Strategy Construction using a Synthesis of Heuristic and Decision-Theoretic Methods

By Robert F. Sproull

XEROX

Strategy Construction using a Synthesis of Heuristic and Decision-Theoretic Methods

by Robert F. Sproull

CSL-77-2 JULY 1977

This report describes a framework for constructing plans, or *strategies*, in which aspects of mathematical decision theory are incorporated into symbolic problem-solving techniques currently dominant in artificial intelligence. The utility function of decision theory is used to reveal tradeoffs among competing strategies for achieving various goals, taking into account reliability, the complexity of steps in the strategy, the value of the goal, and so forth. The utility function aids searching for good strategies, acquiring a world model, allocating planning effort, and organizaing a hierarchical problem-solving system.

A problem-solving system that prepares travel itineraries is presented as a case study in integrating the techniques of decision theory and artificial intelligence. The system uses a model of the traveler's utility to organize a search for good solutions. The hierarchical structure of the search narrows the search by finding crude plans and then further refining them.

A central observation of this work is that locating an *optimal strategy* is not the proper procedure when the costs of the planning itself are taken into account. Instead, we desire to engage in *optimal planning*, in which the total expenditure of effort to find and execute the solution is in some sense optimal.

KEY WORDS AND PHRASES

Artificial intelligence, problem-solving, heuristics, hierarchical planning.

CR CATEGORIES

3.64, 3.66.

XEROX
PALO ALTO RESEARCH CENTER
3333 Coyote Hill Road / Palo Alto / California 94304

Acknowledgements

This report reproduces a dissertation submitted to the Stanford University, Computer Science Department. Any clarity of expression or thinking that the reader finds among these pages is predominantly the result of careful reading and criticisms of many people: Jerry Feldman, Cordell Green, and Terry Winograd (the reading committee); also Ivan Sutherland, Dan Bobrow, Jonathan King, and Marty Epstein.

This dissertation is built upon the stimulation of various other efforts in AI. Earl Sacerdoti's work on ABSTRIPS and especially on NOAH is inspirational. The work of fellow students at the Stanford Artificial Intelligence Project, particularly Yoram Yakimovsky, Russ Taylor and Bob Bolles, has also helped push this effort along. I am grateful for the discussions and help of these individuals.

The institutions that have supported this work deserve special thanks. I am most grateful to the people of the Xerox Palo Alto Research Center for their support and for their unexcelled patience with my slow progress on this work. PARC is a wonderful place to learn much about computing; its attractions are in part responsible for what follows. For support during my early years of graduate study, I am grateful to the National Science Foundation and to the Advanced Research Projects Agency.

It is a pleasant custom in dissertation acknowledgements to repay in part the enormous debt of student to teacher. I feel extremely fortunate in the schooling I have received, but especially in the generosity of five individuals. My parents have been my best teachers: it was my father who forced me to learn logarithms before the slide rule, calculus before physics; it was the exercises he generated and the questions he asked that made me "get it right." Ivan Sutherland, whose course in computer graphics I took almost by chance as an undergraduate, irrevocably attracted me to computing; since then working with him has been especially exciting, stimulating and gratifying. Jerry Feldman, in addition to advising me on this dissertation, introduced me to the "rest" of computer science; his forte is goading me into occasional action by pointing out my weaknesses. Lee Sonastine Sproull has offered me much that will not submit to expression; she has also labored valiantly to help improve my written expression!

Table of Contents

Chapter 1
Introduction

How can a computer be programmed to plan and execute strategies that achieve some desired goal? Variants of this problem have nurtured efforts to develop the necessary epistemology and computational techniques, as part of a pursuit of "artificial intelligence" (AI). The variant known as "robotics" has attempted not only to plan strategies involving actions similar to those of humans in the real world, but also to build electronic "eyes" to observe the world and computer-controlled arms and wheels to achieve motor activity.

A characteristic of most of these planning systems is that they generate reasonable plans, but not "optimal" ones. In fact, the techniques used in AI planning currently allow little discrimination of preference among reasonable plans. A certain amount of tinkering and adjustment offer only limited *ad hoc* relief.

What is needed is a combination of the powerful AI techniques for restricting attention to those plans that are reasonable, together with techniques for finding good plans among the reasonable ones. A simple but wasteful formulation of the combination might involve enumerating all reasonable plans and then evaluating each one to identify the best plan. One of the questions addressed in this dissertation is: how can a system benefit from a tighter integration of the techniques?

The notion of optimality that we shall introduce into AI planning is borrowed from mathematical decision theory. The central idea of the discipline is that a numerical *utility function* can be used to evaluate decisions (see Chernoff and Moses, 1959, or Raiffa, 1970, for introductions to decision theory). A single numerical value summarizes the advantages of a set of actions, including effects of uncertainty, risk, and value of outcomes. A typical utility function would be the profits realized from a particular investment outcome.

The aim of this research is to combine decision theory and AI techniques so that planning uses models of optimality that accompany a decision-theoretic formulation, and also uses heuristic information to avoid the large searches characteristic of problems cast solely in decision-theoretic terms. The basic approach is to conduct an organized search of plan alternatives using the utility function to measure the promise of partially-complete plans. Promising avenues are explored incrementally to build and refine plans. Between each increment, the promise of the plan is reassessed. As new information is gathered, the

pending plans are re-evaluated in light of the new information. Similarly, as plan execution progresses and the state of the world changes, pending plans are re-evaluated to take account of information acquired during execution or are discarded if they cease to apply to the case at hand.

Decision Theory

A dominant tenet of this work is that the utility function and the model of costs and outcomes accompanying it constitute a convenient way for representing certain kinds of information used to solve a problem. Two perspectives on the utility notions are important: what sorts of information are naturally represented in this fashion, and what computational techniques can be applied to this information in pursuit of problem solutions?

Many of the arguments for the utility function are the result of years of development of mathematical decision theory and of various *utility theories* (Fishburn, 1970). The justification of the utility function applies to many problems attacked in AI; consider some introductory arguments:

- The first, and perhaps the best, argument for a numerical utility function is that the choice between alternative courses of action is often inherently numerical. One chooses the cheapest, or fastest, or strongest alternative. Many problems in robot problem-solving are virtually inexpressible in non-numerical terms. Examples also abound in medicine: it is known only that a certain fraction of schizophrenics respond to a certain drug, and there is *no* way to determine in advance whether a given patient will respond.

- Another main use of a utility function can be called "comparing the incomparable." If flying is faster and safer than driving, but more expensive and subject to delay, how can we choose which to do? What change in price would cause us to choose otherwise? The expected utilities of the alternative decisions answer these questions. By contrast, a heuristic program capable of comparing differing strategies symbolically will have rules covering many combinations of goals and circumstances, and the addition of new entities may require significant reprogramming. But if utility information about trains were added to a representation of transportation based on utilities, no further rules would be required to

relate trains to other modes of transportation. Decision theory provides a uniform way of treating information related to choosing a course of action, given the relevant utility and probability values.

- Many of the properties of a plan that the planning system needs in reasoning are revealed by the utility function. The numerical utility measure permits two plans to be compared, permits the value of potential improvements to plans to be assessed, permits the planning activity to organize an orderly search for the best plans, and permits the effective allocation of resources to planning, acting and information-gathering.

This dissertation applies the utility function to reveal tradeoffs among competing strategies for achieving various goals, taking into account reliability, the complexity of steps in the strategy, the value of the goal, and so forth. Techniques of decision theory can be applied to satisfy a number of objectives when constructing and executing plans:

- Comparing alternative plans. Often these comparisons are made among *instances* of a common plan outline or skeleton.

- Coping with uncertainty. The calculation of expected utility permits the reliability of a plan to be considered when assessing its worth.

- Finding good plans. The utility of partially-completed plans, together with bounds on the utility of the completion required, can be used to organize and guide a search for the best plan.

- Improving a plan. Failure paths in plans can be elaborated with recovery plans. The increase in expected utility of the elaborated plan measures the effect of generating a plan to cope with the failure path.

- Acquiring information. Utility measures can be used to plan efficient strategies for acquiring information needed for further planning.

- Allocating resources among planning, acting or information-gathering. The utility measure can also be used to decide which activities are most beneficial to the system.

The last observation, that a utility measure can assess the value of planning, is particularly noteworthy. Knowledge of the performance behavior of the planning

system, however crude, is widely applicable. From an external viewpoint, it can be used to choose among planning, executing and information-gathering activities. Within the planning system, it can be used to allocate resources among competing planning techniques because the promise of a particular approach, measured by an estimate of its utility, can be balanced by an estimate of the planning effort required to complete the plan.

The objective of a problem-solver should be to maximize the utility of the planning process and the execution process jointly. The effort expended to develop a plan and the effort expended to execute the plan are equally important to the objective. We shall be concerned with developing techniques for such *optimal planning*. These methods will find *P-optimal* solutions, in contrast to solutions that offer only optimal execution utilities.

Symbolic Problem-Solving

The notions of optimality must be fused with planning techniques developed to find reasonable plans from symbolic models of a problem. AI research in general-purpose systems for planning and acting has drawn heavily on the closely related task of problem-solving.[1] An early paradigm for problem-solving relied on the techniques of predicate calculus and theorem-proving for representing the model and reasoning processes: the model of the behavior of actions and of the state of the world is expressed as axioms in predicate calculus; the problem statement is expressed as a theorem to be proved; the sequence of actions that will accomplish the desired result is extracted from a mechanical proof of the theorem (Green, 1969). This approach is limited by available theorem-proving techniques: it is not possible to cope with models of much complexity.

These difficulties have led to emphasis on procedural embeddings of similar models, often aided by one of a host of "AI programming languages" (Bobrow and Raphael, 1974). These techniques admit larger models and allow specification, in the form of a computer program, of solution methods that are likely to be fruitful. This ability to direct the processing along preferred avenues has been used in *ad hoc* ways to extract "good" solutions.

---

[1]We shall use the term *planning* to mean the process of creating a plan, or sequence of actions, which will achieve a given goal. Originally, *planning* was a technical term in AI that referred to techniques of planning out problem solutions before actually trying a detailed solution. Robotics work has typically used the term in the looser sense we adopt here.

The use of both of these methods, however, has been largely devoted to demonstrating the existence of one solution to the given problem, and not to exploring multiple solutions in order to find the best one. This convenient practice cannot suffice for our purposes: several alternative strategies must be examined and the *best* one selected for execution.

Often, the alternative strategies that must be considered in order to find the optimal solution are variants of a simple "skeleton." A real-world example of this sort of strategy is the "wheelless student problem," buying a used automobile. A typical procedure is first to read newspaper advertisements and bulletin boards to assess the situation generally. Then, at relatively low cost, one can telephone various purveyors of cars and inquire about them. At some point, one must actually go to the effort of seeing and driving certain of these. There are professional diagnostic services that can be employed, at considerable cost, to further test the car. In each of these steps, one must decide when to stop that stage and go on to the next one. One does not, of course, proceed in strict order; there will normally be alternatives at several different levels of investigation. Notice that the "plan" itself is trivial: read, telephone, look, drive, professionally test and buy. It is the application of this plan to the world situation that is difficult. This behavior is typical: many planning activities are characterized by complex applications of simple plans.

Reading this document

This dissertation uses two case studies to explore the synthesis of the AI and decision-theoretic approaches. Chapter 2 is devoted to the first, which uses an elaboration of the classical "monkey and bananas" problem to present a tutorial on aspects of decision theory and introduces several specializations for our purposes.

The second case study, described in Chapter 3, introduces additional specialized techniques. The chief features are a hierarchical structuring of symbolic and decision-theoretic reasoning, and an attention to the costs of planning activities. The study itself is a computer program that plans travel itineraries. A user states a problem such as "journey from Stanford to the University of Rochester," and gives some information about his utility function. The problem is attacked at several hierarchical levels of processing, giving rise to increasingly detailed plans. Each level summarizes progress on a plan by computing an upper bound

on the utility of the plan at this and all subsequent levels of processing. A promising partial plan is subjected to more detailed processing that tries to discover rapidly the major difficulties with the plan, often by adding constraints or complexities that cast increasing doubt on a previously good plan, and thereby reducing the utility of executing the plan. This process can be summarized as *finding fault with crude plans by considering more details.*

Chapter 4 explores some of the techniques used in the case studies in more detail. It aims both to offer more elaboration of some of the techniques and to suggest ways some may be generalized.

Chapter 5 is devoted to an analysis of the approach in light of previous and current AI work. The techniques explored in the case studies can be applied in a number of active areas. Chapter 6 recalls the main ideas of the dissertation and relates the fields of AI and Decision Theory.

The following table may help to point the reader to sections that describe points of particular interest:

Chapter 2
Introduction to Decision Theory and AI Planning

This chapter presents in a tutorial form the essential ingredients of the synthetic approach: the concepts of mathematical decision theory and their relation to conventional AI problem-solving techniques. The presentation uses the "monkey and bananas" problem as an example, exposing a sequence of techniques as the example is embellished. The description assumes familiarity with AI problem-solving (see Fikes, 1976, for a summary); it assumes little knowledge of decision theory (introductions can be found in Raiffa, 1970, or Chernoff and Moses, 1959).

2.1 Decision Theory in Symbolic Problem-Solving

Decision theory helps a symbolic problem-solver search for the best plan to achieve a given goal. A utility function on plans can govern a search strategy that explores plans of high utility; the search terminates by announcing the plan of highest utility.

We shall illustrate how symbolic problem-solving and decision analysis can be combined with the classical example: *A hungry monkey is in a room where a bunch of bananas hangs from the ceiling. The monkey cannot reach the bananas. There is, however, a movable box in the room; if the box is under the bananas and the monkey stands on the box he can reach the bananas and eat.* The goal for a symbolic problem solver is to find a plan that will feed the monkey.

A typical problem-solver is given a symbolic model of the problem and searches for a combination of "actions" that achieves a given goal. A possible symbolic model, specified in the style of a modern AI language, records information about the position of objects with an "AT" relation that associates an object and its Cartesian coordinate position. Additional relations declare boxes to be climbable and pushable and bananas to be edible. An initial set of relations might be:

```
(AT MONKEY 9 9 0)
(AT BANANAS 0 0 5)
(AT BOX 2 2 0)
(HEIGHT BOX 5)
(CLIMBABLE BOX)
(PUSHABLE BOX)
(EDIBLE BANANAS)
```

This set is a symbolic representation of a "state of nature," or simply a *state*.

The symbolic model must also include a model of the *actions* that are capable of effecting transitions among states. The actions are specified below. If all relations in the list of preconditions are in the set of relations describing the current state, then the operator can be applied. Application of an operator causes a transition to a new state by deleting relations in the delete list from the current set, and adding those in the add list. (The functions X, Y and Z refer to the coordinate entries in the AT relation. The symbol $ will match any value in the corresponding position in the relation.)

WALKTO(α)
- Preconditions: (AT MONKEY $ $ 0)
- Delete list: (AT MONKEY $ $ 0)
- Add list: (AT MONKEY X(α) Y(α) 0)

PUSHTO(α,β)
- Preconditions: (PUSHABLE α)
  (AT MONKEY X(α) Y(α) 0)
  (AT α $ $ 0)
- Delete list: (AT MONKEY $ $ 0)
  (AT α $ $ 0)
- Add list: (AT MONKEY X(β) Y(β) 0)
  (AT α X(β) Y(β) 0)

CLIMB(α)
- Preconditions: (CLIMBABLE α)
  (AT MONKEY X(α) Y(α) 0)
- Delete list: (AT MONKEY $ $ 0)
- Add list: (AT MONKEY X(α) Y(α) HEIGHT(α))

CONSUME(α)
- Preconditions: (EDIBLE α)
  (AT MONKEY X(α) Y(α) Z(α))
- Delete list: (EDIBLE α)
- Add list: (FED)

The problem-solver, given the goal (FED), tries to find a sequence of actions leading from the initial state to one that includes the (FED) relation. It might generate the sequence:

WALKTO(BOX)
PUSHTO(BOX,BANANAS)
CLIMB(BOX)
CONSUME(BANANAS)

We shall refer to such a sequence as a *plan* to achieve the goal. In a robotics experiment, the original relations and this plan can be used as a set of commands to software and hardware subsystems that cause a robot to simulate the actions of the monkey.

If the problem statement and the corresponding symbolic information given to the problem-solver were expanded to include multiple tools, multiple sources of food, or multiple goals, the problem-solver could generate other plans as well. If the initial relations model several boxes, a plan for each box could be generated. But if the number of alternative plans grows at all large, the combinatorial explosion will overwhelm any present problem-solver.

2.1.1 Computing the Utility of a Plan

The utility of one of the WALKTO, PUSHTO, CLIMB, CONSUME plans is derived from a *utility model* that accompanies the symbolic model. It is a measure of the value of achieving the goal, diminished by the cost of executing each of the steps in the plan. We shall assume for now that this utility can be expressed as a sum of contributions from individual steps and a contribution representing the value of achieving the goal. (Note: This treatment omits important considerations such as risk that are discussed in Chapter 4.)

For each goal, we assign a function that evaluates the utility of achieving the goal. In our example, we shall assign U_e=200 to the goal of eating, i.e., to achieving a state in which the monkey is fed. Goals of less value to the monkey are assigned correspondingly smaller utilities. For later reference, we shall assume that the next most desirable goal is "don't bother trying to eat," which has utility U_{db}.

The utility associated with executing each step of the plan is often called the "cost" of the step. A robotics experiment that simulates each operator with a collection of processes, including computation and control of a robot vehicle or manipulator, might use cost assignments that express the consumption of resources required to accomplish each step. The table below specifies an assignment of negative-valued cost functions C that reflect the expenditure of resources required for each step.

WALKTO(α). The monkey walks from its present location to (X(α),Y(α)). The cost is C_w = -1 $\times$ the distance between the present position and the destination.

PUSHTO(α,β). The monkey pushes the object α to the location X(β),Y(β). C_p = -10 $\times$ the distance the object is pushed.

CLIMB(α). The monkey climbs the object α. C_b = -20.

CONSUME(α). The monkey consumes the food α. C_c = -5. The cost of the CONSUME action does not include the utility U_e of achieving the goal because, in general, an action may have several outcomes. The contribution U_e is a property of the outcome, not of the action.

Using this model, the total utility of the symbolic plan WALKTO, PUSHTO, CLIMB, CONSUME is: $U_{total}=C_w+C_p+C_b+C_c+U_e$. The utility of the next-best plan is U_{db}.

2.1.2 Comparing Alternative Plans

The plan with greatest utility can be selected for execution: it is the "best" of the plans generated by the symbolic problem-solver, as evaluated by the utility model. To illustrate the power of comparing plans, consider generating plans using each of the four boxes shown in the "map" of Figure 2-1. Table 1 shows the total utilities of the WALKTO, PUSHTO, CLIMB, CONSUME plans using the different boxes. The plan to use box B has the greatest utility and is therefore selected as the best plan.

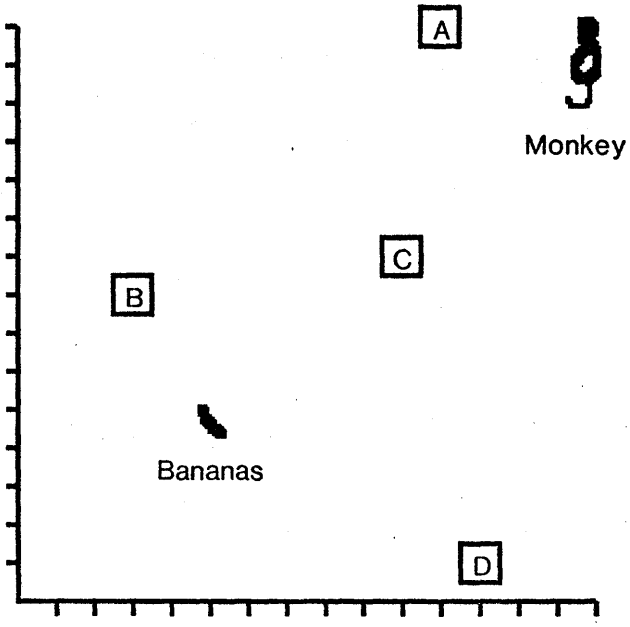

Figure 2-1. Map showing a hypothetical location of the monkey, the bananas, and four boxes. The axes indicate an (x,y) coordinate system. Which box should the monkey push under the bananas and mount to reach the food?

Table 1

	C_w	C_p	C_b	C_c	U_e	U_{total}
Box A	-3	-117	-20	-5	200	55
Box B	-13	-36	-20	-5	200	126
Box C	-6	-64	-20	-5	200	105
Box D	-13	-81	-20	-5	200	81

This simple utility model adds considerable capability to the problem-solver. The location of the boxes and the cost functions determine which box is selected as the best one to use. For example, if $C_p = C_w$, box C will be preferred rather than B. If the initial position of the monkey changes, different boxes may be preferred. Figure 2-2a shows a map of regions in which the monkey might start out, together with the preferred box in each region.

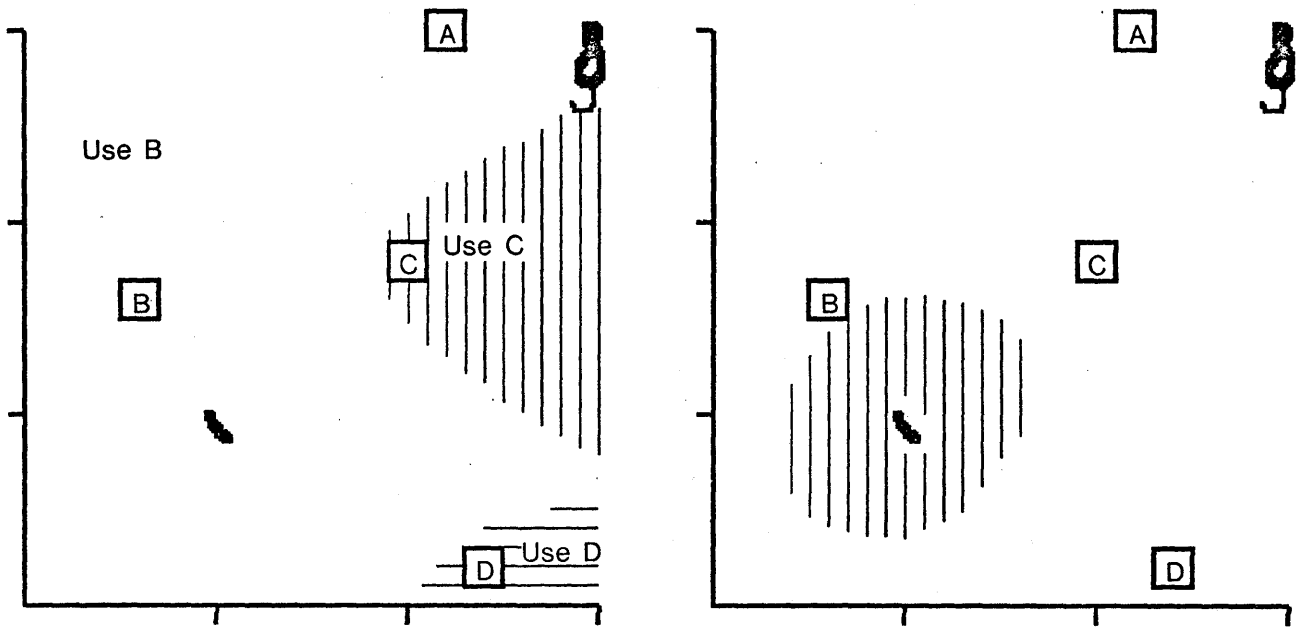

Figure 2-2. (a) Regions of box preference are shown based on the initial location of the monkey. (b) The shaded region indicates where a box preferred to B must lie. Both maps are made assuming C_p=-2Xdistance.

