BDM

AUTOMATED PLANNING WITH SPECIAL RELEVANCE TO ASSOCIATE SYSTEMS TECHNOLOGY AND MISSION PLANNING FINAL REPORT

BDM/VSQ-91-0742-TR

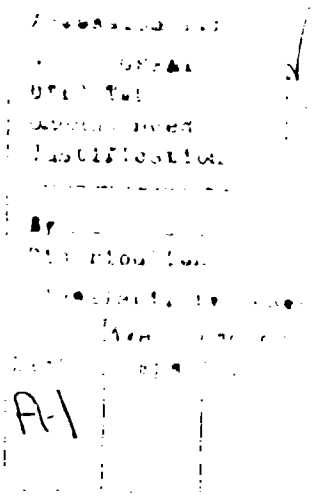
September 1991

BDM INTERNATIONAL, INC.
4001 NORTH FAIRFAX DRIVE
SUITE 750
ARLINGTON, VIRGINIA 22203
(703) 351-6900

AUTOMATED PLANNING WITH SPECIAL RELEVANCE TO ASSOCIATE SYSTEMS TECHNOLOGY AND MISSION PLANNING FINAL REPORT

BDM/VSQ-91-0742-TR

September 1991



ARPA ORDER NO. 6707/4
CONTRACT NO. MDA972-90-C-0039

Sponsored by
Defense Advanced Research Projects Agency
Advanced Systems Technology Office
3701 N. Fairfax Dr.
Arlington, VA 22203

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1991	FINAL 5/90 - 9/91

4. TITLE AND SUBTITLE
Automated Planning with Special Relevance to Associate Systems Technology and Mission Planning. (Automated Planning).

5. FUNDING NUMBERS
ARPA

6. AUTHOR(S)
Mr. Philip A. Merkel (BDM) Dr. John B.Gilmer (BDM)
Dr. Paul E. Lehner (GMU)
Dr. Roger A. Geesey (BDM)

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
BDM International, Inc.
4001 N. Fairfax Dr., Suite #750
Arlington, VA 22203

8. PERFORMING ORGANIZATION REPORT NUMBER
BDM/VSQ-91-0742-TR

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
Advanced Systems Technology Office
3701 N. Fairfax Drive
Arlington, VA 22203

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
CONTRACT NO. MDA072-90-C-0039, Task 2 Final Report

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report describes the results of work performed to establish the technical basis for a future R&D effort in mission planning technology. One known operational need is for aircrew decision aiding for dynamic replanning during the performance of modern military air missions where information flow, knowledge and detailed reasoning can radically alter the outcome. Artificial Intelligence technologies have been applied to create associate concepts with automated planning being a key construct to aid decision making in time constrained, rapidly changing situations. Basic and advanced research programs such as Pilot's Associate, Rotorcraft Pilot's Associate and the Submarine Operational Automation System have each introduced relevant advances to meet service-specific needs. This report provides an overview of automated planning techniques followed by an applications and technology map. The material is supported by information on the relationships of automated planning to the techniques of mathematical optimization, decision theory, et. al. In conclusion, the general assessment of automated planning technologies is offered followed by recommendations for directions of new research and associate systems development.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Intelligent Systems, Decision Making, Automation Real-Time, Artificial Intelligence, Planning, Situation Assessment, Applications, Knowledge	113
	16. PRICE CODE

17. SECURITY CLASSIFICATION 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 reports. 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, and 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 number(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 co nt 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 report.

Block 9. Sponsoring/Monitoring Agency Name(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 contains 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.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES

FOREWORD

With the advent of programs such as Pilot's Associate, Rotorcraft Pilot's Associate, Submarine Operational Automation System, and various others there has emerged an increasing interest in associate systems technology - the develop of systems that provide real-time support for planning and decision making in rapidly evolving situations. The objective of this report is to explore the relevance of automated planning in Artificial Intelligence (AI) to associate systems technology. Specifically this report achieves four objectives. First, it provides a general overview of automated planning techniques. Although the automated planning literature is extensive it lacks a good introduction. Consequently, we have prepared this report so that it may serve as a general introduction. Second, for each group of automated planning techniques the potential for associate systems technology applications is explored. Third, we explore the relationship between automated planning and other technologies (viz., mathematical optimization, decision theory, hardware engineering) with respect to their potential relevance to associate systems technology. Finally, we merge the previous discussions into a general assessment of automated planning and recommend directions for future research in automated planning that would directly contribute to better associate systems technology systems.

This report contains three parts. Part I (Chapters 1 - 6) examines alternative paradigms for automated planning and the relevance of each paradigm to associate systems technology. Part II (Chapters 7 - 9) examines the relationship between AI automated planning techniques and related techniques in other disciplines, specifically Operations Research, Decision Theory and Hardware Engineering. As indicated there, we feel that associate systems could benefit considerably from an effective merging of these disciplines. Finally, Chapter 10 presents our recommendation for future technology investments that are relevant to associate systems technology.

BDM would like to acknowledge the dedication of Dr. Paul E. Lehner of George Mason University who served as a consultant for this project and provided material for this report.

INTRODUCTION

This report describes the work performed by BDM and its technical team in establishing the technical basis for a future research and development program in the area of mission planning technology. The perceived operational need is that of dynamic repianning -- planning performed during the execution of air missions in response to changes in the objectives and constraints surrounding the original planning process. This work was performed for DARPA/ASTO under the Pilot's Associate System Engineering and Technical Analysis project. It was the intention of DARPA that this work complement the work done by SRS Technologies under a similar task which focused on the assessment of current tactical and strategic mission planning capabilities and the projection of future operational needs. The BDM project serves to characterize the technical nature of the dynamic mission planning problem and to suggest technical approaches which might warrant future investment.

The process used by BDM was to first start with future operational scenarios. Both tactical and strategic scenarios were postulated and analyzed. The strategic mission scenario included the full range of mission functions that could be envisioned for the future, including provisions for search and reconnaissance. The strategic mission planning problem was then characterized in a general fashion. With that generalization, a generic planning problem was defined which included all of the mathematical complexity of the original but cast the problem in a non-military light. This part of the work was performed by ORCA, a California-based small business. The problem which was found to represent the military mission planning problem was one centered on a Old West Marketeer, a complex extension of the classic travelling salesman problem.

The tactical scenario was generated by the operations research department of BDM. Based on emerging world tensions, BDM postulated an invasion of Kuwait and Saudi Arabia by Iraq. Forces based in Saudi Arabia took on the familiar offensive counter air, defensive counter air, close air support, and battlefield interdiction roles. In order to analyze these missions from a technology perspective, a series in relationship graphs were generated. These graphs depicted the objective of the mission at the center and arrayed all of the contributing factors around with arrows indicating that, for instance, avoiding detection is a function of range speed, radar effectiveness, countermeasure effectiveness, and RCS (to name a few).

This work led to description of the mission planning problem in the form of constraint satisfaction. A constraint representation syntax was developed in first order predicate calculus which would be useful at some future time in order for the actual computation of flight paths for aircraft, formations, and mission packages as part of theater scale air operation. The work in constraint representation language for mission planning led into the techniques for solution of such constraint reasoning problems. The majority of this report is devoted to the description and assessment of those techniques in the context of planning within an associate system.

The work led a natural distinction between the optimization approaches to problem solving and the constraint-based (AI) approaches to problem solving. The investigators for this project absolutely satisfied themselves that both approaches are necessary, and that complex problems will not be solved satisfactorily by either method alone. An optimization problem does not scale linearly with size. The scale factor is closer to a power of 7. A pure constraint-based solution may not provide adequate assurance of optimality. Clearly, the problems represented by dynamic mission replanning quickly become combinatorially explosive. A different method must be applied to avoid the combinatorics trap. This implies a form of constraint-based preprocessing to limit the actual problem to size that lends itself to initial and subsequent iterative solution by a variety of optimization techniques. A future R&D program which addresses dynamic mission replanning should include investigation into combinations of solution techniques integrated into a single solution environment.

In addition to AI based automated planning techniques presented in Chapters 1-6, there are a variety of technologies that are relevant to automated planning and specifically relevant to mission planning and associate systems. In Chapters 7-9 we examine three such areas: mathematical optimization, decision theory, and hardware.

Mathematical optimization (also called mathematical programming) is an area of *Operations Research* oriented toward the development of automated procedures that find optimal solutions to a variety of problems. In the area of automated planning, mathematical optimization represents a paradigm that competes with the AI paradigms. In the AI paradigm, automated planning problems are viewed as finding a satisficing solution that is consistent with a set of symbolic constraints. In the OR paradigm, planning is viewed as finding a solution that scores high (perhaps optimally) on an objective function (measure of merit) while staying consistent with a set of mathematical constraints (a set of equations and

inequalities). In Chapter 7 we will examine the merits and benefits of each approach, and discuss an appropriate melding of these two perspectives.

Decision Theory is an area of research devoted to the development of normative theories of inference and decision making. In the decision theory perspective planning is viewed simply as a problem of finding actions that maximize expected utility. Like mathematical optimization, decision theory provides a paradigm for automated planning that competes with the constraint satisfaction/satisficing approach of the AI paradigms. In the last few years, decision theory has had a significant resurgence within the AI community. This is evidenced by the most recent AAAI (1991) conference, where a substantial portion of the papers presented decision theoretic approaches. In Chapter 8 we examine the decision theoretic paradigm as well as the variety of ways it is currently being applied to automated planning problems.

Finally, Chapter 9 examines current developments in hardware technology and its possible application to automated planning. Unlike some other areas in AI (e.g., image understanding) very little has been done to develop hardware capabilities that are uniquely tailored to automated planning. The potential for such developments is explored in Chapter 9.

The complete set of recommendations resulting from this project are described in Chapter 10.

CHAPTER I

PLANNING AS SEARCH

1.0 Introduction

Within the AI community, there exist several competing paradigms for automated planning. Each paradigm provides a general perspective on the following questions. What is a plan? How should plan-relevant knowledge be represented? How should plans be generated and modified?

Our introduction to automated planning is organized around these paradigms. Specifically, we review the following approaches:

Planning as Heuristic Search - In this paradigm, a plan is defined as a sequence of actions that results in a sequence of states that ends with a state satisfying a goal condition. Planning problems are characterized using a state-space representation. Actions are defined as functions that map one state into another. Automated planning is treated as a problem of searching through the state-space. This is achieved by applying general purpose heuristic search procedures. Applications of this approach are found in the path planning algorithms used in Pilot's Associate.

Planning as Subgoal Directed Search - This approach is often referred to as classical planning. Planning is still viewed as a problem in state-space search. However, search through the state-space is goal-directed. Beginning with the goal description, subgoals are defined and refined until specific actions can be found to achieve those subgoals. By achieving a sequence of subgoals, the final goal condition is reached.

Planning as Constraint Posting - The distinguishing feature of this approach is the recognition that a plan does not need to be a fully detailed plan of action. Often it is sufficient to identify the principal actions that must occur and to put some constraints on when and how those actions will occur. Any specific set of behaviors that conforms to these constraints should result in achieving the goal. Planning is the process of identifying the relevant set of constraints.

Although most work in the constraint posting approach assumes a state-space representation, this representation is not essential to the paradigm. Consequently, it is more general that the previous two paradigms. A variant of this approach is Pilot's Associate to generate tactical advice for the pilots.

Planning as Plan Transformation - This paradigm is founded on the recognition that an effective planner does not enter a planning problem *tabula rasa*. The planner will usually have a store of template plans or historical cases from which the planner can quickly retrieve plans that have worked in similar situations. Planning is a two step process: retrieve a plan that is relevant to the current problem, and modify that plan until a satisfactory plan is reached. Automated planning procedures in the Submarine Operational Automation System is based on this approach.

Planning from First-Principles - This paradigm is founded on the belief that a truly general planning system should allow one to declaratively characterize a problem domain and then let a general problem solving mechanism generate plans. Usually First-Order Predicate Logic or some extension is proposed as the language for describing the problem domain. Once the problem domain has been characterized as a set of declarative statements in a logic all reasoning can be achieved using general theorem proving techniques. This includes reasoning necessary to generate plans. As a result, it is not necessary to implement a specialized problem solving mechanism for planning.

Planning and Reacting - This is not a paradigm, but an emerging area of research. Until recently real-time planning has not received a lot of attention in the AI research community. The focus of the planning community was on the abstract plan generation process, and not on the problem of controlling the behavior of robotic agents. Consequently, problems related to plan execution monitoring, plan repair, and real time behavior control were generally ignored. However, recent DARPA programs such as the Autonomous Land Vehicle and Pilot's Associate have focused attention on the problem of real-time planning and control. In Chapter 6 we review several paradigms that have been proposed for addressing this problem.

For each paradigm, we examine its relevance to associate systems technology from three perspectives: plan generation, real time planning, and plan modification. For plan generation we examine the range of

problems for which the paradigm is appropriate. For real time planning, we will examine the extent to which each paradigm can support both rapid planning and anytime problem solving An anytime problem solver (Dean and Boddy, 1988) can be interrupted at any time during problem solving with a request for the current best solution. For plan modification we are interested in the extent to which the paradigm can correctly and rapidly adapt a plan to an evolving situation. The planner should be able to minimize the extent to which the plan is modified.

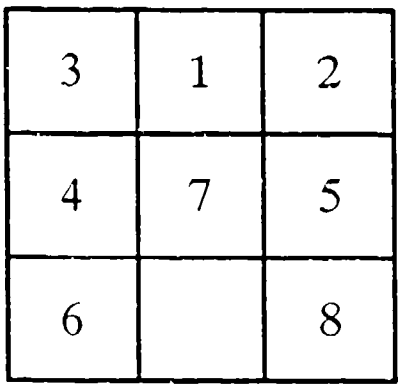
1.1 Planning as Heuristic Search

The planning as search paradigm treats automated planning as a problem in state-space search. To apply this paradigm one must (a) represent the planning problem as a state-space graph, and (b) apply a heuristic search technique to find a solution path in the state-space graph. The solution path is the plan generated. The elements of this approach are described below.

1.1.1 State-Space Graphs

A common way to view planning problems is to decompose the problem domain into a set of possible states. Consider, for instance, the eight puzzle problem shown in Figure 1-1. The objective is to move tiles (numbered 1 - 8) from their initial location to their final location. Each possible configuration of tiles represents a possible state. We can move from one state to another by (implicitly) moving the blank square up, down, left or right. Each of these four possibilities represents a state-change operation. Starting with the initial position, one can depict all possible sequences of state-change operations as a state-space graph. A portion of the state-space graph for Figure 1-1 is shown in Figure 1-2. The objective of state-space search is to find a path from the initial state to a goal state. For the problem in Figure 1-1, one such path is shown in Figure 1-2.

An important property of state-space representations is that of path independence. In going from one state to another, say Si to Sj, the contents of Sj should not depend on the path taken to get from Si to Sj. This property is needed in order to guarantee that the effect of any state-change operation applied to any state Sj can be computed from just the contents of Si and nothing else. All state-space search procedures assume that this property is satisfied.



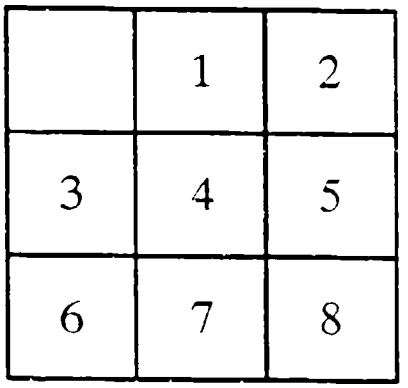


Figure 1-1: Eight Tile Puzzle Problem

As another example, consider the simplified mission planning problem depicted in Figure 1-3. The aircraft at position A must maneuver to position B and then return to A. The darkened areas represent regions of high lethality. A common way to model this problem is to overlay a grid on the map, and to treat aircraft movements as movements from one grid location to an adjacent one. Each grid to grid movement identifies a state-change operation. However, the relevant features of aircraft's status is determined by more than just its current location. Also important are features such as current fuel level, current lethality, total lethality, etc. Consequently, a state-space representation of this problem describes a state in terms of a state vector (e.g., <location, altitude, speed, fuel, current lethality, total lethality, ...>) and state-change operations as functions that calculate a new vector of values from the values in the previous state.

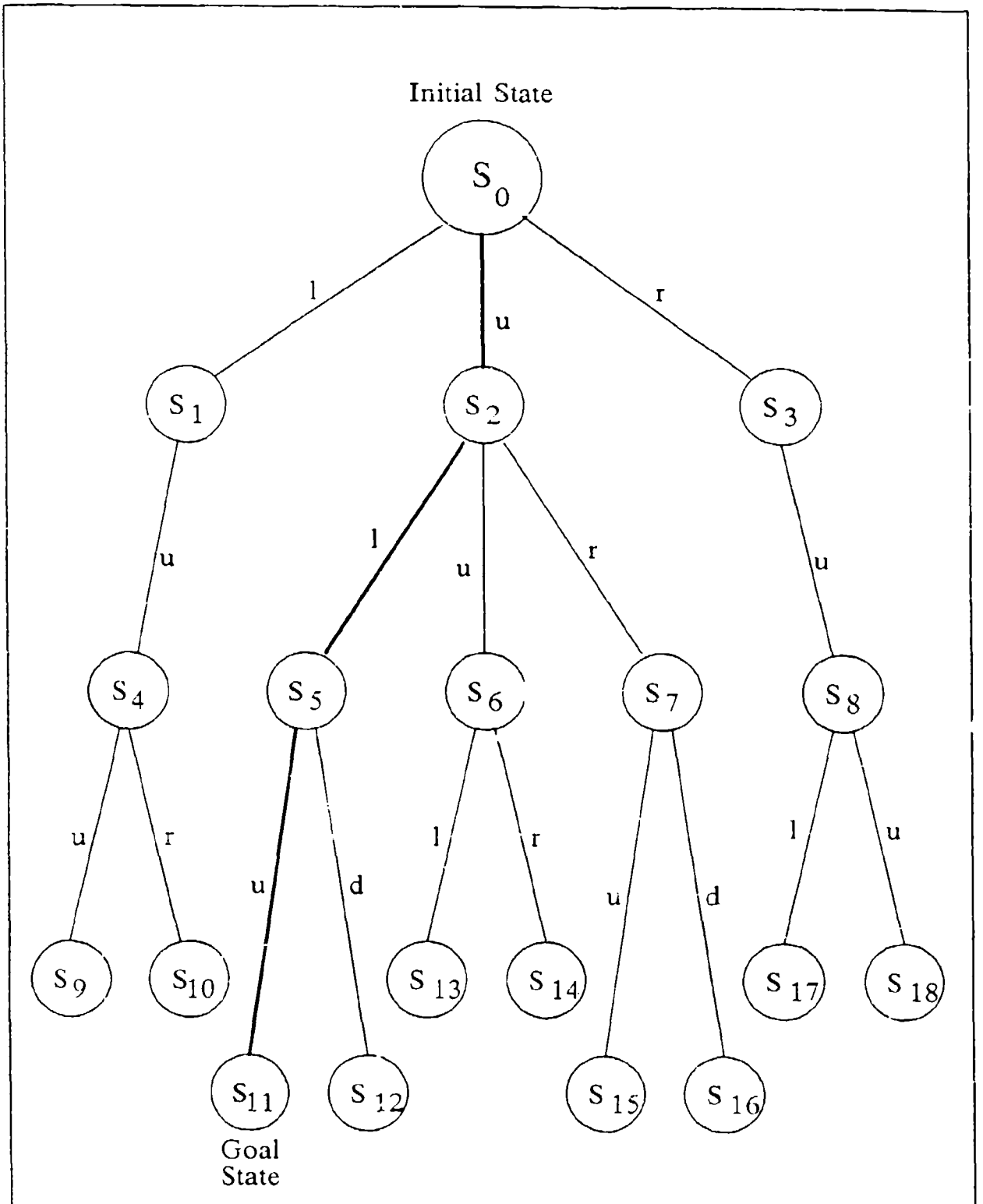


Figure 1-2: Portion of State Space Graph for Eight Puzzle Problem

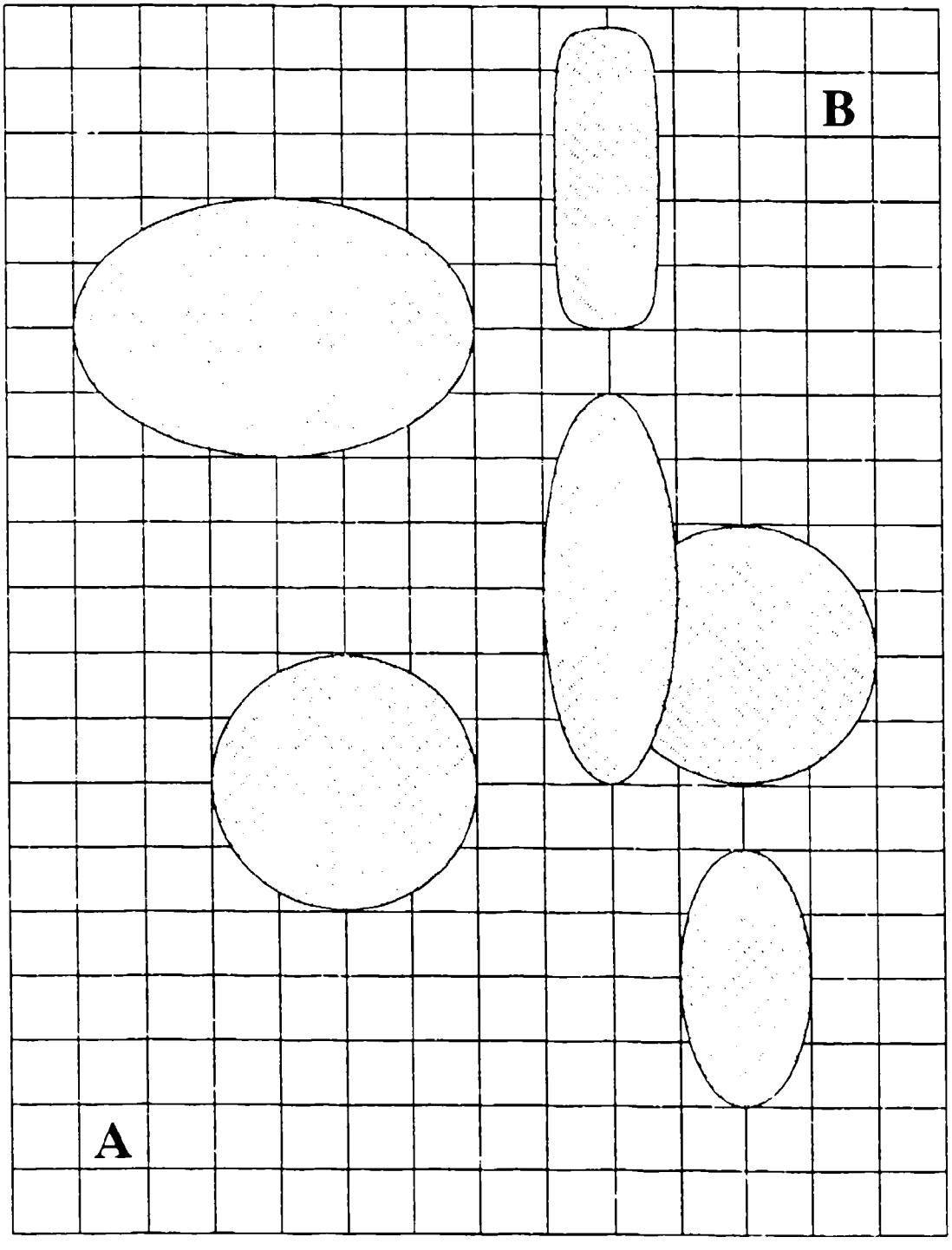


Figure 1-3: Simple Mission Planning Problem

1.1.2 Searching State-Space Graphs

Once a planning problem has been translated into a state-space representation, standard heuristic search techniques can be used to find a path from the initial state to a goal state. Consequently, a special mechanism for plan generation is not required.

Heuristic search techniques can be loosely separated into three categories: undirected, locally-directed, globally-directed. Each of these techniques will be illustrated using the example graph shown in Figure 1-4. In reviewing the search techniques below, keep in mind that

1. each node represents a state,

2. arcs correspond to state transition operations (i.e., actions), and

3. for any node, its subnodes represent the set of possible next states.

In Figure 1-4, for instance, node 1 represents the initial state, and nodes 10 and 16 represent two goal states (i.e., two states that satisfy the goal criteria). The sequence 1-->4-->10 defines a plan of action involving two actions. The first action changes the initial state into state 4 and the second action changes state 4 into state 10 -- a goal state.

1.1.2.1 Undirected Search

Undirected search techniques simply expand nodes in a graph according to a predefined pattern, irrespective of the contents of nodes currently open.[1]

Depth-first search is an example. Whenever a node is visited[2] and expanded, then the first subnode generated is immediately visited until a stopping criteria (e.g., depth limit) is reached. When a subnode, sn, fails (i.e., stopping criteria is reached or all subnodes of Sn have failed) the next subnode (i.e., a sibling of Sn) is visited. This continues until a goal node is

---

[1] A node is expanded if the subnodes of that node have been generated. An open node is a node that has been generated, but not expanded.

[2] A node being visited is processed as follows: (1) check to see if it is a goal node, (2) check to see if the stopping criteria has been reached, and if not (3) generate subnodes.

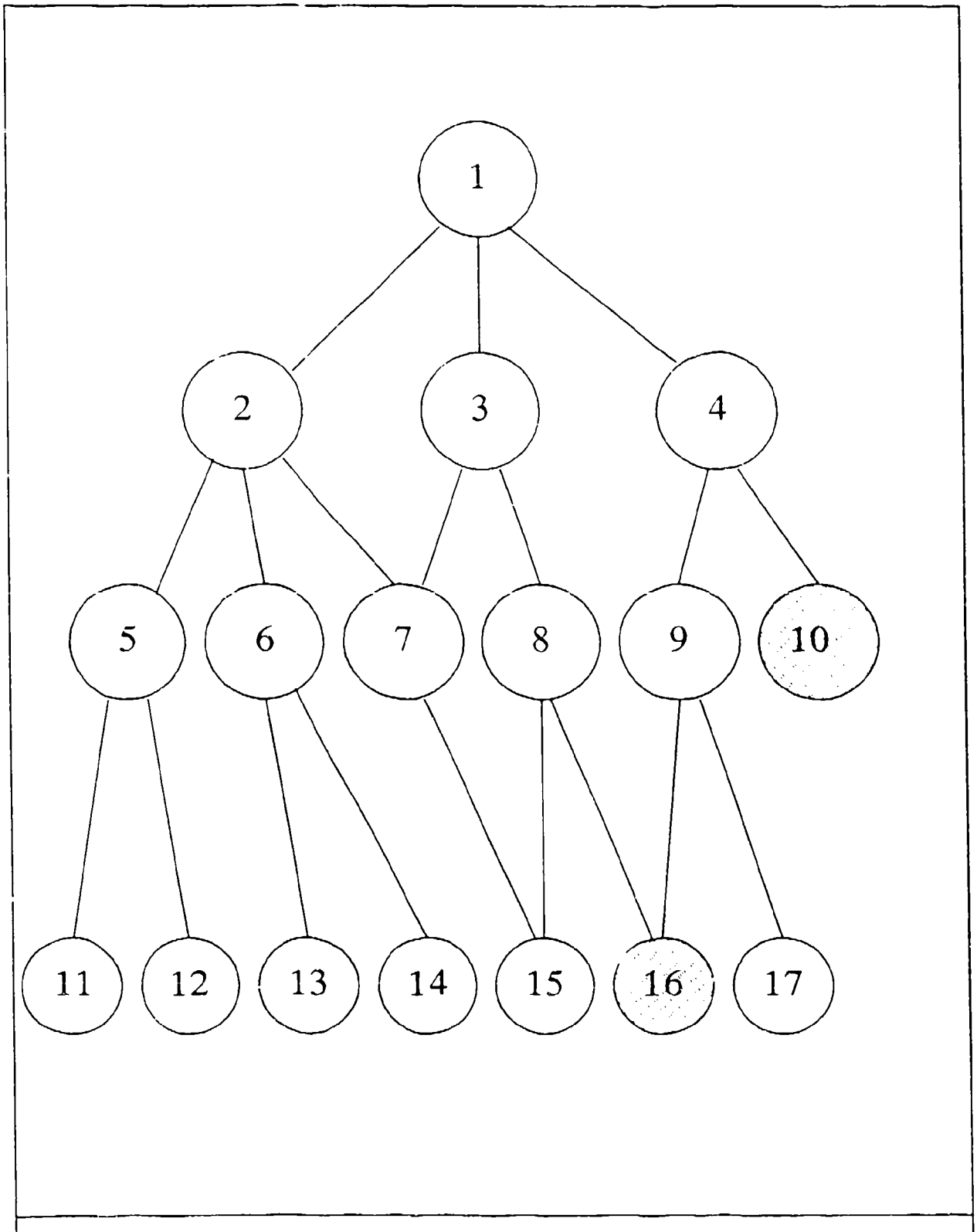


Figure 1-4: Typical State Space Graph

reached or search terminates without a solution. In the case of Figure 1-4 the sequence of nodes visited would be

1, 2, 5, 11, 12, 6, 13, 14, 7, 15, 3, 7, 15, 8, 15, 16.

