An Adaptive Planner Based on Learning of Planning Performance

Kreshna Gopal **Thomas R. Ioerger**
Department of Computer Science, Texas A&M University, College Station, TX, U.S.A.

Abstract

Case-based planners often face the problem of incurring more computational cost for retrieving and modifying a case for reuse, than what can be saved by reusing the case. We present a case-based planning system that learns the performance of a given planner (called the default planner) in a training phase, and exploits this knowledge to retrieve and reuse cases such that planning effort is saved. The system does not involve any modification of the plan being reused. Furthermore, the system uses a very efficient method for matching a new problem with solved cases. The average-case performance of the system has been found to be significantly better than that of the default planner in a test domain. We hypothesize that this approach can be used to improve the performance of other planners as well. The effectiveness of the system hinges mainly on the learning strategy and on extraction of the relevant features.

Keywords: case-based planning, learning, feature extraction

1 Introduction

Domain-independent planning has been shown to be intractable and undecidable [2,3,4]. Thus, saving and adapting previously constructed plans in order to improve the average-case efficiency of planning by avoiding repetition of planning effort is largely regarded as a promising approach. Earliest research on planning from experience is the motion planning of the robot named *Shakey* [6], which is based on the STRIPS planning system [7]. Other systems based on this approach include PLEXUS [1], CHEF [8] and PRIAR [9].

The most serious bottlenecks of case-based planners are retrieval and modification of a good case. Nebel and Koehler [13] argue that it is not possible to achieve a provable efficiency gain of plan reuse over plan generation in the worst case. They show that if planning from scratch is PSPACE-complete or NP-complete, then modifying an existing plan to solve the problem is also PSPACE-complete or NP-complete, respectively. The reverse does not hold, however. That is, there are cases where plan generation can be done in polynomial time, while plan modification is NP-complete. Furthermore, the problem of syntactically matching a new problem to a candidate case for reuse is NP-hard. It should be noted that the matching problem has to be solved for each potentially relevant candidate from the plan library in order to select an optimal reuse candidate.

We present a planning system that involves essentially a default planner, a plan library and learning from a set of training examples (cases solved by the default planner). To solve a new problem, a good case is retrieved from the library, such that the original task of finding a plan is reduced to that of finding two simpler sub-plans. The first sub-plan transforms the initial state of the new problem into the initial state of the retrieved plan. The second sub-plan transforms the goal state of the retrieved plan into the goal state of the new problem. If an appropriate case is retrieved, these two sub-plans can be constructed by the default planner with less computational effort, and they can be combined with the retrieved plan to generate a solution. The goal of the training phase is to learn to predict the time the default planner would take to construct plans from scratch and this knowledge is used to guide the retrieval of a good case from the plan library.

The system does not involve any modification of the plan being reused. Furthermore, matching a new problem with solved cases is very efficiently performed. These are salient aspects of the system, in light of the arguments made by Nebel and Koehler on the potential cost of plan modification and problem matching.

We used a STRIPS-based planning system (by N. Short and L. Dickens, 1992) as the default planner, a neural network for training and the system was tested in the blocks-world domain. This is a simple domain which consists of a set of blocks, a table and a robot arm that can move one block at a time. A planning problem is to find a sequence of block movements that will change an arrangement of blocks into another.

A significant improvement in planning performance was observed on the average in the blocks-world domain. We hypothesize that this method can be used to improve the average-case performance of other planners as well. The success of the system depends mainly on how accurate the performance prediction is. And the major determinants of successful learning of performance prediction are the learning method employed and the relevance of features extracted.

2 The training phase

The main objective of the training phase is to learn to predict, with reasonable accuracy, the time the default planner will take to solve a problem. We assume that the time taken for planning is predictable i.e. if the planner is made to solve the same problem a number of times, then there is not too much variance in the time taken.

The training examples can be constructed by running the default planner on a number of randomly generated (or manually created) problems. Each problem is run several times, and the average time taken can be treated as the actual problem solving time. If there is still high variance, then the lowest and highest few outlying values of time can be dropped before taking the average.

It can be hypothesized that the time the default planner will take to solve a problem is a function of the features of the problem. Thus, an important step in the training phase is to extract the relevant features that determine the planning time. In our implementation of the system in the blocks-world domain, both domain-dependent features as well as domain-independent features have been used. The number of conditions in the goal state that are already satisfied in the initial state is an example of a domain-independent feature. The number of stacks in the initial and goal states are examples of domain-dependent features. The feature weights were learned using a neural network with a single unthresholded linear unit, producing a linear combination of weighted feature values.

