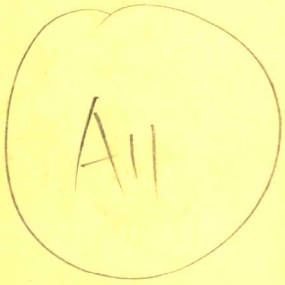
Richard E. Korf

Learning to Solve Problems by Searching for Macro-Operators



Richard E. Korf
Department of Computer Science,

Learning to Solve Problems by Searching for Macro-Operators

Pitman Advanced Publishing Program
BOSTON · LONDON · MELBOURNE

PITMAN PUBLISHING INC
1020 Plain Street, Marshfield, Massachusetts 02050

PITMAN PUBLISHING LIMITED
128 Long Acre, London WC2E 9AN

Associated Companies
Pitman Publishing Pty Ltd, Melbourne
Pitman Publishing New Zealand Ltd, Wellington
Copp Clark Pitman, Toronto

First published 1985

Library of Congress Cataloging in Publication Data

Korf, Richard E.
Learning to solve problems by searching for
macro-operators.
Bibliography: p.
1. Artificial intelligence. 2. Problem solving.
I. Title. II. Title: Macro-operators.
0335.K67 1985 001.53'5 85-3417

ISBN 0-273-08690-1

British Library Cataloguing in Publication Data

Korf, Richard E.
Learning to solve problems by searching for macro-operators.—(Research
notes in artificial intelligence)
1. Problem solving
I. Title II. Series
001.4'2 QA63

ISBN 0-273-08690-1

Reproduced and printed by photolithography
in Great Britain by Biddles Ltd, Guildford

Acknowledgements

This monograph is a revised version of a doctoral dissertation submitted to the Department of Computer Science at Carnegie-Mellon University. A number of people have made important contributions to this research. First and foremost is Herb Simon, my thesis advisor. His willingness to let me follow my own nose, his patience when it led me into dead ends, and his constant encouragement throughout was the most important contribution to the success of this work. Allen Newell provided many insightful critiques of this research. He raised a subtle question about the value of non-serializable subgoals that eventually led to the theory of the method. Ranan Banerji independently arrived at the basic idea behind this work. In addition, he carefully debugged the definitions, theorems, and proofs. Jon Bentley suggested the idea of interleaving the learning and problem solving components of the method. In addition, Herb, Allen, Ranan, and Jon carefully read and suggested many improvements to the manuscript. Merrick Furst exposed me to related work in group theory that provided a key contribution to this research, specifically, how to organize the macro table. Finally, I would like to thank my family, friends, and colleagues. Their moral support made this project possible.

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, and monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539.

Contents

1 Introduction and Summary

This monograph explores the idea of learning efficient strategies for solving problems by searching for *macro-operators*. A macro-operator, or *macro* for short, is simply a sequence of operators chosen from the primitive operators of a problem. The technique is particularly useful for problems with *non-serializable* subgoals, such as Rubik's Cube, for which other weak methods fail. Both a problem-solving program and a learning program are described in detail. The performance of these programs is analyzed in terms of the number of macros required to solve all problem instances, the length of the resulting solutions (expressed as the number of primitive moves), and the amount of time necessary to learn the macros. In addition, a theory of why the method works, and a characterization of the range of problems for which it is useful are presented. The theory introduces a new type of problem structure called *operator decomposability*. Finally, it is concluded that the macro technique is a valuable addition to the class of weak methods, that macro-operators constitute an interesting and important representation of knowledge, and that searching for macros may be a useful general learning paradigm.

1.1 Introduction

One view of the the field of artificial intelligence is that it is the study of *weak methods* [Newell 69]. A weak method is a general problem solving strategy that can be used when not enough knowledge about a problem is available to employ a more powerful solution technique. The virtue of the weak methods is the fact that they only require a small amount of knowledge about a problem and hence are extremely general. The set of weak methods includes generate-and-test, heuristic search, hill-climbing, and means-ends analysis. With the exception of generate and test, most of these techniques rely on a heuristic evaluation function which is used to estimate the distance to the goal. For some problems, however, no such evaluation function is known. This suggests that such problems do not have

sufficient structure to employ any technique more efficient than brute-force search to solve a particular instance of the problem.[1]

Consider, however, a situation where we are not interested in solving just one instance of the problem, but rather are concerned with being able to solve many problem instances. In that case, it may be advantageous to learn a general strategy for solving any instance of the problem, and then apply it to each problem instance. This allows the computational cost of the learning stage to be amortized over all the problem instances. Such an approach will only be useful if there is some structure to the collection of problem instances such that the fixed cost of learning a single strategy plus the marginal cost of applying it to each problem instance is less than the cost of solving each instance from scratch.

