gak gak

gak gak

gak gak

gak gak

gak gak

gak gak

In *Proceedings of the AAAI Spring Symposium on Computational Considerations in Supporting Incremental Modification and Reuse* (1992). Stanford, CA: AAAI Press.

Knowledge and Regularity in Planning

John A. Allen (Allen@ptolemy.arc.nasa.gov)
AI Research Branch (Sterling Software), M/S 269-2,
Nasa Ames Research Center, Moffett Field, CA 94035 USA

Pat Langley (Langley@ptolemy.arc.nasa.gov)
Artificial Intelligence Research Branch, M/S 269-2,
Nasa Ames Research Center, Moffett Field, CA 94035 USA

Stan Matwin (Stan@csi.uottawa.ca)
Department of Computer Science, University of Ottawa,
Ontario K1N 6N5 Canada

1. Introduction

Researchers have explored three broad approaches to using domain-specific knowledge in the planning process. Analogical or case-based methods retrieve stored plans and adapt them to novel problems (Hammond, 1990; Kambhampati, 1990; Veloso & Carbonell, 1989). A second scheme uses stored plans, or macro-operators, as though they were primitive operators (Fikes, Hart, & Nilsson, 1972; Korf, 1982; Iba, 1989). A third approach draws on more distributed search-control knowledge to direct the selection of operators, states, or goals (Laird, Rosenbloom, & Newell, 1986; Minton, 1990a; Jones, 1989).

In this paper we present Talus, a system that can use domain-specific plan knowledge in each of these modes. This lets us compare the three approaches experimentally to determine the conditions under which each is most appropriate. We begin by describing Talus' representation of plan knowledge, followed by its behavior when using this knowledge, analogically, as macro-operators, and as search-control rules. After this we propose hypotheses about the relation among these methods, characteristics of planning domains, and the behavior of the methods in such domains. We then report experimental studies that test these hypotheses. In closing we draw some tentative conclusions and outline directions for future work.

2. The Talus Planner

Previous researchers have assumed that the analogical, macro-operator, and search-control approaches to planning are completely distinct. In contrast, we believe these three methods are special cases of a more general framework, which views the methods as differing along a continuum involving the conditions of reuse. In particular, methods like analogical and case-based techniques fall toward the center of this continuum, in that they check each component of a retrieved plan for its appropriateness to the current subproblem. Methods that treat plan knowledge as macro-operators do not bother to test for relevance; they simply assume that all components are appropriate for use on the current subproblem. Search-control methods also dispense with such tests, but because they assume that subplans are never relevant, and so always search memory for knowledge they can apply to the current subproblem. Thus, the continuum runs from automatic reuse of components, through conditional reuse (which requires testing), to automatic nonreuse.

We have implemented this framework in Talus, a system that can use stored plans in each of these three ways depending on the setting of a global parameter. The system uses a variant of means-ends analysis (Newell & Simon, 1972) to construct totally ordered plans using depth-first search, but we believe the basic ideas carry over to other planning and search schemes. In this section we describe the structures and processes used in Talus. We begin by considering those features shared by the three modes; we then discuss how each method fits into the framework in turn.

2.1 Representation, Retrieval, and Control

An example from the Strips robot domain may clarify the framework that Talus instantiates. The solid lines in Figure 1 depict a plan for moving a robot from one room to another through an open door. This plan takes the form of a means-ends trace, with ellipses representing problems (and subproblems) and rectangles indicating operators. Although not shown, each problem node also specifies the current state and the differences between that state and the desired one.

The top-level plan (`S1`) specifies an operator, in this case `(move-to ?y)`, which leads to one subproblem and its associated subplan (`S2`) for transforming the current state into one that matches the operator's preconditions. In general, such traces can also include a second subproblem (and associated subplan) to transform the state resulting from the operator application[1] into the original desired state, but in this case that is unneces-

[1] In all cases, we use the term 'apply' to indicate simulated application rather than execution in the world.

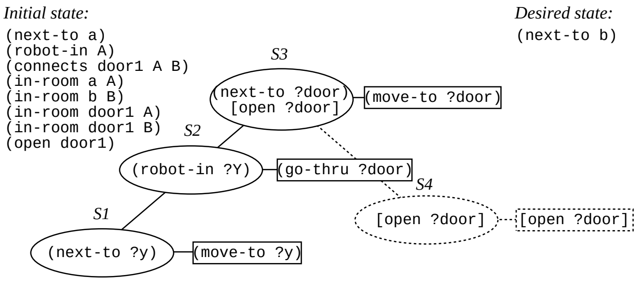

Figure 1: A plan for moving from one room to another through an open door (solid lines and expressions in parentheses) and a revised plan for moving through a closed door (dotted lines and bracketed expressions).

sary. Each subproblem has the same recursive structure, which halts with subproblems that can be solved using a primitive operator.