3 Features

Extraction of the right features is one of the most significant factors determining the effectiveness of the proposed system. This requires insight both in the problem domain as well as in the algorithm of the default planner. In real-world applications, feature extraction is often done manually by domain experts [5], though automated feature extraction methods have also been developed [10].

The proposed system has been implemented using various combinations of features. When only domain-independent features were used, the performance of the system was worse than that of the default planner. A very significant improvement in performance was observed as domain-dependent features were included. This is consistent with the claim that domain characteristics have to be exploited for efficient planning [14].

4 Plan retrieval and reuse

4.1 Notation

- Let $<I, G>$ represent a problem instance i.e. we need to find a plan which will take the world from an initial state I to a goal state G.
- Let *LIBRARY* be a set of n cases $<I_j, G_j, P_j>$, $1 \leq j \leq n$, where P_j is a plan which takes the world from I_j to G_j.
- Given two states S_i and S_j, let $Distance(S_i, S_j, w)$ be the time the default planner is predicted to take to find a plan which takes the world from S_i to S_j. The computation of the predicted time is based on the feature

weight vector w obtained in the training phase.

4.2 Algorithm

Given a new problem $<I_{new}, G_{new}>$, let P_{new} be a plan the default planner would return. We consider all the n cases $<I_k, G_k, P_k>$, $1 \leq k \leq n$, in *LIBRARY*. Let $P_{I,k}$ and $P_{G,k}$ be plans generated by the default planner for problems $<I_{new}, I_k>$ and $<G_k, G_{new}>$ respectively. Thus, a concatenation of $P_{I,k}$, P_k and $P_{G,k}$ constitutes a solution to $<I_{new}, G_{new}>$ (Fig. 1).

$$\begin{tikzcd} I_{new} \arrow[r] \arrow[d, "P_{I,k}"'] & G_{new} \\ I_k \arrow[r, "P_k"'] & G_k \arrow[u, "P_{G,k}"'] \end{tikzcd}$$

Fig. 1. Scheme of reuse

The objective is to retrieve the best case b in *LIBRARY*, and then call the default planner to generate plans $P_{I,b}$ and $P_{G,b}$. These plans will be combined with P_b to generate a solution. The best case b will be the one which saves computational time the most, as compared to developing P_{new} from scratch. But we first check if *Distance*(I_{new}, G_{new}, w) is less than a pre-defined threshold, *MinTime*, which corresponds to a minimum estimated difficulty of solving the problem using the default planner. If it is, then the problem is expected to be sufficiently inexpensive, such that resorting to reuse is unlikely to save time. In this case, the default planner is used to solve the problem.

Otherwise we find *Distance*(I_{new}, I_k, w) and *Distance*(G_k, G_{new}, w) for all cases in *LIBRARY*. The best case $<I_b, G_b, P_b>$ is the one with the highest *Gain*, where *Gain* of the k^{th} case for problem $<I_{new}, G_{new}>$ and weight vector w, is given by:

$$Gain = \frac{Distance(\mathrm{I}_{new}, \mathrm{G}_{new})}{Distance(\mathrm{I}_{new}, \mathrm{I}_k) + Distance(\mathrm{G}_k, \mathrm{G}_{new})}$$

If the highest *Gain* is smaller than a pre-defined threshold, *MinGain*, we again resort to the default planner. Otherwise the *Gain* is deemed adequate, and the default planner is called to solve $<I_{new}, I_b>$ and $<G_b, G_{new}>$, which returns plans $P_{I,b}$ and $P_{G,b}$ respectively. A concatenation of plans $P_{I,b}$, P_b and $P_{G,b}$ is a solution to $<I_{new}, G_{new}>$. The algorithm is shown in Fig. 2.

```
Inputs:  LIBRARY, w, MinGain, MinTime and a
         new problem <I_new, G_new>

if  Distance(I_new, G_new) < MinTime
then  Call default planner to solve <I_new, G_new>
else
  {
    MaxGain = -∞          /* MaxGain records the
                             maximum Gain so far */
    for k = 1 to n do     /* There are n cases in
                             LIBRARY */
      {
        k^th case = <I_k, G_k, P_k>
        D_new =  Distance(I_new, G_new, w)
        D_initial = Distance(I_new, I_k, w)
        D_goal  = Distance(G_k, G_new, w)
        Gain = D_new/[D_initial + D_goal]
        if Gain > MaxGain then
            {
               MaxGain = Gain
               b = k  /* b is the best case
                         found so far */
            }
      }
   if  MaxGain > MinGain
   then
     {
       Call default planner to solve <I_new, I_b >,
          which returns P_I,b
       Call default planner to solve <G_b, G_new>,
          which returns P_G,b
       Return concatenation of P_I,b , P_b and P_G,b
     }
   else
       Call default planner to solve <I_new, G_new>
 }
```

