete, the recognizer is ready to observe the next action-state pair in the current plan. If the plan's goal state has not been reached yet, a subsequent observation triggers the retrieval phase and the recognition cycle is repeated. The effectiveness of retrieval, adaptation and indexing

described in the previous sections and this section is illustrated with experimental evaluations in the next section.

6. Evaluations of the Extended-STRIPS Planning Domain

In this section we analyze the extended-STRIPS planning domain in more detail and present the empirical evaluations that parallel the evaluations we presented for the logistics domain. This domain is an extension of the early STRIPS planning domain.²³ Before we further explore the properties of the extended-STRIPS domain, let us discuss generation of the planning episodes we utilized as our experimental data sets.

6.1. Case generation

Our investigation of the properties of the plan recognition system illustrated in this paper concentrated on the logistics and the extended-STRIPS planning domains. We choose these two planning domains because of their different state-space characteristics. The blocksworld planning domain is utilized throughout this paper as an illustrative tool, because its states can be easily visualized. The blocksworld domain is also characterized by a very small state-space, which results in saturation of some equivalence classes with a large number of concrete states. Therefore, this domain is better suited for the recognition techniques that generalize a large number of concrete instances discussed in Section 2.

The planning problems that served as the input into the PRODIGY 4.0 planner were generated randomly. Random problem generators for each of the two planning domains allow the user to specify the generation parameters that determine the shape of the state-space for the generated problems. One of the common generation parameters is the random seed, so that the random choices can be replicated for a comparison of different recognition strategies. We used two different random seeds to generate cases for both planning domains. Other generation parameters that the two domains have in common include the total number of cases to generate and the total possible number of goal conjuncts. All other generation parameters are domain-specific.

In case of the logistics planning domain, the user specifies the number of cities, which in turn determines the number of post offices and airports, because there is one of each per city. The user can also specify the maximum numbers of trucks, airplanes, and packages. The actual numbers are determined by randomly choosing a value between one and the specified maximum. The initial states of generated problems are determined by randomly choosing the locations of trucks, airplanes and packages, while the goal conjuncts consist of literals specifying the final destinations of the packages.

The generation parameters for the extended-STRIPS domain include the number of rooms¹⁰, maximum number of packages, and the minimum and the maximum number of available doors connecting the rooms. The doors connect only rooms that are adjacent. In order to make the generated problems feasible, the problem generator guarantees that all of the rooms are reachable via doors, by specifying the number of doors for a given state space to be at least one less than the maximum possible number of doors. However, each

¹⁰ The “floor plan” is restricted to be in a form of a rectangle, which influences the total number of rooms possible (e.g., 2, 4, 6, 9, ...).

door is randomly selected to be either locked or unlocked as well as closed or open, and the keys for each of these doors are randomly placed in rooms. Therefore, it is not guaranteed that all of the rooms will be accessible, because all doors to some rooms may be locked and keys for these rooms may be in similar inaccessible locations. Although this property influences the success rate for plan generation, we were able to collect enough planning episodes to analyze the behavior of the recognizer. The goal conjuncts specify the final destinations for the boxes (in rooms or held by the robot), as well as the door status (open or closed, locked or unlocked). The goals are chosen randomly from the possibilities above.

Table 4. The total numbers of generated plans.

Seed 1213		Seed 31307	
domain	# of plans	domain	# of plans
Logistics	7,886	Logistics	7,956
Ex-STRIPS	1,257	Ex-STRIPS	1,282

The generated plans were then executed the PRODIGY4.0 planner. The total number of successfully solved planning episodes for both domains is shown in Table 4. The planner successfully generated solutions (plans) to about 80% of the problems in the logistics domain. On the other hand, the planner was able to successfully generate solutions to only about 8% of the input problems in the extended-STRIPS domain. Some of the plans did not finish in a limited amount of time allowed for their execution. But such drastically low execution success rate for the extended-STRIPS domain is caused by its large complexity. The reason for such a large state-space for relatively undemanding extended-STRIPS planning domain lays in the construction of the room layouts in the random problem generator. To increase the state-space of this domain and observe the recognizer's behavior in such a domain, the plan generation randomizes the names and positions of doors connecting the rooms which the robot explores. In other words, doors are randomly picked from a list and randomly positioned on the walls between two rooms. This problem generation scheme increases the number of possible concrete states and subsequently, increases the size of the concrete state-space.

Generated planning episodes amended with the intermediate state information are then passed one-by-one to the plan recognition system. These plans are grouped into problem sets by the seed for the random generation choices. Consequently, each domain was evaluated on two different problem sets, constructed with differently seeded random problem generators.

The plan recognition process starts with an empty plan library. The library and the abstract state-space are constructed incrementally from the observed planning episodes. The recognizer is equipped with a plan preprocessing module that creates the extended plan representation consisting of the action-state pairs from the amended planner's output. The action-state pairs are processed by the recognizer in a way that simulates a real-time execution of a plan by the PRODIGY planner. In order to implement the state abstraction scheme, the recognizer needs the object hierarchy as one of its inputs. Currently the recognizer automatically extracts the object hierarchy from the description

of a given planning domain¹¹. Table 5 shows the total numbers of observed planning steps (action-state pairs) for each problem set in both planning domains.

Table 5. The total numbers of observed planning steps (action-state pairs).

Seed 1213		Seed 31307	
domain	# planning steps	domain	# planning steps
Logistics	59,640	Logistics	60,330
Ex-STRIPS	13,075	Ex-STRIPS	13,391

The evaluations of the recognizer’s performance with the problem sets for the logistics domain were introduced in the earlier sections. The following sub-section discusses the state-space properties and recognition performance for the extended-STRIPS planning domain.