Talus can use its knowledge of primitive operators, combined with means-ends analysis, to generate plans of this sort. At each step in developing a plan, the system retrieves operators that reduce one or more differences in the current problem. It selects one of these operators based on the current set of differences. The memory system returns all primitive operators that reduce differences in the current problem. The returned set of primitive operators is then ordered by the number of differences they resolve and used as the list of alternatives for the current point of the means-ends search.

Once an operator is selected, Talus creates two associated subproblems, either or both of which may be degenerate. If the first subproblem contains any differences (i.e., the operator's preconditions are not met), Talus invokes means-ends analysis recursively, selecting an operator and (possibly) creating more subproblems. If the system finds a successful subplan, it 'mentally' applies the operator and again invokes means-ends analysis to solve the second subproblem. Of course, if Talus selects an inappropriate operator at any level, it may produce a nonoptimal subplan or fail to find one entirely. In the latter case, it backtracks and considers the next operator in the set.

However, Talus can also use plans like that in Figure 1 to constrain search on new problems. In such cases, the system retains access to primitive operators, but it also retrieves plans whose differences completely match against those in the current problem. In general, Talus heeds advice gleaned from such stored knowledge, since it prefers structures that are more specific (as well as ones that occur higher in a given plan). Nevertheless, the system uses this retrieved knowledge quite differently in its three modes, to which we now turn.

2.2 Using stored plans as macro-operators

As we noted earlier, in 'macro-operator' mode, Talus automatically reuses the components of a retrieved plan. For instance, suppose the system has the plan in Figure 1 stored in memory and encounters a problem in which the door between the two rooms is closed. In this case, Talus retrieves the stored plan `S1` and checks its weakest preconditions to see if it can be applied. Since the preconditions are not satisfied, the system recursively calls itself with the new problem of transforming the current state into one that satisfies the plan's weakest preconditions.

In solving this subproblem, Talus must construct a subplan from primitive operators that move the robot toward `door1`, opens `door1`, then move back next to `a`. The state that results from the subplan satisfies the preconditions of `S1` and the system carries it out. Note that the resulting plan is different from that depicted in Figure 1 in that the subplan described contains three primitive operators and is connected as an upward branch of `S3`. Also, the generated plan is not optimal in that the robot moves to `door1` and back before proceeding to `door1`. The system considers the plan to be acceptable despite its lack of optimality. However, if Talus had been unable to satisfy the preconditions of `S1`, the system would have discarded it and tried to solve the problem using only primitive operators.

2.3 Using stored plans as analogies

In 'analogical' or 'case-based' mode, Talus checks to determine whether subplans are relevant to the task at hand. Given the problem described above, this version of the system decides to apply the `move-to` operator stored at the top level of `S1`. However, before using `S2`, this incarnation checks to see if the preconditions of the primitive operator associated with `S1` (i.e., `move-to`) are satisfied. Since they are not, Talus decides to reuse `S2` and its associated operator `go-thru`, then considers subplan `S3` and decides to use it for the same reason. The preconditions for `S3`'s `move-to` match at this point, so the system uses that operator in its new plan.

As in macro-operator mode, this version finds that the closed door is a problem in that moving to the door does not fully satisfy the preconditions of `go-thru`. But, here Talus tries to repair the plan and retrieves the primitive operator `open` from memory based on the difference `(closed door1)`, preferring it to `S1`, `S2`, `S3`, and other operators. This addition to the subplan (shown with dotted lines in Figure 1) eliminates the remaining differences and lets Talus apply `go-thru` and `move-to`, giving a complete plan for the problem.

2.4 Using plans as search-control knowledge

In 'search control' mode, Talus does not bother to check plan components for relevance because it never reuses them. Given the above problem, the only information that the system borrows from plan `S1` is the associated operator `move-to`. Rather than considering the single component of this plan (`S2`) for reuse, it creates a subproblem from the missing preconditions of the operator and uses this to retrieve potentially useful plans and operators from memory. In this case, the retrieval module selects S2 as its first choice anyway, although this decision requires additional search through memory. As a result, Talus decides to apply go-thru at this level of its plan. This leads to another subproblem, and the system again uses this to retrieve potentially useful knowledge from memory.

