Similarity-Based Retrieval
and Partial Reuse of Macro-Operators

Hua Yang
Douglas Fisher

Technical Report CS-92-13

Department of Computer Science
Vanderbilt University

1992
Similarity-Based Retrieval
and Partial Reuse of Macro-Operators

Hua Yang
Doug Fisher

Department of Computer Science
Box 1679, Station B
Vanderbilt University
Nashville, TN 37235

dfisher@vuse.vanderbilt.edu
(615)343-4111

Abstract: Planning efficiency can be improved by decreasing the effective
depth and/or breadth of search. Macros take large steps in a search space,
thus promising to decrease depth, but at the cost of increased breadth. Ab-
straction over (macro) operator definitions can reduce the branching factor of
search. Our particular approach is one of clustering operator definitions into
similarity classes. Operators are retrieved that 'best match' new problems.
If the retrieved operator is a macro, then it may be modified and reused
effectively. We term this modification and reuse process partial use of macro
operators. Collectively, similarity-based selection and partial use of macros
are advantageous relative to some common alternatives.

Area: Learning, Automated Reasoning (Planning).
1 Introduction

Planning is a critical aspect of intelligence. Traditionally, planning has referred to a deliberative process of projecting one’s actions onto the world before acting. Computational complexity stems from coordinating many actions that may interact in nonobvious ways. The usual strategy for mitigating this complexity is subtasking: a problem is decomposed into smaller parts, each is solved, and the solutions are weaved into a global plan. Mean-ends analysis as found in STRIPS (Fikes, Hart, & Nilsson, 1972) is an exemplar of this strategy, in which one uses the goals specified in a problem to delineate subproblems. In particular, operators for achieving a goal are found, and constrain subplans for achieving subsequent goals. Failure to find a global plan for a conjunctive goal leads to backtracking – the exploration of alternative plans with different operators and/or different goal orderings.

Work in case-based planning takes a very different approach to planning (Hammond, 1989). In response to a new problem, plans are retrieved from a case library that succeeded under similar conditions in the past. These plans are modified to fit the conditions of the new situation and executed. Ideally, planning is rendered efficient relative to a subtasking approach of constructing plans ‘from scratch’. Case-based approaches, however, often lack the ability to plan when the case library is ‘incomplete’ or new problems require plans at a considerably different level of complexity as those stored in the case library.
Ideally, we would like to combine the flexibility of subtasking with the efficiency of case-based planning (or planning through retrieval and reuse). Steps in this direction include the identification and reuse of macro-operators, which are composites of more primitive operators. The composite operators are typically used in a manner similar to primitive operators, but they take bigger steps towards a solution. Thus, macro-operators are motivated by a desire for increased efficiency as are CBR approaches, but systems that learn and reuse macros are not typically able to modify them in a manner that best exploits their power.

This paper describes a system, PLOT, that combines the benefits of subtasking and case-based approaches by learning, modifying, and reusing macro-operators. The system's partial use strategy of exploiting macros is contrasted with more typical full and limited use strategies, which simply use macros as is. In addition, we show that partial use is most advantageous when it is coupled with an appropriate selection strategy; the PRE and POST conditions of the macro must be 'similar' to the task at hand, else modification is unlikely to improve planning efficiency, and is quite liable to hurt it by expending wasted effort. To promote retrieval of appropriate macros, PLOT organizes operators into similarity classes, that collectively form an abstraction hierarchy over operators. We discuss the general advantages of operator abstraction next, then move to our particular construction of similarity classes, and our strategy for macro reuse.
2 Abstraction in Planning

In planning, or any search process generally, we can improve efficiency by reducing the effective depth of search and/or reducing the effective branching factor. As composites, macro operators take large steps in search, thus promising to reduce effective depth, but their introduction also adds considerably to the branching factor. In many cases, a reduction in the effective depth may be outweighed by an increase in effective breadth.

Consider a set of operators defined in terms of their preconditions and projected effects on the environment. Similarity among PRE and POST conditions can confuse a planner’s decision about the ‘best’ operator to apply under current circumstances. This leads to a nonoptimal behavior known as the problem-solving fan effect (Shrager, Hogg, & Huberman, 1987): operators can generally be found most quickly (minimal backtracking) for problems that are most idiosyncratic, since the relevant operators are more likely to be idiosyncratic and thus more likely to be deterministically identified as relevant; operators that are highly similar to other operators are more likely to be sources of confusion for the planner.

Intuitively, we would like problem solving to be most efficient for the problem’s exhibiting the most common patterns! We can achieve this desired result through operator abstraction. In particular, by abstracting over operators we expose the problem solver to ‘abstract operators’ that are relatively distinct so that an unambiguous decision as to the correct abstract
operator can be made at each level of the hierarchy.

