Learning Effective Search Heuristics[1]

Pat Langley
The Robotics Institute
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213 USA

ABSTRACT

SAGE.2 is a production system that improves its search strategies with practice. The program incorporates four different heuristics for assigning credit and blame, and employs a discrimination process to direct its search through the space of move-proposing rules. The system has shown its generality by learning search heuristics in five different task domains. In addition to improving its search behavior on practice problems, SAGE.2 was able to transfer its expertise to scaled-up versions of a task, and in one case transferred its acquired search strategy to problems with different initial and goal states.

INTRODUCTION

The ability to search is central to intelligence, and the ability to *direct* search down profitable paths distinguishes the expert from the novice. Since all experts begin as novices, the transition from one to the other should hold great interest for Artificial Intelligence. In this paper we examine SAGE.2, a program that implements one approach to learning effective search heuristics. Below we present an overview of this system and summarize its operation in various domains. However, before turning to these matters, let us review the nature of the strategy learning task by discussing its components.

Within any system that improves its strategies as a function of experience, we can identify three distinct components. First, such a system must be able to *search*, so that it can generate behaviors upon which to base its learning. Second, a strategy learning system must be able to *assign credit and blame* to components of the system responsible for good and bad behaviors. One approach [1, 2] involves finding a solution to a problem, and then using the solution path to distinguish good moves from bad moves. An alternative approach, explored by Anzai [3] and Ohlsson [4], involves assigning credit while the search is in progress. Finally, a strategy learning program must be able to use credit information to *modify* its performance component so that behavior improves over time. Such modification often involves determining the conditions under which various operators should be applied. Given good and bad examples of these operators, heuristically useful conditions can be determined in three possible ways: through a process of *generalization*; through Mitchell's [1] *version space* method; or, as we shall see shortly, through a process of *discrimination*. Now that we have reviewed the components involved in learning search heuristics, let us examine how these components are implemented in the present system.

[1]This research was supported by Contract N00014-83-K-0074 from the Office of Naval Research. I would like to thank Stephanie Sage, Steve Smith, and John Laird for comments on an earlier version of this paper.

AN OVERVIEW OF SAGE.2

Like most other strategy learning programs, SAGE.2 is stated as a production system. In other words, it is cast as a set of relatively independent condition-action rules or productions, and learning occurs through the addition of new productions. Below we discuss the system's search process, its credit assignment heuristics, and its mechanisms for altering its search strategy in the light of experience.

The Search Process

SAGE.2 represents states as elements in working memory and operators as productions that match against these states. Each production has an associated *weight*. Move-proposing rules with the same weight can be applied in parallel, so more than one move may be proposed at a time. Initially, these rules contain only the legal conditions for applying the associated operators. Thus, the system carries out a breadth-first search until credit can be assigned and improved move-proposing rules can be learned to direct the search process.

When a new rule is first created, it is assigned a low weight. Since a rule's weight is increased every time it is relearned, this number can be viewed as a measure of each rule's *success*, with preference being given to more successful rules. Search becomes much more selective as SAGE.2 begins to prefer productions that have been learned many times, and to shun those that have led to errors in the past. However, it retains the ability to consider multiple paths, as long as those paths are proposed by rules with the same weights.

Assigning Credit and Blame

SAGE.2 can operate in two modes: it can assign credit based only on a complete solution path, or it can do so during the search process. In the first method, exhaustive breadth-first search continues until one or more solution paths are found. The system then marks all moves lying on these paths as good instances of the rules that proposed them, and labels all moves leading off the paths as bad instances of the responsible rules.

The second method does not require knowledge of complete solution paths, relying instead on several rules for recognizing undesirable moves. One of these notes when the system reaches some state that was visited earlier; this rule detects loops as well as unnecessarily long paths. Another applies when a state is found from which no moves can be made (a dead-end). These rules are relatively domain-independent, but less general rules can be employed as well. For instance, one can write problem-specific rules for recognizing when an illegal state has been reached. When any of these rules labels a move as undesirable, SAGE checks to see if it has made any move from the same state that is not labeled as bad. If such a move exists, and if both moves were proposed by the same rule, SAGE passes them to the learning component as good and bad instances of that rule; if not, then the learning component cannot be applied.

Learning Conditions Through Discrimination

As good and bad instances of the move-proposing rules are identified, they are passed to SAGE's discrimination process. This mechanism searches for differences between what Bundy and Silver [5] have called the *selection context* (the state of memory during the good application) and the *rejection context* (the state during the bad application). Differences take the form of sets containing one or more working memory elements that were present in one context but not the other. For each difference that it finds, SAGE creates variants of the overly general rule by including these differences (with certain terms replaced by variables) as one or more extra positive or negated conditions on that rule. Thus, the system begins with overly general rules, and gradually learns more specific versions with heuristically useful conditions that direct search down the desired paths. Many spurious variants are created along with useful ones, but since useful rules are relearned more often, their weights are increased and they come to be preferred.