The solution path is 1-->3-->8-->16.

In breadth-first all nodes generated at one level are visited before any node at a deeper level is visited. In Figure 1-4, for instance, the sequence of nodes visited by breadth-first search is

1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

and the solution path is 1-->4-->10.

Both depth-first and breadth-first search have severe weaknesses. Depth-first search tends to visit far more nodes that is required, and usually returns an unnecessarily long solution path. Breadth-first, on the other hand, is computationally intractable since it requires that all nodes generated must be maintained in memory.

A recent development in blind search techniques (Korf, 1985) is depth-first iterative deepening (DFID). DFID iteratively applies depth-first at increasing depth levels. All nodes to level one are searched, then all nodes to level two are searched, and all nodes to level 3, etc. At each iteration, search begins anew. All nodes generated during previous iteration are generated again. For instance, in the case of Figure 1-4, DFID would visit nodes in the following order:

Level 1: 1, 2, 3, 4
Level 2: 1, 2, 5, 6, 7, 3, 7, 8, 4, 9, 10,

returning the path 1-->4-->10.

Despite the fact that DFID seems wasteful this extra cost of repeatedly revisiting higher level nodes does not increase the order of magnitude of the search. Consequently, with respect to all possible blind search techniques, DFID search is optimal with respect to memory usage (number of nodes maintained in memory) and length of the solution path, and is asymptotically optimal on the order of magnitude of the number of nodes visited. In short, DFID is optimal or nearly optimal on all major criteria for evaluating blind search techniques.

1.1.2.2 Locally-directed Search

Undirected search techniques do not presume any ordering of sibling nodes. For instance, for the sibling nodes 2-4, blind search techniques do not require that these nodes be visited in any particular order. Often, however, it is possible to order sibling nodes according to the relative probability that each sibling is part of a successful path. Although ordering nodes in this way would improve the efficiency of undirected search, these techniques are not designed to exploit this extra information. Locally-directed search techniques are designed to exploit sibling order information.

A recent example of a locally-directed search technique (Ginsburg, 1990) is Iterative Broadening (IB). IB proceeds by iteratively performing depth first search, where on each iteration the number of subnodes visited is expanded. On the first iteration, only the first subnode of each node is visited. On the second iteration, the first and second subnode is visited. On the third iteration, the first three subnodes are visited. And so on. In Figure 1-4, for instance, the sequence of subnodes visited is:

Level 1: 1, 2, 5, 11
Level 2: 1, 2, 5, 11, 12, 6, 13, 14, 3, 7, 15, 8, 15, 16,

returning the solution path 1-->3-->8-->16.

The performance of IB depends heavily on the quality of the ordering information and the density of goal nodes in the state-space graph. For many problems, however, it represents a significant improvement over blind search.

1.1.2.3 Globally-directed Search

Locally-directed search exploits order information on sibling nodes to guide search. Globally-directed search, on the other hand, uses ordering information that applies to all open nodes. This is achieved by applying a function (f') to each open node and then always expanding the open node with the lowest f'-value. As long as the number of nodes with an f'-value less than a goal node is finite, this procedure will eventually find a path to the lowest f'-valued goal node.

Obviously the efficiency of globally-directed search depends on the f'-values assigned to each node. A common approach is to assign a cost value to each arc and then to define f' is as follows:

g(n) = cost of cheapest path from the start node to node n,

g*(n) = estimate of cost of cheapest path from start to node n
= estimate of g(n),

h(n) = cost of cheapest path from node n to goal node,

h*(n) = estimate of h(n),

f*(n) = g*(n) + h*(n),

f'(n) = f'(ni) if ni is a parent of n and f'(ni) > f*(n),
f*(n) otherwise.

In words the f'-value for any node is the cost to get to that node plus an estimate to complete, with the added requirement that the f' value of a node must always be greater than or equal to the f'-value of the parent. If the estimate-to-complete (h*) never overestimates the true cost to complete, then globally-directed search will always find the lowest cost path from the start node to the goal node.

When f' is defined as above, the search procedure is call the A algorithm. If the h* metric is always an underestimate, then the search procedure is called A* (Pearl, 1982).

To relate this to the mission planning problem, consider Figure 1-5. Figure 1-5 is the same as Figure 1-4 except that the arc cost and the h* values are shown (h* values are in brackets next to each node). Consider each node to be a map location (e.g., way-point) where the objective is to go from the initial locations (start node) to a goal location (e.g., one of two alternative targets) while minimizing lethality. Along each arc is an estimate of the lethality for that section of the route. A globally-directed search procedure would visit the nodes in the following order,

1, 2, 3, 8, 4, 9, 16

and would return the path 1-->4-->9-->16.

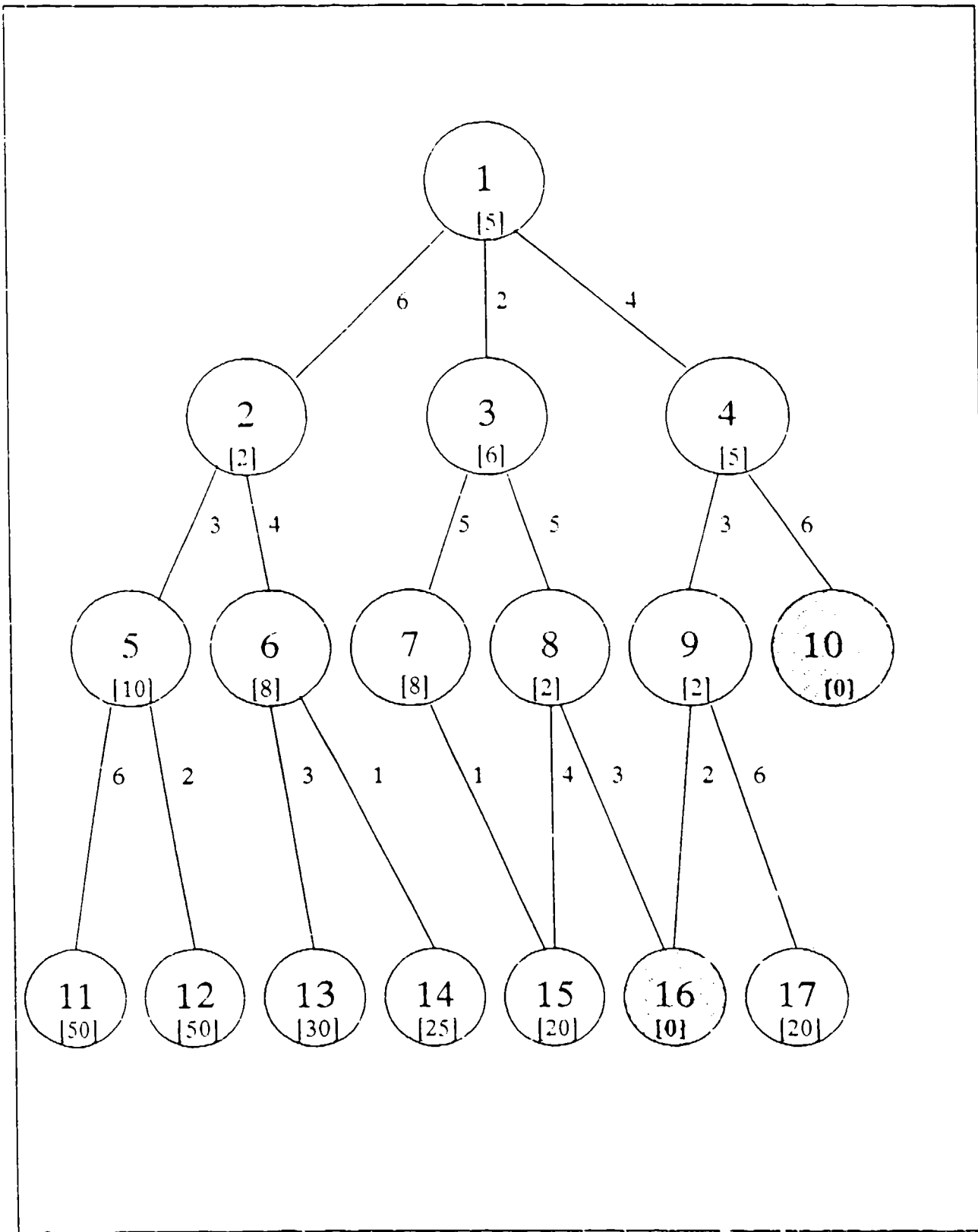


Figure 1-5: A State Space Graph - Weighted

One limitation of globally-directed search procedures is that they cannot be applied to large search problems. A globally-directed procedure must maintain a list of all paths generated during search. Consequently, every node generated must be maintained. Even for toy problems, like the 8-puzzle, thousands of nodes may be generated. For large problems, the number of nodes generated will usually exceed the memory limitation of the host hardware.

For large problems, an alternative procedure is available -- iterative deepening A* (Korf, 1985). Iterative Deepening A* (IDA*) works like iterative deepening, but instead of performing a depth-first search at increasing depths, IDA* performs a depth-first search at increasing f' levels. In Figure 1-5, for instance, IDA* would visit the nodes as follows:

Level	Nodes Searched	max f'
1	1	5
2	1,2,3	8
3	1,2,3,8,4,9.16	9

Note that the max f' value for the next level is set to the minimum f' value of any node that was generated but not expanded. IDA* is not really a globally directed procedure since it does a blind depth-first search within each f' level. However, since it uses f' to select levels it will always find the lowest f'-valued path. Furthermore, because it uses blind depth first search, it only needs to maintain the current path in memory. Consequently, it is suitable for large problems. On the other hand, the number of nodes searched may increase, by an order of magnitude over A*. This is because it is easy to construct pathological state-spaces where each increase in the max f' value adds only one node to the search. However, such state-spaces may be pathological. Consequently, in practice IDA* may often be the preferred procedure for large search problems.

1.1.2.4 Other Search Procedures

As a group, the search procedures described above are often referred to as heuristic search procedures. Heuristic search procedures are distinguished by the fact that they are neither goal driven or knowledge intensive. In a goal dri en search, the selection of nodes to expand is determined by identifying subgoals and finding state change operations that move toward achieving those subgoals. Goal driven search is generally knowledge intensive. It takes a substantial amount of domain knowledge to select appropriate subgoals and control search.

As a search procedure subgoaling has some advantages. It makes it possible to decompose a problem into a set of independent or sequential subproblems. Breaking a problem into subproblems will substantially decrease the amount of search required. It also makes it possible to utilize dependency-directed backtracking. When a search path results in a dead end, and backtracking is required, it may be possible to use the subgoal information to determine how far to backtrack before continuing the search. There is however no guarantee that this procedure won't skip over a desired solution path.

As will be seen in the next three chapters, most AI planning systems combine a heuristic search procedure with some form of goal driven search.

Also, it should be noted that there are a variety of search procedures that exploit probabilistic knowledge as to which nodes will lead to a goal state (e.g., Pearl, 1984). In mission planning/associate system problems, this knowledge is not generally available. Consequently, these procedures are not reviewed here.

1.2 Relevance to Associate Systems Technology

1.2.1 Plan Generation

A mission planning problem can be viewed as a problem in state-space search. To do this one must specify a vector of variables (location, altitude, fuel level, threat exposure...) that identifies all factors relevant to calculating a route from the aircraft's current location. This state-space would be very large, and it is unlikely that an undirected or locally-directed search procedure would be adequate. A globally directed search procedure, such as A*, may be feasible if it is based on a good estimation function. Recall that with a globally directed search procedure the number of nodes searched and amount of memory required increases substantially as the accuracy of the heuristic estimate decreases. A poor heuristic estimation function would result in far too many open nodes for efficient processing.

If one is willing to give up on finding the optimal solution (e.g., lowest cost path), it is possible to substantially increase the efficiency of search. For instance, the processing time of the A* algorithm can sometimes be reduced dramatically by allowing the h* function to be an

over estimate (Freeman, 1991). (For example, let $f^*(n) = g^*(n) + w(h^*(n))$ where w is a multiplier greater than 1.) Alternatively, one can decompose the problem into a set of smaller subproblems (e.g. pre select several way points) and find optimal solutions to each subproblem. As Korf (1988) noted, this can lead to an order of magnitude reduction in search time.

In mission planning applications, we have noted two basic approaches to simplifying the complexity of the search space.

Reduce number of state-space variables - Efficiency can be improved, simply by ignoring some of the variables in the state-space representation. By defining a few aggregate variables, and ignoring others, an efficient globally directed search procedure can be implemented. An example of this is found in Lockheed's path planning system used in the Pilot's Associate. In this system, the state-space is defined in terms of just a few simple variables (fuel level and visibility).

Another example is the path planning system used to generate the mission plans for the F-117A (Mitchell, 1991). Here, once again, a globally optimal search procedure was applied to a subset of the relevant plan variables.

In principal, this approach can sometimes generate poor paths that a more inclusive state-space representation would not. However, careful engineering of the system usually minimizes the frequency of such events. In addition, in some of these systems the human operator has the opportunity to review and modify these plans.

Aggregate state-space representation - Rather than simply ignoring some variables, it is sometimes possible to preprocess the state-space into a more aggregate representation that contains fewer states. One example of this is found in the visibility graph approach to path planning (Meng, et.al., 1991; Silbert, 1991). In this approach, a map is processed to identify a set of distinct "objects" that should be avoided. These objects are then modeled as polygons, where the edges of each polygon defines a path around the object. The vertices of each polygon now define a set of "way points", and the cost of moving from one way point to another line-of-sight way point is precomputed. Path planning is defined as a problem of finding a minimum cost path through a sequence of way points. If the number of domain objects is not excessive, then this representation significantly reduces the complexity of the search space.

1.2.2 Plan Modification

For heuristic search there is no mechanism for performing plan modification as opposed to replanning. Replanning simply requires invocation of the search procedures with the new state. Plan modification requires that the new plan be a minimal modification of the original plan. Within the state-space search paradigm there is no mechanism for implementing this "minimal modification" concept.

1.2.3 Real-time Planning

Heuristic search procedures are not designed for real time planning. Although these procedures may be very fast, they are not designed for anytime problem solving. This is particularly true of undirected and locally-directed search procedures, since they have no metric for assessing whether or not a partial path is promising. On the other hand globally directed search procedures are based on a metric for assessing expected cost of any partial path. It is possible to interrupt search and request information on the most promising partial path so far. An example of this is found in Korf's approach to real-time path planning (Korf, 1987). Korf's planner iterates through the following sequence

1. Execute a depth-limited ID * search from current state.
2. Select the path that minimizes total expected cost.
3. Execute the first action on the path selected.
4. Define the new state as the current state.
5. Go to 1.

Using this procedure, the agent's next actions continuously follow the most promising path.[3]

---

[3]Unfortunately, this approach is not always reliable. This is because the h* function (estimated distance to goal) may not be very sensitive to the true function (estimated distance to goal) many not be very sensitive to the true distance to the goal until search reaches a node that is very close to the goal. As Korf (1988) himself has noticed, in some domains the h* value is nearly constant for all nodes except those very close to the goal node. This suggests that during most of the search process good and bad paths are not distinguishable.

In the section on Planning and Uncertainty Management we will introduce a general mechanism for converting procedures such as A* into anytime problem solvers. As we show there it is possible to interrupt the search process and request not only the best path so far, but also a probability estimate that the current best path is a "good" path.

Finally, it should be noted that the visibility graph approach described above is amenable to real-time planning/replanning problems. By precomputing the shortest path between various regions, the planner can use these precomputed paths to quickly select alternative routes. A discussion of this approach can be found in Meng, et.al. (1991) and Silbert (1991).

CHAPTER 2

CLASSICAL PLANNING: USING SUBGOALS TO CONTROL SEARCH

Beginning in the late 1960's a series of planners where developed that have since come to be known as the classical planners.[1] The classical planning approach is similar to Planning as Heuristic Search paradigm with two significant differences:

1) They all use the same approach to defining actions and state-spaces, and

2) Search through the state-space is goal-driven.

The principal planners in this lineage are described below.

2.1 General Problem Solver (GPS)

One of the earliest automated planners was a system called the General Problem Solver (GPS). Although GPS is not generally considered to be a classical planner, it is widely regarded as the precursor to this line of planners. GPS used a search procedure call means-ends analysis (Newell and Simon, 1963). As applied to planning problems, means-ends analysis begins with a description of the initial state, a desired goal state, and a set of state-change operators. The principle operation of GPS is to iterate through the following steps. Beginning with the initial state, GPS:

1. Compares the present state with the goal state to generate a difference list.

2. IF the difference list is empty,
   THEN exit with plan.

3. Select a state-change operator that has a consequence the removal of the first difference on the difference lists,

[1] There is no clear definition of a classical planner. Although the planners described in this section are usually considered classical, some would also consider the planners described in Chapter 3 as also belonging to the classical planning tradition.

4. IF no operator is found,
   THEN remove last operator added to plan,
   reset the difference list, and
   go to 3 (to select a new operator).

5. IF the operator can be executed,
   THEN add it to the plan,
   update the state description, and
   go to 2.

6. IF the conditions necessary to execute the operator are not contained in the current state,
   THEN add these conditions to the difference list and
   go to 2.

Whenever step 2 finds no differences, GPS terminates and returns the sequence of state-change operators it applied. This sequence is the plan. When step 3 fails to find an operator, then GPS will backtrack on the sequence of operators to an earlier state, find a new operator to apply, and continue processing. In short, GPS is a depth-first state space search procedure where sibling nodes are ordered according to whether or not they remove an element from the difference list.

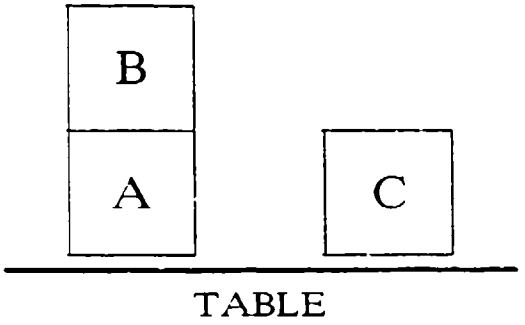
2.2 Stanford Research Institute Problem Solver (STRIPS)

STRIPS is generally regarded as the first of the classical planners. STRIPS was based on GPS, but some additional assumptions were made about how states and state-change operators are represented (Fikes and Nilsson, 1971). In STRIPS states are described as a list of propositions which are sentences of the form BLOCK(A), ON(A,B), CLEAR(C), etc. (See Figure 2-1). STRIPS state-change operators define different classes of actions. Each operator is composed of:

1. A precondition list that lists all propositions that must be contained in the state description before the operator can be executed,

2. an add list that lists the propositions that are added to the state description when the operator is applied, and

3. a delete list that lists the propositions to be deleted from the state description when the operator is applied.



ON(C,TABLE)
ON(A, TABLE)
ON(B,A)
CLEAR(C)
CLEAR(B)
CLEAR(TABLE)

STATE

STATE DESCRIPTION

Figure 2-1: Example of State Description

Figure 2-2 shows some examples of state-change operators.

In STRIPS a planning problem is defined by a state description of the initial state, a list of propositions that must be true of the goal state, and a list of STRIPS operators. (Planning problems defined in this way will be referred to as STRIPS problems.) STRIPS plans by searching for a sequence of state-change operations (actions) that will transform the initial state into a goal state (see Figure 2-3 for an example).

It is important to note that each new state in STRIPS is calculated by adding and deleting the propositions listed in the add and delete lists of the state change operator. Nothing else changes. In effect, STRIPS assumes that the only relevant consequences of an action are those that are always associated with that type of action. There are no side effects. This assumption is prevalent in automated planners and is called the STRIPS Assumption.

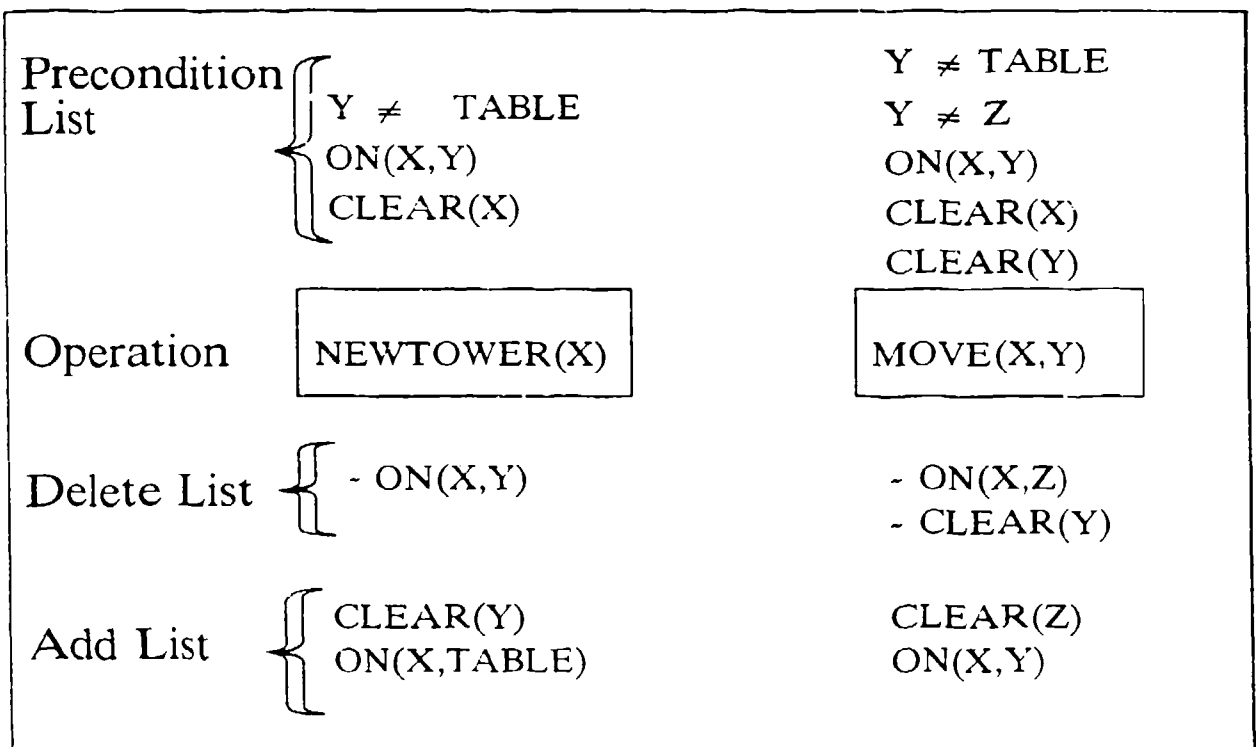


Figure 2-2: Example of STRIPS Operations

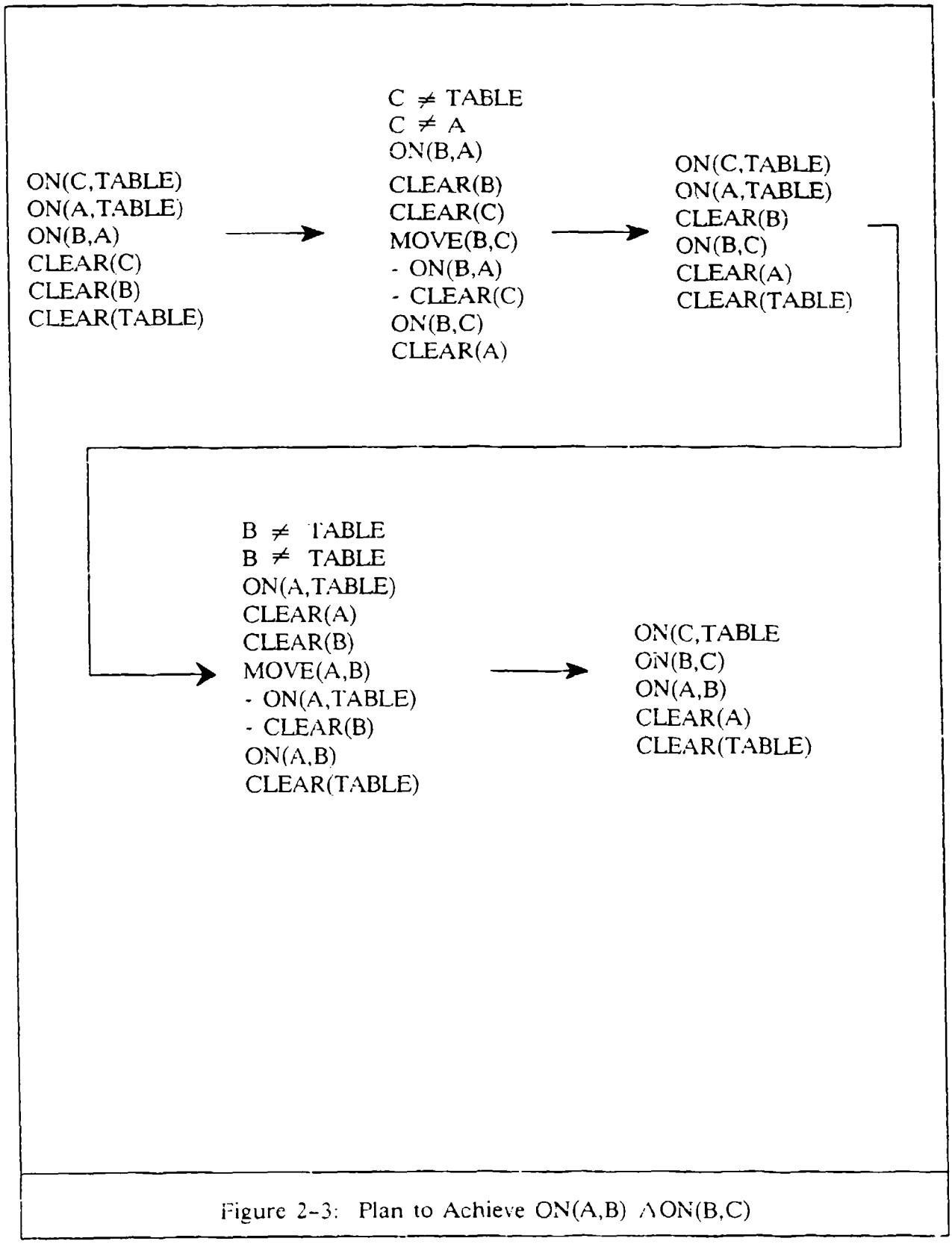


Figure 2-3: Plan to Achieve ON(A,B) ∧ON(B,C)

STRIPS generates plans by maintaining a stack of goals and actions. Let Stack be the stack maintained by STRIPS, G be the list of goals to achieve, and S the initial state.

1. Set Stack to G,
   Plan to nil.

2. IF Stack is empty,
   THEN exist with Plan
   ELSE select first element (e) of Stack.

3. IF e is an action,
   THEN mark current status of S, Stack and Plan as backtrack point,
   add e to Plan ,
   set S to update(e,S).

4. IF e is a goal and e is contained in S
   THEN pop e from Stack.

5. If e is a conjunction of goals and not contained in S
   THEN individually add each of the subgoals of e to Stack.

6. IF e is an individual goal and not contained in S
   THEN find an action A which has e as a consequence has has not been previously tried.

7. IF A is nil
   THEN backtrack and go to 2.

8. IF A is not nil,
   THEN add A to Stack,
   add as a conjunction of goals the pre-conditions of A to Stack,
   set A to nil,
   go to 2.

