# Selectively Generalizing Plans for Problem-Solving

Steven Minton[1]

Computer Science Department, Carnegie-Mellon University
Pittsburgh, PA 15213, USA

## Abstract

Problem solving programs that generalize and save plans in order to improve their subsequent performance inevitably face the danger of being overwhelmed by an ever-increasing number of stored plans. To cope with this problem, methods must be developed for selectively learning only the most valuable aspects of a new plan. This paper describes MORRIS, a heuristic problem solver that measures the utility of plan fragments to determine whether they are worth learning. MORRIS generalizes and saves plan fragments if they are frequently used, or if they are helpful in solving difficult subproblems. Experiments are described comparing the performance of MORRIS to a less selective learning system.

## 1 Introduction

Building problem-solving programs that improve their performance by generalizing and re-using past solutions is one of the goals of machine-learning research. It has been demonstrated that generalized solution sequences, or plans, can be produced by analyzing the constraints inherent in solution instances [3,8]. This method of learning has been successfully employed in domains as diverse as game-playing [6] and mathematical problem-solving [7].

One drawback to learning plans is that the number of stored plans may increase quickly as the problem-solver gains experience. Furthermore, the applicability conditions of long plans tend to be highly specific. Searching through the space of stored plans to find one that is best-suited to the current problem may be as expensive as searching through the original search space. This leads to the following paradox: as the system gains experience it gradually becomes swamped by the knowledge it has acquired. In some cases performance can eventually degrade so dramatically that the system operates even more poorly than a non-learning system.

One of the earliest and best-known plan learning systems was the STRIPS problem solver [4,3]. Having solved a problem, STRIPS produced a parameterized version of the solution, called a Macrop, by generalizing constants while maintaining the dependencies among the steps in the solution. The Macrop might subsequently be used-either in whole or in part-to aid in rapidly solving similar problems. This paper explores how the degradation problem manifests itself in STRIPS-like learning systems. Moreover, two methods are considered whereby a heuristic problem-solver may selectively save fragments of generalized solutions in order to stave off degradation.

## 2 STRIPS and MACROPS

The generic* term "macro-operator" will be used hereafter to refer to a parameterized sequence of operator applications, eg.:

```
GOTO-OBJ(KEYS)
PICK-UP(KEYS)
GOTO-DOOR(drx)
UNLOCK(drx)
OPEN(drx)
GOTHRU-DOOR(drx, rmy)
```

Constants are denoted by capitalized strings, and variables by lower case strings. The macro-operator shown above describes a series of actions for getting the keys and unlocking, opening, and going through a door.[2]

A STRIPS problem-space consists of a world model, represented by a set of well-formed formulas (wffs) in the predicate calculus, and a set of operators. Each operator includes an add-list, a delete-list and a precondition wff. An operator is applicable if its precondition wff is satisfied. Applying an operator simply involves making the changes indicated in the add and delete lists.

STRIPS, like many other problem-solvers, searched through the space of operator sequences in order to solve a problem. Means-ends analysis [1] was used to guide the search. Once a sequence of operators that solved the problem was found, STRIPS produced a Macrop by replacing the problem-specific constants in the operator sequence with problem-independent parameters. Any subsequence of the Macrop could then be used as a composite operator during future planning.

Fikes et al. [3] described a series of 5 problems that STRIPS solved more rapidly when Macrops were learned after each trial. They claimed that "the search tree sizes [were] all smaller when Macrops were used and the Macrops allow longer plans to be formed without necessarily incurring an exponential increase in planning time".

The author is supported by an AT&T Belt Laboratories Scholarship

[2]STRIPS's Macrops also included information helpful for monitoring execution of the operator sequence in the real world.

There appear to be two distinct factors that can contribute to the effectiveness of macro-operators in problem-solving. First, since macro-operators represent sequences of operators, the preconditions and postconditions of the individual operators can be compiled into aggregate preconditions and postconditions for the macro-operator as a whole. Therefore it can be quicker to test whether a macro-operator is applicable than to test whether the corresponding sequence of operators is applicable.[3] . Even more importantly, the use of macro-operators can bias the order in which the search space (of operator sequences) is explored. If relevant Macro-operators tend to be considered before relevant operators, then previously successful paths will generally be explored before other paths. This experiential bias can be a significant source of heuristic power.

Unfortunately, a problem solver that uses macro-, operators in this manner may find that as the number of macro-operators increases, the experiential bias gradually disappears. In the extreme case, if eventually every operator sequence that the problem-solver might conceivably consider in solving a problem becomes a macro-operator, then the ordering advantage will have been effectively negated.

