LEARNING BY DISCOVERING MACROS IN PUZZLE SOLVING

Glenn A. Iba

MITRE Corporation
Bedford, MA. 01730

ABSTRACT

This paper proposes a model of learning by discovery. The model consists of a program which discovers macro operators while conducting a best-first heuristic search in the domain of puzzles. This work extends some recent work on permutation puzzles (Korf, 1982) and operator-decomposable puzzles (Korf, 1983), and is related to the earlier work on MACROPS (Fikes, Hart, and Nilsson, 1972). This work is part of a doctoral dissertation currently in progress at MIT, in which the model will be used to explore learning in conjunction with additional search paradigms and numerous alternative heuristics for macro generation and selection. The specific heuristic reported on here is that of using peaks of the evaluation function to segment the paths of the search tree in order to discover macros. The technique seems particularly valuable in difficult puzzles where only imperfect or approximate evaluation functions are available.

INTRODUCTION

This paper investigates a form of learning in the context of problem solving. A problem solving system (in this case a heuristic best-first search program) starts out with an initial set of operators, and learns by applying heuristic methods to discover a more powerful set of operators, herein called ***macro-operators***. The techniques are general and powerful, and show promise of extension to other search and problem solving paradigms.

Macro-operators (or macros) are a kind of super-operator which is composed of (or defined in terms of) more elementary operators. The composition process, by which a macro is formed, abstracts the defining sequence so that the new macro-operator looks like just another operator to the problem solving system. Thus a new macro enlarges the set of operators available during the search process. The fact that the new operator is defined in terms of other operators is incidental to the search process, and needs only to be invoked once a solution is found in order to convert the solution into its canonical form as a sequence of "primitive" (or initial) operators. This can be done by a straightforward process of macro expansion.

The discovery of macro-operators is a very powerful technique in problem solving, as demonstrated by the STRIPS system (Fikes, Hart, and Nilsson, 1972), which learned a form of macros (MACROPS) in the format of a triangular table, and some recent work (Korf, 1982) (Korf, 1983) in applying the learning of macros to the solution of a specific class of puzzles (operator-decomposable puzzles).

The sense of macro employed here is somewhat more general than that of Korf, since it is applicable in any problem where operators are ***composable***, i.e. where there exists a procedure for abstracting a sequence of operator applications to form a macro-operator represented in a manner consistent with the previous operators.

To illustrate the concept of macro, the familiar class of peg-jump solitaire puzzles will be employed. In these puzzles, legal operators consist of jumping a peg over another peg into a hole. Movement is typically restricted to the horizontal and vertical directions. The peg jumped over is immediately removed. The goal is usually to reduce the number of pegs to exactly 1. A number of board shapes and starting configurations are possible, but perhaps the most famous is the Hi-Q puzzle, which has a board shaped like a plus sign, and starts out with a hole in the center (see diagram 1). The goal is to end up with a single peg left in the center hole.

```
      o o o
      o o o
  o o o o o o o        Key:
  o o o . o o o          o  = peg
  o o o o o o o          .  = hole
      o o o              -  = don't
      o o o                   care
```

Diagram 1

The basic operator can be represented in terms of the initial and final states of the 3 consecutive locations affected by the jump (see diagram 2a). Similarly, a macro composed from a sequence of jumps can be represented in terms of the initial and final states of those locations affected at any point in the sequence. Refer to diagram 2 for a set of particularly useful macros for this class of puzzles. By convention "o" represents a peg, "." a hole, and "-" a don't care. Don't care's can be bound to either holes or pegs, but the bindings must be consistent between initial and final states. In each macro it is possible to get from the initial state to the final state by a sequence of legal basic jumps, which form the "expansion" or proof of the macro. These macros are very similar to the packages presented in (Berlekamp, Conway, and Guy, 1982). It is interesting to note that essentially the same set of macros was discovered independently by different human problem solvers, suggesting that the discovery process may be convergent, and certainly is of psychological interest.

The work presented here represents the first step in a more ambitious project to investigate learning by discovery of macros in a variety of puzzle and related domains (e.g. concept learning, and mathematical theorem proving). The advantage of choosing puzzles as the initial domain is that there already exists a well-developed set of performance systems for carrying out search. The learning components are a natural extension of these performance systems.

THE POWER OF MACROS

Macros derive their power from their ability to shorten the search process. Using macros it is possible to take "larger strides" through the search space, since applying a single macro may be equivalent to a large number of more primitive steps. The largest macros in diagram 2 (d,e, and f) are each equivalent to 6 primitive steps. The result is that a solution may be found in a

significantly smaller number of search steps than would be required by a search using only the basic operator.