The cost functions also provide answers to a number of questions that an intelligent strategist must pose. For example, when should one try to find a box of higher utility than any presently located, and where should one search? Figure 2-2b shows the region in which a box preferred to B would have to lie.

Another important class of strategic questions concerns what decision theorists call "sensitivity analysis:" how much confidence can be placed in the identification of the best plan? Is it substantially better than the next best, or do the utilities show that the planner is nearly indifferent to the choice? Do slight inaccuracies in the map or model cause a substantial change in the choice of best strategy? We shall later return to these important questions.

2.1.3 Coping with Uncertainty

Execution of a plan can go awry and produce outcomes considerably different from the desired goal. Clearly the reliability of a plan must be incorporated into the calculation of its utility. Decision theory shows how to weight the utility of an outcome with its probability of occurrence and thus to calculate a total utility that expresses the consequences of possible failures.

Let us augment the monkey and bananas problem by introducing a simple kind of failure: *There are two kinds of boxes in the room: wooden and cardboard. Cardboard boxes will not support the monkey; wooden ones will.* When the plan outline is applied to a box of unknown type, either the box is wood and the monkey succeeds in eating, or it is cardboard and he fails. This possibility is modeled as an unreliable CLIMB step (see Figure 2-3). In the absence of more precise information about the box to which the plan is applied, we shall use a single probability p_o to express the likelihood that the box is wooden. In addition, we shall assign a utility to the failure outcome. A simple assignment is simply U_{db}, corresponding to abandoning the quest for food. However, failure-recovery plans of higher utility may exist: a plan to clear away the destroyed cardboard box and to try using another box may have a higher utility than U_{db}. Techniques for devising failure recoveries will be more fully developed below; we shall temporarily assume the utility of the failure outcome to be U_{db}.

The utility of the plan is calculated as the mathematical *expectation* of the utilities of the individual outcomes, i.e., $U_{total} = \Sigma\ p_i\ U_i$, where U_i is the total utility of a particular path in the "decision tree," and p_i is the probability of taking the path ($\Sigma\ p_i = 1$). For Figure 2-3; the total utility U_{total} is $p_o(C_w+C_p+C_b+C_c+U_e) + (1-p_o)(C_w+C_p+C_b+U_{db})$.

This technique allows the planner to trade off cost and reliability; classical AI problem-solvers have no means of expressing these tradeoffs. For example, if we use the costs of Table 1 and assume an identical p_o for all boxes, no change occurs in the selection of the best plan. However, if the probabilities differ for various boxes, a reliable plan may be preferred to a less-reliable one. For example, if p_{oc} is the probability that box C is wood, p_{ob} that of box B, and $p_{oc} > p_{cb} + .11$, the expected utility of using box C will be greater than that of using box B, even though a plan to use B is always preferred in the absence of failure.

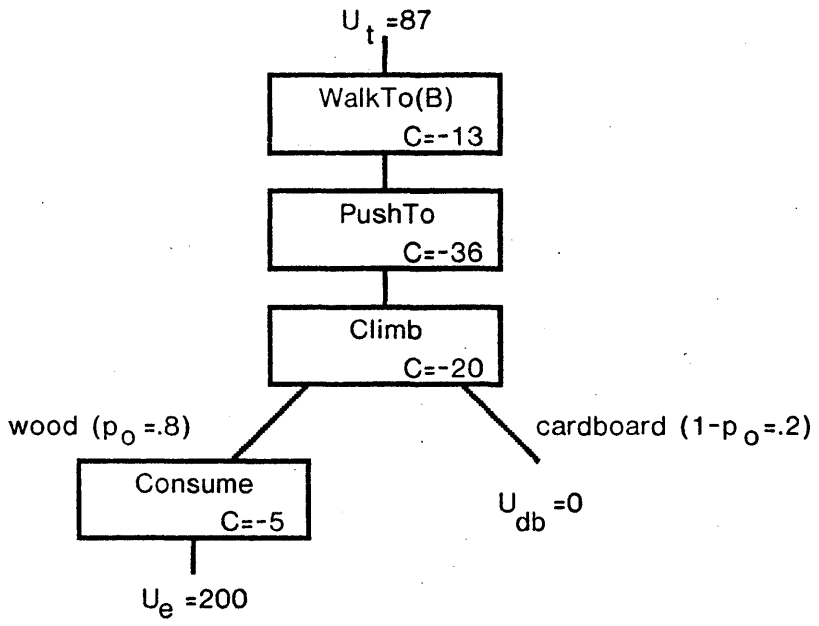

Figure 2-3. A decision tree showing a plan in which the CLIMB step fails with probability $1-p_o$.

The expected utility is a numerical measure of the merits of the *strategy* expressed by the plan. It does not predict that executing the plan will have an outcome of comparable utility, but predicts only the *average* utility of outcomes of many executions. Thus, if we use the *expected utility* as a measure when searching for good plans, we do not guarantee good outcomes, only good strategies.

2.1.4 Finding Good Plans

Because the utility of a plan can be used to compare the merits of competing plans, it can be used to guide a search for good plans. The basic idea is to search by expanding paths of greatest expected utility. A number of algorithms have been devised that can use numerical measures to guide such a search (see surveys in Nilsson, 1971, and Lawler and Wood, 1966).

Using numerical measures to guide search is not new to AI. Many game-playing programs employ a numerical score to represent the desirability of a board position and to guide a search. In fact, a game-playing program that uses a

plausible move generator and a numerical evaluation of progress toward a win is a simple example of a combination of symbolic and *ad hoc* utility models in problem-solving. Robotics problem-solving programs (e.g., STRIPS in Fikes and Nilsson, 1971) have also used simple numerical measures, such as the number of operators in a plan, as a search guide.

Searching can be guided in several ways; we shall illustrate "progressive deepening" and "pruning" as examples. The A^* algorithm (Hart, Nilsson, Raphael, 1968) is typical of a progressive-deepening approach: a non-terminal node, N, of a search tree is expanded if it lies on the most promising path. The measure of promise is an estimate of the utility of the complete plan, computed as the sum of two terms: *g*, a measure of the utility contributions ascribed to the nodes already included in the path (i.e., the total cost[1] of the steps from the root node to N), and *h*, an upper bound estimate of the utility of a path from N to the goal. These terms for the monkey and bananas example might be:

g(node) = $\Sigma\ C_i$ from root to node
h(node) = (upper-bound estimate of costs from node to the goal) + U_e

The calculation of *g* requires calculating the contribution to the utility of the steps of the partially complete plan. It is for this reason that we have formulated our utility model as a sum of terms attributable to individual steps of the plan. The estimate used for *h* can be based on a simple "state-difference" approach. For example, if, at node N, the monkey is not located at the bananas, then an upper bound on *h* is the cost of moving the monkey to the bananas.

Progressive deepening uses a running estimate of the path utility to guide application of further planning effort. One advantage of this technique is that it will automatically attenuate the processing of plans that loop: such plans are abandoned because, as steps are added to the plan, *g* decreases continually without an offsetting increase in *h*.

"Pruning" is characteristic of several kinds of search algorithms that avoid exploring portions of the tree because the optimal plan can be shown to lie elsewhere. Many algorithms in use in operations research, known generically as "branch and bound" algorithms, have this property. The basic idea is to ignore

---

[1]Note that the utility contributions of steps that consume resources are negative. Thus an upper bound on the utility is one for which resource expenditure is least.

paths that have an upper bound on their utility that is less than the utility achievable by some other path. A similar technique for mini-max trees is called "alpha-beta," and has been extended for use with decision trees (Nilsson, 1971, and Slagle and Lee, 1971).

The key information that guides pruning is the *bounds* information: the tighter the bounds the more pruning. Bounding the utility of a plan such as that of Figure 2-3 requires bounding the utility of the part of the plan that is incomplete, the failure path. Bounding the failure path is equivalent to bounding the utility of the possible recovery strategies. One way to do this is as follows:

Lower bound: Don't bother with the current goal, and assign utility U_{db} to the failure. Thus the lower bound of the entire plan is $U_{lower} = C_w+C_p+C_b+p_o(C_c+U_e)+(1-p_o)U_{db}$.

Upper bound: Assume that the failure caused no damage, and that there is an alternative plan as good as the present one. (Note: Given that a good plan fails, we do not have to assume that there exists a recovery plan *better* than the original one, because that will be covered by cases involving other boxes.)

$$U_{upper} = C_w+C_p+C_b+p_o(C_c+U_e)+ (1-p_o)U_{upper}$$
$$U_{upper} = (C_w+C_p+C_b)/p_o +C_c+U_e$$

If we perform these calculations for all boxes, as in Table 1, we find:

Table 2

Box	U_{lower}	U_{upper}
A	16	20
B	87	109
C	66	82
D	42	52

This bounding scheme shows clearly that utilities of plans involving boxes A, C, or D cannot exceed even the lower bound on using box B. Thus portions of the tree that call for boxes A, C and D to be used are pruned.

The numerical utility model thus furnishes information that is useful in guiding search. This information, whether encoded in cost functions or in bounding schemes, can easily involve "domain-dependent" information, as exemplified by our

assignment of a function of distance to the cost of walking. The symbolic model also constrains search: the symbolic preconditions are used to avoid searching foolish plans, e.g., ones that require reaching for the bananas when the monkey is not "AT" the bananas. These techniques can be implemented in the new AI languages (see survey in Bobrow and Raphael, 1974) by instantiating each subgoal pursuit as a separate process and including bounds estimates and costs when proposing new subgoals. A branch and bound algorithm, such as A^*, can then schedule the processes (subgoals), always executing the most promising subgoal. Such dynamic allocation of effort to problem-solving processes motivated the design of the SAIL multiple process structure (Feldman, *et al*, 1972).

2.2 Improving the Plan

In this section, we shall focus on improvements that can be made to a plan prior to its execution. A plan outline often can be altered to yield a greater expected utility by making detailed, often local, improvements to the plan. The measure of improvement in this *plan elaboration* process is the increase in expected utility resulting from filling in details.

A plan can be improved by developing plans to recover from failures in the original outline. The failure in Figure 2-3 can be elaborated with steps to clear away the mess, choose an alternative box, and try to use it to reach the bananas. Such an elaboration is shown in Figure 2-4; the expected utility of the plan has risen from 87 to 96 as a result of the elaboration. The increase occurs because the plan to deal with the failure of box B (i.e., to try again with box C) has a higher utility than that of giving up (45 vs 0). This process can be carried on indefinitely, but if the probability of failure is fairly low, the cost of additional planning may exceed the slight improvement in expected utility. (Generating plans for recovering from failures is similar to the generation of the original plan: a symbolic problem-solver can provide plan outlines; the alternative recovery strategies are compared with utility measurements.)

Elaborating failures tightens the bounds on a plan. Figure 2-4 has a set of bounds shown in brackets as a triple: lower bound, expected value, and upper bound. Bounds are assessed from bottom to top; the triple with a prime symbol is calculated using the U_{upper} formula, then the effects are propagated up through the tree. This process yields a rather tight bound on the utility of using box B (c.f. Table 2).

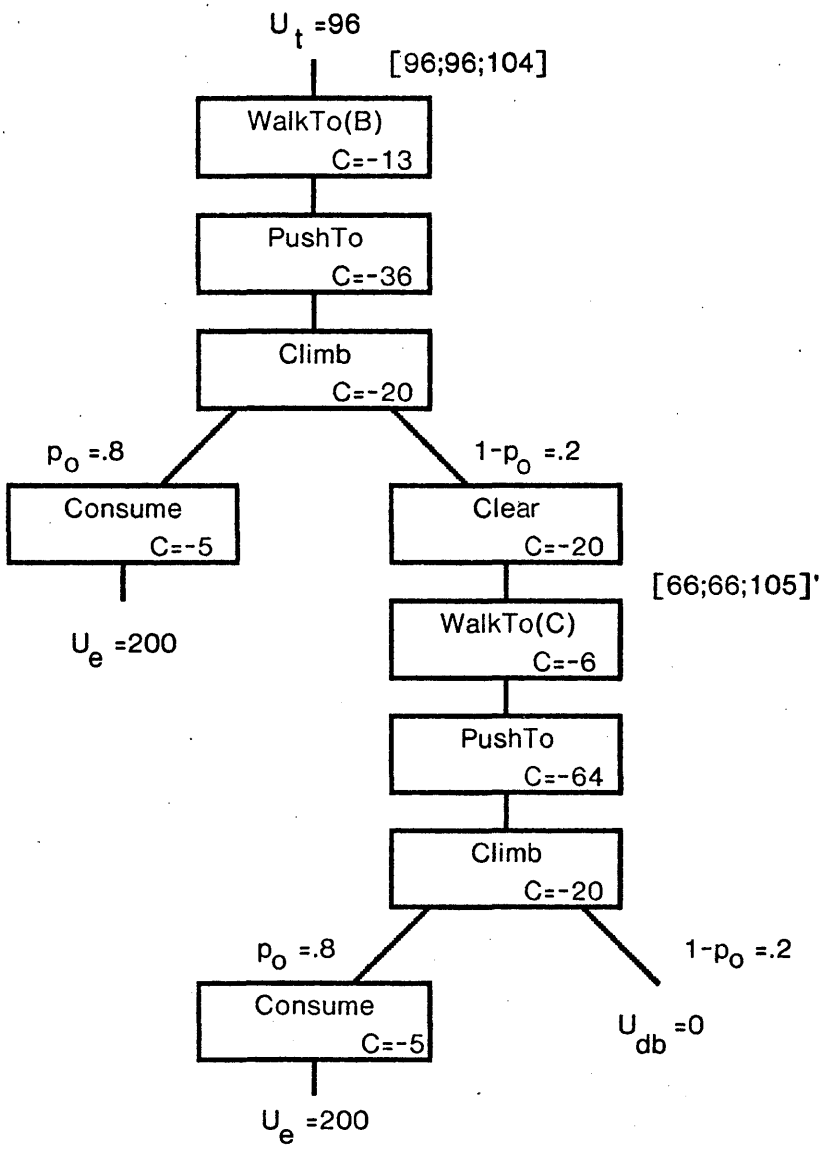

Figure 2-4. The failure of the CLIMB step is elaborated with a recovery strategy. This has the effect of increasing the utility of the plan to 96 (compared to 87 in Figure 2-3). The triples in brackets represent the lower bound, expected value, and upper bound of the plan.

Another kind of plan improvement can be achieved by introducing steps in the plan to gather information, and thereby to reduce the uncertainty in the outcome. A simple example is shown in Figure 2-5: a perfect and costless test determines whether each box is wooden. If the test announces that a box is wooden, which happens with probability p_o, then the plan to use that box is guaranteed to be successful. If the test announces that a box is cardboard, the

next best plan is tried, and so forth. Adding these tests causes U_{total} to rise to 121.

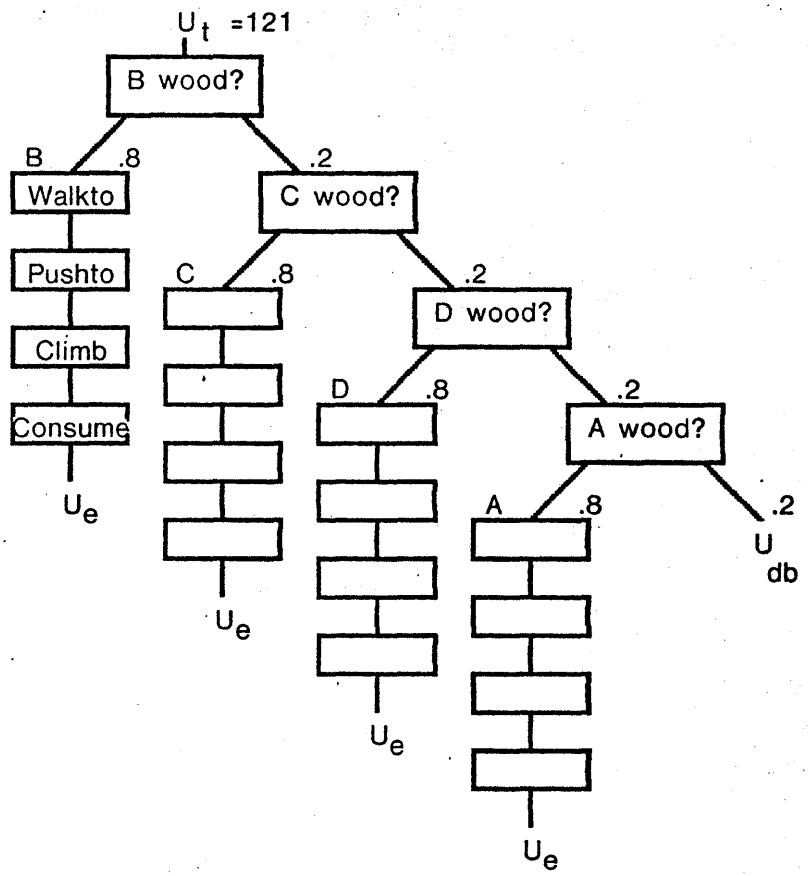

Figure 2-5. Four perfect and costless tests for wooden boxes are inserted into the plan. This illustrates that case when we know only that each box has an independent probability of .8 of being wooden.

A more realistic model of such information gathering accounts for the expenditure of resources required to perform the test and for the possibility that the test gives an incorrect answer. We shall define two such tests that can be used to elaborate the monkey and bananas plan:

TEST-FAR: A visual test measures whether a box is wooden. It does not require that the monkey be located near the box. It has a cost C_{tf}. The answer is characterized by two conditional probabilities p_{fw} and p_{fc}:

p_{fw} = Pr{test announces "wood" | box is wooden}
$1-p_{fw}$ = Pr{test announces "cardboard" | box is wooden}

p_{fc} = Pr{test announces "wood" | box is cardboard}
$1-p_{fc}$ = Pr{test announces "cardboard" | box is cardboard}

If the test always yields correct answers, $p_{fw}=1$ and $p_{fc}=0$.

TEST-NEAR: This test is analogous to TEST-FAR, but the monkey must be

at the same location as the box being tested. This test might involve "thumping" the box. Cost C_{tn}. The behavior is characterized by:

p_{nw} = Pr{test announces "wood" | box is wooden}

p_{nc} = Pr{test announces "wood" | box is cardboard}

If the test always yields correct answers, p_{nw}=1 and p_{nc}=0.

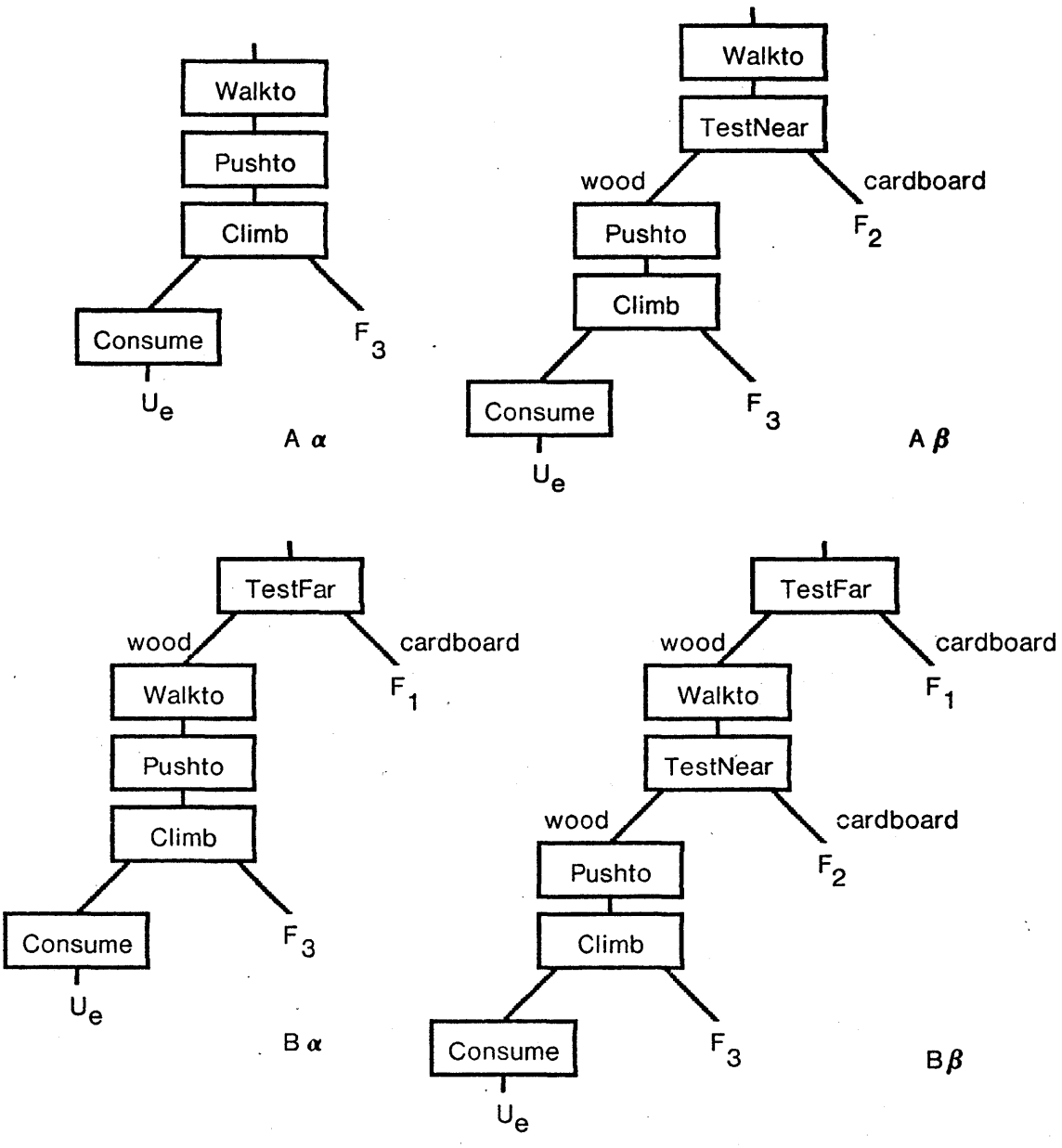

Figure 2-6: Decision trees for the four possible strategies using TEST-NEAR and TEST-FAR.