To illustrate STRIPS operations, consider again problem in Figure 2-1. The initial stack begins with just the initial goal (ON(A,B) & ON(B,C)) and via step 5 adds each part of the conjunction to the stack.

ON(A,B) & ON(B,C)
ON(A,B)
ON(B,C)

STRIPS checks to see if the bottom goal is true in the current state. It isn't. STRIPS then searches for an operator to instantiate that (after instantiation) has the bottom goal (ON(B,C)) on its add list. The instantiated operator and its preconditions are added to the stack.

ON(A,B) & ON(B,C)
ON(A,B)
ON(B,C)
MOVE(B,C)
ON(B,A) & CLEAR(B) & CLEAR(C)

The current state contains CLEAR(C), CLEAR(B) and ON(B,A), so the conjunction of these goals is popped from the stack. MOVE(B,C) is then popped and added to an action list and a new current state is computed. Since ON(B,C) is on the add list of MOVE(B,C), ON(B,C) will be true in the new current state, so it is popped. ON(A,B) is not satisfied, a new action is added to the stack. Here we choose MOVE(A,B). This results in the stack:

ON(A,B) & ON(B,C)
ON(A,B)
MOVE(A,B)
ON(A,TABLE) & CLEAR(A) & CLEAR(B)

CLEAR(B), CLEAR(A), ON(A,TABLE) and the conjunction of these three are all contained in the current state. Consequently, the conjunctive goal is popped from the stack. MOVE(A,B) is popped from the stack and added to the action list. ON(A,B) is then popped from the stack. Finally, ON(A,B) & ON(B,C) is checked. Since MOVE(A,B) does not have ON(B,C) on its delete list, the new current state will still contain ON(B,C). Consequently, ON(A,B) and ON(B,C) is satisfied. Since the stack is now empty, the goal has been achieved. The action list {MOVE(B,C), MOVE(A,B)} becomes the plan.

The reader may have noticed that in the above example there was usually more than one choice for an operator to add to the plan. Also the ordering of the ON(A,B) and ON(B,C) subgoals was fortuitous. If any of these had changed, the resulting plan would have included some unnecessary steps. In general, the quality of a plan generated by a STRIPS like system depends on a set of heuristics for ordering subgoals and selecting actions.

Although STRIPS works reasonably well, there are a some problems that STRIPS can not solve properly. The most famous of these is the Sussman Anomaly, shown in Figure 2-4. In this problem, the two goals ON(A,B) and ON(B,C) can not be serialized. A plan which solves these goals in sequence will inevitably include unnecessary actions. For instance, if the goals are ordered ON(B,C) and ON(A,B), then STRIPS will likely generate the plan

```
MOVE(B,C)
    -->NEWTOWER(B)
        -->NEWTOWER(C)
            -->MOVE(A,B)
                -->NEWTOWER(A)
                    -->MOVE(B,C)
                        -->MOVE(A,B).
```

Inspired by this and other problems, a series of AI planners were developed that are generally known as the classical planners. Virtually all of these planners used STRIPS-like state-change operators and made the STRIPS assumption.

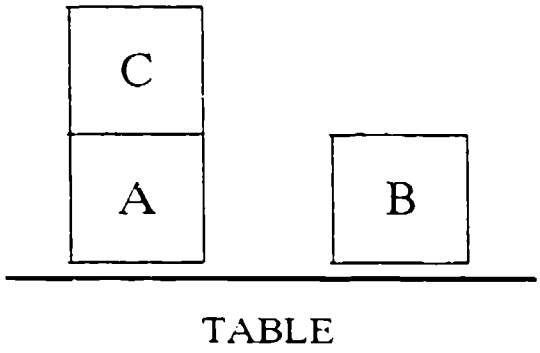


Figure 2-4: The Sussman Anomaly

2.3 Regression

Waldinger (1977) explored the technique of Regression. For any proposition P and action A, the regression of P over A (Regress[P,A)] specifies the conditions that must be true prior to executing A in order for P to be true after executing A. For instance,

Regress[ON(A,B),MOVE(A,B)] = Clear(A) & Clear(B) & ON(A,z)

which are just the preconditions of MOVE(A,B). Note that regression does not presume that P and A are fully specified. For instance,

Regress[Clear(x),MOVE(A,y)] = (Clear(x) & x= /y) or (ON(A,z) & x=z).

Using Regression, it is possible to specify how the goals in the Stack relate, thereby allowing the planner to determine where in the stack an action can be inserted. Consider how regression can be used to solve the Sussman anomaly. Beginning as in STRIPS, the planner generates the stack:

ON(A,B) & ON(B,C)	a		
ON(B,C)	a		
ON(A,B)	a		
MOVE(A,B)		b	
CLEAR(A)			b

CLEAR(B)		b	
ON(A,z)		b	

Unlike STRIPS, however, sets of goals and actions in the stack are labeled as belonging to various protected sets. A protected set specifies a set of goals that must all be true when the entire set is complete. For instance, CLEAR(A), CLEAR(B) and ON(A,z) are part of the protected set b. A protected set is determined by regression. For instance, set b is equal to Regress[ON(A,B),MOVE(A,B)].

The stack is processed by sequentially moving through the stack until an unachieved goal is found -- in this case CLEAR(A). An action and its precondition are inserted into the stack prior to the goal, as long as the inserted action does not undo any previously achieved goals in the protected set in which the goal has been inserted. Continuing with the example, the planner now inserts NEWTOWER(C). This results in the following stack:

```
ON(A,B) & ON(B,C)          a
ON(B,C)                    a
-------------------------
ON(A,B)                    a
MOVE(A,B)                      b
CLEAR(A)                           b
NEWTOWER(C)                        c
CLEAR(C)                                   c
ON(C,A)                            c
CLEAR(B)                       b
ON(A,T)                        b
```

Note that the insertion of NEWTOWER(C) only interrupted set b. Since

Regress(CLEAR(B),NEWTOWER(C)]=CLEAR(B), and

Regress[ON(A,T),NEWTOWER(C)]=ON(A,T),

CLEAR(B) and ON(A,T) will still be true after NEWTOWER(C).

Continuing this example further, suppose the planner now tries to insert MOVE(B,C) as shown below:

```
ON(A,B) & ON(B,C)      a
ON(B,C)                a
-------------------------
                           <---------- MOVE(B,C)
ON(A,B)                a                   CLEAR(C)
MOVE(A,B)                  b               CLEAR(B)
CLEAR(A)                       b           ON(B,T)
NEWTOWER(C)                    c
CLEAR(C)                               c
ON(C,A)                        c
CLEAR(B)                   b
ON(A,T)                    b.
```

Before it can insert this action, it must check to see if MOVE(B,C) impacts ON(A,B). However, Regress[MOVE(B,C),ON(A,B)] = NIL. There is no circumstance in which ON(A,B) will be true immediately after the action MOVE(B,C) is executed. Consequently, MOVE(B,C) can not be inserted into any stack where ON(A,B) is protected and not reestablished after the MOVE(B,C) action. Since MOVE(B,C) is the only action that has ON(B,C) as a consequence, the current stack can not be completed.

To repair this problem, the planner can now reorder the goals in the stack to avoid the problem that lead to the current dead end. Since the attempt to achieve ON(B,C) impacted the protected goal ON(A,B), the planners re-orders ON(B,C) and ON(A,B). Specifically, it reorders the stack so that ON(B,C) occurs prior to the action that achieves ON(A,B). This gives us the stack:

```
ON(A,B) & ON(B,C)      a
ON(A,B)                a
MOVE(A,B)                 b
ON(B,C)                a
---------------------------
CLEAR(A)                     b
NEWTOWER(C)                  c
CLEAR(C)                           c
ON(C,A)                            c
CLEAR(B)                  b
ON(A,T)                   b.
```

Now when we insert the action MOVE(B,C) we get the stack:

```
ON(A,B) & ON(B,C)      a
ON(A,B)                a
MOVE(A,B)                 b
ON(B,C)                a
MOVE(B,C)                          d
CLEAR(C)                                 d
CLEAR(B)                           d
ON(B,T)                            d
CLEAR(A)                     b
NEWTOWER(C)                  c
CLEAR(C)                           c
ON(C,A)                      c
CLEAR(B)                  b
ON(A,T)                   b,
```

where we can confirm through regression that none of the actions impact any of the previously achieved goals in the protected set. This gives us the final plan NEWTOWER(C)-->MOVE(B,C)-->MOVE(A,B).

2.4 Nonlinear Planning

Another enhancement to STRIPS that is that of nonlinear planning (Sacerdoti, 1977). Nonlinear planning is based on the idea that the actions in a plan do not need to be fully ordered. Furthermore, a planner should not impose an ordering on a set of actions unless it needs to. For instance, in solving the Sussman Anomaly, a nonlinear planner might proceed by initially splitting the two goals and finding plans to achieve each individually. This is shown below.

NEWTOWER(C) ---> MOVE(A,B)

MOVE(B,C).

Upon examining this plan, the planner discovers a problem. Namely that CLEAR(B), which is a precondition for MOVE(B,C), is on the Delete list of MOVE(A,B). (A mechanism for discovering this type of problem is discussed in the next section). Consequently, this plan will not work if MOVE(A,B) occurs before MOVE(B,C). To resolve this problem MOVE(B,C) must occur before MOVE(A,B), so the plan is constrained to satisfy this ordering:

NEWTOWER(C) ---> MOVE(A,B)

MOVE(B,C).

However, this is still not satisfactory in as much as CLEAR(C), a precondition of NEWTOWER(C), is on the delete list of MOVE(B,C). Consequently, NEWTOWER(C) must be constrained to occur before MOVE(B,C). This gives us:

NEWTOWER(C) MOVE(A,B) MOVE(B,C).

as the final plan.

2.5 Hierarchical Planning

Realistically, complex planning problems require that the planner separate significant planning factors from details. Initially, a partial plan is developed that accounts for the significant factors, after which details are worked out. For instance, in planning a cross country trip, one should

first identify the flight to take before worrying about the details of how to get to and from each airport.

In the AI automated planning literature each level of detail is referred to as a level of abstraction. Planners that operate by generating plans at decreasing levels of abstraction are called hierarchical planners (Sacerdoti, 1977). For instance, multiple levels of abstraction can be defined in a STRIPS problem by assigning priority levels to the preconditions of an operator. Hierarchical planning then proceeds by initially generating a plan considering only first priority preconditions, then inserting steps into the plan to account for second priority preconditions, again with third priority preconditions, and so on. At each level of planning the plan generated at the previous level serves as an outline to which additional actions are inserted.

Levels of abstraction can be defined in other ways as well. In addition to prioritizing preconditions, one can prioritize the operators themselves or the operator consequences. Furthermore, operators and propositions can be defined that are unique to each level of abstraction. This is common in military planning where the units being planned change with differing levels of command.

2.6 Planning and Learning

Although we do not review it in this document, it is worth noting that research in the classical planning paradigm has often been associated with research in automated learning (e.g., Fikes, et.al., 1977; Minton, 1988). The objective of this research is generally to improve the efficiency of the search through the state-space by extracting from previous plans macro-operators and useful control rules.

2.7 Applications to Associate Technology

Most of the early work in AI planning was done as part of the classical planning tradition. Many of the techniques used by classical planners (nonlinear planning, hierarchical planning, regression) are embedded in the more recent paradigms.

Regarding applications, however, classical planners are not well-suited for associate systems. The limitations of the STRIPS representation are usually to constricting to make this approach viable. Although there are some domains for which STRIPS-like action models are heuristically

adequate (e.g., Wilkins, 1988), these seems more the exception than the rule. Indeed, as Chapman (1987) has noted, the STRIPS representation even has difficulty handling blocks worlds problems where there are blocks of more than one size.

CHAPTER 3

PLANNING AS PLAN REFINEMENT

The plan refinement approach to automated planning, treats planning as a process of constraint posting. That is, beginning with an unconstrained plan (do anything), a series of constraints are posted until it can be determined that any further instantiation of the actions specified consistent with the posted constraints, will achieve the specified goal. This approach is also sometimes called least commitment planning.

The plan refinement approach is closely associated with the classical planning tradition described in the last section. Many of the ideas grew out of nonlinear planning approach where temporal constraints are only posted when necessary. Stefik (1981) developed this idea further by developing a planner that operated by posting variable constraints.

Planners in this tradition fall into two groups. The first group represents a set of planners that address problems that conform to the STRIPS represent. The second represents planners that are designed to handle a broader spectrum of constraints.

3.1 TWEAK

Nonlinear planning is an example of least commitment planning. In this approach, a plan is viewed as a list of constraints and the process of planning is one of adding additional constraints to the plan. The constraint posting process is complete when it can be shown that any further constraints on the actions specified will still result in achieving the goal.

In Sacerdoti's nonlinear planner constraint posting was limited to temporal constraints. However, other types of constraints are possible. In fact, it has been shown (Chapman, 1987) that only three types of constraints are required to solve STRIPS problems:

operator insertions - specifies an operator the must be executed,

variable constraints - restricts the set of possible values for a variable in an operator, and

temporal constraints - requires that a pair of operators be executed

in a certain order.

TWEAK (Chapman, 1987) is a planner that operates entirely by constraint posting. TWEAK's processing is based on a plan evaluation mechanism called the necessary truth criterion. Given the STRIPS assumption, a plan is guaranteed to solve a STRIPS planning problem if and only if every goal and precondition satisfies the necessary truth criteria.

In order to state the necessary truth criterion, we need define the following.

Codesignation - A proposition P codesignates with Q (P==Q) if they represent the same relation and must have the same values for the variables. For examples CLEAR(A)==CLEAR(A), x=y implies ON(A,x)==ON(A,y). On the other hand, $x \neq y$ implies ~(CLEAR(x)==CLEAR(y)).

Asserted-in - A proposition P is asserted in a state s (written asserted-in(P,s)) if and only if P is true in state s.

Asserts - An action A asserts a proposition P (asserts(A,P)) if P is on the add list of A.

Denies - An action A denies a proposition P (denies(A,P)) if P is on the delete list of A.

Necessarily - Given a list of constraints, necessarily P (written []P) is true if and only if no additional variable or temporal constraints can be consistently added which would result in ~P. For instance, if we begin with the Sussman anomaly, then the single action plan MOVE(B,C) would be sufficient to deduce []Es asserted-in(ON(B,C),s). That is, there is necessarily a state in which ON(B,C) is true.

With the above definition, the necessary truth criterion can be formally stated as follows.

[]asserted-in(P,s)

if and only if

$[]\exists t \quad (t<s)$ & asserted-in(P,t) $\wedge$
$\forall C \; [](s<C) \vee$

$$
\begin{aligned}
&[](C<t) \vee \\
&\forall Q\ [] \neg denies(C,Q) \vee \\
&\qquad [] \neg (Q==F) \vee \\
&\qquad \exists W \quad [](C<W) \wedge \\
&\qquad\qquad [](W<s) \wedge \\
&\qquad\qquad \exists R\ asserts(W,R)\ \&\ P==Q \rightarrow Q==R.
\end{aligned}
$$

where P, Q and R are propositions, C and W are actions, and t and s are possible states. In words, the necessary truth criterion reads somewhat as follows: "For any proposition P and state s, P is necessarily true in s if

I. it is necessarily the case that before s there is a state t containing P,

II. for any action C, that action

IIa. occurs before t or after s, or

IIb. C never denies P, or

IIc. whenever C denies P, there is another action W which

IIc1. occurs after C and before s, and

IIc2. asserts P.

A plan is labeled successful if each goal is necessarily true in the final state, and each precondition of each action is necessarily true in the state in which the action is performed. Whenever a goal or precondition is not necessarily true in the appropriate state, then a violation of the necessary truth criterion has occurred.

Using the necessary truth criterion, TWEAK proceeds to solve planning problems as follows.

1. Examine current list of constraints for violations of necessary truth criterion.

2. If no violations exist, exit with current constraints as the plan.

3. If a violation exists, find a constraint that removes violation.

4. If no constraint can be found, backtrack on the constraint last posted and go to 1.

5. If a constraint is found, post the constraint and go to 1.

To illustrate the operation of TWEAK consider, for one last time, the Sussman anomaly. TWEAK first checks to see if the goals ON(A,B) and ON(B,C) are contained in the current state. Since they are not, and the plan is currently null, the necessary truth criterion is violated for both of these goals. This results in adding two step constraints and several variable constraints to the constraint list.

CONSTRAINT TYPE	CONSTRAINT
STEP INSERTION	MOVE(x1,y1)
VARIABLE CONSTRAINT	x1=A
VARIABLE CONSTRAINT	y1=B
STEP INSERTION	MOVE(x2,y2)
VARIABLE CONSTRAINT	x2=B
VARIABLE CONSTRAINT	y2=C

The current plan is shown in Figure 3-1. Note that the two actions are unordered. The current plan is not guaranteed to achieve a goal state because some of the preconditions are not necessarily true. For instance, ON(A,z1) is not necessarily true simply because z1 could be instantiated to an object other than TABLE. Similarly for ON(B,z2). Also CLEAR(B) is not necessarily true because there is an action MOVE(A,B) which denies CLEAR(B), that could be executed prior to the situation in which CLEAR(B) needs to be true. This suggests adding three additional constraints as shown below:

Y1 ≠ TABLE	C ≠ TABLE
Y1 ≠ Z1	C ≠ Z2
ON(A,Z1)	ON(B,Z2)
*CLEAR(A)	CLEAR(B)
CLEAR(B)	CLEAR(C)
MOVE(A,B)	MOVE(B,C)
- ON(A,Z1)	- ON(B,Z2)
- CLEAR(B)	- CLEAR(C)
CLEAR(Z1)	CLEAR(Z2)
ON(A,B)	ON(B,C)

Figure 3-1: First Partial Plan

CONSTRAINT TYPE	CONSTRAINT
STEP INSERTION	MOVE(x1,y1)
VARIABLE	x1=A
VARIABLE	y1=B
STEP INSERTION	MOVE(x2,y2)
VARIABLE	x2=B
VARIABLE	y2=C
VARIABLE	z1=TABLE
VARIABLE	z2=TABLE
TEMPORAL ORDER	MOVE(x2,y2) before MOVE(x1,y1).

The current plan is now shown in Figure 3-2. It is still not complete since CLEAR(A) is still not satisfied. Since no additional variable or temporal order constraints can be added, a new step is added to the plan that asserts CLEAR(A). This new step must occur prior to the situation in which CLEAR(A) is needed. This results in the following constraint list.

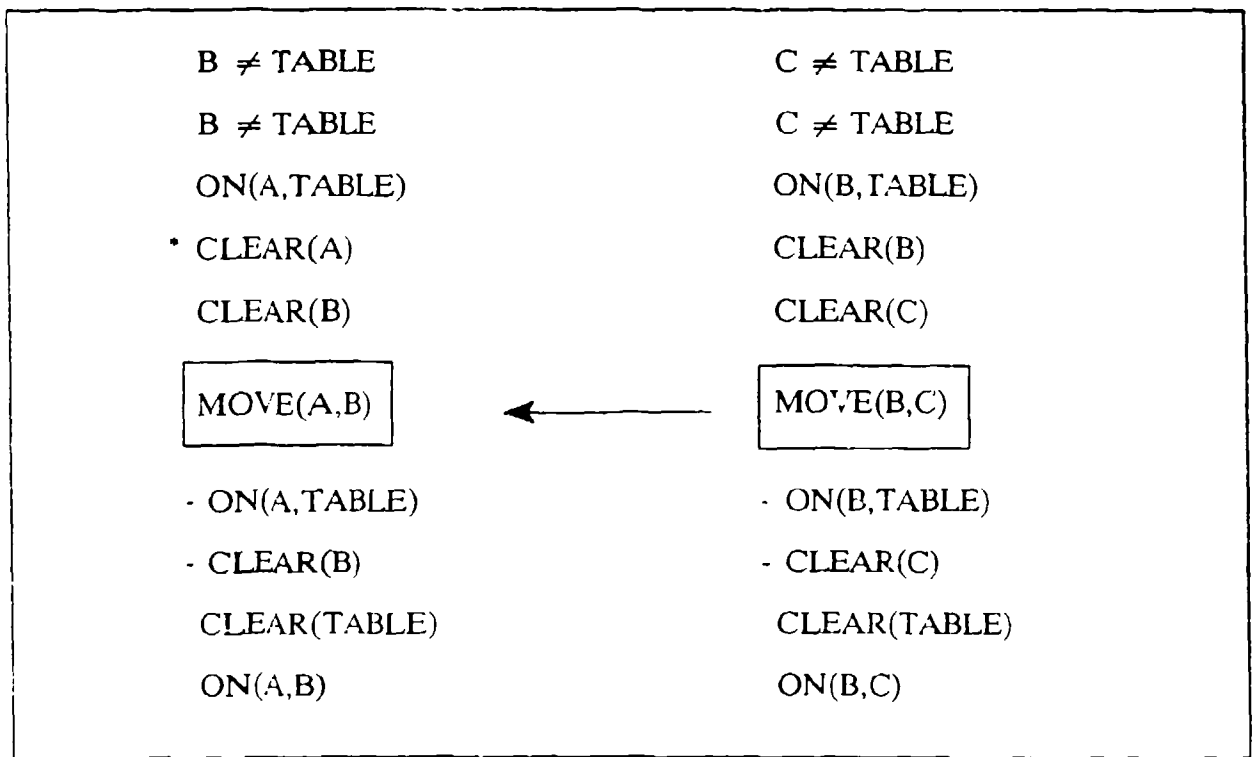


Figure 3-2: Second Partial Plan

CONSTRAINT TYPE	CONSTRAINT
STEP INSERTION	MOVE(x1,y1)
VARIABLE	x1=A
VARIABLE	y1=B
STEP INSERTION	MOVE(x2,y2)
VARIABLE	x2=B
VARIABLE	y2=C
VARIABLE	z1=TABLE
VARIABLE	z2=TABLE
TEMPORAL ORDER	MOVE(x2,y2) before MOVE(x1,y1)
STEP INSERTION	NEWTOWER(x3)
VARIABLE	x3=C
VARIABLE	z3=A
TEMPORAL ORDER	NEWTOWER(x3) before MOVE(x1,y1).

The current plan is shown in Figure 3-3. There is one final problem. The precondition CLEAR(C) is denied by the action MOVE(B,C). Consequently, MOVE(B,C) must occur before CLEAR(C) is needed. This results in the final constraint list:

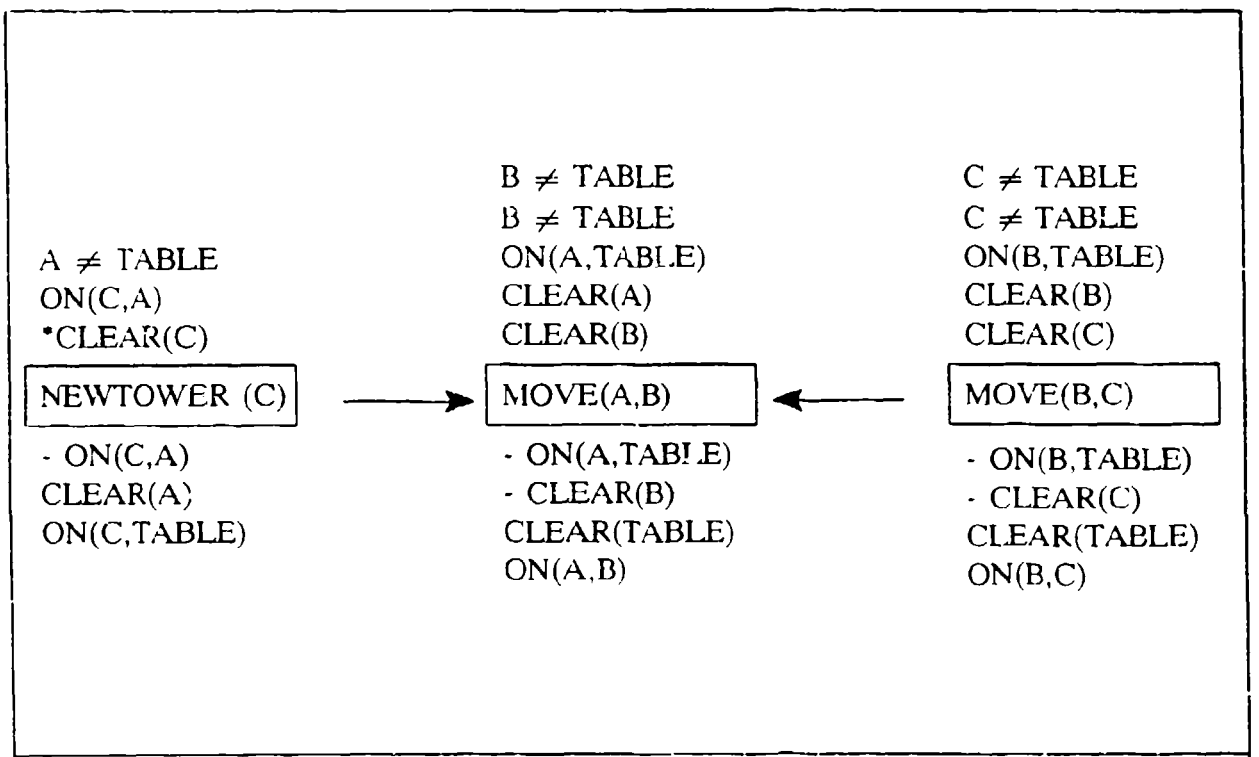


Figure 3-3: Third Partial Plan

CONSTRAINT TYPE	CONSTRAINT
STEP INSERTION	MOVE(x1,y1)
VARIABLE	x1=A
VARIABLE	y1=B
STEP INSERTION	MOVE(x2,y2)
VARIABLE	x2=B
VARIABLE	y2=C
VARIABLE	z1=TABLE
VARIABLE	z2=TABLE
TEMPORAL ORDER	MOVE(x2,y2) before MOVE(x1,y1)
STEP INSERTION	NEWTOWER(x3)
VARIABLE	x3=C
VARIABLE	z3=A
TEMPORAL ORDER	NEWTOWER(x3) before MOVE(x1,y1)
TEMPORAL ORDER	NEWTOWER(x3) before MOVE(x2,y2).

The final plan is shown in Figure 3-4.

Note that the performance of TWEAK depends on the selection of constraints to add to the plan. In the above example, we fortuitously selected the best constraint to add at each choice point. In general the quality of the plans generated by TWEAK will depend on the heuristics for selecting among alternative possible constraints.

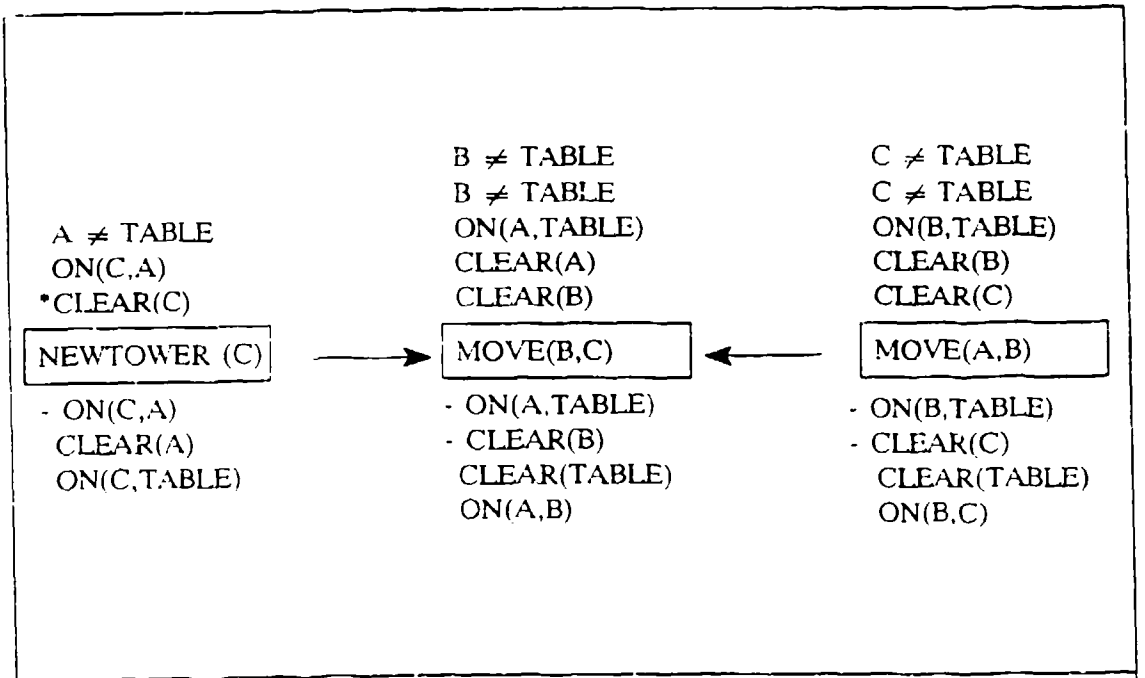


Figure 3-4: Final Plan

In some ways TWEAK is the culmination of the line of research in classical planners. This is because TWEAK satisfies the following property (Chapman, 1987).