6.2. State-space properties – extended-STRIPS

As discussed earlier in Section 2, the state-space for the extended-STRIPS domain is much larger than the state-space for the logistics domain. The discrepancy in the size of state-spaces is due to a larger number of predicate combinations that constitute the world states and the randomization of room layout across plans in the extended-STRIPS domain. This results in a smaller number of both cases and observed planning steps in this domain. We saw in Figure 12 that the slope for the curves representing number of created bins and equivalence classes reached a small positive value in the logistics domain in a long term. Figure 35 shows that this is not so in case of the extended-STRIPS domain. The slopes of the corresponding curves here are decreasing, but the rate of decrease is much smaller than in the logistics domain. Notice that the curve for the concrete states is linear, indicating that very few concrete states are encountered more than once during a recognition run. In fact, during observation of about 13,000 recognition steps in a 4-room extended-STRIPS domain, no concrete state has been observed twice for one of the problem sets, while the second problem set yielded repeated concrete states on only two occasions. Even in such diversified problem sets, the recognizer will be able to perform relatively well.

Further evidence of a large state-space for the extended-STRIPS planning domain is illustrated in Figure 36, which shows the percentages of the planning (recognition) steps at which no predictions could be made. An observance of a state with a novel abstract representation is the only example of a step where no predictions can be made, because a match at any level will result in a formation of a prediction about the planner’s intentions. Therefore, this rate is inversely proportional to the rate of the bin creation. Figure 37 indicates that the prediction inability rate at the end-points of the curves is still rather high (about 40%) when compared to that of the logistics domain curve in Figure 34 (less than 5%). Unlike the problem sets for the logistics domain, the extended-STRIPS problem sets do not allow the creation rates to slow down and allow them to reach near steady-state behavior.

¹¹ The description consists of a *domain* and a default problem files.

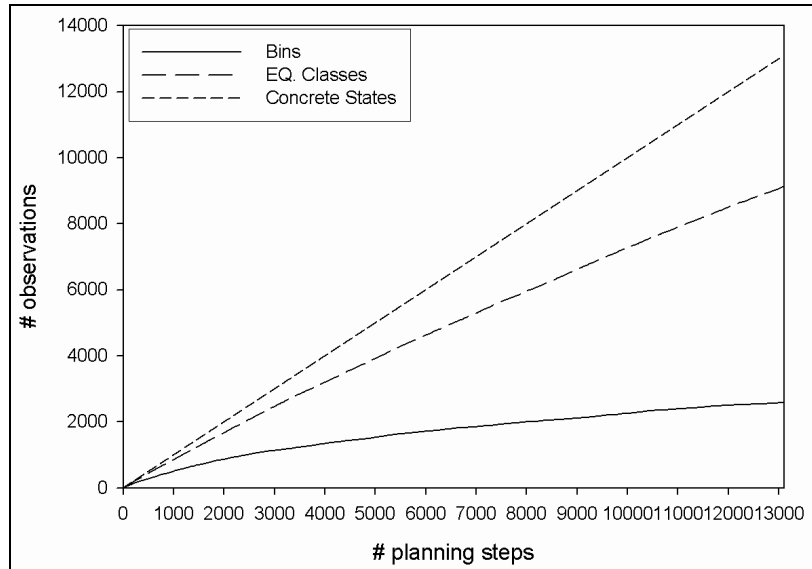


Figure 35. The number of abstract states (bins), equivalence classes and concrete states in the extended-STRIPS domain

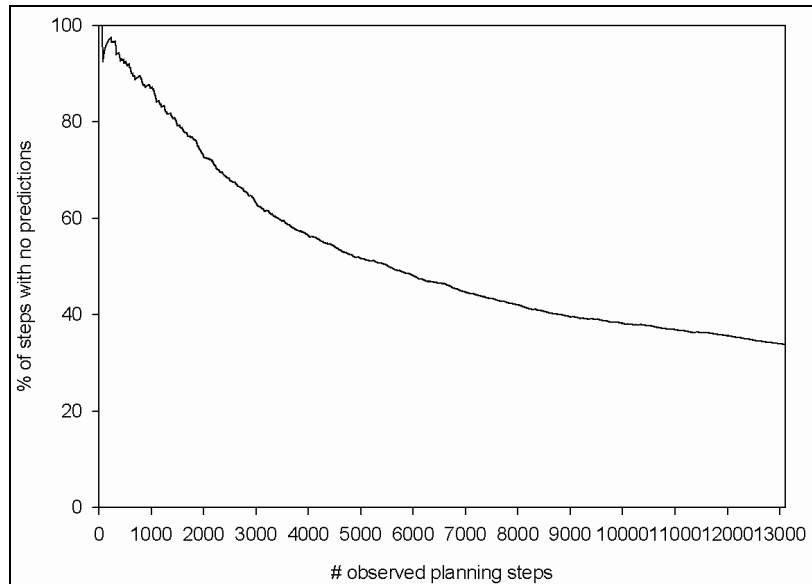


Figure 36. Percentages of planning steps at which no predictions could be made for the extended-STRIPS domain.

Therefore, the two explored planning domains provide evaluations of the next action prediction for the state-spaces of different sizes and characteristics. Evaluations of

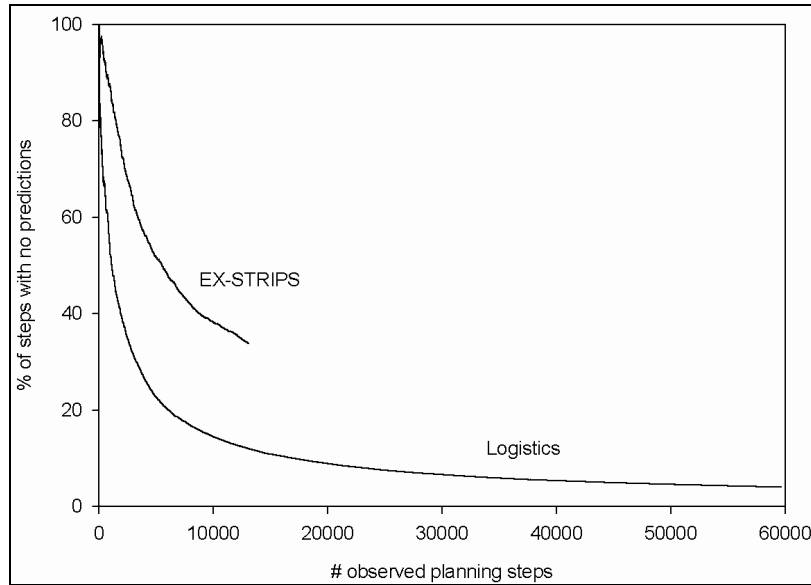


Figure 37. Percentages of planning steps at which no predictions could be made, for both the logistics and extended-STRIPS domains.

