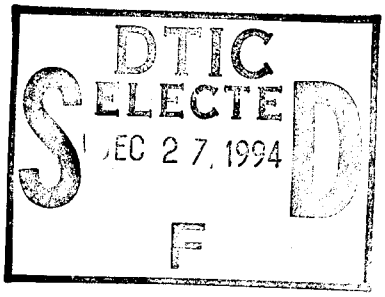
MULTI-METHOD PLANNING

by

Soowon Lee
USC/Information Sciences Institute
4676 Admiralty Way
Marina del Rey, CA 90292-6695



ISI/RR-94-390

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

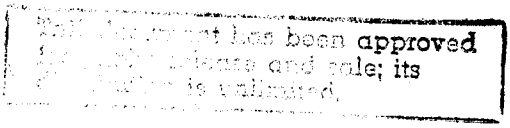
UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(Computer Science)



April 1994

REPORT DOCUMENTATION PAGE

FORM APPROVED
OMB NO. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1994	Research Report

4. TITLE AND SUBTITLE
Multi-Method Planning

5. FUNDING NUMBERS
N00014-92-K-2015

6. AUTHOR(S)
Soowon Lee

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
USC INFORMATION SCIENCES INSTITUTE
4676 ADMIRALTY WAY
MARINA DEL REY, CA 90292-6695

8. PERFORMING ORGANIZATON REPORT NUMBER
RR-94-390

9. SPONSORING/MONITORING AGENCY NAMES(S) AND ADDRESS(ES)
ARPA
3701 Fairfax Drive
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12A. DISTRIBUTION/AVAILABILITY STATEMENT
UNCLASSIFIED/UNLIMITED

12B. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

The ability to find a low execution-cost plan efficiently over a wide domain of applicability is the core of domain-independent planning systems. The approach investigated here to building such a planning system begins with two hypotheses: (1) no single method will satisfy both sufficiency and efficiency for all situations; and (2) multi-method planning can outperform single-method planning in terms of sufficiency and efficiency. To evaluate these hypotheses, a set of single-method planners for the domains investigated show that these planners have trouble performing efficiently over a wide range of problems.

As an alternative to single-method planning, *multi-method planning* is investigated in this thesis. A multi-method planner consists of a coordinated set of methods which have different performance and scope. Given a set of created methods, the key issue in multi-method planning is how to coordinate individual methods in an efficient manner so that the multi-method planner can have high performance. The multi-method framework presented here provides one way to do this based on the notion of bias-relaxation. In a bias-relaxation multi-method planner, planning starts by trying highly restricted and efficient methods, and then successively relaxes restrictions until a sufficient method is found.

A class of bias-relaxation multi-method planners has been developed. These planners vary in the granularity at which individual methods are selected and used. Depending on the granularity of method switching, two variations on strongly monotonic multi-method planners are implemented: coarse-grained multi-method planners, where methods are switched on a problem basis; and finegrained multi-method planners, where methods are switched on a problem-problem basis; and fine grained multi-method planners, where methods are switched on a goal-by-goal basis.

The experimental results indicate that, at least for the domains investigated, both coarse-grained and fine-grained multi-method planning can reduce plan length significantly compared with single-methodplaning, and fine-grained planning can improve the planning time significantly compared with coarse-grained and single-method planninge. Applications to simulated agent domain also shows one way that multi-method planning can be used in more complex domains.

14. SUBJECT TERMS
Planning, Bias, Multi-method, Monotonicity, Bias-relaxation, Granularity, Learning

15. NUMBER OF PAGES
140

16. PRICE CODE

17. SECURITY CLASSIFICTION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reoprts. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to stay within the lines to meet optical scanning requirements.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month,a nd year, if available (e.g. 1 jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element numbers(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	PR - Project
G - Grant	TA - Task
PE - Program Element	WU - Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the repor.

Block 9. Sponsoring/Monitoring Agency Names(s) and Address(es). Self-explanatory

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of ...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."
DOE - See authorities.
NASA - See Handbook NHB 2200.2.
NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - Leave blank.
DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.
NASA - Leave blank.
NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (NTIS only).

Blocks 17.-19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contins classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

UNIVERSITY OF SOUTHERN CALIFORNIA
THE GRADUATE SCHOOL
UNIVERSITY PARK
LOS ANGELES, CALIFORNIA 90007

This dissertation, written by

SOOWON LEE

under the direction of his Dissertation Committee, and approved by all its members, has been presented to and accepted by The Graduate School, in partial fulfillment of requirements for the degree of

DOCTOR OF PHILOSOPHY

Barbara Solomon

Dean of Graduate Studies

Date April 19, 1994

DISSERTATION COMMITTEE

Paul S. Rosenbloom

Chairperson

Dedication

To My Parents

Acknowledgements

First, I would like to thank Paul Rosenbloom, my advisor, for his superb guidance, encouragement, and support throughout the course of my research. He always provided me with insightful suggestions and invaluable comments on both the work and the draft.

I would like to thank Shankar Rajamoney, Craig Knoblock, and Behrokh Khoshnevis, the other members of my dissertation committee, for many helpful suggestions, criticism and advice on the thesis. Craig's comments on an early draft were particularly helpful. I would also like to thank Ari Requicha and Dan Moldovan for serving on my guidance committee and providing a lot of useful feedback.

I would like to thank the members of ISI Soar group, including Jose-Luis Ambite-Molina, Bonghan Cho, Randy Hill, Lewis Johnson, Jihie Kim, Hidemi Ogasawara, Miran Park, Karl Schwamb, Ben Smith, Iain Stobie, and Milind Tambe for interesting discussions and helpful suggestions on this research. I am also grateful to Allen Newell, John Laird, and members of the Soar group for their comments and encouragement.

I would like to thank Mike Barley, Haym Hirsh, Pat Langley, and Steve Minton for their helpful comments on my work.

I would like to thank my friends at USC, including Sanghoi Koo, Sungbok Kim, and Daeyeon Park who have made these past years so enjoyable.

I would like to thank my parents and all others in my family for their constant love, encouragement, and support. Finally, I thank my wife, Yoolim, for her love and understanding during the long course leading to this dissertation.

This research has been sponsored by the Defense Advanced Research Projects Agency and the Office of Naval Research under contract number N00014-89-K-0155, and under subcontract to the University of Southern California Information Sciences Institute from the University of Michigan, as part of contract N00014-92-K-2015 from the Advanced Research Projects Agency and the Naval Research Laboratory.

Contents

Appendix B

List Of Figures

List Of Tables

Abstract

The ability to find a low execution-cost plan efficiently over a wide domain of applicability is the core of domain-independent planning systems. The approach investigated here to building such a planning system begins with two hypotheses: (1) no single method will satisfy both sufficiency and efficiency for all situations; and (2) multi-method planning can outperform single-method planning in terms of sufficiency and efficiency. To evaluate these hypotheses, a set of single-method planners has been constructed. The results obtained from the experiments with these planners for the domains investigated show that these planners have trouble performing efficiently over a wide range of problems.

As an alternative to single-method planning, *multi-method planning* is investigated in this thesis. A multi-method planner consists of a coordinated set of methods which have different performance and scope. Given a set of created methods, the key issue in multi-method planning is how to coordinate individual methods in an efficient manner so that the multi-method planner can have high performance. The multi-method planning framework presented here provides one way to do this based on the notion of *bias-relaxation.* In a bias-relaxation multi-method planner, planning starts by trying highly restricted and efficient methods, and then successively relaxes restrictions until a sufficient method is found.

A class of bias-relaxation multi-method planners has been developed. These planners vary in the granularity at which individual methods are selected and used. Depending on the granularity of method switching, two variations on strongly monotonic multi-method planners are implemented: coarse-grained multi-method

planners, where methods are switched on a problem-by-problem basis; and fine-grained multi-method planners, where methods are switched on a goal-by-goal basis.

The experimental results indicate that, at least for the domains investigated, both coarse-grained and fine-grained multi-method planning can reduce plan length significantly compared with single-method planning, and fine-grained planning can improve the planning time significantly compared with coarse-grained and single-method planning. Application to a simulated agent domain also shows one way that multi-method planning can be used in more complex domains.

Chapter 1

Introduction

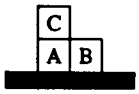
Research in domain-independent planning systems has been a main theme in the area of AI planning. These systems vary according to the way in which the search space is defined and traversed, the way in which plans are represented, the way in which goal interactions are dealt with, the way in which time and resources are handled, the way in which planning interacts with execution, and so on [Allen *et al.*, 1990]. Among the criteria used to evaluate these systems, three typical ones are the amount of time required to find the plan; the execution cost of the plan itself; and the ability to find some plan, or an optimal plan, for any problem in an arbitrary domain. Thus, finding a low execution-cost plan efficiently over a wide domain of applicability is the core of domain-independent planning systems. The key issue here in building such a system is how to construct a single planning method, or a coordinated set of different planning methods.

The hypotheses underlying this research are (1) no single method will satisfy both sufficiency and efficiency for all situations; and (2) multi-method planning can outperform single-method planning in terms of sufficiency and efficiency. The first hypothesis is based on the observation that most conventional planning systems which encode planning behaviors within a single fixed method — such as linear planning, nonlinear planning, abstraction, and so on — have a limitation in performing efficiently over a wide range of problems.

For example, STRIPS-type planners can generate plans quite efficiently for some problems by using the linearity assumption [Fikes and Nilsson, 1971]. With this

Initial State

(on C A)
(on A Table)
(on B Table)



Goal

G1: (on A B)
G2: (on B C)

Operators

(MOVE-ONTO-TABLE <x>):

- *Putdown block <x> onto table.*

Preconditions:
(on <x> <z>)
(clear <x>)
Add lists:
(on <x> Table)
(clear <z>)
Delete lists:
(on <x> <z>)

(MOVE-ONTO-BLOCK <x> <y>):

- *Stack block <x> onto block <y>.*

Preconditions:
(on <x> <z>)
(clear <x>)
(clear <y>)
(type <y> Block)
Add lists:
(on <x> <y>)
(clear <z>)
Delete lists:
(on <x> <z>)
(clear <y>)

Figure 1.1: The Sussman's anomaly in the blocks-world domain.

assumption, the number of goal conjuncts considered at each planning step can be reduced, so that planning time can be saved. However, this assumption makes the planners unable to generate an optimal plan in certain domains, and fail to find a plan in domains with irreversible operators. Sussman's anomaly in the blocks-world domain is a classical problem where an optimal solution cannot be found by a linear planner [Sussman, 1973]. Figure 1.1 shows the initial state, goal conjuncts, and operators for this problem.[1] Since a linear planner does not consider the other goal conjuncts until the current goal conjunct is achieved, both goal orderings — `(on A B)` followed by `(on B C)`, or `(on B C)` followed by `(on A B)` — generate non-optimal operator sequences.

[1]Throughout this thesis, variables in operators are denoted by angle brackets, as in <a>.

Initial State

(at O1 LocA)
(at O2 LocA)
(at Rocket LocA)

Goal

G1: (at O1 LocB)
G2: (at O2 LocB)

Operators

(LOAD <obj>)

- Load <obj> into Rocket.

Preconditions:
(at <obj> <loc>)
(at Rocket <loc>)
Add lists:
(inside <obj> Rocket)
Delete lists:
(at <obj> <loc>)

(UNLOAD <obj>)

- Unload <obj> from Rocket.

Preconditions:
(inside <obj> Rocket)
(at Rocket <loc>)
Add lists:
(at <obj> <loc>)
Delete lists:
(inside <obj> Rocket)

(MOVE-ROCKET)

- Move Rocket from LocA to LocB.

Preconditions:
(at Rocket LocA)
Add lists:
(at Rocket LocB)
Delete lists:
(at Rocket LocA)

Figure 1.2: A problem in the one-way-rocket domain.

A more serious problem occurs in domains with irreversible operators. Figure 1.2 shows a problem in the one-way-rocket domain which cannot be solved by a linear planner [Veloso, 1989]. In this problem, achieving either goal conjunct individually inhibits achieving the other goal conjunct. For example, after achieving the first goal conjunct (at O1 LocB) by applying (LOAD O1) → (MOVE-ROCKET) → (UNLOAD O1), the second goal conjunct (at O2 LocB) cannot be achieved because the Rocket cannot return to pick up the remaining object.

Nonlinear planners can generate optimal plans for these problems because they are free from the linearity assumption.[2] However, for other problems that could be solved by a linear planner, nonlinear planners may be less efficient than linear planners. For example, a nonlinear planner which uses a goal set as opposed to a goal stack, such as NOLIMIT [Veloso, 1989], has more choices to consider at each goal-selection point. This allows an optimal plan to be generated for a given problem; however, the overall planning performance may be decreased by the increased branching factor.

It has been known that partial-order planners can efficiently solve problems in which the specific order of the plan steps is critical [Sacerdoti, 1975, Tate, 1977, Chapman, 1987, McAllester and Rosenblitt, 1991, Barrett and Weld, 1992]. This is done by delaying step-ordering decisions as long as possible, so that the size of the plan space can be smaller than those of total-order planners. However, they pay the cost of having a more complex ordering procedure [Minton *et al.*, 1991].

For example, the partial-order planner SNLP [McAllester and Rosenblitt, 1991, Barrett and Weld, 1992], detects a threat between a step and a causal link whenever a new step or causal link is added. The ordering procedure searches over the space of ordering constraints to resolve the detected threat. This scheme can be quite effective if there are many threats in a problem. However, if there are only a few trivially-resolvable threats in a problem, it is generally less efficient to use such a complex threat-detecting and resolving algorithm for the entire problem.

Figure 1.3 illustrates the scope and performance for a hypothetical set of single-method planners. The inherent trade-off between a planner's scope and its performance, as shown in the figure, suggests that single-method planning has a limitation in performing efficiently over a wide range of problems, so that a more flexible planning approach is needed.

[2]The term "nonlinear" in this context implies that it is allowable to interleave operators in service of different goal conjuncts. It does not necessarily mean that either partial-order or least-commitment planning are used.

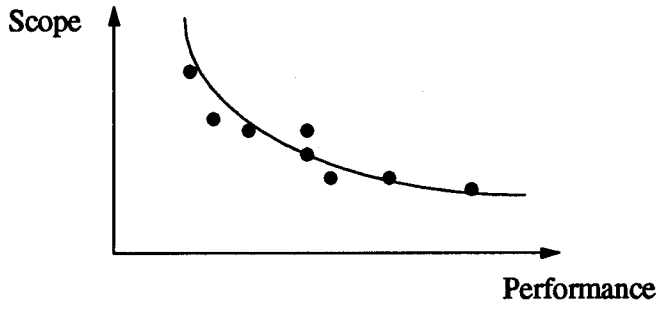


Figure 1.3: Hypothetical trade-off between single-method planners' scope and performance.

1.1 Overview of the Approach

As an alternative to single-method planning, *multi-method planning* is investigated in this thesis. A multi-method planner is an integrated system which utilizes a coordinated set of methods, where each method has different scope and performance [Lee and Rosenbloom, 1992, Lee and Rosenbloom, 1993]. The focus in this thesis is on multi-method planning in a single serial environment.[3] Figure 1.4 shows an example of a multi-method planner in which two different methods — linear planning and nonlinear planning methods — are coordinated sequentially in a single serial environment. In this planner, the linear method has better overall performance than the nonlinear method, while the nonlinear method can solve more problems than the linear method. Given a problem, the linear method is tried first to solve that problem. If it cannot solve the problem, the nonlinear method is tried.

The potential advantage of multi-method planning over single-method planning is that multi-method planning can achieve both applicability and efficiency at the same time. Theoretically, the scope of a multi-method planner can be the union of

[3]In a multi-agent environment, multi-method planning can be accomplished by running the methods in parallel until the problem is solved via one of the methods [Bond and Gasser, 1988]. However, detailed discussion on this issue is beyond the scope of this thesis.

Figure 1.4: An example of a multi-method planner.

the scopes of all individual single-method planners in the multi-method planner. Thus, a multi-method planner is at least as applicable as the most general single-method planner within the multi-method planner. If a single-method planner is complete for a domain — that is, it can solve all problems in a domain — then any multi-method planner which includes the single-method planner is also complete. With respect to efficiency, if a multi-method planner includes a method which is very efficient for some classes of problems, and that method can be selected for those classes of problems without too much extra effort, then multi-method planning can have an overall efficiency gain over single-method planning.

With this potential advantage of multi-method planning in hand, the ideal multi-method planner would be able to solve each problem with the most efficient method that is sufficient to solve it. In general, however, it is not known a priori which method is the most appropriate one for a given problem. The best way to approach this ideal is to learn about which methods to use for which classes of problems from a training set of problems. This type of method learning can be accomplished by either an analytical approach or an empirical approach.

The analytical approach to learning is based on reasoning about why the given training problem is solved (or cannot be solved) by the current method. If the problem is solved by the method, one constructs an explanation which proves that the problem is a positive instance of the goal concept 'solved'. Then, this

explanation is generalized and positive control knowledge is learned which selects the method for later similar problems. In contrast, if the method fails to solve the problem, one constructs an explanation which proves that the problem is a positive instance of the goal concept 'unsolvable'. In this case, negative control knowledge is learned which avoids the method for later similar problems.

The empirical approach to learning is based on the performance of those methods for a training set of problems. Instead of learning control knowledge by analyzing a solution trace for each problem, this approach extracts the information needed to select an appropriate method or to avoid a set of inappropriate methods for the set of problems under a fixed distribution. Since the extracted information is a function of the problem distribution, this approach can be used flexibly for other problem distributions or other domains. The multi-method planning framework investigated in this thesis is based on the empirical approach; however, the analytical approach will also be discussed later in more detail.

Within the empirical multi-method planning framework, the main goal of this research is to create a set of multi-method planners which are more efficient and applicable than single-method planners. Towards this end, the basic issues to be investigated are: (1) how to create individual methods which have different performance and scope so that the created multi-method planner can have both highly efficient methods and highly applicable methods; and (2) how to coordinate the created methods in an efficient manner so that the multi-method planner can have high performance. Each of these issues is discussed in turn in the following subsections.

1.1.1 Method Creation

In order for a multi-method planner to satisfy both efficiency and applicability, the single methods in the multi-method planner should range from highly efficient methods to a complete method. For this purpose, a methodology to create a set of methods with different performance and scope is developed. This methodology is based on the notion of *bias* in planning [Rosenbloom *et al.*, 1993]. Bias in planning

is any basis for choosing one plan over another other than plan correctness. With the view of planning as search over a space of plans [Korf, 1987], a bias is a restriction over the space of plans considered that determines which portion of the entire plan space can or will be the output of the planning process. For example, a linearity bias eliminates plans in which operators for different goal conjuncts are interleaved.

In general, a bias can potentially reduce computational effort by reducing the number of plans that must be examined, and it can potentially generate shorter plans by avoiding plans containing inefficient operator sequences. However, this is not always the case. For example, if the space eliminated by a bias is not large enough or the eliminated space does not include a sufficient number of inefficient plans, the bias has no effect. Whether or not these cases happen relies on the domain characteristics. Thus, it is important to devise biases which are really effective for a given domain in terms of performance improvement.

For a training set of problems in a given domain, a bias is called *effective*, if the average planning effort for the biased method over the training problem set is less than the average planning effort for the method that does not use that bias, and the average length of plans generated from the biased method over the training problem set is less than the average length of the plans generated from the method which does not use that bias.

By varying the effective biases, a set of methods with different performance and scope can be created. Given a set of effective biases, the most restricted method — which uses all of the biases — is the most efficient one, but can be incomplete if the desired plans are eliminated. On the other hand, the least restricted method — which uses no bias — is the least efficient one, but can be a complete method since no plans are eliminated.

1.1.2 Method Coordination

Once a set of methods with different performance and scope is created, these methods need to be coordinated efficiently so that the created multi-method planner can

satisfy both efficiency and applicability. Method coordination, as used here, refers to (1) the selection of appropriate methods as situations arise, and (2) the granularity of method switching as the situational demands shift.

Method selection: For method selection, individual methods need to be organized so that a higher level control structure can determine which method to use first and which method to use next if the current method fails. Two straightforward ways of organizing individual methods in a multi-method planner are *sequential* and *time-shared.* A sequential multi-method planner consists of a sequence of single-method planners. A time-shared multi-method planner consists of a set of single-method planners in which each method is active in turn for a given time slice [Barley, 1991]. In either approach, the key issue is how to reduce the effort wasted in using inappropriate methods.

The wasted effort in a sequential multi-method planner is the cost of trying inappropriate earlier methods in the sequence, whereas the wasted effort in a time-shared multi-method planner is the cost of trying all methods in the method set except the one that actually solves the problem. The wasted effort in sequential multi-method planning is sensitive to the ordering of the methods because it takes too much time if inappropriate earlier methods are not efficient enough, or in an extreme case, it may not be able to generate a plan at all if one of the inappropriate earlier methods does not halt. On the other hand, the wasted effort in time-shared multi-method planning is sensitive to the number of individual methods. Also, time-shared multi-method planning switches among methods more often than sequential multi-method planning, and it has more overhead for context switching.

The planning approach primarily investigated in this thesis is a special type of sequential multi-method planning, called *monotonic multi-method planning* [Lee and Rosenbloom, 1992]. In a monotonic multi-method planner, individual methods are sequenced so that the earlier methods are more efficient and have less coverage than the later methods. Compared with the single-method approach with planner completeness and the time-shared multi-method approach, the monotonic multi-method approach can potentially generate plans more efficiently. The idea is that

if the biases used in efficient methods can prune the search space, the problems solvable by efficient methods should be solved more quickly, while problems requiring less biases should not waste too much extra time trying out the insufficient early methods. In this way, a monotonic multi-method planner can retain planner completeness by allowing the least restricted method to be used, while it can generate low cost plans efficiently by using more restricted methods.

One way to construct a monotonic multi-method planner is to use the biases which themselves increase efficiency. Individual methods are sequenced so that the set of biases used in a method is a subset of the biases used in earlier methods, and the later methods have more coverage than the earlier methods. This means that planning starts by trying highly efficient methods, and then successively relaxing biases until a sufficient method is found. This type of planning is called *bias-relaxation multi-method planning.* A bias relaxation multi-method planner is not necessarily a monotonic multi-method planner if there are interactions among biases. However, one can generate monotonic multi-method planners via bias-relaxation by just testing whether monotonicity holds for the created bias-relaxation multi-method planners. In bias-relaxation multi-method planning, each bias is evaluated independently by comparing a method which uses that bias only and a method which uses no bias. Thus, bias-relaxation multi-method planning has a restricted scope in creating and comparing individual methods.

Granularity of method switching: The second issue of method coordination is the granularity at which individual methods are switched [Lee and Rosenbloom, 1993]. This issue is important in terms of a planner's performance, because the performance of a multi-method planner can be changed according to the granularity of shifting control from method to method. The family of multi-method planning systems can be viewed on a granularity spectrum. At one extreme there is the normal single-method approach, where one method is selected ahead of time for the entire set of problems. At another point of this spectrum are *coarse-grained multi-method planners*, where methods are switched for a whole problem when no solution can be found within the current method. Toward the other extreme, there

are *fine-grained multi-method planners*, where methods are switched at any point during a problem at which a new set of subgoals is formulated. Time-shared multi-method planners, where methods are switched based on the time slice, also can be viewed on the spectrum.

There is a trade-off between coarse-grained multi-method planning and fine-grained multi-method planning. A coarse-grained multi-method planner examines all paths within the current biased space until a solution is found or all paths are exhausted. Thus, a coarse-grained multi-method planner finds a solution within the first method that has one at the cost of searching the entire biased space in the worst case (unless some form of within-method learning or heuristics are used to prune out some portions of the space, or unless the time limit is exceeded). On the other hand, a fine-grained multi-method planner falls back on the next method whenever the partial plan for the current solution path cannot be expanded without violating the biases used in the current method. Thus, it can save the effort of backtracking within the current method. However, it does not guarantee to find a solution that may exist within the current biased space.

1.2 Implementation

A set of single-method planners and bias-relaxation multi-method planners — both coarse-grained and fine-grained versions — have been implemented in the context of the Soar architecture [Laird *et al.*, 1987, Rosenbloom *et al.*, 1991]. Soar is a useful vehicle for this work because its impasse-driven subgoaling scheme provides the necessary context for planning and its multiple problem-space scheme facilitates the multi-method planning approach, though it is difficult to implement context switching for time-shared multi-method planners in Soar.

Speed-up learning is used in both single-method planners and multi-method planners for each problem, but only within-trial transfer was allowed; that is, rules learned during one problem are not used for other problems. However, learned rules were allowed to transfer from an earlier method to a later method (for the

same problem). That is, if a search path is evaluated within one method, and the results of the evaluation depend only on aspects of the method that are shared by a second method, then it should not be necessary to repeat that path when the second method is tried. Learned rules do not transfer across trials, because some rules are expensive so that they may increase the planning time for later problems [Tambe and Newell, 1988]. Restricting expressiveness such as by the unique-attribute scheme can solve this problem [Tambe *et al.*, 1990]; however this thesis uses the multi-attribute scheme to learn rules with higher generality.

The implemented multi-method planners are compared with single-method planners theoretically and experimentally in three domains: blocks-world, machine shop scheduling, and a simulated agent domain. In this thesis, the focus is on plans represented by STRIPS-like operators; however, since the multi-method framework in this thesis is independent of the operator representation, this framework should be extendable to planners with more expressive plan representations.

1.3 Contributions

The primary contributions of this thesis include the following:

1. *A methodology for building a set of planning methods with different performance and scope.* A bias determines the portion of the entire plan space considered. In particular, an effective bias improves planning performance by reducing the number of candidate plans and generates shorter plans by avoiding inefficient operator sequences. A methodology is developed to select a set of effective biases based on performance over a training problem set. By varying the selected effective biases, a set of methods with different performance and scope can be created.

2. *A new planning framework for multiple methods.* A new multi-method planning framework is developed based on the relaxation of biases. Issues

arising in multi-method planning, such as how to efficiently coordinate individual methods within a multi-method framework and the granularity at which methods can be switched, are investigated.

3. *Performance improvement over single-method planning.* Bias-relaxation multi-method planning with various granularities of method switching provides a planning system which can improve planning efficiency and reduce plan length without loss of planner completeness. In fact, for the domains investigated, both coarse-grained and fine-grained multi-method planning can reduce plan length significantly compared with single-method planning, and fine-grained planning can improve the planning time significantly compared with coarse-grained and single-method planning.

1.4 Guide to the Thesis

The body of this thesis consists of six chapters.

Chapter 2 defines the notion of bias in planning. Examples of planning bias are presented along with the justifications on which these biases depend. The differences between bias and search control heuristics are described.

Chapter 3 explains two bias dimensions — goal flexibility and goal protection — and defines single-method planners that vary along these dimensions. The implementation of these planners in Soar is described, and learning in Soar for single-method planning is discussed. Finally, experimental results in the blocks-world and machine-shop scheduling domains are provided.

Chapter 4 specifies how to build monotonic multi-method planners and bias-relaxation multi-method planners from a set of single-method planners. The issue of granularity at which individual methods can be switched is investigated, and learning in multi-method planning is discussed. Experimental results for coarse-grained and fine-grained multi-method planners are presented and compared with the results for single-method planners. Finally, the performance of multi-method planning is compared with the performance of partial-order planning.

Chapter 5 shows how this approach can be applied to a more complex domain such as a simulated agent domain.

Finally, chapters 6 and 7 discuss related work and conclusions.

Chapter 2

Bias in Planning

Bias was originally defined in the context of concept learning from preclassified training instances as any basis for choosing one generalization over another, other than strict consistency with the observed training instances [Mitchell, 1980]. Transferring the notion of bias to planning, it can be defined as "any basis for choosing one plan over another other than plan correctness", where a plan is correct if the application of the plan transforms the initial state into the goal state [Rosenbloom *et al.*, 1993].

The notion of bias is useful in planning, because bias can reduce computation effort by reducing the number of plans that must be examined, and it can potentially generate shorter plans by avoiding plans containing inefficient operator sequences such as ones that undo achieved goals or loop on states. The notion is particularly useful in multi-method planning, because bias can provide a basis for building a set of planning methods with different performance and scope. Also, method switching in multi-method planning can be easily accomplished by changing the set of biases used in the individual methods.

This chapter begins with the notion of bias in inductive concept learning, and then describes how this notion is applied to planning. Some examples of planning biases are presented, and finally, the relationship between search control and bias is discussed.

2.1 Bias in Inductive Learning

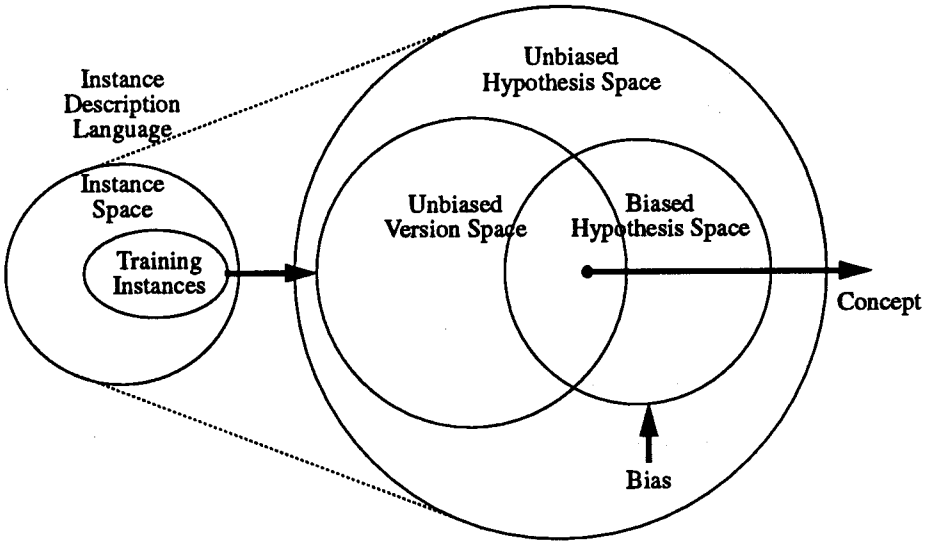
An induction problem is, in general, given an instance description language and a set of training instances, to determine a generalization that is consistent with the training instances [Mitchell, 1980]. In induction, an *unbiased hypothesis space*, denoted as $\mathcal{H}$, consists of every possible generalization on the instance space — that is, the power set of the training instances. The *unbiased version space*, denoted as $\mathcal{H}_v \subseteq \mathcal{H}$, is the portion of $\mathcal{H}$ that is consistent with the observed training instances. Then, a bias b determines a *biased hypothesis space*, denoted as $\mathcal{H}^b \subseteq \mathcal{H}$, so that the output generalization can be selected from $\mathcal{H}_v \cap \mathcal{H}^b$.

It has been shown that bias plays an important role in induction, because it influences hypothesis selection [Utgoff, 1986]. Without bias, an induction system has no basis for choosing one generalization over another. In other words, bias enables induction systems to determine how to go beyond the training instances; that is, which inductive leaps to make.

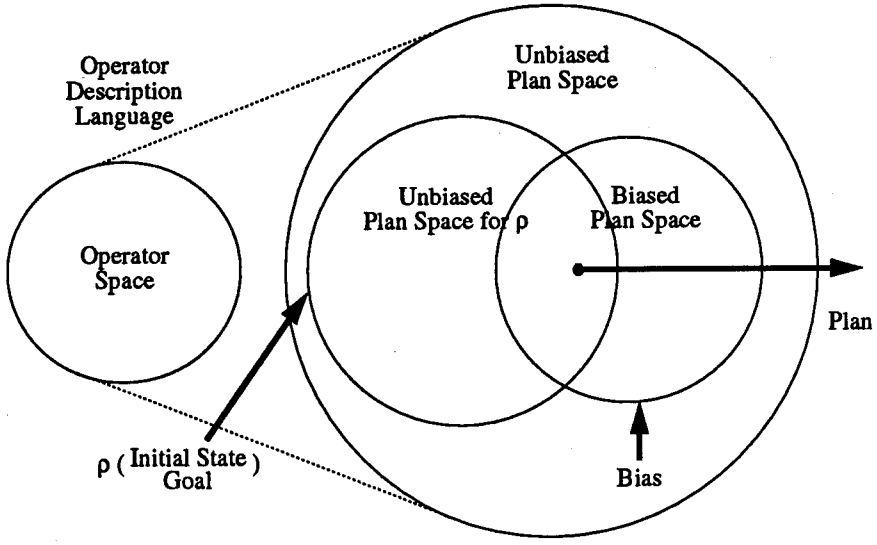
Bias can be either absolute or relative. An absolute bias completely removes parts of the unbiased version space. For example, a generalization language provides an absolute bias by eliminating any element of the unbiased version space not expressible in the language. A relative bias defines a partial order over portions of the unbiased version space. For example, one can prefer one hypothesis to another based on measures such as simplicity of the hypothesis [Michalski *et al.*, 1986].

2.2 Application of Bias to Planning

As in inductive learning, the notion of bias can be formalized in planning. Planning can be defined in terms of the notion of a *problem space* [Newell *et al.*, 1991]. A problem space consists of a set of states S, and a set of operators O. A problem, denoted as $\rho = (S_0, S_g)$, consists of two components, $S_0 \in S$ and $S_g \in S$, where S_0 is a description of an initial state of the world and S_g is a partial description of a desired state. A *plan* for a problem $\rho = (S_0, S_g)$ can be defined as



(a) Inductive concept learning



(b) Planning

Figure 2.1: Analogy between concept learning and planning.

a structure that represents the sequence of operators in O that achieves S_g from S_0 by applying each operator to each of the resulting states in the sequence.

An *unbiased plan space*, denoted as $\mathcal{P}$, is the "power sequence" — that is, the set of all sequences — of the possible operators in O.[1] For a given problem ρ, the *unbiased plan space for* ρ, denoted as $\mathcal{P}_\rho \subseteq \mathcal{P}$, is the portion of $\mathcal{P}$, for which each element of $\mathcal{P}_\rho$ solves ρ. Then, a bias b determines the *biased plan space*, denoted as $\mathcal{P}^b \subseteq \mathcal{P}$, so that the output plan can be selected from $\mathcal{P}_\rho \cap \mathcal{P}^b$.

Figure 2.1 shows the analogy between the processes of inductive concept learning and planning. In both cases the output of the process is to be some element of the unbiased hypothesis space that is consistent with the process's correctness criterion. Where the two cases differ is in the definitions of "unbiased hypothesis space" and "correctness criterion". In concept learning, the unbiased hypothesis space is the power set of the training instances, and the correctness criterion is consistency with the observed training instances. In planning, the unbiased hypothesis space is the power sequence of the possible operators, and the correctness criterion is whether the application of the plan achieves the goal state from the initial state. In spite of these differences, bias together with the process's correctness criterion, in both cases, determines which portion of the unbiased space can be the output of the process.

As in the case of induction, an absolute use of bias in planning engenders incompleteness in the planner. This incompleteness can be used to speed up the planner by reducing the number of plans that the planner can possibly generate for particular problems. However, it only really helps if the bias is an appropriate one; otherwise, desired plans can be eliminated. A relative use of bias does not introduce incompleteness. However, if the bias is not an appropriate one, generated plans may not be the desired ones. Thus, in order to show that using a bias is plausible, some form of appropriate justification is needed. For example, with a

[1]The specification here assumes that the plan space contains only totally-ordered sequences of operators, but it does not rule out a search strategy that incrementally specifies an element of the plan space by refining a partially-ordered plan structure.

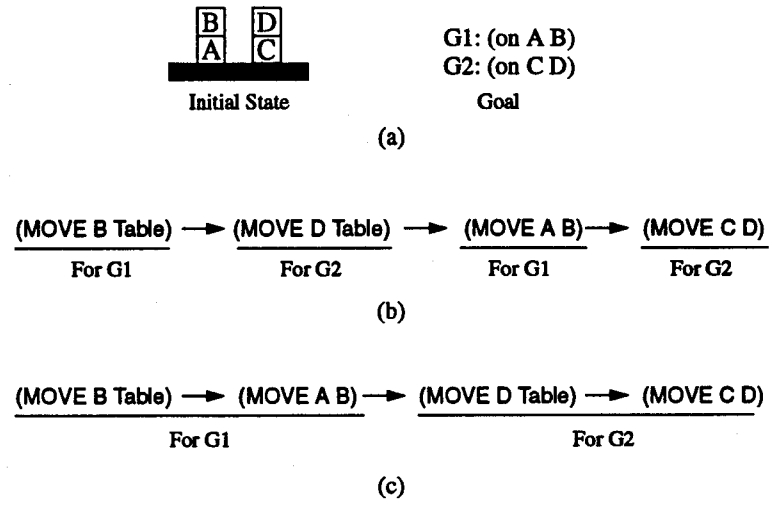


Figure 2.2: Example of the effects of a *linearity* bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

independence justification, one assumes that goal conjuncts are achieved by independent processes without interfering with other goal conjuncts in a conjunctive goal problem. With a *progress* justification, one assumes that it is always possible to move forward to solve the problem, and never required to move backward. A *boundedness* justification limits the total effort that it is reasonable to expend in solving a problem or a set of problems. In the next section, examples of planning biases based on these justifications are presented.