> *Property 1*: For any STRIPS problem, TWEAK will find a plan in finite time if one exists. If no plan exists, TWEAK will either return with no solution or will continue processing.

Furthermore, for STRIPS problems in general one can show the following property.

> *Property 2*: There does not exist a procedure which is guaranteed to terminate in finite time for all STRIPS problems without a solution, yet still be guaranteed to find a solution if one exists.

In other words, from a decidability perspective, no planner can improve on TWEAKs performance. Although there are other planners that are more efficient than TWEAK (e.g., McAllester and Rosenblitt, 1991), fundamental improvements are not possible.

3.2 Context-Dependent Consequences

One of the principle criticisms of planners designed to handle STRIPS problems is that the add/delete model of action consequences is unrealistic and severely constrains the class of problems to which these planners can be applied. The main problem is that add/delete lists are *context independent*. An action has the same consequences no matter what situation the action is executed in. As a result, add/delete lists are sometimes difficult to specify. Consider, for instance, a blocks world with blocks of several different sizes. In this domain, the consequences of MOVE(A,B) depends on the size of blocks A and B and the amount of free space initially on block A. In short, the consequence of MOVE(x,y) is a function of the input situation and cannot be specified with a context independent add/delete list.

Also, simple add/delete list make it difficult to plan in multi-agent environments, where the consequence of an action depends on the actions simultaneously being pursued by other agents.

To overcome these limitations, several researchers have extended the classical and nonlinear planners to incorporate *context-dependent* consequences. Formal theory development (e.g., Pednault, 1988) has

resulted in the specification criteria that are comparable to the modal truth criterion but are applicable to operators with context-dependent consequences. Unfortunately, when context-dependent effects are allowed, the problem of determining whether a proposed plan violates these criteria is no longer decidable. Consequentially, even if a plan exists a planner that allows context-dependent consequences can not be guaranteed to terminate with a solution.

Implemented planners with context-dependent effects take a more practical approach. The most well-known example is Wilkin's (1988) System for Interactive Planning and Execution (SIPE). In SIPE, a predetermined set of context-dependent effects are calculated as needed. This approach is not guaranteed to generate sound plans, but is according to Wilkins "heuristically adequate." It is unlikely that SIPE will generate a plan where an uncalculated context dependent effect makes the plan unsound.

It is important to note that both Wilkins and Pednault make what Wilkins calls the *Extended STRIPS Assumption*. This assumes that the only things that change as a consequence of an action are the direct consequences of that action (e.g., added and deleted propositions), and *indirect consequences* that can be deduced as having changed. That is, any statement P that was true prior to executing an action is assumed to be true after the action is executed, unless it can be deduced that it is possibly false.

3.3 Generalized Constraint Processing

The general idea of plan refinement is that the planner posts constraints until it is satisfied (by some evaluation criteria) that any specific plan that is consistent with all posted constraint will achieve the goal state. This general strategy is not limited to domains satisfying the STRIPS assumption, but can be applied to any domain where constraints can be specified. Although this approach to planning has not been articulated as an independent paradigm, there is a great deal of theoretical and applied work that follows this perspective. This work includes the following.

Constraint Reasoning - There is a growing literature directly addressing constraint satisfaction problems (Mackworth, 1987). These systems accept as input a set of variable constraints and attempt to prove the consistency or inconsistency of these constraints. Often consistency

proofs are achieved by finding specific values that will satisfy all constraints.

Temporal Constraint Reasoning - An important class of constraints deals with temporal variables. Here the objective is to determine if a set of temporally ordered statements (e.g., E1 before E2, E2 starts after 3:00, E1 ends after 2:30, ...) is consistent. Here the approaches differ depending on whether they deal with qualitative constraints on time intervals (e.g., Allen, 1983; Ligozat, 1991), quantitative constraints on time points (e.g., Dechter, et. al., 1989), or some combination of qualitative and quantitative constraints (e.g., Kautz, H. and Ladkin, P., 1991). In either case these techniques are emerging as a powerful tool for reasoning about the temporal consistency of a proposed plan.

Temporal Data Management - As described by Dean and McDermott (1987) Temporal Data Base Management Systems (TDMBS) go beyond temporal constraint reasoning. They also provide some non-monotonic temporal inferencing. For instance, after asserting that proposition P1 became true after action A1 occurred, a TDBMS would "assume" P1 remained true until some other event occurs that would make P1 possibly false. This allows the TDBMS to make stronger deductions than are warranted by simple temporal constraint reasoning. For instance, from the statements:

TRUE(P1,t) & OCCURS(E1,t) --> TRUE(P2,t+1),
TRUE(P1,T1),
OCCURS(E1,T2),
T2>T1,

a TDBMS could deduce TRUE(P2,T2+1) because it had no reason to deduce that between times T1 and T2 the proposition P1 might become false.

Temporal Data Management and Reason Maintenance - In addition to non-monotonically jumping to conclusions, the TDBMS must also be able to retract these infeasible inferences. In the above example, for instance, if the TDBMS later learns

OCCURS(E2,T3)
OCCURS(E2,t) --> ~TRUE(P1,t+1),

then the TDBMS should retract the deduction TRUE(P2,T2+1) because T3 may be after T1 but before T2.[1] To achieve this, the TDBMS needs to incorporate a reason maintenance capability to keep track of the justification for each non-monotonic conclusion. Dean and McDermott's TDBMS had some reason maintenance capability. Recent work in this area (Hamscher, 1991; Goldstone, 1991) have further developed the relationship between temporal constraint reasoning and reason maintenance systems.

In addition to the theoretical work, the plan refinement approach is also found in a number of application systems. A well known example is the TEMPLAR (Tactical Expert Mission Planner). This system fills out a daily air tasking order by sequentially posting a sequence of resource constraints (i.e., each aircraft, ordnance, etc. assignment is processed as a constraint on that resource). Another example is found in Meng and Lehnert (1991) where strike plans are directly represented as a set of temporal constraints. Maintaining consistency between multiple plans is then treated as a temporal constraint satisfaction problem.

3.4 Skeletal Planning

In the above discussion, we assumed that the planning process begins from scratch. However, in some systems planning begins by retrieving a partial partial plan that contains the major steps of the plan. A pre-stored partial plan is often referred to as a *skeletal plan*. *Skeletal planning* is a variant of the constraint-based approach that solves planning problems by trying to instantiate one of a set of pre-stored skeletal plans (Stefik, 1981). Each skeletal plan can be viewed as a set of hard constraints. Additional constraints are added until the plan is complete (as determined by a mechanism such as the necessary truth criteria), or it is determined that the skeletal plan cannot be instantiated (i.e., a hard constraint needs to be retracted).

[1] Alternatively, it may recognize that the conclusions TRUE(P2, T2+1) is justified by the assumption T3>T2 or T3<T1.

3.5 Relevance to Associate System Technology

3.5.1 Plan Generation

The plan refinement approach is very general. In principle, any mission planning problem can be addressed using this paradigm. However, to effectively apply this paradigm to mission planning two issues must be addressed. First, we need to identify how to represent constraints relevant to mission planning. Second, we need to specify a reasonable mechanism for searching through the space of partial plans. One approach to addressing both of these issues is provided below. Although the approach described below can certainly be improved upon, it does illustrate the main point. Namely that mission planning is a problem domain that is fully amenable to a constraint posting approach to automated planning.

The problem of representing constraints is not trivial. Any constraint language one develops should satisfy several criteria. First, the full spectrum of constraints relevant to mission planning should be representable. It does little good to develop a language that can only handle a subset of the constraints, since solutions satisfying these constraints are as likely as not to be unrealistic. Second, the constraint language should be understandable to the user community -- mission planners. This will make it convenient for the users to control the automated planning process. Third, the language should conform to standard AI practices. Otherwise, the mapping of appropriate AI techniques becomes more difficult.

As it turns out, there already exists within the mission planning community a set of well-defined models that can be used to characterize and evaluate proposed missions. These models are in the form of functions that predict various mission characteristics as a function of input variables. Figure 3-5 shows graphically the functional relationships between some of the variables that characterize a mission plan. For each node in the graph a function is defined that specifies the value on that node as a function of the values on the input nodes. Consequently, these functions can be used to assess whether a fully specified mission plan satisfies a set of goal constraints.

The constraint posting paradigm requires a mechanism to evaluate partial plans. This is achieved by using a set of simple heuristics to temporarily complete a plan, and then evaluate the completed plan. In this way, a *worst case completion* of a partial plan is generated. If the worst case completion satisfies the goal constraints then the partial plan is

accepted, since the planner now knows that a satisficing completion of this partial plan is always achievable. In addition, a set of *best case estimates* are also needed. For each incomplete segment of the plan a heuristic overestimate of the best possible value achievable on the goal criteria is calculated. For instance, if two way points have been specified, but not the route between them, then minimum fuel consumption can be specified using a high altitude, straight line path.

The process of planning may now proceed very much as it does in TWEAK. The goal state is characterized as a set of hard constraints (time-over-target, other timing constraints, required way points, etc.) and evaluation criteria (minimum probability of arrival, minimum probability of destruction, etc.). Beginning with this initial list, planning proceeds as follows:

1. Using the current list of constraints, evaluate the worst case plan to determine if any criteria are violated.

2. If no violations exist and the plan is complete, exit with current constraints as the plan.

3. If no violation exists and the plan is incomplete, heuristically select constraints that will improve the plan on the evaluation criteria.

4. If a violation exists, heuristically select constraints that will improve the plan on the violated evaluation criteria.

5. Evaluate the plan using the best case estimates and determine if any evaluation criteria are violated. If there is a best case violation, backtrack on the most recently posted constraints.

6. If backtracking fails, exit with no plan.

7. If constraint are found, post the constraints.

8. Go to 1.

As the above discussion illustrates, the general mission planning problem is compatible with a constraint processing approach. There are, however, two possible problems with this approach. First, the plan refinement paradigm is a satisficing paradigm. There is no guarantee that a plan generated via plan refinement will be the lowest cost or even a low cost plan. This contrasts sharply with globally-directed search procedures,

such as A*, that always generate the lowest cost path. Indeed even for moderately complex problems there is some evidence (Freeman, 1991) that indicates that a STRIPS-like approaches will generate very poor plans. Second, the efficiency of this procedure is unclear. There is no a priori reason to believe that searching through a large space of possible constraints involves a lesser computational burden than searching the state space.

One way around both of these problems is to rely on a carefully engineered library of skeletal plans each of which specifies the main steps in a plan. When skeletal plans are used, then the focus of the plan refinement process is to instantiate the skeletal plan. Often this is just a matter of finding values for the variables in the skeletal plan, although some temporal ordering and step insertion constraints may be involved. If the refinement process begins with a reasonable skeletal plan, then a satisfactory plan should be quickly generated.

3.5.2 Plan Modification

Refinement based procedures are not particularly well-designed for addressing plan modification problems. The principal way to do plan modification (e.g., Wilkins, 1988) is to (1) identify why the current plan is invalid, (2) sequentially remove constraints until a partial plan is found that is no longer invalid, (3) initiate the refinement process using the partial plan as the starting point. In short, replan by beginning with a partial, but viable plan. Note that this approach is likely to show a tendency to find modified plans that just barely avoid the initial problem.

3.5.3 Real-time Planning

The most obvious approach to achieving real-time performance with the plan refinement strategy is to rely on a library of well-engineered skeletal plans. Planning then becomes a relatively simple problem of selecting and instantiating appropriate skeletal plans. Skeletal planning is the approach used for generating tactical advice in the Pilot's Associate. The main weakness of this approach is its dependence on the quality of the skeletal plans. It presupposes an ability to anticipate the problems that an operator will face and to engineer a priori solutions to those problems.

An alternative approach to achieving real time performance has been recently suggested by Boddy (1991). Boddy's approach is to employ two

levels of problem solvers. The first level is a constraint posting problem solver that sequentially adds consistent temporal and variable constraints to the plan. The second is a simple and quick problem solver that will quickly complete any partial plan, but without any guarantee of avoiding constraint violations. Planning proceeds by using the constraint posting problem solver, but if the planning process is interrupted with a need for an immediate answer, then the more rapid problem solver can be invoked to quickly complete the plan. In this way anytime problem solving (Dean and Boddy, 1988) behavior is achieved; where the quality of a plan improves as time available for problem solving increases.

CHAPTER 4

PLANNING AS PLAN TRANSFORMATION AND CASE-BASED PLANNING

In the plan refinement paradigm planning is achieved by beginning with a null or abstract (skeletal) plan and incrementally adding details to a plan. There is no mechanism for modifying a detailed plan to fit a new situation. In contrast, the plan transformation paradigm suggest that planning is often a matter of modifying a detailed plan so as to fit a new circumstance. Proponents of the transformational approach seem to differ as to the extent to which they claim planning is solely a matter of detailed plan transformation. There seem to be three loose camps

Transformations for Plan Repair - Plan transformations are primarily useful for addressing plan repair problems. An initial plan may have been carefully constructed using the refinement approach and transformations are used to repair the plan when unexpected events occur (e.g., Wilkins, 1988; Ambros-Ingerson & Steel, 1988).

Transformations for Plan Improvement - Plan transformation can be used to transform a hastily constructed plan into a viable plan. Plan generation is achieved by using a crude mechanism for generating an initial plan and then modifying that plan until a satisfactory plan emerges (e.g., Linden, 198?).

Case-Based Planning - Planning is largely a matter of adapting plans that have worked in similar situations. Through experience the planner builds a library of past cases. When addressing a new problem, a retrieval mechanism posts a plan that worked in a similar situation. The retrieved plan is then modified to fit the particulars of the current situation (Hammond, 1989).

4.1 Mechanisms for Plan Transformation

All three of the transformational paradigms require an effective procedure for modifying a plan. By far the most common approach is to use a set of *failure-fix* rules, where the precondition of the rule is a problem test (a "failure") and the consequence of the rule is a procedure for modifying the plan (a "fix"). Failure-fix rules can range in complexity

from very simple rules (e.g., drop a variable binding) to a complex set of changes that represents a single modification.

To illustrate the process of plan transformation, consider the problem shown in Figure 4-1. Assume a hypothetical planner that must move the blocks from the initial state to the goal state. As in Section 3.0 we will assume that plans are represented as a set of constraints. Since the plan generated by TWEAK achieves a goal with the same structure as the current goal, it is recalled as a plan to try to transform. We therefore begin with the following plan.

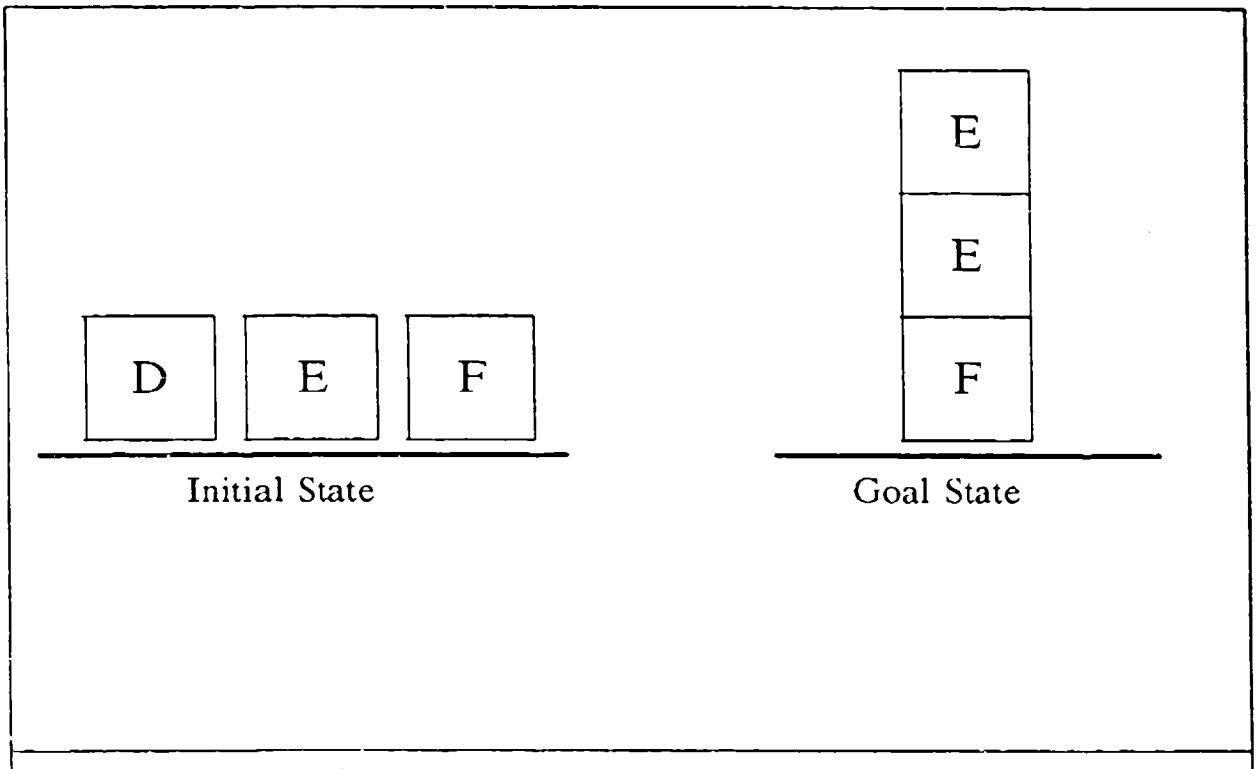


Figure 4-1: Blocks World Problem for Transformational Planning Example

CONSTRAINT TYPE	CONSTRAINTS	MODIFICATION
STEP INSERTION	MOVE(x1,y1)	NONE
VARIABLE	x1=A	
VARIABLE	y1=B	
STEP INSERTION	MOVE(x2,y2)	
VARIABLE	x2=B	
VARIABLE	y2=C	
VARIABLE	z1=TABLE	
VARIABLE	z2=TABLE	
TEMPORAL ORDER	MOVE(x2,y2) before MOVE(x1,y1)	
STEP INSERTION	NEWTOWER(x3)	
VARIABLE	x3=y2	
VARIABLE	z3=x1	
TEMPORAL ORDER	NEWTOWER(x3) before MOVE(x1,y1)	
TEMPORAL ORDER	NEWTOWER(x3) before MOVE(x2,y2).	

Upon evaluating this plan, our hypothetical planner notes that the objects A, B, and C are not relevant to the current problem. This "incorrect bindings" problem can be repaired with an "unbind variables" procedure. This leads to the following constraint list:

CONSTRAINT TYPE	CONSTRAINTS	MODIFICATION
STEP INSERTION	MOVE(x1,y1)	
VARIABLE		unbind(x1)
VARIABLE		unbind(y1)
STEP INSERTION	MOVE(x2,y2)	
VARIABLE		unbind(x2)
VARIABLE		unbind(y2)
VARIABLE	z1=TABLE	
VARIABLE	z2=TABLE	
TEMPORAL ORDER	MOVE(x2,y2) before MOVE(x1,y1)	
STEP INSERTION	NEWTOWER(x3)	
VARIABLE	x3=y2	
VARIABLE	z3=x1	
TEMPORAL ORDER	NEWTOWER(x3) before MOVE(x1,y1)	
TEMPORAL ORDER	NEWTOWER(x3) before MOVE(x2,y2).	

Now the planner notices that the goal state ON(D,E) is not asserted anywhere in the plan. This problem can be repaired by binding x1 to D

and y1 to E. Similarly, ON(E,F) can be asserted by binding x2 and y2 appropriately. This results in the following plan.

CONSTRAINT TYPE	CONSTRAINTS	MODIFICATION
STEP INSERTION	MOVE(x1,y1)	
VARIABLE	x1=D	bind(x1)
VARIABLE	y1=E	bind(y1)
STEP INSERTION	MOVE(x2,y2)	
VARIABLE	x2=E	bind(x2)
VARIABLE	y2=F	bind(y2)
VARIABLE	z1=TABLE	
VARIABLE	z2=TABLE	
TEMPORAL ORDER	MOVE(x2,y2) before MOVE(x1,y1)	
STEP INSERTION	NEWTOWER(x3)	
VARIABLE	x3=y2	
VARIABLE	z3=x1	
TEMPORAL ORDER	NEWTOWER(x3) before MOVE(x1,y1)	
TEMPORAL ORDER	NEWTOWER(x3) before MOVE(x2,y2)	

At this point the planner discovers that the consequences of NEWTOWER(F) are already true in the current state. Consequently, the plan has an "irrelevant action" problem that can be repaired with a "remove action" procedure. This leads to the final plan:

CONSTRAINT TYPE	CONSTRAINTS	MODIFICATION
STEP INSERTION	MOVE(x1,y1)	remove NEWTOWER(x3)
VARIABLE	x1=D	unbind(x3)
VARIABLE	y1=E	unbind(z3)
STEP INSERTION	MOVE(x2,y2)	
VARIABLE	x2=E	
VARIABLE	y2=F	
VARIABLE	z1=TABLE	
VARIABLE	z2=TABLE	
TEMPORAL ORDER	MOVE(x2,y2) before MOVE(x1,y1)	

As this example illustrates, a set of failure-fix rules can be used to transform an invalid plan into one that is valid. Our hypothetical planner, however, is lacking one feature often found in transformational planner - a plan justification (Ambros-Ingerson & Steel, 1988, Kambhampati, S., 1990). Plan justifications are a record of why each element of the plan was added. Such a record makes it easier to identify appropriate transformations,

since the justifications can be examined to determine if they are still valid. For instance, in the plan generated by TWEAK, the action NEWTOWER(x3) & x3=x1 was added to the plan so that a precondition of MOVE(x1,y1) could be established, namely CLEAR(x1). In the situation described above, once x1=D the condition CLEAR(x1) was true in the situation prior to NEWTOWER(x3). Consequently, the justification for including NEWTOWER(x3) in the plan becomes invalid.

4.2 Case-Based Planning

The case-based planning paradigm emphasizes the importance of episodic memory in automated planning. Through experience the planner builds a library of cases where each case represents a specific episode that is relevant to planning. Episodic case information is used in a variety of ways. As discussed in Hammond (1989) these include:

Problem Anticipation - Potential planning problems are identified by matching current situation features with features of cases of past planning problems.

Plan Retrieval - Plans are retrieved by finding previous plans that match as many of the current goals as possible while minimizing the number of problems anticipated.

Plan Modification - Plans are modified by applying a set of failure-fix rules. These rules may themselves be retrieved by recalling cases of similar failure and the modifications that corrected those failures.

Plan Repair - Like plan modification, plan repair involves applying failure-fix rules that may correspond to previous of cases plan failure and repair.

The success of a case-based planner depends in large measure on its ability to appropriately index and retrieve relevant cases and modifications. If the retrieval mechanism retrieves inappropriate cases, then it is unlikely that the case-based planner will iterate to a satisfactory plan.

4.3 Relevance to Associate Technology

4.3.1 Plan Generation

For plan generation the transformational paradigm is comparable to the refinement paradigm. A transformation can be treated as two refinement steps: (1) relax some constraints and (2) post new constraints. The principal difference is that the transformational approach begins the search process with a specific plan, whereas the refinement approach begins with a null or skeletal plan. There is no a priori reason to believe that one or the other approach will be more efficient or generate better plans. Rather efficiency and plan quality will depend more on the initial starting point (skeletal plan or case) and the nature of the refinements (transformations) than on any inherent characteristic of the paradigm.

The SOAS project is currently committed to a transformational approach (Berg-Cross, 1991). In particular, the planner architecture in SOAS uses a variant of the case-based approach. The SOAS planner uses a case-based planning architecture. However, rather than rely on episodic memory the cases are knowledge engineered. In this way, the planner is likely to find "cases" of plans and plan modifications that are applicable to any situation encountered. This is similar to the skeletal planning approach used in Pilot's Associate except that rather than instantiating a knowledge engineered skeletal plan, the SOAS planner must modify a knowledge engineered detailed plan.

4.3.2 Plan Modification

A transformation can be viewed as the compilation of several refinement steps. Consequently the refinement and transformational approach are comparable in addressing plan modification problems. However, a planner that uses a well-engineered set of transformations is likely to be considerable more efficient than a refinement-based approach. This will be particularly true if the refinement procedures requires that the planner search through multiple possible sequences of refinements before it "discovers" the successful transformation.

4.3.3 Real-time Planning

As with plan generation in general, the quality and efficiency of real-time planning will depend on the quality of the implementation -- and not

whether a refinement or transformational approach is selected. Particularly key to the transformational approach is whether the retrieval mechanism quickly identifies appropriate modifications or whether the planner must backtrack from a set of inappropriate modifications. As noted above, SOAS attempts to rapid planning by carefully selecting appropriate initial plans and modifications.

CHAPTER 5

PLANNING FROM FIRST-PRINCIPALS

Within AI there is strong tradition of logicism -- a general approach to reasoning that attempts to reduce every problem to formal logic. This is true in automated planning as well. Specifically, the pure logicist would use statements in a formal logic to represent a problem domain and then proceed to use automated theorem proving techniques to find a plan.

A formal logic is usually composed of three parts: language, deduction system, and semantics. The language specifies all the allowable symbols of a logic and how those symbols may be combined to form statements. The deduction system specifies how to apply the language to generate deductions. The semantics specify what the symbols and statements in a language "mean". Here "meaning" typically is defined by what each elements of the language denotes or represents. For instance, the simple statement ON(A,B) is intended to represent a situation where the block that symbol A denotes has the relation *on top of* with respect to the block that symbol B denotes, where *on top of* is a relation that the symbol ON denotes.

In the philosophical logic literature, there is a long standing debate as to the adequacy of alternative logics (Haack, 1978). *Classical* logicians argue that the only correct and necessary logic is predicate calculus, specifically first-order predicate calculus (FOPC). *Nonclassical* logicians suggest that predicate calculus is generally inadequate and that alternative logics are appropriate for different circumstances. The debate between classical and nonclassical logic is also found in the AI literature. For virtually any type of automated reasoning problem one finds both a classical and nonclassical approach to addressing that problem.

In this chapter, we will focus on the application of FOPC to automated planning and a simple extension called default logic. All of the major issues can be demonstrated using these logics.

5.1 First Order Predicate Calculus (FOPC)

As mentioned above a logic is composed of three elements: language, deduction system and semantics. Each of these parts for FOPC are described below.

5.1.1 Language and Intended Denotation

The symbols of FOPC and their intended denotation are described below.[1]

Constant symbols serve as names for the objects being denoted.

Variable symbols serve to denote any of a set of objects.

Function symbols reference functions that map from one or more objects into an object.

Predicate symbols denote properties of individual objects and relations between objects.

Any constant or variable symbol is a *term*. Any function of a term(s) is also a term. A predicate with the correct number of terms as arguments is called a *well-formed formula* (wff) or a statement. For example, the wff Type-label(AC#7,F-14) denotes that the object denoted by the constant symbol AC#7, and the object denoted by the constant symbol F-14 have the relation Type-label. Similarly, the expression Is-pilot(commander(AC#7)) asserts that the object that corresponds to the term commander(AC#17) satisfies the property Is-pilot.

Operators allow the construction of complex wffs from simple wffs. The operators normally used in FOPC are ~ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), and $\rightarrow$ (implication).[2]

Quantifiers allow the construction of wffs that make assertions about sets of objects. The quantifiers normally used in FOPC are $\forall$ (universal quantification) and $\exists$ (existential quantification).

For instance, the expression

$\forall x$[Type-label(x,F-14) $\rightarrow$ (Role(x,FIGHTER) $\vee$ Role(x,BOMBER))]

---

[1]Please note that this is a very informal introduction to FOPC. In a rigorous introduction, the language and semantics of a logic would be carefully separated. For our purposes, however, this is unnecessary.