Adding these tests to the plan outline produces the four strategies shown in Figure 2-6. In order to calculate the expected utilities of these plans, and thereby to choose the best one, we must describe the consequences of performing a test. We shall use a simple Bayesian model: a test causes a change in the probability that the tested box is wooden, according to Bayes' rule:

$$\Pr\{\text{box is wood} \mid \text{TEST-FAR announces "wood"}\} = \frac{p_{fw}\,\Pr\{\text{box is wood}\}}{p_{fw}\,\Pr\{\text{box is wood}\} + p_{fc}\,(1-\Pr\{\text{box is wood}\})}$$

$$\Pr\{\text{box is wood} \mid \text{TEST-FAR announces "cardboard"}\} = \frac{(1-p_{fw})\,\Pr\{\text{box is wood}\}}{(1-p_{fw})\,\Pr\{\text{box is wood}\} + (1-p_{fc})\,(1-\Pr\{\text{box is wood}\})}$$

In these equations, Pr{box is wood} is the estimate of the probability that the box is wooden before the test is performed (the *a priori probability*), and Pr{box is wood|TEST-FAR announces ...} is the estimate derived from the answer to the test (the *a posteriori probability*). Analogous relations hold for TEST-NEAR.

We also need to calculate the probabilities of taking each of the two paths that emanate from the TEST operation:

$$\Pr\{\text{test announces "wood"}\} = p_{fw}\,\Pr\{\text{box is wood}\} + p_{fc}\,(1-\Pr\{\text{box is wood}\})$$

$$\Pr\{\text{test announces "cardboard"}\} = 1 - \Pr\{\text{test announces "wood"}\}$$

As an example of these calculations, we shall evaluate the expected utility of the Bα strategy of Figure 2-6 applied to box B, with the prior probability of finding a wooden box, p_o, set to 0.8, the performance of TEST-FAR characterized by C_{tf}=-20, p_{fw}=0.9 and p_{fc}=0.1, and the failure utilities U_{F_1} and U_{F_2} set to 0. For each of the three paths through the tree, we must calculate the probability the path is taken and the utility of the path:

Path	U_i	Path probability
TEST-FAR, F_1	-20	$\Pr\{\text{test announces "cardboard"}\} = 1-(p_{fw}p_o + p_{fc}(1-p_o)) = .26$
TEST-FAR, WALKTO, PUSHTO, CLIMB, F_3	-89	$\Pr\{\text{box is cardboard}\mid\text{test announces "wood"}\} \times \Pr\{\text{test announces wood}\} = [1-(p_{fw}p_o)/(p_{fw}p_o+p_{fc}(1-p_o))] \times .74 = .02$
TEST-FAR, WALKTO, PUSHTO, CLIMB, CONSUME, U_e	106	$\Pr\{\text{box is wood}\mid\text{test announces "wood"}\} \times \Pr\{\text{test announces "wood"}\} = [(p_{fw}p_o)/(p_{fw}p_o+p_{fc}(1-p_o))] \times .74 = .72$

The expected utility is $EU = \Sigma\, p_i U_i = 69$. Similar calculations for all four

strategies, applied to box B, are recorded in Table 3. For the given set of costs, utilities and probabilities, strategy $A\beta$ is selected. The strategy can be improved still further by elaboration to cope with the failures F_2 and F_3, as described above. Using both methods, a strategy with expected utility 105 turns out to be optimal.

Table 3

Strategy	Expected utility
$A\alpha$	87 (c.f. Figure 2-3)
$A\beta$	88
$B\alpha$	69
$B\beta$	63

Parameters:

C_w=-1 × distance	C_{tf}=-20
C_p=-10 × distance	C_{tn}=-10
C_b=-20	p_{fw}=.9
C_c=-5	p_{fc}=.1
U_e=200	p_{nw}=1
U_{F_i}=0	p_{nc}=0
	p_o=.8

Comparison of the four strategies shown in Figure 2-6, showing that strategy $A\beta$ has the greatest expected utility.

In addition to providing information, a test may also cause a change in state. For example, a medical test may present some risk to the patient: the outcome may worsen the patient's condition in addition to providing diagnostic data. Or it may have cumulative toxic effects, as in the case of tests that require X-ray exposure.

The model of testing reveals tradeoffs among various information-gathering strategies as differences in utility. If the insertion of tests in a plan causes the expected utility of a plan to rise, the test is providing information that helps reduce the uncertainty of the outcome. Decision theory calls this increase in utility the "value of information."

If different strategies have nearly identical utilities, as do $A\alpha$ and $A\beta$ in Table 3, the planner might announce indifference between the strategies, and perhaps use other methods to decide which one to pursue. Such small differences may be insignificant when uncertainties in the probability or utility models are taken into account. Although we may in principle reduce these errors by refining the model, we shall always be faced with insignificantly small differences.

Elaborations cause the search space to grow quite large because of the various choices of inclusion and exclusion of tests, the increased number of failures that require recovery strategies, etc. The search would be wholly impractical without a guide such as the branch and bound algorithm. We shall address below other methods of combatting the "combinatorial explosion" during elaboration.

2.3 World Model Acquisition

The planning activities described in previous sections have assumed that the planner begins with a complete model of the world. Because acquiring such a "world model" and locating all the boxes is a sizeable task, an efficient strategy for feeding the monkey must make efficient allocation of resources to build the model.

A decision-theoretic model of the acquisition process can express the cost and reliability of an acquisition operator and the utility and probability of locating an object in the world. Once again, the utility measure can be used to search for an efficient strategy. A key concept in this approach is that the expected utility of a plan that uses an object, as computed in section 2.1.1, can be used to estimate the value of locating the object.

The vision strategy must decide where to look. For our example, we shall use a grid to divide the world into regions and use a utility calculation to decide which region should be scrutinized. We shall use a simple acquisition operator LOOKAT:

LOOKAT(x,y). Examine the unit square at (x,y) with a vision system to determine if a box lies in the square. The cost of the operator is $C_{x,y}$. The outcome of the operator could be characterized by the two probabilities:

p_{lb} = Pr{LOOKAT(x,y) announces "box"|box at (x,y)}

p_{ln} = Pr{LOOKAT(x,y) announces "box"|no box at (x,y)}

In the remainder of the example, we shall assume $p_{lb}=1$ and $p_{ln}=0$.

In addition, we shall require *a priori* estimates of the probability that a box lies in a square, Pr{box at (x,y)}. The utility of looking at a square is thus:

$$U_{look,x,y} = C_{x,y} + Pr\{box\ at\ (x,y)\}\ U_{box,x,y} + (1 - Pr\{box\ at\ (x,y)\})\ U_{fail}$$

where $U_{box,x,y}$ is the utility of using a box found at square (x,y), which is estimated by evaluating the utility of a plan outline (e.g., Figure 2-3) without

elaboration. The $U_{look,x,y}$ values are calculated for all squares, and the square with the largest value is chosen as the best place to look for a box. The vision plan can also be elaborated. If the LOOKAT operator fails to locate a box, we might apply the LOOKAT operator to another square, and so forth (Figure 2-7). This is just like coping with failure in CLIMB -- we chose an alternative.

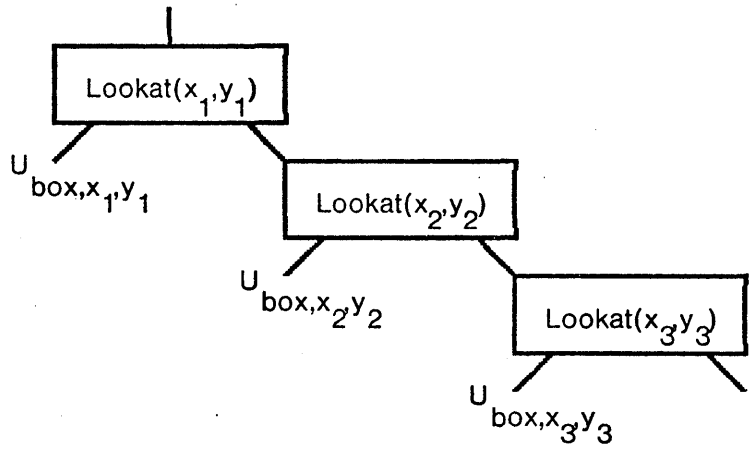

Figure 2-7. Elaboration generates a sequence of LOOKAT operators that examines additional squares until an object is located.

The results of the LOOKAT operation change information in the world model. If a box is located, it is recorded in the model. In all cases, the probability that a box is located in the scrutinized square, Pr{box at (x,y)}, is modified. This is analogous to the treatment of TEST-NEAR and TEST-FAR: Bayes' rule is used to update Pr{box at (x,y)} just as it is used to update Pr{box is wood} as a result of the TESTs. This means that once a square is looked at and found not to contain a box, it will probably not be tested again.

The *a priori* values for the Pr{box at (x,y)} are supplied by a function that can contain considerable information about the world. If boxes are more common in the garage than in the house, this can be expressed in the probability assessments.

Because acquisition operators change the world model, the results can cause widespread changes to the utilities of current plans. We could, in principle, model an acquisition operator with a large number of outcomes and generate

plans for each contingency although a large number of eventually useless plans would result. A mechanism to control the amount of planning ahead and to permit periodic re-evaluation of plans is clearly needed. The next section addresses this topic.

2.4 The Trinity: Look, Think, Act

At some point, the planning operations sketched in the previous sections must be halted and the best plan actually executed. In fact, planning must be severely limited, lest resources be wasted in any of numerous ways such as generating detailed plans for paths that are never encountered or planning without adequate world model information or pursuing complicated elaborations that increase plan utilities only slightly. However, if planning is curtailed, we must be able to resume planning later on.

What is needed is an efficient scheduling of planning, looking and acting. The scheduler decides in some way which activity is most beneficial at the moment, grants it a resource quantum, and then repeats. A natural quantum for looking and acting is execution of one of the "operators" such as LOOKAT or WALKTO. A natural quantum for planning might be one iteration of a branch and bound algorithm, or the addition of one elaboration to a plan. Although allocating effort to planning can at worst cause wasted effort, allocating effort to action has considerably more import.

The decision to plan or to execute can be made with a utility measure. We compare the utility of looking (i.e., executing a step in the best information-acquisition plan), acting (i.e., executing a step in the best action plan), or additional planning (i.e., elaborating existing plans with branch and bound as a guide, or developing more symbolic plans). Unfortunately, specifying a utility function that reflects the benefits of future planning is quite difficult.

> Decision theorists have addressed a problem called "cost of analysis," which is loosely related to the notion of planning cost used here (Matheson, 1968). In a practical analysis, the cost of building a model and assessing probability and utility values is often large enough to invite the question: what is the utility of developing a model with a certain amount of detail? An estimate of this utility can be derived by modeling a set of initial tests of varying cost that give different sorts of information about prevailing probability distributions. The tests and their costs correspond to the various analysis choices.
>
> In our case, planning is the application of the model to a particular situation, which may involve substantial symbolic reasoning, tree expansion, etc. We desire simply to discount the value of a partial plan by the cost of the processing required to generate the details needed for execution.

A simple *ad hoc* approach can be used to *limit* planning activities by specifying a stopping criterion. The difference between the upper bound and expected utilities of a plan for vision or action is a limit on the improvement in the plan that infinite planning would achieve. We have the choice between executing the plan as it stands, and receiving, on the average, the expected utility, and spending more effort planning and receiving, *at most*, the upper bound on the plan. Comparing the utilities of these two alternatives, we have:

Execute:	$U_{execute} = U_{plan\ expectation}$
Plan with cost C_{plan}:	$U_{plan} \leq C_{plan} + U_{upper\ bound}$

If $U_{plan} > U_{execute}$, we choose to plan. This requires that the planning effort be constrained: $C_{plan} \geq (U_{plan\ expectation})-(U_{upper\ bound})$. Thus the additional planning effort is limited by the difference between the upper bound and the expected utility. Obviously this is a crude approximation and could be refined.

This approach essentially compares the risk of the current plan (as estimated by the difference between the upper bound and the expectation) with the cost of further planning. It does not attempt to predict the actual value of planning, but rather measures the cost and maximum value of planning steps. It would certainly be better to use the expected value of the benefits of planning if this quantity could be computed; Chapters 3 and 4 take up this topic.

The main loop of the system plans until such a stopping criterion is reached, and then either looks or acts, whichever has the greater utility. Then the process repeats. The outcomes of looking or acting are, of course, recorded and cause adjustments in the utilities of various available plans. This mixing of planning and acting is a uniform framework for providing "monitoring" and "verification" functions in current robotics systems (Munson, 1971; Grape, 1973; Bolles, 1976).

Techniques for control of planning activity such as those presented here cannot be borrowed from decision theory. Conventional decision analysis is performed by an individual, who uses human judgement in allocating effort to the analysis. Formal techniques are used for modeling or for evaluating decision trees, but not for controlling the planning process itself.

2.5 Summary

This chapter has presented an introduction to decision theory in an AI problem-solving context. The utility calculations of decision theory have been applied to a number of planning needs:

- Comparing alternative plans. Often these comparisons are made among *instances* of a common plan outline or skeleton.
- Coping with uncertainty. The calculation of expected utility permits the reliability of a plan to be considered when assessing its worth.
- Finding good plans. The utility of partially-completed plans, together with bounds on the utility of the completion required, can be used to organize and guide a search for the best plan.
- Improving a plan. Failure paths in plans can be elaborated with recovery plans. The increase in expected utility of the elaborated plan measures the effect of generating a plan to cope with the failure path.
- Acquiring information. Utility measures can be used to plan efficient strategies for acquiring information needed for further planning.
- Allocating resources among planning, acting or information-gathering activities. The utility measure can also be used to decide which activities are most beneficial to the system.

Many of these techniques will appear in the PEGASUS implementation described in Chapter 3 and will be considered in more detail in Chapter 4.

Chapter 3
PEGASUS, The Travel Planner

In order to explore the effectiveness of an approach combining decision theory and symbolic planning, a computer program to formulate certain travel itineraries has been developed. This chapter presents the external behavior and internal operation of the program, PEGASUS, named after "surface" transportation that flew. This chapter serves as a case study of the application of the techniques sketched in Chapter 2. Further extensions of the techniques, based on the experience of the PEGASUS program, are given in Chapter 4.

The chapter presents explanations of PEGASUS at several levels of detail. Section 3.1 is devoted to examples intended to show PEGASUS's reasonably realistic model of travel and the sorts of itineraries generated. Section 3.2 provides a first explanation of the program structure, and emphasizes the hierarchical planning techniques incorporated in PEGASUS. The section includes a discussion of the allocation of planning effort (section 3.2.3). The remainder of the chapter is devoted to more detailed explanations of PEGASUS's model (section 3.3) and the processing undertaken in the various levels of abstraction (section 3.4).

3.1 Generating Travel Itineraries

The problem statement consists of an origin location, a destination location, and a collection of constraints that restrict the acceptable times of travel:

```
From:        HOME (Palo Alto, California)
To:          ROC (Rochester, N.Y. airport)
Constraint:  LEAVE HOME AFTER (JUL 13 8:00) timeDecay=10 value=100
```

A good deal of information is available to the program about geographical locations (36 spots), scheduled conveyances (102 bus trips, 338 airplane trips, 30 train trips), and unscheduled conveyances (taxi, rental car, walking). Given the problem statement and some computing effort, an itinerary is proposed:

```
-> Itinerary # 1. Utility: 660.58
TAXI. From HOME to SFO. Departure JUL 13 9:38 transit time 0:22.
Layover at SFO starting JUL 13 10:00 lasting 0:05.
AA 182. From SFO to ORD. Departure JUL 13 10:05 transit time 3:44.
Layover at ORD starting JUL 13 15:49 lasting 0:41.
AA 500. From ORD to ROC. Departure JUL 13 16:30 transit time 1:24.
Arriving at destination JUL 13 18:54.
```

Listed with the itinerary is its *utility*, computed from the itinerary and an internal utility model that expresses the value of the plan in terms of the resources required to complete it:

- Time. The transit and layover times are considered to be "costs" of the trip. Other things being equal, trips with shorter elapsed times are preferred to those with longer elapsed times.

- Money. Cash expenditures for fares on the various conveyances are also considered to be costs of the trip. Other things being equal, inexpensive trips are preferred to expensive ones.

- Stress. Various events in the course of a trip may contribute to the traveler's stress. Examples are running to make a tight connection, having to change planes in the middle of the night, or using a mode of conveyance that, for whatever reason, is abhorrent to the traveler. Other things being equal, trips that impose low stress will be preferred to those with higher stress.

- Miscellaneous. Contributions to the utility that do not fall into one of the three categories above are included in this "resource."

The utility function aggregates these "resource requirements" for a trip into one metric. Revealed in the function is the way that tradeoffs are made: how should a plan that is inexpensive but slow be compared to one that is expensive but fast? For present purposes, we shall assume that the function is linear:

$$U = - C_t \times \text{Time} - C_m \times \text{Money} - C_s \times \text{Stress} - C_x \times \text{Miscellaneous} + \text{constant}$$

Because the coefficients may vary among travelers, PEGASUS provides facilities for altering them and reevaluating the plan.

```
-> Utility function modifications.
Coefficient for money: 1.0 -> OK
Coefficient for time: .33 -> .05
Coefficient for stress: .1 -> OK
-> Itinerary # 1. Utility: 772.24
TAXI. From HOME to SFO. Departure JUL 13 9:38 transit time 0:22.
Layover at SFO starting JUL 13 10:00 lasting 0:05.
AA 182. From SFO to ORD. Departure JUL 13 10:05 transit time 3:44.
Layover at ORD starting JUL 13 15:49 lasting 0:41.
AA 500. From ORD to ROC. Departure JUL 13 16:30 transit time 1:24.
Arriving at destination JUL 13 18:54.
```

Just as the utility function contains an assessment of certain of the traveler's preferences, so must the specification of the time constraints. The constraint "leave home after 8:00" might be an expression of mild objection to arising

absurdly early. However, "leave Stanford after 15:00" might reflect required attendance a class that does not end until 15:00. Consequently, a constraint requires two parameters in addition to the plan: first, how important is the strict observance of the constraint (value); second, how firm is the time (timeDecay).

Each itinerary also contains "backup" suggestions in case something goes wrong with the planned itinerary. If a flight is cancelled, or if the traveler misses a flight, the backup may be invoked. In any case, some estimate of the backup alternatives is required to assess properly the utility of the main plan. The itinerary mentioned above has two backups:

```
-> Itinerary # 1. Utility: 772.24
TAXI. From HOME to SFO. Departure JUL 13 9:38 transit time 0:22.
Layover at SFO starting JUL 13 10:00 lasting 0:05.
AA 182. From SFO to ORD. Departure JUL 13 10:05 transit time 3:44.
Layover at ORD starting JUL 13 15:49 lasting 0:41.
AA 500. From ORD to ROC. Departure JUL 13 16:30 transit time 1:24.
Arriving at destination JUL 13 18:54.

Backup at SFO (Utility: 787.9):
 Layover at SFO starting JUL 13 10:00 lasting 0:15.
 UA 126. From SFO to ORD. Departure JUL 13 10:15 transit time 3:55.
 Layover at ORD starting JUL 13 16:10 lasting 1:30.
 UA 794. From ORD to ROC. Departure JUL 13 17:40 transit time 1:30.
 Arriving at destination JUL 13 20:10.

Backup at ORD (Utility: 789.5):
 Layover at ORD starting JUL 13 15:49 lasting 1:51.
 UA 794. From ORD to ROC. Departure JUL 13 17:40 transit time 1:30.
 Arriving at destination JUL 13 20:10.
```

If the itinerary proposed by the system is unacceptable, some aspect of the problem statement or of PEGASUS's model of the traveler must be erroneous. The user may alter the problem statement (e.g., by altering the statement of constraints) or certain of the system parameters (e.g., the coefficients of the utility function). After any such changes, the solution is reassessed. In the following example, the traveler decides to place more emphasis on leaving early. The solution utility falls:

```
-> Itinerary # 1. Utility: 772.24
 ...
-> Constraint editor:
--> Add a constraint: Leave HOME before JUL 13 9:00 timeDecay=.1 value=10
-> Itinerary # 1. Utility: 660.72
 ...
```

If the traveler adopts an itinerary, he may wish to make reservations on various flights. He can record the success or failure at making a reservation; the system keeps abreast of progress:

```
-> Itinerary # 1. Utility: 660.72
 ...
-> Trip editor:
--> Modify AA 182. From SFO to ORD. Departure JUL 13 10:05,
    to include reservation in class: Y
-> Itinerary # 1. Utility: 661.0
 ...
```

The utility of the final plan is increased somewhat because reservations increase the probability that the traveler will be able to follow the plan successfully. The utility would rise even more if a reservation on the second leg of the trip were secured.

To this point, PEGASUS has demonstrated only the ability to formulate and refine a plan, and to re-evaluate the plan if the user changes utility assessments and the like. Of equal importance is the ability to *act*: to instruct the traveler which flights to take, and to process new information as the trip proceeds. Relevant information concerns cancelled trips, delayed trips, observations of traffic delays, weather, and so forth.[1]

```
-> Now is: JUL 13 9:00
-> Observations.
 Property: WEATHER
 Spot: ORD
 New value: 4 (very bad)
-> Itinerary # 1. Utility: 750
 ...
```

As the traveler progresses to ORD, the plans are updated. The originally planned leg becomes:

```
-> Itinerary # 1. Utility: 732
 AA 500. From ORD to ROC. Departure 16:30 transit time 1:24.
 Arriving at destination 18:54.
```

There is, however, a plan with higher utility:

```
-> Itinerary # 6. Utility: 781
 AA 524. From ORD to ROC. Departure 21:30 transit time 1:22.
 Arriving at destination 23:52.
```

---

[1]Unfortunately, this sort of information only subtly affects common travel itineraries. The examples I have chosen, therefore, are somewhat extreme.

The reason that this flight has greater value is that the model of weather used by PEGASUS expects the exceptional weather conditions to regress toward the normal weather for ORD at this time of year.