2.3 Examples of Planning Biases

Two typical planning biases justified by an independence justification are *linearity* and *protection*. A linearity bias removes all plans in which operators for different unachieved goal conjuncts occur in succession; that is, once an operator for one unachieved goal conjunct is in the plan, operators for other conjuncts can be placed only after the first goal conjunct has been achieved. For example, given

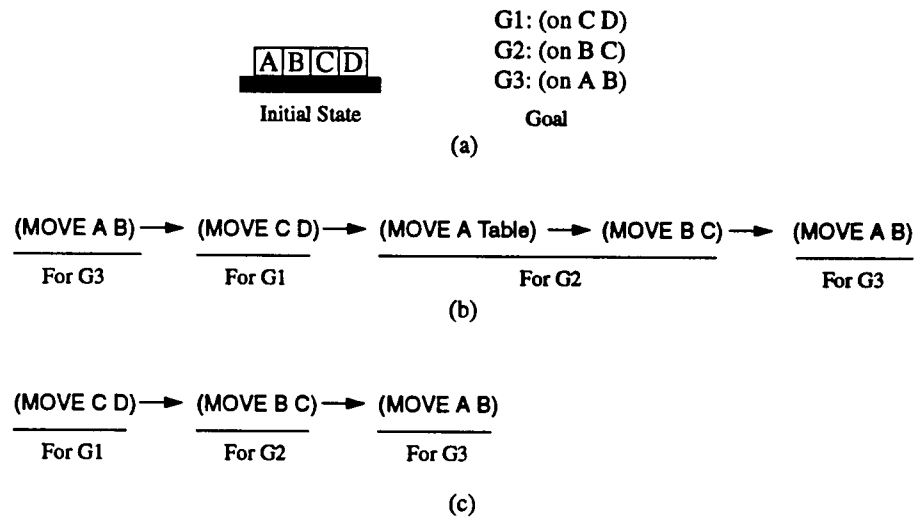


Figure 2.3: Example of the effects of a *protection* bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

the initial state and the goal conjuncts in Figure 2.2(a), plans such as the one in Figure 2.2(b) would be eliminated, while plans such as the one in Figure 2.2(c) would remain. Linearity depends on an independence justification because one assumes that while solving one goal conjunct, operators for other goal conjuncts need not be considered.

A protection bias eliminates all plans in which an operator undoes a goal conjunct that is either true in the initial state or established by an earlier operator in the sequence.[2] For example, given the initial state and the goal conjuncts in Figure 2.3(a), plans such as the one in Figure 2.3(b) would be eliminated since the

[2]The notion of protection used here was introduced by Sussman [1973]. Other forms of protection can be found in the planning literature. For example, one can protect the current goal conjunct from being clobbered by other operators while regressing an operator or a goal through a partial linear plan [Warren, 1976, Waldinger, 1977]. In partial-order planning, one can protect a causal link from being clobbered by any other planning steps, within the interval where the causal link is needed [Tate, 1977, Chapman, 1987, Barrett and Weld, 1992].

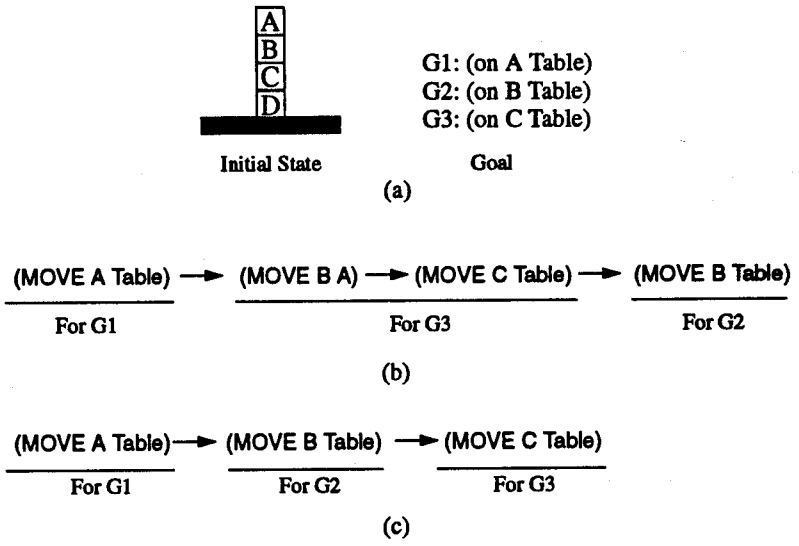


Figure 2.4: Example of the effects of a *directness* bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

operator `(MOVE A Table)` undoes the goal conjunct `(on A B)` which is established by the earlier operator `(MOVE A B)`, while plans such as the one in Figure 2.3(c) would remain. Protection is based on an independence justification since one assumes that while solving one goal conjunct, operators that interact negatively with previous goal conjuncts need not be considered.

A progress justification underlies all greedy biases. For example, protection is also justified by a progress justification, because once a goal is achieved, it would never be undone. Another bias justified by a progress justification is *directness*. A directness bias eliminates all plans in which there is at least one operator that does not directly achieve a goal conjunct included in the problem definition. For example, given the goal conjuncts and operators in Figure 2.4(a), plans such as the one in Figure 2.4(b) would be eliminated since the operator `(MOVE B A)` does not directly achieve any of the goal conjuncts in the problem definition, while plans such as the one in Figure 2.4(c) would remain. Directness is justified by

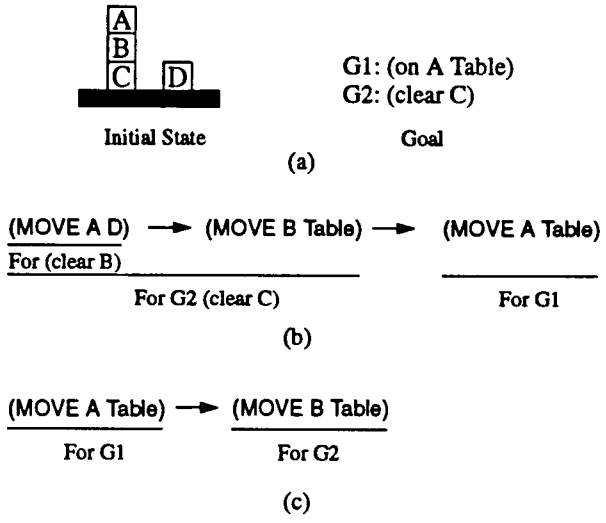


Figure 2.5: Example of the effects of a *goal-nonrepetition* bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

a progress justification, because whenever an operator is applied, a new goal is always achieved, increasing the degree of goal achievement for the entire problem. Directness is a quite interesting bias because it ensures that the number of operators to achieve each of the goal conjuncts is bounded by one.

Biases justified by a boundedness justification include *goal-depth*, *goal-breadth*, *plan length*, and *goal-nonrepetition*. Both goal-depth and goal-breadth limit the size of the goal hierarchy used in planners based on means-ends analysis (MEA), so that planning effort can be reduced. For a predefined bound n, a goal-depth bias eliminates from the hypothesis space all plans that require more than n levels of subgoals to generate, while a goal-breadth bias eliminates all plans that require more than n conjunctive subgoals for a single goal. Directness is also justified by boundedness. In fact, directness can be viewed as a special case of goal-depth bias, since it allows no generation of subgoals, thus ensuring that the depth of the goal hierarchy is bounded by one.

A plan-length bias eliminates all plans which consist of a sequence of more than n operators, for a predefined bound n, so that the length of the output plan can be bounded. It can be used either on a problem-by-problem basis or on a goal-by-goal basis. If plan-length is used on a problem-by-problem basis, the length of the output plan for the entire problem is guaranteed to be no more than n. If it is used on a goal-by-goal basis for a set of conjunctive goals, the length of the plan for the conjunctive goals is bounded by n times the number of goal conjuncts. In fact, directness is also a special case of goal-by-goal plan-length bias, where $n = 1$, because the length of the plan for each goal is bounded by one.

A goal-nonrepetition bias eliminates all plans that require a repetition on a goal literal; that is, if satisfying an unmet precondition for a selected operator requires a new goal conjunct whose literal is equivalent to the literal of its ancestor in the goal hierarchy, then that plan would be eliminated. For example, given the goal conjuncts and operators in Figure 2.5(a), plans such as the one in Figure 2.5(b) would be eliminated because it requires a repetition on a goal literal — operator `(MOVE B Table)` is chosen for conjunct `(clear C)`, but in making it applicable, an iterative `clear` conjunct `(clear B)` is generated (resulting in the selection of `(MOVE A D)` as the first operator). On the other hand, plans such as the one in Figure 2.5(c) would remain. The prime reason to use a goal-nonrepetition bias is that it forces learning from non-repetitive paths by eliminating all plans that require a repetition on a goal conjunct, so that learning specific rules for each size of repetition can be avoided. In this way, it is closely related to Etzioni's [1990b] work on restricting EBL to learn from only non-recursive explanations. This relationship will be discussed later in more detail.

2.4 Relationship with Search Control

As described before, bias affects the output of the planning process. However, if bias can be incorporated directly into the planning procedure, then it can also have a significant impact on the efficiency of the planning process by reducing the

number of candidate plans that are generated. In this way, bias can lead to effective control of search.

1. If the goal of the problem is achieved, show the output plan and stop; else continue
2. Select a goal from the goal hierarchy.
3. Select an operator to achieve the selected goal.
4. If the selected operator is applicable to the current state, create a new state by applying the operator, and remove achieved goals from the goal hierarchy. Go to step 1.
5. If the selected operator is not applicable to the current state, create subgoals to establish the unmet preconditions of the operator and add them to the goal hierarchy. Go to step 1.

Figure 2.6: A recursive planning procedure based on means-ends analysis.

For example, consider a recursive planning procedure based on means-ends analysis, as shown in Figure 2.6 [3]. Table 2.1 shows the planning biases classified according to the way they can be incorporated into this procedure. Linearity can be incorporated into goal selection (step 2) by selecting a new goal conjunct only after the current goal conjunct is achieved. Protection and goal-length can be incorporated into operator selection (step 3) by rejecting operators which violate the criterion for the bias. Goal-depth, goal-breadth, directness, and goal-nonrepetition can be incorporated into goal expansion (step 5) by limiting the expansion of the goal hierarchy.

However, despite this close relationship between search control and bias, there is a distinction between the two. Bias determines which plan is generated from the *plan space*, while search strategies determine the efficiency with which that plan is found from the *search space*. In general, the search space is not necessarily equivalent to the plan space. For example, a node in the search space for a MEA-based planner with the above procedure can be defined as a combination of the

[3]This algorithm is comparable to the one used in NOLIMIT [Veloso, 1989].

Class	Bias	Justification		
		Independence	Progress	Boundedness
Goal selection	Linearity	o		
Operator selection	Protection	o	o	
	Plan-length			o
Goal expansion	Directness		o	o
	Goal-depth			o
	Goal-breadth			o
	Goal-nonrepetition			o

Table 2.1: Examples of planning biases and their justifications classified according to a MEA-based planning procedure.

current state and goal hierarchy, whereas a node in the plan space for this planner can be defined as a partial sequence of operators. Whenever an operator is applied to the current state, a node is expanded both in the search space and the plan space. However, if the selected operator is not applicable to the current state, a node in the plan space is not expanded, while a node in the search space is expanded for the new goal hierarchy.

2.5 Summary

In this chapter, the notion of bias is applied to planning. An analogy between the processes of concept learning and planning is presented in terms of the usage of bias in the process. Since bias determines which portion of the unbiased space can be the output of the process, using an appropriate bias is critical; if it is too weak it has no effect, but if it is too strong it can eliminate the desired output. Some examples of planning biases are introduced which can be justified by independence, progress, or boundedness. The relationship between search control and bias is discussed.

Chapter 3

Single-Method Planners

As described in the introduction, the first hypothesis underlying this research is that no single method will satisfy both sufficiency and efficiency for all situations. The ideal way to evaluate the hypothesis would be to construct all possible single-method planners and to evaluate their performance and scope for every possible domain. However, this is not possible. In this research, a system is constructed that can utilize a set of different planning methods, which vary in the amount of bias used. These methods are implemented in the context of Soar, an architecture which integrates basic capabilities for problem-solving, use of knowledge, learning, and perceptual-motor behavior [Laird *et al.*, 1987, Rosenbloom *et al.*, 1991].

Soar has not traditionally been seen as a planning architecture, partly because it does not create structures that resemble traditional plans, such as totally-ordered plans or partially-ordered plans, and partly because its problem-solving approach does not closely resemble the traditional planning methods [Rosenbloom *et al.*, 1993]. However, recent work on a Soar-based framework for planning has demonstrated how versions of such standard planning methods as linear, nonlinear, and abstraction planning can be derived from the Soar architecture [Rosenbloom *et al.*, 1990].

This chapter begins with an overview of planning in Soar and introduces a set of different planning methods as implemented in Soar (version 6). The effect of learning in these methods with respect to the performance of planning is discussed.

Finally, these methods are evaluated experimentally in terms of planner completeness (for sufficiency), planning time and plan length (for planning efficiency and execution efficiency, respectively) in two domains.

3.1 Planning in Soar

3.1.1 Overview of Soar

Soar is based on the hypothesis that all symbolic goal-oriented behavior may be represented in terms of *problem spaces* [Newell *et al.*, 1991]. A problem space is defined by a set of states and a set of operators. The states represent situations, and the operators represent actions which apply to current states to yield new states. Problem-solving in Soar is driven by applying operators to states within a problem space to achieve a goal. A *goal context* consists of a goal, together with the current problem space, state, and operator.

Figure 3.1 illustrates the architectural structure of Soar. Knowledge is stored in a permanent recognition memory and a temporary working memory. Recognition memory consists of a set of variabilized rules, where each rule is a condition-action pair. The conditions of each rule match against the content of working memory. Conditions can contain variables, so that a single condition can match against different data in working memory. If the conditions of a rule are matched, the actions of the rule are instantiated to propose *preferences* that change the working memory. The most typical preferences are feasibility (acceptable, reject) and desirability (best, better, indifferent, worse, worst) preferences. These preferences are held in preference memory and used by a *decision procedure* to determine what changes are made to working memory.[1]

If the system does not have sufficient information about a situation to make a decision for that situation, then an *impasse* arises. For example, if the system

[1]The current Soar version has a separate preference memory, which is not included in this figure for simplicity.

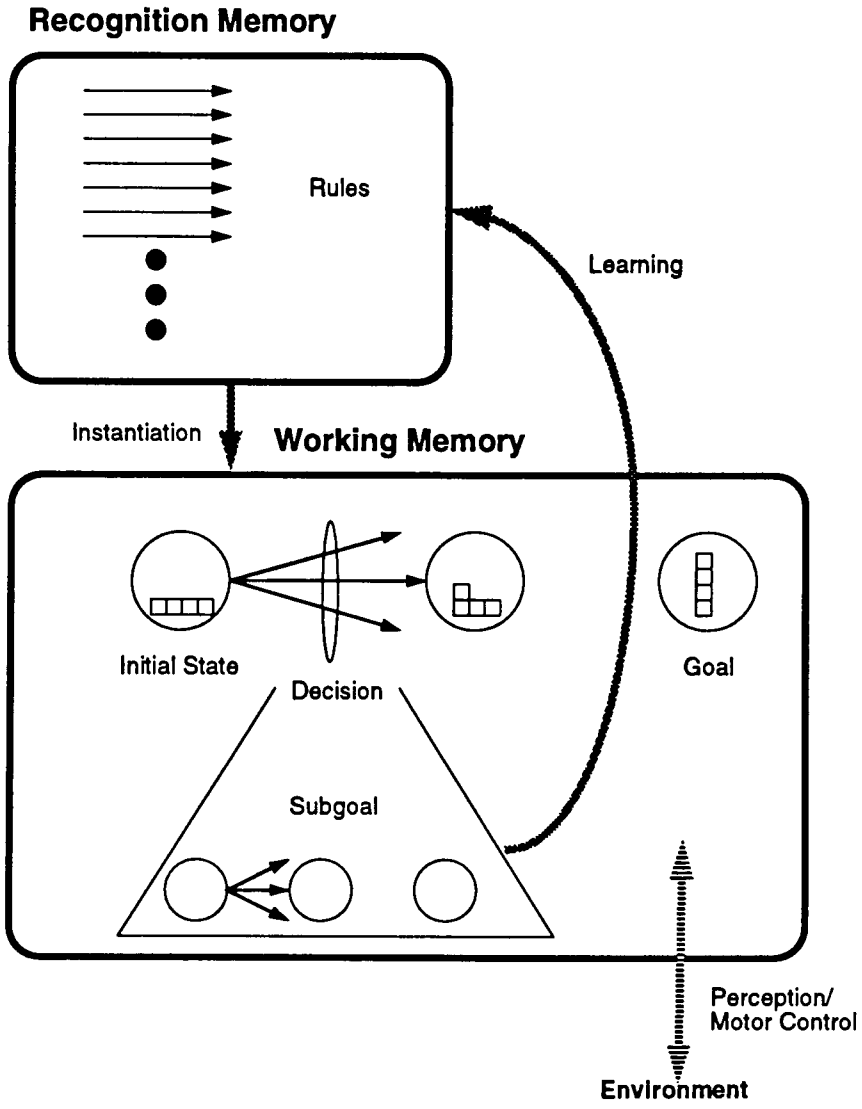


Figure 3.1: The Soar architecture.

is unable to select the next operator from a set of candidate operators, then an impasse, called a *tie impasse* (or *selection impasse*) arises. Other types of impasses are generated if the system fails to generate a set of candidate operators (*generation impasse*), or fails to execute the selected operator (*execution impasse*) [Rosenbloom *et al.*, 1990].

In response to an impasse, a subgoal is automatically generated. Within the subgoal, Soar searches for more information that can lead to the resolution of the impasse. As the result of the subgoal, new preferences are generated and new rules are learned (via a *chunking* process) whose actions are based on the preferences that are the results of the subgoal, and whose conditions are based on the working-memory elements in supergoals that led to the results. In effect, chunking is much like explanation-based learning [Rosenbloom and Laird, 1986].

Note that the notions of subgoal and operator in a Soar should be distinguished from those in traditional planning. A Soar subgoal is generated in response to an impasse whenever progress cannot be made on the current goal, and terminated when the impasse is resolved. On the other hand, a planning subgoal is generated in response to a precondition violation and terminated when the violated condition is achieved. A precondition violation may or may not create an impasse in Soar depending on whether or not knowledge to achieve the violated condition is available in the current goal context.

In the planning framework for this research, planning goals (together with subgoals) and their hierarchy are explicitly represented as augmentations of Soar states. Precondition violation is handled in a single goal context without creating a Soar subgoal. However, if there is no information about how to apply an operator (yielding an execution impasse) or how to select among the candidate operators (yielding a tie impasse), a Soar subgoal is created. A planning operator is represented as a set of variabilized rules which create and apply an instantiated Soar operator to change the current Soar state, where the current planning state and the goal hierarchy are represented.

1. If the goal of the problem is achieved, stop; else continue.
2. Select an operator to achieve one of the active goal conjuncts in the goal hierarchy.
3. If the selected operator is applicable to the current state, create a new state by applying the operator, and remove achieved goals from the goal hierarchy. Go to step 1.
4. If the selected operator is not applicable to the current state, create subgoals to establish the unmet preconditions of the operator and add them to the goal hierarchy. Go to step 2.

Figure 3.2: The planning algorithm based on means-ends analysis as implemented in Soar.

In this work, the predominant planning method in Soar is means-ends analysis (MEA). The version of means-ends analysis implemented in this work is close to the algorithm described in Figure 2.6. Figure 3.2 shows the skeleton of the MEA-based planning algorithm implemented in Soar for the framework of this thesis. There are only two differences between this algorithm and the one shown in Figure 2.6. First, in this algorithm, a goal conjunct is selected implicitly from the goal hierarchy when an operator is selected in step 2. By merging two steps (goal selection and operator selection) into a single operator selection step, the number of decisions required to generate a plan can be reduced. Second, there is no explicit output plan to print in step 1 in this algorithm. This is because a plan in Soar is rarely represented as a unitary entity like a totally-ordered or partially-ordered plan. Instead, a plan in Soar is represented as a set of control rules or a set of preferences which jointly specify which operators should be executed at each point in time.

In the following sections, operator representation, plan representation, and planning in Soar are described in more detail.

3.1.2 Operator Representation in Soar

In the planning framework for this thesis, the implementations of planning operators are represented by three classes of variabilized rules in recognition memory

If the problem-space is *blocks-world*
  ∧ There exists an active goal (on <x> <y>)
  ∧ (on <x> <y>) is not achieved
⇒
  Propose an operator (MOVE <x> <y>) for (on <x> <y>)

(a) An operator proposal rule for (on <x> <y>).

If the problem-space is *blocks-world*
  ∧ There exists an active goal (clear <x>)
  ∧ (clear <x>) is not achieved
  ∧ There exists a block <top> on top of <x>
  ∧ There exists an object <y> which is different from <x> and <top>
⇒
  Propose an operator (MOVE <top> <y>) for (clear <x>)

(b) An operator proposal rule for (clear <x>).

Figure 3.3: Examples of operator proposal rules.

— operator proposal rules, operator application rules, and goal expansion rules — plus instantiated operator objects in working memory. An operator proposal rule implements a bit of means-ends analysis, determining when it is appropriate to propose operators. This rule is instantiated (possibly multiple times) based on the current goal hierarchy represented in the working memory, creating a set of instantiated Soar operators in the working memory.

Figure 3.3 shows examples of operator proposal rules in the blocks-world domain. In our implementation of blocks-world, there is a single general operator, `MOVE`, which moves a block from one location to another. However, depending on the type of goal this operator is trying to achieve, different operator proposal rules can be specified, as shown in Figure 3.3(a) and (b). A single operator proposal rule can be instantiated with different components of the state, yielding multiple instantiated operators. In Figure 3.3(a), for example, if the goal of a problem is to stack a set of n blocks, represented by (and (on $Block_1$ $Block_2$) (on $Block_2$ $Block_3$)

If the problem-space is *blocks-world*
  ∧ The operator is (MOVE <x> <z>) for goal <w>
  ∧ <x> is on <y>
  ∧ <x> and <z> are clear
  ∧ <z> is the table
⇒
  <x> is not on <y>
  ∧ <x> is on <z>
  ∧ <y> is clear
  ∧ <w> is achieved

(a) An operator application rule to put down a block onto the table.

If the problem-space is *blocks-world*
  ∧ The operator is (MOVE <x> <z>) for goal <w>
  ∧ <x> is on <y>
  ∧ <x> and <z> are clear
  ∧ <z> is a block
⇒
  <x> is not on <y>
  ∧ <x> is on <z>
  ∧ <y> is clear
  ∧ <z> is not clear
  ∧ <w> is achieved

(b) An operator application rule to stack a block onto another block.

Figure 3.4: Examples of operator application rules.

... (on $Block_{n-1}$ $Block_n$)), then (on <x> <y>) is instantiated with each of the $n-1$ goal conjuncts when the problem solving starts.

Once an operator has been selected for the current state by the decision procedure, it can be applied to generate a new state if its preconditions are met. Figure 3.4 shows examples of operator application rules for the MOVE operator. In this implementation of blocks-world, two operator application rules are used for this operator: one to put down a block onto the table, and one to stack a block onto another block. Once an operator has been applied, operator proposal rules

If the problem-space is *blocks-world*
  ∧ The operator is (MOVE <x> <z>) for goal <w>
  ∧ <x> is not clear
⇒
  Create a new goal <new> to clear <x>
  ∧ The parent of <new> is <w> in the goal hierarchy

(a) A goal expansion rule in which the block to be moved is not clear.

If the problem-space is *blocks-world*
  ∧ The operator is (MOVE <x> <z>) for goal <w>
  ∧ <z> is not clear
⇒
  Create a new goal <new> to clear <z>
  ∧ The parent of <new> is <w> in the goal hierarchy

(b) A goal expansion rule in which the destination block is not clear.

Figure 3.5: Examples of goal expansion rules.

are matched with the new state and the updated goal hierarchy, generating the next set of candidate operators.

If the selected operator is not applicable, goal expansion rules (as shown in Figure 3.5) are instantiated to generate a new goal hierarchy. In effect, goal expansion rules implement operator subgoaling in means-ends analysis.

3.1.3 Plan Representation in Soar

As described in Chapter 2, a plan for a problem can be defined as a structure that represents the sequence of actions to be taken for that problem [Rosenbloom *et al.*, 1993]. With this definition of a plan in hand, two predominant structures can be identified that serve as plans in Soar. The first structure is the set of *variabilized control rules* in recognition memory that serves as generalized plans for classes of potential goals. Control rules are different from operator representation rules described in the previous section in that control rules generate instantiated

If the problem-space is *blocks-world*
  ∧ Goal protection is assumed to hold
  ∧ There exists an active goal (on <x> <y>)
  ∧ There exists an active goal (on <y> <z>)
  ∧ (on <x> <y>) and (on <y> <z>) are not achieved
  ∧ <x> and <y> are blocks
  ∧ <x> and <y> are clear
  ∧ The proposed operator is (MOVE <x> <y>)
⇒
  The operator is worst

Figure 3.6: A generalized plan.

preferences to help select the current operator from the candidate operators, thus yielding indirectly a sequence of operators. The second structure is the set of *instantiated preferences* in preference memory that serves as instantiated plans for active goals. The instantiated preferences can be generated either by the generalized plans or as the results of subgoals (that is, by planning).

Figure 3.6 shows an example of a generalized plan for a set of problems shown in Figure 3.7(a). This rule implies that if goal protection is assumed to hold, one wants a stack of at least three blocks, neither of the top two blocks (out of the three) are in position, both of the top two blocks (out of the three) are clear, and an operator is proposed to put the top one on the second one, then that operator is worst. Figure 3.7(b) shows the sequence of steps to generate a sequence of operators for a four-block-stacking problem. For each step, it shows the current state, the goal conjuncts that have not yet been achieved, the operators proposed, and the portion of the instantiated plan generated from the generalized plan in Figure 3.6. Figure 3.7(c) then shows the actual operator sequence this plan generates.

The plan representation in Soar has many interesting aspects. First, the preference language has an imperative construct (*best*) that allows relatively direct specification of the next action to perform; however, it also goes beyond this. For example, partial orders can be specified by using binary preferences such as *worse*

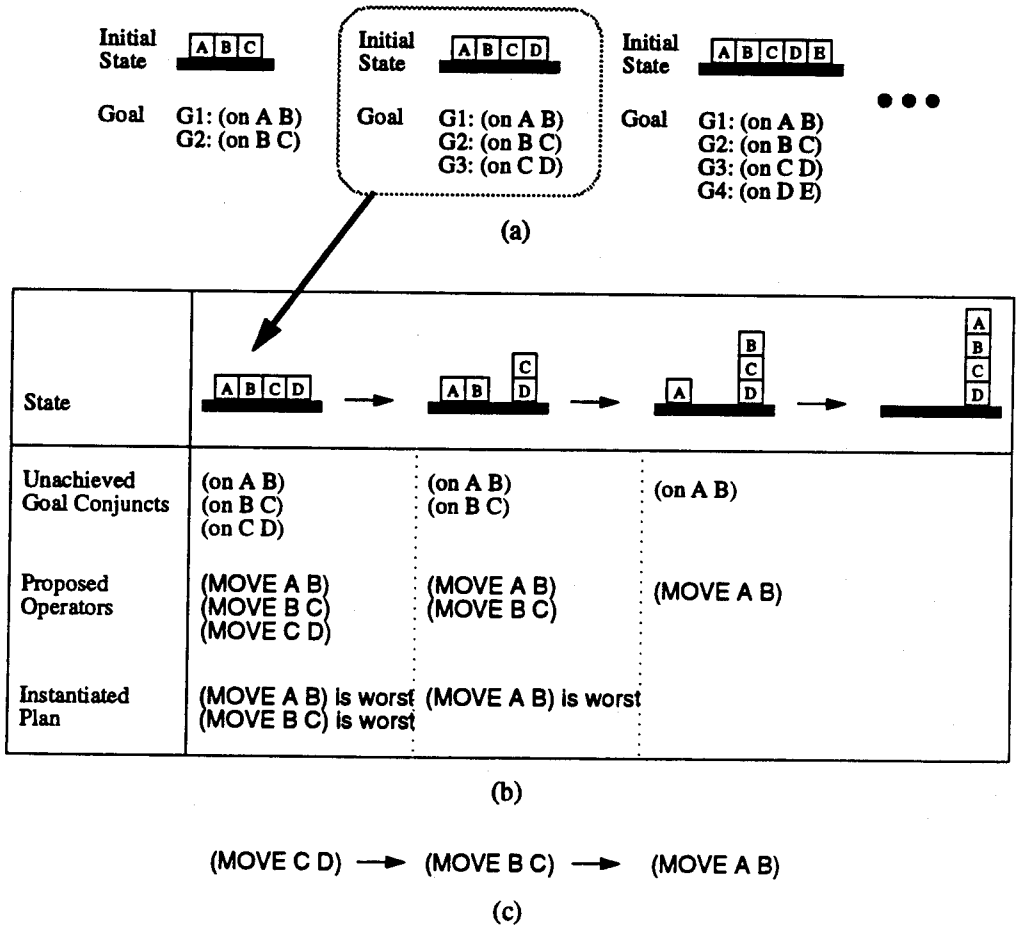


Figure 3.7: The plan representation in Soar: (a) a set of problems which are solvable by the rule in Figure 3.6, (b) the sequence of steps for a four-block-stacking problem, (c) the sequence of operators.

and *better*, and also operator avoidance can be specified by using *worst* and *reject* preferences. Second, the use of control rules provides a fine-grained conditionality and context sensitivity that allows it to easily encode such control structures as conditionals and loops. In addition, the variabilization of the control rules allows a single plan fragment to be instantiated for multiple related decisions.

3.1.4 Planning Methods in Soar

Although the plan representation in Soar is different from conventional plan representations, recent work on a Soar-based framework of planning has demonstrated how versions of such standard planning methods as linear, nonlinear, and abstraction planning can be derived by adding method increments that include core means-ends knowledge about what operators to suggest for consideration, and varying knowledge about how to respond to impasses resulting from precondition failures [Rosenbloom *et al.*, 1990].

Figure 3.8 illustrates initial traces of particular versions of these three forms of planning as implemented in Soar for Sussman's anomaly (Figure 1.1) in the blocks-world.[2] They all start with a top-level operator that is to achieve the entire conjunctive goal — `(and (on B C) (on A B))` — directly from the initial state, and reach an execution impasse if there is no information about how to do this. In response to this impasse, a subgoal is created where means-ends analysis is used to generate the set of candidate operators — `(MOVE B C)` and `(MOVE A B)` — that are known to potentially be able to achieve any of the goal conjuncts. A tie impasse then occurs unless there is information about how to pick among them (or unless only one operator is generated). In this tie impasse, a look-ahead search begins by selecting one of the alternatives to evaluate — here it is `(MOVE A B)`. Its preconditions are tested and if the operator is known to be applicable, it is

[2]Abstraction in the blocks world is shown for comparison purpose. Although abstraction has not actually implemented within the planning framework for this research, it has been implemented in Soar by Unruh [1993].

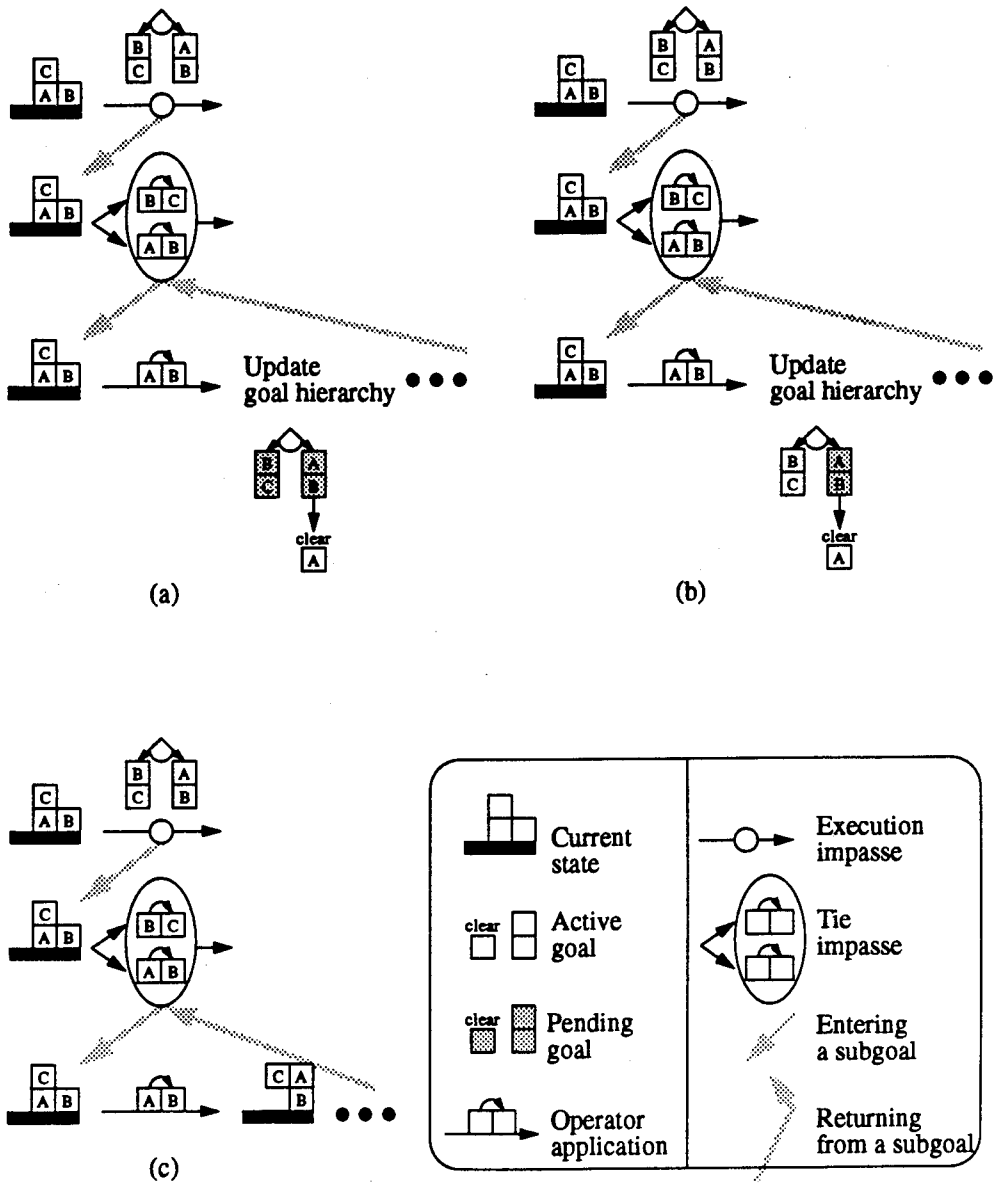


Figure 3.8: Planning in the blocks-world using (a) linear; (b) nonlinear; and (c) abstraction planning.

executed to create a new state. If it is not known to be applicable — as here — what happens next depends on the planning methods.

With abstraction, the operator is executed anyway and problem solving just continues. In Figure 3.8(c), for example, operator `(MOVE A B)` is executed even though block A is not clear. Without abstraction, as in Figure 3.8(a) and (b), a new set of goal conjuncts is generated from the operator's unmet preconditions.

The difference between linear and nonlinear planning, at least for these versions, is in the focus of operator generation from the new goal hierarchy. Linear planning shifts focus completely to the new conjunct — `(clear A)` as in Figure 3.8(a). It stays with the new conjunct until it is achieved, and then pops back to the original conjunct that led to the precondition violation. Processing shifts to one of its siblings (if there are any) only after the original conjunct is achieved.

Nonlinear planning instead shifts focus to an expanded set of conjuncts that includes the new set plus the original set minus the conjunct that led to the precondition violation, yielding `(on B C)` and `(clear A)` here (Figure 3.8(b)). At any point in time an operator can be selected for any of these conjuncts, enabling operator sequences to be interleaved as necessary (similar to the *casual-commitment* approach to nonlinear planning [Veloso, 1989]). For both planning methods, once the new focus has been determined, planning continues recursively by using means-ends analysis to generate candidate operators from the new goal hierarchy.

So far, we have been referring to these methods as "planning methods", because they are versions of classical methods used in the creation of plans. With this notion in hand, the question to be asked then is how they actually yields plans. As mentioned earlier, a plan in Soar consists of a set of *plan fragments* — that is, a set of either instantiated preferences or generalized control rules. Instantiated plan fragments are generated whenever operator preferences are created in working memory. This can happen simply by the instantiation of a generalized plan fragment (by the execution of a control rule) or as a result of projection in an operator-selection subgoal.

In projection, one or more operators are tried out in look-ahead search to see which ones lead to success or failure. Success engenders *best* preferences and failure engenders *worst* preferences. For example, in Figure 3.8(a) a best preference is returned from the selection subgoal if the result of evaluating (`MOVE A B`) is success, whereas a worst preference is returned if the result is failure. These preferences act directly as fragments of a plan for the currently active goals. In addition, whenever a preference is returned as a result of a subgoal, it triggers Soar's chunking process, which creates and stores a control rule that acts as a generalized plan fragment for classes of problems. These relationships are summarized by the following two influence paths.

Planning method ⇒ Projection ⇒ Instantiated plan

Planning method ⇒ Projection ⇒ Learning ⇒ Generalized plan

While projection plays an integral role in determining which plans are created, what is projected and what is considered to be success or failure are determined by the planning method. Within this framework, planning biases are implemented by altering the planning method, which then determines which plans are created. through the influence paths above. For example, a protection bias is implemented by altering the planning method to terminate look-ahead with failure any time a projected path leads to a protection violation. In comparison to the same planner without this bias, the protection planner will lead to the creation of worst preferences (and negative control rules) which will avoid paths that violate protection.