[2]Formally, only two operators (e.g., ~ and -->) are needed to define an FOPC language. The others can be defined in terms of the two initially selected. Similarly, existential quantification can be defined in terms of unviersal quantification, and vice versa. So only one of these is needed to define the language.

asserts that for all objects, if the object is an F-14 then the role of that object is either FIGHTER or BOMBER.[3]

5.1.2 Deduction

A *deduction system* allows one to draw conclusions from an initial database of wffs. In FOPC the deduction system is often characterized by an *axiom system*, that specifies a set of *axiom schemas* and *rules of inference*. A typical axiom system for FOPC is the following

Axiom Schemas

(p, q and r represent any wffs, t represents any term, x represents any variable)

(A1) $p \rightarrow (q \rightarrow p)$
(A2) $(p \rightarrow (q \rightarrow r)) \rightarrow ((p \rightarrow q) \rightarrow (p \rightarrow r))$
(A3) $(\sim q \rightarrow \sim p) \rightarrow (p \rightarrow q)$
(A4) $\forall x[(p \rightarrow q) \rightarrow (p \rightarrow \forall x[q])]$[4]
(A5) $\forall x[p(x)] \rightarrow p(t/x)$
where $p(t/x)$ means p with all unbound instances of x are replaced with the term t.[5]

Rules of Inference (DB is an initial database of wffs)

Modus Ponens (MP) - If DB contains the wffs p and $p \rightarrow q$, then conclude q.

Generalization (GEN) - If DB contains the wff p then conclude $\forall x p$.

In FOPC the only conclusion allowed are wffs that are either (a) contained in the original data base, (b) instantiate one of the axiom schemas, or (c) can be derived from repeated applications of the rules of inference. For instance, from the data base

---

[3]As needed, we also throw in nonlogical symbols (",")", "[", and "]") to help mark the scope of each operator and quantifier. These symbols are only markers, and have no intended meaning.

[4]Subject to the constraint that the wff p does not contain the variable x outside the scope of a quantifier.

[5]Subject to the constraint that all variables in t are free wherever x occurs free in p.

DB = { $\forall$x[Type-label(x,F-14) $\rightarrow$ (Role(x,FIGHTER) v Role(x,BOMBER))],
$\forall$x[Member-of(x,SQUADRON#2) $\rightarrow$ Type-label(x,F-14)],
Member-of(AC#6,SQUADRON#2) }.

The conclusion

Role(AC#6,FIGHTER) v Role(AC#6,BOMBER)

can be generated via the following sequence of conclusions:

		How Deduced
C1	$\forall$x[Member-of(x,SQUADRON#2)) $\rightarrow$ Type-label(x,F-14)] $\rightarrow$ Member-of(AC#6,SQUADRON#2) $\rightarrow$ Type-label(AC#6,F-14)	A5
C2	(Member-of(AC#6,SQUADRON#2) $\rightarrow$ Type-label(AC#6,F-14))	DB2+C1+MP
C3	Type-label(AC#6,F-14)	DB3+C2+MP
C4	$\forall$x[Type-label(x,F-14) $\rightarrow$ (Role(x,FIGHTER) v Role(x,BOMBER))] $\rightarrow$ (Type-label(AC#6,F-14) $\rightarrow$ (Role(AC#6,FIGHTER) v Role(AC#6,BOMBER)))	A5
C5	(Type-label(AC#6,F-14) $\rightarrow$ (Role(AC#6,FIGHTER) v Role(AC#6,BOMBER)))	DB1+C4+MP
C6	Role(AC#6,FIGHTER) v Role(AC#6,BOMBER)))	C3+C5+MP

5.1.3 Soundness and Completeness

The usefulness of FOPC comes from the fact that it satisfies the following two properties.[6]

Soundness - No matter what objects or relations are denoted, if a conclusion C is deduced from a database DB, then there does not exist a possible state where the statements in DB are true and the statement C is not true.

[6]This is a very informal description.

The soundness property asserts that the FOPC deduction system will only deduce truths from truths. Or put another way, if the premises of a data base hold true, then the conclusions drawn from that data base are inevitably true.

> *Completeness* - If all the possible states that are consistent with a database DB contain the wff C, then C can be deduced from the database DB in finite time.

The completeness property asserts that every valid conclusion of a data base can be deduced using an FOPC deduction system.

Because FOPC is sound and complete, it provides a very general and useful approach to inference. If a problem can be formulated as a set of FOPC wffs, and a solution to that problem exists, then it is guaranteed that an FOPC deduction system can find that solution. This property extends to planning problems as well. If the problem can be formulated as a set of wffs, and we can express the question "Does there exist a plan that ..." as a wff, then FOPC is guaranteed to find a plan if one exists. Also, the FOPC deduction system will never return a plan which, by the formulation of the problem, does not work. (Note however that if no solution exists then the FOPC deduction system may not be able to make that deduction and continue processing forever.)

5.2 The Situation Calculus and other Temporal Logics

5.2.1 The Situation Calculus

The situation calculus is an example of an FOPC approach to formulating planning problems. It provides a convenient mechanism for expressing situation-specific truths. To illustrate, consider the wff Fuel-status(AC#6,LOW) indicating that AC#6 is low on fuel. In the situation calculus a statement such as this would be expressed as

Holds(fuel-status(AC#6,LOW),S1)

indicating that in situation S1 fuel-status(AC#6,LOW) is true. Note that the expression fuel-status(AC#6,LOW) is now an argument of a predicate. Consequently, it is no longer a wff, but now has the status of a term. Terms that are intended to reflect situation-specific truths are referred to as *fluents*.

Using the situation calculus it is now possible to describe situations and how actions change situations. The consequences of the MOVE operator could be characterized by the following statements:

a1. $\forall axyzs$ [Holds(at(a,x,y,z),s) $\rightarrow$ Holds(alt(a,z),s)

a2. $\forall axzs$ [(Holds(alt(a,z),s) & ~(x=z)) $\rightarrow$ ~Holds(alt(a,x),s)]

a3. $\forall awxyzs$ [Holds(at(a,x,y,z),s) $\rightarrow$
Holds(at(a,x,y,plus(z,w)),result(increase-alt(a,w),s)

a4. $\forall awxyzs$ [Holds(at(a,x,y,z),s) $\rightarrow$
Holds(at(a,x,plus(y,w),z),result(move-north(a,w),s)

a1 and a2 are domain axioms that describe necessary relationships between fluents in a situation. a3 and a4 are effect axioms that describe the consequences of various actions. In particular, a1 uses the term result(increase-alt(a,w),s) to indirectly reference "the situation that results from increasing altitude w units in situation s." By nesting "results" we can reference the situation that results from a sequence of actions. For example,

```
result(move-north(AC#6,20),
   result(increase-alt(AC#6,100),
      result(increase-alt(AC#6,50),S1)
```

denotes the situation that is the result of moving AC#6 north 20 units in the situation that results from increasing altitude 100 units in the situation that results from increasing altitude 50 units in situation S1. That is, it denotes the situation that results from performing the following sequence of actions in S1:

increase-alt(AC#6,50) -> increase-alt(AC#6,100) -> move-north(AC#6,20).

Now that we see how to describe situation specific facts and action sequences, the next step is straight forward. In order to find a plan to achieve a goal G one simply applies an FOPC theorem prover to a wff of the form $\exists s$[Holds(G,s)]. If a situation exists that satisfies the goal then the theorem will find that situation and will refer to it by stating the sequence of actions required to reach that situation. For instance, if we begin with a data base that contains a1 through a4 above and the wff

Holds(at(AC#6,10,10,10),S1)

and submit for proof the theorem ∃s[Holds(at(AC#6,10,30,20),s]. The theorem prover will return a proof of a wff of the form:

Holds(at(AC#6,10,30,20),
result(increase-alt(AC#6,10),
result(move-north(AC#6,20),S1)).

5.2.2 Temporal Logics

The situation calculus was one of the first logics proposed for use in automated planning (Green, 1969; Kowolski, 1979). It was soon realized, however, that its ability to represent temporally ordered facts was limited. For example, in the situation calculus actions and their effects are discrete. The ability to describe processes that evolve gradually over time is limited. Also it is difficult to represent situations where multiple events and processes occur simultaneously.

Because of the limits of the situation calculus, a variety of more expressive temporal logics were developed. In these logics, rather than bind the truth value of a fluent to situations, they are bound to time points or time intervals. For instance, in Shoham's first-order temporal logic (1988) the statement

Holds(fuel-status(AC#6,LOW),T1,T2)

indicates that at the interval <T1,T2> AC#6 is low on fuel. Since fuel needs to be replenished, it is possible to express this fact with the assertion

∀a l1 l2 t1 t4
[Holds(fuel-status(a,l1),t1,t1) ∧
~∃t2t3 t1<t2 ∧ t2<t3 ∧ t3<t4 ∧ Holds(refuel(a),t2,t3)]
→ Holds(fuel-status(a,l2),t4,t4) ∧ l2<l1,

which states that unless an aircraft is refueled, its fuel level will not increase. Note that terms such as T1, T2 etc. are abstract time points. To relate them to clock time, one could write expressions such as

∀ay t1 t2 Holds(refuel(a),t1,t2) ∧ Holds(clock-reads(x),t1,t1)
→ Holds(clock-reads(y),t2,t2) ∧ y>plus(x,12).

Which is intended to mean it takes at least 12 minutes to refuel.

As this example illustrates, temporal logics are considerably more expressive than the situation calculus. However, the basic mechanisms of deduction remain the same. Consequently, the strategy for generating plans is still one of applying a theorem prover to try to prove that a situation, time point or interval exists that satisfies a set of desired conditions.

5.3 Fundamental Problems with Formal Logics for Planning

Consider the following data base:

DB = { $\forall$oxys Holds(type(o,CAR),s) $\wedge$ Holds(at(o,x),s)
$\rightarrow$ Holds(at(o,y),result(drive(x,y,s)))
Holds(type(#337,CAR),S1)
Holds(at(#337,Home-loc),S1) }.

The first sentence asserts that driving a car from location x to location y will result in that car being in location y. The second sentence asserts that in situation S1, the object #337 is a CAR. Although apparently simple, these sentences can be very problematic for a formal reasoning system.

The first sentence is an overstatement. There are in fact an infinite number of things that could occur that would prevent the car from arriving at location y: a flat tire, engine trouble, an earth quake, etc. Although none of these things are likely to happen, any one of them *may* happen. If one does, then the formal reasoning system may find itself with an inconsistent deduction. From DB, for instance, FOPC would deduce the sentence

Holds(at(#337,Work-loc),result(drive(#337,Home-loc,Work-loc),S1))

even if it received direct data asserting

~Holds(at(#337,Work-loc),result(drive(#337,Home-loc,Work-loc),S1)).

To avoid this problem is to necessary to somehow *qualify* the first sentence in DB by inserting extra conditions. For example,

$\forall$oxys
[Holds(type(o,CAR),s) $\wedge$ Holds(at(o,x),s)
$\wedge$ normal(at(o,y),drive(o,x,y),s)]
$\rightarrow$ Holds(at(o,y),result(drive(o,x,y,s))).

[~Holds(flat-tire(o),s)
$\wedge$ ~Holds(engine-trouble(o),s)
$\wedge$ ~Holds(earth-quake,s)
$\wedge$ ....
$\rightarrow$ normal(at(o,y),drive(o,x,y),s)].

Unfortunately, qualified sentences of this form are useless. They require that an infinite number of conditions be met before a deduction can be made. However, if any conditions are dropped, then the possibility of deducing inconsistent statements reappears. The *qualification problem* is the problem of both efficiently and correctly reasoning about the conditions under which an action has an intended effect.

The second sentence Type(#337,Car,S1) suffers a similar problem. Consider the action of drive(#337,Home-loc,Work-loc,S1). Clearly, the act driving a car from home to work has no impact on the fact that #337 is still a car. However, in FOPC it is not possible to deduce that the act of driving does no have this impact. Consequently, it is unable to deduce

Holds(type(#337,Car),result(drive(#337,Home-loc,Work-loc,S1)).

This illustrates a second problem, namely that FOPC can only deduce that an action does not affect a fluent if it can explicitly prove that no change has occurred. Consequently, to be complete the situation calculus must include a series of *frame axioms* that specify the relevant non-consequences of each type of action. Obviously, for reasonable size problems this is infeasible. The *frame problem* is the problem of finding an efficient and correct mechanism for reasoning about the non-consequences of an action.[7]

5.4 Logics for Non-Monotonic Reasoning

Among the logicists in AI, it is believed that if a logic could be developed that solves the qualification and frame problem while retaining

[7]See Brown (1987) for some other formulations of the qualification and frame problem.

the desirable properties of FOPC (generality, soundness and completeness), then a truly general and effective formal logic for automated reasoning will have been developed. Such a logic, in turn could be applied to planning problems in the same way as FOPC can. Logics for *non-monotonic reasoning* are designed to meet this objective.

FOPC and the situation calculus are monotonic logics. Once a deduction is made the logic provides no facility for retracting that deduction. Consequently, the set of deductions must increase monotonically. A logic for non-monotonic reasoning, however, can retract previous deduction in the face of new evidence. Consequently the set of conclusions is non-monotonic -- sometimes increasing, sometimes decreasing. The ability to retract conclusions allows the logic to "jump to conclusions" that are stronger than pure deduction allows without risking later inconsistencies.

A variety of logics for non-monotonic reasoning have been developed (for review see Reiter, 1987). It is beyond the scope of this report to attempt to review even a subset of them. Instead, we provide a simple example below and then note some of the problems with these logics.

5.4.1 An Example from Default Logic.

Default logic (Reiter, 1980) is an extended logic that adds to FOPC a set of default rules of the form A:B|-dB. This rule states that if A can be deduced, and as long as it is consistent to deduce B then deduce B. Using default logic we can modify DB to be

DB = { ∀oxys [Hold(type(o,CAR),s)
∧ Hold(at(o,x),s)
∧ Normal(at(o,y),drive(o,x,y),s)]
→ [Hold(at(o,y),result(drive(o,x,y),s)))
∧ ~Persist(at(o,x),drive(o,x,y),s)]
∀xas Hold(x,s) ∧ Persist(x,a,s) → Hold(x,result(a,s)
Holds(type(#337,CAR),S1)
Holds(at(#337,Home-loc),S1) },

and add the default rules

DF = { :Normal(f,a,s)|-dNormal(f,a,s),
Holds(f,s):Persist(f,a,s))|-dpersist(f,a,s) }.

The first default rule asserts that as long as it is consistent to deduce that an action is normal with respect to a fluent, then this deduction can be made. Similar, the second rule asserts that if a fluent holds in a situation, and it is consistent to deduce that it will persist after the action is performed, then that deduction can be made.

From DB and DF an automated reasoning system can now deduce:

	DEDUCTION	SOURCE
D1	Normal(drive(#337,Home-loc,Work-loc),S1)	DF1
D2	Holds(at(#337,Work-loc), result(drive(#337,Home-loc,Work-loc),S1))	DB, DF1
D3	Holds(type(#337,Car), result(drive(#337,Home-loc,Work-loc,S1))	DB2, DF2

Now, if the reasoning system later learns

~Holds(at(#337,Work-loc),result(drive(#337,Home-loc,Work-loc),S1)),

then the consistency criterion in DF1 is no longer satisfied. Consequently, DF1 can not be used to deduce D2. In effect the deduction D2 has been retracted. This illustrates how a logic for non-monotonic reasoning can be used to jump to conclusions that can later retracted.

Sometimes the default deductions lead to multiple extensions. Each extension includes a consistent set of default deductions, but the extensions are inconsistent with each other. For instance, the database {A,C} and set of defaults {A:B|-dB, C:~B|-d~B), has two extensions. The first contains the default deduction B, while the second contains the default deduction ~B.

5.4.2 Problems with Logics for Non-Monotonic Reasoning

Unfortunately, a satisfactory logic for non-monotonic reasoning has yet to be developed. In one form or another, all of these logics are subject to a general problem called the *anomalous extension problem*. Anomalous extensions occur whenever the default reasoning procedures generate deductions that were not intended by the knowledge engineer or fail to

generate deductions that were intended. For instance, the database and default rules section 5.4.1 could not be used to deduce

```
Holds(at(#337,Work-loc),
      result(drive(#337,Home-loc,Work-loc),result(do-nothing,S1))).
```

This is because there is an extension that contains from the assertion

```
~persist(at(#337,Home-loc),result(do-nothing,S1)),  and
```

```
~Holds(at(#337,Work-loc),
      result(drive(#337,Home-loc,Work-loc),result(do-nothing,S1)))
```

thereby preventing DF2 from being applied. Consequently, the obvious conclusion that do nothing does not change the location of an object can not be deduced.

Although problems such as this can be individually repaired by careful knowledge engineering a satisfactory general solution to these problems has yet to emerge.

5.5 Possible Worlds Planning

Before closing this chapter, we should mention a system that merges the FOPC approach with the planning as heuristic search paradigm. This is Ginsburg's (1987) *possible worlds* planner. It works roughly as follows. Initially, the planning domain is described in terms of a set of domain axioms. Operators are defined in terms of a set of preconditions and an add list. For any given state description, the description of the next state is determined by using logical deduction to remove from the initial state description all propositions that are logically inconsistent with the propositions on the add list. Similarly, the distance of a state description from the goal state is estimated by logically deducing the number of propositions that must be removed from the state description in order to be consistent with the goal state. Planning is then performed by using A* to search through the space of state descriptions where h* is based on this distance estimate.

5.6 Relevance to Associate Technology

5.6.1 Plan Generation and Modification

The logicist objective is to build a general logic-based approach to automated reasoning where inference, decision making, planning, etc. are all deductive activities that are based on a set of first-principles. If ever developed, an effective first principles reasoning system would have many advantages. The most important advantage is that many software and knowledge engineering problems disappear. It would no longer be necessary to carefully engineer a knowledge base for each domain. Rather, the knowledge engineer would only need to specify a set of axioms that describe the properties of the domain. Plan generation and plan modification problems could then be solved by executing a general logic-based reasoning mechanism.

Unfortunately, a general purpose logic-based reasoning system remains an ideal. Current systems do not perform well enough to serve as realistic planners. The logic-based approach to planning bears watching, but near term applications planning are unlikely.

5.6.2 Real-time Planning

At first look the first principles approach appears incompatible with real time planning. Theorem provers are not generally noted for their efficiency even when they are tailored to certain kinds of deductions. Furthermore, expressive logics are inevitably undecidable. Therefore, a useful first principals planner cannot be guaranteed to terminate in finite time, much less "in time." However, the picture is not really as bleak as it looks. Recent work (Ginsburg, 1990; Lehner, 1991; Horvitz, 1991) has shown that it may be possible to have a theorem prover generate intermediate results that are usually correct. Consequently, it is possible to interrupt the automated reasoning process to request the "best answer so far." Once developed, such a capability would make it possible for a first principles planner to support anytime planning. This work is very exploratory, but bears watching.

CHAPTER 6

PLANNING AND REACTING: ARCHITECTURES FOR PLANNING IN REAL-TIME

Associate systems are designed to provide real time planning and decision support. Unfortunately, AI research in automated planning has traditionally focused on the deliberation aspect of planning. Relatively little attention was paid to the issue of real time (i.e., in time) reactivity. In the last few years, however, this orientation has changed. There is an emerging body of research directly addressing the issue of real-time planning.

Provided below is a brief overview of some of the principal architectures that have been proposed for supporting the need to plan deliberation and the need to react in real time. Please note that this is still an immature research area and stable paradigms or "schools of thought" in this area are still in transition.

6.1 Situated Activity/Universal Planning

Loosely characterized, this approach emphasizes the reactive element of intelligent behavior to the point of suggesting that deliberate/"look ahead" planning is a largely irrelevant aspect of intelligent behavior (Agre and Chapman, 1987; Brooks, 1991; Kaelbling and Rosenschein 1990). This approach is based on the belief that a well-designed set of situation-action rules can lead to a sequence of actions that appear to be part of a coherent plan, even though each action was the result of applying an individual rule. Since the agent's behavior is being directed by simple situation-action rules, rather than time-consuming plan generation, rapid reactions are guaranteed.

To illustrate, consider Figure 6-1. The robot knows that the goal location is the corner marked with an X. Its behavior is guided by two rules. First, if an object is encountered, move around the object in the general direction of the goal location. Second, if the object is in sight, move directly toward the object. A path that could result from these two rules is shown in Figure 6-1. Although the path itself looks like it may have been the product of careful planning, it in fact was simple the result of the application of two simple rules.

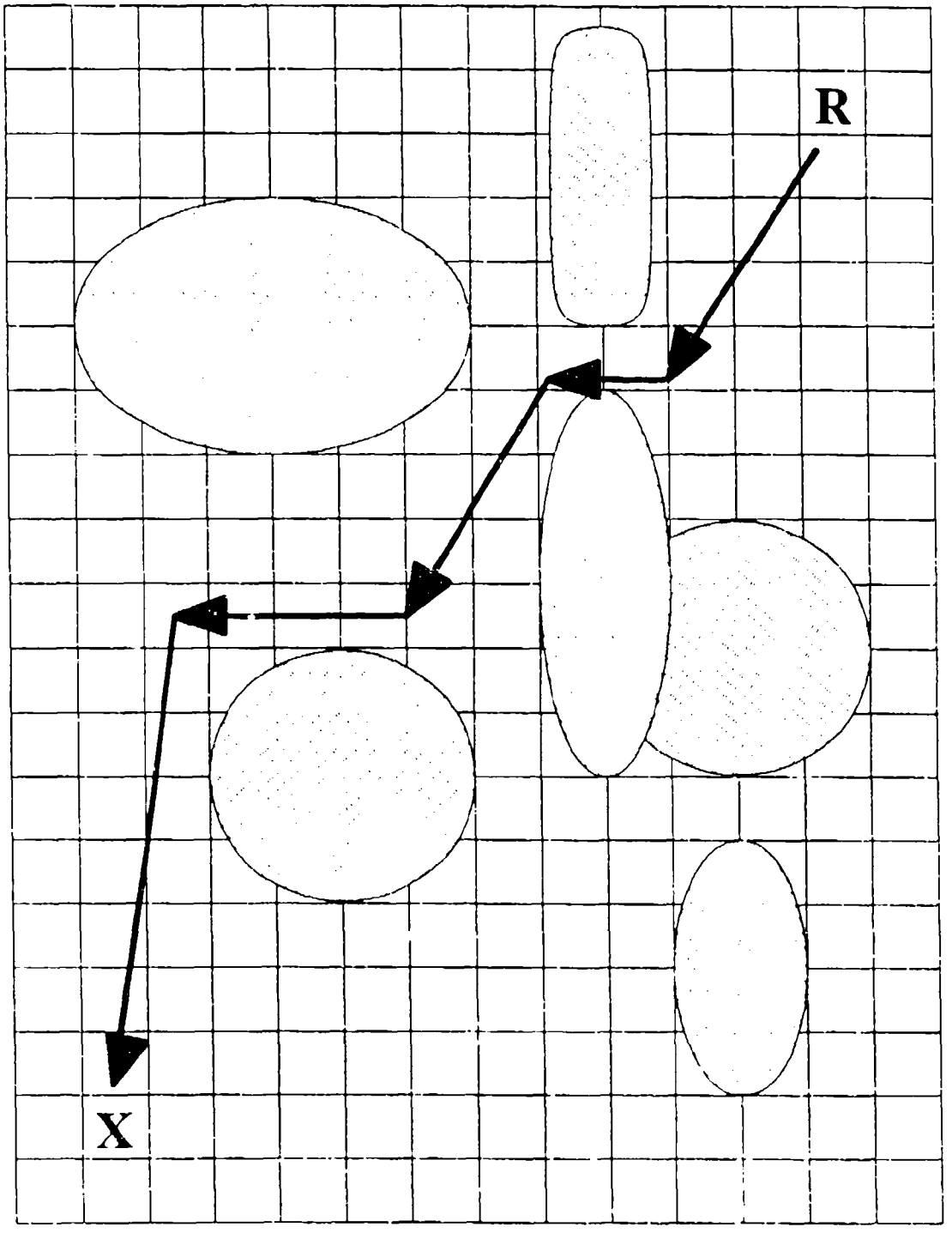


Figure 6-1: A Reasonable "Plan" from a Reactive Robot

When first presented, this viewpoint had a significant impact on the automated planning community. It presented an approach that was in sharp contrast to the traditional, deliberation-based viewpoint. However, these contrasting views evolved toward a more common perspective -- that intelligent behavior requires a balanced combination of planning and reacting. Many of the architectures presented below represent proposals for achieving such a balance.

6.2 Layers of Planning

This approach recommends the use of multiple simultaneous planning layers. Time-consuming planning and deliberation occurs at the "higher" levels, while lower level procedures are designed for rapid action generation (Kaelbling, 1987). As each planning layer completes its processing, the actions it recommends are posted on a blackboard. As the situation evolves, the system checks the blackboard continually for the highest level actions that have been posted. If the situation evolves slowly and is relatively stable, then there will usually be sufficient time for a higher level planner to complete its deliberation and the agents behavior will be guided by an explicit plan. On the other hand, if the situation is dynamic, with unexpected events occurring often, then the actions posted by the lower level planners will be the only ones on the blackboard. Consequently, in such situations the agents behavior will be largely reactive.

6.3 Scheduling, Planning, Reacting, and Control Activities

The previous paradigm strategy assumes that different levels of planning can occur in parallel. Even if feasible, this approach may represent a waste of computational resources. An alternative approach would be to dynamically schedule the extent to which the system engages deliberate vs. reactive planning (e.g., Hendler and Agrawala, 1990). In rapidly changing situations, the scheduler would assign a higher priority (more computational resources) to reactive planning, thereby reducing reaction times. Similarly, in slowly evolving situations, more computation would be applied to explicit planning.

6.4 Decision Theoretic Control of Planning

In the last few years, decision theory has regained some popularity in the AI community. The distinction between symbolic problem solving and numeric reasoning has largely disappeared. With regard to the deliberation vs. action tradeoff decision theory methods have been proposed for explicitly controlling the scheduling of these activities. The general idea here is that the reasoning system always maintain a list of options with calculated expected utility (EU). Initially, EU is calculated by employing a decision model that considers just a few factors. As time permits, the decision model can be expanded to consider additional factors. One of the interesting features of this approach is that the decision theoretic calculations can be used to calculate the expected utility of considering additional factors (Henrion, 1991).[1]

6.5 Planning for Reaction

In AI, a plan has traditionally been viewed as a program for action that can be loaded and executed by an execution system. An alternative view is to use the product of deliberate planning (which may not be an explicit plan) as a guide for a reactive system. The are a number of variations on this idea. For instance, Martin and Allen (1990) take the approach that a plan can serve as a set of instructions for a reactive system, but that the instructions set may vary in its level of detail. Consequently, the instruction set may be very abstract, providing general guidance to the reactive system. More extreme is the approach promoted by Payton (1990), suggesting that the product of deliberate planning not be a plan, but a set local rules that, if followed, will lead to behavior that seems to conform to a plan.[2]

6.6 Anytime Problem Solving

Another approach to real time reactivity is to develop planning systems that can be interrupted with a request to produce the "best answer so far." Such a planner would be an instance of an anytime

---

[1]This is different than expected value of information (EVOI). In EVOI, the current decision model is used to determine whether additional information (e.g., sensor test) should be obtained. In the case of expanding the decision model, the decision needs to be made as to whether the current decision model should be expanded. This decision must be made outside the current decision model.

[2]Payton's gradient field can be viewed as a set of local rules.

problem solver (Dean and Boddy, 1988). Slagle and Hamburger (1985) achieve anytime behavior in their expert system for artillery fire allocation by quickly generating an initial solution, and then iteratively modifying that solution as time permits. Boddy (1991) suggests that anytime behavior can be achieved by maintaining two problem solvers. A complex problem solver is used as long as time is available. When time is short, a simple and quick problem solver is invoked to complete any partial plan generated by the more complex problem solver. Ginsburg (1990) suggests that declarative nonmonotonic logics can be used to support anytime problem solving, because such logics can be designed to quickly jump to default conclusions that may be retracted later as a result of additional deductions.[3], Lehner (1991) and Horvitz (1991) have noted that probabilistic reasoning can be used to convert many symbolic problem solving algorithms into anytime problem solvers. This approach is discussed in Section 7.2. finally, it should be noted that transformational planners (see section 4.0) are generally adaptable to anytime problem solving, since these planners operate by initially invoking and then debugging plans. A complete, current plan is always available, albeit it may be a faulty one.