Fig. 2. The algorithm for plan retrieval and reuse

5 Empirical evaluation

Results obtained from experiments in the blocks-world domain are presented and analyzed in this section. The programs for learning, plan retrieval and reuse as well as the default planner were written in Allegro Common LISP. The data reported were obtained by running compiled Common LISP code on a SunSPARC-4. Problems for different numbers of blocks were treated as different classes of problems with their own training examples, features, weight vectors and libraries. The feature weight vectors for different numbers of blocks were found during the learning phase, where 1000 examples were used to train an unthresholded single-output perceptron, using gradient descent search.

There were around 1000 cases in the plan libraries. Problems with more blocks required more cases in its library so as to have sufficiently good cases to make reuse effective. But more cases led to an increase in search time.

5.1 Optimum gain in efficiency

First, the upper bound on performance improvement of reuse is assessed. That is, the maximum possible gain in efficiency that can be obtained by planning with reuse compared to planning from scratch is analyzed. In fact, the actual best case is identified in a given library. The *Gain* metric used for retrieving the best case is only an imperfect measure of the appropriateness of a case. The optimum efficiency gain is found by exhaustively trying the reuse of each and every plan in the library for a given problem.

Table 1 compares the mean time taken over 10 problems for 3-7 blocks (in milliseconds) by the default planner to the mean optimum time of planning with reuse, using a library of 1000 cases. Fig. 3 shows the optimum reuse time as a percentage of the default planner time.

It can be observed that the optimum time of planning with reuse is significantly less than the default planner's time if a library of 1000 cases is used. It follows that, given a problem, there are good plans in this library of 1000 cases which can be used and thereby saving on planning time. And with more cases in the library, there would be even better plans. The problem is how to find the best possible plan. Also, how much time will it take to find this best possible plan? Will the overhead of search surpass the savings obtained by reuse? Also, if we reduce the size of the library so as to decrease search time, then the probability of having (and finding) a good case also decreases. A suitable trade-off has to be found. As will be discussed in the next section, assuming successful learning of the default planner's performance, a judicious choice of the library size, *MinTime* and *MinGain* can lead to efficiency gain with reuse, even if the search time is accounted for.

Table 1. Default planner vs optimum reuse.

No of blocks	*Average default planner time (ms)*	*Average optimum time (ms)*
3	65	8
4	288	52
5	780	143
6	2262	275
7	9156	533

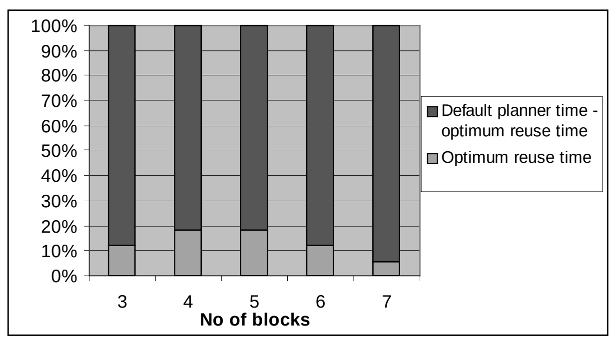

Fig. 3. Default planner vs optimum reuse

5.2 Efficiency gain with reuse

Table 2 shows the average performances (over 30 randomly generated problems for 3-7 blocks) of the default planner and of planning with reuse (both including and excluding search time). The same 30 problems were solved by both methods. The parameters used, namely the number of cases in *LIBRARY* (n) and the minimum time required to resort to reuse (*MinTime*) are shown. *MinGain* was fixed to 1.0. Time performance is in milliseconds. The size of the library had to be appropriately chosen for gain in average performance. The size ranged from 50 (for 3 blocks) to 900 (for 7 blocks).

Table 2. Default planner vs planning with reuse.

No of blocks	*MinTime (ms)*	*n*	*T_1 (ms)*	*T_2 (ms)*	*T_3 (ms)*
3	10	50	65	18	46
4	50	75	278	148	211
5	200	250	993	523	749
6	1000	850	2599	1033	1744
7	1500	900	6536	3075	4068

T_1 = Average time using default planner
T_2 = Average time of reuse (excluding search)
T_3 = Average time of reuse (including search)

Fig. 4 displays the results in Table 2 graphically. Time for searching and time for planning with reuse (excluding search time) are expressed as a percentage of the default planner time.