The macro notion is very similar to the well-known chunking phenomena of psychology. Chunks provide significant combinatorial benefits for representing knowledge and reducing memory burden. In an analogous manner, macros allow for more efficient search, as well as more economical representation and recall of solutions.

Another aspect of macros is that they may be used to bundle up a lot of untidy details. In many puzzles, for example, it is necessary to pass through a number of "messy" states to get to the next "clean" state. Illustrations of this occur in the 15 puzzle and the Rubik's Cube, where it is frequently necessary to (temporarily) undo already satisfied subgoals in order to achieve the next subgoal. Macros provide a mechanism for getting through the mess to the next "good" state. With a macro, any messy intermediate states are hidden from the search process, since all that matters to it are the initial and final states of the macro. The obvious evaluation functions for puzzles such as these have the property that their values fluctuate up and down along a solution path. Macros can work especially well in conjunction with such imperfect evaluation functions, because the macro can aid the search process by guiding it past the intermediate messy (locally lower valued) states to the next good state. This observation directly suggests the learning heuristic for proposing macros which this paper describes: the peak-to-peak heuristic, where operator subsequences bounded by peaks in the evaluation function are proposed as candidate macros. This will be described in detail in the next section.

As a concrete illustration of the power of macros, the Hi-Q puzzle (diagram 1), which is extremely difficult to solve by conventional search techniques (the final solution is 31 steps long), can be solved quite easily by applying the macros of diagram 2. The solution in this case is only 7 (macro) steps long.

```
o       .      o       .      o o .       o . .
o  -->  .      o  -->  o        o    -->    .
.       o      .       .        o           .
               o       .
Basic-Jump     Take-Two       Three-in-a-Row
   (a)            (b)                (c)

o - .          o - .        . - o        . - o
  o o o  -->     . . .        o o o  -->   . . .
  o o o          . . .        o o o        . . .

     Remove-Six-A                Remove-Six-B
         (d)                          (e)

     o              .
   . o o          . . o          Key:
     o      -->     .               o  = peg
     o o o          . . .           .  = hole
                                    -  = don't
         Remove-L                       care
           (f)
```

Diagram 2

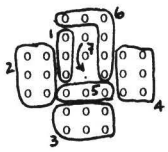
To concretely make the comparison, a best-first search strategy was implemented using the following 3 element evaluation function:

1. minus the number of connected components of pegs

2. minus the number of connected components of holes

3. minus the number of pegs remaining

where these 3 elements are strictly ordered, so that when comparing two evaluation vectors, any given element comes into play only when the prior elements are tied. The components of pegs (and holes) are computed assuming orthogonal adjacency (so that diagonal relationships do not connect components). The intent of these heuristics is to keep the pegs close together in a solid mass. The negations are used since for each of the three elements a smaller absolute value is better, thus the highest attainable value is (-1 -1 -1) when there is but one peg left. These heuristics were the first that were tried out, and they performed beyond expectations. Although the problem was still intractable in terms of searching with the basic-jump alone, the same search process, using the macros of diagram 2, found the solution of diagram 3 which uses 6 macro steps and a basic-jump. The search process was almost a perfect straight-line solution, with only one minor instance of backing up! Thus the use of macros converted an essentially intractable problem into one that was easily solved. An interesting side note is that the solution found by the program was one (macro) step shorter than the best solution I had found using the same set of macros.




```
Key:
  o  = peg
  .  = hole
  -  = don't
       care
```

Diagram 3

DISCOVERY OF MACROS

The model of learning driving this research is a simple but potentially powerful one. It is fundamentally a variant of generate and test, where macros are proposed, and subsequently filtered by both a static and a dynamic filtering process. It is expected that the proposing methods will generate too many macros, including some good ones along with many poor ones. The static filter will evaluate macros solely on the basis of their description, and discard those that fall below a threshold. The remaining macros will be tried out by the search process on a trial basis. Statistics will be maintained on how well each macro performs dynamically, and periodically the underachievers will be purged.

This paper explores one approach to the proposal of macros, and tests the learning effect with a "null" filter, that is, all candidate macros get accepted. The approach used, as part of a best-first search paradigm, is an implementation of the "peak-to-peak" heuristic. During the search process, all partial search paths are examined for peaks in the evaluation function. A peak occurs when a node of the search path has a higher value than both the immediately preceding and immediately following nodes. When a peak is encountered, the path is searched in the reverse direction until either a prior peak is detected, or the beginning of the path is reached. In either case the subsequence thus delimited (all operators after the starting peak node and preceding the terminal peak node) is proposed and converted into a macro by the following procedure:

1. Find the set of all locations used by any operator of the sequence. Call these the support locations.

2. Find the smallest rectangular window which contains all the support locations.

3. Using this window take a "snapshot" of the puzzle state at the start of the sequence. Mark the non-support locations (the complement of the support set relative to the window) as don't cares.

