

Local Predictions for Case-based Plan Recognition

Boris Kerkez and Michael T. Cox

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

Abstract. This paper presents a novel case-based plan recognition system that interprets observations of plan behavior using a case library of past observations. The system is novel in that it represents a plan as a sequence of action-state pairs rather than a sequence of actions preceded by some initial state and followed by some final goal state. The system utilizes a unique abstraction scheme to represent indices into the case base. The paper examines and evaluates three different methods for prediction. The first method is prediction without adaptation; the second is predication with adaptation, and the third is prediction with heuristics. We show that the first method is better than a baseline random prediction, that the second method is an improvement over the first, and that the second and the third methods combined are the best overall strategy.

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).

Although the case-based approach to plan recognition is not new (Bares *et al.*, 1994), the novelty of our approach primarily arises from the manner in which we represent a plan (i.e., a case) that records past observations and the representation of indices by which we store and retrieve such old observations when interpreting a current observation. Unlike most plan recognition systems (e.g., Kautz, 1991) that represent a plan as a sequence of actions bracketed by an initial state and a goal state, we represent a plan as a sequence of action-state pairs such that the initial pair is ($\langle \text{null-action} \rangle$, $\langle \text{initial-state} \rangle$), and the last action is ($\langle \text{final-action} \rangle$, $\langle \text{goal-state} \rangle$) (Kerkez and Cox, 2001). 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 (Veloso, 1994) that contains trucks that transport packages from one location to another. A single state in this domain might be the case 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 of two ground literals $\{ (at-obj \text{ Package-1 PostOffice-1}), (at-truck \text{ Truck-1 PostOffice-1}) \}$. An alternate state might be $\{ (at-obj \text{ Package-1 PostOffice-2}), (at-truck \text{ Truck-1 PostOffice-1}) \}$. 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 $(at-truck \text{ Truck-1 PostOffice-1})$ 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 respective generalized literal $(at-obj \text{ package location})$, 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,



Figure 1. A simple example of the logistics planning scenario with 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*.¹ The literals are shown in Table 1. In this domain an abstract state is represented as a feature vector having

Table 1. Ground literals representing the state in Figure 1

AT-TRUCK	TRUCK A	POST-OFFICE A	AT-AIRPLANE	PLANE B	AIRPORT B
AT-TRUCK	TRUCK B	AIRPORT B	AT-OBJ	OBJECT A	POST-OFFICE A
AT-TRUCK	TRUCK C	AIRPORT A	AT-OBJ	OBJECT C	POST-OFFICE B
AT-AIRPLANE	PLANE A	AIRPORT A	INSIDE-TRUCK	OBJECT B	TRUCK B

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 Table 1 is $[3 \ 2 \ 2 \ 1 \ 0]$. The reason 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

¹ 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.

simple states from the previous paragraph likewise, they would both be represented by the vector $[1\ 0\ 1\ 0\ 0]$.²

As detailed in Kerkez and Cox (2001), a plan can be represented in abstract form as a graph within an abstract space. 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. 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. When retrieving old cases

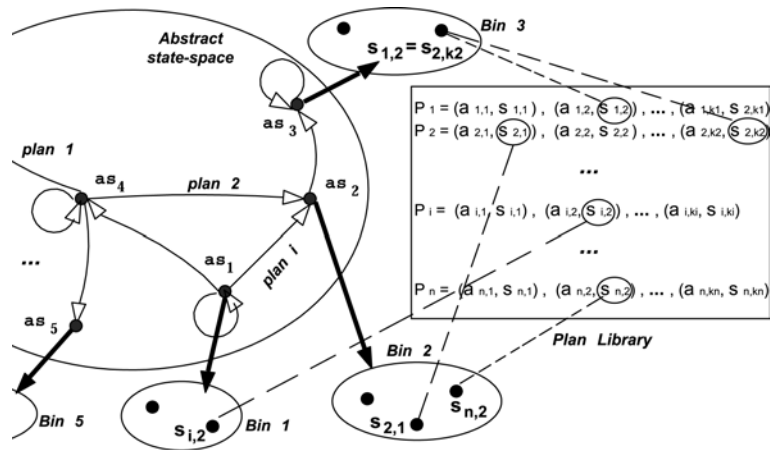


Figure 2. Indexing and storage structures. Abstract states (as_i) point to bins (bold lines), containing world states (s_j), which point (dashed lines) to past plans (P_j) in which they occurred

given a new observation, a system need only transform the currently observed concrete state to its corresponding abstract state (linear time), 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). As described in (Kerkez, 2001), our system does not operate with a complete plan library, but rather it incrementally builds its library from the observed planning behavior. Such incremental construction further decreases the complexity of the state-space and minimizes the existence of extraneous plans (Lesh and Etzioni, 1996).