Table 6. State-space statistics.

	Logistics		Extended-STRIPS	
	Seed 1	Seed 2	Seed 1	Seed 2
# observed steps	59,640	60,330	13,075	13,391
# bins	806	811	2,557	2,624
# eqv. Classes	6,307	6,468	9,122	9,282
# concrete states	32,696	33,179	13,073	13,391
Avg. class size	5.18	5.13	1.43	1.44
Max. class size	76	120	24	31
Min. class size	1	1	1	1

the logistics domain paint a picture of the recognition accuracy once the state-space has been adequately explored and most of the indexing structures have been created. The statistics for the state-spaces are shown in Table 6. Note also that while observing about 8,000 plans executed by the planner in the 3-city logistics domain, the recognizer encounters a case that has already been stored in the library about 1,100 times. On the other hand, the recognition on the sparsely-explored planning domains can be evaluated using the extended-STRIPS planning domain. Results obtained for this particular domain will indicate the recognizer's ability to cope with diverse planning environments in which planning states do not repeat or repeat very little.

6.3. Empirical evaluations – extended-STRIPS

We saw in Sections 3 and 4 that the recognizer performs relatively well as far as the prediction accuracy in the logistics domain is concerned. We now present similar experimental evaluations concerning the extended-STRIPS domain.

Figure 38 shows the next action prediction accuracies at the abstract level. We can see that both random elimination (RE) and the most frequent (F) action candidate selection strategies outperform the baseline (B) selection strategy, which simply selects a random action out of all previously observed actions. However, RE and F strategies show no significant differences in their performance in a long run under a visual inspection. This is because the rate of the creation of indexing structures is still high for this domain, given the size of its state-space and a smaller number of observed planning steps. Nevertheless, these differences are statistically significant under Scheffe's test with $\alpha=0.05$. The same can be said for the curves in Figure 39, which show the prediction accuracies at the concrete level. In this figure we can see that the differences between RE and F strategies are even less visible, although still statistically significant.

The adaptation process also significantly increases the prediction accuracy of the recognizer as expected. Figure 40 shows the curves for the prediction accuracies with adaptation (suffix "W_S"), while Figure 41 shows all of the previously discussed curves for the extended-STRIPS domain in a single graph. The recognizer is able to correctly predict more than half of the next actions at the abstract level, while the accuracy drops to about one out of five correct predictions at the concrete level. These results are very encouraging, especially when we take into consideration a large state-space and non steady-state behavior in the extended-STRIPS domain.

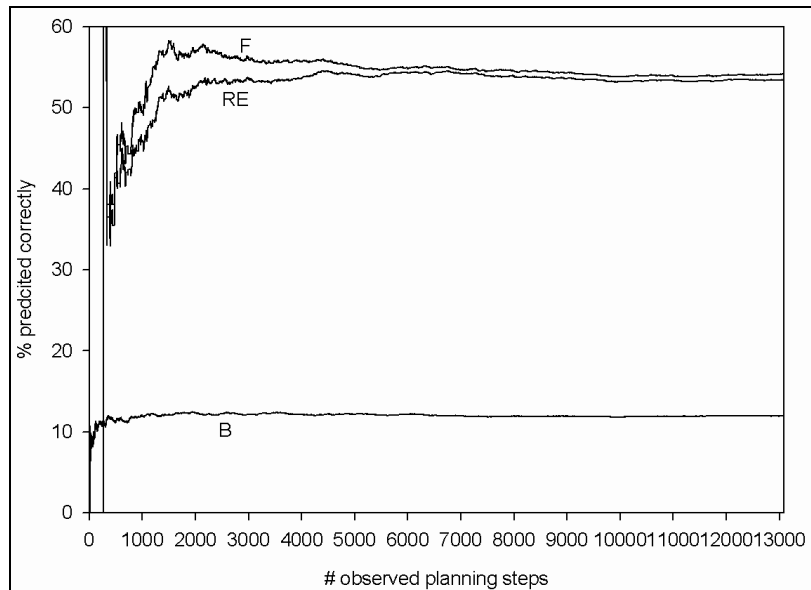


Figure 38. Percentages of correctly predicted *abstract* actions in the *extended-STRIPS* domain, under the baseline (B), random elimination (RE) and the most frequent (F) recognition strategies.

