Planning strategy representation in DoLittle

Jacky Baltes
Department of Computer Science
Tamaki Campus
University of Auckland
Private Bag 92019

Telephone: +64 (9) 373-7599 Ext. 8744
Fax: +64 (9) 308-2377
Email: h.baltes@auckland.ac.nz

May 5, 1997

Keywords: Multi-strategy planning, macro-operators, abstraction, case-based planning.

Abstract

This paper describes multi-strategy planning and its implementation in the DoLittle system, which can combine many different planning strategies, including means-ends analysis, macro-based planning, abstraction-based planning (reduced and relaxed), and case-based planning on a single problem. *Planning strategies* are defined as methods to reduce the search space by exploiting some assumptions (so-called *planning biases*) about the problem domain. *General operators* are a generalization of standard Strips operators that conveniently represent many different planning strategies. The focus of this work is to develop a representation weak enough to represent a wide variety of different strategies, but still strong enough to emulate them. The search control method applies different general operators based on a strongest first principle; planning biases that are expected to lead to small search spaces are tried first. An empirical evaluation in three domains showed that multi-strategy planning performed significantly better than the best single strategy planners in some domains.

1 Introduction

The classical planning paradigm has long been an active research area in artificial intelligence. One reason for this popularity is that many practical problems can be interpreted as strategical planning problems, e.g., scheduling of machines in a factory, file system maintenance in an operating system, or cargo delivery.

This work aims at developing a strategical planning system for a kitchen robot. To simulate this environment, I developed a kitchen domain; it consists of an one-armed, mobile robot whose task it is to prepare different beverages. The kitchen domain is a complex domain. It contains 51 operators and 45 objects and has an average branching factor of 3.5. Plans contain many primitive operators, e.g., making a cup of tea, which is one of the simplest tasks, takes 30 steps.

Unfortunately, theoretical results show that planning is intractable even in simple domains [Byl91, Che91]. To be practical, a planner must therefore reduce the size of the search space. Based on the notion of an inductive bias in machine learning [Mit90], I introduce *planning bias* to describe these assumptions. Examples of planning biases include assumptions about the structure of the search space, the domain description, the plan structure, the problem set, and the order of the problems.

A *planning strategy* is any method that exploits some planning bias by removing, re-ordering, or restructuring part of the search space. The distinction between planning bias and planning strategy is important since (a) there may be different strategies for exploiting a bias, and (b) since different planning biases may lead to the same planning strategy. For example, if the designer of a domain assumes that earlier plans are subtasks of later ones (a problem order bias), she may either create macros to encapsulate earlier solutions or use a case-based system with a specific similarity metric. On the other hand, Iba's and Korf's macro-planners use the same planning strategy (macros), but are based on very different planning biases (peak-to-peak heuristic ([Iba89]) vs. serial operator decomposability ([Kor87]).

The following paragraphs give a brief introduction to some popular planning strategies and their underlying biases.

Means-ends analysis attempts to solve problems by looking at the differences between the current state and the goal state [NS63]. A difference is then selected and an operator is added to the plan to reduce this difference. Means-ends analysis works well in domains, where there are few operators to reduce a difference and the order in which the differences are reduced is not important.

Macro-based planning attempts to reduce the length of the solution by reasoning about sequences of operators instead of just primitive operators [FHN72, Iba89, Min85]. It is based on the assumption that a small number of fixed sequences of primitive operators occur frequently in solutions. For example, when moving an object from one room to another with a closed door, a robot has to execute the following subsequence: turn, put down an object, turn, open the door, turn, pick up the object, and turn to walk through the door. By adding an operator to encapsulate this subsequence, the reasoning process may be sped up through the reduction of the solution length. Macro-based planners must balance this reduction with the increase in the branching factor and the matchcost.