The case-based plan recognition system introduced here can be classified with respect to the abstraction framework for the case-based reasoning (Bergmann and Wilke, 1996). The system explicitly stores concrete cases and abstract states, while the abstraction generation process is automatic. Although complete abstract cases are not used for indexing, abstracted situations form the basis of the indexing and the retrieval. Past abstract solutions are reused, because the next action prediction is first formed at the abstract level, after which the action is refined by specifying the action

² Like the abstract vector formed from Table 1, this vector excludes static predicates.

arguments. Case deletion policy is not yet implemented and therefore the system could have storage problems given a limited storage space.

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. The abstraction scheme is extremely simple in nature and applicable to a wide variety of planning domains. One requirement for the recognizer presented in this work is the ability to monitor the intermediate planning states, along with a prerequisite that the 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 be able to abstract 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.

Similar to the approach in the PARIS system (Bergmann and Wilke, 1995), the recognizer does not form abstraction by simply dropping sentences. 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.

This paper reports the results of an empirical evaluation of the recognition with the abstract indexing scheme. Given the observation of an action-state pair in a new plan, the system's task is to predict the next action that will be observed. First the system retrieves all cases that match the abstract representation of the observed state. Note that each case may contain more than one concrete state whose abstract representation is equal to the abstracted observed state. Therefore many candidate actions may exist across the retrieved cases that can serve as a prediction for the action to be observed next. Such predictions are local to a specific location in a plan, as opposed to a global prediction such as a goal for the entire plan.

We evaluate three different ways in which the action can be selected from the available candidates. The first and most simple method is to select the most common action in the candidate set. If more than one action is most frequent, the system chooses among the most frequent at random. Given that many candidates may still exist among this chosen action, the system chooses from the reduced set, again at random. A somewhat more intelligent method is to choose the most frequent action (randomly if needed), but then to compute an argument substitution for the action rather than to simply re-use the same arguments in the past case. The action with new arguments constitutes an adaptation of the old case element. The former method described above we call prediction without adaptation, whereas the latter we call prediction with adaptation. A third method is to use the knowledge of cases that worked already in the current sequence of predictions to bias the choice of candidate-action selection. We call this method prediction with heuristics. After a prediction is made using one of these three methods, the system will receive the next observation to determine the accuracy of the prediction and to provide the next state for subsequent predictions.

Given this description, the next section in this paper will examine the prediction without adaptation method. It will also present empirical results that show that the method outperforms the baseline performance that randomly chooses an action and arguments without the use of past cases. The following section will present results for prediction with adaptation. The subsequent section examines and shows results for prediction with heuristics. The paper ends with a brief summary and conclusion.

Local Action Prediction without Adaptation

The case-based plan recognition system presented in this work utilizes the indexing scheme described in the previous section as the basis for its reminding processes. After the planning agent executes a planning step, the recognizer observes the executed action, along with the world state of the planner reached by that action. The recognition cycle is initiated upon every observation of a planning action. The current planning situation (represented by the current world state of the planner) is transformed into its abstract form and subsequently matched to a bin. In case of a successful match, retrieval is focused on past situations within the single indexing bin.

The work presented here focuses on the local predictions of the planner's intent, in particular, the predictions of the next immediate action. The action predictions are formed at the two levels. Although concrete action predictions are much more informed than abstract predictions, the latter can also be very useful. For example, it is somewhat successful to recognize that the planner is about to load a truck, although we may not be able to tell with certainty which package is to be loaded. Our future research efforts will explore the predictions of the intermediate world states and the predictions of goals.

When the recognizer is able to find a unique past case having only one concrete state that matches the current abstract state, prediction is determined to be the past action following the matching state. However, when multiple past actions exist, the recognizer needs to determine which action is the most probable current intent of the planner. The next section describes the benefits of the abstract indexing in predicting the next action and evaluations of two different action-choosing strategies.

