Combination of Problem Solving and Learning from Experience

Extended Abstract

Linda Briesemeister, Barbara van Schewick and Tobias Scheffer
Technische Universität Berlin

1 Introduction

Most AI problem-solving systems, when presented with the same problem repeatedly, always solve it the same way and in about the same amount of time. It seems shortsighted that they do not adjust their behavior on the basis of their experience. Therefore, learning by observing the process of solving a problem can lead to some form of knowledge, which can be used when new problems arise.

In recent years, attempts to improve the efficiency of a problem solver, especially to improve the speed with which it can solve problems (*speedup learning*), have been made by investigating mainly two techniques:

One technique is to learn new *macro operators* by composing sequences of original operators. The macro operators are added to the set of operators considered by the problem solver, and they allow it to take "big steps" in the search space (e. g. an extension of Strips [FHN72]).

The other technique is to learn some form of control knowledge, which helps to determine which control action to try next. Ideally the former "blind" search with exponential complexity is replaced by a strategy to move straight forward in direction of the goal. The control knowledge can take many forms including *evaluation functions, operator strengths*, and *operator-selection and rejection rules* (e. g. Lex [MUB83], Soar [LRN86], Prodigy [Mi90]).

This article also outlines an approach to speed up problem solving by learning from previous results; but instead of modifying the problem solver itself, the approach aims to replace it as far as possible. Our work was inspired by [MW95] who combine classification learning and problem solving methods in the same way. While their system was designed to solve a special problem, the system proposed in this article can be applied to various state-space problems. Additionally, the acquired knowledge can be extended, if it turns out to be insufficient.

2 Theory

The suggested system is used in two phases with different entry points (see figure 1 and 2). In the preparatory phase, a problem solver is applied to several states in the problem space. Thus, a control action is assigned to each state which is part of a solution path. This action is optimal if the problem solver delivers the optimal path to the goal state. An incremental learner then processes these pairs, induces classification rules which link states and control actions and stores them in a knowledge structure.

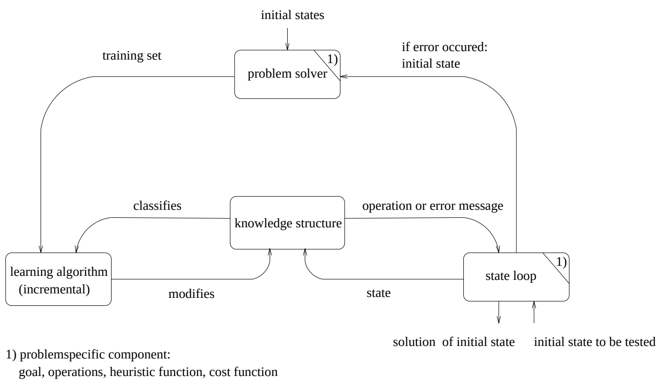
initial states
training set
problem solver
1)
if error occured:
initial state
classifies
knowledge structure
operation or error message
learning algorithm
(incremental)
modifies
state
state loop
1)
solution of initial state
initial state to be tested
1) problemspecific component:
goal, operations, heuristic function, cost function

Figure 1: System architecture

In the application phase, the system tries to solve problems by using this structure first. Starting with the initial state, the state loop applies the knowledge structure recursively to determine the right control action until the goal is reached. If the actual state cannot be classified or a cycle appears, the whole task is passed on to the problem solver. The problem solver then produces a solution, which, as in the preparatory phase, is also processed by the learner. Thus, the knowledge structure is improved continuously.

In an infinite state space, it is only semi-decidable whether the working state loop will produce a solution

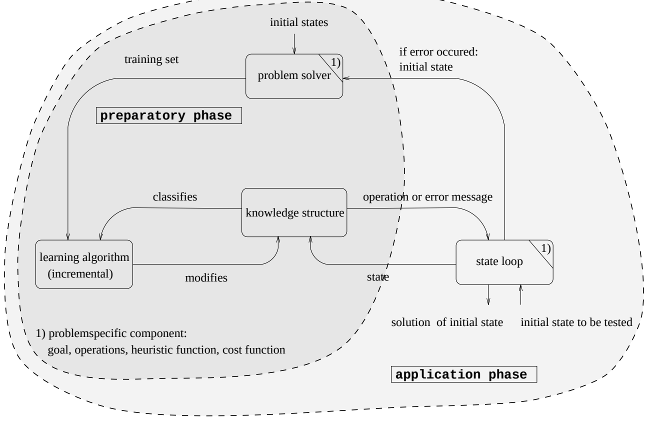

Figure 2: Preparatory and application phase

in the end. Therefore, if the length of a possible solution exceeds an arbitrary but fixed boundary, the problem solver is started as well.