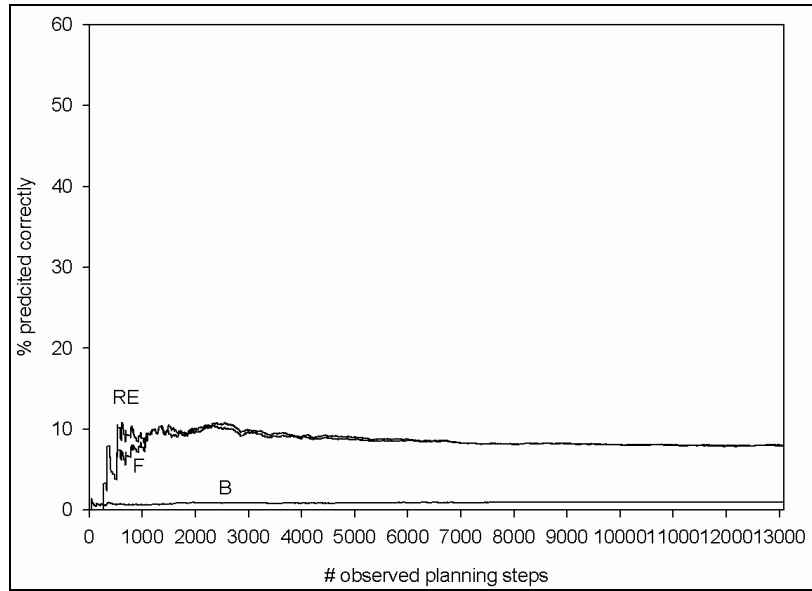


Figure 39. Percentages of correctly predicted *concrete* actions in the *extended-STRIPS* domain, under the baseline (*B*), random elimination (*RE*) and the most frequent (*F*) recognition strategies.

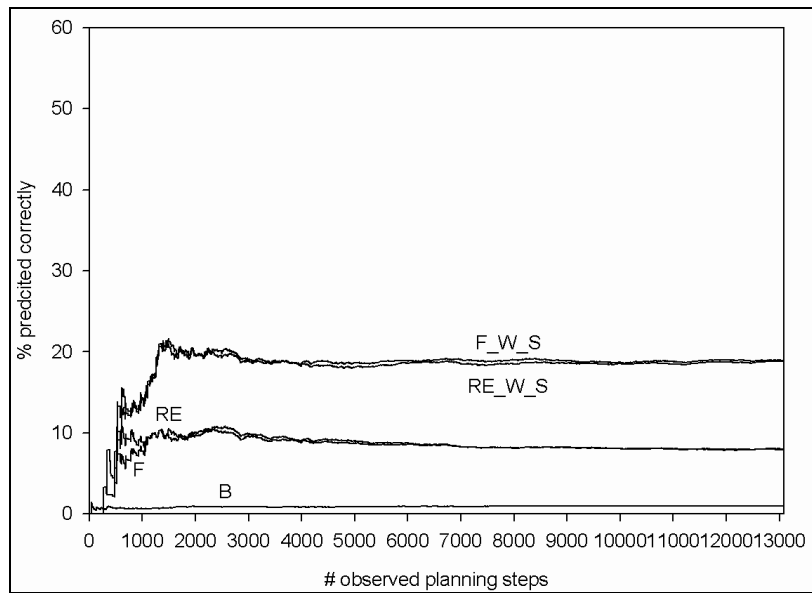


Figure 40. Percentages of correctly predicted *concrete* actions in the *extended-STRIPS* domain, under the baseline (*B*), random elimination (*RE*) the most frequent (*F*), random elimination with substitutions (*RE_W_S*) and the most frequent with substitutions (*F_W_S*) recognition strategies.

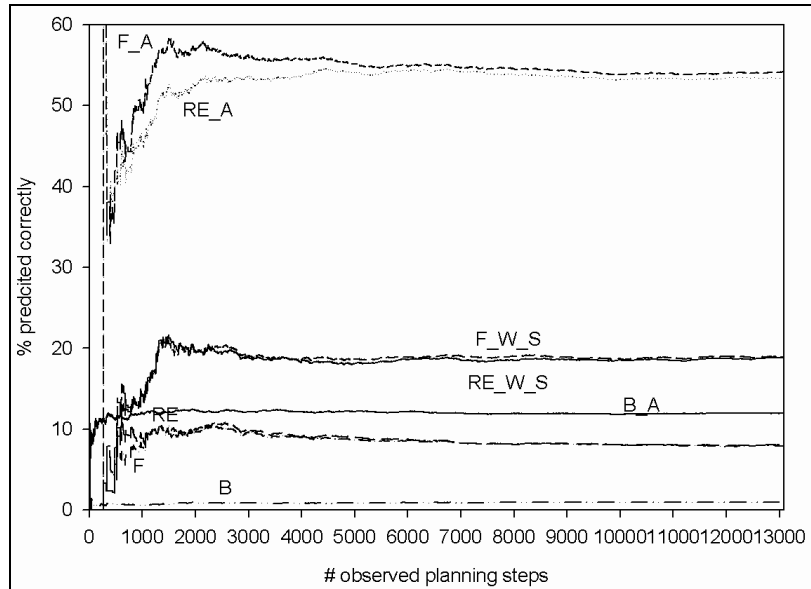


Figure 41. Percentages of correctly predicted next actions, abstract and concrete, for the extended-STRIPS domain.

Comparing these evaluations with the evaluations from the logistics domain in Sections 3 and 4, we can draw some generalizations of the recognizer's performance. It is evident that focusing the retrieval process on a small subset of the whole state-space represented by the equivalence classes does improve the prediction accuracy. The evidence of this is found in the accuracy curves presented for both domains, where choosing an action at random from an equivalence class (random elimination strategy) outperforms choosing an action at random from all previously observed actions (baseline strategy). Furthermore, it is evident that the adaptation (suffix "W_S") significantly increases recognizer's local prediction accuracy when compared to the corresponding strategies that simply reuse previous action arguments. On the other hand, it is not clear whether the most frequent (F) action candidate selection strategy differs in performance from the random elimination strategy. The former strategy outperforms the latter in case of the logistics domain, while the differences are only slight, yet statistically significant in the case of the extended-STRIPS domain. This is probably due to a smaller state-space and the steady-state nature of the logistics planning domain.

The results presented in this paper concerned two planning domains with different characteristics. In order to generalize the results gathered in here, further evaluations on different planning domains are needed. The next section discusses some of our possible future research directions and summarizes the related research.