This time the retrieval mechanism accesses four structures – the plan S1, the subplan S3, the operator move-to, and the operator open – that match the differences equally well and from which it picks randomly. If Talus selects S1, S3, or the first operator, matters proceed much as in the analogical case, with the system finding that move-to does not eliminate all differences, accessing memory to retrieve the open operator, and producing the plan shown in Figure 1. On the other hand, if Talus selects open instead, it finds this operator's preconditions unmet and creates another subproblem, leading it to retrieve move-to in response. The resulting plan differs in structure from that in Figure 1, being completely upward branching, although the order in which the two plans apply operators is the same.

3. The Behavior of Talus

The literature presents conflicting stories about which planning method is most desirable. Laird et al. (1986) argue for the superiority of search-control knowledge over macro-operators in terms of greater transfer, and Minton (1990b) favors the former because they let one interleave operators. Conversely, some arguments for case-based and analogical approaches center on the advantage of retrieving entire plans rather than constructing them from primitive operators. Claims in favor of macro-operators often assume that certain sequences of operators occur in solutions many times, either in individual problems or across problems.

Our intuitions suggest that each method is desirable in some domains but not others, and inspection of the alternative approaches suggests a more explicit claim.

Hypothesis 1. *Macro-operator methods are desirable in* regular *domains, search-control methods are desirable in* nonregular *domains, and analogical methods are desirable in domains of medium regularity.*

Highly regular domains are ones in which a small percentage of the possible problem types actually occur. The macro-operator approach should work well in such domains because the same problems occur over and over again, and these can be solved with a few macro-operators. In contrast, nonregular domains are ones in which most of the possible problems actually occur. The search-control approach should work well in such domains because it provides distributed knowledge that one can apply in a wide variety of situations. Of course, the claim assumes that plan knowledge is present in memory, and that this knowledge is relevant to the problems that arise.

This hypothesis is subject to experimental test, but such tests require a more formal definition of regularity. For any given problem space, we can enumerate all possible states, and we can define the set of possible problems as the set of ordered pairs of such states. Thus, given a problem space that contains s states, there exist $s(s-1)$ possible problems. However, due to symmetries, many problems may have isomorphic solutions. We will refer to each equivalence class of isomorphic tasks as a *problem type*. Furthermore, even if a problem space generates t problem types, only o of the types may actually occur in the domain. We will use this ratio o/t as our measure of the regularity of a planning domain.[2]

This definition lets us further instantiate our hypothesis, but it also suggests a second hypothesis involving the knowledge available to the planner.

Hypothesis 2. *Macro-operator methods are desirable in the presence of considerable plan knowledge, search-control methods are desirable when little knowledge is available, and analogical methods are desirable in the presence of medium levels of knowledge.*

Clearly, if memory contains plans for every possible problem, then macro-operators should fare quite well, since they can solve every problem without modification. Similarly, if memory contains only one or two plans, then search-control rules should do better, since even a few plans may contain knowledge about many operators that can be used in quite different problems. As with regularity, analogy should occupy the middle ground with respect to knowledge. Moreover, the effects of regularity and knowledge should interact. As one introduces more knowledge in nonregular domains, the difference between macro-operators and search control modes should decrease. Similarly, in the presence of enough knowledge, changes in regularity should produce few differences in behavior among planning modes.

Our two hypotheses seem plausible, and they are consistent with arguments in the literature, but intuitions can be misleading. Thus, it was important that we test our predictions about the interactions among planning method, domain regularity, and plan knowledge. To this end, we ran Talus under a variety of conditions and measured its performance in each situation.

We used the total number of matches as the measure of efficiency in our experiments. This represents the amount of work done by Talus' matcher, both in retrieving plans and operators from memory, and in checking their preconditions for applicability to the current state. More specifically, it measures the number of constants and variables the system tested for equality. Different calls to the matcher could cost different amounts, but the total number of matches provides a rough metric

[2] This formulation ignores the possibility of problems in which final states are partially specified. It also ignores the probability of each problem type, as well as regularities due to shared subproblems.

of planning effort. We also recorded a second measure – the percentage of test problems that TALUS solved within its computational limits (10,000 matches).