3.2 Implemented Planning Biases

Within the context of Soar, an integrated planning system has been constructed which utilizes a set of different methods. These methods vary in the amount of bias used. The planning biases that have been concentrated on in this research are *directness*, *linearity* and *protection*. Linearity and protection are chosen because they have been widely used in the planning literature, and directness is chosen

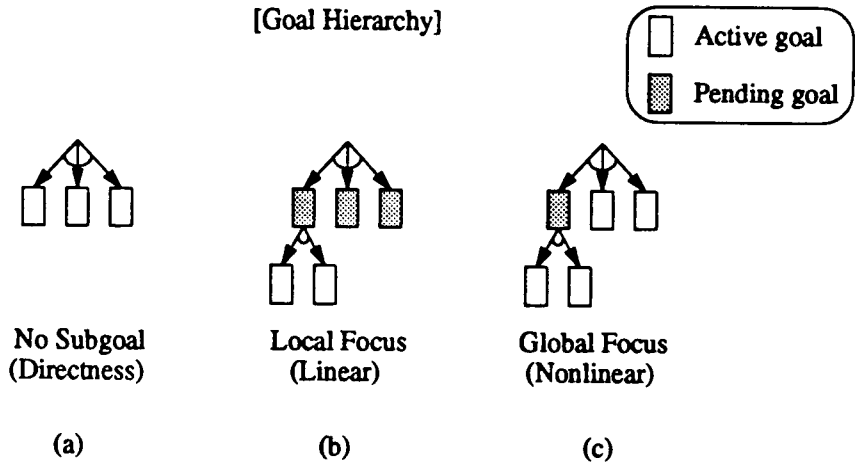


Figure 3.9: Goal-flexibility dimension.

because it can generate an efficient plan very quickly for a number of problems. These three biases are defined along two bias dimensions — goal flexibility and goal protection.

The goal-flexibility dimension is shown in Figure 3.9. It determines the degree of flexibility the planner has in generating new subgoals and in shifting the focus in the goal hierarchy. This dimension subsumes the directness and linearity biases. The most restricted point along this dimension allows no generation of new subgoals for precondition violations (Figure 3.9(a)), yielding a single-level goal hierarchy. This implements a directness bias by ensuring that each of the operators in a plan directly achieves an initial goal conjunct, rather than an unmet precondition of another operator.

The second point along the flexibility dimension allows generation of new subgoals, but only a single local set of conjuncts are attended to at any point in time (Figure 3.9(b)). This local focus of attention has two main consequences for the planner. First, it reduces the branching factor of the planners's search — with

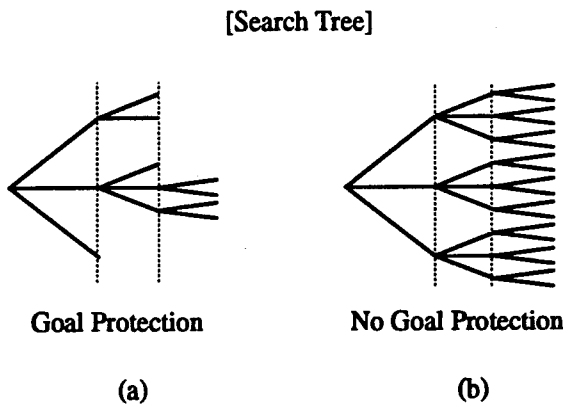


Figure 3.10: Goal-protection dimension.

respect to the nonlinear planner — by restricting the set of operators that the planner can consider at any point in time to just those that are able to achieve the local conjuncts. Second, with the assumption that an operator achieves only one goal conjunct and that the placement of operators in the plan is restricted within the context of the local conjuncts from which it arose, it enforces linearity on the resulting plans (thus implementing linear planning) by ensuring that operators for different goal conjuncts cannot be interleaved in the output plans.

The third point along the flexibility dimension allows the global use of subgoals; that is, new goal conjuncts are generated for unmet preconditions, and operators are simultaneously considered for all unsatisfied conjuncts (Figure 3.9(c)). This is the least restricted version, and implements nonlinear planning by allowing operators for different goal conjuncts to be interleaved, as in NOLIMIT [Veloso, 1989].

The goal-protection (GP) dimension is shown in Figure 3.10. The two points implemented along this dimension correspond to goal protection (Figure 3.10(a)) — that is, every achieved top-level goal conjunct is protected between the time it is achieved and the time it is no longer needed — and to no goal protection

(Figure 3.10(b)). The main consequence of using goal-protection is that it shrinks the search space by cutting off sequences of operators which violate goal protection.

Figure 3.11 characterizes the 3×2 set of planning methods derived from these bias dimensions. Each of the cells in Figure 3.11 shows a label representing the planner for that cell along with a problem that is just hard enough to require that planner; that is the problem can be solved optimally by the planner represented by that cell, but not by either the planner to its left or the planner above it. The bottom-left cell represents an extended blocks-world problem where a block that is second from the top of a tower can be moved [Etzioni, 1990a]. The most restricted planner (M_1) — a direct goal-protection planner — is in the top-left cell of the figure. While quite restrictive, it is sufficient to solve the block-stacking problem shown in that cell of the figure. The least restricted planner (M_6) — a nonlinear planner without goal protection — is in the bottom-right cell of the figure. It is the only planner in the figure capable of generating an optimal solution to the blocks-world problem shown in that cell. Between these two extremes, moving up or to the left yields more bias, while moving down or to the right yields less bias. In each of these intermediate cells, the problem shown is one that is just hard enough to require that planner; that is, the problem can be solved optimally by the planner represented by that cell, but not by either the planner to its left or the planner above it.

Note that in the blocks-world domain, both M_5 and M_6 are complete planners in that they can potentially solve every problem, though M_5 may not be able to generate an optimal solution for some problems, However, in domains with irreversible operators as shown in Figure 1.2, M_6 is the only complete planner.

Figure 3.12 compares the traces of these methods for Sussman's anomaly. They all start with a combination of the initial state, the entire conjunctive goal — `(and (on B C) (on A B))` — and the initial set of candidate operators — `(MOVE B C)` and `(MOVE A B)` — which are generated by means-ends analysis. If there is no information about which operator to select, a tie impasse occurs.[3] In this tie

[3]For simplicity of presentation, these traces only show tie impasses.

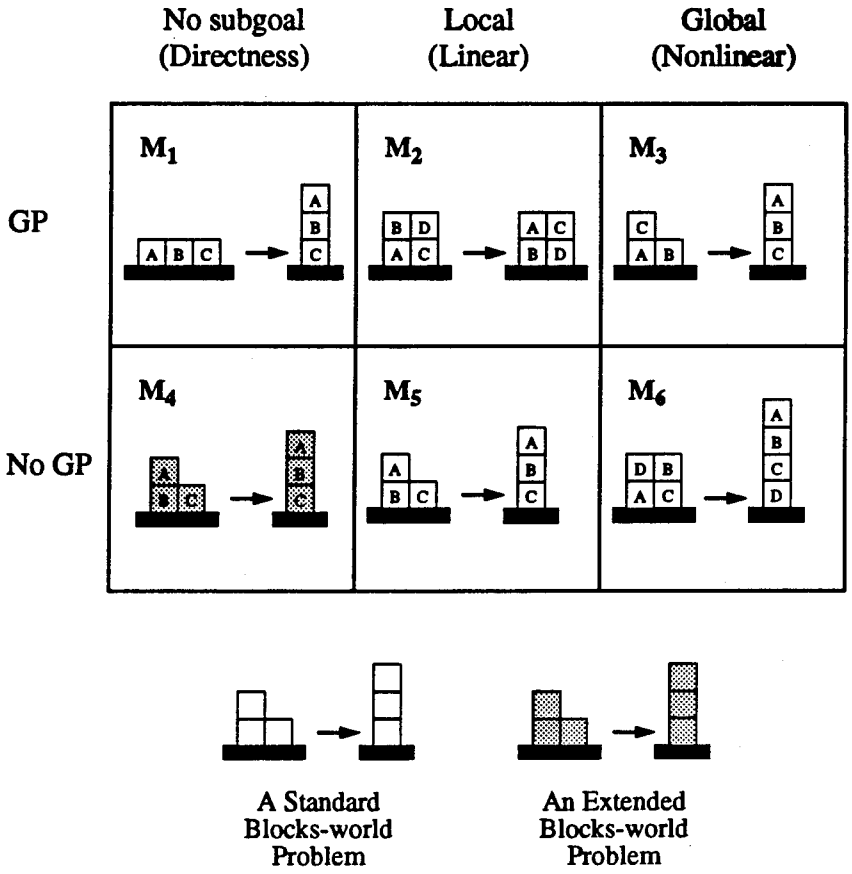
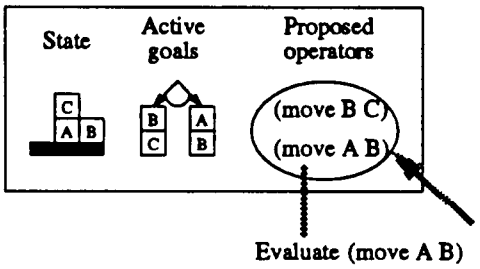
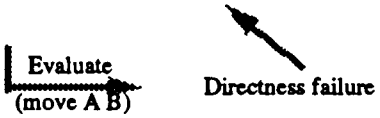


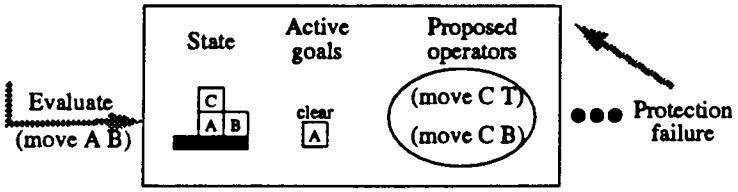
Figure 3.11: The planning methods generated by the bias dimensions.



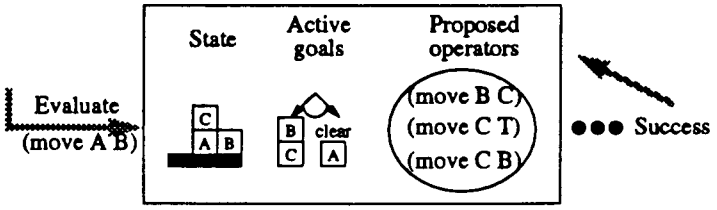
(a) The initial state, goals, and operators



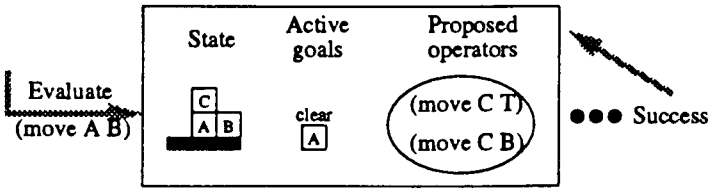
(b) Directness & protection (M_1)



(c) Linear & protection (M_2)



(d) Nonlinear & protection (M_3)



(e) Linear & no protection (M_5)

Figure 3.12: Planning in Soar.

impasse, a look-ahead search begins by selecting one of the alternatives to evaluate — here it is `(MOVE A B)`.

If the directness bias is used — as in Figure 3.12(b) — the evaluation of `(MOVE A B)` is terminated immediatedly with failure as the evaluation value, and the other operator `(MOVE B C)` is selected. If the directness bias is not used, a new set of goal conjuncts are generated from the operator's unmet preconditions (Figure 3.12(c-e)).

Linear planning focuses on the new conjunct — `(clear A)` as in Figure 3.12(c) and (e) — until it is achieved, and then returns to the original conjunct that led to the impasse — here, `(on A B)`. Sibling conjuncts — here, `(on B C)` — are considered only after the original conjunct is achieved. In this problem, this eventually leads to failure if a protection bias is used (Figure 3.12(c)), or generates a non-optimal plan if a protection bias is not used (Figure 3.12(e)). Nonlinear planning instead shifts focus to the entire set of goal conjuncts (except the one that led to the impasse) — `(and (on B C) (clear A)` as in (Figure 3.12(d)). This eventually can yield an optimal plan for this problem regardless of the use of a protection bias.

3.3 Learning in Single-Method Planners

For each of the single-method planners, chunking is performed over the planner's projection (look-ahead) process: the elements to be explained are the preferences generated during projection, and the explanations are the traces of the projections that led to the preferences. Both positive rules and negative rules can be learned from projections. Figures 3.13 and 3.15 provide a simple example of this.

Figure 3.13 shows a path projected by the nonlinear planner for a simple four-block-unstacking problem. This projection proceeds through multiple tie impasses until the problem is successfully solved. In this example, `(MOVE A Table)` is evaluated in the first operator-selection subgoal, and `(MOVE B Table)` is evaluated in the second operator-selection subgoal. As shown in Figure 3.14, this results in a

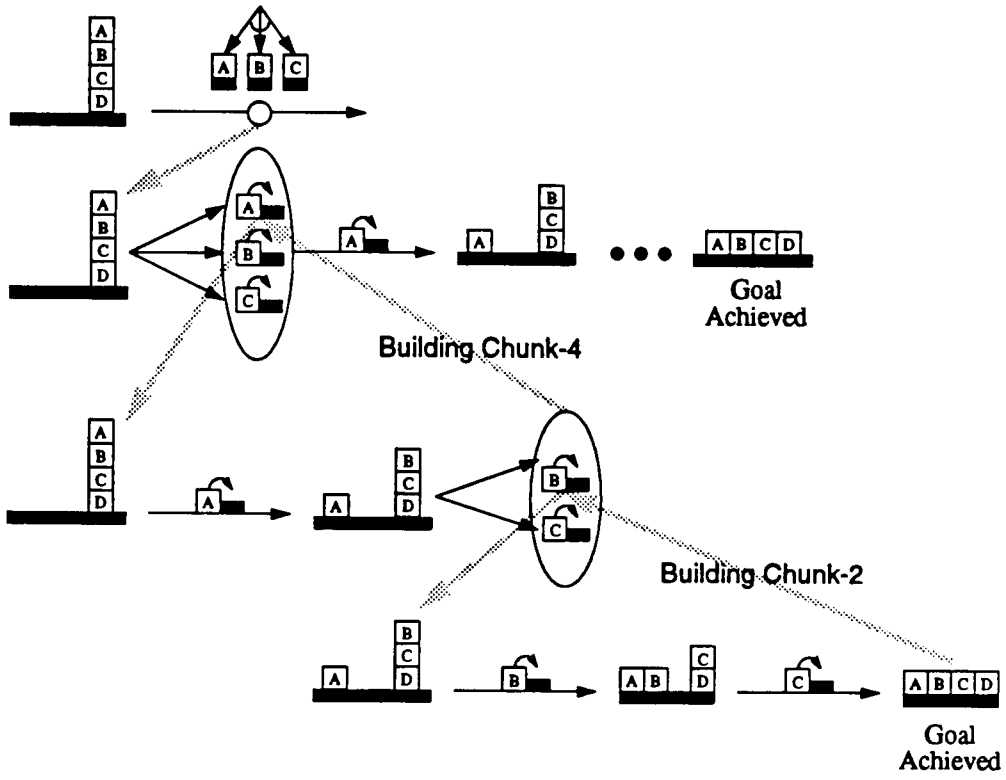


Figure 3.13: Four block unstacking with nonlinear planning.

pair of positive control rules, one for each correct decision on the solution path. Evaluating an operator which is not directly applicable in the current state — here `(MOVE B Table)` or `(MOVE C Table)` in the first operator-selection subgoal — also leads to success in nonlinear planning, though the learned rules are more complex.

Figure 3.15 shows a path projected with a directness bias for the same block-unstacking problem. In contrast to the previous case, the projection is terminated with failure as soon as the non-applicable operator `(MOVE B Table)` is selected. As shown in Figure 3.16, this yields a negative control rule for the incorrect decision on the solution path.

Chunk-2: If the problem-space is *blocks-world*
∧ There exists an active goal (on <y> Table)
∧ There exists an active goal (on <z> Table)
∧ (on <y> Table) and (on <z> Table) are not achieved
∧ <y> is on <z>
∧ <y> is clear
∧ The proposed operator is (MOVE <y> Table)
⇒
The operator is best

(a)

Chunk-4: If the problem-space is *blocks-world*
∧ There exists an active goal (on <x> Table)
∧ There exists an active goal (on <y> Table)
∧ There exists an active goal (on <z> Table)
∧ (on <x> Table), (on <y> Table), and (on <z> Table) are not achieved
∧ <x> is on <y>
∧ <y> is on <z>
∧ <x> is clear
∧ The proposed operator is (MOVE <x> Table)
⇒
The operator is best

(b)

Figure 3.14: Learned rules for block unstacking with nonlinear planning.

Note that if the planner's bias is reflected in an altered planning method, which in turn yields an altered projector, then the planner's bias can indirectly induce a bias in the resulting learning process. For example, the rules in Figure 3.14 are relatively specialized, because each must encapsulate the entire explanation for why a particular operator will eventually lead to success. In larger problems these explanations get even larger, and the rules end up being even more specialized.

On the other hand, the explanation for the rule in Figure 3.16 is quite short — based as it is on the explicit assumption that directness can hold and on the failure of the first selected operator to be applicable. As it turns out, this single rule is

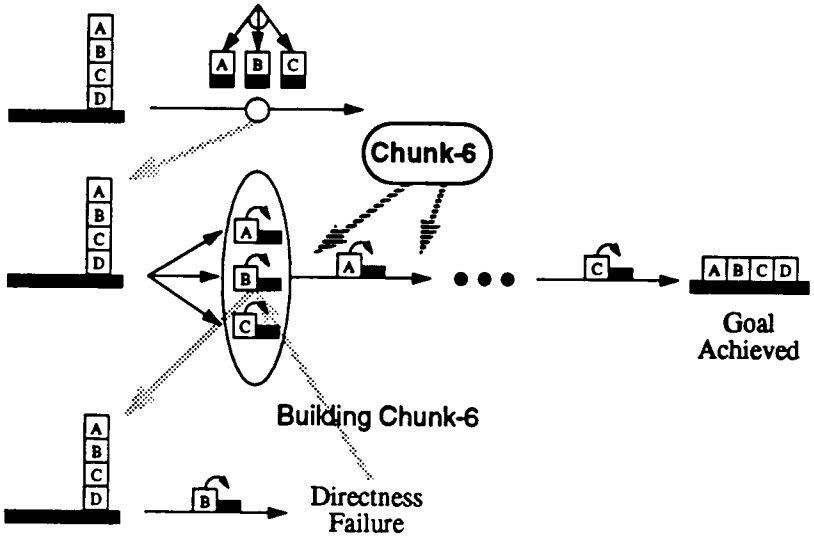


Figure 3.15: Four block unstacking with directness.

Chunk-6: If the problem-space is *blocks-world*
∧ Directness is assumed to hold
∧ There exists an active goal (on <y> Table)
∧ (on <y> Table) is not achieved
∧ <y> is not clear
∧ The proposed operator is (MOVE <y> Table)
⇒
The operator is worst

Figure 3.16: A learned rule for block unstacking with directness.

general enough to handle the entire problem, by removing from consideration all operators that attempt to move unclear blocks onto the table. The bias in this case has thus yielded faster planning and learning — because of shorter projections and explanations — and has resulted in the acquisition of fewer, more general rules.

Implicit in this example is one approach to producing *generalization to N* [Boström, H., 1990, Cohen, 1988, Shavlik, 1989, Subramanian and Feldman, 1990], where a plan learned for a problem of a particular size can transfer to solve problems with the same structure but of arbitrary size [Rosenbloom *et al.*, 1993]. Without directness, the control rules are specific to particular numbers of blocks, and thus can only be used to directly solve terminal subregions of larger problems. However, with directness, a single rule is learned that removes from consideration at each decision all operators that move unclear blocks to the table, *no matter how many unclear blocks there are.* This idea can be applied to other problems and biases as well. Figure 3.17, for example, shows a path projected with protection for a four-block-stacking problem. As with the directness bias in block unstacking, a protection bias leads here to learning a single negative rule (Figure 3.18) that can be applied to stacking problems of arbitrary size.

A third type of bias that can also induce generalization to N is *complete* protection. Complete protection is a variant on goal protection that provides a very strong bias by not only protecting established goals, but also protecting established operator sequences. That is, it disallows any backtracking on operator selection, thus letting projection be terminated with success whenever an operator is selected, rather than waiting until the entire problem has been solved. As with the directness example, projection is terminated here after the first operator is selected (Figure 3.19(a)). However, in this case it is terminated with success as soon as the top block is moved to the table. The explanation for this success depends only on the explicit assumption of complete protection and on the fact that the operator was successfully applied, so a relatively general, positive control rule is learned (Figure 3.20). Although this is a positive rule, it also turns out to produce generalization to N, but now by always specifying that the one clear block that is not

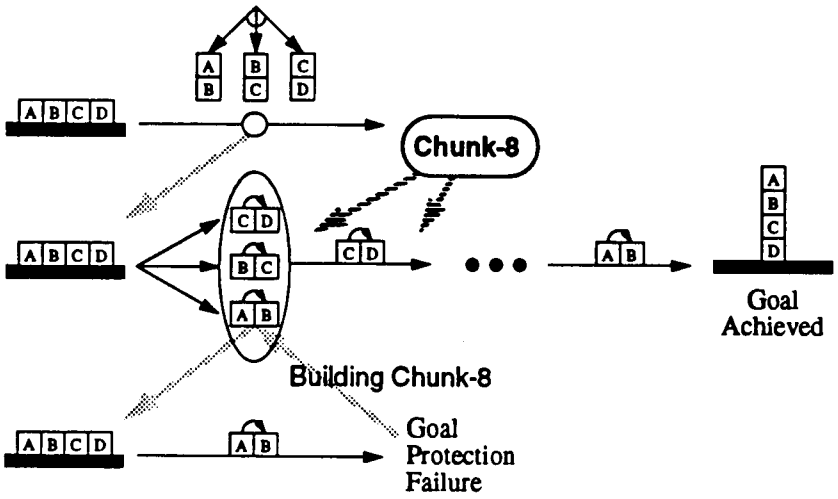


Figure 3.17: Four block stacking with protection

Chunk-8: If the problem-space is *blocks-world*
∧ Goal protection is assumed to hold
∧ There exists an active goal (on <x> <y>)
∧ There exists an active goal (on <y> <z>)
∧ (on <x> <y>) and (on <y> <z>) are not achieved
∧ <x> and <y> are blocks
∧ <x> and <y> are clear
∧ The proposed operator is (MOVE <x> <y>)
⇒
The operator is worst

Figure 3.18: A learned rule for four block stacking with protection.

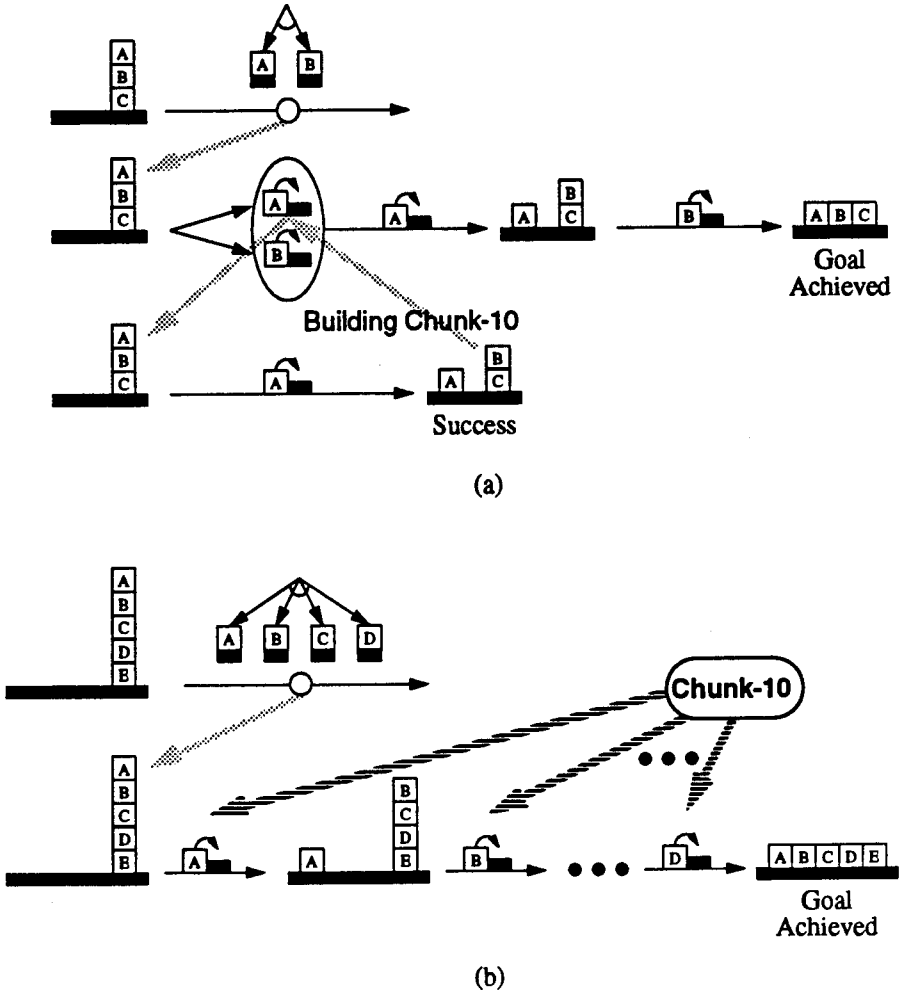


Figure 3.19: Three and five block unstacking with complete protection: (a) a projected path, (b) transfer of the learned rule to a different number.

Chunk-10: If the problem-space is *blocks-world*
∧ Complete protection holds
∧ There exists an active goal (on <x> Table)
∧ (on <x> Table) is not achieved
∧ <x> is clear
∧ The proposed operator is (MOVE <x> Table)
⇒
The operator is best

Figure 3.20: A learned rule for four block unstacking with complete protection.

already on the table — if it were already on the table, there would be no active goal conjunct for it — should be moved to the table. The resulting rule can transfer to any number of iterations, as shown in Figure 3.19(b).

The key to producing generalization to N with these biases is that they enable learning from non-iterative paths — in this way it is similar to Etzioni's [1990a] work on restricting EBL to learn from only non-recursive paths. In the directness and protection cases, the success paths are iterative, but (negative) rules can instead be learned from non-iterative failure paths. In the complete-protection case, learning occurs from a fragment of the success path that corresponds to just a single cycle of iteration. In both cases, the resulting rules can transfer to any number of iterations.

3.4 Experimental Results

Experimental results from the six planners in two planning domains — the blocks-world domain and the machine-shop scheduling domain — are shown in Tables 3.1 - 3.3. The data comes from running each planner on the same set of 100 problems for each domain. For each problem in the blocks-world domain, the number of blocks was randomly selected between three and four. Given the number of blocks, an initial state was randomly generated among the possible configurations of the blocks

and the table (3 configurations for 3 blocks, and 5 configurations for 4 blocks). The generated initial state was represented as a set of (on x_i y_i)-type predicates. Likewise a set of (on x_j y_j)-type goal conjuncts was randomly generated that numbered between two and the number of blocks in the initial state. For each goal conjunct, x_j was selected randomly from the initial set of blocks, and then y_j was selected randomly from among the table and the blocks which have not yet been selected as y_k $(k < j)$. The number of possible combinations of goal conjuncts for n-block problems is $O(n^n)$, because for each of the n blocks, there are n possible locations.

A task in the machine-shop scheduling domain is to determine a sequence of machining operations to produce the desired objects so as to meet the given requirements [Minton, 1988]. [4] The shop contains several machines, including `ROLL`, `LATHE`, `PUNCH`, `DRILL-PRESS`, `POLISH`, `GRINDER`, `SPRAY-PAINT`, and `IMMERSION--PAINT`. Each object has five attributes — `shape`, `has-hole`, `surface-condition`, `painted`, and `temperature`. Each attribute can have one of two to four types of values. For each problem, the initial state was generated by assigning a randomly generated type to each attribute for an object (except that the initial temperature is always `cold`). The number of goal conjuncts for each problem was fixed as five. The goal conjuncts for each problem were generated randomly as with the initial-state generation.

Learning was turned on for each problem, but only within-trial transfer was allowed; that is, rules learned during one problem were not used for other problems. Planning time is mainly measured in terms of *decisions*, the basic behavioral cycle in Soar. This measure is not quite identical to the more traditional measure of number of planning operators executed, but should still correlate with it relatively closely.

[4] The version of the machine-shop domain used in this research is almost identical to the original PRODIGY version presented in [Minton, 1988]. The only difference between the two versions is that the time augmentation for each generated operation in the original version is not specified in our version, because our main focus here is on the sequence of operations rather than the time when to execute the operations.

	No subgoal (Directness)	Local (Linear)	Global (Nonlinear)
	M_1	M_2	M_3
GP	68 (A_1)	95 (A_2)	96 (A_3)
	M_4	M_5	M_6
No GP	68 (A_4)	100 (A_5)	100 (A_6)

(a) Blocks world domain.

	No subgoal (Directness)	Local (Linear)	Global (Nonlinear)
	M_1	M_2	M_3
GP	70 (A_1)	70 (A_2)	70 (A_3)
	M_4	M_5	M_6
No GP	100 (A_4)	100 (A_5)	100 (A_6)

(b) Machine-shop scheduling domain.

Table 3.1: Number of problems solved.

		No subgoal (Directness)	Local (Linear)	Global (Nonlinear)
		M_1	M_2	M_3
GP	$A_1(A_4)$	8.63 (0.17)	15.06 (0.41)	16.24 (0.43)
	A_2	-	22.67 (0.92)	23.66 (0.73)
	A_3	-	-	23.60 (0.73)
	$A_5(A_6)$	-	-	-
		M_4	M_5	M_6
No GP	$A_1(A_4)$	12.34 (0.29)	22.21 (0.67)	33.40 (2.19)
	A_2	-	29.41 (1.06)	47.12 (3.66)
	A_3	-	29.48 (1.06)	48.06 (3.68)
	$A_5(A_6)$	-	29.22 (1.04)	47.93 (3.62)

(a) Blocks world domain.

		No subgoal (Directness)	Local (Linear)	Global (Nonlinear)
		M_1	M_2	M_3
GP	$A_1(A_2, A_3)$	20.73 (0.49)	20.73 (0.49)	20.73 (0.49)
	$A_4(A_5, A_6)$	-	-	-
		M_4	M_5	M_6
No GP	$A_1(A_2, A_3)$	31.47 (0.85)	31.47 (0.85)	31.47 (0.85)
	$A_4(A_5, A_6)$	33.97 (0.92)	33.97 (0.92)	33.97 (0.92)

(b) Machine-shop scheduling domain.

Table 3.2: Average number of decisions (and CPU time (sec.)) per problem.

		No subgoal (Directness)	Local (Linear)	Global (Nonlinear)
		M_1	M_2	M_3
GP	$A_1(A_4)$	1.82	1.85	2.00
	A_2	-	2.35	2.54
	A_3	-	-	2.54
	$A_5(A_6)$	-	-	-
		M_4	M_5	M_6
No GP	$A_1(A_4)$	1.82	3.00	2.90
	A_2	-	3.78	3.88
	A_3	-	3.83	4.07
	$A_5(A_6)$	-	3.82	4.14

(a) Blocks world domain.

		No subgoal (Directness)	Local (Linear)	Global (Nonlinear)
		M_1	M_2	M_3
GP	$A_1(A_2, A_3)$	2.43	2.43	2.43
	$A_4(A_5, A_6)$	-	-	-
		M_4	M_5	M_6
No GP	$A_1(A_2, A_3)$	4.13	4.13	4.13
	$A_4(A_5, A_6)$	4.47	4.47	4.47

(b) Machine-shop scheduling domain.

Table 3.3: Average plan length per problem.

Table 3.1 shows the number of problems solved by each cell's planner, as defined in Figure 3.11, for these two domains. The label A_i denotes the problem set that the method M_i implicitly defines. With a sufficient time limit, every problem solvable in principle by a planner was actually solved. Not surprisingly, the data show a monotonic relationship between planner bias and scope, from a low of 68 problems in the blocks-world domain and 70 problems in the machine-shop scheduling domain for the most restricted planner to a high of 100 problems in both domains for the least restricted planner.

Tables 3.2 and 3.3 show the average number of decisions, average CPU time, and average plan lengths — which should positively correlate with execution time — for each distinct problem sets defined in Table 3.1. In the standard blocks-world domain, four distinct problem sets are defined. This is because A_4 is the same as A_1 since if a problem is not solvable with protection, it also is not solvable with directness; and A_5 is the same as A_6 since both M_5 and M_6 are complete in this domain, though M_5 may not be able to generate an optimal solution. These four problem sets are associated with the four rows within each cell. In the machine-shop scheduling domain, no precondition subgoals are required because there is no operator which achieves any of the unmet preconditions. Thus both directness and linearity are irrelevant. However, there are strong interactions among the operators, so protection violations are still relevant. In consequence, two distinct problems sets are defined, A_1 and A_4.

The timing results are shown in Table 3.2. The two columns within each cell show the average number of decisions and the average CPU time, respectively, which are required to generate plans for the problems. The table shows that planning effort is also a monotonically decreasing function of the amount of bias along these dimensions (only for protection in the machine-shop scheduling domain). For example, for problem set A_1 in the blocks-world domain, effort ranged from a low of 8.63 decisions for the most biased method (that is, the direct goal-protection method) to a high of 33.40 decisions for the least biased method (that is nonlinear

planning without goal-protection). This trade off between efficiency and completeness implies that selecting an appropriate amount of bias for a given problem is critical for finding a solution quickly. Table 3.3 exhibits a similar monotonic relationship between plan length and the amount of bias used.

3.5 Summary

Six single methods are defined along two bias dimensions: goal-flexibility, and goal-protection. These methods are implemented in Soar, in which generated plans are represented as sets of control rules that jointly specify which operators should be executed at each point in time. The six implemented methods are compared empirically in terms of planner completeness, planning time, and plan length. The experimental results show a trade-off between completeness and efficiency. This implies that the planning system would be best served if it could always opt for the most restricted method adequate for its current situation.

Chapter 4

Multi-Method Planners

One of the main problems with the planners examined in the previous chapter is that each is either incomplete or performs a significant amount of excess work for some of the problems (both in planning and execution). An alternative approach is to build a *multi-method planner* which utilizes a coordinated set of planning methods, where each individual method has different scope and performance. The basic idea underlying this thesis is to select and coordinate a set of individual methods based on the empirical performance of those methods for a training set of problems.

Within the empirical multi-method planning framework, the main goal of this research is to create a set of multi-method planners which are more efficient and applicable than single-method planners. The previous chapter introduced a methodology to create individual methods which have different performance and scope based on the amount of bias used. Given a set of created methods, the key issue is then how to coordinate the methods in an efficient manner so that the multi-method planner can have high performance. Method coordination refers to (1) the selection of appropriate methods as situations arise, and (2) the granularity of method switching as the situational demands shift.

For method selection, individual methods need to be organized so that a higher level control structure can determine which method to use first, and which method to use next when the current method fails. Two straightforward ways of organizing

individual methods are a sequential and a time-shared manner. A sequential multi-method planner consists of a sequence of single-method planners. A time-shared multi-method planner consists of a set of single-method planners in which each method is active in turn for a given time slice [Barley, 1991]. In this thesis, a special type of sequential multi-method planning, called *monotonic multi-method planning* is focused on. In a monotonic multi-method planner, the single methods are sequenced according to increasing coverage and decreasing efficiency. With the assumptions that earlier methods terminate, and that methods which are efficient when they succeed do not waste too much time when they fail, monotonic multi-method planners can generate plans efficiently by using more restricted methods earlier in the sequence [Lee and Rosenbloom, 1992].

One way to construct a monotonic multi-method planner is to use the biases which themselves increase efficiency. Individual methods are sequenced so that the set of biases used in a method is a subset of the biases used in earlier methods, and the later methods have more coverage than the earlier methods. This means that planning starts by trying highly efficient methods, and then successively relaxing biases until a sufficient method is found. This type of planning is called *bias-relaxation multi-method planning.* A bias-relaxation multi-method planner is not necessarily a monotonic multi-method planner if there are interactions among biases. However, one can generate monotonic multi-method planners via bias-relaxation by just testing whether monotonicity holds for the created bias-relaxation multi-method planners. In bias-relaxation multi-method planning, each bias is evaluated independently by comparing a method which uses that bias only and a method which uses no bias. Thus, bias-relaxation multi-method planning has more restricted scope in creating and comparing individual methods than monotonic multi-method planning[1].

[1]Strongly-monotonic multi-method planning described in [Lee and Rosenbloom, 1993] also uses a bias-relaxation scheme. However, it evaluates each bias along with other biases, thus examining more methods than bias-relaxation multi-method planning.

The second issue of method coordination is the granularity at which individual methods are switched. This issue is important in terms of a planner's performance, because the performance of a multi-method planner can be changed according to the granularity of shifting control from method to method. Depending on the granularity of method switching, multi-method planners can be further specialized: *coarse-grained multi-method planners*, where methods are switched on a problem-by-problem basis; and *fine-grained multi-method planners*, where methods are switched on a goal-by-goal basis [Lee and Rosenbloom, 1993].

This chapter investigates these two issues of coordinating individual methods in multi-method planning in depth. To investigate the method organization issue, a scheme to construct monotonic multi-method planners from a set of single-method planners is provided, and then a formal model is presented to compare the performance of constructed monotonic multi-method planners with time-shared multi-method planners and single-method planners. Also, a scheme to construct a set of bias-relaxation multi-method planners is provided, and the constructed bias-relaxation multi-method planners are compared experimentally with single-method planners. To investigate the granularity of method switching, the performance of coarse-grained bias-relaxation multi-method planners and fine-grained bias-relaxation multi-method planners (called simply coarse-grained multi-method planners and fine-grained multi-method planners, respectively, throughout this thesis) are evaluated experimentally, and compared with the performance of single-method planners.

Partial-order planning is one of the most popular approaches in the planning literature. At the end of this chapter, multi-method planning is compared with partial-order planning in terms of planning performance.

4.1 Monotonic Multi-Method Planners

In a monotonic multi-method planner, individual methods are sequenced so that the earlier methods are more efficient and have less coverage than the later methods.

The idea is that if the biases used in efficient methods can prune the search space, the problems solvable by efficient methods should be solved more quickly, while problems requiring less bias should not waste too much extra time trying out the insufficient early methods.