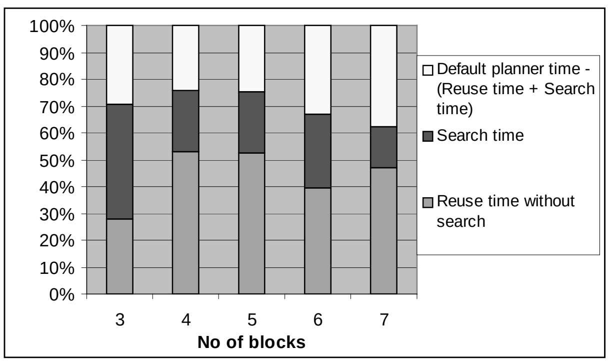

Fig. 4. Default planner vs planning with reuse

The samples of 30 time performances for each row in Table 2 are dependent and thus, we use the paired *t* test (based on Student's *t*-distribution) to evaluate the hypothesis that planning with reuse is more efficient than the default planner. This is done by computing the test statistic *t* defined below, where D is the mean of the differences between the two time performances, S_d is the standard deviation of the performance differences and n is the sample size.

$$t = \frac{D}{S_d/\sqrt{n}}$$

The results are summarized in Table 3, which shows the values of *t* for performance of the default planner compared to planning with reuse (both including and excluding search time).

Table 3. *t*-values for default planner vs reuse.

No of blocks	*t (excluding search time)*	*t (including search time)*
3	3.968	1.729
4	3.956	2.234
5	3.856	1.940
6	2.363	1.318
7	2.844	2.041

We can observe that all values of *t* in the second column of Table 3 are greater than 2.045 ($t_{.025,29}$) i.e. significant at a 95% confidence level or higher. Thus, we can conclude that solving the two sub-problems $<I_{new}, I_b>$ and $<G_b, G_{new}>$ takes less time than solving the original problem $<I_{new}, G_{new}>$ on the average. In the third column of Table 3, all values of *t* are greater than 1.311 ($t_{.1,29}$) i.e. significant at an 80% confidence level or higher. That is, even if search time is taken into consideration, the performance improvement is significant, although at a lower confidence level compared to when search time is excluded. This is probably due to high variance of the individual planning times.

5.3 Efficiency gain and library size

There are two ways in which the size of the library determines the effectiveness of the system: (1) the bigger the library, the more time is spent on search, and (2) more cases in the library will potentially lead to retrieval of a more appropriate case for a given problem. In this section, the time taken for planning with reuse is compared to the time taken by the default planner for different library sizes. The search time is ignored; this will enable an objective assessment of the impact of the library size on the gain of solving the two sub-problems over solving the entire problem from scratch. A higher efficiency gain implies that a more appropriate case was retrieved from the library.

Table 4 shows the average percentage gain in efficiency of planning with reuse over planning from scratch (over 30 problems, ignoring search time) for different library sizes. The *MinTime* used is indicated. *MinGain* was fixed to 1.0.

Table 4. Efficiency gain and library size.

No of blocks	*MinTime (ms)*	*% gain in efficiency of planning with reuse over planning from scratch (ignoring search time)*			
		Library size			
		250	500	750	1000
3	10	90.9	91.6	95.1	96.0
4	50	68.1	73.7	78.4	77.0
5	200	37.8	44.8	53.7	55.0
6	1000	-6.1	18	31.9	44.4
7	1500	-2.5	16.7	20.1	24.2

It can be observed that the efficiency gain increases with library size. Furthermore, problems with more blocks will have less efficiency gain for the same number of cases in the library. These results are as expected, and they lead to the conclusion that the size of the library has to be carefully chosen, since it is a trade-off between time gained in planning and time spent searching the library.

6 Conclusions

Improving planning efficiency by adapting previously constructed plans has received widespread research interest. But most case-based planners involve plan modification, and therefore are not a guaranteed cure for intractability because plan modification has been shown to be at least as expensive as generating a plan from scratch, in the worst case [13]. Furthermore, syntactically matching a problem with a solved case has been shown to be NP-hard [13]. We propose a simple case-based planning approach that involves learning attributes like performance of a given default planner in a training phase, and exploiting this knowledge to retrieve a case, which when reused, can reduce the original problem into a computationally less expensive one for the default planner. We tested the proposed planning system in the blocks-world domain, and we found that, on the average, planning with reuse was significantly faster than the default planner we chose. This method can be used to improve the performance of other planners as well.