If we assume that the hierarchy is such that at each level a 'correct' operator is selected deterministically, and that each node contains $n$ (abstract) operators as children, and that there are $T$ leaves (i.e., executable operators), then we can show formally (Yang, 1992) that the number of operators that need be inspected is reduced from $O(T)$ (with no abstraction) to $O(\ln T)$ for appropriate $n$ at each planning step.

The analysis that leads to this result is related to and inspired by similar analyses (Korf, 1897; Knoblock, 1990) concerned with decreasing effective depth by searching in state abstraction spaces. Here we are not concerned with such a search, but only in the selection of appropriate executable operators at each step in a search through the original state space by exploiting an operator abstraction hierarchy. The effect is to reduce effective branching and to mitigate the problem solving fan effect. The critical assumption if we are to approximate the best case behavior indicated by the analysis is that we can deterministically traverse the abstract operator hierarchy from the root to an appropriate leaf. In general, this is not possible, thus backtracking is still required, though the hierarchy can guide the selection of alternatives. In any case, however, operators must be organized by some similarity metric that facilitates near-deterministic traversal.
3 Operator Similarity Classes

In traditional means-ends problem solving the problem solver finds operators that most reduce the differences between the current conditions and goal state. In STRIPS, this idea is implemented as a search for operators with ADD conditions that match the goal. Intuitively, we would like more than this: operators should both achieve unsatisfied goals and possess preconditions that best match the current state, thus reducing the need for backward chaining on the preconditions of the selected operator. Operator similarity classes are formed by clustering the operator definitions using a measure of match between operator PRE and ADD conditions. The particular clustering strategy that is used is based on Fisher's (1987) COBWEB, which forms a categorization hierarchy of operator similarity classes.

Each abstract operator discovered through clustering is probabilistically defined. Figure 1 illustrates an abstract operator definition over two primitive operators. The operator definition is represented at three levels: by the PRE, ADD, and DEL lists, by the predicates occurring in each list, and by the arguments to each predicate. Frequency counts of the number of occurrences over class members for each predicate and for each argument are stored. In addition, the number of members (i.e., primitive operators) are stored with the abstract operator. Intuitively, we wish to group operators that agree on many dimensions – their predicates within each list and their arguments to each predicate.
Class (Goto-Door Goto-Object): 2.0

Precondition:
inroom: 2.0
  parameter 1: robot (robot 2.0)
  parameter 2: room (room 2.0)
inroom: 2.0
  parameter 1: X? (door 1.0) (object 1.0)
  parameter 2: room (room 2.0)

Delete:
nextto: 2.0
  parameter 1: robot (robot 2.0)
  parameter 2: anything (anything 2.0)

Add:
nextto: 2.0
  parameter 1: robot (robot 2.0)
  parameter 2: X? (door 1.0) (object 1.0)

Figure 16: A class represented in the structured probabilistic format.

Figure 1: An abstract operator definition.

The quality of a class or abstract operator is defined by

\[ P(\text{class}) \times \sum_{\text{List} \in \{\text{PRE}, \text{ADD}\}} \sum_{\text{pred} \in \text{List}(\text{class})} \sum_{\text{par} \in \text{pred}} [P(\text{par} = \text{val} | \text{class})^2 - P(\text{par} = \text{val})^2] \]

This measure guides the search for meaningful operator classes. Intuitively, an abstract operator is good if it is likely to be applicable (i.e., reflected in \( P(\text{class}) \)), and it is likely to be useful, which is reflected by the triple sum. This latter term quantifies usefulness as our certainty in the truth and appropriateness of predicates and parameters of PRE and ADD conditions, respectively, at the time of operator selection. We will not delve into the measure beyond this, but note that it is a function of both PRE and ADD conditions, and that it separates operators into classes of high
Figure 2: An abstraction hierarchy over STRIPS operators in the robot planning domain.

within-category similarity and low between-category similarity. The best partitioning of operators into classes is determined, and the procedure is recursively applied to each class, thus generating an abstraction hierarchy with primitive operators at the leaves.

Figure 2 illustrates an operator abstraction hierarchy over the eight STRIPS operators of the robot planning domain. We do not include the full probabilistic definition of each abstract operator, but only those conditions that are common to all members of the class (i.e., those that occur with probability 1.0). A final relevant aspect of this clustering procedure is that it is incremental. The measure used to construct the initial hierarchy can guide categorization and placement of new operators. This is an important feature,
if new macro operators are to be learned and stored for reuse.

The operator abstraction hierarchy is used in planning by treating the problem statement as a hypothetical operator definition; the initial state serves as a PRE list, and the conjunctive goal serves as an ADD list. Intuitively, the leaf of the hierarchy found through categorization is an actual operator that best matches the ‘hypothetical’ operator that will solve the problem. This actual operator is retrieved for means ends planning. The categorization path is remembered, and can guide backtracking, if necessary.