This approach is inspired by iterative deepening [Korf, 1985]. In iterative deepening, a sequence of depth-first searches are performed, each to a greater depth than the previous one. If a solution is found at a shallow depth, the cost of searching to a greater depth is saved. If a solution is not found at a particular depth, a deeper search is performed. The cost of doing the shallower searches is then wasted, but since the deeper search costs at least β times the cost of the shallower search — where β is the branching factor of the search tree — this cost can be relatively quite small. Thus, if the proportion of problems solvable at shallow depths is large enough, and the ratio of costs for successive levels is large enough, there should be a net gain.

A monotonic multi-method planner can be defined formally by using a *restricted dominance relation* [Lee and Rosenbloom, 1992].

4.1.1 Restricted Dominance Relation

Let M_i be a single-method planner. Let A be a sample set of problems, and let $A_i \subseteq A$ be the subset of A which is solvable in principle by M_i. The functions $s(M_i, A_S)$ and $l(M_i, A_S)$ represent respectively the average cost that M_i requires to succeed and the average length of plans generated by M_i, for the problems in $A_S \subseteq A_i$. Similarly, $f(M_i, A_F)$ represents the average wasted cost for M_i to fail for the problems in $A_F \subseteq A - A_i$.

Given a set of methods $\{M_i\}$ $(i{=}1, ..., n)$, a restricted dominance relation $M_x \prec M_y$ is defined between two different single-method planners, M_x and M_y, if the following conditions hold:

(1) $A_x \subset A_y$

(2) $s(M_x, A_i) \leq s(M_y, A_i)$, for every $A_i \subseteq A_x$

Planner	Decisions			Plan length		
	A_1	A_2	A_5	A_1	A_2	A_5
M_1 (directness, protection)	12.50	-	-	1.56	-	-
M_2 (linearity, protection)	13.00	18.90	-	1.56	2.32	-
M_3 (protection)	13.21	26.91	-	1.62	2.49	-
M_4 (directness)	13.54	-	-	1.56	-	-
M_5 (linearity)	14.81	24.47	24.84	2.10	3.22	3.34
M_6	16.23	40.85	40.96	2.02	3.17	3.37

(a) Blocks world domain.

Planner	Decisions		Plan length	
	A_1	A_4	A_1	A_4
M_1 (directness, protection)	22.14	-	2.68	-
M_2 (linearity, protection)	22.14	-	2.68	-
M_3 (protection)	22.14	-	2.68	-
M_4 (directness)	35.33	37.42	4.36	4.68
M_5 (linearity)	34.27	36.10	4.45	4.82
M_6	34.27	36.10	4.45	4.82

(b) Machine-shop scheduling domain.

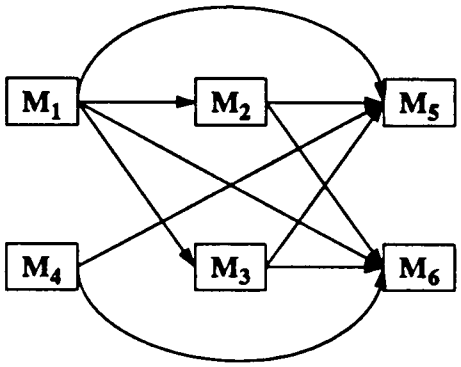
Table 4.1: The performance of the six single-method planners for the problem sets defined by the scopes of the planners.

---

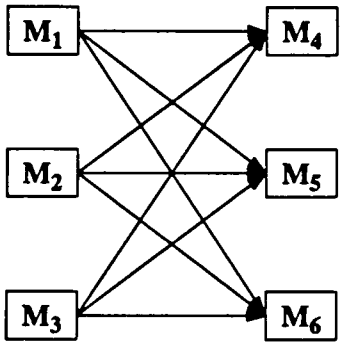
(3) $l(M_x, A_i) \leq l(M_y, A_i)$, for every $A_i \subseteq A_x$.

A sequential multi-method planner which consists of n different single-method planners is denoted as $M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}$. A sequential multi-method planner $M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}$ is called *monotonic* if $M_{k_i} \prec M_{k_{i+1}}$ holds for each $i = 1, ..., n-1$.

The straightforward way to build monotonic multi-method planners is to run each of the individual methods on a set of training problems, and then from the resulting data to generate method sequences for which monotonicity holds. Table 4.1 shows the average number of decisions, $s(M_{k_i}, A_{k_j})$, and the average plan lengths, $l(M_{k_i}, A_{k_j})$, over a training problem set for the six single-method planners



(a) Blocks-world domain



(b) Machine-shop scheduling domain

Figure 4.1: Restricted dominance graphs for the single-method planners.

defined in Chapter 3, for the blocks-world domain and the machine-shop scheduling domain.

For each domain, the problem set consists of 30 problems which are randomly generated as in Chapter 3. In the blocks-world domain, A_2 and A_3 are different sets in principle, because problems such as Sussman's anomaly cannot be solved by a linear planner with protection (M_2) but can be by a nonlinear planner with protection (M_3). However, among the 30 training problems, these "anomaly" problems did not occur, yielding $A_2 = A_3$ for this set of problems.

Figure 4.1 exhibits restricted dominance graphs based on the results in Table 4.1. Each node in a graph represents a single-method planner, and an arc from M_x to M_y implies that $M_x \prec M_y$ holds. Thus every path in the graph corresponds to a monotonic multi-method planner. A monotonic multi-method planner $M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}$ is complete, if M_{k_n} is complete. In the blocks-world domain, seven complete 2-method planners and four complete 3-method planners can be constructed, whereas in the machine-shop scheduling domain, nine complete 2-method planners can be constructed.

The next section compares a monotonic multi-method planner with its corresponding time-shared multi-method planner and single-method planner in terms of planning time and plan length.

4.1.2 Performance Analysis

Planning time: In this section, it is assumed that the individual methods in a sequential multi-method planner are switched on a problem-by-problem basis. For a given problem $a \in A$, the planning time of a sequential multi-method planner $M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}$, where M_{k_i} is the first method which solves a, can be represented as

$$s(M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}, \{a\}) = s(M_{k_i}, \{a\}) + \sum_{j=1}^{i-1} f(M_{k_j}, \{a\}), \qquad (4.1)$$

where $s(M_{k_i}, \{a\})$ is the cost for M_{k_i} to solve a, and $\sum_{j=1}^{i-1} f(M_{k_j}, \{a\})$ is the sum of the costs for inappropriate earlier methods to fail for a.

The corresponding time-shared multi-method planner consists of the same set of single methods, denoted as $M_{k_1} \| M_{k_2} \| ... \| M_{k_n}$. Let M_{k_i} be the first method that solves a in a horse-race manner. Suppose that the switching in a time-shared multi-method planner is based on a unit time slice. Then, the expected planning time of $M_{k_1} \| M_{k_2} \| ... \| M_{k_n}$ for a problem $a \in A$ can be represented as

$$s(M_{k_1} \| M_{k_2} \| ... \| M_{k_n}, \{a\}) = s(M_{k_i}, \{a\}) + \sum_{j=1, j \neq i}^{n} min[f(M_{k_j}, \{a\}), s(M_{k_i}, \{a\})], \qquad (4.2)$$

where the first term is the cost for the method that actually solves a, and the second term is the sum of the costs for the rest of the methods either to fail for a ($f(M_{k_j}, \{a\})$) or to try to solve a ($s(M_{k_i}, \{a\})$).

The average planning time for a problem set A can be represented by using a probability function. Let $P_{k_i} (= |A_{k_i}|/|A|)$ be the probability that an arbitrary problem in A is solvable by M_{k_i}. Let M_{k_0} be a null planner which cannot solve any problem; that is $A_{k_0} = \phi$ and $P_{k_0} = 0$. Let $A'_{k_i} = A_{k_i} - A_{k_{i-1}}$, for $1 \leq i \leq n$, be the set of problems which are solvable by M_{k_i} but not by $M_{k_{i-1}}$, and let $P'_{k_i} = |A'_{k_i}|/|A|$, for $1 \leq i \leq n$. Let $s(M_{k_0}, A_S) = l(M_{k_0}, A_S) = 0$, for any A_S, and $f(M_{k_0}, A_F) = 0$, for any A_F.

For a problem set A, the planning time of a complete sequential muli-method planner $M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}$ can be rewritten as the sum of the average planning time for the disjoint problem sets $A'_{k_i} (1 \leq i \leq n)$:

$$s(M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}, A) = \sum_{i=1}^{n} (P'_{k_i} * s(M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}, A'_{k_i})),$$

where

$$s(M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}, A'_{k_i}) =$$
$$s(M_{k_i}, A'_{k_i}) + \sum_{j=1}^{i-1} f(M_{k_j}, A'_{k_i}). \qquad (4.3)$$

The planning time of the corresponding time-shared multi-method planner can be rewritten as

$$s(M_{k_1} \| M_{k_2} \| ... \| M_{k_n}, A) = \sum_{i=1}^{n} (P'_{k_i} * s(M_{k_1} \| M_{k_2} \| ... \| M_{k_n}, A'_{k_i})),$$

where

$$s(M_{k_1} \| M_{k_2} \| ... \| M_{k_n}, A'_{k_i}) =$$
$$s(M_{k_i}, A'_{k_i}) + \sum_{j=1, j \neq i}^{n} \frac{\sum_{a \in A'_{k_i}} min[f(M_{k_j}, \{a\}), s(M_{k_i}, \{a\})]}{|A'_{k_i}|}. \qquad (4.4)$$

The relative performance between a complete sequential multi-method planner and the corresponding time-shared multi-method planner depends on the ordering of the methods and the cost of $f(M_{k_j}, \{a\})$ and $s(M_{k_i}, \{a\})$. If $M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}$

is monotonic, later methods ($M_j, j{=}i{+}1, ..., n$) would not fail to solve a, from the definition. Thus,

$$s(M_{k_1} \| M_{k_2} \| ... \| M_{k_n}, A'_{k_i}) =$$

$$((n-i+1) * s(M_{k_i}, A'_{k_i}) + \sum_{j=1}^{i-1} \frac{\sum_{a \in A'_{k_i}} min[f(M_{k_j}, \{a\}), s(M_{k_i}, \{a\})]}{|A'_{k_i}|}. \quad (4.5)$$

In particular, if $f(M_{k_j}, \{a\}) < s(M_{k_i}, \{a\})$ $(1 \leq j \leq i-1)$ for each a, we have

$$s(M_{k_1} \| M_{k_2} \| ... \| M_{k_n}, A'_{k_i}) =$$

$$((n-i+1) * s(M_{k_i}, A'_{k_i}) + \sum_{j=1}^{i-1} f(M_{k_j}, A'_{k_i}). \quad (4.6)$$

Thus, the performance difference between a monotonic multi-method planner and the corresponding time-shared multi-method planner is

$$s(M_{k_1} \| M_{k_2} \| ... \| M_{k_n}, A) - s(M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}, A) =$$

$$\sum_{i=1}^{n} [P'_{k_i} * (n-i) * s(M_{k_i}, A'_{k_i})] \geq 0. \quad (4.7)$$

This implies that if the cost of failure for a restricted planner is always less than the cost of success for a more relaxed planner, then monotonic multi-method planners outperform corresponding time-shared multi-method planners; otherwise, time-shared multi-method planners may perform better. This all depends on the relative search space size for the restricted planner and the density and distribution of solutions in the search space. However, if the biases used in a restricted method are strong enough to cut off all the failure paths at shallow depths, the cost to determine whether a method fails may be less than the cost to determine whether a method succeeds. Moreover, if the rule learned from a failure path can transfer to other failure paths, the cost of failure can be even less.

The performance of monotonic multi-method planners and single-method planners is compared as follows. For each monotonic multi-method planner $M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}$, there is a corresponding single-method planner M_{k_n} which has the same coverage of solvable problems. If $M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}$ is complete, M_{k_n} is also

complete. We compare a complete monotonic multi-method planner with its corresponding single-method planner in terms of planning time.

The performance of M_{k_n} is

$$s(M_{k_n}, A) = \sum_{i=1}^{n} [P'_{k_i} * s(M_{k_n}, A'_{k_i})]. \tag{4.8}$$

To compare the performance of $M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}$ with M_{k_n}, it is necessary to subtract (4.3) from (4.8), yielding

$$s(M_{k_n}, A) - s(M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}, A) =$$

$$\sum_{i=1}^{n} [P'_{k_i} * (s(M_{k_n}, A'_{k_i}) - s(M_{k_i}, A'_{k_i}) - \sum_{j=1}^{i-1} f(M_{k_j}, A'_{k_i}))]. \tag{4.9}$$

This means that if the performance gain by using a cheaper method ($s(M_{k_n}, A'_{k_i}) - s(M_{k_i}, A'_{k_i})$) is greater than the wasted time from using inappropriate methods ($\sum_{j=1}^{i-1} f(M_{k_j}, A'_{k_i})$) in a monotonic multi-method planner, then it is preferable to use that method over the single-method planner; otherwise, the single-method planner is preferred (at least where planning time is concerned).

Plan length: The plan length l for a complete monotonic multi-method planner and for the corresponding time-shared multi-method planner is the same and equal to

$$l(M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}, A) = l(M_{k_1} \| M_{k_2} \| ... \| M_{k_n}, A) =$$

$$\sum_{i=1}^{n} [P'_{k_i} * l(M_{k_i}, A'_{k_i})], \tag{4.10}$$

while the plan length for the corresponding single-method planner M_{k_n} is

$$l(M_{k_n}, A) = \sum_{i=1}^{n} [P'_{k_i} * l(M_{k_n}, A'_{k_i})]. \tag{4.11}$$

Since $M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}$ is monotonic, then $l(M_{k_i}, A'_{k_i}) \leq l(M_{k_n}, A'_{k_i})$. Therefore, the lengths of plans generated from a monotonic multi-method planner and the corresponding time-shared multi-method planner are always less than or equal to the length of plans generated from the corresponding single-method planner.

4.1.3 Learning in Multi-Method Planning

The analytical results in the previous section show that monotonicity can yield performance gain in sequential multi-method planning by using cheaper methods earlier in the sequence. The performance of sequential multi-method planning can be further improved by ameliorating the effects of wasting effort on insufficient planners via learning — in particular, of two sorts. The first sort of learning is within-planner learning that can transfer across planners (possibly for the same problem). If a projection is performed within one planner, and the results of the projection depend only on aspects of the planner that are shared by a second planner, then it should not be necessary to repeat that projection when the second planner is tried. For example, a rule learned from a plan violating goal protection in the direct goal-protection planner should be able to transfer to the nonlinear goal-protection planner, where it prevents the planner from reprojecting along paths that violate goal protection.

The second sort of learning is about which methods to use for which classes of problems. To the extent that this can be done, the effort wasted in trying inadequate methods can be avoided in the future. In our Soar-based implementation, bias selection is structured just as would be any other selection, so this sort of learning can happen automatically by chunking. From an experiment with such learning, Figure 4.2 shows a rule learned to avoid using the most restricted method — that is, direct goal-protection — under specific circumstances where there is only one active goal conjunct but (at least) two blocks must be moved to achieve it. This rule was learned during the first problem and can be used in three later problems to avoid even trying this method.

Though we have examined instances of learning about which methods to use for which classes of problems in the context of multi-method planning, no systematic study has yet been made of their effectiveness or of whether issues of overgeneralization and/or undergeneralization will prove troublesome, which they are likely to be. Another reason why this form of learning is not used is that the current multi-attribute encoding creates some expensive chunks for some of the problems.

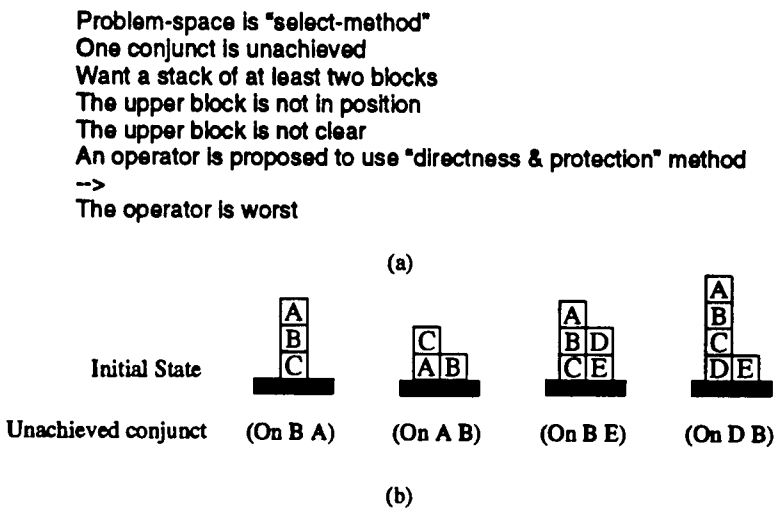


Figure 4.2: Example of learning which planners to use for which classes of problems: (a) a learned rule to avoid the direct goal-protection planner, (b) a class of problems in which this rule is applicable.

Future work should include rerunning the experiments summarized in Table 4.1 with this form of learning enabled.

4.2 Bias-Relaxation Multi-Method Planners

Section 4.1.1 showed an approach to creating monotonic multi-method planners by using a restricted dominance graph. This approach is quite straightforward in the sense that each pair of methods is directly compared. However, given a set of methods, it does not specify which methods should be created and which pairs of methods should be compared. Let k be the number of biases. Then, the number of single methods generated by every combination of these biases is $\mathcal{O}(2^k)$. The number of comparisons for creating a restricted dominance graph for these methods is $\mathcal{O}(2^{k^2})$. Although it is tractable to generate all monotonic multi-method planners by using a restricted dominance graph with a small set of initial biases — as in the

case of Section 4.1.1 — it may not be tractable if the number of biases considered is increased. Therefore, a scheme is needed to restrict the scope of methods to be generated and compared.

One way to remedy this problem is to compare the effectiveness of each bias in isolation, instead of comparing the performance of methods which are generated with respect to combinations of these biases. The approach presented in this section is based on bias-relaxation [Lee and Rosenbloom, 1993]. **Bias-relaxation** multi-method planners can be created as combinations of *effective* **biases** only, so that later methods can embody subsets of the effective biases incorporated into earlier methods[2]. Method switching is implemented by relaxing some of these biases; that is, planning starts with a set of effective biases, and then successively relaxes one or more biases until a solution is found within the method. This can be formalized as follows.

Let B_{k_i} be the set of biases used in M_{k_i}. A bias b is called *effective* in a problem set A and a method set $\{M_{k_i}\}$, if for a pair of methods M_{k_x} and M_{k_y} in $\{M_{k_i}\}$ such that $B_{k_x} = \{b\}$ and $B_{k_y} = \phi$,

$$(1)\quad s(M_{k_x}, A_{k_x}) \leq s(M_{k_y}, A_{k_x}), \text{ and}$$
$$(2)\quad l(M_{k_x}, A_{k_x}) \leq l(M_{k_y}, A_{k_x}).$$

A sequential multi-method planner $M_{k_1} \rightarrow M_{k_2} \rightarrow ... \rightarrow M_{k_n}$ is called a *bias-relaxation* multi-method planner, if

$$(1)\quad B_{k_{i-1}} \supset B_{k_i}, \text{ for } 2 \leq i \leq n, \text{ and}$$
$$(2)\quad B_{k_{i-1}} - B_{k_i} \text{ consists of effective biases only, for } 2 \leq i \leq n.$$

Given a set of k biases, the time complexity of testing whether these biases are effective or not is $\mathcal{O}(k)$ (by factoring out the complexity of solving problems), which is exponentially smaller than $\mathcal{O}(2^{k^2})$.

[2]Positive bias, as defined in [Lee and Rosenbloom, 1993], is a different notion from effective bias here in that the effectiveness of a positive bias is evaluated along with other biases.

The results in Table 4.1 imply that directness and protection are effective in the blocks-world domain, while linearity is not, since $l(M_5, A_1) > l(M_6, A_1)$ and $l(M_5, A_2) > l(M_6, A_2)$. If one uses linearity as an independent bias — so that one set of multi-method planners is generated using it and one set without it — and vary directness and protection within the individual multi-method planners, we get a set of ten different bias-relaxation multi-method planners (four three-method planners and six two-method planners) as shown in Table 4.2. In the machine-shop scheduling domain, only protection is effective. In consequence, only one bias-relaxation multi-method planner is generated $M_1 \rightarrow M_4$.

Type	Multi-method planners	Type	Multi-method planners
Linear	$M_1 \rightarrow M_2 \rightarrow M_5$	Nonlinear	$M_1 \rightarrow M_3 \rightarrow M_6$
	$M_1 \rightarrow M_4 \rightarrow M_5$		$M_1 \rightarrow M_4 \rightarrow M_6$
	$M_1 \rightarrow M_5$		$M_1 \rightarrow M_6$
	$M_2 \rightarrow M_5$		$M_3 \rightarrow M_6$
	$M_4 \rightarrow M_5$		$M_4 \rightarrow M_6$

Table 4.2: Ten bias-relaxation multi-method planners in the blocks-world.

Note that if there are no interactions among effective biases, a bias-relaxation multi-method planner is a special case of a monotonic multi-method planner. However, this is not necessarily true if there are interactions among them. For example, although directness is effective, this does not necessarily mean that the method that uses directness and protection is more efficient than the method that uses protection only.

In order to generate monotonic multi-method planners via bias-relaxation, one can just test whether monotonicity holds for the created bias-relaxation multi-method planners. The time complexity for this procedure is linear in terms of the number of biases, because at least one bias is relaxed whenever a method is switched.

In the next section, the experimental results for all of the created bias-relaxation multi-method planners are presented.

Planner	Decisions				Plan length			
	A_1	A_2	A_3	A_5	A_1	A_2	A_3	A_5
M_5	22.21	29.41	29.48	29.22	3.00	3.78	3.83	3.82
M_6	33.40	47.12	48.06	47.93	2.90	3.88	4.07	4.14
Average				38.58				3.98
$M_1 \rightarrow M_2 \rightarrow M_5$	13.26	24.69	25.07	26.13	1.82	2.48	2.54	2.58
$M_1 \rightarrow M_3 \rightarrow M_6$	13.26	26.34	26.55	28.91	1.82	2.52	2.54	2.59
$M_1 \rightarrow M_4 \rightarrow M_5$	13.26	26.16	26.41	26.79	1.82	2.85	**2.92**	2.94
$M_1 \rightarrow M_4 \rightarrow M_6$	13.26	36.78	37.40	37.30	1.82	2.91	**2.99**	3.02
$M_1 \rightarrow M_5$	13.26	25.68	25.86	26.04	1.82	2.96	**3.02**	3.03
$M_1 \rightarrow M_6$	13.26	31.54	31.85	31.77	1.82	2.89	2.94	2.97
$M_2 \rightarrow M_5$	19.54	27.89	28.18	29.34	1.85	2.43	2.49	2.58
$M_3 \rightarrow M_6$	21.22	28.46	28.41	30.67	2.00	2.52	2.52	2.57
$M_4 \rightarrow M_5$	16.85	27.81	27.95	28.38	1.82	2.83	2.88	2.93
$M_4 \rightarrow M_6$	16.85	33.33	33.59	34.47	1.82	2.83	2.85	2.95
Average				29.98				2.82

(a) Blocks-world domain.

Planner	Decisions		Plan length	
	A_1	A_4	A_1	A_4
M_4, M_5, M_6	31.47	33.97	4.13	4.47
$M_1 \rightarrow M_4$, $M_2 \rightarrow M_5$, $M_3 \rightarrow M_6$	26.17	35.91	2.43	3.58

(a) Machine-shop scheduling domain.

Table 4.3: Single-method and bias-relaxation multi-method planning.

4.2.1 Experimental Results

We have implemented the ten bias-relaxation multi-methods planners in Soar6. Each single-method planner in a bias-relaxation multi-method planner was implemented as a specialization of a general problem-space. Based on the sequence of single-method planners, a set of meta-level control rules was provided to coordinate which problem-space is tried next if the current problem-space does not generate a plan for the given problem. Only within-trial learning was turned on for each problem, as in the experiments with the single-method planners, but learned rules

were also allowed to transfer from an earlier method to a later method (for the same problem). This is equivalent to the type of transfer allowed in the single-method planners, because the scope of transfer is limited to the current trial only.

Table 4.3 compares the ten bias-relaxation multi-method planners with the two complete single-method planners over the test set of 100 randomly generated problems used in Chapter 3 (this test set is different from the 30-problem training set used in developing the multi-method planners). Paired-sample Z-tests are made for the average performance on A_5 — because it is the only complete problem set in this domain — between bias-relaxation multi-method planners and single-method planners. The results reveal that bias-relaxation multi-method planners take significantly less planning time (z=2.27, p<.05), and generate significantly shorter plans than single-method planners (z=4.86, p<.01). In the machine-shop scheduling domain, paired-sample Z-tests are made for the average performance on A_4. The results show that bias-relaxation multi-method planners take slightly more planning time than single-method planners; however, no significance is found at a 5% level (z=1.00). In terms of plan length, bias-relaxation multi-method planners generate significantly shorter plans than single-method planners (z=3.15, p<.01) in this domain also.

Although it has been shown that bias-relaxation multi-method planners can outperform single-method planners (in the blocks-world domain), it does not necessarily mean that, for all situations, there exists a bias-relaxation multi-method planner which outperforms the most efficient single-method planner. In fact, the performance of these planners depends on the biases used in the bias-relaxation multi-method planners and the problem set used in the experiments. For example, if the problems are so complex that most of the problems are solvable only by the least restricted method, the performance loss by trying inappropriate earlier methods in multi-method planners might be relatively considerable. On the other hand, if the problems are so trivial that it takes only a few decisions for the least restricted method to solve the problems, the slight performance gain by using more

restricted methods in multi-method planners might be overridden by the complexity of the meta-level processing required to coordinate the sequence of primitive planners.

4.3 Fine-Grained Multi-Method Planners

The approach to multi-method planning described so far starts with a restricted method and switches to a less restricted method whenever the current method fails. This switch is always made on a problem-by-problem basis. However, this is not the only granularity at which methods could be switched. The family of multi-method planning systems can be viewed on a granularity spectrum. While in coarse-grained multi-method planners, methods are switched for a whole problem when no solution can be found for the problem within the current method, in fine-grained multi-method planners (denoted as $M_{k_1 \rightarrow k_2 \rightarrow \ldots \rightarrow k_n}$), methods can be switched at any point during a problem at which a new set of subgoals is formulated, and the switch only occurs for that set of subgoals (and not for the entire problem) [Lee and Rosenbloom, 1993]. At this finer level of granularity it is conceivable that the planner could use a highly-restricted and efficient method over much of a problem, but fall back on a nonlinear method without protection for those critical subregions where there are tricky interactions.

With this flexibility of method switching, fine-grained multi-method planning can potentially outperform both coarse-grained multi-method planning and single-method planning. Compared with coarse-grained multi-method planning, it can save the effort of backtracking when the current method can not find a solution or the current partial plan violates the biases used in the current method. Moreover, it can save the extra effort of using a less restricted method on later parts of the problem, just because one early part requires it. As compared with single-method planning, a fine-grained multi-method planner can utilize biases which would cause incompleteness in a single-method planner — such as directness or protection in the blocks-world domain — while still remaining complete. The result is that a

Planner	Decisions				Plan length			
	A_1	A_2	A_3	A_5	A_1	A_2	A_3	A_5
M_5	22.21	29.41	29.48	29.22	3.00	3.78	3.83	**3.82**
M_6	33.40	47.12	48.06	47.93	2.90	3.88	4.07	**4.14**
Average				38.58				**3.98**
$M_1 \rightarrow M_2 \rightarrow M_5$	13.26	24.69	25.07	26.13	1.82	2.48	2.54	2.58
$M_1 \rightarrow M_3 \rightarrow M_6$	13.26	26.34	26.55	28.91	1.82	2.52	2.54	2.59
$M_1 \rightarrow M_4 \rightarrow M_5$	13.26	26.16	26.41	26.79	1.82	2.85	2.92	2.94
$M_1 \rightarrow M_4 \rightarrow M_6$	13.26	36.78	37.40	37.30	1.82	2.91	2.99	3.02
$M_1 \rightarrow M_5$	13.26	25.68	25.86	26.04	1.82	2.96	3.02	3.03
$M_1 \rightarrow M_6$	13.26	31.54	31.85	31.77	1.82	2.89	2.94	2.97
$M_2 \rightarrow M_5$	19.54	27.89	28.18	29.34	1.85	2.43	2.49	2.58
$M_3 \rightarrow M_6$	21.22	28.46	28.41	30.67	2.00	2.52	2.52	2.57
$M_4 \rightarrow M_5$	16.85	27.81	27.95	28.38	1.82	2.83	2.88	2.93
$M_4 \rightarrow M_6$	16.85	33.33	33.59	34.47	1.82	2.83	2.85	2.95
Average				29.98				2.82
$M_{1 \rightarrow 2 \rightarrow 5}$	8.63	12.87	13.00	13.01	1.82	2.80	2.84	2.90
$M_{1 \rightarrow 3 \rightarrow 6}$	8.63	13.38	13.43	13.56	1.82	2.53	2.53	2.59
$M_{1 \rightarrow 4 \rightarrow 5}$	8.63	13.19	13.29	13.25	1.82	3.25	3.32	3.34
$M_{1 \rightarrow 4 \rightarrow 6}$	8.63	13.48	13.73	13.63	1.82	2.87	2.96	2.97
$M_{1 \rightarrow 5}$	8.63	12.21	12.36	12.51	1.82	2.63	2.73	2.81
$M_{1 \rightarrow 6}$	8.63	13.22	13.27	13.23	1.82	2.68	2.69	2.73
$M_{2 \rightarrow 5}$	19.19	23.75	23.76	23.80	2.56	3.07	3.11	3.16
$M_{3 \rightarrow 6}$	16.62	23.45	23.56	24.22	2.03	2.56	2.57	2.71
$M_{4 \rightarrow 5}$	13.57	17.24	17.30	17.38	2.44	3.71	3.77	3.77
$M_{4 \rightarrow 6}$	14.10	19.28	19.58	19.83	2.41	3.33	3.43	3.46
Average				16.44				3.04

Table 4.4: Single-method and coarse-grained multi-method vs. fine-grained multi-method planning in the blocks-world domain.

			Planning Type	
		Average Decisions	Single	Coarse-Grained
Planning Type	Single	38.58	-	-
	Coarse-Grained	29.98	2.27*	-
	Fine-Grained	16.44	5.37**	6.72**

(a) Decisions.

			Planning Type	
		Average Plan Length	Single	Coarse-Grained
Planning Type	Single	3.98	-	-
	Coarse-Grained	2.82	4.86**	-
	Fine-Grained	3.04	3.42**	1.77

(b) Plan length

Table 4.5: Significance test results for the blocks-world domain.

fine-grained multi-method planner can potentially be more efficient than a single-method planner that has the same coverage of solvable problems.

4.3.1 Experimental Results

Table 4.4 compares the bias-relaxation fine-grained multi-method planners with the corresponding bias-relaxation coarse-grained multi-method planners and (complete) single-method planners over the same 100 test set as used in Table 4.3 in the blocks-world domain. Paired-sample Z-tests on this data, as shown in Table 4.5, reveal that fine-grained multi-method planners take significantly less planning time than both single-method planners (z=5.35, p<.01) and coarse-grained multi-method planners (z=6.72, p<.01). This likely stems from fine-grain multi-method planners preferring to search within the more efficient spaces defined by the biases — thus tending to outperform single-method planners — but being able

Planner	Decisions		Plan length	
	A_1	A_4	A_1	A_4
M_4, M_5, M_6	31.47	33.97	4.13	4.47
$M_1 \rightarrow M_4$, $M_2 \rightarrow M_5$, $M_3 \rightarrow M_6$	26.17	35.91	2.43	3.58
$M_{1 \rightarrow 4}$, $M_{2 \rightarrow 5}$, $M_{3 \rightarrow 6}$	18.71	19.07	2.87	3.29

(a) Experimental results for the scheduling domain.

Table 4.6: Single-method and coarse-grained multi-method vs. fine-grained multi-method planning in the machine-shop scheduling domain.

to recover from bias failure without throwing away everything already done for a problem (thus tending to outperform coarse-grained multi-method planners).

Fine-grained multi-method planners also generate significantly shorter plans than single-method planners (z=3.42, p<.01). They generate slightly longer plans than coarse-grained multi-method planners; however, no significance is found at a 5% level (z=1.77). These results likely arise because, whenever possible, both types of multi-method planners use the more restrictive methods that yield shorter plan lengths, while there may be little difference between the methods that ultimately succeed for the two types of multi-method planners.

Table 4.6 illustrates the performance of these three types of planners over the same test set of 100 problems used in Table 4.3 in the machine-shop scheduling domain. As with the blocks-world domain, paired-sample z-tests in the scheduling domain, as shown in Table 4.7, indicate that fine-grained planners dominate both single-method planners (z=10.91, p<.01) and coarse-grained planners (z=8.95, p<.01) in terms of planning time. Fine-grained planners also generate significantly shorter plans than do the single-method planners (z=6.49, p<.01). They generate slightly shorter plans than coarse-grained multi-method planners; however, no significance is found at a 5% level (z=1.28).

Figures 4.3 and 4.4 plot the average number of decisions versus the average plan lengths for the data in Tables 4.4 and 4.6. These figures graphically illustrate

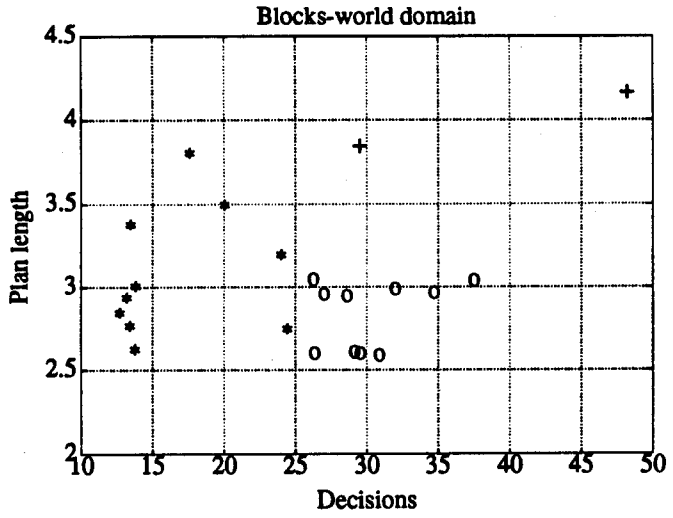


Figure 4.3: Performance of single-method planners (+), coarse-grained multi-method planners (o), and fine-grained multi-method planners (*) in the blocks-world domain.

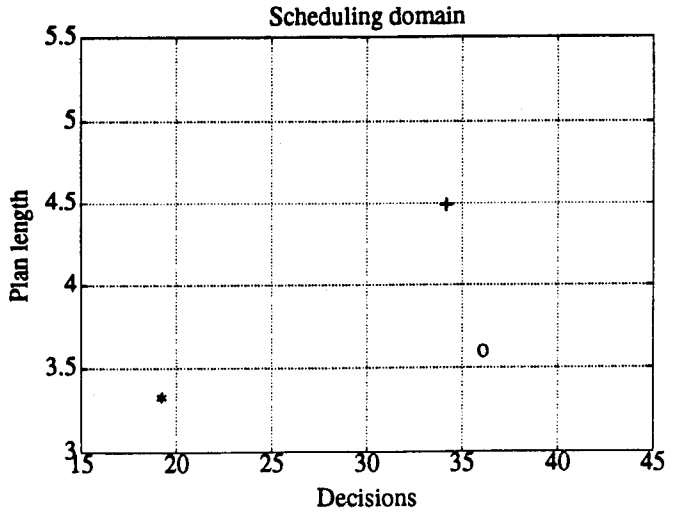


Figure 4.4: Performance of single-method planners (+), coarse-grained multi-method planners (o), and fine-grained multi-method planners (*) in the scheduling domain.

		Average Decisions	Planning Type: Single	Planning Type: Coarse-Grained
Planning Type	Single	33.97	-	-
	Coarse-Grained	35.91	1.00	-
	Fine-Grained	19.07	10.91**	8.95**

(b) The significance test result for decisions.

		Average Plan Length	Planning Type: Single	Planning Type: Coarse-Grained
Planning Type	Single	4.47	-	-
	Coarse-Grained	3.58	3.15**	-
	Fine-Grained	3.29	6.49**	1.28

(c) The significance test result for plan lengths.

Table 4.7: Significance test results for the machine-shop scheduling domain.

how the coarse-grained approach primarily reduces plan length in comparison to the single-method approach, and how the fine-grained approach primarily improves efficiency in comparison to the coarse-grained approach.

4.4 Comparison with Partial-Order Planning

Partial-order planning can be more efficient than total-order planning, because partial-order planning avoids premature commitment to an incorrect ordering between operators, and thus reduces the size of the search space [Minton *et al.*, 1991, Barrett and Weld, 1992]. In particular, Barrett and Weld [1992] showed experimentally that the total-order planner TOCL exhibited apparently exponential time complexity while the partial-order planner POCL maintained near-linear performance, as the number of problems was increased in the D^1S^1 domain. This section compares multi-method planning with partial order planning, and shows that multi-method planners can perform as well as partial-order planners in this domain.

A template for generating operators in D^1S^1 is illustrated as follows:

$$(define\text{-}operator \ : action \ A_i \ : precondition \ \{I_i\} \ : add \ \{G_i\} \\ : delete \ \{I_{i-1}\}).$$

Note that operator A_i deletes the preconditions of operator A_{i-1}[3]. This implies that for any problem in this domain, there exists a single ordering of operators which solves that problem.