We selected the blocks world as the source of problems and plans for our domains, focusing on tasks involving four blocks. The problem space for the blocks world is highly symmetrical, producing many problems with isomorphic initial and final states (i.e., having the same problem type). We selected ten problem types from the four-block space, and used these to generate random test problems for each; we also generated, by hand, abstract means-ends traces that specified optimal solutions for each problem type. To control for problem complexity, all problem types and stored plans had optimal solutions involving six primitive operators.

In our initial study, we selected one of these abstract plans and stored it in TALUS' memory. We then ran the system on varying numbers of problem types that included ones corresponding to the stored plan, but that included other types as well. This let us systematically vary the regularity of the domain from $1/t$ through $10/t$, even though we did not know t, the exact number of possible problem types. Any particular problem type might be nonrepresentative; thus, we averaged over each of the ten plans, and over all possible sets of test cases that contained the problem solvable by the plan in memory. We ran TALUS on these test problems in each of its three modes, producing a 10×3 experimental design.

Figure 2 (a) shows the results of this experiment, which are mainly consistent with our first hypothesis. With one plan in memory, a decrease in regularity (the fraction of problem types present in the test set) increases the total match cost for all three methods, this effect being strongest for analogy. However, only two of the three predicted crossovers occur. At very high regularity, macro-operators appear to be most useful, but the difference between this mode and analogy is minimal. The results also bear out the prediction that search control is most useful in nonregular domains, with this method showing a distinct improvement over both macro-operators and analogy. Yet at no point on the graph does analogy outperform both macro-operators and search control. At least at this level of knowledge, the third crossover does not occur.

Figure 2 (b) shows the results of testing the three modes at a second level of knowledge. In this experimental condition, we ran the system with thirty of the possible ways in which five plans could exist in memory. Again, we averaged over all possible test sets containing these types for each of the three modes.[3] With five plans in memory, we see only one of the three expected crossovers. Although macro-operators are doing well when the domain is highly regular, analogy fares even better. The relationship between analogy and search control is also unexpected, though judging from the slopes of the curves, they might intersect if we examined even less regular domains. The predicted crossover of macro-operators and search control does occur, though at a later point than seen in Figure 2 (a).

The differences between the two graphs relate to our second hypothesis, which predicts that the efficiencies of each mode across several knowledge levels should also cross over. The curves in Figure 2 appear to show the beginnings of this effect. When TALUS has little knowledge in plan memory, search control is the most efficient method, but in the presence of more knowledge, analogy is the most efficient mode. To get further evidence for this crossover, we examined behavior under a third condition in which plan memory contained ten plans; in this case, analogy remained more efficient than search control, as predicted by the hypothesis. However, the experimental results violated our expectation that macro-operators would cross over both analogy and search control, with the former method being the least efficient across all levels of knowledge, once we factored out the influence of domain regularity.

We believe this unexpected behavior is due to the underspecified terms *little*, *medium*, and *considerable* found in the second hypothesis. Although the first two levels of knowledge seem to modeled by one and five plans, respectively, we suspect that ten plans may not represent a 'considerable' amount of plan knowledge. On reflection, we think that, to approximate high levels of knowledge, the number of plans in memory should more closely approach t, the total number of plan types that are possible in the domain. In future experiments, we anticipate that this revised metric for the amount of plan knowledge will produce all the crossovers predicted by our second hypothesis.

Our experiments with TALUS' three problem-solving strategies have borne out certain aspects of our hypotheses, but we could not demonstrate the conditions under which macro-operators are more efficient than both search control and analogy. However, in general we found that changes in regularity does affect the relative efficiency of the three modes, and that changes in the amount of plan knowledge memory also affects this measure. Although not detailed above, we also found a high correlation between match cost and our second metric, the percentage of solved problems. These results provide us with directions in which to refine and retest our hypotheses in future work.

4. Directions for Future Work

Our discussion has assumed that a single method is desirable throughout the domain, in that the same degree of regularity holds across all sets of problems. However, not all domains have this characteristic. One can imagine planning domains in which some problems occur frequently but others occur rarely, or in which all prob-

[3] We did not measure performance on test sets that did not contain these problem types because we wanted to ensure that all information in memory was relevant. Although the effect of irrelevant plans is potentially interesting issue, that topic is beyond the scope of this paper.