6.7 Relevance to Pilots Associate

The relevance here is obvious. One of the key elements of associate technolgy systems is that of providing real time planning support. Consequently, any architecture that merges planning and reai time behavior should be of interest to developers of associate systems.

---

[3]The interesting feature in Ginsburg's approach is that the default conclusions may be retracted as a result of new deductions made from a stable data base. This is different than most nonmonotonic logics where new information is required to retract a default conclusion.

CHAPTER 7

AUTOMATED PLANNING AND MATHEMATICAL OPTIMIZATION

Mathematical optimization (also called mathematical programming) is an area of Operations Research oriented toward the development of automated procedures that find optimal solutions to a variety of problems. The objective of a mathematical optimization procedure is to find optimal solutions to well-defined and mathematical representable problems. Typically, a mathematical programming procedure decomposes a problem into two components: an objective function and a constraint space. The objective function provides a measure of merit by which proposed solutions are rated. The constraint space defines the set of acceptable solutions. The goal is to find a solution that is optimal with respect to the objective function, but is consistent with the constraint space.

For instance, in a *linear programming* problem the objective function is defined as a linear expression while the constraint space is defined as a set of linear equalities and inequalities. For instance,

Minimize	c1 + .4c2
Subject to	c1 - 2x1 = 0
	c2 - x1 = 0
	x1 > 10
	c1 > 0
	c2 > 0.

In general, optimization procedures can be characterized in terms of a *power/generality tradeoff*. A problem solving technique is *general* to the extent that it can be applied to a diversity of problems. It is *powerful* to the extent that, when applied, it generates a good answer. Linear programming techniques are very powerful. Given that a problem can be represented as a linear program, efficient procedures exist for finding globally optimal solution -- the best solution found in the constraint space. Linear programming is an example of a very powerful technique that has little generality. As expressions defining the objective function and constraint space become less constrained (e.g., allow nonlinear expressions, discrete variables, lexicographic relations, etc.) generality increases, but power decreases. These more general procedures are not guaranteed to find globally optimal solutions, and begin look instead for *locally optimal* solutions.

The introduction of AI technology can be viewed as an extension of this power/generality tradeoff. AI problem solvers tend toward the extreme of the more general/less power end of this tradeoff. Most AI planning system employ some form of satisficing strategy where the system searches until it finds *any* solution that satisfies a set of constraints. There is usually no explicit attempt to find a plan that scores high on some measure of merit. On the other hand, AI planning systems are very general. This is because much of the work in AI is oriented toward developing flexible knowledge representation schemes where it is possible to express any knowledge relevant to the problem solving domain.[1]

Associate systems need to provide real time planning support. Consequently, it is desirable to have systems that score high on both power and generality. Power is needed simply because a system that does not reliably generate good advice should be ignored (Lehner, 1989). Clearly, optimization procedures are relevant here. On the other hand, it is hard to circumscribe a priori the types of constraints that may be relevant to a problem. A mission planning system that is designed to find an optimal path may be faced with a problem involving the coordination of two or more missions, diversionary legs on a route, etc. Such constraints may not be easy to represent in the constraint language of a powerful optimization system. Flexible knowledge representation schemes, on the other hand, have little difficulty with such constraints. As depicted in Figure 7-1, associate systems should have embedded planning techniques that place it well above the current power/generality curve.

7.1 Breaking the Power/Generality Tradeoff

One way to break the power/generality tradeoff is to engineer systems that in-corporate both heuristic and optimization-based problem solving methods. One architecture for achieving this is shown in Figure 7-2. The basic idea is that the problem solving activities of a heuristic problem solver can be guided by the outputs of an optimization procedure.

---

[1]It could be argued that knowledge-based problem solving does not fall into this category, since effective knowledge engineering should lead the problem solver to generate good solutions. However this argument doesn't hold for the reason that knowledge-based procedures have no mechanism for determining if a solution found is "good." Consequently, there is no mechanism that guarantees avoiding worst case results.

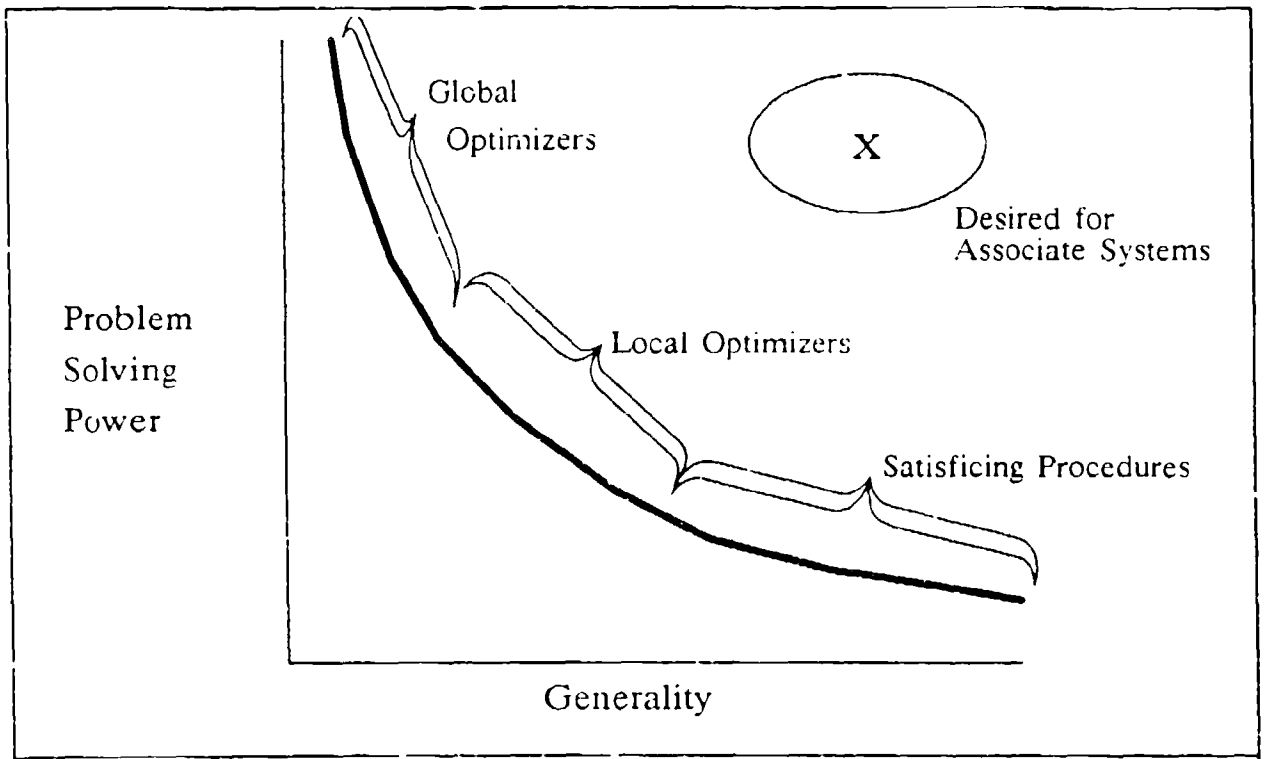


Figure 7-1: Power/Generality Tradeoff

Initially a partial set of constraints is submitted to the Optimizer which attempts to find an optimal solution for the partially constrained problem. The principal output of the optimizer is an *idealistic solution.* It is a solution that scores well on the measure of merit, but may not satisfy all constraints.

The heuristic problem solver is designed to generate realistic solutions that satisfy all relevant constraints. As noted earlier, by itself there is no way to guarantee that the heuristic problem solver will generate a solution that scores well on the measure of merit. There is always the possibility that a substantially better solution exists.

In our approach we provide the heuristic problem solver with an additional input - the idealistic solution generated by the Optimizer. The heuristic problem solver is then given the task of finding a *realistic solution* that achieves some percentage (say 90%) of the measure of merit value of the idealistic solution -- a high-valued, realistic solution. If such a solution cannot be found, then the heuristic problem solver will need to generate additional constraints that can be added to the partial constraint list of the Optimizer. With these additional constraints, the optimizer is rerun and a new idealistic solution is generated.

Although a variety of problem solving methods can be embedded in the heuristic problem solver, the most natural approach would be one based on the plan transformation paradigm. The idealistic solution would serve as the initial plan. It would undergo a series of transformations until a plan emerged that satisfied all constraints. If a satisfactory plan does not emerge, then the heuristic problem solver would identify some steps in the plan that appear to be essential. These steps define a partially specified plan that becomes the additional constraints to be submitted to the optimizer.

7.2 Relevance to Associate Technology

The application of optimization techniques to mission planning problems has been an active area of investigation for more than thirty years. A number of mission planning systems already exist that use these techniques. Unfortunately, these systems are subject to the difficulties we noted above. Specifically, these systems only consider some of the constraints that are relevant to mission planning. For instance, the Lockheed planner in the Pilot's Associate is an optimization system that optimizes by minimizing fuel expenditure and site visibility. It does not

consider path vulnerability (visibility is a surrogate), timing factors and constraints, and coordination with other missions.

Combining optimization and AI-heuristic reasoning techniques in the manner described above is one way to overcome the current limitation of optimization-based mission planning systems. This will be discussed further in Chapter 10.

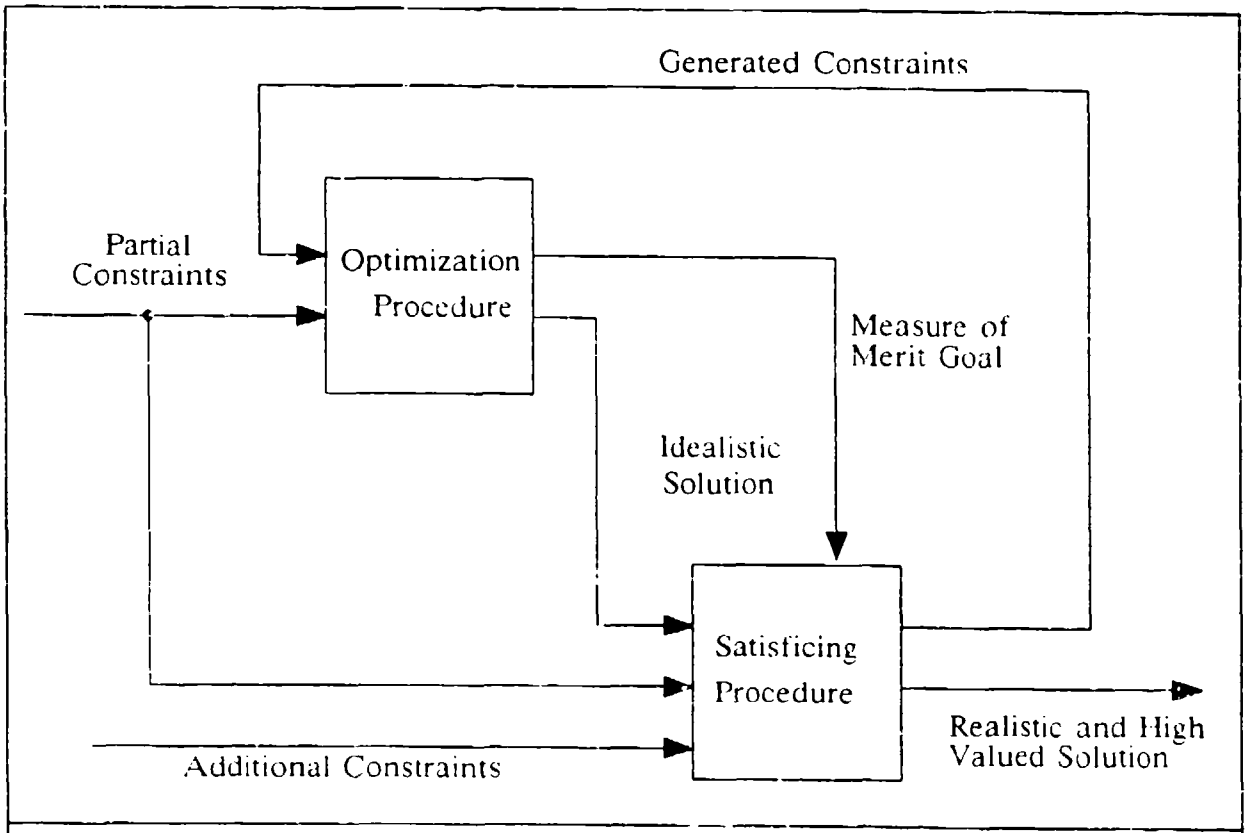


Figure 7-2: Architecture for Merging Optimization and Satisficing Procedures

CHAPTER 8

AUTOMATED PLANNING, UNCERTAINTY HANDLING AND DECISION THEORY

In this chapter we examine the relationship between automated planning in AI and the problem of uncertainty management. Many problem domains for which associate technology systems have been proposed are inherently probabilistic and the uncertainties associated with the various state variables and action outcomes need to be considered in the planning process. Here we examine this relationship from two perspectives. First, we look at the problem of incorporating uncertainty management into the planning process. Second, we examine the possible application of uncertainty management techniques to the control of the planning process.

In the AI community, there are two competing approaches to the problem of handling uncertainty. The first approach suggests that uncertainty is a matter of degree, and that uncertainty handling is a problem of calculating belief values. Among the strongest proponents of this approach are the Bayesians, who argue that a rational system for behaving in uncertain situations must act as though it maintains a set of beliefs values that conform to the probability calculus. The second approach argues that uncertainty handling is a matter of making and revising assumptions. At all times behavior conforms to a set of assumptions about the current world state that need to be revised if evidence surfaces that is contrary to the current assumption set.

8.1 Uncertainty Management with a Quantitative Belief Calculus

A variety of calculi have been proposed as a basis for calculating belief values. In this section we will examine only one of these - the Bayesian approach. Others may have their merits, but the main points to be made here can be done so by examining just the one approach.

Bayesianism is a school of though that argues that a rational system of belief values must conform to the probability calculus, and that upon learning new information (evidence), belief values should be updated using Bayes rule. This rule asserts that given two Hypotheses (H1 and H2) and some evidence item (E), then the relative belief given E should be calculated using the rule:

$$\frac{B(H1|E)}{B(H2|E)} = \frac{B(E|H1)*B(H1)}{B(E|H2)*B(H2)}.$$

It is important to note that there is nothing in the Bayesian approach that limits its use to situation where the true probabilities are known. Indeed many proponents of the Bayesian approach would argue that the notion of a "true" probability is itself spurious. The statement "The probability that the coin will land Heads is .5" is a statement expressing our degree of belief that the coin will land Heads or Tails. The coin itself will either land Heads or Tails.

When applied to planning problems, belief values are merged with outcome utilities so that choices that maximize expected utility can be made. To illustrate how this works, consider the simple route planning problem depicted in Figure 8-1. A robot, initially at x,y coordinate (6,11) must find a path to the refueling cell at (5,5). At the refueling cell is one hundred units of energy. It cost one unit of energy to move into a blank cell, 200 units of energy to move into a dark cell. The cost of moving into a cell marked with a ? is either 1 or 200 energy units. The robots objective is to maximize its energy store. Finally, when a robot is next to a ? cell, it can use a sensor to test whether or not the ? cell is a blank cell. When the sensor is working reliably its reports are completely accurate. When the sensor is not working, it reports either "Clear" or "Dark" about equally often, independent of whether the ? cell is truly Clear or Dark. The cost of a sensor test is 20 energy units.

Now we include some belief values. Of the 100 non ? cells, 40 are dark, so we initially assert B(Clear|Cell is type ?) = .6. The sensor, because of prior testing, operates reliably about 80% of the time.

We now examine the plan proposed in Figure 8-2. The lower portion lays out the possible events sequences that may occur if this plans is executed. Each complete path on this tree is called a chronicle (Hanks, 1990). The first chronicle asserts that the robot will go several steps to coordinate (6,6), then to coordinate (6,5) which will be a blank square, and then to coordinate (5,5). Chronicle two is the same except that (6,5) is dark. The expected utility of this plan is calculated by multiplying the probability of each path times its cost. For Plan 1, this gives us

EU(PLAN 1) = .6*(100-7) + .4*(100-206) = 13.4.

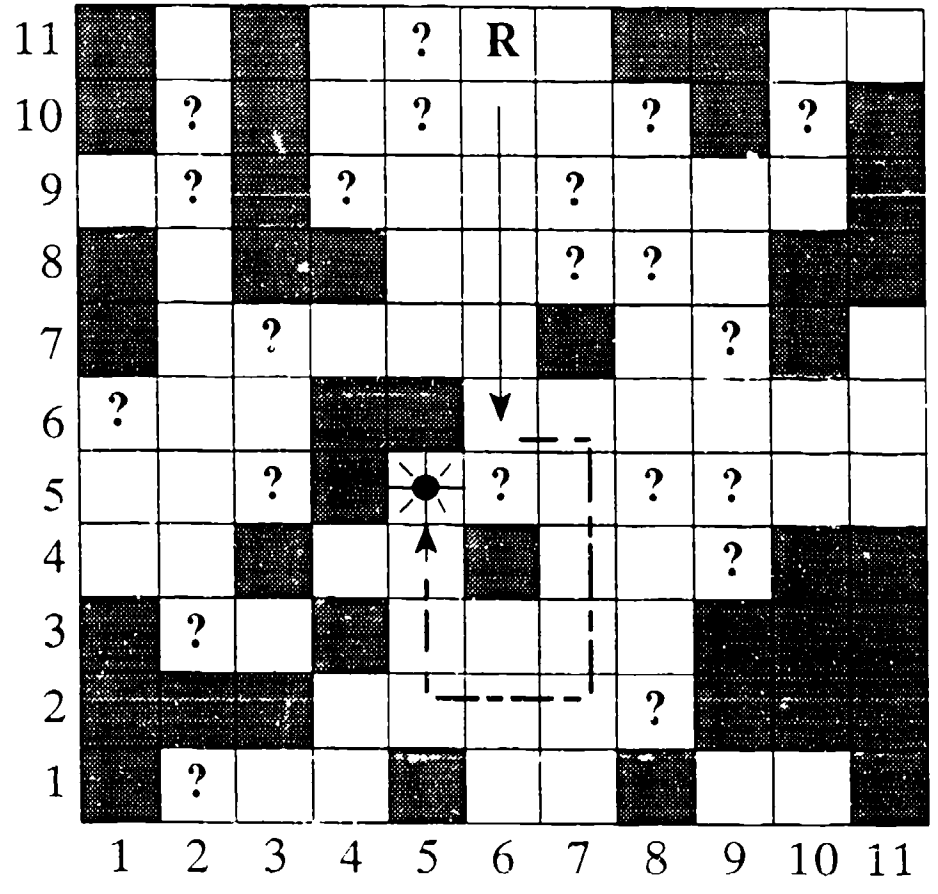


Figure 8-1: Probabilistic Planning Problem

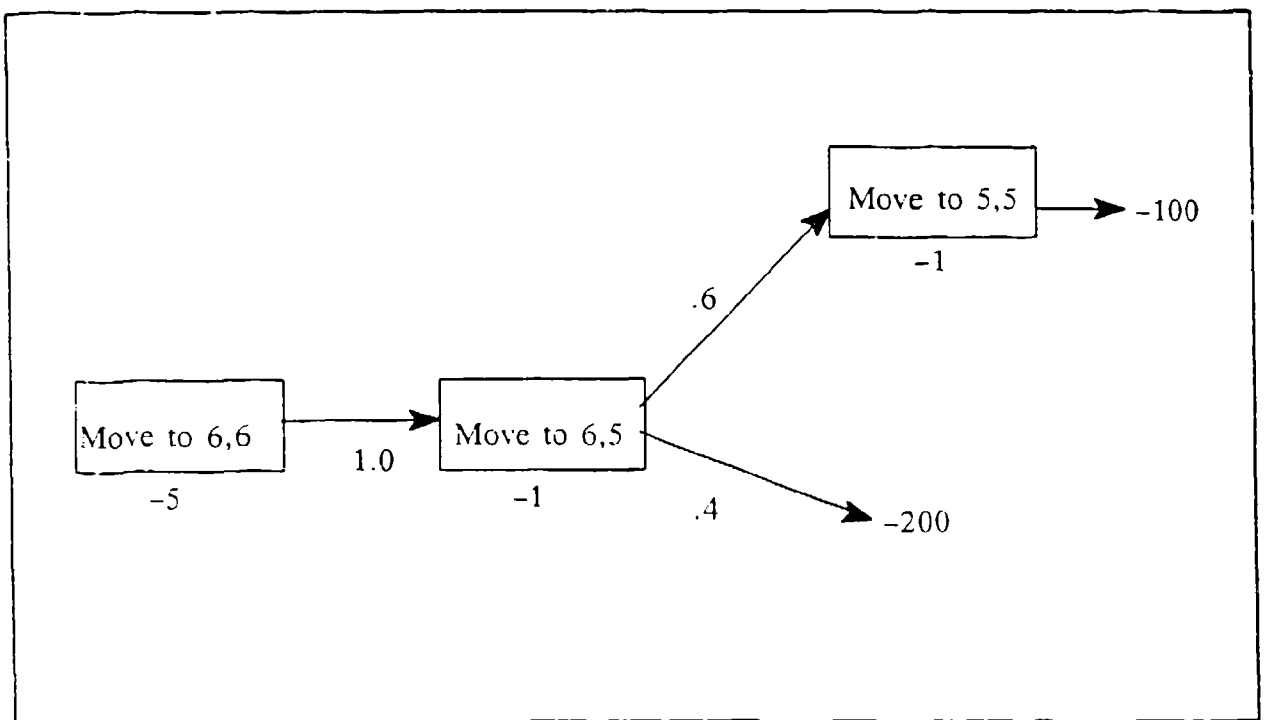


Figure 8-2: First Chronicle for Robot Planning Problem

An alternative plan is to go to (6,5), get a sensor reading on (6,6) and then go around if the sensor reports "Dark". Figure 8-3 lays out this plan, along with the probability that each step will occur. The probability that a sensor reading will report back "Clear" is calculated as follows:

B("Clear") =
B("Clear"|Clear & Reliable)*B(Clear|Reliable)*B(Reliable)
+ B("Clear"|Dark & Reliable)*B(Dark|Reliable)*B(Reliable)
+ B("Clear"|Clear & ~Reliable)*B(Clear|~Reliable)*B(~Reliable)
+ B("Clear"|Dark & ~Reliable)*B(Dark|~Reliable)*B(~Reliable)

= 1.0*.6*.8 + 0.0*.4*.8 + .5*.6*.2 + .5*.4*.2

= .58.

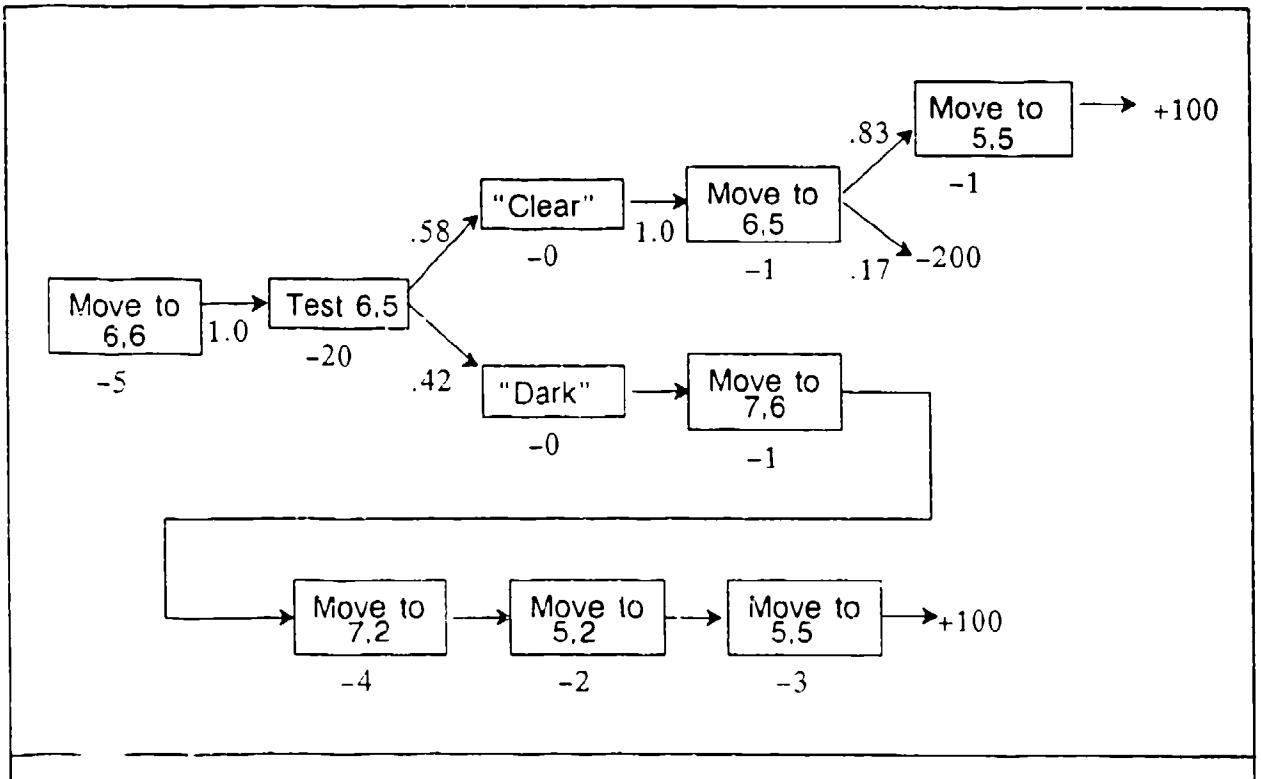


Figure 8-3: Second Chronicle for Robot Planning Problem

The probability that (6,5) is in fact Clear, given that the sensor reported "Clear" is calculate using Bayes rule:

B(Clear|"Clear") =

B("Clear"|Clear)*B(Clear)/B("Clear") =

$$\frac{[B(\text{"Clear"}|\text{Clear} \ \& \ \text{Reliable})*B(\text{Reliable}|\text{Clear}) + B(\text{"Clear"}|\text{Clear} \ \& \sim \text{Reliable})*B(\sim\text{Reliable}|\text{Clear})] * B(\text{Clear})}{B(\text{"Clear"})}$$

[1.0*.8 + 0.0*.2]*.6/.58 = .83.

This gives us B(Clear|"Clear") = .83. Plan 2 has three chronicles with an expected value of 84.3. This, in turn. indicates that the expected value of incorporating the sensor test into the plan is 40.2 - 13.4 = 26.8.

This plan can be further elaborated by adding in a second sensor test if the first test indicates "Clear". As shown in Figure 8-4 this plan has an expected value of 38.5. A second sensor test is more costly than the expected gain.

As this example indicates, probability calculations can be quite useful in determining what steps to add to a plan, and for calculating the expected benefits and costs of each step.