4 Learning Macro Operators

As new problems are solved the planner has the opportunity to store macros for future use. In particular, the system analyzes a successful planning tree—a binary tree in which each node is an operator, the left subtree is composed of operators that achieve unsatisfied preconditions of the node, and the right subtree contains operators that achieve the remaining unachieved goals of the node’s parent task. PLOT packages together operators that collectively achieve one goal. In particular, every node and its entire left subtree (i.e., its support) are used to construct a macro operator. The rationale for this packaging strategy is that it yields macros that can be applied independently to achieve a top-level goal, regardless of the conjunctive goal in which this goal appears. Successful operator orderings (i.e., sensitive to goal interactions) for achieving the conjunctive goal which constitutes the preconditions of a
Figure 3: Packaging macros from a planning tree.

node are still retained as the left subtree of that node. Figure 3 illustrates the macros that are formed from a general planning tree.

Having packaged a macro, we wish to place it into the operator hierarchy, so that it can be retrieved and reused under appropriate circumstances by the means ends planner. To do this we must construct PRE, ADD, and DEL lists for the macro. PLOT considers the linear order of the operators obtained by an inorder traversal of the appropriate plan subtree. We then initializes the PRE, ADD, and DEL lists of the macro as follows:

\[
\begin{align*}
PRE_1 &= PRE(Op_1) \\
ADD_1 &= ADD(Op_1) \\
DEL_1 &= DEL(Op_1)
\end{align*}
\]

PLOT iteratively analyzes each subsequent operator and updates the
macro lists at the \( m \)th operator so that

\[
PRE_m = PRE_{m-1} \cup [PRE(Op_m) - ADD_{m-1}]
\]

\[
ADD_m = ADD(Op_m) \cup [ADD_{m-1} - DEL(Op_m)]
\]

\[
DEL_m = DEL(Op_m) \cup [DEL_{m-1} - ADD(Op_m)]
\]

Intuitively, preconditions of a macro are the preconditions of each operator that are not achieved by ADD conditions of preceding operators. Similar intuitions apply for the ADD and DEL lists. The macro and its planning subtree are stored in the operator hierarchy based on their similarity to existing classes along the PRE and ADD lists.

5 Planning with Partial Use of Macro Operators

As operators are incorporated into the abstraction hierarchy they can be retrieved and reused in the same manner as primitive operators. This straightforward strategy is known as full use of macro-operators. Unfortunately, full use has some problems. For example, if a macro operator of going through 15 consecutive rooms with all doors open is used to solve a very similar problem of going through the same 15 rooms with the second-to-last door closed, it is unacceptable to let the Robot go through these rooms, open the door, and return, just so that we can reuse the macro. This has motivated a limited use strategy (Mooney, 1989), in which macros are only used if they completely solve a given subproblem – no backward chaining on the macro’s precondi-
tions is allowed. This eliminates the problems associated with full use, but its extreme stance does not exploit macros in many situations where they might be useful with some slight modification.

5.1 Partial Use of Macro Operators

We propose a partial-use strategy. If all preconditions of a macro are satisfied by the current state then it is used as in limited use. If some preconditions are unsatisfied then instead of initially backward chaining on its unsatisfied preconditions as in full use, or discarding it as in limited use, the planner examines each operator within the macro-operator, and adds and/or deletes certain operators to make the macro workable for the given problem.

This modification process is initiated by restoring the planning tree stored with the macro and considering its root. If the root operator does not achieve any of the desired current goals (i.e., it is useless), or it achieves goals that are already satisfied in the current state (i.e., it is redundant), then it is removed, otherwise it is kept since it achieves an unsatisfied goal (i.e., it is useful). If the root is kept, then analysis moves to its left subtree to see whether operators stored here achieve the necessary preconditions. This analysis process is recursive, in that operators of the left subtree are evaluated relative to their parents preconditions. If an operator at a node is a primitive operator, then examination of its ADD list determines whether it achieves any of its parent's unsatisfied preconditions and it is removed or retained accordingly. However, if a node is a macro operator itself, and it is useful,
then it is recursively evaluated by partial use to make it workable under current conditions.

After examining the left subtree of a node, if there are no remaining unsatisfied preconditions of the node’s parent task, then the right subtree of the node is truncated, since its operators are no longer needed to achieve any parent preconditions. If parent preconditions do remain unsatisfied, then the node’s right subtree is recursively evaluated; if no right subtree exists, then the means-ends planner is called to add additional operators to achieve remaining parent preconditions.