In other words, even though a given instance of a problem does not have sufficient structure to allow an efficient solution, a collection of problem instances may have some common structure that allows the whole set to be solved with much less work than the sum of solving each instance individually. This suggests the existence of weak methods for learning, as opposed to problem solving, based on such structure. This work explores one such weak method of learning, that of searching for macro-operators.

1.2 Chapter Summaries

This section presents a short summary of each of the remaining chapters.

1.2.1 Chapter 2: The Need for a New Problem Solving Method

Chapter 2 demonstrates that there exist problems that have efficient solution strategies that cannot be explained by any of the current stock of weak methods, and presents a 2x2x2 version of Rubik's Cube as an example. The goal state of this problem is naturally described as a conjunction of a set of subgoals. It is observed that all known algorithms for this problem require that previously satisfied subgoals

---

[1] The terms "problem" and "problem space" in this monograph refer to a set of states and a collection of operators that connect them. A "problem instance" is a problem with a specified pair of initial and goal states.

be violated later in the solution path. Such a set of subgoals is referred to as *non-serializable*. However, the standard technique for solving problems with subgoals, means-ends analysis, does not allow non-serializable subgoals. Furthermore, we present empirical evidence that several natural heuristic evaluation functions for the simplified Rubik's Cube provide no useful estimate of distance to the goal, suggesting that heuristic search is of no use in solving the problem. Hence, Rubik's Cube cannot be solved by any of these techniques.

1.2.2 Chapter 3: Previous Work

Other work related to this research is reviewed in Chapter 3. Ernst and Goldstein wrote one of the first programs that learned efficient strategies for solving problems, by learning differences for the General Problem Solving program of Newell and Simon. Non-serializable subgoals were studied extensively in the context of the blocks world by Sussman, Sacerdoti, Warren, Tate, Manna and Waldinger, and others. Macro-operators were first learned and used by the STRIPS problem solver and later by the REFLECT system of Dawson and Siklossy. Banerji suggested the use of macros to deal with the non-serializable subgoals of the Rubik's Cube and the Fifteen Puzzle. Finally, Sims and others showed how to organize sets of macros to solve permutation puzzles, of which Rubik's Cube is an example, and demonstrated one way the macros could be learned.

1.2.3 Chapter 4: The Macro Problem Solver

Chapter 4 describes the *Macro Problem Solver*, an extension of the General Problem Solver to include macro-operators. The basic idea of the method is to apply macros that may temporarily violate previously satisfied subgoals within their application, but that restore all previous subgoals to their satisfied states by the end of the macro, and satisfy an additional subgoal as well. The macros are stored in a two dimensional table, called a *macro table*, in which each column of the table contains the macros necessary to satisfy a particular subgoal. The subgoals are solved one at a time, by applying a single macro from each column of the table. The Macro Problem Solver generates very efficient solutions to several classical problems, some of which cannot be handled by other weak methods. The examples include Rubik's Cube, the Eight and Fifteen Puzzles, the Think-a-Dot problem, and the Towers of Hanoi problem.

1.2.4 Chapter 5: Learning Macro-Operators

The question of how macros are learned or acquired is the subject of Chapter 5. The simplest technique is a brute-force search. However, by using a technique related to bidirectional search, the depth of the search can be cut in half. Finally, existing macros can be composed to find macros that are beyond the search limits. These techniques are sufficient for learning the necessary set of macros for the example problems. In addition, a design for a general *Macro Learning Program* is presented. The design clearly separates the problem-dependent components of the method from the problem-independent features. A key property of the learning program is that *all* the macros necessary to solve *any* problem instance are found in a *single* search from the goal state.

1.2.5 Chapter 6: The Theory of Macro Problem Solving

Chapter 6 explains the theory of macro problem solving and characterizes the range of problems for which it is effective. The theory is presented in two parts: a special case in which a state is represented by a vector of state variables, and the general theory that encompasses arbitrary state representations. A necessary and sufficient condition for the success of the method is a new type of problem structure called *operator decomposability*. A totally decomposable operator is one that may affect more than one state variable, but whose effect can be decomposed into its effect on each state variable independently. The degree of operator decomposability in a problem constrains the ordering of the subgoals, ranging from complete freedom in the case of Rubik's Cube, to a total ordering for the Towers of Hanoi problem. In addition, further generalizations of the method are presented. For example, we show that in some cases, efficient solution strategies can be learned based on randomly generated subgoals!