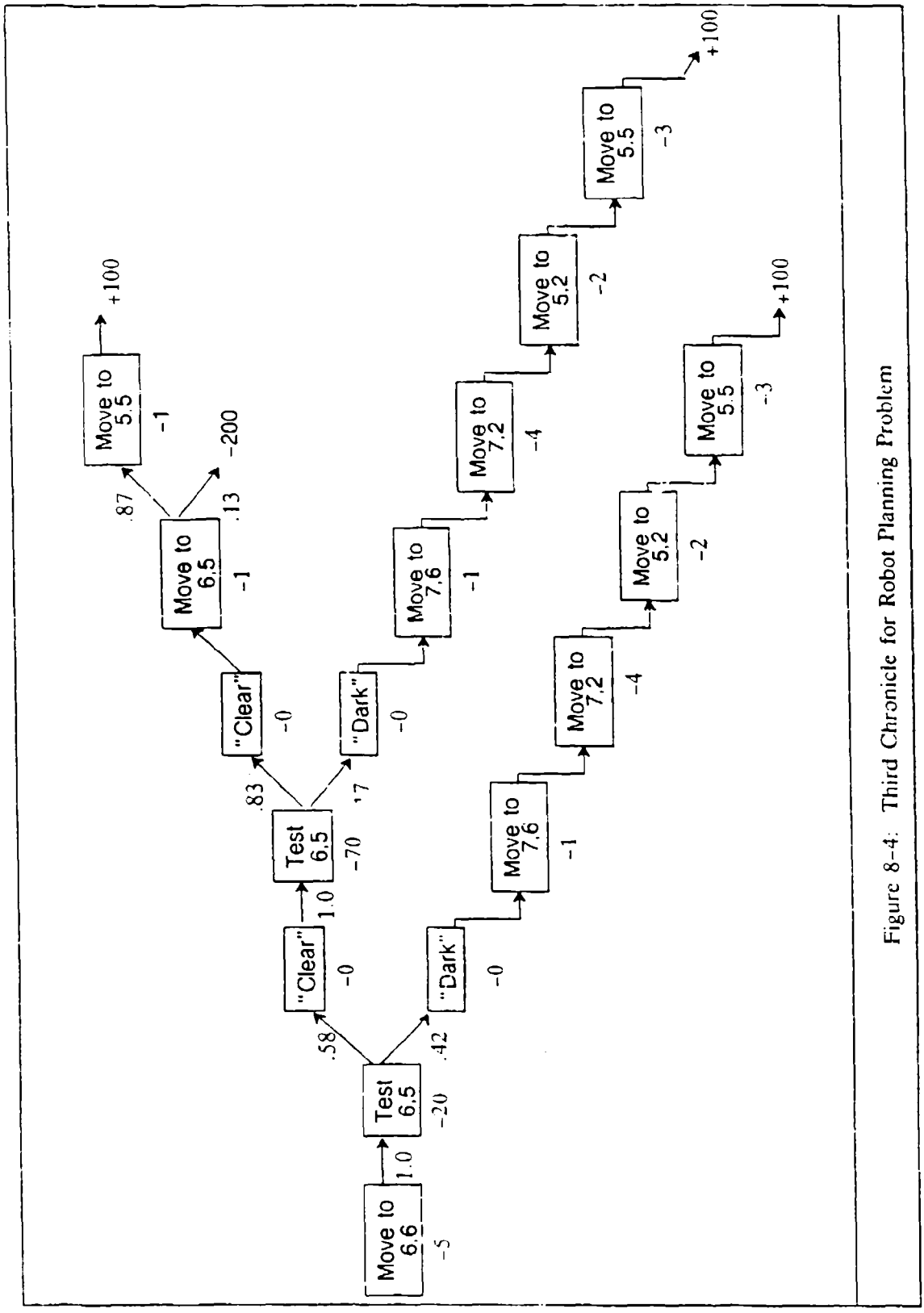


Figure 8-4: Third Chronicle for Robot Planning Problem

8.2 Applications of Decision Theory to Automated Planning

In the last few years, there has emerged a resurgence of interest in the application of decision theory to automated problem solving. Below we briefly note some directions this work is taking.

Decision Theory-based Decisions - The most direct application is to use expected utility calculations to select actions. This was illustrated in section 8.1. Unfortunately, this approach may be difficult to estimate since it may require an exorbitant number of probability assessments. One approach to overcoming this problem is to implement a qualitative inference system that conforms to the probability calculus (Wellman, 1990). Alternatively, one can develop procedures for constructing small, problem-specific decision models (e.g., Laskey, 1990; Laskey and Lehner, 1991;, Peot and Breese, 1991). Reasoning *within* these models involves explicit probability and utility calculations. However, reasoning *about* these models may be heuristic.

Probabilistic Domain Models - Probabilistic models have been proposed as an alternative to the simplistic action models embedded in STRIPS-like planners. These probabilistic models can then be used to reason about the probability of alternative futures (e.g., Dean and Kanazawa, 1989;). Much of the research in this area is focused on efficient probabilistic projection, where computation is focused only on relevant and probable futures (e.g., Hanks, 1990)

Rationalization of Heuristic Procedures - Automated planning research is traditionally based on symbolic reasoning techniques. This has made it difficult to relate this research to decision theoretic notions of planning and action selections. Recently, several researchers have attempted to provide a general mapping between concepts relevant to the symbolic approach and decision theory. Much of this work focuses on the relationship between symbolic goals and utilities (Haddawy and Hanks, 1990; Wellman and Doyle; 1991).

Decision Theoretic Control - Here the objective is to use decision theory to select which planning steps to execute next. The VOI example provided in section 8.1 is an example of this form of control. In particular, it is an example of reasoning within a decision model to make decisions as to the next best step to execute.[1] The work of Tom Dean is a good example

[1] As long as the system is reasoning from within a decision model, then decision theory does not formally distinguish actions and control decisions.

of this line of research (Dean, 1990). An alternative approach is to use probabilistic knowledge about an planning system to make control decisions about how planning should proceed (Lehner, 1991). In this approach decision theoretic reasoning is used at the metaplanning level and can be applied to controlling problem solving procedures that are not themselves decision theoretic.

8.3 Uncertainty Management using Reason Maintenance

An alternative approach to planning under uncertainty is to make an explicit set of assumptions which can be later retracted if found to be invalid. The problem of keeping track of assumptions and the deductions that depend on each assumptions is called *reason maintenance* or *truth maintenance*. Systems for tracking assumption fall into two categories. *Reason Maintenance Systems* (RMS) maintain a single, consistent set of assumptions and deductions (Doyle, 1979). In an RMS, when a new conclusion is made that contradicts a previous deduction, the RMS will identify the assumptions upon which the previous deduction depends; select one or more assumptions to retract; and remove all other deductions that depend on those assumptions. In this was, the RMS always maintains a single, consistent view of the world. *Assumption-based Truth Maintenance Systems* (ATMS) simultaneously keep track of all assumptions and deductions (DeKleer, 1986).[2] For each deduction, the ATMS will maintain a *label* that specifies them minimal assumption sets under which the deduction can be derived. Furthermore, the ATMS tracks all inconsistencies in the assumptions. As a result, an ATMS can quickly determine (1) all deductions that are justified by an arbitrary set of assumptions, and (2) all assumption sets which justify a deduction.

Although reason maintenance is a major research area in AI, it has had relatively little impact on the planning research community. This is somewhat surprising since much of the research in the area was initially motivated by a desire to solve the qualification and frame problem in automated planning. The most direct application of reason maintenance to automated planning is found in Morris (1987). In addition, temporal data

[2]We should note here that this is an inherently limited capability. Systems for reason maintenance of *propositional* systems. They simply record the sequence of deductions made by a problem solver, but have no real problem solving capability of their own. For instance, if the problem solver deduced and submitted to the reason maintenance system F(x)-->P(x), and F(A), the reason maintenance system could not deduce P(A). All it could do is *record* the fact that problem solver used the strings 'F(x)-->P(x)" and "F(A)" to *justify* P(A).

base management systems (e.g., Dean and McDermott, 1988) usually embed some form of reason maintenance.

8.4 Merging Probabilistic and Assumption-based Reasoning

For many the appropriate solution to uncertainty management is to merge the quantitative and qualitative approaches. Several ways to do this have been proposed. So far none have proven very satisfactory. One popular approach (see Laskey and Lehner, 1989; 1990) is to use an ATMS to keep track of assumptions, and then to attach probabilities to the assumptions. The belief in any set of assumptions is calculated by multiplying out the probabilities of each assumption. The belief in any proposition is calculated by summing up the beliefs of the assumption sets that imply that proposition. Unfortunately, as shown in Laskey and Lehner (1989) this approach is formally equivalent to the Shaferian calculus for belief management and is therefore subject to all of the problems associated with the Shaferian calculus (Pearl, 1990).

8.5 Relevance to Associate Technology.

8.5.1 Plan Generation

As discussed above, domains such as mission planning are generally probabilistic. Automated planning systems that address such domains should have an explicit mechanism for addressing uncertainty handling problems. In general, there seem to be three basic approaches to dealing with uncertainty.

Ignore Uncertainty - Simply ignore uncertainties, generate plans assuming a certain situation description, and replan later as necessary.

Make Assumptions Explicit - This is similar to the previous approach except that the assumptions that were used to generate a certain situation description are made explicit, recorded and tracked. In this way the conditions under which replanning is required can be identified early.

Make Uncertainties Explicit - The probabilistic reasoning approach requires that the levels of uncertainty be made explicit. Plan generation involves explicit calculation of the probabilities and uncertainties associated each action.

Each approach has its advantages. The first approach is clearly the most computationally tractable. If the domain is such that there is little cost associate with replanning, then it seems preferable. The second approach adds some computational burden, but it also makes it easier to focus replanning activities. The third approach can add a significant computational burden. However, it is the preferred approach if the domain is such that there are severe negative costs associated with "mistakes" in the plan.

8.5.2 Replanning and Real-time Planning

Incorporating uncertainty management into automated planning is still a relative new area within the automated planning community. It is not clear at this point the extent to which it will eventually promote or prevent real time planning and replanning. If plan generation includes probabilistic reasoning, then this will obviously slow things down. On the other hand, as discussed above decision theoretic procedures can be used to control the planning process, thereby focusing the planning process important planning steps. This is clearly an area that deserves further investigation.

CHAPTER 9

HARDWARE ISSUES IN AUTOMATED PLANNING

This chapter summarizes the results of an investigation into the applicability of various processor architectures to planning and, specifically, mission planning. This investigation of hardware technologies was initiated in August, 1990. The activities under this tasking included surveys of the academic and trade literature for application of high performance hardware to the planning problem or closely related applications. The examination of hardware directions also sought promising possibilities not yet being pursued for planning. Focus was on high performance engines and techniques which have been applied only to narrow applications, but which may prove applicable to planning. Specific points of departure for the search included machine intelligence, logic programming, parallel processing, and general literature on computers and component devices.

The perspective guiding the examination of hardware technology is the hierarchical nature of technologies upon which computer based problem solving is founded. Figure 9-1 shows such a hierarchy for the planning problem. Each level is an abstraction that defines the nature of a machine, such that lower levels need not be considered. For example, the abstractions in a register transfer language allow design at that level without concern for the internal gate or transistor structures internal to the registers, multiplexes, etc. of the components, at least to the first order. Likewise, a user of assembly language need know only the computer's programming model, not its microcode or internal bus structure. Higher level languages extend this hierarchy farther toward the problem domain. For planning, these may be procedural or declarative. The higher level languages are general in purpose, being used to express in executable form techniques applied specifically to the planning problem.

9.1 Planning Technology Hierarchy Perspective

The benefit of such a hierarchy is that each level of abstraction is a manageable step towards useful problem solving. Progress is made by successively adding abstractions toward the top which reduce the gap between means and ends. The cost is that, at each level of abstraction, all possible operations at the next higher level must be supported, rather than only a specialized task at hand. This carries a cost in efficiency which may

be as high as an order of magnitude per level. Research and development within each level aims at improving performance without changing the boundary specifications, reducing the cost of the level and the computational power of all levels above it. Other developments have allowed levels to be skipped or combined, thus cutting out inefficiencies related to abstraction boundaries. The Lisp machine cuts out conventional assembly language; RISC processors in effect cut out a level by combining assembly and microcode, and shifting the interface to the higher level language compiler downward.

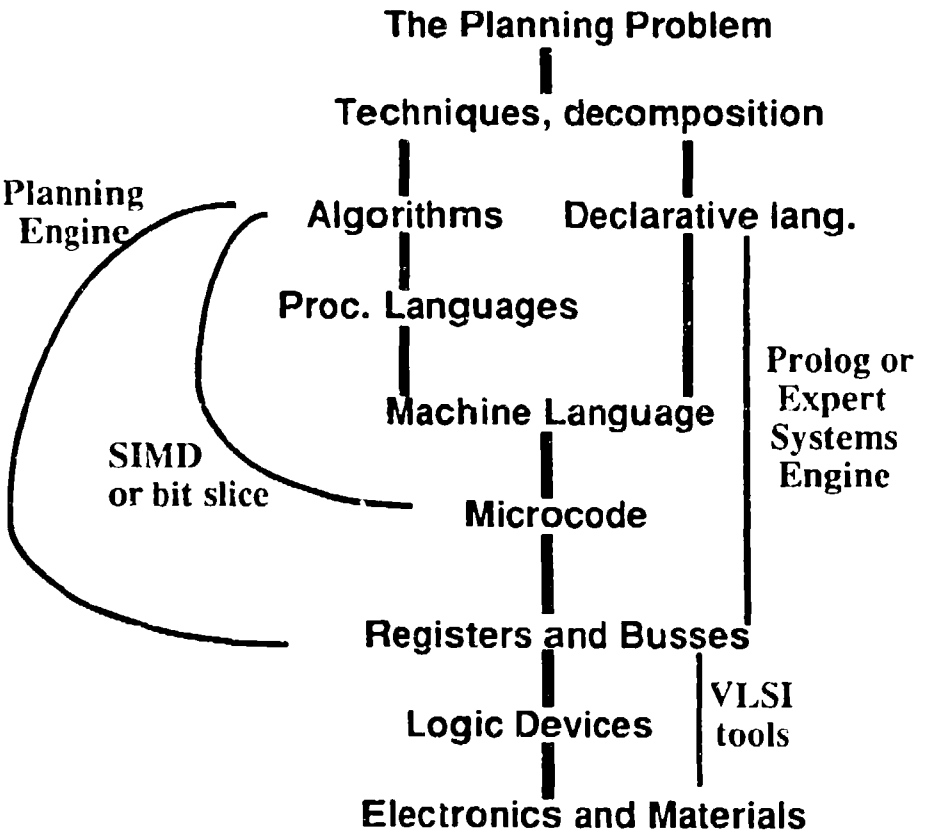


Figure 9-1 Computer Technology Hierarchy for Planning

If one can skip over one or more levels of abstraction in the diagram, considerable efficiency in speed, weight, power use, and size may be gained. A hardware device that directly implements a planning algorithm using dedicated registers and busses avoids all of the levels associated with the von Neumann computer. One may expect perhaps three orders of magnitude performance benefit, given comparable hardware technology at the register and gate level. Only one device of this sort was found, the

Lockheed Zap processor. Others may well exist but as proprietary devices may not have been published. It is also possible to gain considerable benefit by using a custom processor of the bit slice type, in effect implementing the algorithm with custom microcode.

Less planning specific applications of hardware technology include specialized engines that execute particular languages in which planning problems may be expressed. These include Lisp processors and processors dedicated to executing declarative languages. A number of Prolog and expert systems engines were noted which generally gave about an order of magnitude speedup over more conventional processors. Some specialized processors were dedicated to the support of new higher level abstractions, such as fuzzy logic.

One approach to performance not shown on the hierarchy figure is parallelism. Parallelism can be inserted at a number of different levels. SIMD (Single Instruction stream, Multiple Data stream) parallelism takes place at the microcode and possibly the register/bus levels, in that the single controller must account for interactions among the different processors. MIMD (Multiple Instruction stream, Multiple Data stream) parallelism occurs above the machine language level. The problem solver may explicitly invoke parallelism by designating it in his functional decomposition, or it may be invoked at or unseen below the language level with processors managed as resources. Parallelism allows the use of machines with much greater raw computational horsepower, but adds a level to the hierarchy. The costs of that level are both speedup limitations like bottlenecks that limit efficiency and changes in the computational model (e.g. memory access restrictions) that restrict the range of expression in the next higher level.

A bigger problem with parallelism than loss of efficiency is that for many forms of parallelism the intermediate abstractions to bridge the gap to the planning problem domain have not yet been found. Techniques for applying MIMD to planning are few, and for SIMD even fewer. The classes of SIMD that restrict communications to adjacent nodes, or MIMD machines having very tightly synchronized communications (e.g. transputer or Warp arrays) are difficult to apply to amorphous problems like planning. To date they have been applied to array organized problems. Optical machines are even more of a challenge. Yet these technologies have enormous raw potential, if the abstractions necessary to apply them can be found.

9.2 Survey Results

The survey was conducted during 1990 and 1991. Various journals and conference proceedings in relevant fields were surveyed. The findings below fall into categories illustrated in Figure 9-2. Note that some incorporate more than one element of interest, parallelism together with a specialized processor. The projects and products cited should be regarded as representative rather than exhaustive.

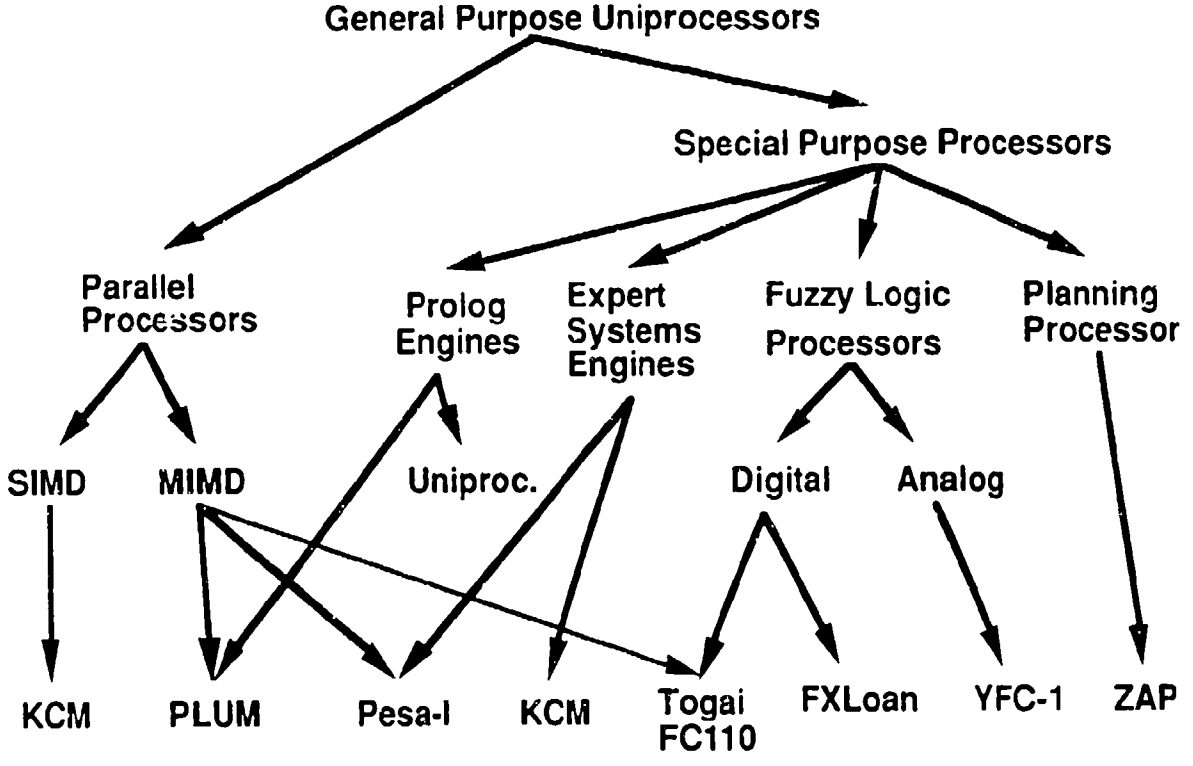


Figure 9-2 Representative Examples of Hardware Applicable to Planning

9.2.1 Zap Processor:

This is the only instance of specifically planning hardware known. It was developed by Lockheed and is proprietary, with most details not being releasable. The information given here was made available by Marty Broadwell of Lockheed, Georgia. The ZAP is a single 6"x 6" VME board special processor for the dynamic programming algorithm. The planning space represented is 128x128x8x8x8, with the dimensions used to represent latitude, longitude, altitude, heading, and bank respectively. A raster-like optimization process builds a cost matrix from a seed point to all other points in the space. Templates to represent cost components, such as a cookie-cutter pattern for exposure to a missile site, can be added or

deleted very quickly. The processor is built with off-the-shelf components. Programmable Gate Arrays are used for the processor, that searches a large amount of memory at a time. The current version of the ZAP searches at 2 to 3 times the speed of an R3000 (25Mhz RISC), a new model will be 30 times faster and have added flexibility. This will give a n execution time of less than a second for a full scale problem, or faster for smaller scaled problems.[1]

9.2.2 Parallel Prolog:

Ramkumar and Kale give benchmarks for a variety of MIMD parallel Prolog implementations on Alliant, iPSC/2, Multimax, and Symmetry machines. Speedups were near linear for the tests with up to 20 processors, though all showed single processor performance nearly always worse than either single processor implementation used for comparison (SB Prolog on Multimax and Quintus on a Sun 3/60). The uni-processor implementations did not use "cut". The benchmarks here appear promising, but are for very simple problems (Fib26, Qn9, Fib+, and Occur) and with relatively few processors [2]. Saletore and Kale give some additional benchmarks [3]. Shyam Mudambi shows the Aurora OR-Parallelism based parallel system achieving 10 times the performance of a Sun 3/50 on a 40 processor BBN GP-1000. Some benchmark problems bottlenecked quickly, though [4]. Additional Aurora benchmarks are given by Peter Szeredi [5]. Ashok Singhal and Yale Pratt describe PLUM (Parallel Unification Machine), a system that performs Prolog Bookkeeping and Unification in parallel to give a speedup of 1.7 beyond that available though other parallelism [6][7].

Recently, a Transputer based commercial parallel PROLOG product from Paralogic has been announced. Both AND and OR type parallelism is used. Processor boards bearing 5 Transputer processors are available for the IBM PC-AT [8].

9.2.3 Specialized Prolog Engines

A number of examples of specialized back-end Prolog or production rule crunching machines were found. For example, H. Benker et. al. report on a fairly complex board level processor specialized for Prolog. It runs about 5 to 10 times faster than Quintus Prolog on the SUN 3/280, the Quintus Prolog seeming to be a commonly cited basis of comparison [9] (The Sun 3/280 is 68020, 25Mhz with 20Mhz FPU.) The KCM type

approach, upgraded for more recent improvements in component technology, would likely not compare as favorably to RISC type processors.) Konagaya, et. al. report on a processor that runs 1.5 times faster than Quintus Prolog on a Sun 4/280 despite having a 200ns clock vs 66ns for the Sun. Interpreted performance (with "Dynamic Clause Compilation") runs 6 to 8 times faster. This is part of the "Fifth Generation" project.[10]

A VLSI approach to a specialized Artificial Intelligence oriented processor described by Maeda et. al. takes a 5 stage pipelining approach to achieve as much fine grain parallelism within the processor as possible. The IP1704 processor achieves a performance of 1.5M Prolog LIPS executing the Append benchmark, using 11 clock cycles per inference. The processor also can execute Lisp [32].

Such processors follow Von Neumann or Harvard architecture principles, but incorporate specialized elements or greater fine-grained parallelism to give an advantage. About an order of magnitude improvement over general purpose processors seems to be the limit of such dedicated special purpose processors for most problems. That is the range of performance found in literature surveyed to date. This advantage would seem to be shrinking for a number of reasons, including differences in time to market, the necessary customization of hardware for the specialized engine requiring a greater lead time, and the lesser resources available for development of a more specialized engine. The trend seen for specialized Lisp processors is likely also applicable to processors specialized for Prolog and other languages. The mass market price benefits, increasing relative capability, and software advances for general purpose micros has swamped most of the advantages of specialized Lisp Machines in the marketplace.

9.2.4 Production Rule Systems:

A number of papers about processors for production rules (as in expert systems) rather than Prolog were found. PESA I, a "Parallel Architecture for Production Systems" was an interesting example. Up to 32 MIMD processors are arrayed in layers which process rules in a somewhat pipeline manner. The last stage of operation is "Conflict set resolution". Speedup drops greatly beyond 8 processors, however. Performance of 8000 rule firings per second was achieved, using 1.6 MIP processor elements. A limit of about an order of magnitude speedup over sequential seems to apply to both Prolog and rule based systems [11]. The

processing of production rules in this system follows the "Rete" algorithm in which the compiler maps rules to an acyclic data flow graph, then the rules are interpreted. This was also true for the system developed by Anurag Acharya and Milind Tambe [12]. The rule system used for both was OPS5. Benefit is gained by storing the partial results of matches from previous cycles to use in later ones (since only a small part of working memory changes on each cycle) and by sharing the results of computations shared by rules being simultaneously processed. These principles seem applicable to sequential as well as parallel processing.

9.2.5 Fuzzy Logic:

A starting point for surveying hardware for fuzzy logic is Gupta and Yamakawa [13]. It contains papers describing elements that contribute to fuzzy computing engines. These include inference engine elements, memory, and controller [14]. A fuzzy flipflop is described [15]. However, all of this is very far from practicality for the Pilot's Associate program. A number of software based fuzzy logic systems exist, which were not surveyed given the hardware focus of this effort. An example is the VAX based ERNEST [16], which includes "arbitrary procedures to allow probabilistic and fuzzy reasoning". An example of a specialized fuzzy logic processor applied to an application is FXLoan, a system that evaluates loan applications. Although this system is simulated (as of the publication date), the hardware design shows the features necessary for fuzzy logic processing [34].

Togai Infralogic, Inc. produces a family of Digital Fuzzy Processor boards for VME and PC/AT busses as well as a stand-alone modules. A special fuzzy processor chip is advertised as processing over 370K two-premise fuzzy production rules per second (or over 1M Boolean rules). The VME product uses four such processors. Considerable software support for development is available. Details of the processor's internal structure were not available.[29]

9.2.6 Other Specialized Hardware:

Perhaps the most interesting specialized processor is the IXM, a multi-processor engine for implementing a semantic net [17]. A prototype handles 64K links using associative memories. It has 32 processors and 128K words of 40 bits. The language IXL is a Prolog superset that includes predicates for semantic network processing. The architecture is essentially

SIMD with a pyramid of connections. This was one of the most innovative pieces of work found.

9.2.7 Application of SIMD to Planning:

Delivery of the greatest amount of raw computation for a given price seems to be in SIMD type architectures. However, planning applications have been hard to find. One example is from the robotics domain: RAMBO. The Connection Machine is used to simultaneously determine, for several trajectories from one point in a target body relative space to another, several possible paths associated with different times to make the transition. For each discrete time step along each path derivatives are calculated in parallel. The path having the shortest time yet not exceeding limits on the derivatives, is chosen [30].

9.3 Other Hardware having Potential Application to Planning

The machines discussed in the preceding section were designed to be used for machine intelligence types of problems, of which planning is an example. In addition, machines exist which do not currently fit in the planning technology hierarchy, but which may prove useful for planning if appropriate mappings from the planning domain can be developed. In some cases, the machines discussed are already being applied to closely related problems. This is especially true for machines.applied to vision.

9.3.1 Vision Architectures:

One thread of potential application that seems worth pursuing is the possibility of mapping between the planning problem and the machine vision problem. If such a mapping can be found, the hardware (and software) techniques currently being applied to machine vision and image understanding may be applied to planning also. Dechter, Meiri and Pearl describe the application of graphs to constraint satisfaction, which is seen as a key part of the planning process [18]. It may prove possible to show formally that vision and planning are related using graph theory. Such proof would establish the applicability of vision oriented architectures to both problems.

The computer vision community has made considerable progress toward high performance hardware for vision computation, and towards

benchmarking. Performance for various vision algorithms on several sequential and parallel machines has been reported [19]. Vision algorithms have been mapped to a variety of general purpose MIMD and SIMD computers and to special purpose processors. Commercial products such as KBVision (which incorporates much of the developments from the University of Massachusetts) are emerging [20]. The extent to which the planning problem can benefit from this extensive work should be a high priority for investigation.

A number of papers have described mappings of graph and image related processing onto specialized processors built for vision algorithms, but which are in fact SIMD machines with more general capabilities. Heaton, Blevins, and Davis describe a 128 processor SIMD chip having a 22 bit control bus and capable of 20Mhz operation. This 1.1 Million device chip uses 1 micron CMOS and executes an instruction every clock cycle [21]. This and similar SIMD's meant specifically for image work, such as the much earlier 72 processor GAPP chip (Martin Marietta and NCR) tend to have memory that is on-chip and thus smaller and faster than the more "general purpose" SIMD's such as the Connection Machine [22]. The paucity of memory per processor gives such machines a different characteristic; they simply do not hold enough data to support the usual message oriented graph abstractions. They are usually limited to a mesh interconnect structure. However, they might prove capable of supporting other network or planning domain abstractions, in which case their relatively high computational advantage can be brought to bear.

Widespread application of image processing machinery is resulting in reduced prices for commercial specialized hardware systems, some of which might be useful for the planning problem. For example, VZITec Inc. has announced a $20K single board imaging system that performs at 175 MOPS and supports C, Motif, and X-Windows. [23].