In practice, however, only a subset of these sequences become macro-operators. For any given domain, the crucial issue is whether or not the use of macro-operators will effectively compress the search space associated with that domain. If a small set of macro-operators can be generated that "cover" the relevant problems in the domain, then search will be confined within this smaller space. For example, Korf's Macro Problem Solver [5] is powerful enough to generate a set of macros that completely eliminates search, unfortunately, his technique only works for domains that exhibit operator decomposability. With STRIPS, every unique subsequence of all previously acquired solutions Is a potential macro-operator; the STRIPS technique for generating macro-operators does not have strong domain requirements, but neither does it guarantee that the search space will be adequately compressed. Indeed, it has been our experience that even in small domains, the STRIPS approach can quickly lead to an explosion of macro-operators. For example, in STRIPS even "useless" operator sequences, such as STACK(x, y) followed by UNSTACK(x, y), can become macro-operators due to STRIPS's methods of generating and editing solution sequences.

## 3 MORRIS: A Selective Learner

To avoid being swamped by too many macro-operators, a problem-solver can endeavor to retain only those macro-operators that are most useful. MORRIS ("the finicky learner") is a heuristic problem-solver in the STRIPS tradition[4] that demonstrates the importance of selective learning. Currently MORRIS saves two types of macro-operators, s-macros and t-macros. S-macros, or "scripts", are frequently used operator sequences. T-macros, or "tricks", are operator sequences for solving difficult problems.

### 3.1 S-Macros

The strategy of retaining only the most frequently used macro-operators was suggested by Fikes et. al. [3], but never implemented in STRIPS. MORRIS accomplishes this by maintaining a record of the problems solved and their generalized solutions. Each time a new solution is acquired, it is compared to the previously acquired solutions in order to locate common subsequences. When two unifiable subsequences are found, the more general of the two subsequences is added to the list of s-macros.

A limit is maintained on the number of s-macros kept active by the system. Once this limit is exceeded, the s-macros that were least-used during their lifetimes are deleted from the active set. The net result of this process is a set of relatively short macro-operators, which is desirable, since the time cost of evaluating whether a macro-operator is applicable can grow exponentially with the number of preconditions it has[5].

### 3.2 T-Macros

T-macros are macro-operators that represent "non-obvious" solutions to difficult problems. The notion of non-obviousness is defined by MORRIS'S heuristic evaluation function, called $H_{diff}$. $H_{diff}$ is used to estimate the progress that an operator (or macro-operator) makes with respect to the current set of goals. At each node in the search tree, MORRIS collects the relevant set of operators (and macro-operators) for extending the current path and evaluates them with respect to $H_{djff}$. Since MORRIS employs a best-first search through the tree, operators that appear to make the most progress are considered before less promising operators.

In evaluating the progress made by an operator, $H_{diff}$ takes into the account the number of goals that remain to be solved as well as the *criticality* of each of these goals. A criticality value is a difficulty estimate assigned to each literal (i.e. potential goal) in the domain [10]. Higher criticality goals are attacked before goals of lesser criticality. (Generally one literal is given a higher criticality value than another literal if achieving the first literal typically undoes the second literal.)

[3]Clever methods for storing macro-operators and ordenng f preconditions may also effect the efficiency of the search, although attention has been given to these issues

Although patterned after STRIPS, MORRIS operates within a closed world, and therefore uses matching rather than theorem proving to test whether the preconditions of an operator are satisfied.

[5]There is generally a linear relationship between the average number of preconditions and the length of a macro.

Occasionally the values given by $H_{diff}$ will be misleading; if the real solution path is estimated to be less promising than alternative paths, MORRIS may be led far astray in its search. For example, consider the situation depicted in Figure 1. Paths A and B appear to be productive, but eventually lead to dead ends. Path C, despite its unpromising rating at node $N_o$, actually leads to a solution. (The heuristic is of the hill-climbing variety; up Indicates apparent progress towards the solution.) In such cases, the mistaken estimate may be uncovered only after many alternatives have been explored, since MORRIS uses a best-first search to traverse the search space.

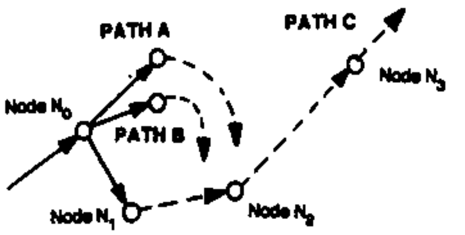

Figure 1: Misleading Path Ratings

The operator subsequence from node $N_o$ to node $N_3$ is *locally anomalous*. Its initial segment appears to make no progress, but the subsequence as a whole is rated as advantageous. Parameterizing and saving this 3-step operator sequence as a t-macro will enable MORRIS to avoid similar pitfalls in the future. If the goal at node $N_o$ is encountered again, MORRIS will evaluate its relevant operators as usual, but now the saved t-macro will be among them. Consequently, a more accurate heuristic estimate of this path will be generated. This strategy helps MORRIS avoid states which, in hill-climbing terms, are local maxima.