Experimental Results

To illustrate the practical benefits of next action prediction utilizing abstract indexing and retrieval, we present the evaluations of the action predictions in the logistics planning domain. The experimental evaluation focused on randomly generated planning problems with 3, 5, and 7 cities in the logistics domain. Evaluations presented in this paper concentrate mostly on the results for problems with three cities. This is because asymptotic behavior of the recognizer can clearly be observed given the smaller state-space for three cities. Next, two different problem sets with 5000 problems each were randomly generated for each number of cities. The random problem generator in each of these two problem sets was initialized with a different random

seed. Generated problems were solved by the PRODIGY state-space planner (Carbonell *et al.* 1992) that generated solutions to about 80% of the input problems.³

The solved planning problems and the generated solutions were then given as the input into the case-based plan recognition system. The system simulates plan execution by parsing the input data and by sending the planning steps to the recognizer one at the time. As described in the previous section, the system is concerned with predictions of the local planning behavior. The recognizer first predicts the next action at the abstract level. Subsequently, the recognizer refines it into a concrete action. Therefore, the accuracy of the concrete action predictions is bounded-above by the accuracy of the concrete action predictions.

We evaluated two different strategies for choosing the past planning action for the prediction in light of multiple action matches. Both of these strategies employ the abstract indexing scheme. The first strategy randomly selects the past planning action among all of the matches within a single bin, while the second strategy considers only the most frequently pursued past actions within the bin. Note that there may exist multiple most frequent past actions when many actions in a given past situation were pursued with an equal frequency. In the logistics domain with 3 cities, multiple most frequent actions are found in about 10% of cases, while in the domain with 7 cities, multiple most frequent actions were found in about 30% of cases. When multiple most frequent past actions are found, the recognizer randomly chooses among them.

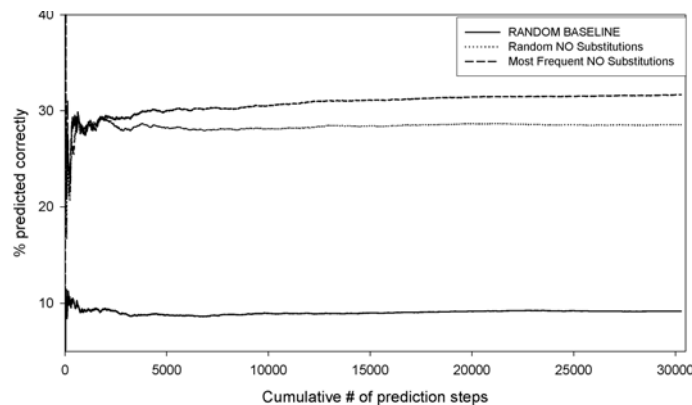


Figure 3. Percentages of correctly predicted *abstract* actions in 3-city logistics domain with abstract indexing scheme over 30,000 recognition steps

Figures 3 and 4 show the percentages of correctly predicted actions at the abstract and at the concrete level, respectively, for the two past action choice strategies and the baseline test. The baseline test consists of randomly choosing the past action out of all previously observed planning actions without the utilization of the abstract indexing and retrieval scheme. These results as well as all subsequent results in this paper are averages of the two problem sets that use different random seeds in the three-city lo-

³ The reason for the 20% planning failure rate is due to the fact that some random problems are impossible to be solved. In these situations the planner returns from its execution cycle without creating a plan once appropriate maximum execution time threshold is reached.

gistics domain. It is clear from the above figures that utilization of the abstract indexing scheme indeed focuses the prediction choices and significantly improves the prediction accuracy with respect to the baseline. The most frequent action strategy performs slightly better than random selection of a past action extracted from the cases retrieved using the abstract indexing scheme as in Figures 3 and 4.