The system can assimilate several sorts of information that concern the execution of specific itineraries:

Departure delays
Arrival delays
Weather reports
Traffic delays
    Take-off delays due to traffic
    Landing delays due to traffic
Load-factor information
Modifications to constraints
Modifications to the utility function

Acquiring information of this sort is not always easy--the traveler may have to walk to information desks, make telephone calls, etc. Ideally, the importance of a piece of information and the expense of obtaining it should be considered when deciding what information to gather. PEGASUS provides a simple form of this calculation, and answers the question "What is it worth to inquire about Boston weather?" or "What is it worth to make a reservation?" A "value of information calculation" can be made:

-> Itinerary # 1. Utility: 787.7
AA 500. From ORD to ROC. Departure JUL 13 16:30 transit time 1:24.
Arriving at destination JUL 13 18:54.
-> Itinerary # 2. Utility: 783.9
UA 794. From ORD to ROC. Departure JUL 13 17:40 transit time 1:30.
Arriving at destination JUL 13 20:10.
-> Value of weather at ROC measured on JUL 13 16:10 is .1.
(If ROC has bad weather, itinerary #2 will be selected because
the weather model expects the weather to be better later.)

Note that the value of weather information is very small, because the prevailing weather in ROC in July favors successful completion of any flight. If the same trip were attempted on January 13, the value of weather information rises to 11.9!

3.2 A First Explanation

This section describes briefly the techniques used to implement the travel itinerary planner. The behavior of this program, demonstrated in the previous section, is achieved with a combination of heuristic and decision-theoretic

techniques. The important ingredient introduced in this section is the hierarchical nature of the planning in PEGASUS, and the structure of the symbolic and utility models that make it possible.

3.2.1 Primitives

The PEGASUS program constructs *itineraries.* An itinerary is a tree of *steps* corresponding to the primitive "actions" that a traveler can be expected to undertake (e.g., "Take United 424 from SFO to ORD."). Each step has certain essential information with it:

- Origin and Destination (often referred to as "from" and "to"). These label the segment of the trip represented by the step; both origin and destination are *spots.* A spot may be a train station, bus depot, airport, city, home, place of business, etc.

- Conveyance (often called the "step type"). This entry specifies the generic class of conveyance used to achieve this step, e.g., AIR, TRAIN, WALK.

- Resource Vector (often called "factors"). This vector lists the quantities of various resources that will be required in order to execute the step, and is denoted by the boldface symbol **r**. The elements of the resource vector are:

  - Time. This is a measure (or estimate) of the elapsed time required to complete the step, measured in minutes. In certain cases, it includes an "interface" time required before the succeeding step may commence.

  - Money. This is a measure of cash expenditures required, usually for fares.

  - Stress. This is a measure of a hypothetical human resource: the reserves of an individual to cope with situations that strain the person. The contribution of a step to this quantity may vary from person to person, and includes such things as: stress caused by hurrying to make a connection, by changing planes in the middle of the night, by taking "red-eye" flights, etc. Another contribution is the "stress of time," which expresses preferences for time spent in various spots or on various conveyances. For example, waiting an hour at

HOME may have a higher utility than waiting 30 minutes in a bus terminal.

- Miscellaneous. This is a catchall for contributions to the utility of the step that do not fit the three other factors. Contributions to this factor will be explained as needed.

- Starting time, duration. If itinerary planning has progressed to the point where departure and arrival times are chosen, these two entries record the choices for this step. All times are recorded in minutes with respect to a fixed reference time.

When other information needs to be associated with a step, it is placed on a property list attached to the step description. Examples of such information are constraints and pointers to records that describe a particular trip in detail.

The utility of an itinerary is computed as a function of the resources required to execute the itinerary. This involves computing the resources **r** required for each step of the itinerary, summing the resources of the individual steps to calculate the resources required for the entire itinerary, and finally applying the utility function U(**r**) to the sum.

3.2.2 Program Structure at a Glance

The growth of good itineraries is managed by a progression of processing at different levels of detail. The first level is used to make only very crude plans; the last level molds an itinerary with the most detailed analysis of which PEGASUS's model is capable.

A level can be visualized as a single routine and various subroutines for performing the processing. The routine is responsible for creating a *task* to solve a specific problem or subproblem, for executing the task to analyze and solve the problem, for eventually returning one or more *solutions* to the problem, and for processing miscellaneous inquiries and imperatives relating to the task. When a level is presented with a new problem, it creates a task to search for solutions to the problem, and returns solutions as they are uncovered in the course of the search. The search task itself proceeds incrementally, being allocated computing time by a control program. A level is thus a program that creates tasks, or coroutines, which are periodically allocated computing time to solve their appointed problem.

The overall organization requires that each level must report solutions in "best-first" order, using an upper bound on the utility of the solution as a measure of "best." We shall return later to examine the implications of this requirement.

A solution from one level is presented to the next level for analysis in more detail. The lower level will use the solution from the higher level as a guide in performing its search, but will be concerned chiefly with adding to the planned itinerary new detail that was not considered at higher levels. As we shall see, a single "problem" often gives rise to several solutions. These percolate through the levels of processing, becoming alternative strategy choices at each level.[2]

Before describing the control of tasks in more detail, we shall present a capsule description of the processing at each level in PEGASUS.

1. TOP. The TOP level is given the problem presented by the user: the origin and destination for the trip. The purpose of this level is to consider several primary means of conveyance and to compute an upper bound on the utility of all itineraries that can be constructed using such a primary means. For example, if the problem is to go from HOME (Palo Alto, California) to ROC (Rochester, N.Y. airport), we might generate 5 answers:

Solution 1: AIR	U^*=774.3 (!)
Solution 2: RENTAL CAR	U^*=-592.4
Solution 3: BUS	U^*=-667.4
Solution 4: TAXI	U^*=-1969.6
Solution 5: WALK	U^*=-29619.5

(The answer labeled (!) is passed to the next level in our example.)

The upper bound for AIR is calculated by assuming that both origin and destination are on the transportation graph connected by AIR (i.e., that both are airports), and that a direct flight is available that flies as fast as the fastest flight known in the data base, and is as cheap (per mile) as the cheapest flight

[2]The hierarchical task-oriented design of PEGASUS is related to other AI problem-solving systems, and was inspired in part by NOAH (Sacerdoti, 1975). See section 5.2 for further discussion of the relation to other AI work.

known. Upper bounds for other conveyances are calculated using similar reasoning.

Our initial presentation of PEGASUS will assume that the best answer at each level is passed on to the next level for more detailed processing. The solution thus promoted is labeled with (!) in the list above. In fact, PEGASUS uses a more complicated control structure to pursue alternative solution paths.

2. SP. The SP level conducts a shortest-path search on the graph formed from spots and known trips that use the specified conveyance. In our example, TOP passed down the solution "Go from HOME to ROC by AIR." Since HOME is not on the graph of spots connected by AIR (i.e., it is not an airport), SP puts in a dummy step, and *assumes* that there is a route from HOME to a nearby airport. There are many possible routes that SP ultimately returns. The first few are:

Solution 1:	
HOME-[assume]-SJC-ORD-ROC	U^*=700.6 (!)
Solution 2:	
HOME-[assume]-SFO-DTW-ROC	U^*=699.5
Solution 3:	
HOME-[assume]-SFO-ORD-ROC	U^*=698.4
Solution 4:	
HOME-[assume]-OAK-ORD-ROC	U^*=696.7
Solution 5:	
HOME-[assume]-SMF-ORD-ROC	U^*=657.0
...	

The three-letter codes are standard abbreviations for airports: SJC is San Jose, ORD is Chicago O'Hare, DTW is Detroit, SFO is San Francisco, OAK is Oakland, SMF is Monterey.

The upper bound is calculated by assuming that each leg takes the minimum time of all applicable trips in the data base, that the fare is the cheapest of all possibilities, that the stress is the least, etc. Notice that the utility of the best plan at this level is less than that of the answer at level TOP. Several factors account for this: (1) HOME is not an airport, which means slower and more expensive means are required to journey from HOME to an airport; (2) No non-stop trips are available, which means that minimum travel time is longer than anticipated in TOP; (3) The cheapest available fare is not as cheap as estimated in TOP, and (4) The "stress of time" contributions reduce the utilities according to the traveler's relative preference for various airports and conveyances.

3. INSTANTIATE. This level is responsible for choosing specific instances of trips described in the data base. In the example, specific flights from SJC to ORD and from ORD to ROC must be selected. In addition to the route passed from SP, the user's constraints are taken into account. For the first time, "interface" steps (called WAIT steps) are inserted between legs of a trip to represent time spent in layover waiting for a connection or waiting to embark, etc. The most promising flight selections returned by INSTANTIATE are:

Solution 1:
HOME-SJC: [assume], departing 7:01
SJC: wait 30 minutes
SJC-ORD: UA 464, departing 7:45, arriving 13:30
ORD: wait 55 minutes
ORD-ROC: UA 362, departing 14:25, arriving 16:52 U^*=687.9 (!)

Solution 2:
HOME-SJC: [assume], departing 12:21
SJC: wait 30 minutes
SJC-ORD: UA 356, departing 13:05, arriving 19:00
ORD: wait 150 minutes
ORD-ROC: AA 524, departing 21:30, arriving 23:52 U^*=651.8

Solution 3:
HOME-SJC: [assume], departing 7:01
SJC: wait 30 minutes
SJC-ORD: UA 464, departing 7:45, arriving 13:30
ORD: wait 180 minutes
ORD-ROC: AA 500, departing 16:30, arriving 18:54 U^*=644.0

Solution 4:
HOME-SJC: [assume], departing 7:01
SJC: wait 30 minutes
SJC-ORD: UA 464, departing 7:45, arriving 13:30
ORD: wait 250 minutes
ORD-ROC: UA 794, departing 17:40, arriving 20:10 U^*=616.5

...

Again, the upper bounds at this level are less than in SP: (1) Waiting times have increased the total time required to complete the trip; (2) Because specific flights have been chosen, the cheapest fare is now the cheapest that can be arranged given the constraint that the pair of flights is used; (3) Various stress contributions associated with scheduled times may have appeared; (4) The WAIT nodes have introduced more "stress of time" contributions due to relative preference of the traveler for flying or waiting in an airport.

4. DOLLARS. This level considers all possible fares that might be charged in order to pursue the itinerary passed down from INSTANTIATE. In our example, the cheapest fare is a coach through fare ($156). If, for some reason, a coach

seat is unavailable on one leg, there are coach/first class combinations, and so forth:

Solution 1:
SJC-ORD: UA 464, Coach
ORD-ROC: UA 362, Coach (through fare) U^*=687.9 (!)

Solution 2:
SJC-ORD: UA 464, Coach
ORD-ROC: UA 362, First (no through fare) U^*=655.9

Solution 3:
SJC-ORD: UA 464, First
ORD-ROC: UA 362, First (through fare) U^*=640.9

Solution 4:
SJC-ORD: UA 464, First
ORD-ROC: UA 362, Coach (no through fare) U^*=631.9

5. FILLIN. This level fills in the details of any legs of the trip that still have no executable plans, i.e., are marked as *assuming* the leg can be achieved. In our example, the leg from HOME to SJC requires such work. The FILLIN processing is accomplished by a recursive call on the entire planning system. We get:

Solution 1:
HOME-SJC: TAXI, departing 7:01
SJC: wait 30 minutes
SJC-ORD: UA 464, departing 7:45, arriving 13:30
ORD: wait 55 minutes
ORD-ROC: UA 362, departing 14:25, arriving 16:52 U^*=681.8 (!)

Solution 2:
HOME-SJC: RENTED-CAR, departing 6:46
SJC: wait 30 minutes
SJC-ORD: UA 464, departing 7:45, arriving 13:30
ORD: wait 55 minutes
ORD-ROC: UA 362, departing 14:25, arriving 16:52 U^*=673.2

Again, the upper bound on utility has decreased because a taxi is not both the cheapest and fastest ground transportation available.

6. PROB. This level interprets an itinerary in light of all probabilistic information that is available. For example, the probabilities of missing a connection, of cancelling a flight, and of airports closing due to bad weather are all included in the assessment of the plan. PROB also optimizes the departure times of unscheduled conveyances such as taxis to achieve the maximum utility.

HOME-SJC: TAXI, departing 7:16
SJC: wait 15 minutes
SJC-ORD: UA 464, departing 7:45, arriving 13:30
ORD: wait 55 minutes
ORD-ROC: UA 362, departing 14:25, arriving 16:52 U^*=681.8
E U=650.9

Here, the upper bound, U^*, remains undiminished compared to the answer returned from FILLIN, because the plan does not differ from that returned by FILLIN if all aspects of the plan proceed smoothly (the upper bound assumption). However, the *expected utility* is somewhat lower because the failure paths in the itinerary will require additional resources to journey to the destination.

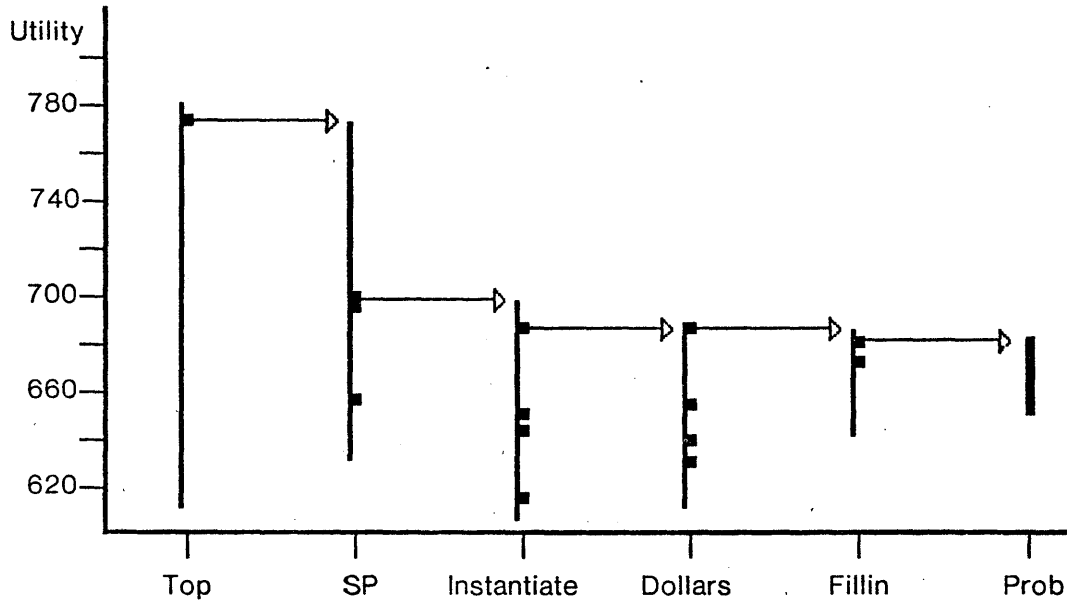

Figure 3-1: Graph summarizing the progress of plans through the six hierarchical levels of processing and the drop in utility that accompanies more detailed analysis. Each dark square represents a solution. In our overview, we have assumed that the best solution within each level is passed on for more detailed processing at the next level (arrows). The heavy dark line in the PROB level represents the range between the upper bound and the expected utility.

This completes the processing of the itinerary. The progress through the several levels is summarized in Figure 3-1, which demonstrates the propagation of solutions through the six levels of processing and the fall in utility as more detailed planning is performed.

3.2.3 Control of Planning Effort

PEGASUS requires a scheme to allocate planning resources that goes beyond the simple propagation of "best" solutions demonstrated in the example. Although this procedure will rapidly yield a complete plan, it is not guaranteed to be the optimal one, or even a very good one. Figure 3-2 shows an example of the propagation of plans in which a plan judged inferior by SP turns out to be the best.

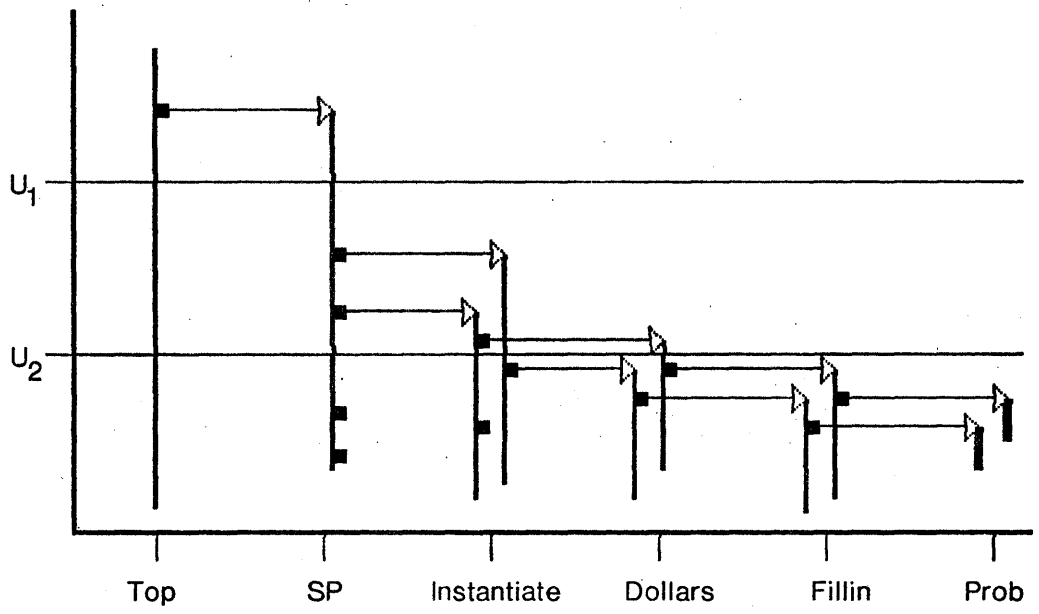

Figure 3-2: A collection of problems in various stages of processing in PEGASUS. Each square represents a solution; its propagation to the next level is represented by an arrow. The two lines U_1 and U_2 show two cuts of uniform utility through the processing. Notice that the most attractive solution to the SP level does not in fact yield the best detailed plan.

PEGASUS needs to propagate other solutions, in addition to the best at each level, and needs to allocate planning resources to a number of tasks in various stages of planning at various levels. This allocation is, of course, derived from utility estimates of the partially-complete plans being examined by the various tasks.

The upper bound on the utility of plans being generated by a particular planning task is a measure of planning progress. In order to provide proper

controls over planning effort, each task is required to implement two primitive operations:

- Return the upper bound U^* on the utility of the plan (or partially-complete plan) being processed by the task.

- Apply planning effort on this task until the upper bound falls below a parameter provided by the control routine. This primitive may terminate before reaching the stated upper bound if a new solution is uncovered during the processing. The upper bound may only decrease as more planning effort is applied: this stipulation represents a restriction on the way the processing within a level is organized (see section 4.3 for more discussion).

For example, the problem "Go from HOME to ROC by AIR" presented to level SP may require a good deal of computation to answer. We know, before any computing is done, that solutions will have $U \leq 774.3$, the bound established by the TOP level. The routines associated with level SP can be directed to work on the problem until either a solution is generated, or until the current U^* of the problem *falls below* 760. The search for answers must of course be organized so that answers of high utility are explored first, then answers of lower utility, etc. The U^* used to measure the progress of planning simply tracks the falling hopes of the planning undertaken by a task. Figure 3-3 shows an empirical plot of the fall of the utility in processing the SP level of our example as a function of computing time.

Uniform planning. A simple utility-based allocation strategy is to run the task that reports the largest upper bound. We periodically survey all tasks at all levels; each reports its current U^*, the upper bound on itineraries that the approach will yield. Suppose that task i has the largest value of U^*, and task j the next highest (If only one task exists, j is assigned to a dummy task that always has $U^*=-\infty$). Effort is then allocated to i using the primitive described above, until it returns an answer or U^*_i falls below U^*_j. Then the survey and allocation of effort is repeated, etc. This technique has the effect of devoting effort to an approach until it produces a solution or until it ceases to be better than other approaches.

Another way to view this technique is that all tasks are actively competing for

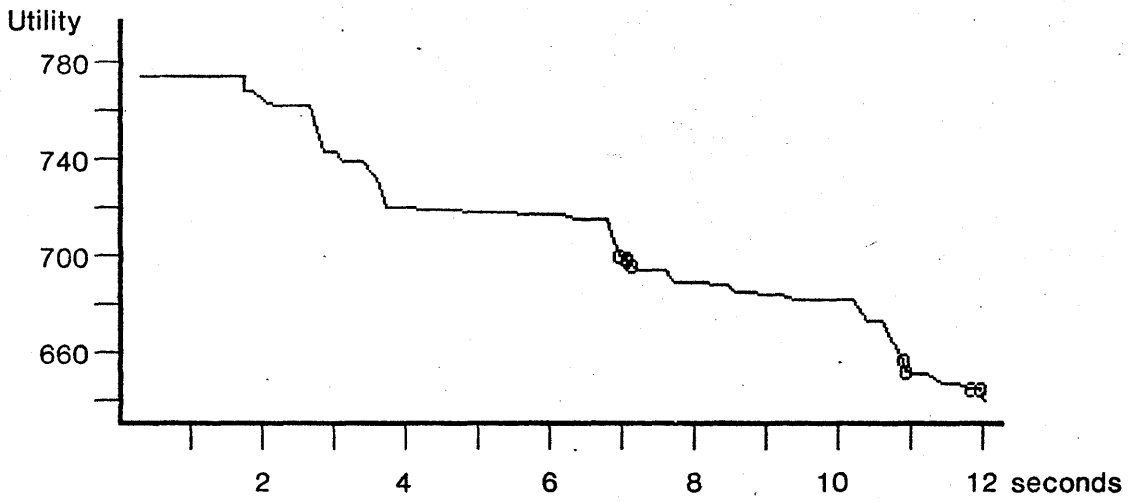

Figure 3-3: An empirical graph of the fall of the upper bound U* as processing proceeds in the SP level. The times were measured for the problem presented in section 3.2.2. Circles represent points at which solutions to the problem are reported (there are four solutions clustered near 7 seconds).

computation resources. The utility measure corresponds to a *priority* in a scheduling algorithm: resources are granted to the task of highest priority until that priority falls below that of some other pending task, or the task blocks waiting for results from some other process (e.g., an input/output process).[3] The key point is that the upper bound on the utility, a single numerical measure of promise of a planning task, can be used to allocate planning resources.

The uniform planning scheme finds the optimum plan. The essentially breadth-first approach will pursue many alternative solutions in the crude levels of the hierarchy before even a single plan is propagated to the final PROB level. When the control strategy attempts to allocate resources to a task that can perform no more planning because it represents a plan already developed in full detail, the planner terminates and announces this plan as the "best" one. The uniform strategy guarantees that there is no other plan with an upper bound that exceeds that of this "best" plan.

Depth-first planning. Although the uniform approach does arrive at the best

[3]The present implementation "polls" the tasks at various levels to decide which has highest priority. Another design might implement each task as an independent process that is periodically checked by a scheduler. The scheduler switches tasks whenever it discovers that the upper bound of the running process has fallen below that of some suspended process.

answer or set of answers according to the utility function on itineraries, these answers may no longer be best when the cost of the computation required to construct them is included in the utility assessment. If computing time was quite expensive, we might use a depth-first approach (used in the example of section 3.2.2): whenever the first solution is produced when processing a task, pass it to the next level, compute on the new task until the first solution emerges, and so forth until a solution is delivered by the most detailed planning level. Because of its depth-first character, this method yields a completely detailed plan quickly, although the plan is not necessarily optimal or even very good (see Figure 3-2).