The discrimination method has two important advantages over other methods. Generalization-based systems (including the version space technique) find conditions that are held in common by all positive instances, and so are biased toward learning *conjunctive* rules. The approach can be extended to handle disjuncts, but this involves expensive backtracking through the rule space. In contrast, the discrimination method compares a *single* good instance to a *single* bad instance, enabling it to discover disjunctive rules as easily as conjunctive ones. In addition, the discrimination method employs the weighting method to direct search through the space of rules. Thus, SAGE learns more slowly than generalization-based systems, since it must gather statistics on the usefulness of competing variants. However, this approach makes the system quite robust with respect to noise, so that if an occasional error in made during credit assignment, the system will still be able to learn useful rules.

EXAMPLES OF SAGE.2 AT WORK

Our overview of SAGE.2 is now complete, but to gain a better understanding of how the system learns search strategies, we must examine its workings in specific domains. Below we discuss the program's learning sequence on the Tower of Hanoi puzzle, both when it uses only complete solution paths to assign credit and when it learns during the search process. We also summarize the program's behavior in four other task domains.

SAGE.2 on the Tower of Hanoi Puzzle

In the Tower of Hanoi puzzle, N disks of decreasing size are placed on one of three pegs. The goal is to move all of them to one of the other two pegs. Disks are moved one at a time, with two constraints. First, only the smallest disk on a peg can be moved. Second, a disk cannot be moved to a peg on which a smaller one resides. These constraints limit the set of legal moves, so that for the three-disk puzzle, only 27 states are possible. However, the number of connections between these states is high, making for a challenging problem.

Using the complete solution path heuristic for credit assignment, SAGE learns only after a solution is reached. On its first pass through the problem space, SAGE.2 made 56 moves in a breadth-first search before it found a solution. At this point, the system examined the two (symmetrical) solution paths, assigning credit and applying the discrimination mechanism. Through this process, five variants on the original operator (let us call it MOVE) were created. One variant, MOVE-1, was built each time the system looped back to a previous state in the search tree. This rule contained a condition that prevented it from undoing a move that the system had just made. Other variants with useful conditions were learned from other errors. However, MOVE-1 emerged as the strongest contender, since the looping error that led to its creation was more frequent than other errors.

On the second run, the system's performance improved considerably, since MOVE-1's weight had come to exceed that of MOVE. In this case, 38 moves were made before the solutions were reached. MOVE-1 prevented backing up, but other errors still occurred, leading to the construction of three variants on this rule. Of these, MOVE-2, which included a condition to prevent moving the same disk twice in a row, was learned most often. Thus, on the third run, the system neither made looping errors, nor moved the same disk twice in a row. The few remaining errors led to the creation of MOVE-3, which prevented moving a disk back to the position it had occupied two moves earlier. This heuristic was sufficient to eliminate search on the Tower of Hanoi task, and when the problem was presented a fourth time, SAGE reached the solutions without taking any false steps. After mastering this simple version of the problem, the system was able to solve the scaled-up four-disk task as well. However, SAGE.2 is not at present able to transfer its learning to versions of Tower of Hanoi with different initial and goal states.

In learning while doing, SAGE followed a very similar learning sequence. During the initial breadth-first search, the revisited state heuristic noted a number of loops in the search tree, and labeled the responsible moves as undesirable. As before, these led to the creation of the variant MOVE-1, which proposes only non-looping moves. In later runs, the variants MOVE-2 (based on unnecessarily long paths) and MOVE-3 (based on dead-ends) were created, with the latter eventually eliminating search.

Applying SAGE.2 to Other Domains

One important dimension on which AI systems are judged is their generality, and the most obvious test of a program's generality is to apply it to a number of different domains. Accordingly, we have tested SAGE in four additional task domains, which we discuss below.

In the Slide-Jump puzzle, one is presented with N quarters and N nickels placed in a row and separated by a blank space. Legal moves include *sliding* into a blank space or *jumping* over one other coin into a blank space, while the goal is to exchange the positions of the quarters and nickels. SAGE.2 was given two initial move-proposing rules – one for suggesting slide moves and the other for suggesting jumps. The system learned a search heuristic that proposed sliding a coin into a blank space only if another coin of the same type had just been jumped from that space. In addition, this rule never exceeded the original jump rule in weight, so jumps were made whenever possible. In the learning while doing runs, the system proceeded in a very similar manner, learning from both revisited states and dead ends.

Ohlsson [4] has described the Tiles and Squares puzzle, which involves N numbered tiles and N + 1 squares on which they are placed. Only one legal move is possible: moving a tile from its current position to the blank square. The goal is simple: get all of the tiles from their initial positions to some explicitly specified goal positions. On this task, SAGE.2 acquired two simple heuristics for directing search. The first of these states that if possible, one should move a tile into its goal position; the

other states that one should never move a tile *out* of its goal position once it has been placed there. Note that these rules are *disjunctive*; neither heuristic is sufficient to completely direct the search process by itself, but taken together they eliminate search. Thus, the ability of the discrimination process to consider disjunctive heuristics shows its potential in this puzzle. Another interesting characteristic of this problem is that SAGE.2 incorporated information about the goal state in the conditions it discovered. As a result, the heuristics the system learned could be applied not only to more complex problems, but to problems with differing initial and goal states. In learning while doing on this task, SAGE learned from both loops and unnecessarily long paths.