Thus, the system always finds a solution, provided a solution exists and the problem solver is complete: If the solution is implied by the knowledge structure and the length of the solution path is less or equal the boundary, it is produced by the state loop; otherwise, it is computed by the problem solver.

The system does not necessarily produce the optimal solution to a problem, even if the problem solver does. As soon as generalized knowledge is used, the assigned control action may not be optimal, as the process of inducing classification rules does not maintain this property.

The suggested approach tries to reduce the computational complexity of problem solving by avoiding problem-space search and using learned knowledge instead. Therefore, its effectiveness depends on the quality of the knowledge structure: The more solutions it implies, the more time and space is saved. The best results are reached, if the problem space contains areas of states, which can be linked to the same control action. A suitably biased learning algorithm will then be able to induce generalized classification rules that can classify most other states correctly when given only a small training set.

If the problem is not structured at all, the induced rules will only be consistent with the training examples. In this case, the system only memorizes the training set; but since it can solve all states which were visited as part of a previous solution path by using the knowledge structure alone, the computational effort is still reduced.

As PAC learning theory [Va84] shows, the predictive accuracy of learning at a given level of confidence can also be improved by increasing the number of training examples. Therefore, the quality of the knowledge structure is also influenced by the size of the training set.

3 The implemented system

The implemented system is a general-purpose architecture, which accesses the problem specific information via interface. It has been implemented in LISP.

As problem solver, the A* algorithm [Ni71] is used. This algorithm finds an optimal solution, if a solution exists.

As incremental learner, the Cal2 learning algorithm [UW81], pp. 25 has been chosen. It induces classification rules which are expressed as a decision tree. As it operates on discrete values, our system can only solve problems with a discrete problem space. Cal2 requires disjoint classes. If there are several optimal paths to the goal, one state can be classified in different ways; but as each action is optimal, it is possible to delete all but one before learning. Thus, the training set will always contain disjoint classes.

4 Experiments

The system has been applied to different problems with a finite, discrete and static state space.

The efficiency of the system during the application phase depends on the learner, the problem itself and the number of initial states solved during the preparatory phase. As the learner is fixed, the experiments focussed on the influence of the last two factors.

To test the influence of the size of the initial state set, it was enlarged successively. For each size, the preparatory phase was executed. The resulting decision tree was then applied to a fixed test set of initial states, simulating the application phase except from restarting the problem solver. The proportion of states solved successfully indicates the efficiency of the system, because for them, the problem solver does not have to be started again. To get reliable results, all state sets used during the experiments were representative selections from the problem space.

To determine the influence of the problem itself, these experiments were conducted using variations of the problem.

4.1 Rubik's Cube

The famous Rubik's Cube was developed by Ernö Rubik in 1975. Twisting the slices results in a disordered state. Finding a way back to the goal state (without using the inverse moves) is a task too complex to be solved just by trial and error: There are 43,252,003,274,489,856,000 different states and only one of them encodes the goal.

A state in the problem space is defined as a vector of the length 54 (6 surfaces $\cdot$ 9 small faces on each). Each position equals a small surface and can hold one of six colors. In total there are 9 layers, which can be moved (3 slices $\cdot$ 3 dimensions). As each layer can be turned clockwise or anticlockwise, we deal with 18 different operations.

Figure 3: Goal state of Rubik's Cube

We varied the problem by changing the *problem depth (PD)*. PD denotes the number of randomly chosen turns that are applied to the goal state to create an initial state. This state can then be solved by the problem solver in PD or less steps. We have tested problem depths from 1 to 5 moves, because the problem exceeds computational resources for greater PD.