Abstraction-based planning first ignores low level details of the problem, to create an abstract solution [AF88, Kno91, Sac74]. The abstract solution is then refined by adding more and more detail until all abstract operators are replaced by primitive ones. For example, assume that the planner is given the problem of moving a robot from Auckland to Banff. An abstract planner would first create a plan with three abstract operators: (a) go from the current location to the Auckland airport, (b) fly from Auckland to Calgary, and (c) go from Calgary to Banff. A planning system that reasons about the primitive actions (e.g., go forward, turn right, turn left) will quickly be overcome by the tyranny of detail.

Case-based planning reuses previous solutions when solving new problems [Ham89]. Given a problem, a similar plan is retrieved and adapted to the new situation. The adaption methods are specific plan debugging routines. The assumption is that a similar plan can be found efficiently and that the cost of adapting an old solution is less than that of creating one from scratch. For example, in software design people often use somebody else's code and adapt it to their requirements.

2 Motivation

Planning systems based on particular planning strategies work well if the underlying assumptions are met, but fail (often spectacularly) if they are not. Many different planning strategies have been developed, but no single bias has been found to be superior or even sufficient in all domains. Recent research shows that planners must use different planning biases in different domains [SVB94].

The motivation for multi-strategy planning is that instead of developing a new planning strategy, the planner is based on partially successful, well known planning strategies. So far the work focused on macro-based, case-based, and abstraction-based planning. The problem is to determine when a given planning strategy is appropriate for a domain and solve the problem with it. Unfortunately, there are many examples in which a single planning strategy is not sufficient for a domain or even a single problem. Instead, some parts of the problem can be solved efficiently by a particular planning strategy, but another planning strategy is needed for the remainder. Therefore, a multi-strategy planning system must (a) break the problem up into subproblems, (b) select planning strategies and solve the subproblems, and (c) combine the solutions to the subproblems.

The remainder of this section is a brief example, to give the reader a feeling for the essence of multi-strategy planning. Therefore, the comparison is based solely on the necessary search depth and ignores other factors such as the branching factor. A completely worked example can be found in [Bal96].

Assume that in the kitchen domain, the goal is to prepare a cup of instant coffee with sugar. The solution to this problem contains 42 primitive operators.

Macro-based planning exploits often used operator sequences in the domain. Because of the large variety of possible location for the utensils in the kitchen domain, there are few long recurring operator sequences. The average length of the useful macros in the kitchen domain is about four operators. For example, the following macro fills a cup with water (put cup in sink, fill cup with water, turn water off, pick up cup). Given that macros only contain small number of operators, they can not reduce the search space sufficiently. For example, in this case, the search depth is still at least ten steps.

Abstraction-based planning creates an abstract plan to make instant coffee with sugar: (get a cup, fill cup with water, heat the water by either using the microwave or the stove, add instant coffee, get the sugar jar, add sugar). However, the refinement of each of those abstract operators is non-trivial in itself, and abstraction-based planning does not provide any guidance when searching for the refinement. For example, the refinement of the "heat-the-water" operator consists of ten primitive operators.

Case-based planning retrieves a plan (in this example, the plan for making tea is retrieved) and adapts it. This requires replacing the tea bag with instant coffee. However, the case-based planner has to create a suffix plan to get the sugar jar and open it, and then scoop the sugar into the coffee. This takes an additional 19 steps in the kitchen domain.

Multi-strategy planning uses all planning strategies to make instant coffee with sugar. It uses case-based planning to find an initial plan. However, it is able to employ other planning strategies when searching for the suffix plan to add sugar. An abstract plan to add the sugar is easily found: get the sugar jar and add the sugar. The refinement of these abstract operators is sped up by providing macros for often recurring subsequences, e.g., fetching a jar or opening a jar. In this case, the search depth is two steps only.

As can be seen in the previous example, no single problem solving strategy (macros, cases, abstractions) was able to solve the problem efficiently, but a combination had to be used.