7. Related and Future Research

Plan recognition techniques have been investigated a number of different domains, in which there is a need to recognize the goals of some planning agent. The earliest plan recognition systems were concerned with the story understanding, question answering, and natural language dialogue understanding.^{57,47,4} The plans of a perceived agent were

conceived in the form of a natural language. The understanding of the agent's intent is facilitated by association of one or more goals the agent pursues with the actions in a story or the utterances in a dialogue. That is, the agent is thought of as a rational entity, and the agent's plans and goals are considered to be explanations of the agent's rational behavior. Subsequent plan recognition systems were applied to other plan- and goal-oriented domains, such as speech-to-speech translation, help systems, collaborative problem solving, and computer-aided design, just to name a few.^{2,44,42,26} The field of plan recognition was formalized when Kautz published his seminal work on the theory of plan recognition.³¹

Finding the matching plans and selecting the most appropriate plan among the competing plan hypothesis are two of the most important and most widely investigated components of both the plan recognition process and the case-based process. Scientific methods used in this critical phase include Bayesian reasoning,^{16,49} dynamic belief networks,³ decision-theoretic approaches,⁵¹ rationality of coherency,⁴ Dempster-Shafer theory,⁸ abduction,⁶ and case-based reasoning,¹³ just to name a few. However, all of the above-mentioned systems utilize complete plan libraries constructed *a priori*. The novelty of our approach enables the recognition of novel planning actions caused by incomplete plan libraries.

When the plan library is not complete, the knowledge about the intermediate world states introduces more flexibility into the recognition process than if the system relied on past actions alone. Competing hypotheses about possible pursued plans and goals based on past actions can further be discriminated on the basis of the current state of the world. Furthermore, the theory of *dynamic memory* suggests that human planners compare *situations* in which they find themselves with previous situations in their memory in order to perform reasoning.⁵⁰ Cognitive studies have shown that humans utilize both planning actions and situational assessment during the planning.⁴⁵ States of the planner's world, representing situations in which the planner finds itself, provide a basis for the recognizer's reasoning processes. Recognizers that know only about the initial and the goal plan states are limited in their ability to compare and retrieve similar situations. This occurs when the situations in the plan library do not coincide with the observed planning situations. Other researchers have explored the role of the state knowledge to a limited degree. Albrecht and his colleagues explore plan recognition that utilizes different Bayesian net recognition models, one of which recognizes plans based on the state knowledge.³ They postulate that such a model would be applicable in domains with limited number of actions¹². Their work is different from ours in that the states they encounter are simplified¹³ and the number of different states is relatively small (about 4000). Moreover, the plan libraries in this work are complete.

Although the abstraction scheme in the case-based reasoning is not a novel concept, it is certainly novel in a way in which it is used in the context of the case-based plan recognition. Our abstraction scheme is structural in nature and it does not require any sophisticated knowledge engineering in order to be applied to different planning domains. This abstraction scheme exploits the intrinsic properties of the world states represented as predicates and allows for efficient indexing and retrieval of previous plans, even with the evident space complexity. One requirement for the recognizer presented in this work is the ability to monitor the intermediate planning states, along with a prerequisite that the

¹² We are referring to *locationModel*.

¹³ States in their research are locations of a player in the context of online dungeon game. These states lack the detail of information provided by combinations of literals corresponding to sensory information.

states are represented as collections of literals with domain objects as literal arguments. The only other requirement is an associated object type hierarchy in order to perform the abstraction of the concrete states. Although our work utilizes two levels of abstraction, it is certainly possible to extend the abstraction on as many different levels as the object type hierarchy allows.

The case-based plan recognition system introduced here can be classified with respect to the abstraction framework for the case-based reasoning.¹⁰ Our system stores only concrete cases in the library, and because state abstraction is an efficient process, we could automatically generate abstract cases from the concrete cases if there is a need for that. One of the major differences between their and our approach to abstraction is that our system does not use abstract cases for indexing. Instead, abstracted states that are components of cases are used as indices for storage and retrieval. One of the main reasons for this discrepancy is the nature of the plan recognition task with incomplete plan libraries, which is to recognize the planner's intentions, including its goals, when the intentions themselves may not be known (present in the library). Reasoning with incomplete libraries is facilitated by indexing *situations* instead of the cases, so that the recognizer may reason in light of newly observed situations. Although some case-based planning systems use goal states from the cases for the indexing, our system utilizes all observed situations as indices to trigger the reminding process.⁵² Adaptation and reuse of abstraction is implicit in our work, because the abstract predictions are refined to produce concrete predictions. Our abstraction generation process is automatic. Our system does not implement case-deletion policy; this will be addressed in the future research. The summary of the work presented in this paper with respect to the framework by Bergman and Wilke is shown in Table 7.

Similar to the approach in the PARIS system, the recognizer does not form abstraction by simply dropping sentences.⁹ However, the simplicity of our abstraction scheme allows for the minimal abstraction theory, since we are simply counting a number of occurrences of a literal of a certain type among a collection of literals representing a world state. Because the state abstraction is used as a way to quickly trigger appropriate past memories, this approach does not require a domain expert to specify the abstract language either, besides the object type hierarchy. When recognizing the plans from the PRODIGY planner, the recognizer automatically extracts the object hierarchy from the domain description, shielding the end user of any knowledge engineering details.

Table 7. Classification of case-based plan recognition with incomplete libraries with respect to the framework for case-based abstraction.