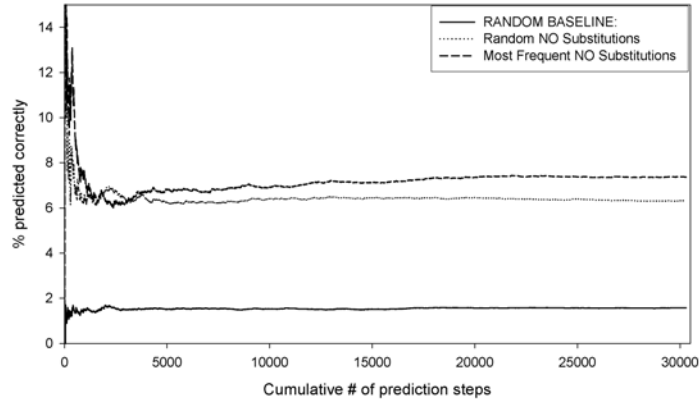


Figure 4. Percentages of correctly predicted *concrete* actions in 3-city logistics domain with abstract indexing scheme over 30,000 recognition steps

Although the most frequent strategy performs better than the random action choice, the improvement is not very significant. This is because of the fact that at this point the recognizer does not perform argument substitutions in the chosen action. That is, the recognizer performs the next action prediction from a previous case without adaptation of the arguments of a previous case. The next section discusses the problems with argument substitutions in general and proposes a feasible, non-optimal method for detecting argument substitutions.

Local Predictions with Adaptation

Instead of re-using an action selected from a retrieved case as is, adaptation involves calculating a set of new arguments that can be substituted for the old arguments. In certain complex domains, however, the cost associated with finding an argument substitution for an action can outweigh the benefits of substitutions themselves. To illustrate the predicate argument substitutions, consider Figure 5 with two different world states from a simple blockworld planning domain. The two states in the figure have the same abstract representation and the 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 5, the substitution σ defined by

$$\sigma = \{A/B, B/A, C/D, D/C\} \quad (1)$$

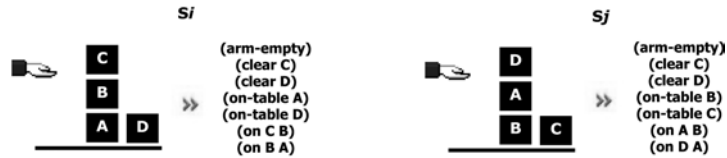


Figure 5. Two world states s_i and s_j from the simple blocksworld planning domain

represents the unique substitution of arguments of state s_i into arguments of state s_j . If state s_i is the currently observed state and state s_j 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, then the substitution mechanism enables the recognizer to adapt the argument of the predicted pick-up action from *blockC* to *blockD*, because *blockD* from state s_j substitutes for *blockC* in state s_i . The argument substitution can also help the recognizer find the most similar previous situation among potential multiple matches. For example, a past state with most identity substitutions ($\sigma(a)=a$) can be considered the most similar past situation for the currently observed planning state.

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 6 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



Figure 6. An example of a state from the blocksworld domain where the complexity of the substitution scheme is large

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

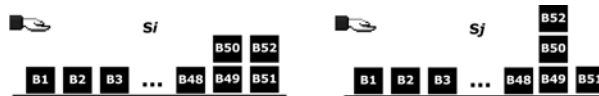


Figure 7. Two structurally different blocksworld states with identical abstract representations

Figure 7 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 case of the example in Figure 7, 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.

Our approach of coping with the problem of the argument substitutions utilizes techniques common for handling intractable problems. We introduce a simple and computationally effective, sub-optimal process that enables the recognizer to perform argument substitutions. This process utilizes a change of representation technique in which world states represented by ground literals are transformed into corresponding graphs, called *state graphs* (Kerkez and Cox, 2001; Kerkez, 2001). Such representational change facilitates the complexity analysis of the argument substitution scheme.

Practical Adaptation with Argument Substitution

It is possible to utilize knowledge about a domain theory to assure that the argument substitution scheme for state predicates is feasible. When all state graphs in a given planning domain belong to a 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 may not all be planar (Kerkez, 2002).

Another approach to cope with the need for the argument substitution is to limit the substitutions to a subset of all domain objects. A natural choice is to limit the substitution to include only the arguments of a predicted 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 isomorphism mapping of two graphs induced by the vertices that are the arguments of the considered action. Because the induced graphs can be much smaller than the complete state graphs, finding isomorphism among induced graphs could be simpler. In extreme cases, a vertex representing an action argument can be adjacent to a large number of vertices or even to every other vertex. In such situations, the complexity associated with the argument substitutions is still too great.