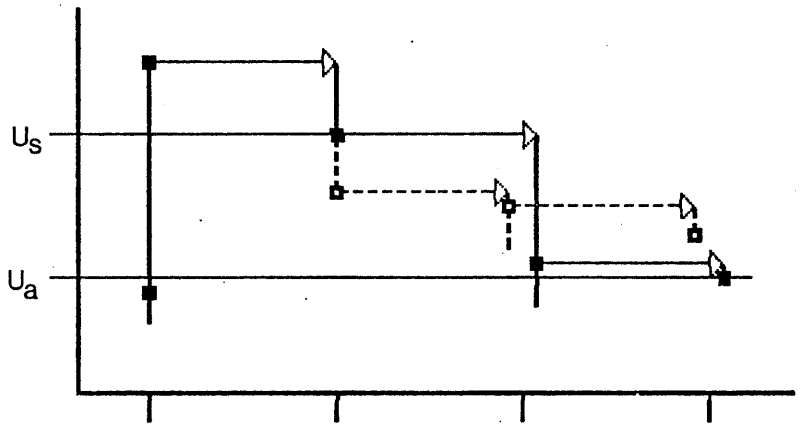

Figure 3-4: After the first solution to the entire problem is generated, more planning may be indicated. In the illustration, the solid lines represent the depth-first method, which yields a solution with utility U_a. At this point, there remain other problems with higher upper bounds; in particular, the second level has not been pursued enough to discover solutions with utility less than U_S. It may happen that such pursuit would uncover a better plan (dashed lines). In any case, we know that the best solution must lie between U_a and U_S.

Hybrid planning. The information contained in the U^* values for partial plans in the suspended tasks at various levels is used to pursue a strategy intermediate between the uniform approach and the depth-first approach. Suppose we apply the depth-first approach until a detailed answer is generated; let this answer have utility U^*_a (see Figure 3-4). Let U^*_S be the maximum of the U^*'s of the suspended tasks. Then clearly, if U^*_o is the utility of the optimal plan, $U^*_a \leq U^*_o \leq U^*_S$. If the user is indifferent to small errors in utility, less than $U^*_S - U^*_a$, then no more planning is required.

Otherwise, the difference between U^*_s and U^*_a bounds the amount of further planning effort we should invest. Let r_c be the resource vector describing computing resources required to complete the planning of the problem that corresponds to $U^*_s = U(r^*_s)$ (r^*_s is the resource vector used to compute the upper bound for problem s). We can compare the worth of the plan, together with the computing effort we devote to it, to the utility of our available plan: as long as $U(r^*_s+r_c) > U^*_a$, we can continue planning. Eventually, increases in both resources r^*_s and r_c will cause the inequality to fail; at this point, we have exceeded the planning bound.

A less crude approximation to the cost of planning can be used more effectively. If, for each partially-planned problem in the level structures, we can estimate the processing resources needed to complete the planning, we can calculate a modified upper bound $U^*_c = U(r^* + r_c)$, where r^* is the resource vector used to calculate the upper bound on the problem in the structure (i.e., $U^*=U(r^*)$), and r_c is the computation resource estimate. For the one solution generated by the depth-first approach, $r_c=0$. This model is a first approximation: it assumes that the investment of processing effort (r_c) will simply add to the detail of the plan without increasing the resources (r^*) its execution requires, and thus decreasing the total U^*. A more sophisticated attempt might involve modeling the expected decrease in U^*.

An estimate of r_c is not hard to generate in this case: the initial processing using the depth-first approach demonstrates the resource requirements of each level while working on planning tasks that are *very similar* to the tasks that may need completing. We simply record the extent of this processing for each level; r_c is then the sum of these requirements for all levels of detail yet to be applied to the plan.

3.2.4 The Central Theme

The preceding paragraphs have outlined the main theme of the PEGASUS implementation:

> Problems move through several levels of processing, giving rise to increasingly detailed plans. Each level summarizes progress on a planning task by computing an upper bound, U^*, on the utility of the plan at this and all subsequent levels of processing. The upper

bound decreases monotonically as the solution methods available to the routines in a level become exhausted. Among other implications of this requirement is that solutions to a problem are reported best-first.

The U^* bound also decreases as a plan enters levels of more detailed analysis. Detailed analysis often takes the form of added constraints or complexities that cast increasing doubt on a previously good plan. The optimistic upper bound is reported to the task scheduler that controls allocation of computing resources to the various competing tasks. However, the more rapidly the (relative) disadvantages of a plan are uncovered by the processing, the more rapidly it ceases to compete for computing resources, and thus permits better plans to be examined more carefully.

This process can be summarized as *finding fault with crude plans by adding detail.*

3.3 The Model

The model of travel used by PEGASUS must not only be realistic, but must also be organized to ease the various kinds of utility and upper bound calculations needed in the hierarchical planning organization. In particular, the utility bounds calculated at different levels of detail, and based on different plan features, must all be consistent. This section describes features of the model not already unveiled, as a preface to a more detailed examination of PEGASUS processing.

We shall occasionally point out details that the PEGASUS model does not confront. Often in these cases the problem is one of implementation scope rather than of limitations of the approach. That is, the decision-theoretic framework for representing and processing information and for expressing tradeoffs in travel planning has much more power than PEGASUS's model actually exploits.

3.3.1 Utility Function

The utility function maps the resource vector (or factors), consisting of measurements of time, money, stress and miscellaneous into a single utility. This function has a linear form with four parameters that can be adjusted to accomodate preferences of different travelers:

Let r = [Time Money Stress Miscellaneous]

$U(r) = [\ C_t\ C_m\ C_s\ C_x\ ] \cdot r + \text{constant}$

The evaluation of U(r) is accomplished by a single function in PEGASUS, so that the form of the utility function can be changed easily. The planning routines in PEGASUS strive to be independent of the form of the utility function, in order to ensure the planning techniques will work properly with more elaborate functions.

Although the operation of PEGASUS does not require that the utility function be linear, it is essential that U(r) be monotonic in the individual dimensions Time, Money, Stress, and Miscellaneous, i.e.,

$$\partial U(r)/\partial \text{Time} \leq 0 \qquad \forall r$$
$$\partial U(r)/\partial \text{Money} \leq 0 \qquad \forall r$$
$$\partial U(r)/\partial \text{Stress} \leq 0 \qquad \forall r$$
$$\partial U(r)/\partial \text{Miscellaneous} \leq 0 \qquad \forall r$$

This property simplifies calculation of the upper bounds U^*. To calculate the upper bound for a trip, we ascribe to each step of the itinerary the *minimum* resources required to accomplish the step, consistent with the constraints imposed by the current level of planning detail. The *minimum* resources are computed independently for each step: the minimum time, minimum money, etc. Let

$$r^*_i = [\ \min_j(\text{Time}_{ij});\ \min_j(\text{Money}_{ij});\ \min_j(\text{Stress}_{ij});\ \min_j(\text{Miscellaneous}_{ij})\ ]$$

where j indexes the alternative ways of accomplishing the i th step; thus Stress_{ij} is the stress resource required to execute the ith step with the jth alternative method. Then

$$U^*(\text{trip}) = U(\Sigma_i\ r^*_i)$$

where r^*_i is given above. More examples of bounds calculations are given in section 4.1.9.

3.3.2 Spots and Conveyances

In order to plan trips, a data base of geographical locations and available conveyances is required. PEGASUS does not attempt to record exhaustively all transportation available in the United States, but rather includes examples.

Spots are geographical locations, with entries to specify latitude, longitude and time zone. At present, 36 spots are known to PEGASUS.

Conveyance information is entered into PEGASUS in a form as it might appear in an airlines guide, and then indexed in several ways. For each path between two spots connected by a particular conveyance, entries are associated with the departure spot; in this way, spots are linked into graphs for each means of transportation. Currently, 338 airplane flights (November, 1973 airlines guide) among 25 cities, 102 bus trips on the San Francisco peninsula, and 30 train trips are included in the data base.

3.3.3 Constraints

What does a traveler mean by the requirement: "I want to leave home after 6:00 on July 1?" PEGASUS adds two complexities to this seemingly simple statement. First, the traveler has an implicit matching constraint: "...but before 17:00." If such a matching constraint is absent, there is an unbounded number of equally good plans that satisfy the constraint. As a convenience, therefore, the system supplies a second constraint, 8 hours later. (The number 8 is an arbitrary choice made by PEGASUS; the traveler could be required to make his wishes explicit.)

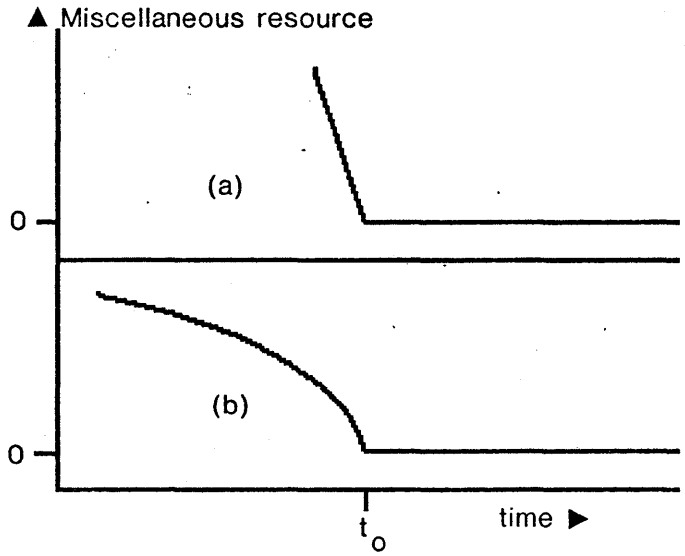

Figure 3-5: Miscellaneous resource contributions of constraints that express a desire to leave after t_0. Note that for $t>t_0$, there is no contribution to the miscellaneous resource. (a) A strong constraint. (b) A weaker constraint.

The other intricacy that PEGASUS attaches to the interpretation of a constraint is a measure of its absoluteness. For example, the traveler may really mean: "I want to leave after 6:00, but if there is a trip that leaves at 5:58, I will consider it to be only a little poorer than one at 6:02." Once again, we can use a utility model to express this preference: we express the effect of a constraint as the contribution to the "miscellaneous" resource. The constraint function, of which two samples are shown in Figure 3-5, is parameterized by severity of the constraint (β) and by a time constant (α):

A constraint is a triple (t_0, α, β)

Let Violated(t) = 0 if ($t \geq t_0$ and $\beta \geq 0$) or ($t < t_0$ and $\beta < 0$)
1 otherwise

Let Limit(t) = α ($|t-t_0|$) /20

$\text{Miscelianeous}_{\text{constraint}}(t)$ = Violated(t) $|\beta|$ [1 - $e^{-\alpha|t-t_0|}$ + Limit(t)]

Here t is the departure (or arrival) time, t_0 is the time mentioned in the constraint, α and β are parameters ($\beta \geq 0$ if the constraint reads "...after..."; $\beta < 0$ if it reads "...before..."). The contributions of a "strong" constraint always exceed those of a "weak" constraint. The function is always positive, with zero the minimum value, which represents no degradation to the utility of a plan due to constraints. Thus planning at high levels that does not include constraint effects can calculate upper bounds U^* by assuming a zero contribution to the Miscellaneous factor from constraint effects.

To determine precisely the shape of the constraint preference function for a traveler is a complex problem in utility assessment. PEGASUS does not address this problem, and simply lets the traveler specify values of β (a critical constraint might use 100, a desirable one 50) and of α (a firm time estimate might have a time constant of 10/minutes, a looser constraint .1/minutes). If the traveler finds that utility assessments made with these choices are not to his liking, he can modify the constraint and re-evaluate plans.

3.3.4 Stress of Time

If the PEGASUS utility model used only the components explained so far, a large class of tradeoffs could not be made by the system. If, for example, two alternative itineraries require the same elapsed time but use different conveyances, the program's model will be unable to distinguish between them, or to express preferences of the form "I prefer going by bus to driving because I

can get work done en route." Similarly, travelers may prefer to spend time in some spots (e.g., waiting time at home between a departure time expressed in a constraint -- leave HOME after 7:00 -- and the departure of an actual conveyance) and not in others (e.g., waiting in the Chicago airport).

The stress of time model simply adds a contribution a*(elapsedTime)+b to the Stress factor, where a and b are derived from information stored with spots and conveyances. The contributions to stress of time must always be positive, for the same reasons explained above for constraint contributions. In order to reflect the user's preferences, it may be necessary to cause a value of a or b to become negative. In this case, it is necessary to add constant terms to a's and b's of all spots and conveyances so that min(a)=0; min(b)=0 in order to preserve the property that the minimum resource contribution is 0. After such an adjustment, the stress contributions for all steps in pending tasks must be recomputed.[4]

This simple stress model is not intended to be accurate; its inclusion is essential if the system is to exhibit reasonable problem-solving behavior in the travel domain. If we were to construct a planner that attempted to organize all personal activities, we would require a rather different utility model. For example, "constraints" as used above are really expressions of the relative preference for time spent traveling compared to time spent in other activities at both ends of the trip.

3.3.5 Distributions

Probability density distributions are used to represent information of various sorts in PEGASUS: departure, transit and arrival times of conveyances, weather, load factors, etc. These all make use of a common representation for distributions. The representation was chosen with the following criteria:

- All calculations required by the system must have closed forms.
- Skew distributions are needed: the probability that an airplane departs 5 minutes early is not the same as the probability that it departs 5 minutes late.

---

[4]Modifying all a's and b's in a large system might be impractical. In this case, we can define two global variables, a_g and b_g, and re-formulate the stress of time contribution as $(a+a_g)$*(elapsedTime)+$(b+b_g)$. The upper bound conditions become $\min(a+a_g)=0$; $\min(b+b_g)=0$, and it suffices to modify a_g or b_g to meet these constraints.

- A continuous form is desirable, in order to permit incremental calculations to be performed. Although discrete approximations might suffice for several of the quantities, a continuous form always admits a discrete interpretation, and not vice-versa.

The following two-tailed exponential density function was chosen:

$$p(x;x_0,\lambda_1,\lambda_2) = [\lambda_1\lambda_2/(\lambda_1+\lambda_2)]\, e^{\lambda_1(x-x_0)} \quad \text{for } x<x_0, \text{ or}$$
$$[\lambda_1\lambda_2/(\lambda_1+\lambda_2)]\, e^{-\lambda_2(x-x_0)} \quad \text{for } x>x_0$$

Parameters: x_0, $\lambda_1>0$, $\lambda_2>0$

Conceptually, this can be viewed as two different exponential distributions, one for $x<x_0$, one for $x>x_0$; the first has decay parameter λ_1, the second λ_2. A normalization constant is chosen so that the entire function integrates to 1. We shall write $X[x_0,\lambda_1,\lambda_2]$ to indicate a random variable with such a density function, i.e.,

$$\Pr\{X[x_0,\lambda_1,\lambda_2]<t\} = \int_{-\infty}^{t} p(x;x_0,\lambda_1,\lambda_2)\, dx$$

The mean of the random variable $X[x_0,\lambda_1,\lambda_2]$ is $x_0-1/\lambda_1+1/\lambda_2$; the variance is $1/\lambda_1^2+1/\lambda_2^2$. Consequently, we can think of translating the loose statement "the value of x is a, with variation -b, +c" into the distribution $X[a-b+c,1/b,1/c]$.[5] We approximate the sum of two such distributions by summing the means and individual variances. A random variable can be multiplied by a scalar:

$$aX[x_0,\lambda_1,\lambda_2] = Z[a\ x_0,\lambda_1/a,\lambda_2/a]$$

These distributions are used to model several quantities:

Weather. The weather condition at a spot is measured on a linear scale from 0 to about 5. Roughly, 0 is the best weather there is, 3 starts to affect driving a car, 4 is hazardous driving and airports beginning to close, 5 closes virtually all airports. Thus if $W[x_0,\lambda_1,\lambda_2]$ describes the distribution of ORD weather, we might let

$$\Pr\{\text{ORD closed}\} = \int_{4.5}^{\infty} p(x;x_0,\lambda_1,\lambda_2)\, dx$$

[5]The argument is that $1/\lambda_1$ is the standard deviation below the mean (i.e, corresponds to c), and $1/\lambda_2$ is the standard deviation above the mean (i.e., corresponds to b).

Load Factors. The ratio of the number of seats filled on a flight to the number of seats on the empty aircraft is called the load factor. If we augment this definition to calculate the number of seats that could have been filled divided by the capacity, then load factors greater than 1 are possible. Hence if $F[x_0,\lambda_1,\lambda_2]$ is the load factor distribution for a flight, then

$$\Pr\{\text{no space on flight without reservation}\} = \int_1^\infty p(x;x_0,\lambda_1,\lambda_2)\,dx$$

Departure and Arrival Times. Times are important quantities to model in order to assess the probability of making a connection. Suppose $D[x_0,\lambda_1,\lambda_2]$ represents the departure time of an airplane. The arrival time **A** at the destination can be viewed as the sum of the random variable **D** with:

D_{delay} Departure delay. After leaving the gate, how long is it until you are airborne? This is really a quantity that absorbs all components of delay due solely to conditions at the departure spot.

T Transit time. Variation here may be due to winds (airplanes) or traffic delays (automobile). **T** is computed by multiplying the nominal transit time by a random variable that describes proportional delay.

A_{delay} Arrival delay. Delay components at the arrival spot, e.g., delay in approach patterns, ground services, etc. are lumped in this quantity.

P_{delay} Processing delay. This delay models the time spent between arrival and the ability of the passenger to choose options freely. If he has baggage, this time is longer than if he does not.

PEGASUS views all these delays as independent, and simply sums the distributions: $A=D+D_{delay}+T*nominalTransitTime+A_{delay}+P_{delay}$. The departure distribution **D** is computed in one of two ways. If we are computing **D** for an initial departure (e.g, the "origin of a flight"), we simply sum the nominal departure time with a starting delay $S[x_0,\lambda_1,\lambda_2]$. If we are computing **D** for

a continuation leg of a multi-stop trip (e.g., the DTW-ROC leg of a flight that flies ORD-DTW-ROC), then the departure distribution must be affected by the arrival time of the previous leg: we set $D=A+B[x_0,\lambda_1,\lambda_2]$, where **B** represents the distribution of time between arrival and actual departure. Note that the distribution **B** usually depends on the difference between **A** and the nominal departure time.

Armed with distributions of arrival and departure times of connecting conveyances, we can compute the probability that the connection can be completed successfully:[6]

$A[x_0,\lambda_1,\lambda_2]$ ==> arrival of flight
$D[x_0',\lambda_1',\lambda_2']$ ==> departure of connecting flight

$$\Pr\{\text{make connection}\} = \int_{-\infty}^{\infty} \Pr\{A<t\}\ p(t;x_0',\lambda_1',\lambda_2')\ dt$$

$$= 1 + (\lambda_1\ \lambda_2\ \lambda_1'\ \lambda_2') \times$$
$$\{e^{-\lambda_1'(x_0'-x_0)} /[(\lambda_1'+\lambda_2')\lambda_1'(\lambda_1'+\lambda_1)(\lambda_1'-\lambda_2)] +$$
$$e^{-\lambda_2(x_0'-x_0)} /[(\lambda_1+\lambda_2)\lambda_2(\lambda_2+\lambda_2')(\lambda_2-\lambda_1')]\ \}$$

These distributions are only useful to PEGASUS because they change, either because different situations give rise to different *a priori* distributions or because information from observations causes the distributions to change.

Whenever an observation is recorded, estimates of a distribution change. Unfortunately, PEGASUS usually needs to compute distributions that apply some time *after* the observation: PEGASUS thus needs a model of how these quantities change with time. Because the quantities mentioned above often change slowly and uniformly, we use an exponential decay to model the effect of observation. If the random variable **X** is observed at t_0 to be x_0, we have at some $t>t_0$,

$$X_t = e^{-\alpha(t-t_0)}\ X[a,0,0] + (1 - e^{-\alpha(t-t_0)})\ X[x_0,\lambda_1,\lambda_2]$$

where $X[x_0,\lambda_1,\lambda_2]$ is the *a priori* distribution that applies at time t; α is typically .005/minute.[7] Thus, two-day old observations of SFO weather carry little weight; two-hour old observations much more. The exponentials are, in effect, blending functions for distributions.

---

[6]Although this expression appears to diverge as λ_1' approaches λ_2, it does not!

[7]In the expression, the observation X[a,0,0] could also be a distribution. Such a distribution might result from applying Bayes' rule to model an imperfect test.

PEGASUS maintains a library of *a priori* and measured distributions that describe the quantities mentioned above. The library is indexed by distribution name (e.g., weather, load factor, A_{delay}) and by spot (e.g., ORD, SFO), and optionally by means (e.g., AIR, TRAIN), by carrier (e.g, UA, AL), by weather conditions, and by time pattern (e.g., day of the week, time of day). The index thus permits us to say "United Airlines load factors between 13:00 and 23:00 Fridays are .8 -.3 +.1," or "Traffic delays leaving ORD from 15:30 Fridays to 18:00 are 15 -2 +5 minutes," or "The weather at JFK at 13:00 July 3 was 2." The library is represented as property lists attached to spots, and is searched to find a match in distribution name, means, carrier, weather and time whenever a distribution is needed. The search will retrieve all relevant information, whether entered initially or as a result of a measurement; if no probability information is associated with a spot, the special spot DEFAULTSPOT is interrogated.

3.3.6 Summary of the Model

This is a convenient point to summarize the components of PEGASUS's travel model. The model is basically dynamic, but is initialized by a static data base consisting of *a priori* measurements and schedules. During planning and execution, the model is altered to reflect the traveler's requirements (e.g., utility function changes) and to reflect current observations.

Utility model: The utility of a plan is computed by applying the utility function (part of the model) to a *resource vector* that summarizes the requirements of the plan. The contributions to the resource vector are:

Time:	Elapsed time in transit
	Layover time
Money:	Fares for transportation
	Computer time for the cost-of-planning model
Stress:	Contributions due to time of travel (e.g., night)
	Stress of time contributions (from spots or conveyances)
Miscellaneous:	Contributions due to constraint evaluation

Probability model: The model permits calculation of the probability that a connection fails because of timing, that a flight will be cancelled (either due to weather problems or mechanical failure), that a passenger will find a seat available on a flight, that a flight will be able to land successfully at its

destination, or that an airport is closed. The underlying quantities used to compute these probabilities are:

Weather (numerical approximation)
Load Factors
Departure delays
Arrival delays
Traffic delays
Processing delays

These quantities have *a priori* distributions recorded that may be altered by observations given to the system.