4. Take a similar "snapshot" of the puzzle state at the end of the sequence, again marking (the same set) don't care's.

5. Form a macro using the first snapshot as the macro's initial state, and the second snapshot as the macro's final state.

The snapshot process abstracts the macro away from the specific locations, and provides the possibility of the macro applying in a variety of positions and orientations, just like the basic-jump operator. Note that the matcher automatically tries out all rotations and reflections of each macro, just as it does with the basic-jump.

The evaluation function used was the same one discussed in the previous section. The learning process was tested out on a smaller version of the puzzle (see diagram 4). This puzzle was solved in 6 steps using the basic-jump operator alone. During this search two macros were proposed and created, though they were not immediately incorporated in the search. A second pass was taken on the same puzzle, this time incorporating the new macros to study the effect of learning. The second solution was discovered in less time, with fewer nodes expanded. The solution used the Three-in-a-Row macro twice to find a 2 (macro) step solution (diagram 5). The comparison of the first and second passes is summarized in Table 1. The two macros proposed during the first pass are listed in diagram 6.

```
o o o .        o o o .
. o o .        . o o .
. o o .        . o o .
```

Diagram 4 Diagram 5

```
o o .      o . .    o o      . .
  o   -->    .      o o -->  . .
  o          .      . .      o o
```

Diagram 6

Table 1

	Runtime (sec)	Realtime (sec)	Nodes Expanded	Nodes Evaluated	Solution Length	Macros Generated
Pass 1	6.3	66	15	34	6	2
Pass 2	5.0	27	3	17	2	0

The results indicate that interesting learning has taken place. Performance was uniformly better on the second pass. Moreover, at least one macro (Three-in-a-Row) has been learned which is known to be useful in solving related problems. For example, using the macros learned from the experience with this small problem (Diagram 4), this same best-first search procedure could solve diagram 7 in a straight-line fashion (diagram 8), whereas without the macros, the problem remains combinatorially intractable. This is an interesting example of transfer learning.

```
o o o o o o o o o o o .
. o o o o o o o o o o .
. o o o o o o o o o o .
```

Diagram 7

```
o o o o o o o o o o o .
. o o o o o o o o o o .
. o o o o o o o o o o .
```

Diagram 8

CONCLUSION

Macros are a powerful technique in problem solving, and the discovery of macros is an interesting form of learning worthy of additional study. The peak-to-peak heuristic for discovering macros has been successfully applied to small problems in the class of peg-jump puzzles, and generates some interesting and useful macros. Further study is called for to gauge its usefulness on larger problems, and to determine the ratio of good to poor macros generated by this technique.

The major limitation of the technique is that increasing the number of operators will increase the branching factor of the tree, and thus tend to slow down the performance system in its search. It is not clear whether the benefits of new macros will normally compensate for the increased search cost. Further work is required to evaluate this tradeoff.

Future work will explore techniques for filtering macros so that only the most promising will be learned. Additional experimentation will be carried out with related search paradigms such as GPS to explore goal-related heuristics for generating macros.

To the extent that macros are strongly analogous with theorems in mathematics and concepts in concept learning, the success of this research effort may shed significant light on issues of learning in these domains. It may be the case that powerful general heuristics for learning and discovery of composite structures may apply equally well in all three areas.

REFERENCES

[1] Berlekamp, E., J. Conway, and R. Guy, Winning Ways, vol.2, New York: Academic Press, 1982, pp. 697-704.

[2] Fikes, R., P. Hart, and N. Nilsson, "Learning and Executing Generalized Robot Plans", Artificial Intelligence 3:4 (1972) 251-288.

[3] Korf,R., "A program that learns to solve Rubik's Cube", in Proc. AAAI-82, Pittsburgh, PA., 1982, pp. 164-167.

[4] Korf,R., "Operator Decomposability: A new type of problem structure", in Proc. AAAI-83, Washington, D.C., 1983, pp. 206-209.

[5] Laird,J., P. Rosenbloom, and A. Newell, "Towards Chunking as a General Learning Mechanism", in Proc. AAAI-84, Austin, Texas, 1984, pp. 188-192.