Planning in TOCL is similar to planning in M_6 (without learning) in the sense that both are complete and search over the space of sequences of task operators. In the search, operators can be tried which are not provably right, and backtracking can occur if the choice is wrong. In contrast, POCL searches over the space of ordering constraints. It tries out constraints that may be right, and then backtracks over them if they prove wrong. In general, POCL can outperform TOCL by avoiding premature step-ordering constraints.

The performance of TOCL can be improved by adding EBL. The idea is that if a control rule is learned by EBL when a failure occurs and this rule can cut off all similar failure paths, then TOCL may perform as well as POCL. However, this may not lead to linear performance, because EBL is committing even less than least commitment planning with respect to adding constraints. In EBL, constraints (i.e. preference rules) are generated only when they are provably correct, and they are never backtracked over. Proving that a constraint is correct can be a non-trivial task, and may not guarantee a polynomial complexity in planning time. On the other hand, in POCL the added constraints are not proved correct. In the D^1S^1 domain, however, the constraints generated by POCL do work without backtracking, since there is no operator which adds a precondition of another operator.

Bias-relaxation multi-method planning can improve the performance of TOCL, because a bias allows learning constraints based on weaker proofs if they prove wrong, and multi-method planning allows backtracking over these constraints. In

[3] D^1S^1 means that there is one entry in its operator's delete set and it only takes one step to achieve a goal.

this domain, a bias, called *precondition protection* is used, which eliminates all plans in which disachieved preconditions are reachieved. A multi-method planner can be constructed which consists of a method using precondition protection (denoted as M_p) and one without it — that is, the least restricted planner (M_6). In fact, a precondition protection bias is weak enough to solve all problems (thus M_p is complete) in this domain. However, M_p itself may not be complete for other domains. In that case, Backtracking across incorrect preference rules (that is across methods) can happen in $M_p \rightarrow M_6$.

	$M_p \rightarrow M_6$		POCL	
Number of Goals	Number of Nodes (S)	CPU Time (Sec.)	Number of Nodes ($S' + O$)	CPU Time (Sec.)
1	1	0.07	2	0.03
2	2	0.09	4	0.05
3	3	0.09	6	0.06
4	4	0.09	8	0.10
5	5	0.13	10	0.13
6	6	0.56	12	0.15
7	7	0.12	14	0.17
8	8	0.14	16	0.19
9	9	0.17	18	0.23
10	10	0.18	20	0.26
11	11	0.20	22	0.31
12	12	0.21	24	0.31
13	13	0.24	26	0.33

Table 4.8: Experimental results for $M_p \rightarrow M_6$ and POCL

Table 4.8 shows experimental results for $M_p \rightarrow M_6$ and POCL for 13 problems in the D^1S^1 domain. In terms of the number of nodes visited, both planners show linear performance. Note that the definitions of node in the two planners are different because their search spaces are different. In $M_p \rightarrow M_6$, a node represents an element of the space of operator sequences (S), whereas in POCL a node represents an element of the space of the set of operators (S') plus the set of ordering

constraints among them (O). This explains the factor of two between these two columns.

The data for CPU time also exhibit (near) linear performance in both planners. Although the simulation for these planners is done on the same machine, the differences in CPU time for these two planners do not imply much, because they are coded in different languages (Soar on top of C for $M_p \rightarrow M_6$ and Lisp for POCL). Nevertheless, the (near) linear performance of $M_p \rightarrow M_6$ suggests that $M_p \rightarrow M_6$ can perform as well as POCL, in this domain.

4.5 Summary

In this chapter the notion of monotonicity in sequential multi-method planning is investigated. In a monotonic multi-method planner, the single methods are sequenced according to increasing coverage and decreasing efficiency. A formal analysis shows that (1) if the cost of failure for a restricted planner is always less than the cost of success for a more relaxed planner, then monotonic multi-method planners outperform corresponding time-shared multi-method planners; otherwise, time-shared multi-method planners may perform better; (2) a monotonic multi-method planner takes less planning time than the corresponding single-method planner, if the performance gain by using a cheaper method is greater than the wasted time by using inappropriate methods in the monotonic multi-method planner; and (3) the lengths of plans generated from a monotonic multi-method planner and the corresponding time-shared multi-method planner are less than or equal to the length of plans generated from the corresponding single-method planner.

A set of bias-relaxation multi-method planners has been constructed. In bias-relaxation multi-method planning, each bias is evaluated in isolation. Thus, bias-relaxation multi-method planning has a restricted scope in creating and comparing individual methods. The constructed bias-relaxation multi-method planners vary in the granularity at which individual methods are selected and used. Depending on the granularity of method switching, two variations on bias-relaxation

multi-method planners are implemented: coarse-grained multi-method planners, where methods are switched on a problem-by-problem basis; and fine-grained multi-method planners, where methods are switched on a goal-by-goal basis.

The experimental results in the blocks-world and machine-shop-scheduling domains imply that (1) in terms of planning time, fine-grained multi-method planners can be significantly more efficient than coarse-grained multi-method planners and single-method planners; and (2) in terms of plan length, both fine-grained and coarse-grained multi-method planners can be significantly more efficient than single-method planners.

Finally, the comparison of multi-method planning with partial-order planning in D^1S^1 suggests that multi-method planning can be as efficient as partial-order planning in terms of planning performance.

Chapter 5

Application to a Complex Domain

The investigations of multi-method planning in the previous chapter have occurred in the context of the blocks-world and machine-shop scheduling domains. These are classical planning domains that provide good environments for developing and evaluating multi-method planners. However, the intent here is to transfer the multi-method planning technology to a more realistic domain; in particular, a simulated battlefield domain.

The task focused on in this domain is to simulate automated intelligent agents that can accomplish tactical missions in navy fighters. One interesting aspect of this domain is that the main criterion to evaluate planning is how well the missions can be accomplished, whereas planning time and plan length are secondary criteria. This chapter shows how the multi-method planning framework can be applied to domains with such a criterion.

Since this domain involves the complexity of the real world and the domain itself is not clearly defined, the full implementation of a planner that can be used in such agents is beyond the scope of this thesis. The focus in this thesis is on investigating the issues related to planning in this domain and demonstrating planning capabilities that multi-method planners have, rather than developing a real planner that can actually be deployed. The application of multi-method planning to this domain will help both in evaluating the degree of domain independence provided by the multi-method planning framework, as well as being a step toward integrating the technology into a broader agent.

This chapter begins with an overview of simulated battlefield environments and describes the tactical air simulation task. Then, it demonstrates how multi-method planning can be applied to this task.

5.1 Simulated Battlefield Environments

The goal of the work in a simulated battlefield environment is to create agents that act as virtual agents to participate in exercises with real human agents. These exercises are to be used for training as well as for development of tactics. In order for these exercise to be realistic, the agents must be able to behave as much like humans as possible.

To approximate human behaviors, the agents must have capabilities including obeying tactical missions, planning and reacting in real time, adapting to new situations, learning from experience, exhibiting the cognitive limitations and strengths of humans, interacting with other agents, and so on. Developing agents with such capabilities is a non-trivial task with many real-world complexities.

Soar-IFOR is an attempt to build such agents within the Soar architecture. Soar is a promising candidate for developing such agents, because it is a single unified system which can integrate various components of AI technologies such as problem-solving, planning, reasoning, learning, perception, motor control, and so on. In addition, Soar is the basis for the development of unified theories of human cognition [Newell, 1990], and thus can provide an appropriate framework for modeling human like agents.

To begin the effort to build automated intelligent agents for simulated battlefield environments, Soar-IFOR has mainly focused on creating specific automated agents, called *TacAir-Soar*, for simulated tactical air environments [Jones *et al.*, 1993, Rosenbloom *et al.*, 1994].

5.2 Tactical Air Simulation

The goal of building TacAir-Soar is to construct automated intelligent agents for flight simulators that are used to train navy pilots in flight tactics. For example, it can be used in simulating a Barrier Combat Air Patrol (BARCAP) mission; that is, to patrol the skies to protect a High-Value Unit (HVU) such as an aircraft carrier. During the course of the mission, if the agent detects a hostile aircraft, it intercepts the aircraft by firing missiles and then resumes its patrol.

One of the important characteristics of tactical air simulation is that it is a highly reactive, real-time, I/O intensive task. Thus, the agents must be able to make decisions in real-time and react appropriately according to the changes in the environment. On the other hand, it is a highly goal-oriented (or mission-oriented) task. The goals include accomplishing multiple missions and survival. Thus, the agent must have a planning capability which can deal with multiple goals at the same time.

Dealing with multiple goals involves the following issues: how to represent multiple goals and their interaction, how to generate appropriate actions that satisfy multiple goals at a time, how to decide on appropriate actions when multiple goals require conflicting behaviors, which goals can be ignored if necessary, and so on.

The next section presents a prototype agent which employs the multi-method planning technique for tactical air simulation, and demonstrates how multi-method planning can deal with these multiple goal issues. Although TacAir-Soar is a highly reactive agent, the implementation of the prototype agent based on the multi-method planning technique focuses on the planning capabilities only, and not on the reactive capabilities.

5.3 Implementation

The application of multi-method planning in tactical air simulation is based on a beyond-visual-range (BVR) *1-v-1 aggressive bogey* scenario [Jones *et al.*, 1993,

Johnson, 1994, Tambe and Rosenbloom, 1994]. This scenario involves two armed aircraft with similar capabilities. One aircraft (F14) is attempting to protect a high-value unit and the other (MiG29) is attempting to destroy it. When the two aircraft come in contact, they both attempt to intercept and destroy each other, with the overall goals of accomplishing their missions — here, protecting the HVU (or attacking the HVU) — situational awareness, and surviving.

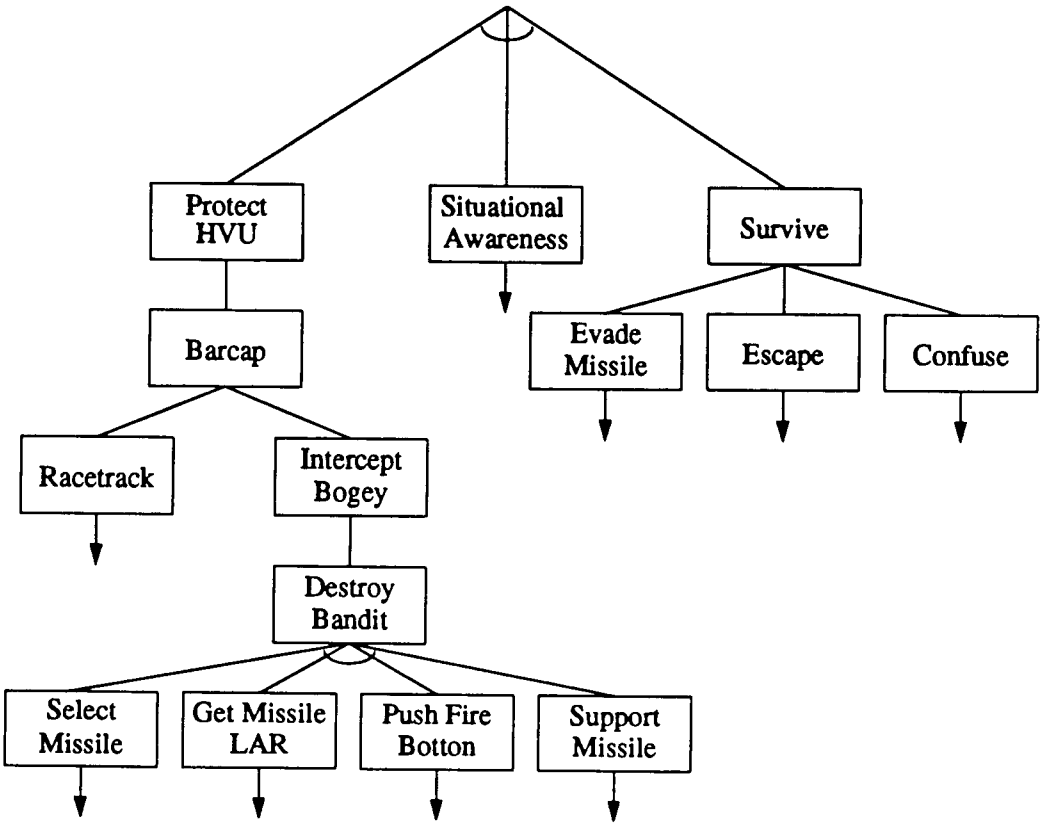


Figure 5.1: A skeleton of the goal hierarchy for the 1-v-1 aggressive bogey scenario.

While performing BARCAP, if a bogey (an unknown aircraft) is noticed, the F14 tries to determine whether the bogey is a bandit (an enemy aircraft). If the bogey is identified as a bandit, the F14 attempts to destroy it by firing missiles. In order to destroy it, the F14 selects a missile — a long-range missile (LRM)

here — and approaches the MiG29 close enough to get into its LRM's launch-acceptability region (LAR). After launching a LRM, the F14 makes an F-pole (a maneuver involving a 25-50 degree turn) to provide radar guidance to the missile, while decreasing the closure between the two aircraft. The fight continues until one aircraft is destroyed or runs away. A skeleton of the goal hierarchy for this scenario from the F14's point of view is shown in Figure 5.1.

The key issue in implementing a planner in this domain is that some goals can never be achieved completely. These goals are called *maintenance goals*. `Protect-HVU`, `situational-awareness` and `survive` are such examples. Thus, the role of an operator for a maintenance goal is not to achieve the goal but to continuously maintain the status of the goal. For example, a single application of the operator `BARCAP` does not achieve the goal `protect-HVU`. By applying this operator continuously, the HVU remains protected.

Maintenance goals make it possible to define multiple *achievement-levels* rather than two levels — achieved and unachieved. For example, the achievement-level for the goal `protect-HVU` is maximum when there is no threat to the HVU, while the achievement-level for this goal is minimum when the HVU is destroyed. When a bogey is noticed, the achievement-level is decreased from the maximum, because the bogey can potentially destroy the HVU. If the bogey is identified as a bandit, the achievement-level is further decreased.

From the opposite point of view, one can define multiple *threat-levels* for each maintenance goal; that is the threat-level is maximum when the achievement-level is minimum, and vise versa. For each maintenance goal, operators which decrease the threat-level for that goal are proposed.

The multiple threat-level scheme allows the notion of protection to be refined for maintenance goals. Instead of protecting achieved goals from being undone, it protects the threat-levels for other goals from being increased. The strongest form of protection in this domain, denoted as GP_0, eliminates all plans in which an operator increases the threat-level of another goal. Weaker forms of protection, denoted as GP_i (i=1, ..., n−2, where n is the number of threat-levels) eliminates

all plans in which an operator increases the threat-level of another goal by more than i[1].

		Goal Flexibility Dimension	
		Directness	Nonlinear
Goal Protection Dimension	GP_0		
	GP_1		
	...		
	...		
	GP_{n-2}		
	$No\ GP$		

Table 5.1: The $2 \times n$ planning methods generated from directness and protection, where there are n threat-levels.

The notions of directness and linearity are not changed here. However, using a linearity bias is dangerous in this domain, because focusing on only one goal conjunct and just ignoring other goals conjuncts until the current one is completely achieved may cause a failure — that is, either the aircraft or the HVU may be destroyed. This yields a set of $2 \times n$ planning methods derived from the two bias dimensions (Table 5.1).

The key to implementing the prototype agent is that the threat-levels for maintenance goals must remain as low as possible. By doing so, the probability that one (or some) of the maintenance goals is seriously threated can be decreased. Single-method planners are not appropriate because if their biases are too strong, they cannot solve the problem without seriously threatening other goals. On the other hand, if their biases are too weak, planning takes too much time since the search space is too large.

Based on the planning methods shown in Table 5.1, a fine-grained multi-method planner is implemented for the 1-v-1 scenario. Figure 5.2 presents an example of

[1]Another possible refinement is to use protection biases which eliminate all plans in which an operator increases the total threat-levels across all other goals by more than i.

Threat-level	Threat-type
0	No threat
1	Potential threat
2	Minor threat
3	Intermediate threat
4	Major threat
5	Fatal threat

(a) Six threat-levels for 1-v-1 aggressive bogey scenario.

Current state: The bandit is aggressive.				
Goal	Proposed Operator	Threat-Level Change		
		Goal	Before	After
protect-HVU	DESTROY-BANDIT	protect-HVU	3	2
		survive	3	4
survive	ESCAPE	survive	3	1
		protect-HVU	3	5
Selected operator: DESTROY-BANDIT.				

(b) Operator selection when the bandit is aggressive.

Table 5.2: Example of fine-grained multi-method planning for tactical air domain.

how the implemented fine-grained multi-method planning actually works. In this implementation, six threat-levels are used as shown in Figure 5.2 (a).

Figure 5.2 (b) shows how an operator is selected when multiple operators are proposed in the situation that the bandit is aggressive. In this situation, the threat-levels for both goals **protect-HVU** and **survive** are set to three, because the aggressive behavior of the bandit is considered as a medium level of threat for protecting the HVU and surviving. For the goal **protect-HVU**, the DESTROY-BANDIT operator is proposed which can decrease the threat-level to minor threat. For the goal **survive**, the ESCAPE operator is proposed which can decrease the threat-level to potential threat. In evaluating these operators, however, applying **escape** increases the threat-level for the other goal **protect-HVU** to fatal threat, whereas applying

DESTROY-BANDIT increases the threat-level for survive to major threat. Since DESTROY-BANDIT increases the threat-level for the other goal less than SURVIVE, DESTROY-BANDIT is selected here.

5.4 Summary

In this chapter, how multi-method planning can be applied to a tactical air domain is briefly discussed. A preliminary investigation is made of some of planning issues in this domain such as how to deal with maintenance goals and how to decide on appropriate actions when multiple goals require conflicting behaviors. In doing this, the notion of protection is refined such that one protects the threat-levels for other goals from being increased. Multi-method planning based on refined protection biases shows how appropriate actions can be generated by this planner.

Chapter 6

Related Work

This chapter describes work related to the multi-method planning framework. Section 6.1 describes biases used in other planning systems. Section 6.2 compares the planning and learning framework in Soar to other planning frameworks. Finally, Section 6.3 compares the presented multi-method planning technique to other related approaches.

6.1 Biases in Planning

Some of the planning biases used here have been introduced by earlier planning systems as planning heuristics. For example, the linearity assumption has been used in planners using a goal stack because of its simplicity [Fikes and Nilsson, 1971]. Also, protection has been used in many planners to reduce the size of the search space and to avoid generating non-optimal plans. These two biases are discussed in more detail here.

6.1.1 Linearity

In a conjunctive goal problem, the assumption that subgoals can be achieved sequentially and thus that the generated plan is a sequence of complete subplans for

the conjunctive goals is known as the *linearity assumption* [Sussman, 1973]. Although many problems cannot be solved without interleaving goal conjuncts, this assumption has two interesting properties.

First, it makes the original problem simpler by allowing decomposition of the problem into a set of subproblems and then solving each subproblem in sequence. Since only a single goal conjunct is considered for each subproblem, the search space to solve the entire problem can be reduced.

Second, it provides a basis for classifying a group of problems in terms of a problem's complexity. Korf [1987] provided a more refined taxonomy about how subgoals interact with each other. He defined a set of subgoals to be *independent* if each operator only changes the distance to a single subgoal. Though this definition is based on a very strong assumption about goal interference, an optimal global solution can be achieved by simply concatenating together optimal solutions to the individual subproblems in any order. Solving a single independent subgoal might be nontrivial, but the complexity of problems with independent subgoals increases only linearly with the number of subgoals.

Also, he defined a set of subgoals to be *serializable* if there exists an ordering among the subgoals such that the subgoals can always be solved sequentially without ever violating a previously solved subgoal in the order. Since this definition is based on the linearity assumption and goal protection, a problem which consists of serializable subgoals can be classified as an element of A_2 — that is, the set of problems solvable by the linear protection method — in the multi-method planning framework.

Barrett and Weld [1992] defined a set of subgoals to be *trivially serializable* if they can be solved in any order without ever violating a previous solved subgoal. From this definition it is implied that if a set of subgoals is independent, it is trivially serializable, and that if a set of subgoals is trivially serializable, it is serializable.

6.1.2 Protection

The notion of protection was introduced in HACKER [Sussman, 1973]. In HACKER, a protection violation is detected if "a protected subgoal is clobbered between the time it is established and the time it is no longer needed" [Sussman, 1973, page 63]. HACKER deals with protection violations by employing procedures called *critics* that recognize such violations. When necessary, HACKER is able to repair the plan by rearranging the steps in the plan.

Waldinger [1977] developed an approach, called *goal regression*, to protect achieved goals. It involves creating a plan to solve one subgoal followed by constructive modifications to achieve the other subgoals. It differs from HACKER in that it uses the notion of goal protection to guide the linear placement of actions in the plan. Rather than building incorrect plans and then debugging them, it builds partial linear plans in non-sequential order and moves subgoals backwards through the partial linear plans to where they do not interfere with other subgoals. Vere [1983] also developed a technique, called *splicing*, which relaxes protection when it has caused a deadlock.

SNLP uses causal links to represent protection intervals and deals with threats to them.

6.2 Planning and Learning in Soar

In the thesis, planning operators are represented by operator proposal rules, operator application rules, goal expansion rules, and instantiated Soar operators in working memory. However, this is not the only way to represent planning operators in Soar. For example, Unruh's [1993] operator representation for abstraction includes rules to check operators' preconditions explicitly before applying operators. Also, in Soar, goal expansion for a violated precondition is usually implemented by creating a new Soar subgoal and achieving the violated condition within the subgoal, via the *operator subgoaling* scheme [Laird *et al.*, 1987].

Planning in Soar is similar to planning in PRODIGY in that both systems use a set of preference-based control rules to yield a sequence of operators. One of the differences between these two systems is that while Soar learns control rules from the result of look-ahead search, PRODIGY learns control rules from its own problem-solving trace [Minton, 1988] or from a static analysis of the domain theory [Etzioni, 1990a]. Another difference is that original PRODIGY (version 2.0) uses a linear planning approach [Minton *et al.*, 1989]. Veloso [1989] developed a nonlinear version of PRODIGY, but the learning method used was a cased-based approach.

6.3 Multi-Method Planning

The basic approach of bias relaxation in multi-method planning is similar to the shift of bias for inductive concept learning [Russell and Grosof, 1987, Utgoff, 1986]. In the planning literature, this approach is closely related to an *ordering modification* which is a control strategy to prefer exploring some plans before others [Gratch and DeJong, 1990]. If the preference is wrong, the alternatives will be eventually reached. Thus, ordering modification retains planner completeness. They explicitly distinguished this modification from *structural modification* which prunes portions of the potential plan space. Planning systems which employ multi-method planning techniques include STEPPINGSTONE [Ruby and Kibler, 1991], and FAILSAFE-2 [Bhatnagar and Mostow, 1990]. These two systems are discussed in more detail here, and a comparison of multi-method planning with partial order planning is presented.

6.3.1 STEPPINGSTONE

STEPPINGSTONE is a learning problem-solver that decomposes a problem into simple and difficult subproblems. It solves the simple subproblems with an inexpensive constrained problem solver. To solve the difficult subproblems, STEPPINGSTONE uses an unconstrained problem solver. Once it solves a difficult subproblem, it uses the solution to generate a sequence of subgoals, or steppingstones, that can be used

by the constrained problem solver to solve this difficult subproblem when it occurs again.

The constrained problem solver takes as input a set of subgoals which are ordered based on a heuristic called *openness*. It attempts to solve the subgoals in the given order, and generate a solution for the subgoals, if one is found. The constraint used in this problem solver is that each solved subgoal is protected. If the constrained problem solver is unable to solve a subgoal, a memory component is called. The memory component is based on a case-based approach. It matches the current problem-solving context — that is, the subgoal currently being solved, the currently protected subgoals, and the current state — with stored contexts, then returns the ordered subgoals for the matching context. When the memory component fails to return any useful subgoal ordering, the unconstrained problem solver is called. The unconstrained problem solver relaxes the protection on the solved subgoals to find a solution.

Since STEPPINGSTONE generate a solution according to the prescribed subgoal ordering, the constrained problem solver is comparable to M_2 (the linear planner with protection), while the unconstrained problem solver is comparable to M_5 (the linear planner without protection). This implies that STEPPINGSTONE is close to the sequential multi-method planner $M_2 \rightarrow M_5$; however, the difference between these two is that STEPPINGSTONE has a cased-based memory component in between M_2 and M_5, which is analogous to the transfer of control rules across problems.

6.3.2 FAILSAFE-2

FAILSAFE-2 (FS2) is a system that performs adaptive search by learning from its failures. The FS2 problem solver uses two types of search control knowledge: *goal selection rules* to constrain the selection of which goal to pick as the next current goal; and *censors* to constrain the selection of which operator to apply to the current state.

There are two types of interactions between the problem solver and learner. The first type occurs when the search is *under-constrained.* The symptoms of under-constrained search include violating a protected goal, reaching a state-loop, and exceeding a preset goal-depth limit. If any of these symptoms is found, the problem solver declares a failure and invokes the learner. If the learner is able to identify the problem solving step that led to the failure, it adds a new censor to prevent similar failures in the future.

The other type of interaction between the problem solver and learner occurs when the search is *over-constrained.* Over-constrained search prunes away all solution paths. Domain-independent heuristics are used to detect over-constrained search. When it is detected, the problem solver calls a heuristic procedure which relaxes a censor. If relaxing the censor leads to achieving the current goal, FS2 infers that the censor was over-general and calls the learner to specialize it.

The basic idea of censor relaxation in FS2 is close to the bias relaxation mechanism in the thesis. However, there are a number of differences, such as the granularity at which censors are relaxed and the way censors are relaxed. Whenever applying an operator to the current state violates a censor, that state is marked as *suspended.* Once the problem solver cannot make progress by forward search with the censor, FS2 selects one of the suspended states that is likely to be closest to the goal based on a heuristic, and uses a weak form of backward chaining (WBC) which recurses on the failed preconditions of an operator one at a time. If a solution is found by this relaxation, the censor is specialized, so that so that it does not prevent the expansion of the search tree in the future.

6.3.3 Partial Order Planners

Fine-grained multi-method planning is related to traditional partial-order planning, where heuristics are used to guide search over the space of partially ordered plans without violating planner completeness. For example, SNLP [McAllester and Rosenblitt, 1991, Barrett and Weld, 1992] uses a heuristic which prefers nodes with fewer unresolved goals. Using directness in fine-grained multi-method planners is

similar to this heuristic, because applying an operator without violating directness reduces the number of unachieved goals by at least one.

The least-commitment approach can be viewed as planning which starts with the strong assumption that the problem can be solved without any ordering constraints, and relaxes that assumption by adding ordering constraints successively only as it is necessary. In this sense, it is similar to the bias-relaxation approach which starts from a set of biases and relaxes the biases only when the problem (or subproblem) cannot be solved with those biases.

Chapter 7

Conclusion

This chapter summarizes the methodology used in the thesis and the results, and then presents some of the limitations of this methodology and future work.

7.1 Summary of the Approach and Results

In this thesis, two hypotheses are investigated in depth: (1) no single planning method will satisfy both sufficiency and efficiency for all situations; and (2) multi-method planning can outperform single-method planning in terms of sufficiency and efficiency. To evaluate these hypotheses, a set of single method planners and a set of multi-method planners have been created. The creation of these planners is based on the notion of bias in planning.

Bias is a useful notion in planning because it can potentially reduce computation effort by reducing the number of plans that must be examined, and it can potentially generate shorter plans by avoiding plans containing inefficient operator sequences. By varying the amount of bias used, a set of planning methods with different performance and scope can be generated.

To evaluate the first hypothesis, a system has been constructed that can utilize different single-methods, which are defined along two bias dimensions: goal-flexibility, and goal-protection. The goal-flexibility dimension determines the degree of flexibility the planner has in generating new subgoals and in shifting the focus in the goal hierarchy. This dimension subsumes directness and linearity

biases. The goal-protection dimension determines whether or not an achieved top-level goal conjunct is protected between the time it is achieved and the time it is no longer needed. By taking the cross-product of these two dimensions, six different methods are created.

These methods have been implemented in Soar. In Soar plans are represented as sets of variabilized control rules and sets of instantiated preferences that jointly specify which operators should be executed at each point in time. The effect of learning in these methods with respect to the performance of planning has been investigated. The six implemented methods have been compared empirically in terms of planner completeness, planning time, and plan length. The experimental results show a trade-off between completeness and efficiency for these methods — that is, if a method is too restricted, it cannot generate plans for some problems, while if it is too relaxed, it takes too much time in generating plans, and the generated plans are inefficient.

As an alternative approach to single-method planners, multi-method planners have been created. A multi-method planner consists of a coordinated set of planning methods, where each individual method has different scope and performance. Given a set of created methods, the key issue here is how to coordinate the methods in an efficient manner so that the multi-method planner can have high performance. This includes issues of selecting appropriate methods as situations arise, and the granularity of method switching as the situational demands shift.

For the method selection issue, two ways of organizing individual methods in a multi-method planner — sequential and time-shared — have been compared analytically. The wasted effort in a sequential multi-method planner is the cost of trying earlier methods in the sequence, whereas the wasted effort in a time-shared multi-method planner is the cost of trying all methods in the method set except the one that actually solves the problem. The wasted effort in sequential multi-method planning is sensitive to the ordering of the methods because it takes too much time if earlier methods are not efficient enough, or in an extreme case, it may not be able to generate a plan at all if one of the earlier methods does not halt. On the

other hand, the wasted effort in time-shared multi-method planning is sensitive to the number of individual methods.

As an approach to reducing the wasted time in sequential multi-method planning, monotonic multi-method planning has been investigated. In a monotonic multi-method planner, the individual methods are ordered according to decreasing efficiency and increasing coverage based on the empirical performance of those methods for a training set of problems. A formal analysis shows that (1) a monotonic multi-method planner takes less planning time than the corresponding single-method planner, if the performance gain by using a cheaper method is greater than the wasted time by using inappropriate methods in the monotonic multi-method planner; and (2) the lengths of plans generated from a monotonic multi-method planner are less than or equal to the length of plans generated from the corresponding single-method planner.

To restrict the scope of individual methods to be generated and compared, a set of bias-relaxation multi-method planners has been constructed based on the notion of effective bias. In a bias-relaxation multi-method planner, planning starts by trying highly efficient methods, and then successively relaxes effective biases until a sufficient method is found.

The second issue of coordinating individual methods in multi-method planning is the granularity at which individual planning methods are be switched. While in coarse-grained multi-method planners, methods are switched for a whole problem when no solution can be found for the problem within the current method, in fine-grained multi-method planners methods can be switched at any point during a problem at which a new set of subgoals is formulated, and the switch only occurs for that set of subgoals (and not for the entire problem). Both coarse-grained multi-method planners and fine-grained multi-method planners are implemented via bias relaxation.

There is a trade-off between coarse-grained multi-method planning and fine-grained multi-method planning. A coarse-grained multi-method planner finds a solution within the first method that has one at the cost of searching the entire

biased space in the worst case. On the other hand, a fine-grained multi-method planner can save the effort of searching all other alternatives within the current method; however, it does not guarantee to find a solution that may exist within the current biased space.

The experimental results in the blocks-world and machine-shop-scheduling domains imply that (1) in terms of planning time, fine-grained multi-method planners can be significantly more efficient than coarse-grained multi-method planners and single-method planners; and (2) in terms of plan length, both fine-grained and coarse-grained multi-method planners can be significantly more efficient than single-method planners.

In summary, the primary contribution of this thesis is to develop a new multi-method planning framework. This framework is developed based on the notion of bias (for method creation), and the notions of monotonicity, bias-relaxation, and the granularity of method switching (for method coordination). The experimental results indicate that, at least for the domains investigated, the created multi-method planners are more efficient than complete single-method planners.

7.2 Limitations and Future Work

The multi-method planning framework investigated in this thesis is based on three biases: linearity, protection, and directness. One way to enhance the multi-method planning framework would be to extend the set of biases available. These biases include ones that limit the size of the goal hierarchy such as goal-depth or goal-breadth (to reduce the search space), limit the length of plans generated such as plan-length (to shorten execution time), and lead to learning more effective rules such as goal-nonrepetition (to increase transfer).

The multi-method planners used here do not guarantee finding optimal plans for a given problem. However, if a plan-length bias is incorporated with coarse-grained multi-method planning, where the bound is incrementally specified along with the sequence of individual methods, the multi-method planner will be able to

find optimal plans for all problems. In fact, this approach implements depth-first iterative-deepening [Korf, 1985] on the length of plans generated.

The bias selection approach used here is based on preprocessing a set of training examples in order to develop fixed sequences of biases (and methods). This approach has a limitation when it is hard to generate testing problems or when the problem distribution is unknown. A more dynamic, run-time approach would be to learn, while doing, which biases (and methods) to use for which classes of problems. If such learned information can transfer to the later problems, much of the effort wasted in trying inappropriate methods may be reduced.

One problem of learning about which methods to use for which classes of problems in the current multi-method framework is that some rules (chunks) are expensive. Restricting expressiveness on the encoding of tasks such as by the unique-attribute scheme can solve this problem [Tambe *et al.*, 1990]; however the learned rules based on this scheme may not be general enough to transfer to the later problems because of the limited expressibility. Another approach to solving the expensive chunk problem is to incorporate search control knowledge into the explanation [Kim and Rosenbloom, 1993]. This approach can solve expensive chunk problem without restricting expressiveness. Learned rules can be used with the cost bounded by the cost of the problem solving from which it was learned.

The methodology for generating a set of monotonic multi-method planners or a set of bias-relaxation multi-method planners does not specify which multi-method planner is the optimal one for a given problem distribution. Greiner [1992] developed an algorithm called PALO which searches the space of performance elements and selects a near locally optimal element by using statistical techniques to approximate the distribution. By employing the PALO algorithm in the multi-method planning framework, it may be possible to generate the optimal multi-method planner for a given distribution.

Within simulated battlefield environments, the focus of this thesis is on planning based on beyond-visual-range 1-v-1 aggressive bogey scenario. One of the directions for future work in this domain includes applying the technique described in

Chapter 5 to other scenarios, such as 1-v-2, 2-v-1, and 2-v-n scenarios, or Within Visual Range scenario. Application to air-to-ground or ground-to-ground combat simulation would be another possibility.

Appendix A

Experimental Results: The Blocks-World Domain

This appendix gives the detailed numeric information from the experiments in the blocks-world domain. Appendix A.1 presents the experimental results for the six single-method planners over 30 training problems. Appendix A.2 presents the experimental results for the six single-method planners and the created multi-method planners over 100 test problems.

A.1 Performance over 30 Training Problems

Method M_1										
Problems			Decisions				Plan Length			
Number	Blocks	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	4	3	-	-	-	-	-	-	-	-
2	4	3	16	16	16	16.0	2	2	2	2.0
3	4	4	-	-	-	-	-	-	-	-
4	3	3	29	31	30	30.0	3	3	3	3.0
5	4	3	8	8	8	8.0	1	1	1	1.0
6	4	2	-	-	-	-	-	-	-	-
7	3	2	-	-	-	-	-	-	-	-
8	3	2	8	8	8	8.0	1	1	1	1.0
9	3	3	-	-	-	-	-	-	-	-
10	4	4	8	8	8	8.0	1	1	1	1.0
11	4	4	-	-	-	-	-	-	-	-
12	4	2	-	-	-	-	-	-	-	-
13	3	3	25	29	30	28.0	3	3	3	3.0
14	4	3	3	3	3	3.0	0	0	0	0.0
15	3	3	17	17	17	17.0	2	2	2	2.0
16	3	2	8	8	8	8.0	1	1	1	1.0
17	3	2	9	9	9	9.0	2	2	2	2.0
18	4	2	16	16	16	16.0	2	2	2	2.0
19	4	4	8	8	8	8.0	1	1	1	1.0
20	3	2	-	-	-	-	-	-	-	-
21	3	3	-	-	-	-	-	-	-	-
22	3	2	8	8	8	8.0	1	1	1	1.0
23	4	3	-	-	-	-	-	-	-	-
24	3	3	8	8	8	8.0	1	1	1	1.0
25	4	2	9	9	9	9.0	2	2	2	2.0
26	4	2	16	16	16	16.0	2	2	2	2.0
27	4	2	-	-	-	-	-	-	-	-
28	4	3	-	-	-	-	-	-	-	-
29	4	3	-	-	-	-	-	-	-	-
30	3	2	-	-	-	-	-	-	-	-
Total			196	202	202	200.0	25	25	25	25.0
Solved Problems			16	16	16	16.0	16	16	16	16.0
Average			12.25	12.62	12.62	12.50	1.56	1.56	1.56	1.56

Table A.1: Performance of M_1 over 30 training problems in the blocks-world domain.

Method M_2										
Problems			Decisions				Plan Length			
Number	Blocks	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	4	3	19	19	40	26.0	4	4	4	4.0
2	4	3	16	16	16	16.0	2	2	2	2.0
3	4	4	-	-	-	-	-	-	-	-
4	3	3	34	33	32	33.0	3	3	3	3.0
5	4	3	8	8	8	8.0	1	1	1	1.0
6	4	2	27	27	31	28.3	4	4	4	4.0
7	3	2	26	27	27	26.7	3	3	3	3.0
8	3	2	8	8	8	8.0	1	1	1	1.0
9	3	3	-	-	-	-	-	-	-	-
10	4	4	8	8	8	8.0	1	1	1	1.0
11	4	4	36	25	35	32.0	4	3	4	3.7
12	4	2	31	33	25	29.7	3	3	3	3.0
13	3	3	33	25	34	30.7	3	3	3	3.0
14	4	3	3	3	3	3.0	0	0	0	0.0
15	3	3	16	16	26	19.3	2	2	2	2.0
16	3	2	8	8	8	8.0	1	1	1	1.0
17	3	2	9	9	9	9.0	2	2	2	2.0
18	4	2	16	16	16	16.0	2	2	2	2.0
19	4	4	8	8	8	8.0	1	1	1	1.0
20	3	2	32	25	25	27.3	3	3	3	3.0
21	3	3	28	28	27	27.7	4	4	3	3.7
22	3	2	8	8	8	8.0	1	1	1	1.0
23	4	3	25	34	25	28.0	3	4	3	3.3
24	3	3	8	8	8	8.0	1	1	1	1.0
25	4	2	9	9	9	9.0	2	2	2	2.0
26	4	2	16	16	16	16.0	2	2	2	2.0
27	4	2	18	27	29	24.7	3	4	5	4.0
28	4	3	25	31	43	33.0	3	3	5	3.7
29	4	3	-	-	-	-	-	-	-	-
30	3	2	-	-	-	-	-	-	-	-
Total			475	475	524	491.3	59	60	62	60.3
Solved Problems			26	26	26	26.0	26	26	26	26.0
Average			18.27	18.27	20.15	18.90	2.27	2.31	2.38	2.32

Table A.2: Performance of M_2 over 30 training problems in the blocks-world domain.