3.4 The Explanation in More Detail

This section re-examines the level processing of PEGASUS, given an understanding of the fundamental organization and of the model. The foci of the descriptions are:

- How decision theory and symbolic techniques interact in the levels.
- How decisions made at one level constrain processing at subsequent levels.
- How the design is influenced by the necessity for working on a problem incrementally (and quite frequently reporting the U^* of the best plan) and for changing the structure incrementally when updates are made.

3.4.1 Implementation of Planning Tasks

The dominant organizational feature of PEGASUS is the collection of routines associated with each level of processing. Additional modules control planning effort, interact with the user, manage the model data base, and perform resource calculations. The implementations of all the levels share a common protocol for interacting with the module responsible for controlling the planning effort. A planning task is given control with one of the following messages:

PROBLEM. This message invokes machinery to create a new task for processing at the given level. Accompanying the message is a solution extracted from the next higher level of processing. The result is a list structure that represents the new task; the message types listed below can be applied to the new task structure.

UTILITY. This message requests the task to return the resource vector that gives rise to the upper bound on the plans that this task will generate. It is by interrogating tasks with the RESOURCES message that the scheduler identifies tasks that represent promising plans.

WORKON. This message instructs the task to continue planning. Along with the message comes an argument that gives a cutoff utility: the task should plan until the upper bound of the plan falls below the cutoff.

SOLUTION. This message requests the task to return the next solution itinerary, if one is available. Even after the planning controller extracts a solution and passes it on to another level, the original task is retained in case additional solutions will be needed.

REEVALUATE. Whenever information in the model is changed, all tasks are passed the REEVALUATE message, together with a list of changed information, organized into several categories such as: utility function changes, transportation graph changes, specific trip changes, constraint changes, and so forth. It is the job of each level's reevaluation processing to modify the task representation incrementally to reflect the changed situation.

ADVANCE. When a step of an itinerary is executed, and the traveler "advances" to a new state, tasks are told to alter their structures accordingly to reflect the new situation. In practice, this reduces to stating a new starting spot and a new constraint that requires departure from the spot to be later than the "present" time. Consequently, ADVANCE requires treatment similar to REEVALUATE. The tasks representing paths not taken by the traveler remove themselves from consideration by arranging to respond to the UTILITY query with an extremely low utility.

PRINT. The task is instructed to print its state for debugging purposes.

The representation of a task is a LISP record that contains, in effect, bindings of variables that describe the state of the planning in the task. When a task is given control, additional bindings are of course made as functions are entered, but the bindings in the instance record alone survive from one activation of the task to the next. This theme represents a weak form of the ideas of Smalltalk (Goldberg and Kay, 1976).

InterLisp-10 (Teitelman, 1975) was chosen for PEGASUS implementation, in part because of excellent program-development facilities. At the outset, it was not clear whether one of the new AI languages would help or hinder combining AI and decision theory ideas in one program. Most of these new languages have sparse facilities for controlling search effort, a lack that I surmised to be crucial when trying to use the utility function to guide search. The chief disadvantage of not using an underlying "language" is that the boundary between concepts of general applicability and of specific application to the travel problem is obscured.

3.4.2 Levels

1. TOP. When the TOP level is presented with a problem, it retrieves a list of all conveyances known to PEGASUS and proceeds to calculate an upper bound on using each conveyance as the *principal* conveyance to solve the problem. It does so by building single-step itineraries using *idealized conveyances* as the means of transportation. The idealized conveyance is a fictitious conveyance that is similar in performance to the real one, but without many of the constraints. Thus AIR* (the final asterisk indicates an idealized conveyance) is a conveyance that can fly non-stop from anywhere to anywhere; it is as fast and as cheap (per mile) as any AIR flight known to PEGASUS, and departs whenever necessary without delay.[8] This "conveyance" thus has a resource vector that is less (dimension by dimension) than the resource vector for any achievable itinerary using AIR transport; hence the utility function applied to this vector yields an upper bound on any AIR itinerary.

TOP makes a list of the steps using idealized conveyances and sorts it by U^*. TOP is always capable of returning a SOLUTION, the next entry on the sorted list. Re-evaluating is merely a matter of applying the utility function again to all elements of the list and re-sorting it.

2. SP. (Shortest Path) The SP level performs an A^* search of the graph of conveyances determined by TOP to be the principal conveyance: TOP thus constrains the search space for SP. The measure of path length used to guide the search is, of course, a utility measure.

---

[8]Although various kinds of transportation are used as examples for discussion, the examples are typical of a wider class of transportation. In most processing of itineraries, PEGASUS distinguishes only between scheduled and unscheduled modes of transportation. Here, for example, AIR is an exemplar of AIR, TRAIN, or BUS.

If the origin or destination are not on the conveyance graph, SP connects them to the graph with ASSUME steps. ASSUME is hypothetical ground transportation that achieves the best performance of any ground transportation known to the system (i.e., has the highest utility, evaluated in the sense of the TOP evaluations). Figure 3-6 shows a part of the search tree generated by SP for our example problem (HOME to ROC).

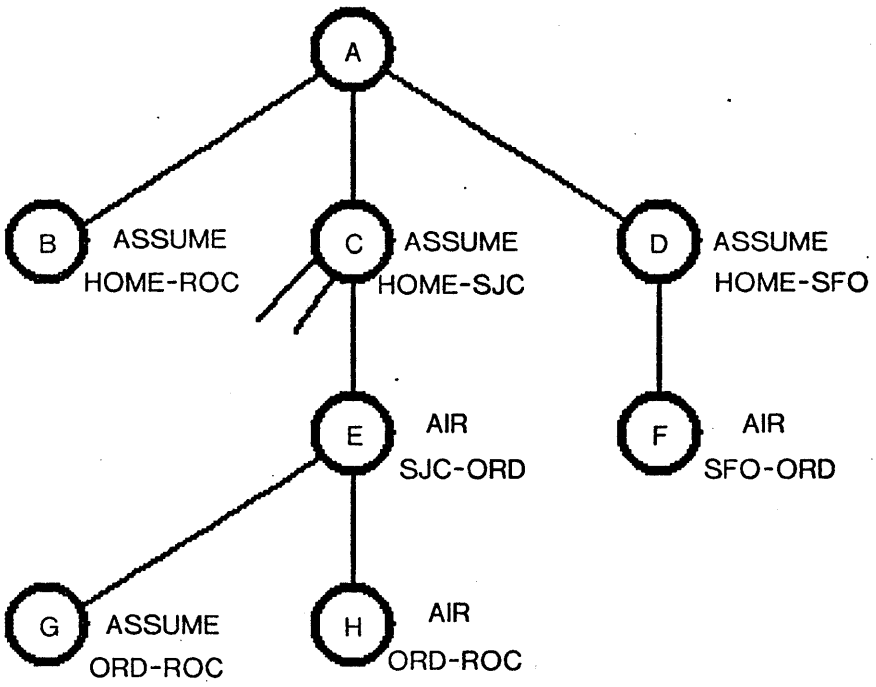

Figure 3-6: A portion of the SP search tree to find a route from HOME to ROC.

SP calculates the promise of each partial path as $U^*=U(\Sigma r_p + r_h)$, where r_p are lower bounds on the resource vectors needed for steps on the path, and r_h is a lower bound on the resources required to complete the path. U^* is therefore an upper bound on all itineraries that have prefixes equal to the part of the path already planned. As an example, consider the path D,F in Figure 3-6. The resource vector for D is computed by the ASSUME hypothesis given above: r_D = [22.8; 1.46; 0; 0] (Recall that the resource vector is [Time; Money; Stress; Miscellaneous]). The resource vector for F is computed from the minimum fare charged on an SFO-ORD air trip and the minimum time that any plane takes to make the trip: r_F = [250; 101; 0; 0]. The resource vector r_h is just the lower

bound on an air route from ORD to ROC, computed with the AIR* idealized conveyance: r_h = [93.4; 28; 0; 0]. Finally we calculate the bound on the utility of the partial plan by evaluating $U(r_D+r_F+r_h)$. This calculation is simplified by storing, with node F, the vector sum of resources required by F and its ancestors in the tree.

SP maintains a list, L, of nodes that are incompletely expanded, sorted so that the partial path with largest U^* is first on the list. When the control program requests work on SP, the following steps take place (simplified):

Let N(*conveyance,from,to*) be the first node on L. L is initialized with a dummy node N(dummy,goalfrom,goalfrom), where goalfrom is the traveler's starting point.

If N's U^* is less than the U limit allocated to the task, return;

else if *to* is the goal, remove N from L and return the answer;

else if *to* is not on the graph of conveyances requested, find several nearby spots that are, add them as successors to N: N'(ASSUME,*to,newspot*), and remove N from L;

else find a path from *to* by AIR that has not yet been considered, say from *to* to *newto*, and add as successor to N: N'(AIR,*to,newto*). Sort this new path into L by its U^* calculation. If no such paths from *to* by AIR remain, add a successor to N: N'(ASSUME,*to,goalto*), and remove N from L.

Repeat the entire process.

This loop permits planning to be suspended after each new node is added in the search. At any time, the U^* of the first node on L is the measure of planning progress on the problem.

Re-evaluation is a straightforward matter of updating the tree and L. It will be required if the utility function changes, if stresses of time change, or if arcs of the conveyance graph are added or deleted. Re-evaluation also causes all solutions previously returned to be re-evaluated. If a solution depends upon an arc in a transportation graph that has been deleted, it is marked invalid. An invalid solution behaves as if it was never generated: tasks at lower levels that depend on it are ignored. When the traveler advances on his trip, all paths but one leaving node A are pruned and corresponding nodes are removed from L.

The new constraints imposed by SP are the symbolic requirements that cause steps of an itinerary to be chosen from a set of existing transportation paths and to be joined consistently (i.e., the origin of a step is the same as the destination of the preceding step).

3. INSTANTIATE. The INSTANTIATE level chooses particular airplane flights (or bus trips, train trips, etc.) for steps in the route returned by SP. The constraints stipulated by the user, as well as a data base of scheduled trips, are used to formulate the new itinerary.

The route computed by SP is used as a template by INSTANTIATE. The "strongest" constraint is identified, and used to establish *departure time estimates*, using the minimum transit times computed by SP. These guesses, which will aid computing upper bounds during the INSTANTIATE search, are almost certainly unachievable, as they assume that planes fly arbitrarily frequently. The first step of the template that can be scheduled (e.g., SJC to ORD by AIR) is called the *template pivot.*

The search proceeds along two dimensions: the choice of a flight to use for the pivot step, and the choice of flights after the pivot (given the pivot choice). The situation for our examples is shown schematically in Figure 3-7. Because the strongest constraint reads "...after...," pivots are chosen with increasingly later departures (increasingly earlier departures are chosen if the strongest constraint is "...before..."). Given the pivot choice, the remaining steps of the itinerary are filled in by a search through successively later connecting trips, as shown by dashed lines in the figure.[9] The remainder of our discussion of INSTANTIATE explains how this search is controlled and limited.

Upper bounds on partial plans must be calculated to guide the searching. A partial plan is scored by referring to the template: the portion of the itinerary not yet planned is assumed to proceed smoothly, without time delays. For example, suppose node A_0 had just been generated (i.e., A_1 and A_2 do not yet exist). Since A_0 has us arriving in ORD at 13:30, 61 minutes later than the template guess, the part of the template corresponding to the unplanned part of the itinerary, i.e., T_2, is evaluated with all times 61 minutes later than in the original template. If, for example, T_2 has a constraint associated with it of the form "arrive in ROC before 18:00," this evaluation will penalize later choices of the pivot. After nodes A_1 and A_2 have been generated, A_0 must still be scored

[9]This search may appear to be of little use: once a good connecting flight is found, why look for others? But it may be that the traveler is very cost-conscious, and the cheapest fare may be available only on the later plane (e.g., night coach). Working against this preference may be constraints on travel time, stress considerations, etc. However, the alternatives must be considered in order to bring these preferences to light and to locate backup flights should the ideal plan fail.

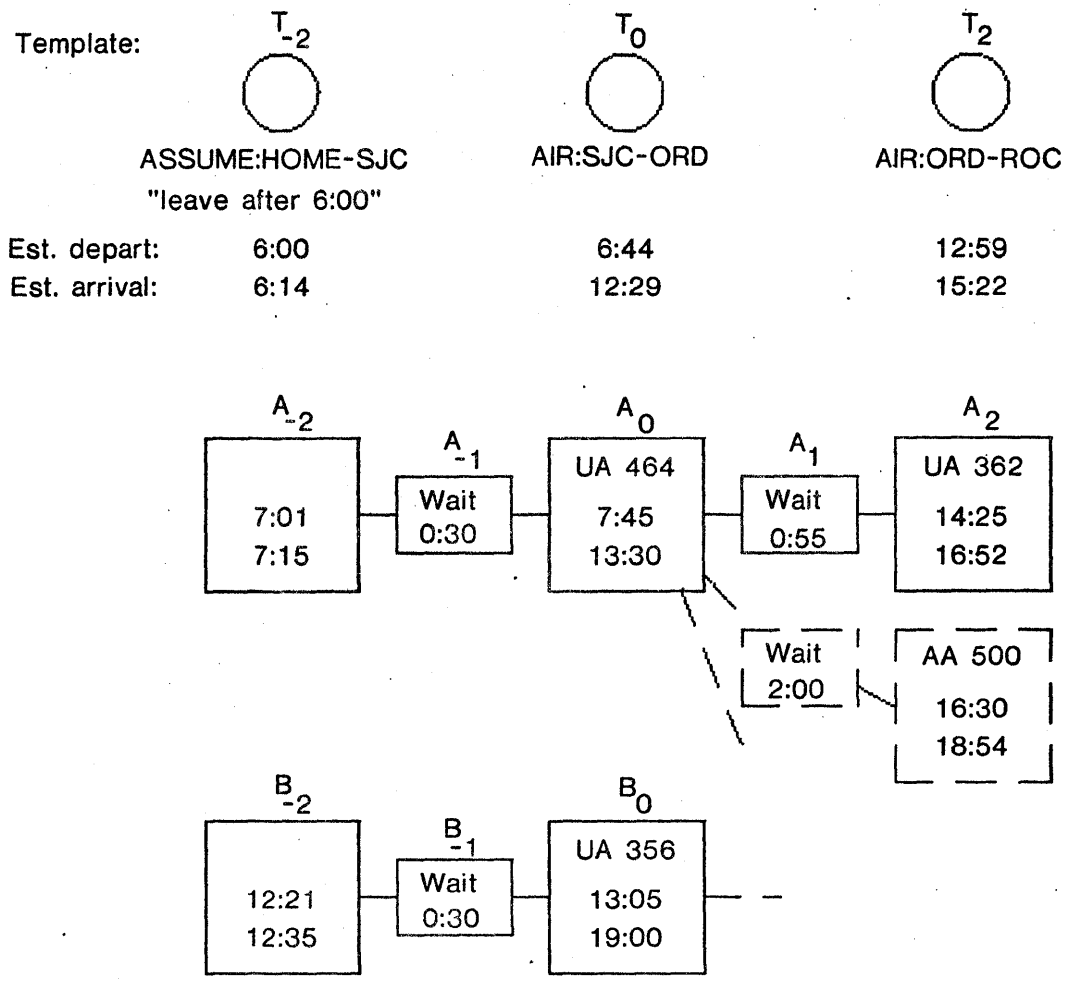

Figure 3-7: A portion of the structure used by INSTANTIATE to choose actual flights. The template is shown at the top; estimated departure and arrival times are made based on the constraint and minimum transit times. A pivot choice (A_0) is made, based on the estimated departure time of 6:44. We briefly work backwards to fill in A_{-1} and A_{-2}, and then forward to propose A_2 and therefore the interface WAIT step. A second pivot choice (B) is shown; it has not been expanded because the estimated termination time (using template estimates for the ORD-ROC leg) yields too low a utility.

(because it is capable of supporting further, later, connections such as the example shown with dashed lines in the figure). But now, further expansions will require that we leave ORD later than the departure time of A_2, i.e., after 14:25, so A_0's evaluation is now calculated by assuming template times are 86 minutes later than given. Hence, as a node is expanded with later and later connections, its utility falls.

These upper bounds are used to guide searching. For a given pivot choice, a list is retained containing all expandable nodes (e.g., A_0, A_2) sorted by upper bound. Whenever an incremental search step is performed on a particular pivot, the first element of the list is expanded, or, if it corresponds to the last template node, it is returned as a solution. Along the other dimension, each pivot's score is computed as the largest upper bound within that pivot. Thus the best pivot to work on is the one with the largest value of its best upper bound.

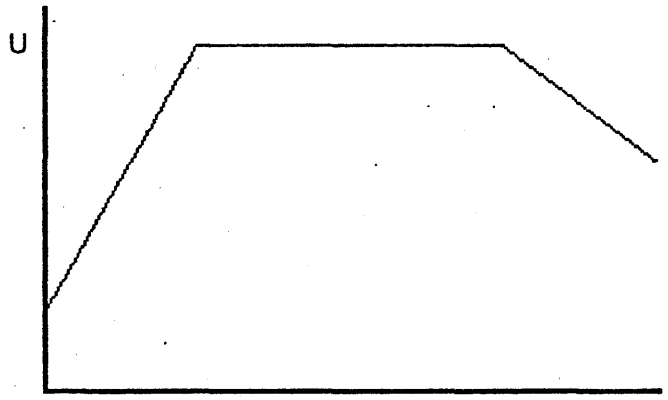

Figure 3-8: Illustration of the effect on utility of departure time of the pivot choice. Too early departures violate the departure constraint; too late departures begin to violate the matching constraint.

But when should new pivot choices be generated? The scheme used by PEGASUS to make this decision is based on the behavior of constraints. Suppose the strongest constraint is "leave...after..." We start out with a good guess at a pivot (e.g., node A_0 in the figure). Earlier pivot choices will have lower upper bounds because the constraint effect is felt by the upper bound calculation. Later pivots will eventually have decreasing upper bounds because the effect of the matching constraint is felt (see Figure 3-8). Thus, whenever INSTANTIATE finds that the "best" pivot to augment with new expansions is the earliest (latest) of the pivots so far examined, a new, earlier (later) pivot is created. The constraints thus limit, by virtue of their utility model, the choice of pivots.

One of the functions, therefore, of the utility model for constraints (section 3.3.3), is to limit this search. Unless the constraints are formulated so that the

utility contribution of a constraint always decreases as the constraint is increasingly violated, the limiting effect will not occur.[10] It is for this reason that the constraint utility model has two terms: (1) a term that corresponds to the traveler's preferences in the region of times where good solutions will lie, and (2) a term that provides proper asymptotic properties to limit the search for instantiations.

Re-evaluation of the structures created by INSTANTIATE is quite tricky, not due to fundamental problems, but simply because there is a lot of structure to be updated. In addition to changes in utility function, stress of time, and so forth, INSTANTIATE must consider planes that are cancelled, changes in departure times or delays en route, new flights created (extra sections), etc.

When the traveler advances during execution of a planned itinerary, the structure can be modified accordingly, including a possible change of pivot step, or it can simply be pruned to discard paths not taken. The present implementation uses the second approach.

The new constraints imposed by INSTANTIATE are the constraints specified by the user and the requirement that trip instances be selected from scheduled offerings of commercial transportation services. Unscheduled conveyances (taxi, rental car, etc.) are handled by INSTANTIATE as well, but complete scheduling freedom for these means that no new constraints are added by these choices.

4. DOLLARS. This level might be dubbed "the fare specialist." It is responsible for enumerating all fares for the itinerary proposed by INSTANTIATE, and returning them best-first. Because standard transportation mechanisms offer so few fare alternatives, DOLLARS makes a list of all possibilities, sorts it by U^* and returns solutions on demand.

An annoyance that DOLLARS must cope with is the chaotic nature of airline fares in the United States. Rules for computing through fares involve distinctions about on- and off-line connections, trunk to feeder connections, layover time, connecting one fare class to another, etc. DOLLARS handles correctly only some of this complexity.

[10]If the magnitude of the constraint contribution is too small, the creation of new pivots will not be adequately constrained, and will not terminate. This situation arises if the decrease in utility due to layover times for the *best* possible trip exceeds the decrease caused by the constraint violations. Such a situation may arise for itineraries with steps on which airplanes fly very infrequently. The behavior of constraint contributions at times far from the constraint time (Figure 3-8) is intended to reduce the liklihood of such looping.

5. FILLIN. The FILLIN level elaborates any ASSUME steps in plans by calling the TOP, SP, INSTANTIATE and FILLIN levels recursively on a new problem: the origin and destination are those of the ASSUME step, and a new constraint is added to insure that the itinerary generated will dovetail with time choices already made and to prevent planning the ASSUME step with the same conveyance already used for the remainder of the plan. The recursive nature permits fairly interesting plans. For example, HOME [assume] SFO [air] ORD might be elaborated to HOME [walk] CALIFORNIA-ST-STATION [train] BURLINGAME-STATION [taxi] SFO [air] ORD.

The recursive call in FILLIN is the closest analog in PEGASUS to a subgoal that must be planned, and it causes a fundamental problem, the *subgoal problem.* The subgoal problem derives from the fact that

$$U(r_1 + r_2) \neq U(r_1) + U(r_2)$$

if the utility function is non-linear. The precise impact of the non-linearity on FILLIN can be demonstrated as follows: let r_1 be the sum of the factors for the steps of the trip already planned out; let r_2 be the factors required by plans to fill in the ASSUME path. The recursive call will generate plans in best-first order, i.e., sorted by $U(r_2)$. But FILLIN is required to return answers best-first as well, i.e., sorted by $U(r_1+r_2)$. Thus if the utility function is non-linear, FILLIN cannot simply seize each solution returned by the recursive search, append the steps already planned (corresponding to r_1) and return the result. (Solutions to this problem are explored in section 4.2.1) Because PEGASUS's utility function is presently linear, the equality holds, and the technique described here is valid.

If two steps of a plan are ASSUME steps (e.g., the first and last steps), FILLIN manages two searches, and ultimately generates all possible combinations of solutions, but returns the best first. Because we desire to return the best solutions first, and because each of the two subgoal searches also return their best solutions first, we can avoid engaging in the subgoal searches unless they are needed. This technique depends on the fact that PEGASUS's utility function is K-linear. (See section 4.2.1 for further discussion of this problem.)

The recursive calls on the entire searching mechanism mean that a "problem" may be presented to a level that is identical to a problem presented earlier, on which progress may have already been made. To permit "problem" processing to

be shared, each level keeps two kinds of information: *problems*, and *task instances*. An instance simply records the parent problem and a list of solutions generated by the problem that have also been returned by the task. Thus, when a new solution is requested from a task, it first checks to see if the problem has uncovered a solution as yet unnoticed by the task. If so, it is added to the list and returned. If not, then the problem is called to work on generating the next solution. The effect is that a task will first return solutions that the problem has already generated; only then will new search effort be undertaken by the problem.