3 Planning paradigm

This section develops a practical definition of planning strategy as a language for evolving plans and a set of plan transformations. This work uses the plan-space search paradigm, which was first introduced to analyze partial order planning [Cha87] but can be extended to include recent new planning paradigms [BF95, Gin96]. Planning is interpreted as search through evolving plans. In this framework, a planner is defined as follows:

Definition 1 (Planner) *A* ***planner*** $\mathcal{P}$ *is a tuple* $(\mathcal{L_S}, \mathcal{L_G}, \mathcal{L_O}, \mathcal{L_P}, \mathcal{T}, \mathcal{M})$.

- $\mathcal{L_S}$ *is the state language, describing possible world states.*
- $\mathcal{L_G}$ *is a goal description language.*
- $\mathcal{L_O}$ *is a description language for operators,*
- $\mathcal{L_P}$ *is the plan language, the language describing evolving plans. The plan language must be able to express plans, that is a set of operators, an ordering on the operators in the set, and a set of constraints on variable instantiations. Early planning systems supported only totally ordered, fully instantiated plans. More recently, partial-order planners support partially ordered plans with co- and non-codesignation constraints of variables.*
- T *is a set of plan transformations. A plan transformation* t *is a function that takes a plan expression from* $\mathcal{L_P}$*, and returns a new candidate plan* c *expressed in* $\mathcal{L_P}$*.*
- M *is the plan selection method. Given a set of possible plans* P *expressed in* $\mathcal{L_P}$*,* M *selects the plan* p *to be tested next.*

The state $\mathcal{L}_{\mathcal{S}}$, goal $\mathcal{L}_{\mathcal{G}}$, and operator $\mathcal{L}_{\mathcal{O}}$ languages determine the description of a domain, since they define the input to the planning system. As many other planning systems, this work uses a variant of the STRIPS representation. Therefore, the representational classification is ignored in the remainder of this paper.

The operational classification is determined by three components: the plan description language $\mathcal{L}_{\mathcal{P}}$, the set of plan transformations T, and the plan selection method M. Plan selection methods are usually derived from well-known search methods such as depth-first, breadth-first, or best-first.

The following subsections discuss some popular planning systems in the plan space search paradigm. Although there are many possible variations of the different strategies, the following discussion focuses on the most common implementations.

3.1 Forward chaining planning

Forward chaining planning is a simple planning system that applies operators until a goal state is found. The plan language represents a totally ordered and fully instantiated operator sequence. There is only a single plan transformation: append an operator to the operator sequence.

3.2 Means-ends analysis

Means-ends analysis contains two totally ordered and fully instantiated sets of operators, the plan head and the plan tail. The plan head contains operators that are already applied and the plan tail contains operators that still need to be applied. There are three plan transformations: (a) append an operator to the plan head, (b) prepend an operator to the plan tail, and (c) apply the first operator of the plan tail.

3.3 Macro-based planning

Macro-based planning is similar to means-ends analysis planning, but the plan language and the plan transformations are extended to include sequences of operators instead of single operators only.

3.4 Abstraction-based planning

Abstraction-based planning extends the plan language to allow for different levels of abstraction. The set of plan transformations from means-ends analysis are extended to include a transformation that adds the generation of a new subproblem space at the next lower abstraction level.

3.5 Case-based planning

Case-based planning uses a similar plan language to means-ends analysis, but a much larger set of plan transformations (insert, remove, reorder, replace, and instantiate an operator). The operator selection method is based on a similarity metric instead of relevance.

Note that this representation is not the only one possible. There are many other weak representations, for example, Gould's APS system uses a pattern weight representation [GL94]. The main focus of this work is a representation that is weak enough to cover a wide variety of strategies, yet strong enough to emulate them. For example, it may seem that the reorder and replace plan transformations in case-based planning are superfluous, since they can be achieved by sequences of insertion and removal of operators. From a representational point of view, operator insertion and removal are the only ones necessary, since any plan can be created with these two transformations alone. However, this neglects the fact that these transformations have specific conditions under which they are applied. For example, in Hammond's Chef planner [Ham89], an operator can only be replaced if it solves a missing precondition or unwanted side effect conflict. As Hammond showed, the power of Chef stems from the fact there is a small set of these transformations and applicability conditions that can solve most problems in the cooking domain.