In the Mattress Factory puzzle, the goal is to fire all N employees at a factory, using two operators. The least senior worker may be hired or fired at any time. However, other workers may only be hired or fired if the person directly below them in seniority is currently employed, and provided that no other person below them is also employed. SAGE.2 was given rules for proposing both types of moves. After finding the single solution path, it arrived at variants of both rules that avoided simple loops. In addition, the variant of the first rule achieved a greater weight, so that it was preferred. In learning while doing, the system learned mainly from loops in its search tree, though one dead end also occurred.

In Piaget's length seriation task, the child is presented with a set of blocks in a pile, and is asked to line them up in order of ascending height. For this problem, SAGE.2 was given a single operator for moving a block from the pile to the end of the current line. Also, the program was given a domain-specific rule for determining illegal states. This stated that if a taller block had been set to the right of a shorter block, the resulting state was undesirable. In learning from complete solution paths, the system generated one useful heuristic, which proposed moving a block only if there was no other block in the pile that was taller than that piece. In learning while doing, both the rule for noting illegal states and the dead-end heuristic came into play, generating the same variant move-proposing rule.

Generality of the Heuristics

As we have seen, SAGE.2 has learned rules for directing search in five different domains, suggesting that the system is a relatively general one. However, each of the learning heuristics can be examined on this dimension as well. Since the discrimination strategy played a central role in each of the runs described above, we can safely infer the generality of this method. However, the situation with respect to the credit assignment heuristics is somewhat more complex. The complete solution path heuristic is very general, and was applied on each of the tasks. The other heuristics were less useful, but still showed evidence of generality. Both the revisited state rule and the dead-end rule led to learning in four of the five domains. The illegal state detector was stated in a domain-specific manner and was used only in the seriation task. However, one can imagine versions of the Tower of Hanoi, Mattress Factory, and Slide-Jump puzzles in which the conditions for legal moves must be learned along with the conditions for good moves. Thus, we can conclude that SAGE's various learning heuristics are general ones, and should prove useful in domains other than those examined here.

CONCLUSIONS

Before closing, let us consider SAGE.2's relation to other systems, and whether it has advanced our understanding of the strategy acquisition process. The system shares Anzai's [3] revisited state heuristic for assigning credit, but it is considerably more general than the earlier system, which was tested in a single domain. Like Brazdil's ELM [6] and Mitchell, Utgoff, Nudel, and Banerji's LEX [1], our system can also assign credit based on complete solution paths. However, ELM and LEX had access to *only* this heuristic, while SAGE.2 incorporates a number of interacting credit assignment methods. Also, these systems focused on transfer between problems of similar complexity, while SAGE is also able to transfer to scaled-up problems. The discrimination technique used by SAGE bears a strong resemblance to ELM's learning method, but SAGE has shown how this approach can be applied to learn disjunctive heuristics. In summary, SAGE has addressed a number of issues that have been overlooked in the earlier work.

Although progress has been made, our understanding of the strategy learning process is far from complete. For example, SAGE.2 was not in general able to transfer its acquired expertise to problems with different initial and goal states from those on which it practiced. However, the system was able to make such a transfer on the Tiles and Squares puzzle by explicitly representing the conditions for goal satisfaction in working memory. Hopefully, by augmenting SAGE's representation for other tasks, it will be able to make similar transfers. A second extension should enable the system to learn in much more complex domains such as chess. If SAGE.2's search control were altered to allow the setting of subgoals, then the heuristic for assigning credit based on complete solution paths could be applied whenever a particular subgoal had been achieved. Variants learned from this path would be specific to that subgoal; that is, they would include a description of the current subgoal as an extra condition, in addition to the other conditions found through discrimination. Thus, while more research remains to be done, the results we have obtained so far are encouraging, indicating that effective search strategies can indeed be learned using simple and general mechanisms.

REFERENCES

[1] Mitchell, T. M., Utgoff, P., Nudel, B., and Banerji, R. B. Learning problem solving heuristics through practice. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1981, 127-134.

[2] Langley, P. Strategy acquisition governed by experimentation. *Proceedings of the European Conference on Artificial Intelligence*, 1982, 171-176.

[3] Anzai, Y. Learning strategies by computer. *Proceedings of the Second National Conference of the Canadian Society for Computational Studies of Intelligence*, 1978, 181-190.

[4] Ohlsson, S. On the automated learning of problem solving rules. *Proceedings of the Sixth European Meeting on Cybernetics and Systems Research*, 1982.

[5] Bundy, A. and Silver, B. A critical survey of rule learning programs. *Proceedings of the European Conference on Artificial Intelligence*, 1982, 151-157.

[6] Brazdil, P. Experimental learning model. *Proceedings of the Third AISB/GI Conference*, 1978, 46-50.