Method M_3										
Problems			Decisions				Plan Length			
Number	Blocks	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	4	3	19	19	41	26.3	4	4	4	4.0
2	4	3	16	16	16	16.0	2	2	2	2.0
3	4	4	-	-	-	-	-	-	-	-
4	3	3	33	32	33	32.7	4	3	4	3.7
5	4	3	8	8	8	8.0	1	1	1	1.0
6	4	2	34	50	202	95.3	4	5	5	4.7
7	3	2	33	33	33	33.0	3	3	3	3.0
8	3	2	8	8	8	8.0	1	1	1	1.0
9	3	3	-	-	-	-	-	-	-	-
10	4	4	8	8	8	8.0	1	1	1	1.0
11	4	4	32	72	48	50.7	3	5	4	4.0
12	4	2	46	32	40	39.3	3	3	3	3.0
13	3	3	32	32	41	35.0	3	3	4	3.3
14	4	3	3	3	3	3.0	0	0	0	0.0
15	3	3	16	24	16	18.7	2	2	2	2.0
16	3	2	8	8	8	8.0	1	1	1	1.0
17	3	2	9	9	9	9.0	2	2	2	2.0
18	4	2	16	16	16	16.0	2	2	2	2.0
19	4	4	8	8	8	8.0	1	1	1	1.0
20	3	2	25	25	32	27.3	3	3	3	3.0
21	3	3	34	49	41	41.3	3	4	3	3.3
22	3	2	8	8	8	8.0	1	1	1	1.0
23	4	3	32	119	74	75.0	3	7	6	5.3
24	3	3	8	8	8	8.0	1	1	1	1.0
25	4	2	9	9	9	9.0	2	2	2	2.0
26	4	2	16	16	16	16.0	2	2	2	2.0
27	4	2	34	58	18	36.7	4	5	3	4.0
28	4	3	94	40	56	63.3	6	3	4	4.3
29	4	3	-	-	-	-	-	-	-	-
30	3	2	-	-	-	-	-	-	-	-
Total			589	710	800	699.7	62	67	65	64.7
Solved Problems			26	26	26	26.0	26	26	26	26.0
Average			22.65	27.31	30.77	26.91	2.38	2.58	2.50	2.49

Table A.3: Performance of M_3 over 30 training problems in the blocks-world domain.

Method M_4										
Problems			Decisions				Plan Length			
Number	Blocks	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	4	3	-	-	-	-	-	-	-	-
2	4	3	16	16	16	16.0	2	2	2	2.0
3	4	4	-	-	-	-	-	-	-	-
4	3	3	25	29	31	28.3	3	3	3	3.0
5	4	3	8	8	8	8.0	1	1	1	1.0
6	4	2	-	-	-	-	-	-	-	-
7	3	2	-	-	-	-	-	-	-	-
8	3	2	8	8	8	8.0	1	1	1	1.0
9	3	3	-	-	-	-	-	-	-	-
10	4	4	8	8	8	8.0	1	1	1	1.0
11	4	4	-	-	-	-	-	-	-	-
12	4	2	-	-	-	-	-	-	-	-
13	3	3	30	29	30	29.7	3	3	3	3.0
14	4	3	3	3	3	3.0	0	0	0	0.0
15	3	3	17	17	17	17.0	2	2	2	2.0
16	3	2	8	8	8	8.0	1	1	1	1.0
17	3	2	18	16	18	17.3	2	2	2	2.0
18	4	2	16	16	16	16.0	2	2	2	2.0
19	4	4	8	8	8	8.0	1	1	1	1.0
20	3	2	-	-	-	-	-	-	-	-
21	3	3	-	-	-	-	-	-	-	-
22	3	2	8	8	8	8.0	1	1	1	1.0
23	4	3	-	-	-	-	-	-	-	-
24	3	3	8	8	8	8.0	1	1	1	1.0
25	4	2	16	18	18	17.3	2	2	2	2.0
26	4	2	16	16	16	16.0	2	2	2	2.0
27	4	2	-	-	-	-	-	-	-	-
28	4	3	-	-	-	-	-	-	-	-
29	4	3	-	-	-	-	-	-	-	-
30	3	2	-	-	-	-	-	-	-	-
Total			213	216	221	216.7	25	25	25	25.0
Solved Problems			16	16	16	16.0	16	16	16	16.0
Average			13.31	13.50	13.81	13.54	1.56	1.56	1.56	1.56

Table A.4: Performance of M_4 over 30 training problems in the blocks-world domain.

Method M_5										
Problems			Decisions				Plan Length			
Number	Blocks	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	4	3	58	68	188	104.7	7	9	26	14.0
2	4	3	16	16	16	16.0	2	2	2	2.0
3	4	4	27	29	18	24.7	4	4	3	3.7
4	3	3	24	32	24	26.7	3	3	3	3.0
5	4	3	8	8	8	8.0	1	1	1	1.0
6	4	2	44	34	44	40.7	6	4	6	5.3
7	3	2	26	25	25	25.3	3	3	3	3.0
8	3	2	8	8	8	8.0	1	1	1	1.0
9	3	3	20	20	18	19.3	4	4	3	3.7
10	4	4	8	8	8	8.0	1	1	1	1.0
11	4	4	25	34	35	31.3	3	4	4	3.7
12	4	2	25	25	36	28.7	3	3	5	3.7
13	3	3	33	49	32	38.0	3	12	7	7.3
14	4	3	3	3	3	3.0	0	0	0	0.0
15	3	3	16	26	26	22.7	2	2	2	2.0
16	3	2	8	8	8	8.0	1	1	1	1.0
17	3	2	25	27	16	22.7	7	8	2	5.7
18	4	2	16	16	16	16.0	2	2	2	2.0
19	4	4	8	8	8	8.0	1	1	1	1.0
20	3	2	25	26	26	25.7	3	3	3	3.0
21	3	3	25	27	25	25.7	3	3	3	3.0
22	3	2	8	8	8	8.0	1	1	1	1.0
23	4	3	64	52	34	50.0	9	6	4	6.3
24	3	3	8	8	8	8.0	1	1	1	1.0
25	4	2	16	16	28	20.0	2	2	4	2.7
26	4	2	16	16	16	16.0	2	2	2	2.0
27	4	2	36	36	28	33.3	5	5	4	4.7
28	4	3	33	44	25	34.0	3	4	3	3.3
29	4	3	54	35	50	46.3	7	5	6	6.0
30	3	2	18	20	18	18.7	3	4	3	3.3
Total			701	732	803	745.3	93	101	107	100.3
Solved Problems			30	30	30	30.0	30	30	30	30.0
Average			23.37	24.40	26.77	24.84	3.10	3.37	3.57	3.34

Table A.5: Performance of M_5 over 30 training problems in the blocks-world domain.

Method M_6										
Problems			Decisions				Plan Length			
Number	Blocks	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	4	3	379	240	436	351.7	13	14	11	12.7
2	4	3	16	16	16	16.0	2	2	2	2.0
3	4	4	29	18	34	27.0	4	3	4	3.7
4	3	3	32	40	40	37.3	3	4	4	3.7
5	4	3	8	8	8	8.0	1	1	1	1.0
6	4	2	154	160	48	120.7	7	14	4	8.3
7	3	2	34	34	33	33.7	3	3	3	3.0
8	3	2	8	8	8	8.0	1	1	1	1.0
9	3	3	20	20	20	20.0	4	4	4	4.0
10	4	4	8	8	8	8.0	1	1	1	1.0
11	4	4	32	80	32	48.0	3	5	3	3.7
12	4	2	40	32	48	40.0	3	3	4	3.3
13	3	3	56	65	40	53.7	8	12	4	8.0
14	4	3	3	3	3	3.0	0	0	0	0.0
15	3	3	26	24	24	24.7	2	2	2	2.0
16	3	2	8	8	8	8.0	1	1	1	1.0
17	3	2	21	16	16	17.7	5	2	2	3.0
18	4	2	16	16	16	16.0	2	2	2	2.0
19	4	4	8	8	8	8.0	1	1	1	1.0
20	3	2	32	33	33	32.7	3	3	3	3.0
21	3	3	33	50	33	38.7	3	4	3	3.3
22	3	2	8	8	8	8.0	1	1	1	1.0
23	4	3	32	89	32	51.0	3	6	3	4.0
24	3	3	8	8	8	8.0	1	1	1	1.0
25	4	2	16	16	26	19.3	2	2	4	2.7
26	4	2	16	16	16	16.0	2	2	2	2.0
27	4	2	76	28	26	43.3	8	4	4	5.3
28	4	3	40	40	48	42.7	3	3	4	3.3
29	4	3	82	147	74	101.0	6	11	6	7.7
30	3	2	18	20	18	18.7	3	4	3	3.3
Total			1259	1259	1168	1228.7	99	116	88	101.0
Solved Problems			30	30	30	30.0	30	30	30	30.0
Average			41.97	41.97	38.93	40.96	3.30	3.87	2.93	3.37

Table A.6: Performance of M_6 over 30 training problems in the blocks-world domain.

A.2 Performance over 100 Testing Problems

The entries in the tables are defined as follows:

N: Problem number

B: Number of blocks

G: Number of goal conjuncts

SP: Solved problems.

D: Number of decisions

L: Plan length

C: CPU time (sec.)

Pbm			M_1			M_2			M_3			M_4			M_5			M_6		
N	B	G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
1	4	3	-	-	-	49	4	1.74	64	5	2.93	-	-	-	49	4	1.69	64	5	2.90
2	3	3	9	2	0.14	9	2	0.14	9	2	0.12	18	2	0.37	16	2	0.28	16	2	0.28
3	4	2	-	-	-	26	3	0.78	33	3	1.01	-	-	-	26	3	0.81	33	3	1.07
4	4	4	9	2	0.25	9	2	0.25	9	2	0.25	16	2	0.42	16	2	0.41	16	2	0.43
5	4	3	10	3	0.29	46	3	2.48	26	4	0.84	24	3	0.66	24	3	0.64	24	3	0.65
6	4	4	9	2	0.25	9	2	0.25	9	2	0.25	18	2	0.56	16	2	0.43	16	2	0.43
7	3	2	-	-	-	44	4	1.08	34	4	0.78	-	-	-	30	5	0.64	45	5	1.07
8	4	2	8	1	0.13	8	1	0.13	8	1	0.13	8	1	0.13	8	1	0.14	8	1	0.13
9	4	4	-	-	-	294	6	31.77	59	5	2.64	-	-	-	63	8	2.74	91	8	4.76
10	3	3	8	1	0.11	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.11	8	1	0.11
11	4	3	9	2	0.22	9	2	0.22	9	2	0.21	18	2	0.53	26	4	0.81	16	2	0.38
12	4	2	9	2	0.21	9	2	0.20	9	2	0.20	18	2	0.48	37	6	1.29	16	2	0.34
13	3	2	-	-	-	27	3	0.54	32	3	0.75	-	-	-	26	3	0.56	39	3	1.06
14	3	2	9	2	0.13	9	2	0.13	9	2	0.13	18	2	0.34	21	5	0.46	16	2	0.26
15	3	2	9	2	0.13	9	2	0.12	9	2	0.12	18	2	0.34	16	2	0.26	16	2	0.26
16	4	4	-	-	-	26	4	0.91	49	5	2.23	-	-	-	33	4	1.12	464	17	108.50
17	4	2	-	-	-	34	4	1.03	48	4	1.53	-	-	-	25	3	0.67	118	9	5.89
18	3	2	8	1	0.09	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.09
19	3	2	8	1	0.09	8	1	0.10	8	1	0.10	8	1	0.09	8	1	0.10	8	1	0.10
20	4	3	-	-	-	43	5	1.50	50	5	1.80	-	-	-	53	7	2.12	80	6	3.47
21	3	3	3	0	0.04	3	0	0.03	3	0	0.03	3	0	0.04	3	0	0.04	3	0	0.04
22	4	3	-	-	-	25	3	0.72	34	4	1.11	-	-	-	25	3	0.71	57	5	2.11
23	4	3	10	3	0.29	24	3	0.67	32	3	0.97	10	3	0.28	31	3	0.99	56	5	2.28
24	4	3	-	-	-	59	4	3.56	57	4	2.46	-	-	-	54	7	2.05	113	7	7.00
25	4	4	-	-	-	32	4	1.21	63	4	2.49	-	-	-	71	6	2.85	73	5	2.99
26	3	3	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.11
27	4	3	10	3	0.30	25	3	0.75	40	3	1.58	10	3	0.28	32	3	0.96	56	5	1.67
28	3	3	8	1	0.11	8	1	0.10	8	1	0.11	8	1	0.11	8	1	0.10	8	1	0.11
29	4	3	10	3	0.29	35	4	1.23	48	4	1.63	10	3	0.28	36	4	1.15	48	4	1.70
30	3	2	-	-	-	17	2	0.28	17	2	0.28	-	-	-	17	2	0.28	17	2	0.27
31	4	3	9	2	0.22	9	2	0.22	9	2	0.22	16	2	0.38	16	2	0.38	16	2	0.37
32	4	4	-	-	-	42	4	1.55	66	5	2.93	-	-	-	34	4	1.24	89	7	3.95
33	3	3	8	1	0.11	8	1	0.11	8	1	0.11	8	1	0.11	8	1	0.11	8	1	0.10
34	3	3	9	2	0.13	9	2	0.14	9	2	0.13	16	2	0.27	16	2	0.28	16	2	0.28
35	4	3	-	-	-	26	4	0.79	42	5	2.18	-	-	-	206	25	16.20	179	15	13.19
36	4	2	8	1	0.15	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14
37	3	3	8	1	0.11	8	1	0.10	8	1	0.11	8	1	0.11	8	1	0.10	8	1	0.10
38	4	2	-	-	-	64	6	2.38	40	4	1.33	-	-	-	53	7	1.94	119	10	5.00
39	3	2	8	1	0.10	8	1	0.10	8	1	0.09	8	1	0.10	8	1	0.09	8	1	0.09
40	3	2	3	0	0.03	3	0	0.03	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.04
41	3	2	-	-	-	19	2	0.35	17	2	0.28	-	-	-	17	2	0.29	17	2	0.27
42	3	2	-	-	-	21	3	0.50	21	3	0.47	-	-	-	27	4	0.58	27	4	0.58
43	4	3	8	1	0.15	8	1	0.13	8	1	0.14	8	1	0.13	8	1	0.14	8	1	0.14
44	4	3	10	3	0.27	25	3	0.71	58	3	2.05	10	3	0.26	25	3	0.72	24	3	0.63
45	4	4	16	2	0.43	16	2	0.44	16	2	0.44	16	2	0.43	16	2	0.43	16	2	0.43
46	4	4	9	2	0.23	16	2	0.40	25	3	0.76	9	2	0.23	16	2	0.40	16	2	0.40
47	3	2	-	-	-	17	2	0.29	17	2	0.28	-	-	-	17	2	0.29	17	2	0.28
48	3	3	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.11	8	1	0.10
49	4	4	11	4	0.41	18	4	0.61	34	5	1.31	42	4	1.88	119	13	6.26	374	10	43.41
50	3	3	10	3	0.17	27	3	0.64	25	3	0.57	19	3	0.39	32	7	0.85	40	4	1.07

Table A.7: Performance of single-method planners over 100 testing problems in the blocks-world domain.

Pbm		M_1			M_2			M_3			M_4			M_5			M_6		
NB	G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
51	4 3	10	3	0.28	39	3	1.50	64	5	2.55	10	3	0.28	34	4	1.12	48	4	1.65
52	3 3	8	1	0.10	8	1	0.10	8	1	0.11	8	1	0.11	8	1	0.10	8	1	0.10
53	3 2	9	2	0.13	9	2	0.12	9	2	0.13	16	2	0.26	21	5	0.46	16	2	0.26
54	4 3	10	3	0.28	24	3	0.66	24	3	0.67	10	3	0.28	50	6	1.84	64	5	2.68
55	4 4	11	4	0.38	49	4	1.83	41	5	1.52	11	4	0.39	53	5	2.23	87	5	4.31
56	3 2	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.09	8	1	0.09
57	4 4	9	2	0.24	16	2	0.40	16	2	0.42	9	2	0.22	16	2	0.40	32	3	1.01
58	3 3	9	2	0.13	26	3	0.59	40	3	1.05	9	2	0.13	26	3	0.59	40	3	1.03
59	4 4	-	-	-	52	5	2.10	48	5	1.87	-	-	-	82	10	3.82	67	6	3.11
60	3 2	3	0	0.04	3	0	0.03	3	0	0.03	3	0	0.04	3	0	0.03	3	0	0.04
61	3 3	9	2	0.15	24	2	0.52	16	2	0.29	9	2	0.14	32	7	0.89	16	2	0.28
62	3 2	-	-	-	-	-	-	18	3	0.32	-	-	-	36	9	1.00	138	22	6.10
63	3 2	9	2	0.14	9	2	0.13	9	2	0.12	16	2	0.26	23	6	0.55	23	6	0.55
64	3 3	9	2	0.13	16	2	0.28	16	2	0.28	9	2	0.12	26	2	0.55	16	2	0.27
65	3 2	9	2	0.12	16	2	0.26	16	2	0.26	9	2	0.12	16	2	0.26	16	2	0.25
66	4 2	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.15	8	1	0.14	8	1	0.14
67	3 3	9	2	0.14	16	2	0.28	24	2	0.51	9	2	0.13	26	2	0.53	24	2	0.49
68	4 2	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.03
69	4 3	-	-	-	46	4	1.88	33	4	1.06	-	-	-	89	12	4.15	207	26	21.14
70	3 2	-	-	-	-	-	-	-	-	-	-	-	-	18	3	0.33	20	4	0.39
71	3 2	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.09	8	1	0.09	8	1	0.09
72	3 3	8	1	0.10	8	1	0.11	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.10
73	4 2	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14
74	4 3	9	2	0.20	16	2	0.37	31	3	1.10	9	2	0.22	98	13	4.67	16	2	0.36
75	3 2	-	-	-	28	4	0.61	64	5	1.67	-	-	-	28	4	0.59	65	5	1.81
76	3 2	9	2	0.13	9	2	0.12	9	2	0.13	18	2	0.34	21	5	0.46	23	6	0.56
77	4 3	-	-	-	-	-	-	-	-	-	-	-	-	36	5	1.30	119	12	6.69
78	4 2	-	-	-	17	2	0.38	41	3	1.47	-	-	-	26	3	0.73	17	2	0.36
79	4 2	-	-	-	25	3	0.66	41	4	1.36	-	-	-	25	3	0.67	25	3	0.65
80	3 2	3	0	0.04	3	0	0.04	3	0	0.03	3	0	0.03	3	0	0.03	3	0	0.04
81	3 3	10	3	0.17	17	3	0.33	17	3	0.33	17	3	0.31	32	7	0.88	45	7	1.35
82	3 3	9	2	0.13	26	2	0.59	25	3	0.56	9	2	0.13	29	2	0.66	16	2	0.28
83	3 3	9	2	0.14	9	2	0.13	9	2	0.13	16	2	0.28	25	7	0.68	29	9	0.93
84	3 3	-	-	-	-	-	-	-	-	-	-	-	-	20	3	0.40	20	3	0.41
85	3 3	10	3	0.17	17	3	0.32	17	3	0.33	26	3	0.55	42	9	1.20	24	3	0.51
86	3 2	8	1	0.10	8	1	0.09	8	1	0.10	8	1	0.10	8	1	0.09	8	1	0.10
87	4 2	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14
88	4 2	-	-	-	26	3	0.71	58	5	2.03	-	-	-	26	3	0.71	41	3	1.19
89	3 2	8	1	0.10	8	1	0.09	8	1	0.09	8	1	0.11	8	1	0.10	8	1	0.09
90	4 2	-	-	-	27	4	0.89	57	5	2.11	-	-	-	96	12	4.68	97	7	4.87
91	3 2	-	-	-	-	-	-	-	-	-	-	-	-	18	3	0.32	20	4	0.39
92	4 4	11	4	0.41	47	4	2.79	26	4	0.93	53	4	2.75	66	8	2.86	292	22	36.63
93	4 2	-	-	-	25	3	0.68	40	3	1.36	-	-	-	69	7	2.51	25	3	0.67
94	4 2	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.15	8	1	0.13
95	4 4	10	3	0.31	10	3	0.31	10	3	0.32	24	3	0.73	45	6	1.70	316	13	33.53
96	3 2	-	-	-	19	2	0.35	19	2	0.35	-	-	-	17	2	0.27	19	2	0.35
97	3 3	9	2	0.14	24	2	0.51	25	3	0.56	9	2	0.13	29	2	0.67	24	2	0.50
98	3 3	10	3	0.16	41	3	1.32	34	3	0.79	10	3	0.16	24	3	0.51	32	3	0.77
99	3 3	10	3	0.17	27	3	0.64	17	3	0.33	17	3	0.32	24	3	0.52	33	8	0.96
100	4 4	10	3	0.33	33	3	1.56	33	4	1.18	17	3	0.49	50	6	1.96	50	6	2.01
Total		587	124	15.36	2154	223	91.51	2266	244	82.75	839	124	23.90	2922	382	104.11	4793	414	361.56
SP		68	68	68	95	95	95	96	96	96	68	68	68	100	100	100	100	100	100
Average		8.63	1.82	0.23	22.67	2.35	0.96	23.60	2.54	0.86	12.34	1.82	0.35	29.22	3.82	1.04	47.93	4.14	3.62

Table A.8: Performance of single-method planners over 100 testing problems in the blocks-world domain (continued).

Pbm			$M_1 \rightarrow M_2 \rightarrow M_5$			$M_1 \rightarrow M_4 \rightarrow M_5$			$M_1 \rightarrow M_5$			$M_2 \rightarrow M_5$			$M_4 \rightarrow M_5$		
N	B	G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
1	4	3	54	5	2.16	83	8	3.87	52	5	1.83	54	4	1.96	52	5	1.80
2	3	3	14	2	0.23	14	2	0.23	14	2	0.23	14	2	0.22	21	2	0.47
3	4	2	38	4	1.35	39	3	1.15	36	4	1.06	31	3	1.00	34	3	0.92
4	4	4	14	2	0.36	14	2	0.36	14	2	0.36	14	2	0.36	23	2	0.69
5	4	3	15	3	0.41	15	3	0.39	15	3	0.38	51	3	2.55	31	3	0.97
6	4	4	14	2	0.36	14	2	0.37	14	2	0.36	14	2	0.36	21	2	0.69
7	3	2	37	4	0.91	41	5	0.92	35	4	0.71	49	4	1.23	36	5	0.76
8	4	2	13	1	0.22	13	1	0.22	13	1	0.22	13	1	0.21	13	1	0.22
9	4	4	72	5	3.10	80	8	3.77	96	8	4.17	299	6	32.89	75	8	3.61
10	3	3	13	1	0.20	13	1	0.19	13	1	0.19	13	1	0.19	13	1	0.19
11	4	3	14	2	0.32	14	2	0.33	14	2	0.33	14	2	0.32	23	2	0.63
12	4	2	14	2	0.30	14	2	0.30	14	2	0.28	14	2	0.29	21	2	0.53
13	3	2	49	3	1.48	39	3	0.88	35	3	0.67	32	3	0.72	35	3	0.68
14	3	2	14	2	0.22	14	2	0.21	14	2	0.21	14	2	0.20	21	2	0.42
15	3	2	14	2	0.21	14	2	0.22	14	2	0.21	14	2	0.20	23	2	0.42
16	4	4	128	5	8.14	128	12	6.20	65	8	2.62	31	4	1.21	220	15	13.70
17	4	2	77	3	3.22	47	4	1.36	43	4	1.29	39	4	1.28	41	3	1.14
18	3	2	13	1	0.17	13	1	0.18	13	1	0.17	13	1	0.18	13	1	0.17
19	3	2	13	1	0.18	13	1	0.19	13	1	0.18	13	1	0.17	13	1	0.18
20	4	3	44	4	1.55	65	7	2.44	61	7	2.26	48	5	1.83	41	3	1.22
21	3	3	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.03	3	0	0.04
22	4	3	62	6	2.47	77	9	3.00	52	6	1.92	30	3	0.99	33	3	0.91
23	4	3	15	3	0.41	15	3	0.40	15	3	0.38	29	3	0.92	15	3	0.39
24	4	3	46	5	1.75	60	7	2.20	62	7	2.24	64	4	3.89	61	7	2.27
25	4	4	74	6	3.37	47	4	1.67	67	6	2.62	37	4	1.51	42	4	1.44
26	3	3	13	1	0.19	13	1	0.19	13	1	0.18	13	1	0.19	13	1	0.19
27	4	3	15	3	0.40	15	3	0.41	15	3	0.39	30	3	1.00	15	3	0.40
28	3	3	13	1	0.19	13	1	0.20	13	1	0.19	13	1	0.19	13	1	0.19
29	4	3	15	3	0.40	15	3	0.41	15	3	0.41	40	4	1.53	15	3	0.39
30	3	2	27	2	0.59	30	2	0.57	25	2	0.46	22	2	0.45	25	2	0.44
31	4	3	14	2	0.34	14	2	0.34	14	2	0.33	14	2	0.34	21	2	0.59
32	4	4	42	4	1.73	61	6	2.28	63	7	2.51	47	4	1.86	54	6	1.92
33	3	3	13	1	0.20	13	1	0.20	13	1	0.20	13	1	0.20	13	1	0.19
34	3	3	14	2	0.23	14	2	0.23	14	2	0.23	14	2	0.23	23	2	0.45
35	4	3	44	4	1.58	57	4	1.95	116	15	5.45	31	4	1.07	155	17	10.06
36	4	2	13	1	0.27	13	1	0.27	13	1	0.26	13	1	0.26	13	1	0.27
37	3	3	13	1	0.20	13	1	0.20	13	1	0.19	13	1	0.19	13	1	0.19
38	4	2	99	8	4.24	67	8	2.35	61	7	2.14	69	6	2.66	74	9	2.82
39	3	2	13	1	0.18	13	1	0.17	13	1	0.17	13	1	0.17	13	1	0.17
40	3	2	3	0	0.03	3	0	0.03	3	0	0.03	3	0	0.03	3	0	0.03
41	3	2	27	2	0.59	30	2	0.57	27	2	0.50	24	2	0.43	25	2	0.43
42	3	2	28	3	0.66	41	4	0.93	35	4	0.75	26	3	0.59	28	4	0.56
43	4	3	13	1	0.23	13	1	0.23	13	1	0.23	13	1	0.23	13	1	0.23
44	4	3	15	3	0.40	15	3	0.39	15	3	0.37	30	3	0.97	15	3	0.37
45	4	4	21	2	0.70	21	2	0.71	21	2	0.70	21	2	0.70	21	2	0.69
46	4	4	14	2	0.34	14	2	0.34	14	2	0.33	21	2	0.66	14	2	0.33
47	3	2	29	3	0.68	32	3	0.64	27	3	0.50	22	2	0.46	25	2	0.43
48	3	3	13	1	0.20	13	1	0.20	13	1	0.19	13	1	0.19	13	1	0.19
49	4	4	16	4	0.54	16	4	0.54	16	4	0.54	23	4	0.91	49	4	2.29
50	3	3	15	3	0.28	15	3	0.27	15	3	0.27	32	3	0.84	22	3	0.51

Table A.9: Performance of coarse-grained multi-method planners over 100 testing problems in the blocks-world domain.

Pbm	$M_1 \rightarrow M_2 \rightarrow M_5$			$M_1 \rightarrow M_4 \rightarrow M_5$			$M_1 \rightarrow M_5$			$M_2 \rightarrow M_5$			$M_4 \rightarrow M_5$		
N B G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
51 4 3	15	3	0.40	15	3	0.40	15	3	0.39	44	3	1.79	15	3	0.39
52 3 3	13	1	0.19	13	1	0.20	13	1	0.18	13	1	0.19	13	1	0.19
53 3 2	14	2	0.21	14	2	0.22	14	2	0.22	14	2	0.21	23	2	0.43
54 4 3	15	3	0.41	15	3	0.40	15	3	0.39	29	3	0.92	15	3	0.38
55 4 4	16	4	0.50	16	4	0.51	16	4	0.49	54	4	2.16	16	4	0.48
56 3 2	13	1	0.18	13	1	0.18	13	1	0.18	13	1	0.17	13	1	0.17
57 4 4	14	2	0.35	14	2	0.34	14	2	0.34	21	2	0.65	14	2	0.33
58 3 3	14	2	0.23	14	2	0.23	14	2	0.23	31	3	0.80	14	2	0.22
59 4 4	118	6	6.16	49	5	1.78	59	6	2.22	57	5	2.41	61	7	2.39
60 3 2	3	0	0.03	3	0	0.03	3	0	0.04	3	0	0.03	3	0	0.03
61 3 3	14	2	0.23	14	2	0.23	14	2	0.22	29	2	0.72	14	2	0.22
62 3 2	61	8	1.65	50	9	1.28	43	9	1.07	55	8	1.49	41	7	0.98
63 3 2	14	2	0.21	14	2	0.21	14	2	0.21	14	2	0.21	21	2	0.43
64 3 3	14	2	0.22	14	2	0.22	14	2	0.22	21	2	0.46	14	2	0.21
65 3 2	14	2	0.21	14	2	0.22	14	2	0.21	29	2	0.66	14	2	0.20
66 4 2	13	1	0.24	13	1	0.24	13	1	0.24	13	1	0.24	13	1	0.23
67 3 3	14	2	0.22	14	2	0.22	14	2	0.22	21	2	0.45	14	2	0.22
68 4 2	3	0	0.03	3	0	0.03	3	0	0.04	3	0	0.05	3	0	0.04
69 4 3	57	4	3.73	170	12	10.86	211	17	15.33	31	4	1.09	106	14	4.63
70 3 2	32	3	0.68	31	3	0.61	26	3	0.48	27	3	0.53	26	3	0.46
71 3 2	13	1	0.18	13	1	0.18	13	1	0.18	13	1	0.17	13	1	0.17
72 3 3	13	1	0.20	13	1	0.19	13	1	0.19	13	1	0.19	13	1	0.19
73 4 2	13	1	0.24	13	1	0.24	13	1	0.24	13	1	0.23	13	1	0.23
74 4 3	14	2	0.30	14	2	0.31	14	2	0.30	35	2	1.25	14	2	0.30
75 3 2	38	4	0.95	38	3	0.80	35	3	0.68	33	4	0.79	33	3	0.65
76 3 2	14	2	0.22	14	2	0.21	14	2	0.21	14	2	0.22	21	2	0.42
77 4 3	107	4	5.24	51	5	1.82	43	4	1.53	145	10	7.21	77	8	3.13
78 4 2	27	2	0.74	41	4	1.21	36	4	1.04	31	3	0.97	25	2	0.54
79 4 2	45	4	1.53	40	3	1.16	43	4	1.25	50	6	2.03	51	4	1.74
80 3 2	3	0	0.03	3	0	0.04	3	0	0.03	3	0	0.04	3	0	0.03
81 3 3	15	3	0.27	15	3	0.27	15	3	0.27	22	3	0.52	24	3	0.49
82 3 3	14	2	0.23	14	2	0.23	14	2	0.22	21	2	0.46	14	2	0.22
83 3 3	14	2	0.23	14	2	0.24	14	2	0.22	14	2	0.23	23	2	0.46
84 3 3	33	3	0.71	31	3	0.65	26	3	0.50	30	3	0.62	26	3	0.50
85 3 3	15	3	0.28	15	3	0.27	15	3	0.26	36	3	0.95	31	3	0.76
86 3 2	13	1	0.19	13	1	0.18	13	1	0.18	13	1	0.18	13	1	0.18
87 4 2	13	1	0.27	13	1	0.27	13	1	0.27	13	1	0.26	13	1	0.26
88 4 2	38	3	1.17	41	3	1.11	69	6	2.25	50	4	1.74	34	3	0.89
89 3 2	13	1	0.18	13	1	0.17	13	1	0.17	13	1	0.17	13	1	0.17
90 4 2	37	4	1.35	108	13	4.56	66	10	2.56	41	5	1.60	43	4	1.37
91 3 2	34	4	0.74	31	3	0.62	26	3	0.47	27	3	0.53	26	3	0.47
92 4 4	16	4	0.56	16	4	0.54	16	4	0.53	23	4	0.89	44	4	1.85
93 4 2	80	7	3.41	40	3	1.14	34	3	0.90	49	5	1.75	60	5	2.11
94 4 2	13	1	0.23	13	1	0.23	13	1	0.22	13	1	0.22	13	1	0.22
95 4 4	15	3	0.46	15	3	0.45	15	3	0.45	15	3	0.44	29	3	1.03
96 3 2	27	2	0.59	32	2	0.64	27	2	0.49	24	2	0.43	27	2	0.49
97 3 3	14	2	0.23	14	2	0.23	14	2	0.22	21	2	0.47	14	2	0.22
98 3 3	15	3	0.26	15	3	0.26	15	3	0.26	32	3	0.77	15	3	0.26
99 3 3	15	3	0.27	15	3	0.27	15	3	0.26	22	3	0.52	24	3	0.49
100 4 4	15	3	0.46	15	3	0.45	15	3	0.45	63	3	3.75	24	3	0.78
Total	2613	258	86.32	2679	294	83.06	2604	303	82.17	2934	258	115.89	2838	293	92.04
SP	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Average	26.13	2.58	0.86	26.79	2.94	0.83	26.04	3.03	0.82	29.34	2.58	1.16	28.38	2.93	0.92

Table A.10: Performance of coarse-grained multi-method planners over 100 testing problems in the blocks-world domain (continued).

Pbm			$M_1 \rightarrow M_3 \rightarrow M_6$			$M_1 \rightarrow M_4 \rightarrow M_6$			$M_1 \rightarrow M_6$			$M_3 \rightarrow M_6$			$M_4 \rightarrow M_6$		
N	B	G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
1	4	3	93	7	4.75	87	6	3.95	91	7	4.33	69	5	3.30	91	7	4.35
2	3	3	14	2	0.23	14	2	0.23	14	2	0.23	14	2	0.23	23	2	0.45
3	4	2	43	3	1.66	46	3	1.42	41	3	1.25	38	3	1.38	33	3	0.88
4	4	4	14	2	0.36	14	2	0.36	14	2	0.37	14	2	0.36	21	2	0.67
5	4	3	15	3	0.40	15	3	0.39	15	3	0.39	31	4	1.11	33	3	1.00
6	4	4	14	2	0.38	14	2	0.37	14	2	0.35	14	2	0.36	23	2	0.69
7	3	2	47	4	1.24	58	5	1.37	43	5	0.98	39	4	0.98	44	5	1.13
8	4	2	13	1	0.24	13	1	0.23	13	1	0.22	13	1	0.21	13	1	0.22
9	4	4	68	5	3.66	136	9	7.53	90	7	4.15	64	5	3.02	207	12	18.71
10	3	3	13	1	0.20	13	1	0.20	13	1	0.19	13	1	0.18	13	1	0.18
11	4	3	14	2	0.33	14	2	0.33	14	2	0.32	14	2	0.33	21	2	0.60
12	4	2	14	2	0.29	14	2	0.30	14	2	0.29	14	2	0.28	21	2	0.54
13	3	2	50	3	1.32	63	4	1.51	41	3	1.01	37	3	0.94	41	3	1.06
14	3	2	14	2	0.22	14	2	0.22	14	2	0.21	14	2	0.21	21	2	0.42
15	3	2	14	2	0.22	14	2	0.21	14	2	0.22	14	2	0.21	23	2	0.43
16	4	4	111	8	6.86	77	5	3.25	72	5	3.08	54	5	2.62	64	5	2.40
17	4	2	42	3	1.29	130	8	4.76	65	5	2.11	53	4	1.81	193	12	10.74
18	3	2	13	1	0.18	13	1	0.18	13	1	0.17	13	1	0.17	13	1	0.18
19	3	2	13	1	0.20	13	1	0.19	13	1	0.18	13	1	0.18	13	1	0.18
20	4	3	67	5	2.60	106	9	5.07	72	5	2.53	55	5	2.14	40	3	1.18
21	3	3	3	0	0.03	3	0	0.03	3	0	0.04	3	0	0.03	3	0	0.04
22	4	3	51	4	1.92	71	6	2.55	64	4	2.36	39	4	1.40	128	8	6.50
23	4	3	15	3	0.40	15	3	0.40	15	3	0.39	37	3	1.25	15	3	0.38
24	4	3	104	6	5.10	180	12	13.95	166	12	10.00	62	4	2.80	108	8	5.40
25	4	4	76	5	3.62	63	4	2.41	106	7	5.19	68	4	2.89	68	6	2.99
26	3	3	13	1	0.19	13	1	0.19	13	1	0.19	13	1	0.18	13	1	0.18
27	4	3	15	3	0.40	15	3	0.40	15	3	0.39	45	3	1.90	15	3	0.38
28	3	3	13	1	0.20	13	1	0.20	13	1	0.19	13	1	0.19	13	1	0.18
29	4	3	15	3	0.40	15	3	0.40	15	3	0.39	53	4	1.95	15	3	0.39
30	3	2	27	2	0.58	30	2	0.56	25	2	0.43	22	2	0.45	25	2	0.43
31	4	3	14	2	0.32	14	2	0.33	14	2	0.34	14	2	0.33	23	2	0.63
32	4	4	75	5	3.19	101	6	4.28	88	6	4.23	71	5	3.38	48	4	1.63
33	3	3	13	1	0.19	13	1	0.19	13	1	0.19	13	1	0.18	13	1	0.19
34	3	3	14	2	0.23	14	2	0.23	14	2	0.22	14	2	0.23	23	2	0.45
35	4	3	52	5	2.30	170	12	10.35	170	13	11.43	47	5	2.54	67	5	2.32
36	4	2	13	1	0.26	13	1	0.27	13	1	0.26	13	1	0.26	13	1	0.27
37	3	3	13	1	0.19	13	1	0.20	13	1	0.19	13	1	0.19	13	1	0.19
38	4	2	51	4	1.97	84	6	3.40	127	8	6.01	45	4	1.60	118	9	5.99
39	3	2	13	1	0.18	13	1	0.18	13	1	0.17	13	1	0.17	13	1	0.17
40	3	2	3	0	0.04	3	0	0.03	3	0	0.04	3	0	0.03	3	0	0.04
41	3	2	29	2	0.58	30	2	0.56	27	2	0.50	22	2	0.45	25	2	0.41
42	3	2	28	3	0.66	33	4	0.71	27	4	0.52	26	3	0.58	28	4	0.55
43	4	3	13	1	0.24	13	1	0.23	13	1	0.23	13	1	0.24	13	1	0.23
44	4	3	15	3	0.38	15	3	0.37	15	3	0.37	63	3	2.36	15	3	0.37
45	4	4	21	2	0.70	21	2	0.69	21	2	0.72	21	2	0.71	21	2	0.68
46	4	4	14	2	0.34	14	2	0.34	14	2	0.34	30	3	1.08	14	2	0.33
47	3	2	29	3	0.66	32	3	0.63	27	3	0.50	22	2	0.45	27	3	0.50
48	3	3	13	1	0.19	13	1	0.19	13	1	0.19	13	1	0.19	13	1	0.19
49	4	4	16	4	0.54	16	4	0.54	16	4	0.54	39	5	1.68	43	4	1.73
50	3	3	15	3	0.27	15	3	0.28	15	3	0.27	30	3	0.78	24	3	0.49

Table A.11: Performance of coarse-grained multi-method planners over 100 testing problems in the blocks-world domain (continued).