4 Multi-strategy planning in DoLittle

This section discusses the design of DoLittle, a multi-strategy planner that can combine forward chaining, means-ends analysis, case-based, automatic subgoaling [RK91], abstraction-based, and macro-based planning.

Extending the analysis of different planning strategies in the plan space paradigm described in section 3, the plan language and the set of plan transformations necessary for a multi-strategy planning system can be determined. In particular, the plan language must be able to represent: (a) totally ordered operator sequences, (b) instantiated variables, (c) a plan skeleton, and (d) trees of problem spaces. The set of plan transformations must include (a) operator transformations (application, insertion, removal, reordering, replacement of an operator sequence), (b) changing a variable binding, and (c) the creation of different subproblem spaces for subproblems, serial subgoals, and abstract spaces.

The design of a multi-strategy planner requires three key components: (a) applicability conditions for planning strategies, (b) a representation for different planning strategies, and (c) a search strategy that emulates different planning strategies.

4.1 Applicability conditions

Previous work has shown that simply extending the set of plan transformations is not sufficient. Minton showed that the creation of macro-operators alone must be carefully controlled to avoid a decrease in performance due to the increase in branching factor [Min88]. The situation is even worth for a multi-strategy system that adds many more plan transformations. Therefore, a multi-strategy planning system must also provide powerful methods for specifying when a given plan transformation should be applied. There are many possible features of a planning process that may be useful in determining whether to apply a planning strategy, e.g., the current state, the goals the planner is trying to achieve, the problem space, the current subgoal hierarchy, the current operator and its binding, the results of the indexer, the set of rejected plans, and additional domain knowledge. Many of these features are planning strategy dependent. For example, there is no concept of a subgoal hierarchy (means-ends analysis) in case-based planning and vice versa for the results of the indexer. Therefore, DoLittle's applicability conditions are based on a common subset of these features: the current state and the set of goals the planner is trying to achieve (so-called open goals).

The language for DoLittle's applicability conditions supports conjunction, disjunction, and negation of preconditions and open goals. For example, DoLittle can specify that a given planning strategy should only be used when the current state contains either (on cup1 table) or (in cup1 microwave) and the planner is trying to achieve (in cup1 sink) but not when it is trying to achieve (on cup1 shelf). For more detail, the reader is encouraged to refer to [Bal96].

4.2 Representation of planning strategies

DoLittle uses general operators, a generalization of Strips operators, to represent applicability conditions and planning strategies. Associated with a general operator is a set of refinements. A refinement is a sequence of general or primitive operators that guarantees that the effects of the parent operator are achieved, but it may have additional pre-conditions and effects.

The following general operator is an example from the kitchen domain and illustrates the key features:

General operator example

GEN-PICK-UP-FROM-CUPBOARD
    Variables $OBJECT
    Preconds (ARM-EMPTY)
        (IS-AT ROBBY AT-TABLE)
        (IS-IN $OBJECT CUPBOARD)
    Open goals (HOLDING $OBJECT)
    Effects (HOLDING $OBJECT)
        (NOT (IS-IN $OBJECT CUPBOARD))
        (NOT (ARM-EMPTY))
    Refine. 1 PICK-UP-FROM-CUPBOARD($OBJECT)
    Refine. 2 ABSTRACT-SUBGOAL

The general operator GEN-PICK-UP-FROM-CUPBOARD can be used to pick up an object from the cupboard independent of whether the cupboard is open or not. The preconditions and open goals refer to the planner state, not the world state. The preconditions of the operator establish that the planning strategies described in the refinements are applicable, if the current world state matches them, i.e., the arm is empty, the robot is at the table, and $OBJECT is in the cupboard. Note that DOLITTLE will not subgoal on the preconditions in the planner state, only on those of the refinements. Furthermore, one goal that the planner is trying to achieve is (HOLDING $OBJECT). Adding another literal to the set of open goals results in a conjunction. To represent a disjunction, a new general operator must be created:

General operator example

GEN-PICK-UP-FROM-CUPBOARD-2
    Variables $OBJECT
    Context
    Preconds (ARM-EMPTY)
        (IS-AT ROBBY AT-TABLE)
        (IS-IN $OBJECT CUPBOARD)
    Open goals (NOT (IS-IN $OBJECT CUPBOARD))
    Effects (HOLDING $OBJECT)
        (NOT (IS-IN $OBJECT CUPBOARD))
        (NOT (ARM-EMPTY))

Refine. 1 PICK-UP-FROM-CUPBOARD($OBJECT)
Refine. 2 ABSTRACT-SUBGOAL

Given the two general operators, the strategies described in their common set of refinements are applicable if the planner is trying *either* to achieve (HOLDING $OBJECT) *or* to negate (IS-IN $OBJECT CUPBOARD).

General operator example (continued)

REFINE. 1: MACRO
Variables $OBJECT
Preconds same as parent plus
(IS-OPEN CUPBOARD)
Effects (HOLDING $OBJECT)
(IS-IN $OBJECT CUPBOARD)
(ARM-EMPTY)
Sequence PICK-UP-FROM-CUPBOARD($OBJECT)

The first refinement is of type MACRO and consists of the single primitive operator PICK-UP-FROM-CUPBOARD. This refinement has the additional precondition that the cupboard must be open when picking up the object. If DOLITTLE selects this refinement, but the cupboard is closed, it will subgoal and generate a plan to open the cupboard. Note that in practice DOLITTLE selects the refinement that best matches the situation and would return the second refinement, discussed below, if the cupboard is closed. The deletions of (IS-IN $OBJECT CUPBOARD) and (ARM-EMPTY) are side-effects of PICK-UP-FROM-CUPBOARD. The difference between a MACRO and CASE refinement is that only CASE refinements can be adapted by for example replacing operators. MACRO refinements can only be added to the plan.

General operator example (continued)

REFINE. 2: ABSTRACT SUBGOAL
Variables $OBJECT
Preconds same as parent plus
(NOT (IS-OPEN CUPBOARD))
Effects (IS-OPEN CUPBOARD)
(HOLDING $OBJECT)
(NOT (IS-IN $OBJECT CUPBOARD))
(NOT (ARM-EMPTY))
Sequence empty

The second refinement is a reduced abstraction refinement, which does not contain an operator sequence. It contains one additional precondition ((NOT (IS-OPEN CUPBOARD))) and one additional effect ((NOT (IS-OPEN CUPBOARD))). DOLITTLE recursively searches for a plan that achieves all effects of the refinement. However, since the search space is classified as an abstract search space, the planner is constrained to plans that *do not* change the values of the literals in the general operator. For example, while searching for the plan to pick up the cup, DOLITTLE will reject any plan that changes the position of the robot ((IS-AT ROBBY TABLE)) or picks up an object ((ARM-EMPTY)).

4.3 DoLittle's search control

The representation of different planning strategies is alone not sufficient for a multi-strategy planning system. The representation of cases and macros are very similar, both are sequences of operators. However, their effect on the search space is very different. Macros are simply selected and concatenated, whereas cases are selected based on a similarity metric and are adapted. The situation is similar to that of asking for the inherent meaning of a bit-pattern (e.g., 11101010), which of course depends on whether it is interpreted as a binary number (signed or unsigned?), a machine code instruction (for which processor?), a character string, or a floating point number. Therefore, a multi-strategy planning system must also provide a search control method that emulates the effect of a given planning strategy on the search space.

DOLITTLE's search control method is based on a strongest-first heuristic: planning strategies that result in the smallest search space are tried first. Checking a small search space first has two benefits: (a) if a solution exists in this space, it can be found quickly, and (b) if no solution exists, the failure can quickly be recognized. Note that since planning strategies are not complete, they may remove the part of the search space with the solution.