For the limited approach to substitution, the recognizer focuses on the next action prediction and attempts to find all possible substitutions of the action arguments only. The substitution process uses the information contained in the transformed state graphs, concentrating on those vertices that represent the currently matched action arguments. The substitution finding process is neither correct nor complete, because it is not guaranteed to find any correct substitutions when one exists. On the other hand, the process is linear in the number of arguments, and it is therefore very efficient.

To explain the substitution algorithm, consider an example with two states s_1 and s_2 with their corresponding state graphs G_1 and G_2 , respectively. Let $G_1=(V_{G_1}, E_{G_1})$, $G_2=(V_{G_2}, E_{G_2})$, with

$$V_{G_1} = \{v_1, \dots, v_k\}, V_{G_2} = \{u_1, \dots, u_k\}.$$

Let us assume that state s_1 is the currently observed state, while state s_2 is the previously observed state that needs to be adapted. Further assume that the action taken in the state s_2 was action A with arguments $u_{A1}, \dots, u_{Aq <= k}$. Given two states graphs G_1 and G_2 , the process of finding the argument substitutions proceeds as follows:

For each action argument u_{Ai} ,

- Find all of the edges in the graph G_2 incident to u_{Ai} .
- Find all vertices v_j in G_1 that have exactly the same edge incidence set as u_{Ai} .
- If no such vertices can be found, no substitutions exist. Otherwise, vertices v_j in G_1 are possible substitutions for the vertex u_{Ai} in action A .

The above process is linear in the number of edges and vertices in the subgraph induced by the vertices acting as the action arguments. This process enables the recognizer to quickly determine that no substitutions exist between the two given states, because the edge incidence set of some vertex u_{Ai} in G_2 may not have a match in the graph G_1 . The process may also find some substitutions that are not applicable outside the subgraph induced by the action arguments and therefore are not feasible substitutions between the two states. This sub-optimal behavior is compensated by the efficiency and the simplicity of the proposed substitution finding process.

Experimental Results

The substitution finding process described in the previous section was tested on the same two problems from the logistics planning domain discussed in the previous

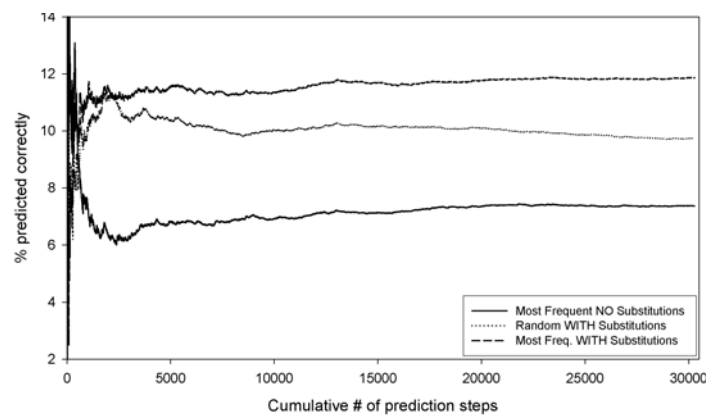


Figure 8. Percentages of correctly predicted concrete actions in the three-city logistics domain, with the simple and effective argument substitutions scheme

results section. The choice of the action at the abstract level remains the same as before, thus the substitution scheme affects only the effectiveness of the action predictions at the concrete level.

Although the process of finding the substitutions is quite simple, it helps the recognizer increase the concrete action prediction accuracy. Figure 8 shows the percentages of the correctly predicted concrete actions over 30,000 recognition steps. The bottom line represents the best performer from previous evaluations in Figure 4. As it can be seen from Figure 8, the random action choice strategy with the substitution scheme outperforms the most frequent action choice strategy without substitutions in a long run. As expected, the most frequent action choice strategy with adaptation significantly outperforms both of its two rivals. Figure 9 shows the comparison of the total

numbers of correctly predicted concrete actions over 30,000 prediction steps. In each case it can be seen that the usage of the argument substitutions increases the recognizer's prediction accuracy regardless of the action choice strategy used. The substitution finding technique is also domain-independent, as long as the states and actions can be represented in a predicate form.