The system does not entail any plan modification, which represents a major advantage in terms of time complexity. Furthermore, the system can perform problem matching very efficiently, since a neural network is used to assess the suitability of a case.

Varying combinations of simple features were used in the blocks-world domain. When solely domain-independent features were used, there was no gain in planning efficiency. Domain-dependent features had to be included for planning with reuse to outperform the default planner.

Domain knowledge is widely recognized as being indispensable for improving planning efficiency in complex environments [11,14]. In this system, domain knowledge is acquired in the form of features, which are related to knowledge about both the problem domain as well as the default planner. The relevance of features is related to how much they reflect the difficulty of a problem.

Despite the fact that domain knowledge is used, the planning system remains essentially domain-independent. Domain knowledge is an explicit input to the planning algorithm in the form of a feature weight vector, just like the domain specification. Feature extraction can be done with the help of domain experts in many ways [5], or they can be automatically extracted [10]. Automatic feature extraction methods will reinforce the domain-independence nature of the case-based planning system.

The target function that was learned in the training phase is the time performance of the default planner. Other useful knowledge can also be learned and predicted, such as the quality of plans produced, the likelihood of failure, the likelihood of facing specific problems or avoiding other problems, etc. Furthermore, the effectiveness of learning and plan retrieval can be improved in various ways:

- Using more relevant and, if necessary, more complex features.
- Using learning algorithms with more powerful target function representations.
- Incorporating prior knowledge in the learning process i.e. combine inductive and analytical learning methods.
- Defining better *Gain* metrics.

The size of the plan library is a major concern, although from a complexity perspective, searching the library can still be done in polynomial time. But in real-world applications, the library size may have to be very large, such that the overhead of searching for a good case can potentially degrade the system performance instead of improving it. This is the *utility problem* and a variety of strategies have been suggested to address it [11].

The presented system has been designed without taking advantage of other planning strategies. In fact, other techniques can be integrated with the presented approach to create a more flexible and efficient planning agent. For instance, the plan library can be indexed in various ways such as goals they achieve or problems they avoid [8], and the search for a case can be based on a combination of criteria. Stored cases can also be generalized such that each stored plan can match a wider range of problems [12].

References

[1] R. Alterman, Adaptive planning, *Cognitive Science*, **12** (1988) 393-421.

[2] T. Bylander, Complexity results in planning, in: *Proceedings IJCAI-91*, Sydney, Australia (Morgan Kaufmann, San Mateo, CA, 1991) 274-279.

[3] D. Chapman, Planning for conjunctive goals, *Artificial Intelligence*, **32** (1987) 333-377.

[4] K. Erol, D.S. Nau and V.S. Subrahmanian, On the complexity of domain-independent planning, in: *Proceedings AAAI-92*, San Jose, CA (MIT Press, Cambridge, MA, 1992) 381-386.

[5] U. M. Fayyad, S. G. Djorgovski and N. Weir, From digitized images to on-line catalogs: Data mining a sky survey, *AI Magazine*, Vol. 17, Issue 2 (1996) 51-66.

[6] R.E. Fikes, P.E. Hart and N.J. Nilsson, Learning and executing generalized robot plans, *Artificial Intelligence*, **3** (1972) 251-288.

[7] R.E. Fikes and N.J. Nilsson, STRIPS: a new approach to the application of theorem proving to problem solving, *Artificial Intelligence*, **2** (1971) 189-208.

[8] K.J. Hammond, *Case-Based Planning: Viewing Planning as a Memory Task* (Academic Press, San Diego, CA, 1989).

[9] S. Kambhampati and H.A. Hendler, A validation-structure-based theory of plan modification and reuse, *Artificial Intelligence*, **55** (1992) 193-258.

[10] H. Liu and H. Motoda, *Feature Extraction, Construction and Selection: A Data Mining Perspective* (Kluwer Academic Publishers, Boston, MA, 1998).

[11] S. Minton, *Learning Search Control Knowledge: An Explanation-Based Approach* (Kluwer Academic Publishers, Boston, MA, 1988).

[12] S. Minton, Selectively generalizing plans for problem-solving, in: *Proceedings IJCAI-85*, Los Altos, CA (Morgan Kaufmann, San Mateo, CA, 1985) 596-599.

[13] B. Nebel and J. Koehler, Plan reuse versus plan generation: a theoretical and empirical analysis, *Artificial Intelligence*, **76** (1995) 427-454.

[14] P. Stone, M. Valeso and J. Blythe, The need for different domain-independent heuristics, in: *Proceedings of the Second International Conference on AI Planning Systems* (Morgan Kaufmann, San Mateo, CA, 1994) 164-169.