Pbm	$M_1 \rightarrow M_3 \rightarrow M_6$			$M_1 \rightarrow M_4 \rightarrow M_6$			$M_1 \rightarrow M_6$			$M_3 \rightarrow M_6$			$M_4 \rightarrow M_6$		
N B G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
51 4 3	15	3	0.40	15	3	0.39	15	3	0.39	69	5	2.98	15	3	0.39
52 3 3	13	1	0.19	13	1	0.19	13	1	0.19	13	1	0.18	13	1	0.18
53 3 2	14	2	0.21	14	2	0.21	14	2	0.22	14	2	0.21	21	2	0.43
54 4 3	15	3	0.40	15	3	0.40	15	3	0.40	29	3	0.91	15	3	0.38
55 4 4	16	4	0.49	16	4	0.50	16	4	0.51	46	5	1.91	16	4	0.49
56 3 2	13	1	0.18	13	1	0.18	13	1	0.17	13	1	0.17	13	1	0.17
57 4 4	14	2	0.35	14	2	0.34	14	2	0.35	21	2	0.65	14	2	0.34
58 3 3	14	2	0.23	14	2	0.23	14	2	0.22	45	3	1.28	14	2	0.22
59 4 4	92	6	4.39	143	9	7.84	132	7	10.90	53	5	2.27	91	6	4.07
60 3 2	3	0	0.04	3	0	0.04	3	0	0.03	3	0	0.04	3	0	0.04
61 3 3	14	2	0.23	14	2	0.24	14	2	0.22	21	2	0.46	14	2	0.22
62 3 2	47	5	1.31	96	11	3.24	62	7	1.71	23	3	0.50	59	5	1.58
63 3 2	14	2	0.22	14	2	0.22	14	2	0.21	14	2	0.20	23	2	0.43
64 3 3	14	2	0.22	14	2	0.22	14	2	0.22	21	2	0.45	14	2	0.22
65 3 2	14	2	0.22	14	2	0.20	14	2	0.20	21	2	0.43	14	2	0.21
66 4 2	13	1	0.24	13	1	0.24	13	1	0.24	13	1	0.24	13	1	0.24
67 3 3	14	2	0.22	14	2	0.22	14	2	0.22	29	2	0.69	14	2	0.21
68 4 2	3	0	0.04	3	0	0.04	3	0	0.03	3	0	0.03	3	0	0.03
69 4 3	67	5	2.92	525	13	90.83	201	17	13.29	38	4	1.36	141	10	7.47
70 3 2	34	4	0.73	33	4	0.68	28	4	0.54	29	4	0.59	28	4	0.53
71 3 2	13	1	0.18	13	1	0.17	13	1	0.18	13	1	0.18	13	1	0.17
72 3 3	13	1	0.20	13	1	0.19	13	1	0.19	13	1	0.19	13	1	0.19
73 4 2	13	1	0.24	13	1	0.24	13	1	0.24	13	1	0.23	13	1	0.25
74 4 3	14	2	0.31	14	2	0.31	14	2	0.30	21	2	0.58	14	2	0.30
75 3 2	42	3	1.04	46	3	1.17	41	3	0.96	46	3	1.26	50	3	1.08
76 3 2	14	2	0.22	14	2	0.22	14	2	0.21	14	2	0.20	21	2	0.42
77 4 3	239	4	13.05	41	4	1.31	35	4	1.07	254	5	13.11	140	11	7.60
78 4 2	27	2	0.72	30	2	0.70	25	2	0.54	22	2	0.57	25	2	0.54
79 4 2	36	3	1.14	40	3	1.13	34	3	0.92	30	3	0.90	49	3	1.51
80 3 2	3	0	0.04	3	0	0.03	3	0	0.03	3	0	0.04	3	0	0.04
81 3 3	15	3	0.27	15	3	0.26	15	3	0.26	22	3	0.52	24	3	0.49
82 3 3	14	2	0.23	14	2	0.22	14	2	0.22	29	2	0.71	14	2	0.22
83 3 3	14	2	0.24	14	2	0.23	14	2	0.23	14	2	0.23	21	2	0.48
84 3 3	35	3	0.77	33	3	0.72	28	3	0.57	30	3	0.62	28	3	0.57
85 3 3	15	3	0.27	15	3	0.27	15	3	0.26	30	3	0.79	31	3	0.76
86 3 2	13	1	0.19	13	1	0.18	13	1	0.18	13	1	0.19	13	1	0.18
87 4 2	13	1	0.27	13	1	0.26	13	1	0.26	13	1	0.26	13	1	0.26
88 4 2	78	6	3.20	46	3	1.28	57	3	1.67	54	4	2.05	41	3	1.12
89 3 2	13	1	0.18	13	1	0.18	13	1	0.18	13	1	0.18	13	1	0.18
90 4 2	135	8	8.30	172	8	8.16	172	9	10.71	108	7	5.45	125	9	6.50
91 3 2	34	4	0.73	33	4	0.68	28	4	0.53	27	3	0.52	26	3	0.46
92 4 4	16	4	0.54	16	4	0.54	16	4	0.54	63	6	2.96	39	4	1.51
93 4 2	51	3	1.91	63	3	2.09	65	4	2.27	53	4	2.07	118	6	8.52
94 4 2	13	1	0.23	13	1	0.23	13	1	0.22	13	1	0.23	13	1	0.22
95 4 4	15	3	0.44	15	3	0.44	15	3	0.44	15	3	0.44	38	3	1.55
96 3 2	29	2	0.57	30	2	0.55	25	2	0.42	24	2	0.45	25	2	0.42
97 3 3	14	2	0.22	14	2	0.23	14	2	0.23	30	3	0.78	14	2	0.23
98 3 3	15	3	0.26	15	3	0.26	15	3	0.25	39	3	1.01	15	3	0.25
99 3 3	15	3	0.27	15	3	0.27	15	3	0.27	22	3	0.52	22	3	0.50
100 4 4	15	3	0.45	15	3	0.45	15	3	0.44	38	4	1.54	24	3	0.78
Total	2891	259	102.81	3730	302	206.58	3177	297	124.46	3067	257	104.82	3447	295	135.39
SP	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Average	28.91	2.59	1.03	37.30	3.02	2.07	31.77	2.97	1.24	30.67	2.57	1.05	34.47	2.95	1.35

Table A.12: Performance of coarse-grained multi-method planners over 100 testing problems in the blocks-world domain (continued).

Problem			$M_{1\to 2\to 5}$			$M_{1\to 4\to 5}$			$M_{1\to 5}$			$M_{2\to 5}$			$M_{4\to 5}$		
N	B	G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
1	4	3	32	6	1.52	32	6	1.52	32	6	1.51	49	4	1.92	32	6	1.49
2	3	3	9	2	0.14	9	2	0.13	9	2	0.15	9	2	0.13	16	2	0.31
3	4	2	26	3	1.02	26	3	1.02	26	3	1.03	26	3	0.84	26	3	0.98
4	4	4	9	2	0.24	9	2	0.25	9	2	0.25	9	2	0.24	35	5	1.66
5	4	3	10	3	0.30	10	3	0.30	10	3	0.29	78	14	5.10	31	7	1.47
6	4	4	9	2	0.26	9	2	0.26	9	2	0.25	9	2	0.24	16	2	0.48
7	3	2	14	5	0.32	14	5	0.33	14	5	0.33	29	5	0.72	13	4	0.25
8	4	2	8	1	0.13	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.13
9	4	4	14	5	0.70	14	5	0.69	14	5	0.70	78	9	5.33	21	5	0.93
10	3	3	8	1	0.11	8	1	0.11	8	1	0.11	8	1	0.10	8	1	0.11
11	4	3	9	2	0.22	9	2	0.23	9	2	0.23	9	2	0.22	26	4	1.00
12	4	2	9	2	0.20	9	2	0.20	9	2	0.20	9	2	0.19	22	2	0.82
13	3	2	11	3	0.20	11	3	0.20	11	3	0.20	25	3	0.56	11	3	0.20
14	3	2	9	2	0.13	9	2	0.12	9	2	0.12	9	2	0.13	16	2	0.29
15	3	2	9	2	0.12	9	2	0.14	9	2	0.13	9	2	0.13	16	2	0.30
16	4	4	19	4	0.83	19	4	0.82	19	4	0.83	77	9	4.36	33	8	1.75
17	4	2	18	3	0.60	18	3	0.60	18	3	0.60	25	3	0.75	26	7	1.34
18	3	2	8	1	0.10	8	1	0.09	8	1	0.10	8	1	0.10	8	1	0.10
19	3	2	8	1	0.10	8	1	0.10	8	1	0.09	8	1	0.10	8	1	0.10
20	4	3	20	4	0.81	20	4	0.81	20	4	0.80	53	7	2.68	18	3	0.64
21	3	3	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.04
22	4	3	41	8	2.18	41	8	2.18	41	8	2.17	34	4	1.23	43	6	2.39
23	4	3	10	3	0.29	10	3	0.30	10	3	0.30	42	5	1.68	10	3	0.29
24	4	3	20	4	0.81	20	4	0.82	20	4	0.81	47	7	2.03	34	8	1.73
25	4	4	39	14	3.50	39	14	3.51	39	14	3.48	42	4	1.76	14	5	0.68
26	3	3	8	1	0.13	8	1	0.10	8	1	0.09	8	1	0.10	8	1	0.10
27	4	3	10	3	0.31	10	3	0.29	10	3	0.30	24	3	0.74	10	3	0.29
28	3	3	8	1	0.10	8	1	0.11	8	1	0.10	8	1	0.10	8	1	0.10
29	4	3	10	3	0.30	10	3	0.30	10	3	0.29	33	4	1.21	10	3	0.29
30	3	2	17	2	0.35	17	2	0.35	17	2	0.35	17	2	0.32	17	2	0.34
31	4	3	9	2	0.23	9	2	0.22	9	2	0.22	9	2	0.22	35	5	1.51
32	4	4	33	11	2.04	16	6	0.93	16	6	0.92	42	5	1.78	14	5	0.68
33	3	3	8	1	0.11	8	1	0.11	8	1	0.11	8	1	0.11	8	1	0.11
34	3	3	9	2	0.14	9	2	0.14	9	2	0.14	9	2	0.13	16	2	0.31
35	4	3	19	4	0.79	30	11	1.93	13	5	0.70	21	5	0.93	45	8	2.29
36	4	2	8	1	0.15	8	1	0.14	8	1	0.15	8	1	0.14	8	1	0.15
37	3	3	8	1	0.11	8	1	0.12	8	1	0.10	8	1	0.11	8	1	0.10
38	4	2	43	13	3.03	41	8	2.48	41	8	2.03	37	6	1.56	39	11	2.12
39	3	2	8	1	0.11	8	1	0.09	8	1	0.10	8	1	0.09	8	1	0.10
40	3	2	3	0	0.03	3	0	0.04	3	0	0.03	3	0	0.04	3	0	0.04
41	3	2	10	2	0.16	10	2	0.16	10	2	0.17	19	2	0.38	10	2	0.16
42	3	2	20	4	0.49	12	4	0.26	20	4	0.48	20	4	0.45	12	4	0.24
43	4	3	8	1	0.15	8	1	0.15	8	1	0.14	8	1	0.14	8	1	0.15
44	4	3	10	3	0.28	10	3	0.32	10	3	0.28	32	3	1.03	10	3	0.28
45	4	4	16	2	0.48	16	2	0.48	16	2	0.48	16	2	0.47	16	2	0.47
46	4	4	9	2	0.24	9	2	0.24	9	2	0.23	16	2	0.44	9	2	0.23
47	3	2	17	2	0.35	19	3	0.42	17	2	0.34	17	2	0.31	17	2	0.35
48	3	3	8	1	0.11	8	1	0.11	8	1	0.10	8	1	0.11	8	1	0.10
49	4	4	11	4	0.43	11	4	0.43	11	4	0.44	239	23	28.31	35	6	1.62
50	3	3	10	3	0.17	10	3	0.17	10	3	0.18	17	3	0.37	22	6	0.59

Table A.13: Performance of fine-grained multi-method planners over 100 testing problems in the blocks-world.

Problem			$M_{1\to2\to5}$			$M_{1\to4\to5}$			$M_{1\to5}$			$M_{2\to5}$			$M_{4\to5}$		
N	B	G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
51	4	3	10	3	0.30	10	3	0.31	10	3	0.30	36	4	1.31	10	3	0.29
52	3	3	8	1	0.10	8	1	0.10	8	1	0.11	8	1	0.10	8	1	0.10
53	3	2	9	2	0.13	9	2	0.13	9	2	0.13	9	2	0.13	23	6	0.62
54	4	3	10	3	0.30	10	3	0.30	10	3	0.30	53	6	2.41	10	3	0.29
55	4	4	11	4	0.40	11	4	0.40	11	4	0.39	49	4	2.09	11	4	0.38
56	3	2	8	1	0.09	8	1	0.09	8	1	0.09	8	1	0.10	8	1	0.10
57	4	4	9	2	0.24	9	2	0.24	9	2	0.24	30	6	1.50	9	2	0.23
58	3	3	9	2	0.14	9	2	0.14	9	2	0.13	26	3	0.66	9	2	0.13
59	4	4	21	5	0.92	18	7	1.07	21	5	0.93	40	7	1.97	22	6	1.05
60	3	2	3	0	0.04	3	0	0.03	3	0	0.04	3	0	0.04	3	0	0.04
61	3	3	9	2	0.14	9	2	0.13	9	2	0.14	16	2	0.31	9	2	0.14
62	3	2	25	7	0.76	23	10	0.79	27	12	1.05	25	7	0.67	23	10	0.73
63	3	2	9	2	0.13	9	2	0.13	9	2	0.13	9	2	0.12	16	2	0.29
64	3	3	9	2	0.13	9	2	0.13	9	2	0.13	26	2	0.59	9	2	0.13
65	3	2	9	2	0.12	9	2	0.12	9	2	0.12	24	2	0.54	9	2	0.12
66	4	2	8	1	0.15	8	1	0.14	8	1	0.15	8	1	0.14	8	1	0.14
67	3	3	9	2	0.13	9	2	0.13	9	2	0.13	16	2	0.31	9	2	0.13
68	4	2	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.04
69	4	3	19	4	0.76	19	4	0.76	19	4	0.76	32	4	1.23	33	4	1.76
70	3	2	13	4	0.29	13	4	0.29	13	4	0.29	11	3	0.20	18	3	0.38
71	3	2	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.09	8	1	0.10
72	3	3	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.11
73	4	2	8	1	0.14	8	1	0.15	8	1	0.14	8	1	0.14	8	1	0.14
74	4	3	9	2	0.20	9	2	0.21	9	2	0.21	16	2	0.40	9	2	0.20
75	3	2	11	3	0.19	11	3	0.20	11	3	0.20	27	3	0.58	14	4	0.29
76	3	2	9	2	0.13	9	2	0.13	9	2	0.14	9	2	0.12	16	2	0.29
77	4	3	12	4	0.51	12	4	0.51	21	5	0.90	60	5	2.91	28	5	1.18
78	4	2	17	2	0.50	17	2	0.49	17	2	0.49	17	2	0.42	17	2	0.48
79	4	2	27	4	0.97	27	4	0.99	18	3	0.56	42	4	1.57	18	3	0.57
80	3	2	3	0	0.03	3	0	0.03	3	0	0.04	3	0	0.04	3	0	0.04
81	3	3	10	3	0.18	10	3	0.17	10	3	0.17	17	3	0.36	17	3	0.35
82	3	3	9	2	0.14	9	2	0.14	9	2	0.13	16	2	0.31	9	2	0.14
83	3	3	9	2	0.14	9	2	0.14	9	2	0.13	9	2	0.14	16	2	0.32
84	3	3	15	5	0.34	11	3	0.22	17	6	0.43	17	6	0.38	11	3	0.21
85	3	3	10	3	0.17	10	3	0.17	10	3	0.17	36	9	1.22	33	6	0.95
86	3	2	8	1	0.09	8	1	0.10	8	1	0.10	8	1	0.09	8	1	0.09
87	4	2	8	1	0.14	8	1	0.15	8	1	0.15	8	1	0.14	8	1	0.15
88	4	2	45	16	3.32	14	4	0.57	35	11	1.97	39	3	1.40	12	3	0.41
89	3	2	8	1	0.09	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.10
90	4	2	46	5	2.50	133	61	63.29	36	5	1.61	33	4	1.36	146	67	70.66
91	3	2	13	4	0.29	13	4	0.29	13	4	0.29	11	3	0.20	20	4	0.47
92	4	4	11	4	0.43	11	4	0.42	11	4	0.44	18	4	0.68	47	11	2.79
93	4	2	27	4	1.00	18	3	0.57	18	3	0.56	44	5	1.68	18	3	0.55
94	4	2	8	1	0.15	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14
95	4	4	10	3	0.32	10	3	0.33	10	3	0.33	10	3	0.32	38	11	2.29
96	3	2	10	2	0.16	10	2	0.16	10	2	0.16	19	2	0.37	10	2	0.16
97	3	3	9	2	0.13	9	2	0.13	9	2	0.14	29	2	0.77	9	2	0.13
98	3	3	10	3	0.16	10	3	0.16	10	3	0.17	37	3	1.04	10	3	0.16
99	3	3	10	3	0.17	10	3	0.18	10	3	0.17	17	3	0.35	17	3	0.34
100	4	4	10	3	0.36	10	3	0.33	10	3	0.34	28	5	1.31	36	6	1.72
Total			1301	290	44.12	1325	334	101.14	1251	281	39.51	2380	316	103.50	1738	377	124.82
SP			100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Average			13.01	2.90	0.44	13.25	3.34	1.01	12.51	2.81	0.40	23.80	3.16	1.03	17.38	3.77	1.25

Table A.14: Performance of fine-grained multi-method planners over 100 testing problems in the blocks-world (continued).

Problem			$M_{1\to3\to6}$			$M_{1\to4\to6}$			$M_{1\to6}$			$M_{3\to6}$			$M_{4\to6}$		
N	B	G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
1	4	3	47	7	2.42	47	7	2.43	47	7	2.42	56	5	2.84	47	7	2.42
2	3	3	9	2	0.14	9	2	0.13	9	2	0.13	9	2	0.13	16	2	0.31
3	4	2	25	3	0.87	25	3	0.87	25	3	0.86	33	3	1.11	25	3	0.86
4	4	4	9	2	0.24	9	2	0.24	9	2	0.24	9	2	0.25	16	2	0.45
5	4	3	10	3	0.29	10	3	0.29	10	3	0.28	33	4	1.20	75	13	4.63
6	4	4	9	2	0.25	9	2	0.27	9	2	0.25	9	2	0.25	16	2	0.47
7	3	2	20	4	0.45	20	4	0.45	20	4	0.45	34	4	0.89	27	4	0.68
8	4	2	8	1	0.13	8	1	0.13	8	1	0.13	8	1	0.14	8	1	0.13
9	4	4	37	6	1.96	37	6	1.97	37	6	1.98	59	5	2.99	20	5	0.86
10	3	3	8	1	0.10	8	1	0.11	8	1	0.11	8	1	0.11	8	1	0.12
11	4	3	9	2	0.22	9	2	0.22	9	2	0.22	9	2	0.22	42	5	1.78
12	4	2	9	2	0.19	9	2	0.20	9	2	0.19	9	2	0.20	16	2	0.36
13	3	2	11	3	0.21	11	3	0.19	11	3	0.20	32	3	0.85	18	3	0.40
14	3	2	9	2	0.13	9	2	0.13	9	2	0.12	9	2	0.13	21	5	0.52
15	3	2	9	2	0.13	9	2	0.13	9	2	0.13	9	2	0.13	16	2	0.29
16	4	4	37	6	1.89	61	17	4.46	34	9	2.08	49	5	2.54	26	4	1.09
17	4	2	20	4	0.76	34	8	1.64	20	4	0.77	48	4	1.75	18	3	0.59
18	3	2	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.10
19	3	2	8	1	0.09	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.10
20	4	3	20	4	0.80	40	8	2.15	28	5	1.33	50	5	2.06	27	4	1.17
21	3	3	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.04
22	4	3	28	5	1.31	29	6	1.37	31	7	1.56	34	4	1.25	62	9	3.23
23	4	3	10	3	0.29	10	3	0.29	10	3	0.29	32	3	1.09	10	3	0.28
24	4	3	52	10	3.03	49	8	2.67	38	7	1.82	57	4	2.79	61	12	3.45
25	4	4	28	5	1.42	12	4	0.54	28	5	1.40	63	4	2.88	28	5	1.38
26	3	3	8	1	0.10	8	1	0.10	8	1	0.11	8	1	0.10	8	1	0.11
27	4	3	10	3	0.30	10	3	0.30	10	3	0.28	40	3	1.78	10	3	0.28
28	3	3	8	1	0.10	8	1	0.11	8	1	0.11	8	1	0.10	8	1	0.11
29	4	3	10	3	0.29	10	3	0.30	10	3	0.30	48	4	1.89	10	3	0.29
30	3	2	17	2	0.33	17	2	0.33	17	2	0.33	17	2	0.31	17	2	0.33
31	4	3	9	2	0.22	9	2	0.22	9	2	0.22	9	2	0.23	52	7	2.33
32	4	4	28	5	1.29	20	4	0.85	12	4	0.53	66	5	3.37	29	6	1.51
33	3	3	8	1	0.10	8	1	0.11	8	1	0.11	8	1	0.10	8	1	0.10
34	3	3	9	2	0.14	9	2	0.14	9	2	0.13	9	2	0.13	16	2	0.32
35	4	3	19	4	0.76	21	10	1.51	21	9	1.88	21	5	0.93	51	11	2.71
36	4	2	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14
37	3	3	8	1	0.10	8	1	0.10	8	1	0.10	8	1	0.11	8	1	0.10
38	4	2	27	4	1.09	32	7	1.49	27	4	1.14	57	5	2.47	82	12	4.87
39	3	2	8	1	0.10	8	1	0.09	8	1	0.09	8	1	0.10	8	1	0.09
40	3	2	3	0	0.03	3	0	0.03	3	0	0.03	3	0	0.03	3	0	0.04
41	3	2	10	2	0.16	10	2	0.15	10	2	0.16	17	2	0.30	10	2	0.15
42	3	2	20	4	0.47	12	4	0.26	18	3	0.37	18	3	0.36	12	4	0.24
43	4	3	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14
44	4	3	10	3	0.27	10	3	0.28	10	3	0.28	58	3	2.34	10	3	0.27
45	4	4	16	2	0.48	16	2	0.47	16	2	0.48	16	2	0.48	16	2	0.47
46	4	4	9	2	0.23	9	2	0.23	9	2	0.23	16	2	0.43	9	2	0.23
47	3	2	17	2	0.34	19	3	0.42	19	3	0.40	19	3	0.37	19	3	0.41
48	3	3	8	1	0.10	8	1	0.10	8	1	0.11	8	1	0.10	8	1	0.11
49	4	4	11	4	0.41	11	4	0.42	11	4	0.42	18	4	0.66	38	4	1.69
50	3	3	10	3	0.17	10	3	0.17	10	3	0.17	17	3	0.35	17	3	0.34

Table A.15: Performance of fine-grained multi-method planners over 100 testing problems in the blocks-world (continued).

Problem	$M_{1\to3\to6}$			$M_{1\to4\to6}$			$M_{1\to6}$			$M_{3\to6}$			$M_{4\to6}$		
N B G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
51 4 3	10	3	0.30	10	3	0.29	10	3	0.29	56	5	2.38	10	3	0.28
52 3 3	8	1	0.10	8	1	0.11	8	1	0.11	8	1	0.11	8	1	0.11
53 3 2	9	2	0.13	9	2	0.13	9	2	0.12	9	2	0.13	25	7	0.73
54 4 3	10	3	0.30	10	3	0.30	10	3	0.29	40	4	1.47	10	3	0.28
55 4 4	11	4	0.38	11	4	0.40	11	4	0.38	48	5	2.28	11	4	0.37
56 3 2	8	1	0.09	8	1	0.09	8	1	0.10	8	1	0.09	8	1	0.10
57 4 4	9	2	0.24	9	2	0.24	9	2	0.23	24	2	0.78	9	2	0.23
58 3 3	9	2	0.13	9	2	0.13	9	2	0.13	32	3	0.86	9	2	0.13
59 4 4	30	6	1.74	15	6	0.83	15	6	0.82	64	6	3.20	30	6	1.45
60 3 2	3	0	0.03	3	0	0.03	3	0	0.04	3	0	0.04	3	0	0.04
61 3 3	9	2	0.14	9	2	0.13	9	2	0.14	24	2	0.58	9	2	0.13
62 3 2	18	3	0.38	37	11	1.59	18	3	0.38	34	4	0.88	57	13	3.22
63 3 2	9	2	0.12	9	2	0.13	9	2	0.13	9	2	0.12	16	2	0.29
64 3 3	9	2	0.13	9	2	0.13	9	2	0.13	16	2	0.31	9	2	0.13
65 3 2	9	2	0.13	9	2	0.13	9	2	0.12	16	2	0.28	9	2	0.12
66 4 2	8	1	0.15	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14
67 3 3	9	2	0.13	9	2	0.12	9	2	0.13	24	2	0.57	9	2	0.13
68 4 2	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.04	3	0	0.04
69 4 3	48	8	2.65	53	12	3.37	27	4	1.11	33	4	1.18	102	19	7.52
70 3 2	13	4	0.28	11	3	0.21	13	4	0.28	13	4	0.27	20	4	0.46
71 3 2	8	1	0.10	8	1	0.10	8	1	0.09	8	1	0.09	8	1	0.10
72 3 3	8	1	0.10	8	1	0.11	8	1	0.11	8	1	0.11	8	1	0.10
73 4 2	8	1	0.15	8	1	0.14	8	1	0.14	8	1	0.14	8	1	0.14
74 4 3	9	2	0.20	9	2	0.20	9	2	0.20	44	6	2.07	9	2	0.20
75 3 2	11	3	0.19	11	3	0.19	11	3	0.19	32	3	0.79	11	3	0.19
76 3 2	9	2	0.13	9	2	0.13	9	2	0.13	9	2	0.13	16	2	0.29
77 4 3	21	5	0.83	12	4	0.49	12	4	0.49	104	6	5.00	45	7	2.15
78 4 2	17	2	0.48	17	2	0.48	17	2	0.46	17	2	0.40	17	2	0.47
79 4 2	18	3	0.56	18	3	0.55	27	4	0.95	56	4	2.45	27	4	0.95
80 3 2	3	0	0.04	3	0	0.03	3	0	0.04	3	0	0.04	3	0	0.04
81 3 3	10	3	0.17	10	3	0.17	10	3	0.17	17	3	0.36	17	3	0.35
82 3 3	9	2	0.13	9	2	0.14	9	2	0.14	25	3	0.62	9	2	0.13
83 3 3	9	2	0.13	9	2	0.14	9	2	0.14	9	2	0.14	16	2	0.31
84 3 3	22	4	0.56	11	3	0.22	11	3	0.22	32	11	1.23	11	3	0.21
85 3 3	10	3	0.17	10	3	0.17	10	3	0.17	41	4	1.28	31	7	0.97
86 3 2	8	1	0.10	8	1	0.10	8	1	0.09	8	1	0.10	8	1	0.10
87 4 2	8	1	0.14	8	1	0.14	8	1	0.15	8	1	0.14	8	1	0.15
88 4 2	36	5	1.51	19	3	0.61	36	5	1.50	49	3	1.75	19	3	0.61
89 3 2	8	1	0.09	8	1	0.09	8	1	0.10	8	1	0.10	8	1	0.09
90 4 2	33	4	1.33	37	9	5.10	56	14	13.64	73	7	3.39	51	10	9.20
91 3 2	11	3	0.21	11	3	0.21	13	4	0.28	11	3	0.19	18	3	0.37
92 4 4	11	4	0.42	11	4	0.43	11	4	0.42	26	4	1.05	50	11	2.84
93 4 2	18	3	0.55	18	3	0.54	27	4	0.96	25	3	0.74	27	4	0.95
94 4 2	8	1	0.14	8	1	0.14	8	1	0.15	8	1	0.14	8	1	0.14
95 4 4	10	3	0.33	10	3	0.32	10	3	0.32	10	3	0.31	30	3	1.30
96 3 2	10	2	0.15	10	2	0.16	10	2	0.16	19	2	0.37	10	2	0.15
97 3 3	9	2	0.14	9	2	0.14	9	2	0.14	25	3	0.63	9	2	0.14
98 3 3	10	3	0.17	10	3	0.17	10	3	0.16	34	3	0.87	10	3	0.16
99 3 3	10	3	0.17	10	3	0.17	10	3	0.17	17	3	0.35	22	6	0.58
100 4 4	10	3	0.33	10	3	0.34	10	3	0.33	25	3	0.91	17	3	0.54
Total	1356	259	42.63	1363	297	50.04	1323	273	52.78	2422	271	84.96	1983	346	82.91
SP	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
Average	13.56	2.59	0.43	13.63	2.97	0.50	13.23	2.73	0.53	24.22	2.71	0.85	19.83	3.46	0.83

Table A.16: Performance of fine-grained multi-method planners over 100 testing problems in the blocks-world (continued).

Appendix B

Experimental Results: The Machine-Shop Scheduling Domain

This appendix gives the detailed numeric information from the experiments in the machine-shop scheduling domain. Appendix B.1 presents the experimental results for the six single-method planners over 30 training problems. Appendix B.2 presents the experimental results for the six single-method planners and the created multi-method planners over 100 test problems.

B.1 Performance over 30 Training Problems

Method M_1									
Problems		Decisions				Plan Length			
Number	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	5	17	23	25	21.7	3	3	3	3.0
2	5	24	24	26	24.7	3	3	3	3.0
3	5	23	23	23	23.0	3	3	3	3.0
4	5	17	23	17	19.0	3	3	3	3.0
5	5	26	24	26	25.3	3	3	3	3.0
6	5	22	22	22	22.0	2	2	2	2.0
7	5	-	-	-	-	-	-	-	-
8	5	39	32	39	36.7	4	4	4	4.0
9	5	-	-	-	-	-	-	-	-
10	5	16	16	16	16.0	2	2	2	2.0
11	5	-	-	-	-	-	-	-	-
12	5	23	23	17	21.0	3	3	3	3.0
13	5	16	16	16	16.0	2	2	2	2.0
14	5	25	25	17	22.3	3	3	3	3.0
15	5	24	24	24	24.0	3	3	3	3.0
16	5	25	25	17	22.3	3	3	3	3.0
17	5	-	-	-	-	-	-	-	-
18	5	-	-	-	-	-	-	-	-
19	5	-	-	-	-	-	-	-	-
20	5	24	24	40	29.3	3	3	3	3.0
21	5	8	8	8	8.0	1	1	1	1.0
22	5	-	-	-	-	-	-	-	-
23	5	25	25	23	24.3	3	3	3	3.0
24	5	40	40	24	34.7	3	3	3	3.0
25	5	32	32	34	32.7	4	4	4	4.0
26	5	16	16	16	16.0	2	2	2	2.0
27	5	8	8	8	8.0	1	1	1	1.0
28	5	-	-	-	-	-	-	-	-
29	5	16	16	16	16.0	2	2	2	2.0
30	5	24	24	24	24.0	3	3	3	3.0
Total Solved Problems		490	493	478	487.0	59	59	59	59.0
		22	22	22	22.0	22	22	22	22.0
Average		22.27	22.41	21.73	22.14	2.68	2.68	2.68	2.68

Table B.1: Performance of M_1 over 30 training problems in the machine-shop scheduling domain.

Method M_2									
Problems		Decisions				Plan Length			
Number	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	5	17	23	25	21.7	3	3	3	3.0
2	5	24	24	26	24.7	3	3	3	3.0
3	5	23	23	23	23.0	3	3	3	3.0
4	5	17	23	17	19.0	3	3	3	3.0
5	5	26	24	26	25.3	3	3	3	3.0
6	5	22	22	22	22.0	2	2	2	2.0
7	5	-	-	-	-	-	-	-	-
8	5	39	32	39	36.7	4	4	4	4.0
9	5	-	-	-	-	-	-	-	-
10	5	16	16	16	16.0	2	2	2	2.0
11	5	-	-	-	-	-	-	-	-
12	5	23	23	17	21.0	3	3	3	3.0
13	5	16	16	16	16.0	2	2	2	2.0
14	5	25	25	17	22.3	3	3	3	3.0
15	5	24	24	24	24.0	3	3	3	3.0
16	5	25	25	17	22.3	3	3	3	3.0
17	5	-	-	-	-	-	-	-	-
18	5	-	-	-	-	-	-	-	-
19	5	-	-	-	-	-	-	-	-
20	5	24	24	40	29.3	3	3	3	3.0
21	5	8	8	8	8.0	1	1	1	1.0
22	5	-	-	-	-	-	-	-	-
23	5	25	25	23	24.3	3	3	3	3.0
24	5	40	40	24	34.7	3	3	3	3.0
25	5	32	32	34	32.7	4	4	4	4.0
26	5	16	16	16	16.0	2	2	2	2.0
27	5	8	8	8	8.0	1	1	1	1.0
28	5	-	-	-	-	-	-	-	-
29	5	16	16	16	16.0	2	2	2	2.0
30	5	24	24	24	24.0	3	3	3	3.0
Total Solved Problems		490	493	478	487.0	59	59	59	59.0
		22	22	22	22.0	22	22	22	22.0
Average		22.27	22.41	21.73	22.14	2.68	2.68	2.68	2.68

Table B.2: Performance of M_2 over 30 training problems in the machine-shop scheduling domain.

Method M_3									
Problems		Decisions				Plan Length			
Number	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	5	17	23	25	21.7	3	3	3	3.0
2	5	24	24	26	24.7	3	3	3	3.0
3	5	23	23	23	23.0	3	3	3	3.0
4	5	17	23	17	19.0	3	3	3	3.0
5	5	26	24	26	25.3	3	3	3	3.0
6	5	22	22	22	22.0	2	2	2	2.0
7	5	-	-	-	-	-	-	-	-
8	5	39	32	39	36.7	4	4	4	4.0
9	5	-	-	-	-	-	-	-	-
10	5	16	16	16	16.0	2	2	2	2.0
11	5	-	-	-	-	-	-	-	-
12	5	23	23	17	21.0	3	3	3	3.0
13	5	16	16	16	16.0	2	2	2	2.0
14	5	25	25	17	22.3	3	3	3	3.0
15	5	24	24	24	24.0	3	3	3	3.0
16	5	25	25	17	22.3	3	3	3	3.0
17	5	-	-	-	-	-	-	-	-
18	5	-	-	-	-	-	-	-	-
19	5	-	-	-	-	-	-	-	-
20	5	24	24	40	29.3	3	3	3	3.0
21	5	8	8	8	8.0	1	1	1	1.0
22	5	-	-	-	-	-	-	-	-
23	5	25	25	23	24.3	3	3	3	3.0
24	5	40	40	24	34.7	3	3	3	3.0
25	5	32	32	34	32.7	4	4	4	4.0
26	5	16	16	16	16.0	2	2	2	2.0
27	5	8	8	8	8.0	1	1	1	1.0
28	5	-	-	-	-	-	-	-	-
29	5	16	16	16	16.0	2	2	2	2.0
30	5	24	24	24	24.0	3	3	3	3.0
Total Solved Problems		490 22	493 22	478 22	487.0 22.0	59 22	59 22	59 22	59.0 22.0
Average		22.27	22.41	21.73	22.14	2.68	2.68	2.68	2.68

Table B.3: Performance of M_3 over 30 training problems in the machine-shop scheduling domain.