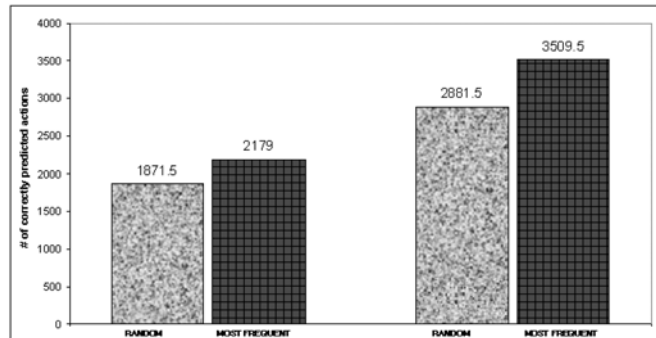


Figure 9. Number of correct concrete action predictions without (left) and with (right) action argument substitution strategies

Although the improvements introduced by the sub-optimal substitution finding mechanism are evident, the overall prediction accuracy is still not very high. Because a concrete action selection is made only after the abstract action selection, the abstract action prediction accuracy serves as an upper bound for the concrete action prediction accuracy. We can see from the figures that in the best case, about $2/5$ of correctly predicted abstract actions remain correct at the concrete level. The next section introduces additional ways in which the concrete action prediction accuracy may come closer to the upper bound determined by the abstract prediction accuracy.

Local Predictions with Failure-Driven Heuristics

We have demonstrated that the use of a simple and efficient argument substitution finding scheme improves the accuracy of the recognizer. The simplicity of the recognition scheme leaves a lot of room for the improvement. The accuracy of the action predictions at the concrete level is bounded above by the accuracy at the abstract level, because the choice of the abstract action determines the choice of the concrete action. Therefore, we are interested in improving the prediction accuracy at both levels of abstraction.

Currently, there are two different strategies for choosing the next action at the abstract level: random and most frequent strategies. The former strategy randomly chooses an action extracted from the retrieved cases, while the most frequent strategy chooses the action that was pursued with the greatest frequency in previous situations. As before when multiple most frequent actions exist, the recognizer randomly chooses one. One way of improving the abstract prediction accuracy is to replace the

random choice of an abstract action in a case of a tie with a better selection technique. An example includes preferring the last recently used actions.

At present time, choosing the concrete actions is accomplished both with and without the action arguments substitution. As the previous results show, the substitution scheme is better in predicting the planner's next action. However, in cases where multiple substitutions are possible, the recognizer simply picks a single substitution at random. One obvious way of improving the concrete action predictions is to explore alternative methods of breaking this tie. Our future research will further address the techniques for improving the prediction accuracy at the both levels of abstraction.

It is certainly true that there exist numerous heuristics, whose application may improve the prediction accuracy. In attempt to improve the recognition accuracy, we constrain the recognition process to attempt retrieval of previous cases only in light of a prediction failure. Learning from failure is a cornerstone of many artificial intelligence systems (e.g., Cox and Ram, 1999; Pazzani, 1994). When the recognizer's previously made prediction turns out to be accurate, the recognizer assumes that the correct past case has been found and chooses to form subsequent predictions on the basis of the correct past case. This past case is utilized for the recognition for as long as the recognizer's predictions are valid. As soon as the recognizer makes an incorrect prediction, the retrieval phase starts again and the past matching case is discarded.

Although the use of the heuristic that initiates the retrieval only when predictions fail is fairly simple, the improvement in the prediction accuracy are evident. The next section discusses an evaluation of this simple yet effective heuristic.

Experimental Results

To evaluate the effectiveness of failure-driven retrieval, we present the prediction results based on the two three-city logistics problem sets we used in previous evaluations. Unlike the substitution finding scheme that only improves the predictions at the

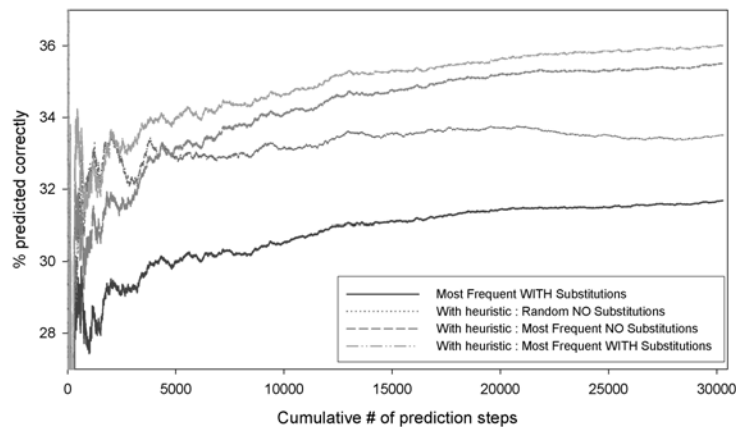


Figure 10. Percentages of correctly predicted *abstract* actions with the failure-driven heuristic