T-macros are also relevant to problems involving interacting goals. Many well-known problems fall into this class; for example, a robot planning problem might require the robot to move a box from a room and turn off the light to the room. If the robot first tries to turn off the light, it will not be able to move into the room to get the box. Because the two goals interact, ordering considerations are important.

If MORRIS finds that re-ordering goals succeeds In solving a problem that could not be solved otherwise, it must be the case that that some interaction between these goals occurred. The eventual solution to such a problem will always include a locally anomalous subsequence. This happens because re-ordering goats corresponds to attacking a lower criticality problem before a higher crtticality problem (usually an unproductive undertaking), and consequently a low $H_{diff}$ rating is generated at that point.

Once the solution is found, MORRIS identifies the goals involved in the re-ordering and constructs a t-macro in the normal fashion. T-macros of this type are particularly effective, since their use can be restricted to situations where the combination of these goals reoccurs.[6]

## 4 Experimental Results

In order to compare the effectiveness of MORRIS against a less-selective learner, a problem-solver called MAX was constructed that closely follows the STRIPS philosophy of saving all usable macro-operators. As does STRIPS, MAX generalizes the entire solution sequence whenever a problem is solved, and considers all composable subsequences to be potential macro-operators. MAX's procedures for saving, editing, and eliminating subsumed macro-operators are all modeled after those used by STRIPS with Macrops.

Once an operator (or macro-operator) is determined to be relevant to the current goal, both MAX and MORRIS use the same method to instantiate the operators. The bindings necessary to produce the relevant additions are first substituted into the operator's precondition list and then a partial-matching process is instituted to find potential instantiations. Since the time cost of the matching process is sensitive to the ordering of the preconditions, the matcher re-orders the preconditions to decrease the cost. Furthermore, during the matching operation, partial instantiations with many unsatisfied conditions may filtered out if the number of potential instantiations grows exceptionally high. Each instantiation produced by the matcher is then evaluated by $H_{djff}$.

MAX and MORRIS are identical programs with respect to their method of exploring the problem space. Since both programs employ a best-first search, misleading heuristic estimates can be costly. The only difference between them is in the types and numbers of macro-operators saved.

MAX and MORRIS were compared in a robot world consisting of 26 operators. This world is similar to the STRIPS experimental domain, but is richer in that many interacting problems can occur. There are 5 rooms, and operators for going to objects, going through doors, standing on, picking up, stacking, unstacking and putting down objects. Boxes can be pushed to various locations. Objects can be fixed and broken with various tools. Lights can be turned on and off, doors can be opened, closed, locked and unlocked. Food can be eaten. Sample problems include the following: "Go into room1 and lock the door to room1"; "Push boxA and boxB together and stand on BoxA"; "Take the keys from rooml to the room3".

The experimental results for a sampling of problems from a series of 25 are shown in Table 1. The table includes results for MORRIS, MAX, and a non-learning problem-solver that uses $H_{djff}$ but does not save any macro-operators. (The non-learning program is a stripped down version of MORRIS). The timing data does not include the time taken to generalize macro-operators, only the time necessary to find a solution. Typically, the learning time is considerably less than the search time.

[6]Presently, MORRIS does not attempt to fully analyse why the interaction occurred. In order insure that application of the t-macro is restricted to circumstances under which the interaction occurs, a slightly weaker form of generalization is used whereby identical constants in the goals are replaced by single variables.

| Problem Number | 1 | 10 | 17 | 22 | 26 |
|---|---|---|---|---|---|
| NON-LEARNING SYSTEM | | | | | |
| # Branches evaluated | 24 | 53 | 704 | 668 | 930 |
| # Nodes expanded | 10 | 22 | 137 | 110 | 186 |
| Solution length | 4 | 10 | 16 | 16 | * |
| CPU Seconds | 1.6 | 3.9 | 78.1 | 62.4 | 100 |
| MORRIS | | | | | |
| # Branches evaluated | 24 | 47 | 55 | 123 | 96 |
| # Nodes expanded | 10 | 20 | 13 | 24 | 19 |
| Solution length | 4 | 10 | 11 | 16 | 16 |
| CPU Seconds | 1.5 | 3.6 | 8.0 | 17.0 | 14.1 |
| MAX | | | | | |
| # Branches evaluated | 24 | 201 | 128 | 781 | 1131 |
| # Nodes expanded | 10 | 20 | 10 | 26 | 17 |
| Solution length | 4 | 10 | 12 | 16 | * |
| CPU Seconds | 1.6 | 13.0 | 16.0 | 68.0 | 100 |

* No solution generated within 100 CPU seconds

Table 1: Experimental Results

Since small variations in the problems can cause large differences in performance for each of these systems, Table 1 is only partially indicative of their relative abilities. However, some points do stand out.