The results of the five experiments are shown in figure 4 and in table 1. The figure displays the percentage of solved initial states from the test set as a function of the amount of initial states being used in the preparatory phase. For all PD the curves lie above the identity function, which is plotted additionally. Growing very rapidly in the beginning, at some point the growth is slowing down. The table summarizes the results for the largest set size of each PD.

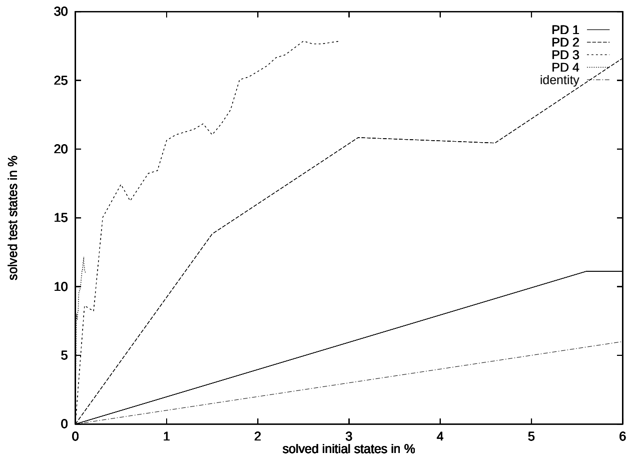

Figure 4: Results of Rubik's Cube

For all set sizes, more initial states were successfully processed in the state loop than had been solved in the preparatory phase. This shows, that the learner has not only memorized the given solutions but generalized them. As the coherence between greater PD and greater ascent of the function indicates, the extension of the problem space seems to lead to better generalization. Nevertheless, the available data is not sufficient to determine whether this is a general tendency or not.

PD	quantity of problem space	solved initial states	solved test states
1	18	100%	100%
2	262	38.2%	56.9%
3	3502	2.9%	27.9%
4	46317	0.11%	11.0%
5	>46317	<0.11%	8.4%

Table 1: Results of Rubik's Cube

4.2 Toad in a maze

A toad is placed into a given two-dimensional maze with only one exit. The toad has to find the exit from different places. After solving these problems the toad should find the way even from unknown positions using the obtained knowledge.

The two-dimensional state space is given by the points of the maze. The walls are represented as forbidden regions in the state space. The toad can perform one of four possible operations by moving "up", "down", "right" or "left".

For the final experiments, mazes of 30 x 30 points were used. An example is shown in figure 5. The initial state set was enlarged successively until it comprised the whole state space. Due to the manageable size of the problem space, it was also possible to use the whole state space as the test set. The system has been applied to five mazes with growing complexity. Starting with a simple maze, increasingly complex mazes were generated by adding new walls successively.

```
##############################
#                            #
#   #######   #    ######    #
#         #   #              #
#      #      ########       #
#      #                     #
#      #   ######        #####
#      #   #                 #
#   ####   #   #######       #
#          #         #       #
#   ########         #   #   #
#   #          #######   #   #
#   #                    #   #
#   #  #########   #######   #
#   #          #             #
#              #      ########
#    ######    #             #
#              ##########    #
#      ####                  #
#         #     #######      #
#    #    #     #        #   #
#    #    #     #     ####   #
#               #            #
#    ########   #     ########
#    #          #            #
#    #    #######        #   #
#               #     #  #   #
#         #           #      #
          #           #      #
##############################
```

Figure 5: Maze4

The results of the experiments are plotted in figure 6. Maze1 is the simplest, maze5 the most complex maze. The figure shows the percentage of correct classifications of the state space as a function of the relative size of the initial state set during the preparatory phase. All functions are significantly greater than the identity function. For all mazes, the system rapidly reaches a large percentage of correct performance, but then improves very slowly. As the problems become more complex, the speed of growth decreases: The more complex the maze, the lower the curve.