DOLITTLE uses a domain independent similarity measure to retrieve the most similar general operator to the current problem and to select the refinements. If the operator or refinement represents a macro that exactly matches a problem, a solution is found (macro-based planning). Otherwise, the operator sequence is adapted to the current situation (case-based). If there is no case or macro available, a subproblem search space is created. At present, there are three different types of search spaces in DOLITTLE: an abstract subgoal space, a serial subgoal space, and a general subgoal. The different search spaces represent different constraints on the search.

5 Evaluation

This section discusses briefly the results of an evaluation of DoLittle's performance on a set of problem domains. For a detailed description of the methodology and the statistical analysis, please refer to [Bal96]. The evaluation included three domains: (a) the blocks-world, (b) the towers of Hanoi, and (c) the kitchen domain.

The goal of this evaluation was to evaluate empirically the performance of multi-strategy planning (DoLittle) against that of four single strategy planners: a means-ends analysis planner, a case-based planner, a macro-based planner, and an abstraction-based planner. The performance of DoLittle was also compared against that of a problem coordinated multi-strategy planner with an oracle (PC-MSP-O), that is a planner that selects from a set of possible planners the best one for a given problem. Although in practice, selecting the best strategy is impossible a priori, the entries for the hypothetical PC-MSP-O planner were generated by selecting the minimum of the single strategy planners. However, in contrast to DoLittle, the planning strategy is fixed for a problem, that is, PC-MSP-O is not able to switch between planning strategies when solving a single problem.

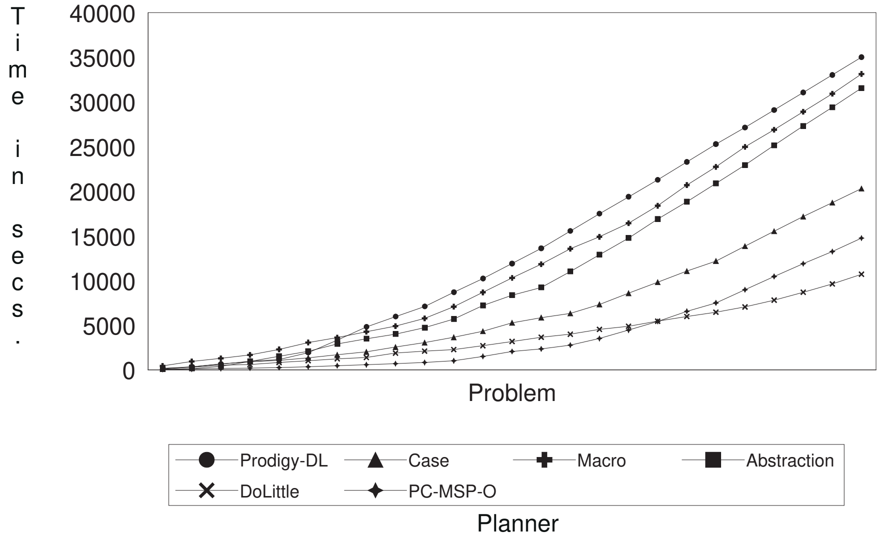
For the evaluation, a set of 150 training problems were randomly generated and the different planners were trained on these problems. Then a new set of 250 test problems were generated and the performance of the different planners was tested. Solving problems in the kitchen domain is difficult. Therefore, the solution length of the problems in the test and training set was slowly increased by increasing the number of drinks and ingredients in a plan. The idea is that by solving small problems first, the planners are able to learn enough information to solve more complex problems later.

The comparison included the cumulative number of nodes generated and the cumulative running times for the different planners as well as the relationship between the time limit and the total running time of the system. Figure 1 shows the cumulative running time in the kitchen domain.