1.2.6 Chapter 7: Performance Analysis

An analysis of the performance of the problem solving and learning programs is presented in Chapter 7. The performance measures include the number of macros that must be stored for a given problem, the amount of time required to learn the macros, and the length of solutions generated in terms of number of primitive

moves, both in the worst case and the average case. The first result is that the total number of macros is the sum of the number of macros in each column whereas the number of states in the space is the product of these values. The total learning time for the macros is shown to be of the same order as the amount of time required to find a solution to a single problem instance without the macros. Finally, if there are N subgoals to a problem, the solution length generated by the Macro Problem Solver is less than or equal to N times the optimal solution length, in the worst case. In addition, an average case analysis of solution length is found to agree with experimental results for the 2x2x2 Rubik's Cube. Furthermore, for the Eight Puzzle and the full 3x3x3 Rubik's Cube, the solution lengths generated by the Macro Problem Solver are close to or shorter than those of an average human problem solver. An important feature of this analysis is that each performance parameter is expressed in terms of a corresponding measure of problem *difficulty*, rather than problem *size*. For example, the worst-case solution length is expressed in terms of the optimal solution length.

1.2.7 Chapter 8: Reflections and Further Work

Several observations and directions for future research are presented in Chapter 8. First, the selection of subgoals and their ordering are two parameters of the Macro Learning Program whose automatic generation requires further research. Next, we show that the Macro Problem Solver can be combined effectively with other problem solving methods such as operator subgoaling, macro generalization, and problem decomposition, to solve problems that no single technique could solve alone. In addition, we argue that given an ordered set of subgoals for a problem, the difficulty of the problem is related to the maximum distance between two successive subgoals, in terms of number of primitive moves. Next, we propose that macro-operators are an important representation for knowledge, based on a brief look at the domains of theorem proving and computer programming, and a detailed examination of the domain of road navigation. Finally, an exploration of the utility of macros in arbitrary problem spaces suggests that searching for macro-operators may be a fairly general learning paradigm.

1.2.8 Chapter 9: Conclusions

Chapter 9 presents the conclusions of this research. They include the finding that the macro learning and problem solving techniques constitute a valuable addition to the collection of weak methods, the idea that macro-operators are an important representation for knowledge, and the suggestion that searching for macros may be a useful paradigm for learning.

2 The Need for a New Problem Solving Method

The purpose of this chapter is to demonstrate that the existing collection of weak methods is incomplete. There exists a problem, namely Rubik's Cube, that cannot be solved efficiently by any of the current stock of weak methods. Yet, people can solve it, and even learn to solve it, quite efficiently. Hence, another method must underly the solution of this problem. In addition, we will argue that for other reasons as well, Rubik's Cube is an excellent domain for studying problem solving and learning.

2.1 Problem Description: 2x2x2 Rubik's Cube

Figure 2-1 shows a 2x2x2 version of the celebrated Rubik's Cube, invented by Erno Rubik in 1975. The puzzle is a cube that is cut by three planes, one normal to each axis, separating it into eight subcubes, referred to as *cubies*[1]. The four cubies on either side of each cutting plane can be rotated in either direction with respect to the other four cubies. Note that these rotations, called *twists*, can be made along each of the three axes. The twists can be 90 degrees in either direction or 180 degrees.

Each of the cubies has three sides facing out, called *facelets*, each a different color. In the goal state of the puzzle, the four facelets on each side of the cube are all the same color, making six different colors in all, one for each side of the cube. The cube is initialized by performing an arbitrary series of twists to mix the colors on each side. The problem then is to *solve* the cube, or find a sequence of twists that will restore the cube to the goal state, i.e. each side showing a single color.

The 2x2x2 cube is a simpler version of Rubik's original cube. The original is a

[1] The terminology used here is standard in the literature of Rubik's Cube [Frey 82].