Kind of stored cases	Concrete (with abstract states stored)
Acquisition of abstract cases	Currently n/a (automatically generated if needed)
Abstract cases for indexing	Yes (situations as indices)
Reuse of abstract solutions	Yes
Adaptation of abstract solutions	Yes (abstract prediction adaptation)
Case deletion policy	No

In this paper, we focused on the local predictions, dealing with recognizing the immediate planning actions taken by the planning agent. Our future research efforts will focus on the global predictions, dealing with recognizing the plans and the goals the planning agent is pursuing. In addition to evaluating the effectiveness of the global predictions on the experimental problem sets introduced in here, we intend to determine

the behavior of the system on additional problem sets characterized by a common set of frequently pursued planning goals. The random problem generator introduced in Section 6.1 produces planning problems whose goal sets are completely random. That is, any goal is a partial goal specification consisting of a random set of predicates. In complex planning domains, goal-oriented agents tend to pursue only a small subset of all possible states as their common goals. The additional experimental problem sets will be constructed with this in mind. Such problem sets emulate task-specific, goal-directed behaviors commonly associated with the human planners.

The evaluation involved in assessing the performance of our recognizer is based on the correctness of the prediction. At the abstract level, a prediction is correct if it matches the name (type) of the observed subsequent action; otherwise it is incorrect. Exact semantic equality is needed for a correct prediction at the concrete level; compared actions need to have identical names and argument lists. The evaluation scheme may also include partially (in)correct predictions, whose correctness assessment can be weighted according to a specified criteria. One such criterion may be the percentage of correctly predicted action arguments. For example, if the recognizer predicts that the planner will pursue *stack BlockA BlockC* action, and the planner executes *stack BlockB BlockC* action, then such a prediction may be considered to be 50% correct because the name, as well as a half of the action arguments, were predicted correctly. Such weighted evaluation schemes may provide more insight about the recognizer's overall performance.

Another aspect of our future research efforts will concern the recognizer's ability to form predictions even in cases when newly abstract states are observed. At the present time, the recognizer does not attempt to form predictions when no matches exist at the bin level. However, given an appropriate similarity metric, the recognizer may attempt to find *similar* abstract states (other similar bins) and utilize those to form its predictions. Because the abstract states are represented as non-negative integer vectors, a k-nearest neighbor similarity metric seems appropriate for this task. Such prediction attempts may result in decreased overall percentages of correctly predicted actions, because these prediction attempts are less informed than predictions created in light of an exact bin (equivalence class) match. On the other hand, the predictions based on the similarities among the abstract states may enable the recognizer to increase the overall number of correctly predicted planner's next actions.

8. Conclusion

The contributions of this research are varying. They include both theoretical and technical implications and promise much in the future. As a short list of what this paper has tried to convey, consider the following research contributions.

- Provided a novel, incremental, robust, generalizable, and scalable integration of case-based reasoning and plan recognition.
- Presented a novel two-level representation of case (plan) libraries indexed by a three-tiered data structure.
- Developed a mathematical analysis of the space complexity in multiple domains and provided empirical evidence to support the analyses.
- Demonstrated good local prediction accuracy at both the abstract and concrete levels.

At the most general level, this research has produced a unique blend of case-based reasoning and plan recognition principles. The plan library is treated as a case base that does not require detailed knowledge engineering of plan templates. Instead the case base incrementally accumulates experience as it interprets observations and thus learns to improve its behavior. The method is robust, because it can make predictions in the face of actions it has never encountered. This new experience is then stored for subsequent observations. It therefore learns from failure as many case-based systems do. As we have demonstrated, the methodology can be used in a number of domains of large size, so it generalizes and scales well.

To handle such a wide set of circumstances, the methodology depends on a unique representation for cases that allows tractable prediction. Plans are represented at two levels. First unlike traditional representations, concrete plans are represented as sequences of action-state pairs. Secondly an abstract plan can be considered as resulting from a truth-preserving data compression. All information that encodes specific object instances is replaced with a type-generalized form. Given these representations, indexes used to store and retrieve a concrete plan given a concrete observation are thus three-fold. Currently observed concrete states *map* to abstract vectors which in turn *hash* to bins whose members *point* to concrete states within previous plans held in the case base. Bins are further divided into disjoint subsets of concrete states by the means of equivalence classes, which are created from the given equivalence relation on the set of all concrete states within a bin.

This paper has also performed a mathematical analysis of the spaces defined by these representations. We showed in a number of domains with specific constraints what the sizes of these spaces imply and how they relate to each other. Not only have we applied an analytic method, but we also showed empirically that these differences do indeed follow as shown by statistical sampling of domains of different complexities.

Finally, we demonstrated empirically across thousands of observations the behavior of a computer implementation of this approach. Although the levels of accuracy may seem to be modest, the results were obtained beginning with sparse plan libraries that increase to very dense distributions. We showed the behavior of the system in both concrete and abstract local prediction tasks and have laid the foundation for further investigations of global interpretation problems.

These contributions promise a new way to attack many applied problems of significant importance. For example, an intelligent user-interface has a similar interpretation problem confronted with human problem-solving behavior. Due to voluminous amounts of information associated with a user's problem-solving task, cognitive overload often results if the information is not presented in a coherent and filtered fashion. Several researchers first considered this problem as a user-interface management problem for which the interface itself could plan.^{12,19,20} Thus the task of the interface is to understand both the user and the context of the user in order to plan to tailor itself to the characteristics of the problem and the user's ability and preferences.

Consider the user's context as consisting of three elements: the window-object characteristics, the windowing content and function, and the user's problem-solving context.^{35,36} The first two elements of the context are under the interface's direct control, but the third element requires the interface to understand the user's intentions and internal state. Although research into collaborative filtering abounds, a general method for understanding the user's complete problem solving context, given only observations of his behavior, has eluded the computational sciences for a very long time. In essence, the

task is a mind-reading problem very much like the spouse trying to understand the significant other without the use of explicit communication. The research presented here represents a new way to approach this task.

Acknowledgments

This paper is supported by the Dayton Area Graduate Studies Institute (DAGSI) under grant #HE-WSU-99-09 (ABMIC), by the Air Force Office of Scientific Research (AFOSR) under grant F49620-99-1-0244, by a grant from the Information Technology Research Institute (ITRI), and by the state of Ohio. We would also like to thank Dr. John Gallagher for his valuable comments.