Method M_4									
Problems		Decisions				Plan Length			
Number	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	5	40	32	57	43.0	5	4	8	5.7
2	5	39	32	32	34.3	4	4	4	4.0
3	5	39	48	48	45.0	4	6	6	5.3
4	5	39	40	39	39.3	4	5	4	4.3
5	5	39	32	32	34.3	4	4	4	4.0
6	5	23	32	32	29.0	2	4	4	3.3
7	5	72	55	56	61.0	9	6	7	7.3
8	5	48	41	57	48.7	6	6	8	6.7
9	5	48	40	57	48.3	6	5	8	6.3
10	5	16	23	23	20.7	2	2	2	2.0
11	5	16	25	25	22.0	2	4	4	3.3
12	5	39	39	40	39.3	4	4	5	4.3
13	5	24	32	24	26.7	3	4	3	3.3
14	5	39	63	56	52.7	4	7	7	6.0
15	5	32	24	32	29.3	4	3	4	3.7
16	5	48	40	65	51.0	6	5	9	6.7
17	5	24	24	40	29.3	3	3	5	3.7
18	5	42	40	41	41.0	7	5	6	6.0
19	5	40	40	40	40.0	5	5	5	5.0
20	5	41	57	41	46.3	6	8	6	6.7
21	5	24	24	24	24.0	3	3	3	3.0
22	5	40	55	39	44.7	5	6	4	5.0
23	5	40	40	40	40.0	5	5	5	5.0
24	5	73	41	41	51.7	10	6	6	7.3
25	5	41	32	41	38.0	6	4	6	5.3
26	5	23	24	16	21.0	2	3	2	2.3
27	5	16	15	15	15.3	2	1	1	1.3
28	5	48	56	73	59.0	6	7	10	7.7
29	5	16	23	16	18.3	2	2	2	2.0
30	5	24	24	40	29.3	3	3	5	3.7
Total		1093	1093	1182	1122.7	134	134	153	140.3
Solved Problems		30	30	30	30.0	30	30	30	30.0
Average		36.43	36.43	39.40	37.42	4.47	4.47	5.10	4.68

Table B.4: Performance of M_4 over 30 training problems in the machine-shop scheduling domain.

Method M_5									
Problems		Decisions				Plan Length			
Number	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	5	50	40	41	43.7	8	5	6	6.3
2	5	32	33	40	35.0	4	5	5	4.7
3	5	57	40	32	43.0	8	5	4	5.7
4	5	41	39	76	52.0	6	4	13	7.7
5	5	33	24	32	29.7	5	3	4	4.0
6	5	23	33	33	29.7	2	5	5	4.0
7	5	66	49	57	57.3	10	7	8	8.3
8	5	39	32	40	37.0	4	4	5	4.3
9	5	75	57	47	59.7	12	8	5	8.3
10	5	16	16	16	16.0	2	2	2	2.0
11	5	16	16	16	16.0	2	2	2	2.0
12	5	32	56	31	39.7	4	7	3	4.7
13	5	24	23	24	23.7	3	2	3	2.7
14	5	93	31	40	54.7	15	3	5	7.7
15	5	31	32	32	31.7	3	4	4	3.7
16	5	40	39	49	42.7	5	4	7	5.3
17	5	49	24	24	32.3	7	3	3	4.3
18	5	33	32	59	41.3	5	4	10	6.3
19	5	24	24	42	30.0	3	3	7	4.3
20	5	50	32	32	38.0	8	4	4	5.3
21	5	24	24	33	27.0	3	3	5	3.7
22	5	40	32	32	34.7	5	4	4	4.3
23	5	65	41	49	51.7	9	6	7	7.3
24	5	32	32	40	34.7	4	4	5	4.3
25	5	48	56	40	48.0	6	7	5	6.0
26	5	16	16	23	18.3	2	2	2	2.0
27	5	16	16	15	15.7	2	2	1	1.7
28	5	49	48	76	57.7	7	6	13	8.7
29	5	16	16	16	16.0	2	2	2	2.0
30	5	24	24	31	26.3	3	3	3	3.0
Total		1154	977	1118	1083.0	159	123	152	144.7
Solved Problems		30	30	30	30.0	30	30	30	30.0
Average		38.47	32.57	37.27	36.10	5.30	4.10	5.07	4.82

Table B.5: Performance of M_5 over 30 training problems in the machine-shop scheduling domain.

Method M_6									
Problems		Decisions				Plan Length			
Number	Goals	Trial 1	Trial 2	Trial 3	Average	Trial 1	Trial 2	Trial 3	Average
1	5	50	40	41	43.7	8	5	6	6.3
2	5	32	33	40	35.0	4	5	5	4.7
3	5	57	40	32	43.0	8	5	4	5.7
4	5	41	39	76	52.0	6	4	13	7.7
5	5	33	24	32	29.7	5	3	4	4.0
6	5	23	33	33	29.7	2	5	5	4.0
7	5	66	49	57	57.3	10	7	8	8.3
8	5	39	32	40	37.0	4	4	5	4.3
9	5	75	57	47	59.7	12	8	5	8.3
10	5	16	16	16	16.0	2	2	2	2.0
11	5	16	16	16	16.0	2	2	2	2.0
12	5	32	56	31	39.7	4	7	3	4.7
13	5	24	23	24	23.7	3	2	3	2.7
14	5	93	31	40	54.7	15	3	5	7.7
15	5	31	32	32	31.7	3	4	4	3.7
16	5	40	39	49	42.7	5	4	7	5.3
17	5	49	24	24	32.3	7	3	3	4.3
18	5	33	32	59	41.3	5	4	10	6.3
19	5	24	24	42	30.0	3	3	7	4.3
20	5	50	32	32	38.0	8	4	4	5.3
21	5	24	24	33	27.0	3	3	5	3.7
22	5	40	32	32	34.7	5	4	4	4.3
23	5	65	41	49	51.7	9	6	7	7.3
24	5	32	32	40	34.7	4	4	5	4.3
25	5	48	56	40	48.0	6	7	5	6.0
26	5	16	16	23	18.3	2	2	2	2.0
27	5	16	16	15	15.7	2	2	1	1.7
28	5	49	48	76	57.7	7	6	13	8.7
29	5	16	16	16	16.0	2	2	2	2.0
30	5	24	24	31	26.3	3	3	3	3.0
Total		1154	977	1118	1083.0	159	123	152	144.7
Solved Problems		30	30	30	30.0	30	30	30	30.0
Average		38.47	32.57	37.27	36.10	5.30	4.10	5.07	4.82

Table B.6: Performance of M_6 over 30 training problems in the machine-shop scheduling domain.

B.2 Performance over 100 Testing Problems

The entries in the tables are defined as follows:

N: Problem number

B: Number of blocks

G: Number of goal conjuncts

SP: Solved problems.

D: Number of decisions

L: Plan length

C: CPU time (sec.)

Problem		M_1			M_2			M_3			M_4			M_5			M_6		
N	G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
1	5	23	3	0.67	23	3	0.67	23	3	0.68	40	5	1.23	40	5	1.24	40	5	1.25
2	5	26	3	0.53	26	3	0.53	26	3	0.54	41	6	1.05	41	6	1.05	41	6	1.06
3	5	17	3	0.33	17	3	0.33	17	3	0.33	71	8	2.32	71	8	2.33	71	8	2.34
4	5	17	3	0.33	17	3	0.33	17	3	0.33	50	8	1.73	50	8	1.74	50	8	1.74
5	5	26	3	0.54	26	3	0.54	26	3	0.54	41	6	1.07	41	6	1.07	41	6	1.07
6	5	16	2	0.30	16	2	0.30	16	2	0.31	42	7	1.53	42	7	1.54	42	7	1.55
7	5	-	-	-	-	-	-	-	-	-	40	5	0.99	40	5	0.98	40	5	0.99
8	5	32	4	0.77	32	4	0.77	32	4	0.79	32	4	0.75	32	4	0.76	32	4	0.75
9	5	-	-	-	-	-	-	-	-	-	57	8	1.70	57	8	1.67	57	8	1.67
10	5	16	2	0.31	16	2	0.31	16	2	0.31	23	2	0.48	23	2	0.49	23	2	0.49
11	5	-	-	-	-	-	-	-	-	-	34	6	1.15	34	6	1.15	34	6	1.18
12	5	25	3	0.74	25	3	0.74	25	3	0.77	50	8	1.78	50	8	1.78	50	8	1.77
13	5	16	2	0.31	16	2	0.31	16	2	0.31	24	3	0.55	24	3	0.55	24	3	0.56
14	5	17	3	0.33	17	3	0.33	17	3	0.33	55	6	1.43	55	6	1.43	55	6	1.43
15	5	24	3	0.50	24	3	0.50	24	3	0.51	24	3	0.52	24	3	0.52	24	3	0.52
16	5	17	3	0.33	17	3	0.33	17	3	0.33	81	11	2.97	81	11	2.96	81	11	2.96
17	5	-	-	-	-	-	-	-	-	-	24	3	0.55	24	3	0.56	24	3	0.56
18	5	-	-	-	-	-	-	-	-	-	58	9	1.76	58	9	1.76	58	9	1.77
19	5	-	-	-	-	-	-	-	-	-	47	5	1.29	47	5	1.27	47	5	1.28
20	5	24	3	0.52	24	3	0.52	24	3	0.49	32	4	0.76	32	4	0.75	32	4	0.75
21	5	8	1	0.17	8	1	0.17	8	1	0.17	15	1	0.29	15	1	0.30	15	1	0.29
22	5	-	-	-	-	-	-	-	-	-	41	6	1.08	41	6	1.08	41	6	1.07
23	5	25	3	0.75	25	3	0.75	25	3	0.80	41	6	1.08	41	6	1.08	41	6	1.07
24	5	30	3	0.77	30	3	0.77	30	3	0.78	40	5	0.99	40	5	0.99	40	5	0.99
25	5	34	4	0.76	34	4	0.76	34	4	0.76	41	6	1.04	41	6	1.06	41	6	1.05
26	5	16	2	0.31	16	2	0.31	16	2	0.31	16	2	0.31	16	2	0.31	16	2	0.31
27	5	8	1	0.17	8	1	0.17	8	1	0.17	15	1	0.29	15	1	0.30	15	1	0.30
28	5	-	-	-	-	-	-	-	-	-	56	7	1.59	56	7	1.59	56	7	1.59
29	5	16	2	0.31	16	2	0.31	16	2	0.31	16	2	0.31	16	2	0.31	16	2	0.31
30	5	24	3	0.51	24	3	0.51	24	3	0.51	24	3	0.52	24	3	0.51	24	3	0.52
31	5	8	1	0.16	8	1	0.16	8	1	0.17	15	1	0.30	15	1	0.29	15	1	0.29
32	5	-	-	-	-	-	-	-	-	-	48	6	1.23	48	6	1.24	48	6	1.23
33	5	26	3	0.54	26	3	0.54	26	3	0.54	33	5	0.81	33	5	0.81	33	5	0.81
34	5	24	3	0.50	24	3	0.50	24	3	0.52	24	3	0.51	24	3	0.50	24	3	0.51
35	5	24	3	0.52	24	3	0.52	24	3	0.51	24	3	0.51	24	3	0.51	24	3	0.51
36	5	-	-	-	-	-	-	-	-	-	40	5	0.99	40	5	0.99	40	5	0.99
37	5	-	-	-	-	-	-	-	-	-	31	3	0.69	31	3	0.69	31	3	0.69
38	5	-	-	-	-	-	-	-	-	-	32	4	0.76	32	4	0.77	32	4	0.77
39	5	-	-	-	-	-	-	-	-	-	34	6	1.07	34	6	1.07	34	6	1.08
40	5	16	2	0.30	16	2	0.30	16	2	0.31	16	2	0.31	16	2	0.31	16	2	0.31
41	5	16	2	0.31	16	2	0.31	16	2	0.31	16	2	0.32	16	2	0.31	16	2	0.31
42	5	-	-	-	-	-	-	-	-	-	40	5	0.96	40	5	0.96	40	5	0.96
43	5	-	-	-	-	-	-	-	-	-	40	5	0.97	40	5	0.95	40	5	0.97
44	5	16	2	0.30	16	2	0.30	16	2	0.30	23	2	0.47	23	2	0.47	23	2	0.46
45	5	33	4	1.01	33	4	1.01	33	4	0.95	73	10	2.31	73	10	2.30	73	10	2.29
46	5	24	3	0.54	24	3	0.54	24	3	0.51	24	3	0.52	24	3	0.53	24	3	0.52
47	5	16	2	0.32	16	2	0.32	16	2	0.30	23	2	0.47	23	2	0.47	23	2	0.47
48	5	24	2	0.75	24	2	0.75	24	2	0.72	33	5	0.96	33	5	0.97	33	5	0.97
49	5	23	3	0.59	23	3	0.59	23	3	0.58	49	7	1.41	49	7	1.43	49	7	1.43
50	5	-	-	-	-	-	-	-	-	-	65	9	2.02	65	9	2.02	65	9	2.03

Table B.7: Performance of single-method planners over 100 testing problems in the machine-shop scheduling domain.

Problem		M_1			M_2			M_3			M_4			M_5			M_6		
N	G	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C	D	L	C
51	5	8	1	0.18	8	1	0.18	8	1	0.16	8	1	0.17	8	1	0.17	8	1	0.16
52	5	24	3	0.53	24	3	0.53	24	3	0.53	68	12	2.85	68	12	2.84	68	12	2.84
53	5	-	-	-	-	-	-	-	-	-	32	4	0.74	32	4	0.74	32	4	0.74
54	5	-	-	-	-	-	-	-	-	-	63	7	1.78	63	7	1.79	63	7	1.79
55	5	26	3	0.54	26	3	0.54	26	3	0.54	40	5	0.99	40	5	0.98	40	5	0.99
56	5	-	-	-	-	-	-	-	-	-	64	8	1.71	64	8	1.72	64	8	1.73
57	5	22	2	0.56	22	2	0.56	22	2	0.55	42	7	1.53	42	7	1.52	42	7	1.55
58	5	-	-	-	-	-	-	-	-	-	16	2	0.36	16	2	0.36	16	2	0.36
59	5	8	1	0.18	8	1	0.18	8	1	0.17	16	2	0.37	16	2	0.37	16	2	0.36
60	5	16	2	0.33	16	2	0.33	16	2	0.31	24	3	0.55	24	3	0.55	24	3	0.55
61	5	-	-	-	-	-	-	-	-	-	25	4	0.69	25	4	0.69	25	4	0.69
62	5	16	2	0.32	16	2	0.32	16	2	0.32	16	2	0.31	16	2	0.31	16	2	0.31
63	5	24	3	0.53	24	3	0.53	24	3	0.52	31	3	0.68	31	3	0.68	31	3	0.69
64	5	24	2	0.76	24	2	0.76	24	2	0.74	23	2	0.47	23	2	0.47	23	2	0.48
65	5	-	-	-	-	-	-	-	-	-	40	5	0.99	40	5	1.00	40	5	0.99
66	5	8	1	0.18	8	1	0.18	8	1	0.16	16	2	0.36	16	2	0.37	16	2	0.36
67	5	3	0	0.06	3	0	0.06	3	0	0.05	3	0	0.05	3	0	0.05	3	0	0.05
68	5	24	3	0.53	24	3	0.53	24	3	0.51	24	3	0.51	24	3	0.52	24	3	0.51
69	5	24	3	0.52	24	3	0.52	24	3	0.50	32	4	0.75	32	4	0.76	32	4	0.76
70	5	34	4	0.79	34	4	0.79	34	4	0.77	49	7	1.31	49	7	1.29	49	7	1.32
71	5	24	2	0.75	24	2	0.75	24	2	0.73	24	3	0.55	24	3	0.55	24	3	0.54
72	5	16	2	0.32	16	2	0.32	16	2	0.32	16	2	0.32	16	2	0.31	16	2	0.31
73	5	39	4	1.02	39	4	1.02	39	4	0.99	47	5	1.17	47	5	1.17	47	5	1.18
74	5	24	2	0.78	24	2	0.78	24	2	0.74	33	5	0.99	33	5	0.98	33	5	0.99
75	5	22	2	0.59	22	2	0.59	22	2	0.55	32	4	0.91	32	4	0.91	32	4	0.90
76	5	16	2	0.33	16	2	0.33	16	2	0.31	16	2	0.30	16	2	0.31	16	2	0.31
77	5	16	2	0.31	16	2	0.31	16	2	0.30	16	2	0.30	16	2	0.31	16	2	0.31
78	5	-	-	-	-	-	-	-	-	-	48	6	1.36	48	6	1.36	48	6	1.36
79	5	-	-	-	-	-	-	-	-	-	32	4	0.74	32	4	0.74	32	4	0.74
80	5	-	-	-	-	-	-	-	-	-	41	6	1.20	41	6	1.20	41	6	1.20
81	5	24	2	0.74	24	2	0.74	24	2	0.72	60	11	2.57	60	11	2.58	60	11	2.57
82	5	-	-	-	-	-	-	-	-	-	31	3	0.69	31	3	0.69	31	3	0.69
83	5	8	1	0.18	8	1	0.18	8	1	0.17	16	2	0.37	16	2	0.37	16	2	0.37
84	5	16	2	0.32	16	2	0.32	16	2	0.30	24	3	0.55	24	3	0.56	24	3	0.56
85	5	24	3	0.54	24	3	0.54	24	3	0.51	32	4	0.73	32	4	0.74	32	4	0.73
86	5	24	2	0.74	24	2	0.74	24	2	0.72	24	3	0.55	24	3	0.54	24	3	0.54
87	5	-	-	-	-	-	-	-	-	-	42	7	1.55	42	7	1.54	42	7	1.53
88	5	31	4	0.83	31	4	0.83	31	4	0.80	58	9	1.78	58	9	1.78	58	9	1.78
89	5	46	3	1.36	46	3	1.36	46	3	1.32	32	4	0.77	32	4	0.77	32	4	0.76
90	5	-	-	-	-	-	-	-	-	-	17	3	0.41	17	3	0.39	17	3	0.40
91	5	24	2	0.76	24	2	0.76	24	2	0.73	24	3	0.54	24	3	0.55	24	3	0.55
92	5	8	1	0.17	8	1	0.17	8	1	0.16	15	1	0.30	15	1	0.29	15	1	0.30
93	5	-	-	-	-	-	-	-	-	-	24	3	0.55	24	3	0.56	24	3	0.55
94	5	16	2	0.32	16	2	0.32	16	2	0.31	16	2	0.31	16	2	0.31	16	2	0.31
95	5	8	1	0.17	8	1	0.17	8	1	0.17	24	3	0.61	24	3	0.61	24	3	0.61
96	5	22	2	0.58	22	2	0.58	22	2	0.55	24	3	0.55	24	3	0.54	24	3	0.54
97	5	17	3	0.36	17	3	0.36	17	3	0.33	40	5	1.00	40	5	0.98	40	5	0.98
98	5	-	-	-	-	-	-	-	-	-	32	4	0.77	32	4	0.77	32	4	0.76
99	5	24	3	0.53	24	3	0.53	24	3	0.52	24	3	0.51	24	3	0.51	24	3	0.51
100	5	34	4	0.79	34	4	0.79	34	4	0.77	47	5	1.16	47	5	1.18	47	5	1.17
Total		1451	170	44.19	1451	170	44.19	1451	170	43.38	3397	447	91.98	3397	447	92.00	3397	447	92.07
SP		70	70	70	70	70	70	70	70	70	100	100	100	100	100	100	100	100	100
AVG		20.73	2.43	0.63	20.73	2.43	0.63	20.73	2.43	0.62	33.97	4.47	0.92	33.97	4.47	0.92	33.97	4.47	0.92

Table B.8: Performance of single-method planners over 100 testing problems in the machine-shop scheduling domain (continued).

Problem		$M_2 \rightarrow M_5$			$M_3 \rightarrow M_6$			$M_{2 \rightarrow 5}$			$M_{3 \rightarrow 6}$		
N	G	D	L	C	D	L	C	D	L	C	D	L	C
1	5	28	3	0.83	28	3	0.84	18	4	0.51	18	4	0.52
2	5	31	3	0.71	31	3	0.70	27	6	0.71	27	6	0.71
3	5	22	3	0.50	22	3	0.50	18	4	0.42	18	4	0.42
4	5	22	3	0.49	22	3	0.50	17	3	0.35	17	3	0.35
5	5	31	3	0.70	31	3	0.71	27	6	0.73	27	6	0.70
6	5	21	2	0.46	21	2	0.46	17	3	0.41	17	3	0.41
7	5	58	5	1.43	58	5	1.43	26	5	0.68	26	5	0.67
8	5	39	4	0.96	39	4	0.96	32	4	0.84	32	4	0.84
9	5	74	7	1.90	74	7	1.92	27	6	0.71	27	6	0.72
10	5	21	2	0.49	21	2	0.49	16	2	0.33	16	2	0.33
11	5	24	2	0.57	24	2	0.58	9	2	0.24	9	2	0.24
12	5	22	3	0.51	22	3	0.50	18	4	0.43	18	4	0.43
13	5	21	2	0.49	21	2	0.48	16	2	0.34	16	2	0.33
14	5	30	3	0.86	30	3	0.85	18	4	0.43	18	4	0.43
15	5	29	3	0.68	29	3	0.69	24	3	0.57	24	3	0.57
16	5	28	3	0.76	28	3	0.75	18	4	0.43	18	4	0.43
17	5	67	7	2.01	67	7	1.97	17	3	0.41	17	3	0.41
18	5	49	6	1.37	49	6	1.38	21	7	0.50	21	7	0.50
19	5	60	7	2.05	60	7	2.05	17	3	0.41	17	3	0.41
20	5	61	3	1.88	61	3	1.89	24	3	0.55	24	3	0.55
21	5	13	1	0.25	13	1	0.26	8	1	0.16	8	1	0.16
22	5	90	12	3.30	90	12	3.30	17	3	0.37	17	3	0.38
23	5	22	3	0.49	22	3	0.50	18	4	0.43	18	4	0.43
24	5	29	3	0.71	29	3	0.69	24	3	0.56	24	3	0.56
25	5	39	4	0.94	39	4	0.95	32	4	0.80	32	4	0.81
26	5	21	2	0.47	21	2	0.48	16	2	0.33	16	2	0.33
27	5	13	1	0.24	13	1	0.24	8	1	0.17	8	1	0.16
28	5	73	6	1.85	73	6	1.89	28	7	0.74	28	7	0.75
29	5	21	2	0.49	21	2	0.47	16	2	0.33	16	2	0.34
30	5	29	3	0.67	29	3	0.68	24	3	0.56	24	3	0.56
31	5	13	1	0.25	13	1	0.25	8	1	0.16	8	1	0.17
32	5	111	16	4.47	111	16	4.47	26	5	0.67	26	5	0.68
33	5	31	3	0.71	31	3	0.71	24	3	0.54	24	3	0.57
34	5	29	3	0.69	29	3	0.70	24	3	0.55	24	3	0.55
35	5	29	3	0.68	29	3	0.68	24	3	0.56	24	3	0.56
36	5	99	14	3.99	99	14	3.98	18	4	0.45	18	4	0.44
37	5	40	4	0.94	40	4	0.93	17	3	0.37	17	3	0.38
38	5	41	4	1.00	41	4	1.01	18	4	0.45	18	4	0.45
39	5	33	4	0.86	33	4	0.86	18	4	0.51	18	4	0.51
40	5	21	2	0.47	21	2	0.47	16	2	0.33	16	2	0.33
41	5	21	2	0.48	21	2	0.48	16	2	0.32	16	2	0.33
42	5	49	5	1.15	49	5	1.15	19	5	0.47	19	5	0.47
43	5	60	5	1.74	60	5	1.74	26	5	0.68	26	5	0.68
44	5	21	2	0.47	21	2	0.47	16	2	0.33	16	2	0.33
45	5	36	4	0.98	36	4	1.00	25	4	0.60	25	4	0.60
46	5	29	3	0.69	29	3	0.69	24	3	0.56	24	3	0.56
47	5	21	2	0.47	21	2	0.47	17	3	0.41	17	3	0.40
48	5	27	2	0.73	27	2	0.74	16	2	0.32	16	2	0.33
49	5	28	3	0.75	28	3	0.75	18	4	0.43	18	4	0.44
50	5	58	5	1.40	58	5	1.38	26	5	0.69	26	5	0.68

Table B.9: Performance of multi-method planners over 100 testing problems in the machine-shop scheduling domain.

Problem		$M_2 \rightarrow M_5$			$M_3 \rightarrow M_6$			$M_{2 \rightarrow 5}$			$M_{3 \rightarrow 6}$		
N	G	D	L	C	D	L	C	D	L	C	D	L	C
51	5	13	1	0.24	13	1	0.25	8	1	0.16	8	1	0.16
52	5	60	3	2.04	60	3	2.04	24	3	0.56	24	3	0.57
53	5	67	5	1.91	67	5	1.91	26	5	0.70	26	5	0.70
54	5	102	14	3.97	102	14	3.96	28	7	0.79	28	7	0.79
55	5	29	3	0.69	29	3	0.69	26	5	0.67	26	5	0.68
56	5	75	10	2.19	75	10	2.21	19	5	0.42	19	5	0.42
57	5	21	2	0.46	21	2	0.47	17	3	0.40	17	3	0.40
58	5	42	6	1.36	42	6	1.36	9	2	0.24	9	2	0.24
59	5	13	1	0.25	13	1	0.24	8	1	0.17	8	1	0.17
60	5	27	2	0.72	27	2	0.72	17	3	0.40	17	3	0.40
61	5	24	2	0.57	24	2	0.57	9	2	0.24	9	2	0.24
62	5	21	2	0.47	21	2	0.48	16	2	0.33	16	2	0.33
63	5	29	3	0.68	29	3	0.68	24	3	0.55	24	3	0.56
64	5	29	2	0.83	29	2	0.82	16	2	0.33	16	2	0.33
65	5	59	4	1.65	59	4	1.65	26	5	0.68	26	5	0.69
66	5	13	1	0.24	13	1	0.24	8	1	0.17	8	1	0.16
67	5	3	0	0.04	3	0	0.05	3	0	0.05	3	0	0.05
68	5	29	3	0.68	29	3	0.68	24	3	0.54	24	3	0.55
69	5	45	3	1.24	45	3	1.24	24	3	0.56	24	3	0.55
70	5	37	4	0.93	37	4	0.93	34	6	0.95	34	6	0.96
71	5	29	2	0.82	29	2	0.82	17	3	0.41	17	3	0.42
72	5	21	2	0.47	21	2	0.47	16	2	0.33	16	2	0.33
73	5	37	4	0.93	37	4	0.91	27	6	0.72	27	6	0.73
74	5	29	2	0.81	29	2	0.82	17	3	0.42	17	3	0.40
75	5	29	2	0.80	29	2	0.80	16	2	0.33	16	2	0.33
76	5	21	2	0.47	21	2	0.47	16	2	0.32	16	2	0.33
77	5	21	2	0.47	21	2	0.47	16	2	0.33	16	2	0.33
78	5	48	5	1.27	48	5	1.28	18	4	0.45	18	4	0.45
79	5	66	4	1.83	66	4	1.85	26	5	0.68	26	5	0.68
80	5	40	4	0.93	40	4	0.93	20	6	0.52	20	6	0.51
81	5	21	2	0.47	21	2	0.48	16	2	0.33	16	2	0.34
82	5	42	3	1.04	42	3	1.04	17	3	0.41	17	3	0.41
83	5	13	1	0.24	13	1	0.25	8	1	0.17	8	1	0.17
84	5	21	2	0.46	21	2	0.47	16	2	0.33	16	2	0.33
85	5	29	3	0.70	29	3	0.69	24	3	0.56	24	3	0.56
86	5	27	2	0.71	27	2	0.72	17	3	0.40	17	3	0.41
87	5	85	11	3.07	85	11	3.09	17	3	0.42	17	3	0.42
88	5	30	4	0.74	30	4	0.73	26	5	0.67	26	5	0.67
89	5	35	3	0.94	35	3	0.94	25	4	0.64	25	4	0.64
90	5	32	3	0.77	32	3	0.78	17	3	0.44	17	3	0.44
91	5	29	2	0.82	29	2	0.82	17	3	0.40	17	3	0.40
92	5	13	1	0.25	13	1	0.25	8	1	0.17	8	1	0.17
93	5	57	4	1.40	57	4	1.40	17	3	0.41	17	3	0.41
94	5	21	2	0.47	21	2	0.48	16	2	0.34	16	2	0.33
95	5	13	1	0.25	13	1	0.25	8	1	0.16	8	1	0.17
96	5	29	2	0.81	29	2	0.82	16	2	0.33	16	2	0.34
97	5	28	3	0.76	28	3	0.75	18	4	0.42	18	4	0.43
98	5	34	4	0.83	34	4	0.83	18	4	0.45	18	4	0.48
99	5	29	3	0.68	29	3	0.69	24	3	0.56	24	3	0.57
100	5	39	4	0.96	39	4	0.96	33	5	0.87	33	5	0.88
Total		3591	358	98.31	3591	358	98.49	1907	329	45.75	1907	329	45.94
SP		100	100	100	100	100	100	100	100	100	100	100	100
AVG		35.91	3.58	0.98	35.91	3.58	0.98	19.07	3.29	0.46	19.07	3.29	0.46

Table B.10: Performance of multi-method planners over 100 testing problems in the machine-shop scheduling domain (continued).

Reference List

[Allen *et al.*, 1990] J. Allen, J. Hendler, and A. Tate. ***Readings in Planning.*** Morgan Kaufmann, San Mateo, CA, 1990.

[Barley, 1991] M Barley. Private communication. Private communication, 1991.

[Barrett and Weld, 1992] A. Barrett and D. S. Weld. Partial-order planning: Evaluating possible efficiency gains. Technical Report 92-05-01, Department of Computer Science and Engineering, University of Washington, 1992.

[Bhatnagar and Mostow, 1990] N. Bhatnagar and J. Mostow. Adaptive search by explanation-based learning of heuristic censors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 895–901, Boston, MA, 1990.

[Bond and Gasser, 1988] A. H. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann, San Mateo, CA, 1988.

[Boström, H., 1990] Boström, H. Generalizing the order of goals as an approach to generalizing numbers. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 260–267, Austin, TX, 1990. Morgan Kaufmann.

[Chapman, 1987] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.

[Cohen, 1988] W. W Cohen. Generalizing number and learning from multiple examples in explanation based learning. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 256–269, Ann Arbor, MI, 1988. Morgan Kaufmann.

[Etzioni, 1990a] O. Etzioni. Why PRODIGY/EBL works. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 916–922, Boston, MA, 1990.

[Etzioni, 1990b] Oren Etzioni. *A Structural Theory of Explanation-Based Learning.* Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, 1990. Available as Technical Report CMU-CS-90-185.

[Fikes and Nilsson, 1971] R. E. Fikes and N Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1971.

[Gratch and DeJong, 1990] J. M. Gratch and G. F. DeJong. A framework for evaluating search control strategies. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 337–347, San Diego, CA, 1990. Morgan Kaufmann.

[Greiner, 1992] Russell Greiner. Probablistic hill-climbing: Theory and applications. In *Proceedings of the Ninth Canadian Conference on Artificial Intelligence*, Vancouver, Canada, 1992.

[Johnson, 1994] W. L. Johnson. Agents that explain their own actions. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavior Representation*, Orlando, FL, 1994.

[Jones *et al.*, 1993] R.M. Jones, M. Tambe, J.E. Laird, and Rosenbloom P.S. Intelligent automated agents for flight training simulators. In *Proceedings of the 3rd Conference on Computer Generated Forces and Behavior Representation*, Orlando, FL, 1993.

[Kim and Rosenbloom, 1993] J. Kim and P. S. Rosenbloom. Constraining learning with search control. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 174–181, 1993.

[Korf, 1985] R. E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

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

[Laird *et al.*, 1987] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[Lee and Rosenbloom, 1992] S. Lee and P. S Rosenbloom. Creating and coordinating multiple planning methods. In *Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence*, pages 89–95, Seoul, Korea, 1992.

[Lee and Rosenbloom, 1993] S. Lee and P. S Rosenbloom. Granularity in multi-method planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 486–491, Washington D. C., 1993.

[McAllester and Rosenblitt, 1991] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634–639, Anaheim, CA, 1991.

[Michalski *et al.*, 1986] R.S. Michalski, I. Mozetic, J. Hong, and N Lavrac. The AQ15 inductive learning system: An overview and experiments. Technical Report UIUCDCS-R-86-1260, Department of Computer Science, University of Illinois at Urbana-Champaign, 1986.

[Minton *et al.*, 1989] S. Minton, C.A. Knoblock, D.R. Kuokka, Y. Gil, R.L. Joseph, and J.G. Carbonell. PRODIGY 2.0: The manual and tutorial. Technical Report CMU-CS-89-146, School of Computer Science, Carnegie Mellon University, 1989.

[Minton *et al.*, 1991] S. Minton, J. Bresina, and M. Drummond. Commitment strategies in planning: A comparative analysis. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, Australia, 1991.

[Minton, 1988] Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* Ph.D. Thesis, Computer Science Department, Carnegie Mellon University, 1988.

[Mitchell, 1980] T. M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Department of Computer Science, Rutgers University, 1980.

[Newell *et al.*, 1991] G. R. Newell, A. Yost, J. E. Laird, P. S. Rosenbloom, and E. Altmann. Formulating the problem space computational model. In R. F. Rashid, editor, *Carnegie Mellon Computer Science: A 25-Year Commemorative.* Addison-Wesley/ACM Press, Reading, MA, 1991.

[Newell, 1990] A Newell. *Unified Theories of Cognition.* Harvard University Press, Cambridge, MA, 1990.

[Rosenbloom and Laird, 1986] P. S. Rosenbloom and J. E. Laird. Mapping explanation-based generalization onto Soar. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 561–567, Philadelphia, PA, 1986.

[Rosenbloom *et al.*, 1990] P. S. Rosenbloom, S. Lee, and A Unruh. Responding to impasses in memory-driven behavior: A framework for planning. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pages 181–191, San Diego, CA, 1990. Morgan Kaufmann.

[Rosenbloom *et al.*, 1991] P. S. Rosenbloom, J. E. Laird, and A. Newell. A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47(1-3):289–325, 1991.

[Rosenbloom *et al.*, 1993] P. S. Rosenbloom, S. Lee, and A. Unruh. Bias in planning and explanation-based learning. In Chipman S. and Meyrowitz A., editors, *Machine Learning: Induction, Analogy and Discovery.* Kluwer Academic Publishers, Hingham, MA, 1993. Also available in S. Minton (Ed.) *Machine Learning Methods for Planning.* Morgan Kaufmann, San Mateo, CA, 1993.

[Rosenbloom *et al.*, 1994] P. S. Rosenbloom, W. L. Johnson, R. M. Jones, F. Koss, J. E. Laird, J. F. Lehman, R. Rubinoff, K. B. Schwamb, and M. Tambe. Intelligent automated agents for tactical air simulation: A progress report. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, 1994. In Press.

[Ruby and Kibler, 1991] D. Ruby and D. Kibler. Steppingstone: An empirical and analytical evaluation. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 527–532, Anaheim, CA, 1991.

[Russell and Grosof, 1987] S. J. Russell and B. N. Grosof. A declarative approach to bias in concept learning. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 505–510, Seattle, WA, 1987.

[Sacerdoti, 1975] E. D. Sacerdoti. The non-linear nature of plans. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, USSR, 1975.

[Shavlik, 1989] J. W. Shavlik. Acquiring recursive concepts with explanation-based learning. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 688–693, Detroit, MI, 1989.

[Subramanian and Feldman, 1990] D. Subramanian and R. Feldman. The utility of EBL in recursive domain theories. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 942–949, Boston, MA, 1990.

[Sussman, 1973] G.J. Sussman. A computational model of skill acquisition. Memo 297, AI Lab, MIT, 1973.

[Tambe and Newell, 1988] M. Tambe and A. Newell. Some chunks are expensive. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 451–458, 1988.

[Tambe and Rosenbloom, 1994] M. Tambe and P.S. Rosenbloom. Event tracking in complex multi-agent environments. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavior Representation*, Orlando, FL, 1994.

[Tambe *et al.*, 1990] M. Tambe, A. Newell, and P. S. Rosenbloom. The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, 5(3):299–348, 1990.

[Tate, 1977] A. Tate. Generating project networks. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 888–893, Cambridge, MA, 1977.

[Unruh, 1993] Any Unruh. *Using Automatic Abstraction for Problem-Solving and Learning.* Ph.D. Thesis, Department of Computer Science, Stanford University, 1993.

[Utgoff, 1986] P. E. Utgoff. Shift of bias for inductive concept learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach, Vol. II.* Morgan Kaufmann, Los Altos, CA, 1986.

[Veloso, 1989] M Veloso. Nonlinear problem solving using intelligent casual-commitment. Technical Report CMU-CS-89-210, School of Computer Science, Carnegie Mellon University, 1989.

[Vere, 1983] Steven A. Vere. Planning in time: Windows and durations for activities and goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(3):246–267, May 1983.

[Waldinger, 1977] R. Waldinger. Achieving several goals simultaneously. In D. Michie, editor, *Machine Intelligence 8*, pages 94–136. Ellis Horwood, Chichester, England, 1977.

[Warren, 1976] D. Warren. WARPLAN: A system for generating plans. Memo 76, Department of Computational Logic, University of Edinburgh, June 1976.