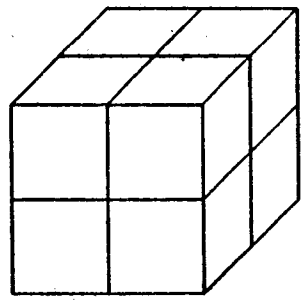


Figure 2-1: 2x2x2 Rubik's Cube

3x3x3 cube with two planes of rotation cutting each axis (see Figure 2-2). The 2x2x2 cube is a subproblem of the 3x3x3 cube: it is isomorphic to a restriction of the full cube in which only the eight cubies on the corners are considered. In other words, if one ignores the interior edge and center cubies of the 3x3x3 cube, then the problem reduces to the 2x2x2 cube. Both problems will be considered in this work.

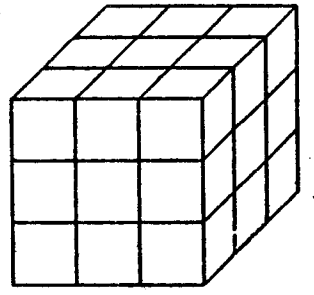


Figure 2-2: 3x3x3 Rubik's Cube

2.2 Rubik's Cube as a Domain for Problem Solving and Learning Research

There are several reasons why Rubik's Cube is an excellent domain for research on problem solving and particularly on learning problem solving strategies.

First, note that there are two levels of tasks associated with the cube. One is the task of given a particular initial configuration, find a sequence of twists that will restore it to the goal state. This is the problem solving task. The other is the

learning task of acquiring a strategy that will solve the cube from any initial state. The reason for this distinction is that the puzzle is really a collection of a very large number of problem instances, one for each possible initial state[2].

An obvious reason for studying Rubik's cube is that the problem is well structured yet very difficult. Since the states and operators are clearly specified and easily represented, one can easily construct a problem space for the problem. That the problem is genuinely difficult is attested to by the phenomenally large number of people who have unsuccessfully worked on scrambled cubes. The published strategies to the problem are all fairly complex in the sense that it is considered a significant achievement to learn one of them. Furthermore, the problem of discovering a strategy is even more difficult. Most people who try it never succeed, and those who do succeed typically require several weeks to several months of effort.

Not only does it take a long time to learn a strategy, but progress toward it is incremental and observable. Many problems are difficult and require a long time to solve, but the solution, once discovered, becomes apparent instantaneously. In Rubik's Cube, progress toward a strategy occurs throughout the learning process and can be measured in terms of the number of cubies that can be correctly positioned relative to the goal state. In addition, it is usually clear what pieces of knowledge are being acquired during the learning. These features of the problem make it an ideal domain for studying the learning of problem solving strategies.

Finally, the most compelling reason for studying Rubik's Cube is the fact that it cannot be solved efficiently by any of the current stock of weak methods. After describing a problem space for the 2x2x2 cube, evidence supporting this claim will be presented.

---

[2]The term "solution" is used to refer to a sequence of primitive moves that maps a particular initial state to a particular goal state. The term "strategy" refers to an algorithm that will generate a solution to any problem instance.

2.3 Problem Space

This section presents a problem space for the 2x2x2 cube by describing a data structure to represent a state or configuration of the cube, and giving a procedural implementation of each of the primitive operators of the puzzle. In general, the task of going from a problem description to a representation of the problem is complex, and if done cleverly can result in a vast reduction in problem solving effort. In this case, however, the representation is based on relatively straightforward observations and does not significantly reduce the difficulty of the problem.

2.3.1 State Representation

The primary issue in generating a problem space for any problem is designing a data structure to represent a state of the problem. Perhaps the most obvious state representation would be to list in some order the colors that appear on each facelet of the cube. However, the choice of a facelet as a primitive results in an inefficient representation. The reason is that the facelets are physically constrained to occur in fixed groups of three by virtue of being attached to particular cubies, which move as units. Incorporating this constraint directly in the representation gives rise to a more efficient representation.

By choosing a cubie as the primitive of the representation, we are led to represent a cube configuration as a permutation of the cubies among the different positions, or *cubicles*, that the cubies can occupy. In addition, a particular cubie can exist in the same position but with its colors twisted in any of three different orientations, one corresponding to each facelet of the cubie. The three orientations will be labelled 0, 1, and 2. The orientation is determined by examining the unique facelet of each cubie that faces either up or down in the goal state of the cubie. Its orientation is the number of 120 degree clockwise rotations of the cubie about an axis from the center of the cube through the corner of the cubie which would map the up or down facelet from the top or bottom side of the cube to its current position.

Thus, each cubie must be represented by both its position and its orientation. This suggests an eight element array of cubies, where each element encodes both