References

- [1] Aho, A., Hopcroft, J., & Ullman, J. (1974). *The design and analysis of computer algorithms*. Reading, MA: Addison-Wesley.
- [2] Alexandersson, J. (1995). Plan recognition in verbmobil. In Mathias Bauer, Sandra Carberry, and Diane Litman, editors, *Proceedings of the ijcai-95 Workshop The Next Generation of Plan Recognition Systems: Challenges for and Insight from Related Areas of AI*, pages 2--7, Montreal, August 1995.
- [3] Albrecht, D. W., Zukerman, I., & Nicholson, A. E. (1998). Bayesian models for keyhole plan recognition in an adventure game. *User modeling and user-adapted interaction*, 8, 5-47.
- [4] Allen, J. F., and Perrault, C. R.: Analyzing intention in dialogues. *Artificial Intelligence* 15(3) (1980) 143-- 178.
- [5] Amarel, S. (1968). *On representations of problems of reasoning about actions*. In Michie (Ed.), *Machine Intelligence 3* (pp. 131-171). Evanston, IL: Edinburgh University Press.
- [6] Appelt D. E., & Pollack, M. E. (1992). Weighted abduction for plan ascription. *User Modeling and User-Adapted Interaction*, 2(1-2), 1-25.
- [7] Bares, M., Canamero, D., Delannoy, J. F., & Kodratoff, Y. (1994). XPlans: Case-based reasoning for plan recognition. *Applied Artificial Intelligence* 8, 617-643.
- [8] Bauer, M. (1996): Machine learning for user modeling and plan recognition. In V. Moustakis J. Herrmann, editor, *Proc. ICML'96 Workshop "Machine Learning meets Human Computer Interaction"* 5--16.
- [9] Bergmann, R., & Wilke, W. (1995). Building and Refining Abstract Planning Cases by Change of Representation Language. *Journal of Artificial Intelligence Research*, 3:53—118.
- [10] Bergmann, R., & Wilke, W. (1996). On the Role of Abstractions in Case-Based Reasoning. In *Proceedings of EWCBR-96 European Conference on Case-Based Reasoning*. Springer, 1996.
- [11] Bodlaender, H. L. (1990). Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. *J. Algorithms*, 11, 631-643.
- [12] Brown, S., & Cox, M. (1999). Planning for information visualization in mixed-initiative systems. In M. T. Cox (Ed.), *Proceedings of the 1999 AAAI-99 Workshop on Mixed-Initiative Intelligence* (pp. 2-10). Menlo Park, CA: AAAI Press.
- [13] Canamero, D., Kodratoff, Y., Delannoy, Bar'es, M., Case-Based Plan Recognition, Working Notes of the Workshop on Planning and Learning: On to Real Applications , AAAI--94 Fall Symposium Series, New Orleans, Louisiana, November 4--6, 1994, 16--22.
- [14] Carberry, S.: *Plan recognition in natural language dialogue*. MIT Press (1990).
- [15] Carbonell, J. G., Blythe, J., Etzioni, O., Gil, Y., Joseph, R., Kahn, D., Knoblock, C., Minton, S., Perez, A., Reilly, S., Veloso, M., & Wang, X. (1992). *PRODIGY 4.0: The Manual and Tutorial* (Tech. Rep. No. CMU-CS-92-150). Carnegie Mellon University, Department of Computer Science, Pittsburgh, PA.
- [16] Charniak, E., & Goldman, R. (1993), A Bayesian Model of Plan Recognition. *Artificial Intelligence* 64:53-79.

- [17] Cohen, P. (1999). The mind reading problem. In M. T. Cox (Ed.), *Proceedings of the 1999 AAAI-99 Workshop on Mixed-Initiative Intelligence* (p. 1). Menlo Park, CA: AAAI Press.
- [18] Colbourn, C. J., & Booth, K. S. (1981). Linear-time automorphism algorithms for trees, interval graphs, and planar graphs. *SIAM J. Comput.*, 10, 203-225.
- [19] Cox, M. T. (2001). Toward tailored information presentation in support of collaborative planning. In B. Bell & E. Santos (Eds.), *Intent Inference for Collaborative Tasks: Papers from the 2001 fall symposium* (pp. 44-50). AAAI Technical Report FS-01-05. Menlo Park, CA: AAAI Press.
- [20] Cox, M., Kerkez, B., Srinivas, C., Edwin, G., Archer, W. (2000). Toward agent-based mixed-initiative interfaces. In H. R. Arabnia (Ed.), *Proceedings of the 2000 International Conference on Artificial Intelligence, Vol. 1* (pp. 309-315). CSREA Press.
- [21] Cox, M. T., & Ram, A. (1999). *On the intersection of story understanding and learning*. In A. Ram & K. Moorman (Eds.), *Understanding language understanding: Computational models of reading*. (pp. 397-434). Cambridge, MA: MIT
- [22] Cox, M. T., & Veloso, M. M. (1998). Goal Transformations in Continuous Planning. In M. desJardins (Ed.), *Proceedings of the 1998 AAAI Fall Symposium on Distributed Continual Planning* (pp. 23-30). Menlo Park, CA: AAAI Press / The MIT Press.
- [23] Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 3, 251-288.
- [24] Fink, E. (1999). *Automatic representation changes in problem solving* (Tech. Rep. No. CMU-CS-99-150). Doctoral thesis, Carnegie Mellon University, Department of Computer Science, Pittsburgh, PA.
- [25] Fish, D. (1995). A dynamic memory organization for case-based reasoning supporting case similarity determination and learning via local clustering. Masters thesis, University of Connecticut, Computer Science Department, Storrs, CT.
- [26] Goodman, B.A. & Litman, D. J. (1992). On the interaction between plan recognition and intelligent interfaces. *User Modeling and User-Adapted Interaction*, 2(12) :55--82.
- [27] Hinrichs, T. R. (1992). *Problem solving in open worlds*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- [28] Kass, A. (1990). *Developing creative hypotheses by adapting explanations*. Doctoral dissertation, Northwestern University, The Institute for the Learning Sciences, Evanston, IL.
- [29] Kass, A. (1991). Adaptation strategies for case-based plan recognition. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society* (pp. 161-166). Hillsdale, NJ: Lawrence Erlbaum Associates.
- [30] Kass, A., Leake, D., & Owens, C. (1986). *SWALE: A program that explains*. In R. C. Schank, *Explanation patterns: Understanding mechanically and creatively*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- [31] Kautz, H. (1991). *A formal theory of plan recognition and its implementation*. In J. Allen, et. al., *Reasoning about plans*. San Francisco: Morgan Kaufmann.
- [32] Kerkez, B. (2002). Learning Plan Libraries for Case-based Plan Recognition. In *Proceedings of the 13th Midwest Artificial Intelligence and Cognitive Science Conference*. IIT, Chicago, IL.
- [33] Kerkez, B. (2001) *Incremental case-based keyhole plan recognition*. Technical Report, WSU-CS-01-01, Department of Computer Science and Engineering, Wright State Univ.
- [34] Kerkez, B., & Cox, M. (2001). Case-based plan recognition using state indices, In D. W. Aha & I. Watson (Eds.), *Case-based Reasoning Research and Development: Proceedings of 4th international conference on case-based reasoning* (pp. 227-242). Berlin: Springer.
- [35] Kerkez, B, & Cox. M. T. (2000). Planning for the user interface: Window characteristics. In *Proceedings of the 11th Midwest Artificial Intelligence and Cognitive Science Conference* (pp.79-84). Menlo Park, CA: AAAI Press.
- [36] Kerkez, B, Cox. M. T., & Srinivas, C. (2000). Planning for the user interface: Window content. In H. R. Arabnia (Ed.), In *Proceedings of the 2000 International Conference on Artificial Intelligence, Vol. 1* (pp 345-351). CSREA Press.