9.3.2 Related Domain Hardware:

High performance hardware has been applied to a number of other problem areas which may have a relation to planning. For example, speech recognition and natural language processing involve massive search, as is also true of planning. M. Motomura et. al. of NEC describe a chip used for word search that is capable of finding word entries that are approximate matches. This content addressable approach allows 2048 words per chip, and handles an input character per clock [24]. Similar kinds of search mechanics may be useful for planning. A. Stölzle, et. al., describe a

processor that implements the Viterbi algorithm for finding the most probable state sequence for the utterance [25]. A comparable way of considering traversing a series of states in planning might allow the use of this or similar high performance hardware.

The fruits of the VHSIC program may prove applicable as well. These and other high performance processors are targeted to signal processing applications. A news report of the joint Motorola-TRW CPUAX central processor unit, a VHSIC phase 2 product, credits the 1.5 x 1.6 inch chip with 4 M transistors, and 200 MFLOP performance. It uses self-repair redundancy, requiring 61 of 142 macrocells for full capacity. This development points to the probability of increased capability for the other hardware approaches already mentioned [26].

Parallel database and query systems research has produced hardware for database machines which may well be applicable to planning. Resources did not permit an investigation of this field [33].

9.3.3 Neural Networks

Artificial Neural Systems have to date been applied more toward deductive processes such as diagnosis and recognition than constructive activities such as planning. However, JPL has reported prototype application to both path planning and to the allocation of resources [28]. There are numerous implementations of neural networks involving analog or digital hardware or simulation on more general purpose machines. These machines were not surveyed.

9.3.4 Other High performance Hardware

This last category includes recently emerging hardware with relatively limited scope of applicability but enormous computational power. The problem is the development of abstractions, programming techniques and problem mappings, that would allow the potential of these machines to be applied to planning. The following description is intended to be illustrative of this category.

The "Datawave" chip developed by Intermetall GmbH/ITT is essentially a MIMD processor intended to be arranged as a Mesh with FIFO's connecting adjacent processors. Each chip includes a register file, 64x48 bits of program RAM, and ALU and Multiplier arranged with four

busses in the chip to allow simultaneous use of resources. The clock is 125 Mhz. The chip is reported as expected to sell for $30 to $40 in a 124 pin plastic package. A 4x4 array is expected to achieve 4 billion operations per second and throughput of 750 Mbytes per second applied to video problems. This illustrates the enormous raw computational power of current technology now being applied to highly structured problems. (The iWarp is another product, which had its origins in the DARPA sponsored Warp project at CMU, that also falls into this general category between SIMD and MIMD, as does the Transputer.) Optical processors are perhaps even more powerful in raw computational power, but even more difficult to apply.

Mapping of planning algorithms directly to hardware, as with the Zap processor, now entails custom design requiring considerable time and resources. This is avoided normally by using programming abstractions to adapt more general purpose, but less efficient, computational machinery to the task. A possible future alternative is compilation of the algorithms to hardware, in the form of VLSI. Some pieces of the infrastructure to support this concept, silicon foundries, design-to-test techniques, design frames and system kits already exist. The key remaining link is translation of a problem oriented software description into hardware definition. Some progress is being made on this front. Barada and El-Amawy have developed a methodology for mapping a restricted class of algorithms to VLSI. These are forms having a series of nested iteration loops around a series of if/else if rules [31]. This form may be appropriate for declarative languages such as the constraint languages of interest for planning, or a subset of them. Aside from custom VLSI, a range of semi-custom fabrication techniques exist which may also serve as vehicles to migrate algorithms to hardware, if the compilation techniques can be developed [35].

9.4 Conclusions and Recommendations

Very little hardware work specifically focused on planning was in evidence. It is quite possible that others exist that are proprietary or classified. A number of specialized machines to support declarative languages such as Prolog were found, but these tended to focus on early declarative language forms and benchmark well known simple programs like n Queens. No hardware aimed at more general constraint languages, such as Prolog 3, was found. Processors for neural nets and fuzzy logic are emerging which may have application to the planning problems of interest to Pilot's Associate. A mapping from the constraint language to the

hardware used for fuzzy logic might be possible. The JPL report shows at least some capability for neural nets in planning. The one processor dedicated to planning, the Zap processor, uses a dynamic programming algorithm which addresses only part of the overall planning problem.

At the same time, there is an explosion of raw computational capability underway in SIMD or SIMD-like MIMD processors. These have so far proved most effective in highly structured problems of matrix form that maps directly to the processor array, such as signal and image processing. Figure 9-3 illustrates the goal of finding means, in the form of new abstractions or adaptations, necessary to increase the applicability of these machines for planning. Where it is possible to map a planning space to a spatial grid, as is also the case for the Zap processor, such machines can be usefully applied. One can project a two dimensional map onto the processor array, for example, and vary constraints by flagging processors as unavailable for path traversal. Mapping more amorphous problem configurations such as graphs is more difficult. The best use of these machines in a manner comparable to that needed for planning appears to be in the domain of machine vision.

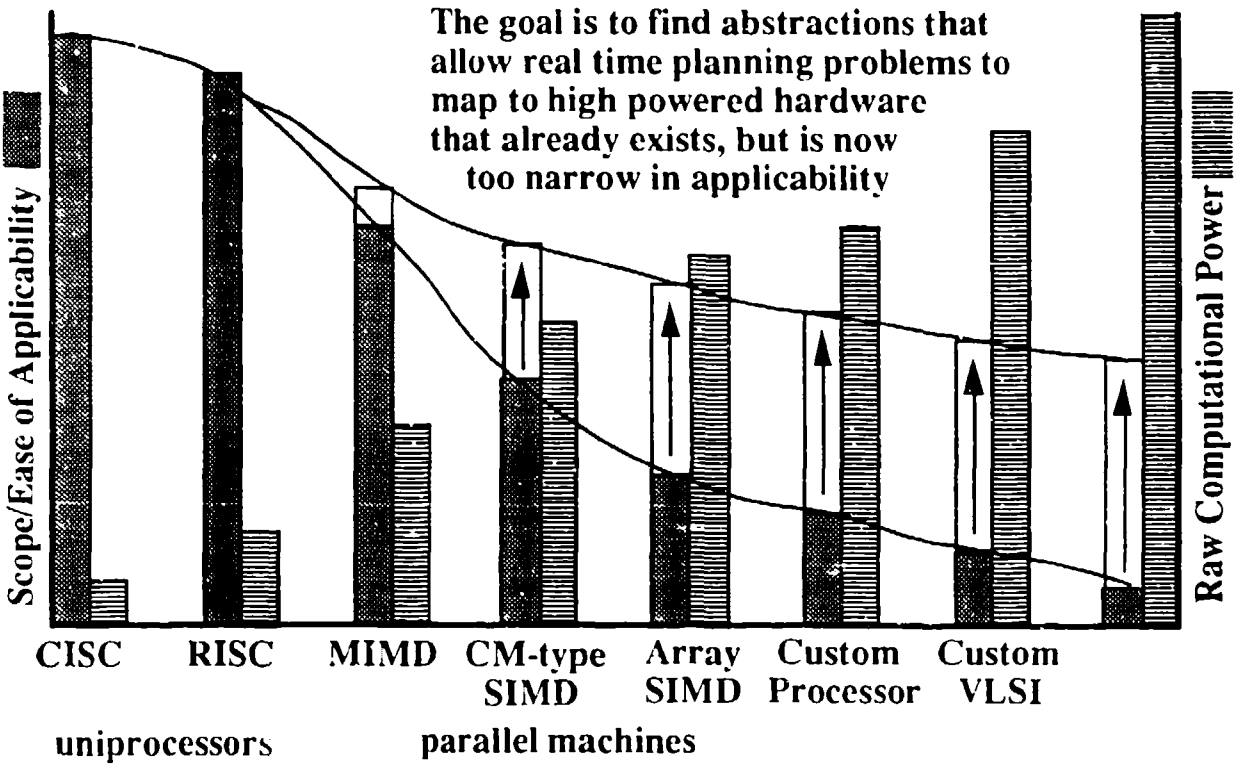


Figure 9-3 Goal of Applying Highly Parallel Forms of Computers

The following recommendations are made in the interest of incorporating high performance hardware into the planning technology for Pilot's Associate:

1. Look for mappings from the planning problem domain to the vision domain, so that vision algorithms and hardware can be exploited for planning as well.

2. Look for other (more direct) mappings of planning onto SIMD and tightly coupled SIMD-like MIMD machines. The search should target both simple grid SIMD machines and those having more complex routing structures.

3. Investigate the possibility of generalizing on the Prolog and Expert systems engines to handle more general constraint languages. Determine whether fuzzy logic engines can be used for this purpose.

4. Investigate implementing primitive operations or algorithms of planning as custom processors and, ultimately, VLSI. Monitor emerging hardware tools such as programmable logic for opportunities to support custom approaches at minimum infrastructure cost.

5. Monitor the application of Neural Nets for suitability for planning.

9.5 References

The references for this chapter are numerically denoted and located in the reference section of the publication.

CHAPTER 10

SUMMARY AND RECOMMENDATIONS

In this document, we have: (1) reviewed the principal paradigms for automated planning, (2) explored the relationship between AI-automated planning technology with some related technologies (decision theory, optimization theory, hardware engineering), and (3) examined the relevance of each class of techniques to associate system technology. With regard to (3), each chapter identified how various automated planning techniques can be applied to associate systems that provide real time planning support. In this section we summarize these potential applications and make our final recommendations.

Planning as Heuristic Search - Most route planning systems draw from this technology. In many cases, A* or some other global search procedures becomes the core algorithm for generating routes. This will continue to be a fruitful area for development. However, there are fundamental limits on the extent to which global search techniques can scale. As the numbe of input variables increases, the complexity of global search techniques generally increases exponentially. We do not see much hope in getting around these scaling problems. Consequently, global search procedures can only account for a subset of all relevant factors and constraints when generating routes. This suggests that the output of global-search-based route planners can never be stand-alone. Some post-processing will inevitably be required to check these routes for realism.

Generative Planning - Classical planners and constraint-based planners are usually considered to be generative planners -- planners that operate by generating plans from scratch or a limited plan skeleton. In general, planners in this category are not well-suited to real-time planning and replanning problems. For a generative planner to be part of a real-time system, two things are necessary: a library of well-engineered skeletal plans and a reactive control system. The skeletal plans are needed so that the planner avoids searching through a very large space of possible plans. A reactive control system is needed so to guarantee real-time reactivity while when the situation is evolving faster than the planner can react. The relationship between planning and real-time reactivity is currently a significant research area in AI. The basic research relating generative planning and reactive control should be monitored.

Transformational Planning - As described in Chapter 4, transformational planners operate by recalling and modifying fully-detailed plans. In contrast to the generative planners, these planners seem be suited to real-time planning problems. Since they always begin with a detailed plan, immediate execution of the first steps of that plan should be feasible. However, there is a risk. If the plan is executed prior to completing the transformation process, then the plan may be faulty and the first steps of that plan may represent inappropriate actions. On balance however this seems like a reasonable approach to real-time planning/replanning. A possible initiative in this area should be explored.

Planning from First-Principals - This remains an interesting research area with potentially high payoff. However, first-principals planners are not ready for serious applications.

Planning, Probability and Decision Theory - Decision and probability theory can be used to project probabilities, select options, control the planning process, allocate resources to control versus planning, support anytime problem solving, etc. The application of probabilistic and decision theoretic techniques to automated planning is just beginning to be explored. Although this work is still exploratory, it deserves careful monitoring.

Planning and Optimization - As discussed in Chapter 8, there is a natural complementarity between AI-based heuristic problem solving and OR-based optimization techniques. We believe that a careful integration of AI-based automated planning techniques with optimization procedures could lead to substantial improvements in automated planning technology. Specifically, we believe that systems can be developed that generate plans in real-time that are of reliably high quality. A specific architecture for merging these approaches was proposed in Chapter 8. Many others are possible. We believe this to be a key area for future investment.

Hardware for Planning - Automated planning is usually a computationally intensive activity. Because of this, it seems that hardware systems tailored to processes necessary for automated planning could be useful. This area has not been explored in great depth, but, as we showed in Chapter 9, there is considerable potential for dedicated hardware to be applied to planning problems.

REFERENCES

Chapters 1-8 References.

Agre, P. and Chapman, D. (1987) "Pengi: An Implementation of a Theory of Activity," *Proceedings of the Sixth National Conference on Artificial Intelligence*. 268-272.

Allen, J. (1983) "Maintaining Knowledge about Temporal Intervals," *Communications of the ACM*, 26, 11, 832-843.

Allen, J. (1984) "A General Theory of Action and Time," *Artificial Intelligence*, 21:121-154.

Ambros-Ingerson, & Steel, 1988 --

Berg-Cross, G. (1991) "Issues for Multiagent-Based Associate Planning Systems," *Proceedings of the Conference on Associate Technology*, 3-16.

Boddy, M. (1991) "Anytime Problem Solving Using Dynamic Programming," *Proceedings of the Ninth National Conference on Artificial Intelligence*, 738-743.

Brooks, R. (1991) "Intelligence without Representation," *Artificial Intelligence*, 47, 139-159.

Brown, F. (ed.) (1987) *The Frame Problem in Artificial Intelligence*, Morgan Kaufmann.

Chapman, D. (1987) "Planning for Conjunctive Goals," *Artificial Intelligence*, 32:333-377.

Dean, T. (1990) "Planning under Uncertainty and Time Pressure," in Katia Sycara (ed.) *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, Morgan Kaufmann Publishers, 390-395.

Dean, T. and Boddy, M. (1988) "An Analysis of Time-Dependent Planning," *Proceedings of the Seventh National Conference on Artificial Intelligence*, 49-54.

Dean, T. and Kanazawa, K. (1989) "Persistence and Probabilistic Projection," *IEEE Transactions on Systems, Man, and Cybernetics*, 19(3), 574-585.

Dean, T. and McDermott, D. (1987) "Temporal Data Base Management," *Artificial Intelligence*, 32(1), 1-55.

Dechter, R. Merir, I and Pearl, J. (1989) "Temporal Constraint Networks," *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, 83-??.

DeKleer, J. (1988) "An Assumption-Based Truth Maintenance System," *Artificial Intelligence*, 28, 127-162.

Doyle, J. (1979) "A Truth Maintenance System," *Artificial Intelligence*, 12, 231-272.

Fagin, R. and Halpern, J. (1988) "Belief Awareness and Limited Reasoning," *Artificial Intelligence*, 34: 480-490.

Fikes, R. and Nilsson, N. (1971) "STRIPS: A new Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, 2, 189-208.

Fikes, R., Hart, P. and Nilsson, N. (1972) "Learning and Executing Generalized Robot Plans," *Artificial Intelligence*, 3, 4.

Freeman, R. (1991) *Searching for a Better Plan*, Ph.D. Dissertation, in preparation.

Ginsburg, M. (1987) Possible Worlds Planning," in M. Georgeff and A. Lansky, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, Morgan Kauffman.

Ginsburg, M.(1990) "Computational Considerations in Reasoning about Actions," in Katia Sycara (ed.) *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control.* San Mateo: Morgan Kaufmann Publishers, Inc.

Green, C. (1969) Applications of Theorem Proving to Problem Solving," *Proceedings of the First International Joint Conference on Artificial Intelligence.*

Haack, S. (1978) *Philosophy of Logic*, Cambridge University Press.

Haddawy, P. and Hanks, S. (1990) "Issues in Decision Theoretic Planning: Symbolic Goals and Numeric Utilities," in Katia Sycara (ed.) *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control.* San Mateo: Morgan Kaufmann Publishers, Inc., 48-58.

Halpern, J. and Shoham, Y. (1986) "A Propositional Modal Logic of Time Intervals," *Proceedings of the Symposium on Logic in Computer Science*, Boston, MA., 279-292.

Hammond, K. (1989) *Case-based Planning: Viewing Planning as a Memory Task.* Academic Press, New York.

Hanks, S. (1990) "Practical Temporal Projection," *Proceedings of the Eighth National Conference on Artificial Intelligence.*

Hendler, J. and Agrawala, A. (1990) "Mission Critical Planning: AI on the MARUTI Real-Time Operating System, in Katia Sycara (ed.) *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control.* San Mateo: Morgan Kaufmann Publishers, Inc.

Henrion, M. (1991) "When is a Model Big Enough," in working notes for the *Workshop on Knowledge-based Construction of Probabilistic and Decision Models.*

Horvitz, E. (1991) Personal communication.

Jackson, P., Reichgelt, H. and van Harmelen, F., *(1989) Logic-Based Knowledge Representation*, Cambridge, MIT Press.

Kaelbling, L. (1987) "An Architecture for Intelligent Reactive Systems," in M. Georgeff and A. Lansky, *Reasoning about Actions and Plans: Proceedings of the 1986 Workshop*, Morgan Kauffman.

Kaelbling, L. and Rosenschein, S. (1990) "Action and planning in embedded agents," in Patti Maes (ed.) *New Architectures for Autonomous Agents: Task-level Decomposition and Emergent Functionality*, Cambridge: MIT Press.

Kambhampati, S. (1990) "A Theory of Plan Modification," *Proceedings of the Eight National Conference on Artificial Intelligence*.

Kautz, H. and Ladkin, P. (1991) "Integrating Metric and Qualitative Temporal Reasoning," *Proceedings of the Ninth National Conference of Artificial Intelligence*, 241-246.

Korf, R.E. "Real-time Path Planning," *Proceedings of the DARPA Workshop on Knowledge-Based Planning*, December 1987.

Korf, R. (1988) "Planning as Search," *Artificial Intelligence*.

Kowolski, R. (1979) *Logic for Problem Solving*, North Holland.

Laskey, K. (1990) "A Probabilistic Reasoning Environment," *Proceedings of the Sixth conference on Uncertainty in Artificial Intelligence*, 415-422.

Laskey, K.B. and Lehner, P.E. (1989) "Assumptions, Beliefs and Probabilities," *Artificial Intelligence*, 41:65-77.

Laskey, K.B. and Lehner, P.E. (1990) "Belief Maintenance: An Integrated Approach to Uncertainty Management," in *Readings in Uncertainty*, Shafer, G. and Pearl, J. (eds.), Morgan Kaufman Publishers, Inc.: San Mateo, CA.

Laskey, K. and Lehner, P. (1991) "Some Maxims for Small Worlds," in working notes for the *Workshop on Knowledge-based Construction of Probabilistic and Decision Models*.

Lehner, P.E. (1990e) "Adversarial Planning Search Procedures with Provable Properties," in *New Directions in Command and Control Systems Engineering*, S. Andriole (ed.), AFCEA International Press: Fairfax, VA.

Lehner, P.E. (1991) "Probability and Anytime Problem Solving," *Proceedings of the Conference on Associate Technology*, 177-183.

Lindley, D. (1982) "Scoring Rules and the Inevitability of Probability," *American Statistical Review*, 50:1-26.

Linden, T. (198?) Transformational Synthesis paper, *AI Magazine*, ...

Ligozat, G. (1991) "On Generalized Interval Calculi," in *Proceedings of the Ninth National Conference on Artificial Intelligence*, 234-240.

Mackworth, A. (1987) "Constraint Satisfaction," in S. Shapiro (ed.) *The Encyclopedia of Artificial Intelligence*, New York: Wiley, 205-211.

Martin, and Allen (1990) "Combining Reactive and Strategic Planning through Decomposition Abstraction," in Katia Sycara (ed.) *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control.* San Mateo: Morgan Kaufmann Publishers, Inc., 137-143.

McAllester, D. and Rosenblitt, D. (1991) "Systematic Nonlinear Planning," Proceedings of the Ninth National Conference on Artificial Intelligence, 634-639.

Meng, A., Ntafos, S. and Tsoukalas, M. "An Approach to Real-Time Path Planning for Handling Targets and Unexpected Threats," in *Proceedings of the Conference on Associate Technology*, George Mason University, June 1991.

Minton, S. (1988) *Learning Search Control Knowledge: An Explanation-based Approach*, Kluwer.

Mitchell, B. (1991) "An Overview of the F-117A Mission Planning Systems," presented at the Associate Technology conference, June 1991.

Newell, R. and Simon, H. (1963) "GPS, a Program that Simulates Human Thought," in Feigenbaum, E. and Feldman, J. *Computers and Thought.* McGraw-Hill.

Payton, D. (1990) "Exploiting Plans as Resources for Actions, in Katia Sycara (ed.) *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control.* San Mateo: Morgan Kaufmann Publishers, Inc.

Pearl, J. (1984) *Heuristics*, Addison-Wesley.

Pearl, J. (1990) "Which is more believable, the probably provable or the provably probable," Technical Note, UCLA, January 1990.

Pednault, E. (1988) "Extending Conventional Planning Techniques to Handle Actions with Context-Dependent Effects," *Proceedings of the Seventh National Conference on Artificial Intelligence*, 55-59.

Peot, M. and Breese, J. (1991) "Model Construction in Planning," in working notes for the *Workshop on Knowledge-based Construction of Probabilistic and Decision Models.*

Reiter, R. (1980) "A Logic for Default Reasoning," *Artificial Intelligence*, 13: 81-132.

Reiter, R. (1987) "Non-monotonic Reasoning," *Annual Review of Computer Science, Vol 2, 1987.*

Sacerdoti, E. (1977) *A Structure for Plans and Behavior*, Elsevier Publishing Co.

Silbert, M. "An efficient algorithm for path planning to avoid dynamic 3D obstacles," in *Proceedings of the Conference on Associate Technology*, George Mason University, June 1991.

Shoham, Y. (1988) *Reasoning About Change*, MIT Press.

Slagle, J. and Hamburger, H. (1985) "An Expert System for a Resource Allocation Problem," *Communications of the ACM*, 28(9), 994-1004.

Stefik, M. (1981) "Planning with Constraints," *Artificial Intelligence*, 16, 111-140.

Tate, A. (1977) "Generating Project Networks," *IJCAI77*, 888-893.

Wellman, M. (1990) Formulation of Tradeoffs in Planning Under Uncertainty. Pitman and Morgan Kaufmann.

Wellman, M. and Doyle, J. (1991) "Preferential Semantics for Goals," *Proceedings of the Ninth National Conference on Artificial Intelligence*, 698-703.

Wilkins, D. (1988) *Practical Planning: Extending the Classical AI Planning Paradigm*, Morgan Kaufmann.

Chapter 9 References.

1. Marty Broadwell, "Mission Planner Subsystem Functional Description", presentation at SAC Subcommittee meeting, March 1991, and subsequent discussions concerning the Zap processor, and conversations with M.G. Madden, also of Lockheed, in March 1991.

2. Ramkumar and Kale, "Compiled Execution of the Reduce-OR Process Model on Multi-processors", Logic Programming, Proceedings of the North American Conference, 1989, pp313-331.

3. Saletore and Kale, "Obtaining First Solutions Faster in AND-OR Parallel Execution of Logic Programs", Logic Programming, Proceedings of the North American Conference, 1989, pp697-712.

4. Shyam Mudambi, "Performance of Aurora on a Switch Based Multi-processor"Logic Programming, Proceedings of the North American Conference, 1989.

5. Peter Szeredi in "Performance Analysis of the Aurora Or-Parallel Prolog System", Logic Programming, Proceedings of the North American Conference, 1989, pp713-732.

6. Ashok Singhal and Yale Pratt, "Unification Parallelism: How much can we exploit?", Logic Programming, Proceedings of the North American Conference, 1989, pp1134-1147.

7. Ashok Singal, "A High Performance Prolog Processor with Multiple Functional Units", Proceedings of the 16th Annual International Symposium on Computer Architecture, IEEE, 1989, p195-202

8. Paralogic, "Parallel PROLOG On Your Desktop", AI Review, Summer 1990.

9. H. Benker et. al. "KCM: A Knowledge Crunching Machine", Proceedings of the 1989 International Symposium on Computer Architecture, pp186-194.

10. Konagaya, Habata, Atarashi, and Yokota (of NEC), in "Performance Evaluation of a Sequential Inference Machine CHI", Logic Programming, Proceedings of the North American Conference, 1989.

11. "PESA I- A Parallel Architecture for Production Systems from the 1987 Parallel Processor Conference Proceedings.

12. Anurag Acharya and Milind Tambe, "Production Systems on Message Passing Computers: Simulation Results and Analysis", also in the 1989 Parallel Processing Conference Proceedings.

13. M.M. Gupta and T. Yamakawa, ed. ,Fuzzy Computing, Theory, Hardware, and Applications

14. M.M. Gupta and T. Yamakawa, ed. ,Fuzzy Computing, Theory, Hardware, and Applications (author & title?) describes the inference engine elements, memory, and controller

15. Hirota and Ozawa in this same book describe a fuzzy flipflop.

16. H.Niemann, G.F.Sagerer, S.Schroder, and F.Kummert, "ERNEST: A Semantic Network System for Pattern Understanding", IEEE Transactions on Pattern Analysis and Machine Intelligence, Sept. 90 Vol 12 No 9 pp 883-905

17. Furuya, Kusumoto, Handa, and Kokubu in "The Prototype of a Semantic Network Machine IXM", 1989 International Conference on Parallel Processing

18. Rina Dechter, Itay Meiri and Judea Pearl, "Temporal Constraint Networks", Knowledge Representation '89 Proceedings

19. Charles Weems, Edward Riseman and Allen Hanson of the University of Massachusetts and Azriel Rosenfeld of the University of Maryland, "A Report on the Results of the DARPA Integrated Image Understanding Benchmark Exercise", Proceedings, 1989 Image Understanding Workshop

20. "KBVision Marketing material provided by Amerinex Artificial Intelligence, Inc.

21. R. Heaton, D. Blevins, and E. Davis, "A Bit Serial VLSI Array Processing Chip for Image Processing", JSSC vol 25, No. 2, Apr 90, pp364-368

22. NCR Corporation,"NCR45CG72 Geometric Arithmetic Parallel Processor", Dayton, Ohio, 1984.

23. Electronics Design, Dec 13 1990, p136.

24. M. Motomura et. al. of NEC, in "A 1.2 Million Transistor 33Mhz 20-b Dictionary Search Processor (DISP), Journal of Solid State Circuits, vol 25, No.5, Oct 90 pp1158-1165.

25. A. Stölzle et. al., "Integrated Circuits for a Real Time Large Vocabulary Continuous Speech Recognition System", JSSC,Vol 26, No 1, Jan 91, pp2-11

26. A news report of the joint Motorola-TRW CPUAX central processor unit (??) (I need to find the original and get a proper citation.)

27. John Gosch, "Video Array Processor breaks speed record", Electronics Design, July 12, 1990 p133

28. Taher Daud, Silvio Eberhardt, and Anil Thakoor, "Electronic Neuroprocessors and Neural Networks for Global Optimization", Technical Exchange Meeting on Parallel Computing and Neural Networks, Jet Propulsion Laboratory, December 18-20, 1990, Volume 1.

29. Togai Infralogic, Inc., Advertising literature distributed February, 1991, 30 Corporate Park, Suite 107, Irvine, CA, 92714.

30. Larry S. Davis, Daniel DeMenthon, Thor Bestul, David Harwood, H.V. Srinivasan, and Sotirios Ziavras, "RAMBO - Vision Planning on the Connection Machine", Proceedings of the 1989 Image Understanding Workshop, pp631-639.

31. H. Barada and A. El-Amawy, "A New Methodology for Mapping Algorithms into VLSI Arrays", Proceedings of the Third Annual Parallel Processing Symposium, IEEE Computer Society, March 29 1989, p31.

32. Ken-ichi Maeda, Takeshi Aikawa, and Mitsuo Saito, "Mechanisms for Achieving Parallel Operations in a Sequential VLSI AI Processor", Proceedings of the Third Annual Parallel Processing Symposium, IEEE Computer Society, March 29 1989, pp153-160.

33. T. Harder, H. Schoning, A. Sikeler, "Evaluation of Hardware Architectures for Parallel Execution of Complex Database Operations", Proceedings of the Third Annual Parallel Processing Symposium, IEEE Computer Society, March 29 1989. This is a survey article.

34. Meng-Hoit Lim and Yoshiyasu Takefuji, "Implementing Fuzzy Rule-Based Systems on Silicon Chips", IEEE Expert, IEEE, February 1990, pp 31-45.

35. Barbara Tuck, "High Level Synthesis Unlocks Potential of FPGA's", Computer Design, April 1, 1991, pp 50-54.