The benefits attributable to MORRIS'S strategy of selectively saving macrops are revealed by the smaller number of branches • relevant operators and macro-operators • that were evaluated by MORRIS during each search as compared to MAX. Saving fewer macro-operators did not hurt MORRIS'S overall performance. Consider, for example, that in solving problem 10 MAX and MORRIS followed the same path to the solution, but MAX evaluated more alternatives along the way. The problem with MAX is that it gradually loses the efficiency advantage provided by $H_{diff}$. Whenever $H_{diff}$ indicates the correct branch, MAX will waste considerable time instantiating many macro-operators, in effect, performing look ahead. Whenever $H_{diff}$ is wrong, MORRIS will be as well prepared as MAX assuming the appropriate t-macro has been saved.

Compared to the non-learning problem-solver, MORRIS generally performed better. Admittedly, the sequence of problems was arranged so that the smaller problems were presented first. In many cases, the t-macros learned while solving these earlier problems were necessary for solving later, more difficult problems. Once a wrong path was taken by the non-learning program, recovery was Impossible to achieve if the number of alternatives was very high, as was typically the case in complex problems.

In the later stages of the experiment, the contrast between MAX and the non-learning program became evident: if the non-learning program could find a solution to a problem, it generally did so more quickly than MAX. Because MAX was busy performing look-ahead at each node (by evaluating all the relevant macro-operators), it it could not take full advantage of $H_{diff}$ In pruning the search.

Overall, the results confirm our expectations. MORRIS'S t-macros appeared to extend the heuristic advantage provided by $H_{diff}$, resulting in a significant improvement in problem-solving ability. S-macros were more frequently useful, but resulted in less significant gains. Occasionally the extra time necessary to test for the applicability of s-macros slowed MORRIS down enough so that the non-learning system performed more efficiently, In either case, the extra computational expense incurred by saving these macros was more than offset by their benefits. We have yet to perform extensive experiments comparing t-macros and s-macros.

## 5 Conclusions

The approach to learning embodied in MORRIS Is rather unusual, since we have focused on the issue of "What to learn?" rather than "How to learn?". This issue can be crucial for a macro-operator learning system; if the acquisition of macro-operators is unbridled, the size of the search space defined by the set of macro-operators may grow rapidly, approaching the size of the original search space.

The two strategies MORRIS employs for evaluating the worth of a macro-operator have been found to be effective in controlling the learning process. Using these strategies, MORRIS maintains a balance between its reliance on knowledge, and its reliance on search.

In the future we hope to improve MORRIS by having it explicitly reason about the utility of control knowledge in order to direct its learning. We suspect that as machine-learning becomes better understood, the problem of deciding what is worth learning will assume greater importance.

## 6 Acknowledgements

The author thanks Jaime Carbonell for his many contributions to this research.

## References

1. Ernst, G. and Newell, A.. *GPS: A Case Study in Generality and Problem Solving.* Academic Press, 1969.
2. Pikes, R. *Monitored execution of Robot Plans produced by STRIPS.* Proceedings IFIP Congress, 1971.
3. Fikes, R., Hart, P. and Nilsson, N. "Learning and Executing Generalized Robot Plans." *Artificial Intelligence* 3,4(1972).
4. Fikes, R. and Nilsson, N. "STRIPS: A new approach to the application of theorem proving to problem solving." *Artificial Intelligence 2* (1971).
5. Korf, Richard E. *Operator Decomposabllity: A New Type of Problem Structure.* Proceedings AAAI-83,1983.
6. Minton, S. *Constraint-Based Generalization.* AAAI Proceedings, 1964.
7. Mitchell, T., Utgoff, P. and Banerjl, R. Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics. In *Machine Learning,* Carbonell, J., Michalski, R. and Mitchell, T., Ed.Tioga Publishing Co., 1983.
8. O'Rorke, Paul. *Generalization for Explanation-based Schema Acquisition.* Proceedings AAAI, 1984.
9. Porter, B. and Kibler, D. *Learning Operator Transformations,* AAAI Proceedings, 1984.
10. Sacerdoti, E. "Planning in a Hierarchy of Abstraction Spaces." *Artificial Intelligence 5* (1974).