Re-evaluating FILLIN tasks is a matter of re-evaluating the planned steps and passing the re-evaluation instructions down to the recursive searches. Advancing due to execution is also a matter of passing the advancement to the relevant lower search or of terminating the search altogether.

6. PROB. The PROB level introduces all aspects of the probability model and calculates the expected utility of a plan from a decision tree. Figure 3-9 shows the tree built for our example. The new piece of structure is the ENODE, or expectation node, that represents possible outcomes and probabilities. The following description of ENODE interpretation refers to the figure:

> SUCCESS: The success entry A represents the successful completion of the HOME-SJC step, and the successful embarcation on UA 464. The WAIT steps are clearly seen to be interface steps. These steps are included in plans because they contribute to the utility, but they confuse thinking about travel itineraries as sequences of *motion* actions.
>
> MISSED: For some reason, the connection failed. This may be a consequence of the arrival distribution of the taxi and the departure distribution of the airplane. For example, traffic along the way may have delayed the taxi. This term also includes the probability that the flight will be cancelled and the probability that no space is available on the flight.
>
> CLOSED: This outcome, node C, represents the closing of the San Jose airport. Thus not only will UA 464 not take off, but no other aircraft will for a while.
>
> ASTRAY: The taxi never got to SJC. This might be due to an accident.

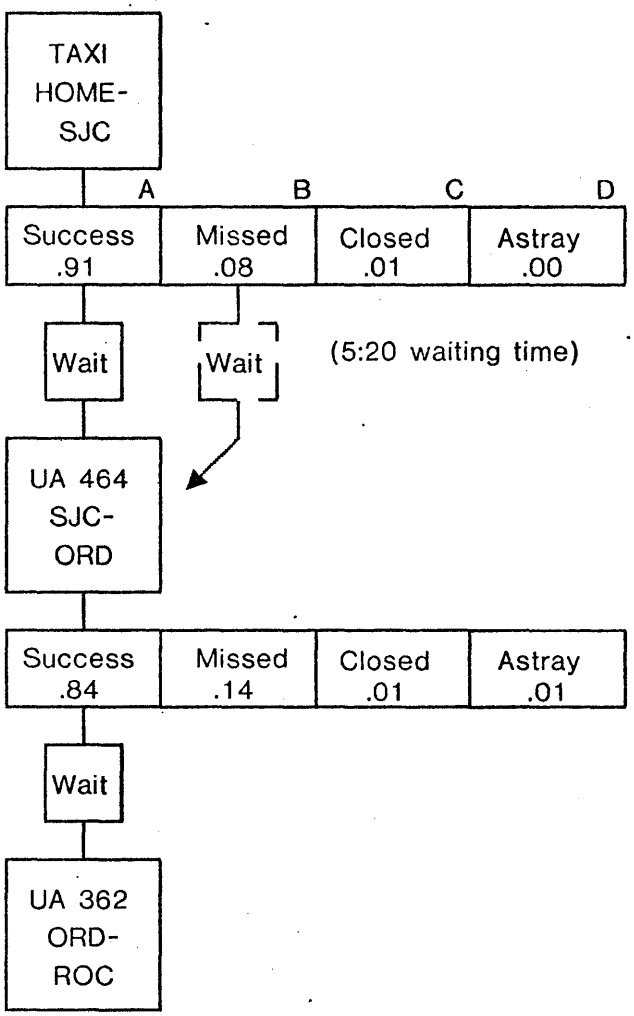

Figure 3-9: Part of the decision tree built by PROB for the sample problem. The horizontal boxes show a classification of outcomes of the connection between the steps above and below the box. The dashed box shows an example of a simple bound estimate: if the traveler misses UA 464, he can resume progress along the same route after waiting 5:20. Of course, this may not represent the best strategy in this case.

In the case of airplanes, it might be that the destination airport was closed or had landing delays so long that the airplane had to divert its landing to another airport to avoid running short of fuel.

The probabilities associated with these outcomes are calculated from the model of distributions described above. Whenever the quantities underlying the probability

model change, the probabilities are re-computed, and the decision tree is re-evaluated.

The interesting issue associated with the PROB level is: what should we do about planning for the various unsuccessful outcomes? Because of the way PEGASUS is structured, it is not necessary to generate actual plans, because such plans will emerge from the planning structure when the traveler "advances" to the actual situation: the advancing mechanism ensures that PEGASUS is always working on the problem of getting from where the traveler is at present to the ultimate destination. What *is* important to PROB is a good utility estimate for these outcomes, because the differences in backup possibilities for various plans is what will make certain plans demonstrably different from others (e.g., if our plan used the last flight of the night from ORD to ROC, the consequences of missing the plane are quite undesirable). Section 2.2 proposed two options:

- Compute upper bounds. Exact emulation of the approach of section 2.2 is of little use. The upper bound viewed from, say, node B can be characterized by the hope that there is another plane that flies to ORD, just ready to go, with space available, etc. But this will have a utility identical to the "successful" plan! Thus, using the upper bound would not adequately represent the consequences of missing the flight.

- Compute lower bounds. Unlike the monkey and bananas example, our travel model rules out lower bounds. The taxi may have gone ASTRAY because it crashed and killed the traveler, a consequence with a uselessly low utility. Even if we could *bound* the utility, this would not express the probable consequences of, say, missing the flight.

What we need to do is to compute an upper bound, but to include enough constraints so that there is information in the modified bound:

1. A crude estimate. Assume that there is as good a continuation plan, but that there will be some delay involved before you can embark on it. For example, at node B we remark that the next SJC-ORD plane leaves at 13:05, which will involve a wait of about 320 minutes. So we estimate the utility of this plan as shown by dotted lines in the figure. This can be viewed as a "low" bound, because options 2 and 3 may yield higher utilities.

2. A better estimate. The crude estimate does not consider alternative routes. If the first plane of an SJC-ORD-CMH route is missed, an SJC-DAL-CMH route may be a good alternative (DAL is Dallas, CMH Columbus). Somewhere in the collection of tasks in PEGASUS is the germ of such a plan. We can find it, and use the current U^* measurement, together with a wait estimate, to estimate the utility of the recovery plan. This will be a "high" bound because the U^* we extract from the task will be itself an upper bound.[11]

3. An expensive estimate. We can call the PEGASUS planner recursively to plan an itinerary with a new time constraint. This is computationally expensive, but yields a good estimate.

How good an estimate do we need? The answer depends on how much error can be tolerated in the calculation of the expected utility of the entire plan. If the error can be ε, then the error in a (single) failure estimate can be ε/p, where p is the failure probability. Thus if the failure probabilities are low, extremely crude estimates suffice.

Another way to view this decision is to look at the cost of computation. If we can model the computation needed to calculate each of the three modified bounds, the model can be used to determine whether the calculation is worthwhile. The upper bound on the original plan, U_0, and on the crude estimate, U_1, bound the range of expected utility for our plan. The maximum improvement that scheme 2 can make is $p(U_0-U_1)$, where p is the failure probability. This can be compared with the expense of the computation. At present, PEGASUS simply uses scheme 1.

3.4.3 Re-evaluation and Execution Monitoring

Re-evaluating plans in the light of fresh information and after planned steps have been executed is a key function of an AI planning program. One way to model the desired effect is to view the new information as an augmentation or replacement of corresponding information in the system's model of the world, followed by a complete re-planning effort. But if the changes are small, an *incremental* approach is clearly preferable.

---

[11]Some care must be exercised, because the U^* found in the structure may exceed the utility of the original strategy. This is because the problem we found had not yet been subjected to sufficient processing, thereby decreasing its U^*.

In PEGASUS, the mechanisms for handling re-evaluation and execution monitoring are closely related. To re-evaluate all plans according to new information, a structure describing changes is passed to each problem at each level. If any of the changed information affects the problem (i.e., if it was used in order to make decisions about or evaluations of the problem), the current state of the problem is modified to reflect the changes.

This process emphasizes the *changes* that have taken place, in order to avoid needless re-evaluation of all plans, including those with unchanged prerequisites. The data structures used to represent planning states must be designed to admit with facility the incremental modifications required by these changes.

During the execution of a plan, when the traveler progresses along a planned itinerary, all plans are incrementally updated to reflect the change. Thus, in the example above, if the traveler successfully gets from HOME to SJC, plans will be updated so that they are solutions (or partial solutions) to the problem: "Go from SJC to ROC, leaving SJC after *now*." This is a special case of re-evaluating plans in the light of new information, i.e., an observed change of state in the world. Although this appears to be a more drastic change, and may alter plan structures in more fundamental ways than, say, a re-evaluation resulting from a 5-minute change in a departure time, the effects of the two sorts of updates on a plan are large or small independent of this distinction.

To a purely symbolic planner, changing the problem from "Go from HOME to ROC ..." to "Go from SJC to ROC ..." is a change in the symbolic problem statement or in the symbolic world-model. The update might involve re-organizing the symbolic structure of the plan, or re-organizing preconditions to some steps, etc. Many of these aspects are present in the PEGASUS system: an update because of progress in executing a plan requires predominantly symbolic, rather than numerical, processes.

The re-evaluation and monitoring schemes are thus a brief glimpse at an area where symbolic and decision-theoretic aspects of planning enjoy analogous treatments. Keeping plans abreast of all changes in available information requires a combination of techniques, largely because the information itself is of varying character.

3.4.4 Calculating the Value of Information

In PEGASUS, the re-evaluation mechanism is used to help calculate the value of the information that may be gained from an observation of some quantity. PEGASUS generates some hypotheses about the outcome of the test, given the *a priori* information about the quantity, and re-evaluates the plans for each hypothetical outcome. The utility of plans, given that the measurement is made, is therefore the expectation of the utilities of the best plans in those situations:

$$U_{\text{with info}} = \sum_i p_i \max_j \{U(\text{trip}_j | \text{measurement is } i)\}$$

In other words, for each value of the measurement, we calculate the utility of the best plan, and then sum these utilities, weighted by the probability that the measurement will in fact have that value. For example, in the case of the weather, we might try four cases: $w \leq 1$, $w=2$, $w=3$, and $w \geq 4$.

We now must decide whether to make the test. If we choose to do so, we receive utility $U_{\text{with info}} + U_{\text{test}} + U_{\text{reevaluate}}$, where U_{test} represents the "cost" of the test, and $U_{\text{reevaluate}}$ represents the cost of incorporating the test results into PEGASUS's plans and of performing the value-of-information calculation itself. If we don't make the measurement, we just have the current best trip, with utility U_{best}. The value of information is thus

$$U_{\text{with info}} - U_{\text{best}} + (U_{\text{test}} + U_{\text{reevaluate}})$$

If this quantity is positive, the test should be performed. Note that bounds arguments can be used to rule out many tests: U_{best} is known; an upper bound on $U_{\text{with info}}$ can be estimated by finding in the levels an ancestor of the plan corresponding to U_{best} that does not include the effects of the information being tested; U_{test} is presumably known; and $U_{\text{reevaluate}}$ can be provided by the cost-of-planning model.

It may happen that no value of the measurement changes the identity of the best plan. If so, there is certainly no point to making the measurement. PEGASUS tests for this case first by choosing the first two hypotheses at extremes of the range of measured values. If the best plan is the same at both extremes, no more tests are made.[12,13]

---

[12]This is a "linear" assumption: if plan A is "best" for extremely bad weather and for extremely good weather, then it will be "best" for all sorts of weather. We could, of course, keep track symbolically of the quantities on which the utility depends, and use symbolic techniques to determine whether this assumption is valid.

The main point of this section is that the incremental update and re-evaluation mechanisms installed in PEGASUS for updating plans during execution or when the traveler changes a parameter are useful for other purposes.

3.5 Conclusion

The organization of PEGASUS's search into levels serves effectively to limit searching required. The policies chosen at each level are such that a large number of different travel itineraries are feasible: because different instances may be chosen, the actual number is unbounded. On the other hand, it seems that the constraints inherent in the problem narrow searching considerably: people cope with this problem daily without recourse to extraordinary intelligence or training.

PEGASUS demonstrates the value of utility estimates for a number of planning purposes: The estimates control planning effort. Utility and symbolic templates transmit guidance generated by cruder levels of processing to searches in more detailed levels. Bounds are used to help estimate the value of information. PEGASUS shows that many of the tradeoffs involved in planning and executing travel itineraries have natural and computable representations in decision-theoretic terms. The travel planner is a fine example of how complications arise when applying to real situations a simple scenario: go to the airport, take an airplane to an airport near the destination, ...

---

[13]Decision theory texts are replete with descriptions of the "value of perfect information," and how to calculate it. In PEGASUS, the re-evaluation mechanism is powerful enough to evaluate the effect of imperfect information, because the model can represent it (e.g., observations only *alter* distributions, and do not collapse them to discrete measurements).

Chapter 4
Critique and Extensions

The combinations of decision theory and artificial intelligence exemplified in the case studies of Chapters 2 and 3 can be broadened. This chapter touches on the issues that concern these studies, and seeks to depict the limitations of the two examples and the power of the general approach. The issues can be divided, perhaps too neatly, into two categories: the construction of the models of the domain used by a problem-solver, and the problem-solution and processing techniques used to formulate solutions.

4.1 Models

A problem-solving system that undertakes problems in a certain domain works with an *abstract model* of the domain. The abstract character of the model is necessary because not every detail of the real domain can be considered in the planning system. The design of the abstracted model of the world is a key to the performance of the system. On the one hand, a repertoire of symbolic operators alone, as used in many robotics systems, generates plans that are blatantly wasteful of resources. The augmented operators described in the monkey and bananas example, which include cost and reliability measures, still fall far short of a complete model of "reality." Yet if the model gets too complicated, with many possible elaborations, outcomes and failures, the search may grow unmanageably large.

The model can be said to *be* the knowledge of the domain encompassed by the system. The techniques for designing, representing and manipulating this knowledge are a central topic in AI (Bobrow, 1975). This thesis is an investigation of interactions that arise when some knowledge is encoded in utility and probability functions and some in symbolic processing techniques or rules.

The knowledge spans a wide range of applicability. Some is specific to a particular invocation of the planner, such as the locations of the monkey and bananas, or the announced destination of a traveler. Some is applicable to all problems in the domain, such as the basic behavior of boxes or the ways in which legs of a trip can be concatenated to form itineraries. And some is more general still, such as the A^* algorithm or techniques for controlling a planning hierarchy. But these are merely extremes of a continuum: the probability of the monkey's successfully testing a box may grow slowly with time as he learns to test well; or *a priori* values of traffic and weather conditions change slowly, but must be combined with any more recent values available.

Along another dimension, we can characterize the domain knowledge in the examples as *symbolic* or *decision-theoretic*. These categories are also extreme, but offer convenient headings for describing the models.

4.1.1 Symbolic Models

The symbolic model of a problem drives a process that generates alternative *feasible* plans in much the same way a plausible move generator finds feasible moves in a game-playing program. The model thus contains knowledge about *symbolic constraints*, which limit the points in the state space that can be reached to those that can be derived from the initial state by applying actions. The representation of the actions contains much of the constraint information. Section 2.1 shows such a model for the monkey and bananas example.

The symbolic model for PEGASUS is represented in several ways. There is substantial data describing the constraints of the transportation system: several transportation graphs, the compendium of scheduled conveyances, and so forth. The knowledge used to build itineraries is represented entirely as procedures for processing at the various levels of detail. For example, the rules for concatenating individual trips into plan sequences exercise symbolic constraint on the sorts of plans considered.

Care is required to be sure that the symbolic constraints do not *prevent* some reasonable plans from being generated. Consider two examples of such errors in PEGASUS:

- The itinerary ROC [air] LGA [bus] JFK [air] ... cannot be generated: the SP search examines only the graph of the "principal conveyance" suggested by TOP. Clearly the model could be altered to permit PEGASUS to search a larger graph for routing solutions, or even to recognize that such "changes of airport" are a common strategy when traveling on the AIR graph.
- The symbolic model builds itineraries that require the nominal departure time of connecting transportation to be later than the nominal arrival time at the point of connection. However, because departures may be delayed (a property taken into consideration at the PROB level), there is some probability that such a plan will succeed.

The second case is mitigated during execution by the arrival of information: if the earlier flight is known to be delayed, PEGASUS will generate a plan to use it.

If symbolic constraints reject feasible plans, the planning system can become

inadmissible with respect to the goal of maximizing the utility of the plan.[1] However, to relax the symbolic constraints might widen the search considerably, and require a great deal more computation to find any solution. As we shall see, a system that takes account of the cost of planning may tolerate inadmissible symbolic processing.

4.1.2 Utility Models

The utility model must reveal the tradeoffs a planning system makes among alternative plans. A requirement, therefore, of the utility model is that it give rise to proper performance of the problem-solver. The central role of the utility function places additional requirements on its formulation: information in the utility model is used to constrain searches by reasoning with utility bounds and to limit planning activities by estimating the value of planning. The cost/outcome model of system operators strives to summarize, in a few functions, the behavior of large, complicated systems. Surely the complexity cannot be captured in a few simple functions. On the other hand, an excessively precise model of the operation of the subsystems would paralyze planning, turning it into a huge simulation.

This section addresses two topics: ways to formulate the utility function to express the necessary tradeoffs, and the requirements placed on the utility function by the computational and problem-solving setting.

4.1.3 Formulating Utility Functions

This section treats utility theories only briefly in order to establish notation and terminology for what follows and to expand somewhat on the simple utility formulations used in Chapters 2 and 3. A lucid survey of utility theories can be found in (Fishburn, 1970).

An informal utility model is implicit in any notion of "best" or "good" solutions to a problem. Such a model need not be expressed in numerical terms. However, if alternative solutions can be ordered by preference, a trivial ordinal utility function can be formed (assign utility n to the "best" solution of n, utility n-1 to the next best, etc.). It is common for heuristic techniques that

---

[1]The terms *admissible* and *complete* are closely related. A *complete* procedure for proving theorems is one that will eventually find a proof, if a proof exists. An *admissible* procedure is one that finds an optimal solution, if there is a solution.

express preference in conventional AI systems to be isomorphic to ordinal utility functions. Decision theories based on ordinal utility theories are severely limited: they cannot cope with uncertainty; no help is provided in calculating bounds; and as the number of alternatives grows, so does the task of associating or "looking up" utility values given states of nature. Arrow's paradox (Arrow, 1951) shows that ordinal theories cannot handle multiple goals properly, a consideration relevant to robot problem-solving.

Cardinal utility functions are, by contrast, very useful indeed, and have received the most attention. They usually take the form of a mapping from a vector of abstract features characterizing the state of nature to a single real number. The feature vector (or "resource vector," as we called it in Chapter 3) includes what might be loosely termed "measurables," e.g., time, money, computer cycles, watt-hours. Although the measurements may include errors of various sorts, different individuals would not hotly dispute the results of the measurements. However, individuals will differ in their choice of mapping function; it is here that the relative importances or values of the measurements are combined.

Although many functions could be used to express the same preferences among states, various additional restrictions are imposed on the utility function. The most common set of restrictions arises from the need to handle uncertainty. It is convenient to have the utility of a *gamble* be expressed as the expectation of the utilities of the two outcomes, i.e., the utility of a gamble that results in outcome O_1 with probability p_1 and in outcome O_2 with probability p_2 is $p_1U(O_1)+p_2U(O_2)$. (The restrictions on the utility function are often expressed as a set of axioms for the utility theory. See Fishburn, 1970.) The ability of a utility theory to deal with gambles enables, of course, a corresponding decision theory to cope with uncertainty in the outcomes of various decisions.

4.1.4 Notation

We shall often refer to the utility models used in the Monkey and Bananas and PEGASUS examples in this section, and will introduce a notation that subsumes both models:

> Define an abstraction of a state of nature with a symbol **s**. An action in a plan is denoted by a_i, and has the effect of changing the state of nature from s_{i-1} to s_i.
>
> Define a *resource assignment function* **R** that maps $\mathbf{s} \rightarrow \mathbf{r}$, a resource vector. The resource assignment for a step of a plan is denoted $\mathbf{r}_i = \mathbf{R}(s_i)-\mathbf{R}(s_{i-1})$ We sometimes refer to the utility of an entire plan as $\mathbf{r}_p =$

$R(s_f)$-$R(s_0)$, where s_0 and s_f are the initial and final states of the plan. Individual elements of the resource vector are referred to using brackets: $r_i[1]$.

The utility function U maps resource vectors to real numbers, thereby assigning a *value* to a particular vector of measurables. We speak of "the utility of plan p," or U(p), and mean $U(r_p)$. This is simply the utility of the resources consumed in executing the plan; the advantages of reaching the goal are reflected in the resources r_p. Note that we are calculating the utility of a single outcome; to obtain the expected utility of a strategy, we must sum the utilities of all outcomes, weighted by their probabilities.

Define the class of functions L as linear mappings from vectors to the real numbers. That is, $L(r) = l_m \cdot r + l_c$, a simple vector dot product.

Define a function to be K-linear if it is of the form K(L(r)), where K is an arbitrary positive monotonic function.

Now the two examples can be expressed in these terms:

Monkey and Bananas: $r_i[1] = -C_i$, where C_i (as used in Chapter 2) is the cost of the ith step. $r_i[2] = 0$ if the ith step is CONSUME, otherwise 1. $r_i[3] = -1$. $U(r)=[-1\ -U_{fed}\ -U_{fed}]\cdot(\Sigma r_i)$. The utility function is therefore linear, of the form $U(r_p) = L(r_p)$

PEGASUS: r_i is the four-dimensional vector [Time; Money; Stress; Miscellaneous]. $U(r)=[a\ b\ c\ d]\cdot(\Sigma r_i)$ + constant; $a,b,c,d \leq 0$. The utility function is linear, of the form $U(r_p) = L(r_p)$

These models calculate utilities in two stages: first, a solution or partial solution is given a resource assignment; second, the utility function is applied to the resource assignment to yield the utility. In a hierarchical planner, either or both of these steps can vary from level to level.

4.1.5 Additivity

Because planning activities focus on building a sequence of individual steps to form complete plans, they need ways of calculating *incremental* changes to plan utilities. This need arises when adding steps to a partially complete plan, when adding detail to an existing step, or when considering alternatives for a particular step of the plan. In each case, the planner needs to compute the change in total utility as a consequence of an incremental change in the structure of the plan. It is advantageous to keep this calculation simple, as it will be performed repeatedly while planning.

The formulation of the basic state space as a vector space of measurable resources insures the incremental additivity property.[2] Thus we can characterize a

[2]The term "additivity" is used differently in some discussions of utility theories (e.g., Fishburn, 1970). It is used there to mean that U(r) can be expressed as a

plan or partial plan by the sum of the resources it requires, r_p. Adding step j and updating the utility requires: (a) setting $r_p \leftarrow r_p + r_j$ and (b) calculating the utility of the plan $u_p = U(r_p)$. If we restrict the utility function to be linear, we need only calculate $u_p \leftarrow u_p + U(r_j)$.