If at any point in recursive analysis, an operator is deleted, then its right-subtree is promoted as the root, since the right subtree was originally included to achieve remaining goals of the parent task; the assumption is that it is more likely to obtain such parent goals in the current context. If a right subtree is empty, as is the case when we are examining the root of an entire macro, then the left subtree is promoted. Figure 4 illustrates this process.

5.2 Related Work

STRIPS can edit a macro to better fit current planning requirements. However, STRIPS particular strategy of using triangle tables may only delete operators. STRIPS applies a full use strategy plus deletion. In contrast, we retain the planning tree so that operators may be added by recursive application of the planning module in an appropriate way.

Partial use is also similar to the kinds of modifications that can occur
in CBR and its ancestors such as HACKER (Sussman, 1975). Deficiencies in a plan (or macro) can be remedied by searching its case library (or answer library in the case of HACKER) for other macros or parts thereof that can patch the deficiency. We view these systems as performing a kind of limited use with the possibility of hard-coded additions. In contrast, partial use exploits the full power of the means-ends planner to patch a macro. Note that the planner can retrieve other macro operators in this process, and thus need not build the patch from scratch.

Allen and Langley (1989) and Anderson and Farley (1988) also form operator hierarchies, but neither stores and reuses macro operators.
5.3 Selection by Similarity and Partial Use

Intuitively, it is important that partial use be used in conjunction with an operator selection strategy that returns (macro) operators that are similar to current conditions and goals, else considerable effort may be wasted. As previous sections indicate, selection by similarity is the strategy that we use. However, other selection strategies have been proposed. One strategy is to examine macro operators in the order in which they were constructed, since macros discovered early are more likely to apply to more common classes of problems – an observation that is true, if a problem population is sampled randomly. A second strategy is to examine macros by size; focusing on the largest macros first may bring about the largest gains in problem solving.

5.4 Experimental Results

To evaluate similarity-based selection and partial use, we examined 60 artificially-constructed robot planning problems. Problems were ordered by the length of the plan required to solve the problem – i.e., the planner was trained from 'easiest' to 'hardest' problems. The planner accumulated and added macros during training. Intermittently, we tested the planner on the full set of 60 problems with learning 'turned off'. Figure 5 illustrates the performance of our planner, a traditional means-ends planner, a planner endowed with an abstraction hierarchy over the primitive operators but no macro learning ability, and an important variation to our planner. The notable phenomena are that
simply employing an abstraction hierarchy over the primitive operators does well ($\approx 220$ unifications) relative to a traditional planner ($616$ unifications). In addition, partial use of acquired macro operators improves performance for a time (to $129$ unifications), but performance degrades after many macros are added to the hierarchy. The basic reason is that probabilistic class definitions become less distinct as more operators are added, thus confusing similarity-based selection. There are many possible fixes to mitigate this, including other categorization structures, but we have implemented a simple strategy of pruning macro operators that are either infrequently used and/or generally incorrect as measured by a simple product of these factors. Macro operators that are evaluated as falling below a threshold of acceptable performance are removed. This strategy significantly mitigates the utility problem (Minton, 1988), and appears fairly robust across a range of thresholds.
Applying this pruning strategy at the point indicated, served to remove 24 macro operators, reducing the set of operators in the hierarchy from 41 to 17 (8 primitives and 9 macros). This led to an increase in performance over the problem set (116 unifications).

Our real goal here is to evaluate the utility of partial use. In particular, we examine partial use with various operator selection strategies. The bar graph of Figure 6 illustrates partial use performance when used in conjunction with selection strategies based on macro operator operator size, learning order, and similarity. This is a snapshot of planner performance at the point where 19 operators are stored in the hierarchy. Performance indicates that partial use only makes sense when we select based on similarity.

Figure 6 indicates the relative merits of partial use with various selec-
Figure 7: Performance of limited use.

tion strategies. Figure 7 evaluates limited use in the same way. In this case, similarity-based ordering of macros is still preferred, in part, because it prefers operators with preconditions that match current conditions, as well as the desired goal conditions. However, comparisons across graphs indicate that limited use is also advantageous in conjunction with selection based on learning order. Finally, we note that similar experiments were performed with a full use strategy, which does very poorly regardless of operator selection strategy.
6 Concluding Remarks

In sum, we have introduced the idea of similarity-based retrieval and partial use of macro operators. These selection and use strategies are compatible, and best applied in conjunction. Both ideas stem from work in CBR, but our goal was to integrate them into a traditional means-ends planner, which retained the flexibility to solve new and novel problems, and to patch deficiencies in macros through subtasking, as well as by using portions of other macros. A secondary idea is that operator selection should be guided both by conditions that an operator can achieve so that forward progress can be made, and by the match between the preconditions and current state, thus minimizing backward chaining on the preconditions of the selected operator.

References