The single strategy planners did improve performance with case-based planning being the best single strategy planner in this domain. In the kitchen domain, variations of earlier problems are often subtasks in later ones, which favors a case-based approach. Multi-strategy planning provided a significant higher speed up than single strategy planning. DoLittle performed better on the more difficult problems than PC-MSP-O. This difference statistically significant, which shows that especially in complex domains, a multi-strategy planner should be able to switch planning strategies while solving a problem.

The empirical evaluation also showed that no single strategy planner was superior. For example, although case-based planning lead to the biggest

Figure 1: Cumulative running times in the kitchen domain



improvement in the kitchen domain, it performed worth than abstraction-based planning in the towers of Hanoi domain.

6 Conclusions

So far, the main focus of the research has been on methods for combining different planning strategies on a single problem. Current work investigates the interaction of different planning strategies.

The uniform representation in DoLittle also makes it an ideal test-bed for the comparison of different planning strategies. This work may result in a better understanding of the applicability conditions of different planning strategies. This will lead to better methods of determining when a given planning strategy is appropriate for a domain.

Development on DoLittle is continuing at the University of Auckland. Currently, we are working on a project that uses DoLittle as the strategic planning component of an autonomous, mobile robot. The intended applications for the robot are mail delivery in an office environment, and other household and security tasks.

References

[AF88] John S. Anderson and Arthur M. Farley. Plan abstraction based on operator generalization. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 100–104, Saint Paul, Minnesota, 1988.

[Bal96] Jacky Baltes. *DoLittle: a learning multi-strategy planning system.* PhD thesis, University of Calgary, June 1996.

[BF95] A. Blum and M. Furst. Fast planning through plan-graph analysis. In *Proceedings IJCAI-95*, 1995.

[Byl91] Tom Bylander. Complexity results for planning. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 274–279, Sydney, 1991.

[Cha87] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

[Che91] Stephen V. Chenoweth. On the np-hardness of blocks world. In *Proceedings Ninth National Conference on Artificial Intelligence*, volume 2, pages 623–628, Stenlo Park, July 1991. AAAI Press/The MIT Press.

[FHN72] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(2), 1972.

[Gin96] Matt Ginsberg. A new algorithm for generative planning. In *Proceedings KR-96*, 1996.

[GL94] Jeffrey Gould and Robert Levinson. Experience-based adaptive search. In Ryszard Michalski and Gheorghe Tecuci, editors, *Machine Learning: A multi-strategy approach*, volume 4, pages 579–603, San Francisco, Ca, 1994. Morgan Kaufmann Publishers.

[Ham89] Kristian J. Hammond. *Case Based Planning.* Academic Press Inc., 1989.

[Iba89] Glenn A. Iba. A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3:285–318, 1989.

[Kno91] Craig A. Knoblock. *Automatically Generating Abstractions for Problem Solving.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1991.

[Kor87] R. E. Korf. Planning as search: A quantitative approach. *Artificial Intelligence*, 33(1):65–88, 1987.

[Min85] Steven Minton. Selectively generalizing plans for problem solving. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. MIT Press, 1985.

[Min88] Steven Minton. *Learning Search Control Knowledge: An Explanation-based Approach*. Kluwer Academic Publishers, Boston, 1988.

[Mit90] Tom Mitchell. The need for biases in learning generalizations. In J. Shavlik and T. Dietterich, editors, *Readings in Machine Learning*, pages 184–191. Morgan Kaufmann, 1990.

[NS63] Alan Newell and Herbert Simon. Gps: A program that simulates human thought. In E. Feigenbaum and J. Feldman, editors, *Computers and thought*, pages 279–298, New York, 1963. McGraw Hill.

[RK91] David Ruby and Dennis Kibler. Steppingstone: An empirical and analytical evaluation. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 527–532, Menlo Park, July 1991. AAAI, AAAI Press/The MIT Press.

[Sac74] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[SVB94] Peter Stone, Manuela Veloso, and Jim Blythe. The need for different domain-independent heuristics. In Kristian Hammond, editor, *Proceedings of the second international conference on artificial intelligence planning systems*, pages 164–169, Menlo Park, 1994. AAAI Press.