concrete action level, the failure-driven heuristics influences predictions at the abstract action level. Because the abstract actions are chosen before the concrete actions, the failure-driven heuristic affects predictions at both levels of abstraction.

Figure 10 shows the action prediction percentages at the abstract action level. The bottom line in the graph is the scheme that utilizes the most frequent action selection and substitutions of action arguments. This is the best performer from previous evaluations. Note that even the random action selection scheme without substitutions and with the failure-driven heuristic outperforms the best performer from previous evaluations. Because concrete action prediction accuracy is bounded above by the abstract action prediction accuracy, there is even more room for the improvements in predicting the concrete actions.

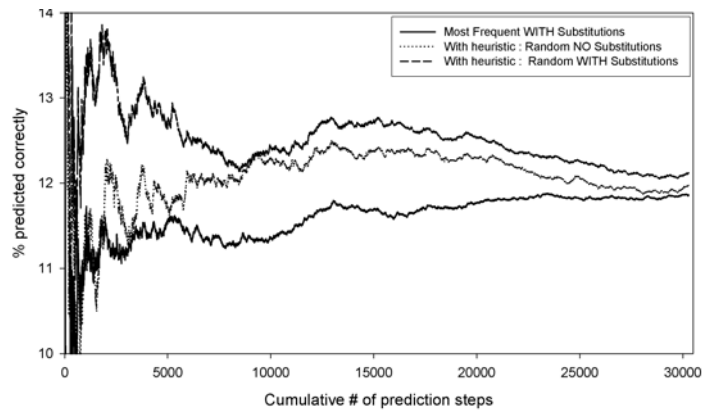


Figure 11. Percentages of correctly predicted *concrete* actions with the failure-driven heuristic and the random action selection strategies

Figure 11 shows the two different random strategies with the failure-driven heuristic compared to the best performer from previous graphs with the most frequent action selection. These evaluations show that while the failure-driven heuristic increases the accuracy percentages of the random predictions initially, in the long run the randomness brings the slope of the curves down to the accuracy level of the most frequent action selection strategy. This is because over a long period of time the density of the concrete states in the indexing bins increases and the random selection strategy is less likely to pick the correct next action. On the other hand, the slopes of the curves representing the most frequent action selection strategy do not decrease in a long run.

Figure 12 shows the accuracy percentages of the most frequent prediction strategies with and without the failure-driven heuristic. Unlike random, the most frequent strategy only gets better with the increased number of observations, because the action frequency distribution becomes more uniform. Figure 13 shows the total numbers of the correctly predicted next concrete actions over 30,000 recognition steps. Two best performers from previous evaluations are shown on the left, both with and without the argument substitution schemes. Note from Figure 12 and Figure 13 that the prediction accuracy of the best performer from the previous evaluations is slightly better than the

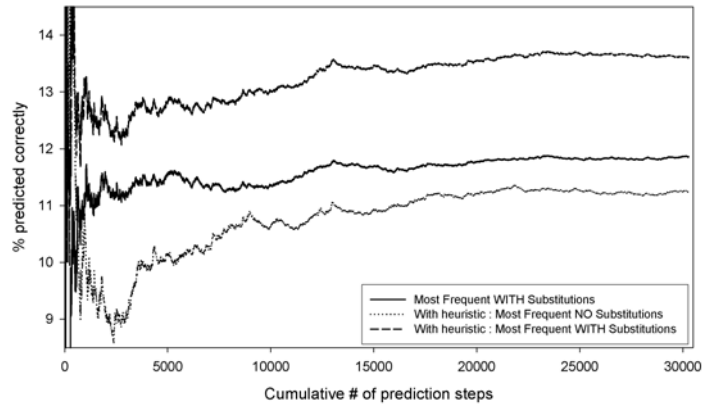


Figure 12. Percentages of correctly predicted *concrete* actions with the failure-driven heuristic and the most frequent action selection strategies

most frequent action selection strategy with the failure-driven heuristic and without substitutions. But unlike the random selection strategies, the most frequent strategies maintain the prediction percentage curves with a small positive slope in a long run. Future research efforts will increase the size of the problem sets in order to analyze the asymptotic behavior of the plan recognition system. As it can be seen from the graphs, the failure-driven heuristic along with the most frequent action selection strategy and substitutions is superior to the all other strategies on this particular dataset.