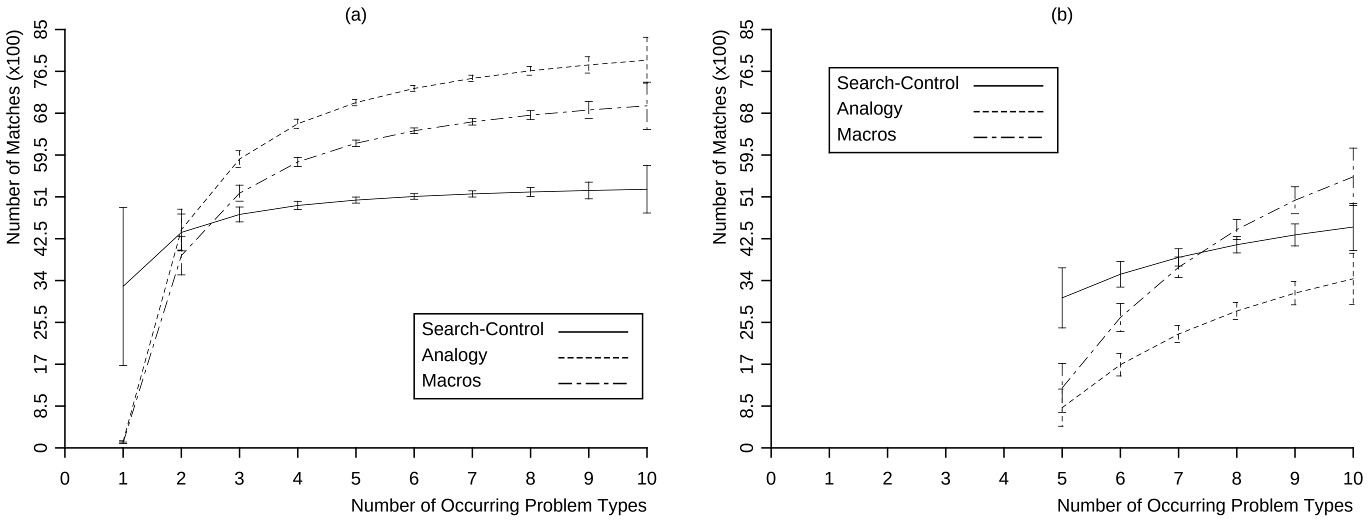

Figure 2: Behavior of TALUS in three planning modes – search control, analogy, and macro-operators – as a function of the number of problem types in the test set. Problem types came from the blocks world, and plan memory contained (a) one out of ten relevant plans, and (b) five out of ten relevant plans.

lems have the same high-level structure but have quite different subproblems. For such domains, one method will be desirable for one subset of problems, whereas another will be desirable for another subset. We suspect that many real-world domains also have this flavor. The notion of a mixed *domain* suggests the idea of a mixed *method*, which would invoke analogy in some contexts, macro-operators others, and search-control in others.

TALUS views analogical reuse, macro-operators, and search control as special cases of a general method. However, a simple extension to the framework allows for mixed methods. Rather than specifying whether one tests a plan component and, if not, whether one reuses it, the extended scheme associates a *probability* of reuse with each component. If a component's score is very high, one automatically reuses it without bothering to check for relevance. Similarly, if the probability is very low, one automatically avoids reuse without bothering to check. In contrast, if a plan component's score is neither high nor low, one inspects it for relevance, and reuse depends on the result of this test.

Such probabilistic information supports arbitrary mixtures of methods that are appropriate for arbitrary distributions of problems. However, it would be tedious and impractical to enter this information manually. In future work, we plan to automate this process using a simple scheme. One begins with a set of stored plans for a domain, each having a prior probability of one half; thus, the default behavior is pure analogical reuse. However, one updates this probability each time a plan contributes or fails to contribute to a successful plan. Over time, the probabilities for some components will approach one, so that they come to be reused automatically. The probabilities for others will approach zero, leading them to be ignored, and still others will have intermediate probabilities, and thus will still be tested for relevance. The result will be a mixed method that has been adapted to the distribution of problems for the domain at hand.

The TALUS implementation does not yet include this learning mechanism or the probabilistic control scheme on which it relies. However, we believe that a version extended in this way could automatically adapt to domains with quite different distributions of problems, and that we can use experiments with synthetic domains to show that its asymptotic behavior matches or outperforms analogical reuse, macro-operator techniques, or search-control methods. We also hope to show that it fares better than these pure methods on real-world planning tasks.

References

Fikes, R. E., Hart, P. E., & Nilsson, N. J. (1972). Learning and executing generalized robot plans. *Artificial Intelligence*, *3*, 251–288.