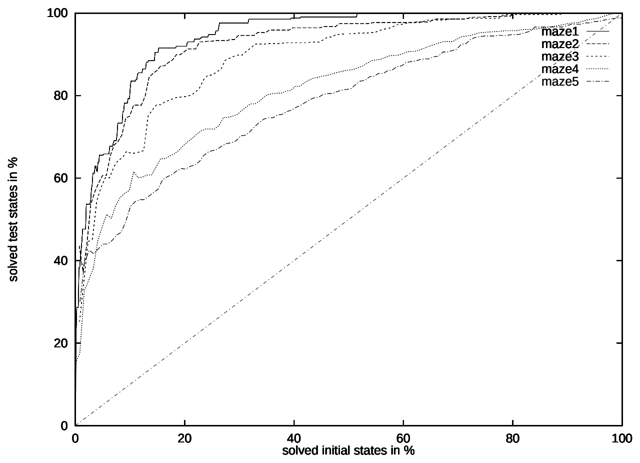

Figure 6: Results of maze problem

As the huge differences between the curves and the identity function reveal, a large amount of knowledge is acquired by generalization. If a large initial state set is processed in the preparatory phase, the resulting knowledge structure will be able to produce a lot of additional solutions due to generalization. When this initial state set is enlarged, the new training set for the learner in the preparatory phase consists of the old training set plus the solutions of the newly added initial states. As some of the new solutions were already implied by the old knowledge structure, their addition does not increase the number of solutions provided by the new knowledge structure. As a result, the ascent of the curves slows down.

Therefore, a lot of problem solving effort during the preparatory phase can be wasted, if the set of initial states is too large. It is more efficient to choose a restricted number of initial states so that a large amount of knowledge is gained through generalization. During the application phase, the few states still misclassified by the knowledge structure are solved by the problem solver. Thus, the problem solving effort is only spent when the missing knowledge is actually needed.

As the results show, the amount of generalized knowledge induced from a given set of initial states depends on the complexity and the structure of the problem. The more complex the problem, the more difficult it is to infer valid generalization from the information provided by the same percentage of the state space. Therefore, the best size of the initial state set has to be determined individually for each variation of a problem.

5 Conclusion and further research

The suggested approach combines problem solving and learning to reduce the computational complexity of problem solving. We have tested the system with two different problems: Rubik's Cube and Toad in a Maze. The resulting curves exemplify the generalization ability of both problems. The gained knowledge induces rules for new tasks and therefore, the effort for solving new problems can be reduced. The experiments show that the amount of savings depends on the problem as well as on the learning algorithm.

The possibility of restarting the problem solver during the application phase guarantees a solution for any state, provided the computational resources are not exceeded. If learning can still be afforded while using the system in the application phase, the incremental learner will save the more time the longer the system runs. Otherwise, the whole problem space has to be solved in the preparatory phase, which requires a finite state space. In real-time systems, the response time for a control action then equals the time for traversing the decision tree.

For future research, connecting the problem solver itself with the knowledge structure seems to model human behavior in a better way. It might also be interesting to implement other learning algorithms that are biased differently. At the moment, the description of possible states is restricted to n-dimensional vectors. To admit more general representations like structured or logical state descriptions could widen the range of feasible applications as well.

References

[FHN72] R. E. Fikes, P. E. Hart and N. J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, Volume 3, pp 251-288, 1972.

[LRN86] J. E. Laird, P. S. Rosenbloom and A. Newell. Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning*, Volume 1, pp 11-46, 1986.

[Mi90] S. N. Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, Volume 42, pp 363-392, 1990.

[MUB83] T. M. Mitchell, P. E. Utgoff and R. B. Banerji. Learning by experimentation: Acquiring and refining problem-solving heuristics. 1903. In: Editors, R. S. Michalski, J. G. Carbonell and T. M. Mitchell. Machine learning: An artificial intelligence approach (Vol. I). San Mateo, California: Morgan Kaufmann.

[MW95] W. Müller and F. Wysotzki. Automatic Synthesis of Control Programs by Combination of Learning and Problem Solving Methods. *Proceedings of the*

European Conference on Machine Learning (ECML - 95), pp 323-326, 1995.

[Ni71] N. Nilsson. *Problem Solving in Artificial Intelligence*. McGraw Hill, 1971.

[UW81] S. Unger und F. Wysotzki. *Lernfähige Klassifizierungssysteme*. Akademie Verlag, Berlin 1981.

[Va84] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, Volume 27, pp 1134-1142.