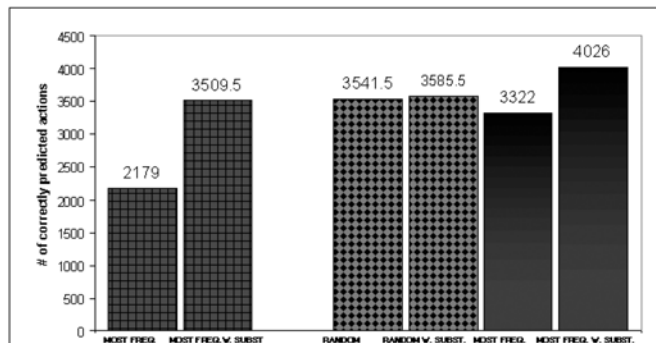


Figure 13. Number of correct *concrete* action predictions with (right) and without (left) the failure-driven heuristic strategies.

The evaluation results presented in this paper show that the utilization of the abstract indexing scheme along with the argument substitutions and the failure-driven heuristics is the best recognition strategy among all other evaluated strategies. In general, the increase in the prediction accuracy of the argument substitutions scheme alone is more beneficial for the recognition than the failure-driven heuristic alone. However, when these two strategies are used side by side, the prediction accuracy is better than any one of them used alone.

Conclusions

This paper presented an approach to the recognition of the planning actions utilizing a novel indexing scheme in the context of the case-based plan recognition with incomplete plan libraries. The application of the abstract indexing scheme increases the prediction accuracy with respect to the baseline test, while the argument substitution mechanism further improves the recognition accuracy. The failure-driven heuristic further increases the recognizer's accuracy and indicates the potential of the heuristic-driven approaches. Future research efforts will explore the global (goal) predictions, as well as other potentially promising and efficient heuristics.

Acknowledgments

This paper is supported by the Dayton Area Graduate Studies Institute (DAGSI) under grant #HE-WSU-99-09 (ABMIC), by a grant from the Information Technology Research Institute (ITRI), and by the state of Ohio.

References

- Bares, M., Canamero, D., Delannoy, J. F., & Kodratoff, Y. (1994). XPlans: Case-based reasoning for plan recognition. *Applied Artificial Intelligence* 8, 617-643.
- Bergmann, R., & Wilke, W. (1995). Building and Refining Abstract Planning Cases by Change of Representation Language. *Journal of Artificial Intelligence Research*, 3:53—118.
- Bergmann, R., & Wilke, W. (1996). On the Role of Abstractions in Case-Based Reasoning. In *EWCBR-96 European Conference on Case-Based Reasoning*. Springer, 1996.
- 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.
- Cox, M. T., & Ram, A. (1999). Introspective multistrategy learning: On the construction of learning strategies. *Artificial Intelligence*, 112, 1-55.
- Kautz, H. (1991). A formal theory of plan recognition and its implementation. In J. Allen, *et al.*, *Reasoning about plans*. San Francisco: Morgan Kaufmann.
- Kerkez, B. (2001) *Incremental case-based keyhole plan recognition*. Technical Report, WSU-CS-01-01, Department of Computer Science and Engineering, Wright State Univ.
- 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.
- 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.
- Lesh, N., & Etzioni, O. (1996). Scaling up goal recognition. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning* (pp 178-189).
- Pazzani, M. (1994) Learning causal patterns: Making a transition from data-driven to theory-driven learning, in: R. Michalski and G. Tecuci, eds., *Machine learning IV: A multistrategy approach* (Morgan Kaufmann, San Francisco, 1994) 267-293.
- Veloso, M. (1994). *Planning and learning by analogical reasoning*. Berlin: Springer.