- [37] Kolodner, J. L. (1984). *Retrieval and organizational strategies in conceptual memory: A computer model*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- [38] Kolodner, J. L. (1989). Selecting the best case for a case-based reasoner. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: LEA.
- [39] Kolodner, J. (1993). *Case-based reasoning*. San Mateo, CA: Morgan Kaufmann.
- [40] Leake, D. (1992). *Evaluating explanations: A content theory*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- [41] Lesh, N., & Etzioni, O. (1996). Scaling up goal recognition. In *Proceedings of the Fifth Internat. Conference on Principles of Knowledge Representation and Reasoning* (pp 178-189).
- [42] Lesh, N., Rich, C. & Sidner, C. (1999). Using plan recognition in human-computer collaboration. In *Proceedings of the Seventh International Conference on User Modeling*, (pp. 23—32).
- [43] Luks, E. M. (1982). Isomorphism of graphs of bounded valence can be tested in polynomial time. *JCSSI*, 25, 42-65.
- [44] Mayfield, J. (1992). Controlling inference in plan recognition. *User Modeling and User-Adapted Interaction*, 2(1-2):55--82.
- [45] Newell, A., & Simon, H. A. (1972). *Human Problem Solving*. In E. A. Feigenbaum and J. Feldman (Eds.), *Computers and Thought* (pp. 279-293). Englewood Cliffs, New Jersey: Prentice Hall.
- [46] Owens, C. (1990a). *Indexing and retrieving abstract planning knowledge*. Doctoral dissertation, Yale University, Department of Computer Science, New Haven, CT.
- [47] Pollack, M. E. (1986). *Inferring domain plans in question-answering*. Doctoral thesis, University of Pennsylvania, Department of Computer Science, Pittsburg, PA.
- [48] Ram, A. (1994). *AQUA: Questions that drive the understanding process*. In R. C. Schank, A. Kass, & C. K. Riesbeck (Eds.), *Inside case-based explanation* (pp. 207-261). Hillsdale, NJ: Lawrence Erlbaum Associates
- [49] B. Raskutti, B. & Zukerman, I. (1991). Generation and selection of likely interpretations during plan recognition. *User Modeling and User-adapted Interaction*, 1(4): (pp. 322—353).
- [50] Schank, R. C. (1982). *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge University Press, New York.
- [51] van Beek, P. & Cohen, R. (1991). Resolving plan ambiguity for cooperative response generation. In *Proc. 12th IJCAI*, (pp. 938—944), Sydney.
- [52] Veloso, M. (1994). *Planning and learning by analogical reasoning*. Berlin: Springer.
- [53] Veloso, M. (1997). Merge strategies for multiple case plan replay. In Case-Based Reasoning Research and Development, *Proceedings of ICCBR-97, the Second International Conference on Case-Based Reasoning*, pages 413-424. Springer Verlag.
- [54] Veloso, M., Carbonell, J. G., Perez, A., Borrajo, D., Fink, E., & Blythe, J. (1995). Integrating planning and learning: The PRODIGY architecture. *Journal of Theoretical and Experimental Artificial Intelligence*, 7(1), 81-120.
- [55] Veloso, M., & Carbonell, J. G. (1994). *Case-based reasoning in PRODIGY*. In R. S. Michalski & G. Tecuci (Eds.), *Machine learning IV: A multistrategy approach* (pp.523-548). San Francisco: Morgan Kaufmann.
- [56] Weida, R. and Litman, D.: Terminological plan reasoning and recognition. In *Proceedings of the Third International Workshop on User Modeling* (1992) 177--191.
- [57] Wilensky, R. (1983). *Planning and understanding: A computational approach to human reasoning*. Reading, MA: Addison-Wesley Publishing.