Hammond, K. J. (1990). Case-based planning: A framework for planning from experience. *Cognitive Science*, *14*, 385–443.

Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, *3*, 285–317.

Jones, R. (1989). *A model of retrieval in problem solving*. Doctoral dissertation, Department of Information & Computer Science, University of California, Irvine.

Kambhampati, S. (1990). Mapping and retrieval during plan reuse: A validation structure based approach. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 170–175). Boston, MA: MIT Press.

Korf, R. E. (1982). A program that learns to solve Rubik's Cube. *Proceedings of the National Conference on Artificial Intelligence* (pp. 164-167). Pittsburgh, PA: Morgan Kaufmann.

Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning*, *1*, 11–46.

Minton, S. N. (1990a). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, *42*, 363–391.

Minton, S. N. (1990b). Issues in the design of operator composition systems. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 304–312). Austin: Morgan Kaufmann.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice Hall.

Veloso, M. M., & Carbonell, J. G. (1989). Learning analogies by analogy: The closed loop of memory organization and problem solving. *Proceedings of the DARPA Workshop on Case-based Reasoning* (pp. 153–158). Pensacola Beach: Morgan Kaufmann.

Appendix A. Characterization of Work

There are many AI systems that address issues of plan reuse. However, differences in domains, methods, and assumptions make a comparison of these systems difficult. In this appendix, we consider the assumptions that Talus makes about its domain, its level of knowledge, and its problem-solving method. Recall that our intent is to determine properties of the domain and knowledge that make one strategy more desirable than another.

A.1 The Task

We designed Talus to assay the merits and shortcomings of three major techniques for domain-specific planning. The system accepts as input a start state and set of goal conditions, and it returns a totally ordered sequence of operator with a trace of its derivation. The generated plan need only solve the presented problem; it need not be optimal or even especially efficient.

Although Talus' three modes differ in important ways, they all support planning through means-ends analysis. Consequently, each mode uses information typically available to this method: a set of primitive Strips operators, a current state, and a set of differences (goals not satisfied by the current state). In addition, Talus can also use stored plans indexed by differences. The resulting system provides a good framework for evaluating the conditions under which each of the three strategies is desirable. Although our work has focused on the efficiency of plan generation, future experiments will examine other metrics like plan optimality.

A.2 The Approach

Each of Talus' strategies takes a different approach to creating plans. In search-control mode, the system uses knowledge in plan memory to decompose a current problem into subproblems. In macro-operator mode, Talus uses entire plan schemas (or subplan schemas), together with primitive operators, to build a solution to the new problem. In analogy mode, the system uses plan schemas as a template for a new solution, deleting unnecessary subplans and adding new subplans that were not needed in the stored plan.

Each of Talus' modes assumes that the environment is static, that primitive operators behave as advertised, and that any solution path is acceptable. The system also assumes that certain types of information are available in plan memory. Each strategy requires that the plans in memory contain information about the differences they resolve. They use this information to retrieve plans, subplans and primitive operators.

In addition, search-control mode requires that each plan include a primitive operator at its top level, and that it have access to this operators' preconditions and effects. Talus uses this information in a manner similar to Strips, testing preconditions to determine operator applicability and using the add/delete list to 'execute' the operator. The macro-operator mode adds the requirements that each plan specify it preconditions for successful application and the sequence of execution for the primitive operators in the plan. This information lets Talus treat each plan as though it were a primitive operator during means-ends analysis. Analogical mode assumes the presence of different information to that required by search control. Unlike macro-operator mode, analogical replay requires that each plan include its derivational structure, which lets the system adaptively reuse and modify each subplan included in the stored plan.

Currently, all plan knowledge is coded and added to memory by the programmer, and this must be done for each domain in which Talus operates. However, we feel that the above types of information are adequate for some real-world planning tasks, and we hope to address this claim in our future work.

A.3 The Overall Model

Despite the simplicity of the domains on which we have tested Talus, we feel that its design meets our research objectives, which involve identifying the conditions under which alternative planning methods are most desirable. The uniform nature in which the system encodes domain knowledge – as means-ends plans with associated derivational structures – let us carry out an experimental study of the three problem-solving methods and compare the results in a principled manner.

Talus is a 'rational' problem solver in that it always produces plans that fall within the deductive closure of its domain operators. Neither the resulting plans nor the process used to produce them are necessarily optimal, but our preliminary experiments suggest that both become more nearly optimal as one adds more plan knowledge to memory.