The chief implication of the additivity property is that we can associate r_p with every plan, even if the plan is incomplete.

4.1.6 Risk

One appeal of the utility function is the ability to express aversion to risk observed in many human, professional (e.g., medical) and corporate decisions. To express risk aversion, we define a utility function that is a convex monotonically increasing function of the original (non risk-averse) utility. Figure 4-1 shows two gambles to illustrate the point. If we choose p=.95, the two bets have identical expectations, $950. However many people would vastly prefer owning bet A rather than B because they are averse to the risk of B. We can express the risk aversion by choosing the utility function for money, U, so that p U($1000) = (1-p) U($19000) where p is altered until individuals believe that bets A and B are equally desirable. Note that p will be less than .95, so U($19000) < 19 U($1000): this effect can be achieved with a suitable convex function.

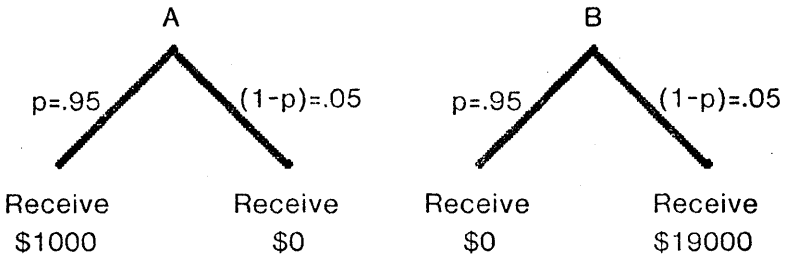

Figure 4-1: Two gambles with equal expected value in dollars ($950). Most people will prefer gamble A because they are averse to risk: the utility of $19000 is less than 19 times the utility of $1000.

The utility functions of the examples in Chapters 2 and 3 can be extended to handle risk by simply applying such a convex function, i.e., define the new utility U' to be U'(•)=K(U(•)), where K is a positive convex monotonically increasing function. Thus, a K-linear utility function can express risks.

---

sum $\Sigma u_i(r[i])$ using a separate utility function u_i for each dimension of the resource vector space.

Although the implications for problem-solving with arbitrary non-linearities in the utility function are severe, the special case of K-linear functions remains tractable. These issues are discussed in the next section.

4.1.7 Monotonicity

Calculating upper bounds on plans is greatly simplified if the utility function has a monotonic property:

$$\partial U(\mathbf{r}) \;/\; \partial r[i] \leq 0 \qquad \forall\ i,\ \mathbf{r}$$

This expression translates to "as a resource requirement of a plan increases, the utility always decreases." Thus an upper bound can be calculated by simple reasoning that determines the *minimum* values of the individual resources.

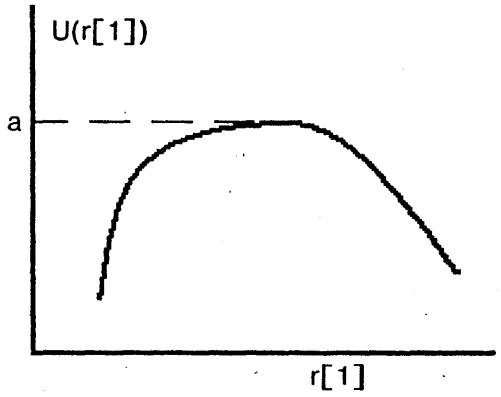

Figure 4-2: A utility function that is not monotonic in a single resource r[1].

If the monotonic property does not hold, it is mathematically possible to reformulate the utility function to restore monotonicity. Suppose a person's utility for the resource "minutes of exercise" looked like Figure 4-2; the resource vector here is simply [exerciseTime]. We can always make a new problem with an additional dimension in the resource vector that records the negative of the troublesome resource: for the example, we devise a two-dimensional resource vector [exerciseTime -exerciseTime]. Now set $U(\mathbf{r}) = u_1(r[1]) + u_2(r[2])$, where u_1 and u_2 are shown in Figure 4-3; this utility function satisfies the monotonicity property. This formulation somewhat complicates the calculation of upper bounds: the upper bound for plan p requires finding a resource vector of minima, i.e., min(exerciseTime) and min(-exerciseTime) = -max(exerciseTime). For

example, if we wish to calculate an upper bound for a plan in which the time devoted to exercise is unknown, we find $\mathbf{r} = [\ 0\ -\infty\ ]$, which gives $U(\mathbf{r}) = u_1(0) + u_2(-\infty) = a$, as it should.

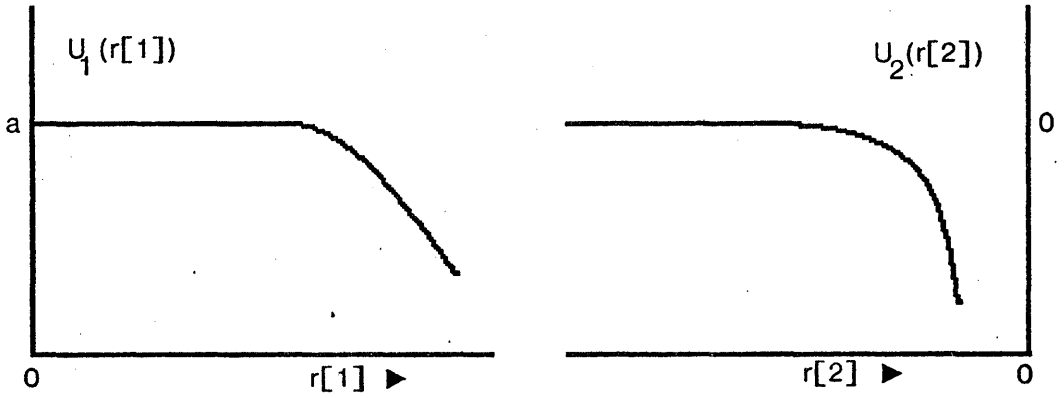

Figure 4-3: Two monotonic utility functions on two resources that sum to the utility function of Figure 4-2.

This formulation is more than a mathematical trick: two separate, nameable effects correspond to the two resources. In our example, r[1] might be called "time fatigued," and r[2] might be called "time sedentary." Now the shapes of the utility functions in Figure 4-3 correspond to intuition. Another example can be found in drug doses: there are often separate therapeutic effects for moderate doses (u_2) and toxic effects for extreme doses (u_1).

4.1.8 Resource Assignments

The link between a symbolic plan, or partial plan, and the utility function is the scheme of *resource assignments.* This scheme maps plans into resource vectors: $\mathbf{r} = R_j(\mathbf{s})$, where **s** represents the steps of the plan and **r** its assigned resources. The function **R** can take several forms; hence its subscript.

A hierarchical planning organization such as PEGASUS will use a different **R** for each level of analysis. This is the mechanism whereby the problem space is progressively explored: coarse analysis employs an **R** that expresses only some of the constraints on the solution. Finer analysis will use a more sophisticated **R**. However, the R's used at different levels must remain consistent in order that the upper bound calculations be correct.

Another reason that **R** takes on different forms is that the particular kind of

resource assignment we require may vary. For example, if we wish to calculate an upper bound on the utility of a plan p, we will want **R** to reveal the minimum resources required to execute p. These upper bound assignments are explored in the next section.

4.1.9 Upper Bound Calculations

As we demonstrated in Chapters 2 and 3, the upper bound on the utility of plans and partial plans is used to guide search. The bounds of the examples are generated by the following general reasoning:

> An upper bound on a step of a plan results from considering all possible alternative ways of executing the step, and bounding the utility of these alternatives. The power of bounding depends on the fact that most bounds can be calculated without explicitly enumerating the alternatives.

The monotonicity property of the utility function simplifies these calculations: an upper bound on utility is generated by an "upper-bound resource assignment" (UBRA, denoted $\mathbf{r}^*$), values of resources that give rise to the highest possible utility.[3]

1. Dimension-by-dimension UBRA. One method for calculating the UBRA is demonstrated in Chapter 3:

$$r_i^* = [\ \min_j(r_{ij}[1])\ \min_j(r_{ij}[2])\ \ldots\ ] \qquad (4\text{-}1a)$$

where r_i^* is the UBRA of the ith step of the plan and r_{ij} is the resource vector of the jth alternative method for achieving the step. These can be summed to get the upper bound on the utility of the entire plan:

$$U^*(r) = U(\Sigma r_i^*) \qquad (4\text{-}1b)$$

Note that the determination of the UBRA for the entire plan (4-1b) does not depend on the details of the utility function, and requires only that the utility function have the monotonicity property.

2. Utility-based UBRA. We could identify r_i^* by looking for the alternative that gives rise to the highest utility:

$$U(r_i^*) \geq U(r_{ij}) \qquad \forall j \qquad (4\text{-}2a)$$

[3]Calculating upper bounds in this fashion assumes that no elaboration of the plan can move "backward" in resource space, i.e., diminish the resources required for any step. This is analogous to the *consistency* property required of shortest-path algorithms (Hart, Nilsson, Raphael, 1968).

If the utility function is linear or K-linear, we are justified in computing the bound:

$$U^*(r) = U(\Sigma r_i^*) \qquad (4\text{-}2b)$$

This bound is tighter than that of (4-1b) because of the way in which r_i^* was identified. This technique suffers in a system in which the utility function may change, because computing an updated upper bound will require enumerating anew all the alternatives (j) for each step. By contrast, the first technique eases re-evaluation: we save Σr_i^*, the UBRA for the entire plan, and simply apply the new utility function to it.

We can see these two techniques in use in the following examples:

1. Treatment of (for example) AIR. When PEGASUS is planning in the SP level, it must compute an upper bound on a path without regard to particular flights chosen. It computes the UBRA as follows:

When the data base for the AIR transportation graph is built, all flights that traffic on each arc are examined, and a minimum resource vector r_a is associated with each arc a. Only the resources for elapsed time and money are considered:

$$r_a[i] = \min_f r_{af}[i]$$

where r_{af} is the resource vector for flight f on arc a. This is the dimension-by-dimension minimum of the resource vectors (eq. 4-1). This bound may not be achievable: for example, the flight with the cheapest fare (e.g., night coach) may require more time (e.g., use slower aircraft). PEGASUS uses this method.

We might be tempted to calculate a tighter bound by applying the utility function to the r_{af}'s and finding the maximum:

$$r_a = r_{aF} \text{ where } U(r_{aF}) \geq U(r_{af}) \quad \forall f$$

This method (similar to eq. 4-2) is awkward if the utility function can change after r_a is selected: keeping r_a current requires accessing all r_{af} in the data base each time a new traveler with a new utility function is presented to the system.

There is no intrinsic reason for choosing among the two methods. The choice will depend on the anticipated use of the system: how frequently will the utility function change? Once again, the decision is related to costs of planning: the second method reduces planning time by providing tighter bounds, but will substantially increase planning time if the utility function changes.

2. Treatment of idealized conveyances. In order to compute an upper bound on the utility of an idealized conveyance, we need a UBRA in which the minimum is calculated dimension-by-dimension. For example, we calculate a resource vector for AIR* as $r[i] = \max(d\ r_d[i], r_m[i])$ where d is the distance to be traveled; r_d is a vector of minimum rates of resource expenditure (per mile); and r_m is a vector of minimum resources possible for this conveyance. Both r_d and r_m are calculated as the AIR graph data base is built, by computing minimum resource rates along all arcs of the graph:

$$r_d[i] = \min_a\ r_a[i]/d_a$$

where r_a is the "upper bound" for arc a described in the previous example, and d_a is the physical distance represented by arc a. Similarly,

$$r_m[i] = \min_a\ r_a[i]$$

The role of this $r_m[i]$ is to prevent certain conveyances from dominating. For example, no actual AIR transportation in the PEGASUS data base costs less than $8 or takes less than 25 minutes. If these minima are not taken into account, AIR* is an overly optimistic upper bound on very short trips (e.g., 1 mile!) and wastefully permits more detailed consideration of airline travel on such trips.

3. Treatment of ASSUME. The PEGASUS shortest-path search of the transportation graph inserts steps using ASSUMEd ground transportation if necessary to get the traveler initially to a node of the graph (e.g., in searching the AIR graph, we assume a traveler could get to a nearby airport). An upper bound for the ASSUME conveyance could be computed by setting the resource vector so that $r[i] = \max(d\ a_d[i], a_m[i])$, where a_d and a_m are found by taking the minima, dimension-by-dimension, of the r_d and r_m for all idealized ground conveyances. This bound turns out to be too optimistic; it fails to constrain search adequately. By way of illustration, consider:

Suppose TOP returns a solution TRAIN* for a very short trip. When this is processed by the SP level, an ASSUME step is found to be the best way to accomplish the *entire* trip. The ASSUME resources involve no money (the cheapest ground transportation is walking) and very little time (fastest is by train); it therefore dominates TRAIN* over any route. What this means, in effect, is that we are using in SP assumptions that do not reflect the constraints imposed by TOP (i.e., exclusion of trains from ASSUME steps).

A solution to this problem is to calculate explicitly the resource vector generated by the relevant idealized conveyances (WALK, RENTED-CAR, TAXI, etc.) and to choose the one that maximizes utility (equation 4-2). This therefore represents a tighter bound. However, it places an additional burden on updating: if the utility function changes in any way, the alternative idealized conveyances must again be evaluated to find the one with greatest utility. In PEGASUS, this is a small burden because there are only 6 idealized conveyances.

All of these bounds examples involve some form of "enumerating the alternatives": sometimes the enumeration is done once when a data base is constructed; sometimes it is done during planning in order to get tighter bounds. In some cases we have simply saved a UBRA in the data base (as for all flights on a given arc); sometimes we have saved a computation formula that will generate the UBRA (as for idealized conveyances).

The powerful use of bounds in PEGASUS occurs because utility calculations applied for reasoning at coarse levels of detail generate upper bounds on the constraints that may be introduced at lower levels. Although this sort of bound can also be viewed as "enumerating the alternatives" (i.e., all possible effects of constraints), we are never tempted to use explicit enumeration because the upper bound, or minimum resource, is usually readily apparent from simple reasoning. For example, waiting times between connecting trips are not considered when finding a route in SP. However, because $\partial U/\partial \text{Time} < 0$, we can derive an upper bound by assuming the waiting times will be 0. This general technique is used to handle all constraints: the upper bound results from a resource vector that includes the *minimum* contribution from these constraints. If, in addition, these minima are all designed to be 0, the upper bound is especially easy to calculate -- simply ignore the effects of the constraint.

One of the prime considerations in choosing bounding techniques is the desire to get adequately tight upper bounds. The third example above shows that the search-limiting nature of bounds may make more expensive bound calculations worthwhile. If resources are allocated to the planning system with regard to cost of computation, looser bounds simply give license for unrealistic expectations of the results additional planning may bring.

4.1.10 Adjusting Model Parameters

Both the utility and probability models contain numerous parameters that must be adjusted to achieve proper performance of the planning system. It is relevant to distinguish between two approaches to adjusting the models: (1) an *empirical* approach, in which physical quantities are measured to some precision and delivered to the planner, and (2) a *subjective* approach, in which a designer or skilled individual is asked to estimate "proper performance" or rationales for the system.

The bulk of the models used in the two examples can be adjusted empirically. The "Airlines Guide" and similar schedules supply nominal values for transit times and fares. They could, but unfortunately do not, provide other information for the model: distributions for delays, effects of weather, etc. Any serious attempt to build a useful travel planner would need to secure these numbers or to measure the quantities directly.

However, not all parameters of the models are subject to direct empirical verification. The utility model is the most obvious example; in PEGASUS this includes a utility function, the stress of time model, and the model of constraints. Only if the objective of the planner is particularly simple (e.g., artificial robotics problems), is it possible to derive these models analytically. If not, utility information must be extracted from a human; this is a complicated process and is likely to elicit inconsistent judgements (Tversky and Kahneman, 1974).

Although the general problem is quite complex, some simple adjustments can be made based on human reactions to proposed plans. If a user prefers plan A to plan B "because it is less expensive," this suggests increasing the weight given to the money factor in the resource vector until the utility of A exceeds that of B. The trick is feasible because the user has identified the trouble. Other

causes might be traced to stress of time parameters: "because I dislike waiting in New York," or "I'm scared to land in Chicago during rush hour." This technique essentially asks a client to sort emerging plans by preference. Presenting alternative plans for comparison is not trivial. Although PEGASUS can easily generate plans of decreasing utility, they are often minor variants of each other (e.g., using slightly different transportation to an airport, but using the same selection of flights), and do not differ enough to extract determined preferences. A selection of cases designed to elicit preferences might better explore more coarsley the space of alternatives, and generate plans with substantial variations in the resource requirements.

PEGASUS probability information is stored in a data base partitioned by certain retrieval keys: in each partition lies an applicable probability distribution. The probability model embodies certain dependencies that we anticipated were necessary, e.g., arrival delays depend on weather conditions and on the airline. A simple retrieval system locates the proper probability distribution given retrieval keys (location, time of day, weather, carrier, means of conveyance). Some of these keys match "ranges" of values in the data base (e.g., a time of day can lie in the range 4-8 PM weekdays; a location can lie in the midwest; a carrier can be a trunk airline).

One advantage of partitioned models is that the partition can be refined to divide the parameter space more finely, and thereby to increase the precision of the model. As an aid to refinement, we can record empirical observations with each cell of the partition. For example, we record the actual observed departure delays in a cell of the partition -- a given airline from a given airport, within a certain range of times -- and periodically verify that the observations that fall within the cell are properly characterized by the distribution associated with the cell. This process can be automated (and would probably be dubbed "learning") or carried out manually (Yakimovsky and Feldman, 1974).

How accurate must the model be? Because an exact model may be difficult or expensive to acquire, it would be useful to anticipate how the performance of the system changes due to errors. If errors in the resource assignment and utility functions can be estimated, the simple form of the utility function and of the calculation of expectations allows us to compute the effect of errors on the final utility value; this process is called "sensitivity analysis" by decision theorists. It has the effect, for example, of requiring only crude utility estimates for outcomes of low probability.

In principle, we could acknowledge errors explicitly, propagate *distributions* of utilities up the planning tree, and apply statistical tests to the distributions of alternative plans to determine whether one is significantly better than another. This is not a practical process, in part because it places extra burdens on the representation of plan utilities, and in part because we must calculate utilities for a conceptually infinite number of outcomes represented by a particular probability distribution (e.g., we have a probability distribution of waiting times, and must calculate a corresponding distribution of utility values).

The planning process itself is capable of reducing certain kinds of errors in the utility of a plan. This is precisely what more detailed analysis will bring. The elaboration process of Chapter 2 reduces errors in this fashion by introducing tests that may reduce errors in the model; it is advantageous to employ them if the more exact model gives rise to plans of higher utility. Consider too a hierarchical planner such as PEGASUS. The final level of detail, PROB, does not supply details necessary to execution of the plan, but is included only to reduce the error in the plan's utility estimate.

There are several more subtle implications of model errors. The first might be called "indifference to small differences in utility." Suppose that calculated utilities U_c differ by up to 20 units from the exact utility function that captures a client's "true" preferences U_t. If a problem-solver finds a "best" plan with utility U_c=304, generating alternative plans with utilities in the range 284-304 is of dubious value; some of these plans will have $U_t > U_c$, and some $U_t < U_c$, but even if these actual preferences are elicited and the model is altered, the precision of the model may not be increased.[4] If errors of this sort are significant, a utility-driven planning system may want to include some hysteresis: in our example there is no point in abandoning pursuit of a plan with U=295 until its utility drops below 284, even if there are partially-planned alternatives with U=290.

Finally, the costs of planning interact with model errors. As we mentioned above, additional planning may reduce the utility error in a plan. But the cost of planning itself represents an "error" to the client. Why should we struggle to calculate an exact utility value for the plan, when the cost of planning will distort the plan's value to the client?

---

[4]If the model does not consider an effect that is important to the client, fluctuations in the property among plans will appear as noise in the utility calculation. If the noise is uncorrelated with the properties that *are* considered by the model, no adjustment of model parameters will decrease the noise.

These observations may be summarized by noting that careful sorting of plan alternatives by utility is not worthwhile in the presence of noise in the utility evaluation. Instead, we want to generate alternatives with differences that are significant compared to the noise level.

4.2 Processing the Models

In this section we examine computational methods for extracting problem solutions from the symbolic and decision-theoretic models. The discussion centers on basic problem-solving techniques, on hierarchical planning, and on the cost of planning. Although the methods themselves are not all new, new problems appear when they are combined.

4.2.1 Problem-Solving Techniques

We shall summarize the individual techniques discussed in Chapters 2 and 3, and point out several interesting problems associated with them.

Search

A fundamental problem-solving technique involves searching a graph for plans of highest utility, subject to symbolic constraints. A broad class of search techniques has been studied extensively by operations researchers; general ideas such as branch and bound (Lawler and Wood, 1966) and dynamic programming (Bellman, 1957) have been developed for this purpose.

The symbolic constraints on a search arise in varied forms. In some cases, an explicit graph must be searched; or a graph is implicit in that arcs correspond to satisfied preconditions; or the possibilities are limited by some deduction (e.g., the search for elaborations need not consider TEST steps that alter model parameters that do not enter into the calculation of the utility of the plan).

Search is not necessarily a dumb, torpor-laden process: the constraints and the power of the search technique may combine to make search a very attractive solution mechanism. Binary division algorithms used in most computer hardware are in fact search algorithms: to generate a bit of the quotient, the alternatives (0 and 1) are considered -- at each stage, half the search tree is pruned by testing the sign of a single subtraction!

The case studies use a number of different search techniques, each appropriate for a particular purpose. We can arrange the techniques into four classes:

1. Enumeration. If the search space is known to be small, enumerate all feasible solutions (i.e., those that meet symbolic constraints); apply the utility analysis to each solution; sort the solutions by the resulting utility value. Examples: TOP, DOLLARS. This technique requires no assumptions about the form of the utility function and no model for calculating upper bounds on partial plans.

2. Shortest path graph search. This very familiar search technique builds optimal plans incrementally, using the utility function and an upper bound on the unexplored part of the plan as a search guide. Examples: SP, monkey and bananas (section 2.1). A large collection of work has been done concerning shortest path searches; see Pohl (1969) for a nice summary.

Many shortest-path algorithms use "heuristic" information to guide the search, but still return optimal solutions. A^* is such an algorithm. Pohl's bi-directional extensions also use heuristic guides (Pohl 1969, also 1973).

Not all shortest-path algorithms make easy the reporting of upper bounds on the search as it progresses. A^* reports upper bounds in a straightforward way (hence its use in SP). However, Pohl's weighted version does not make upper bound calculation easy.

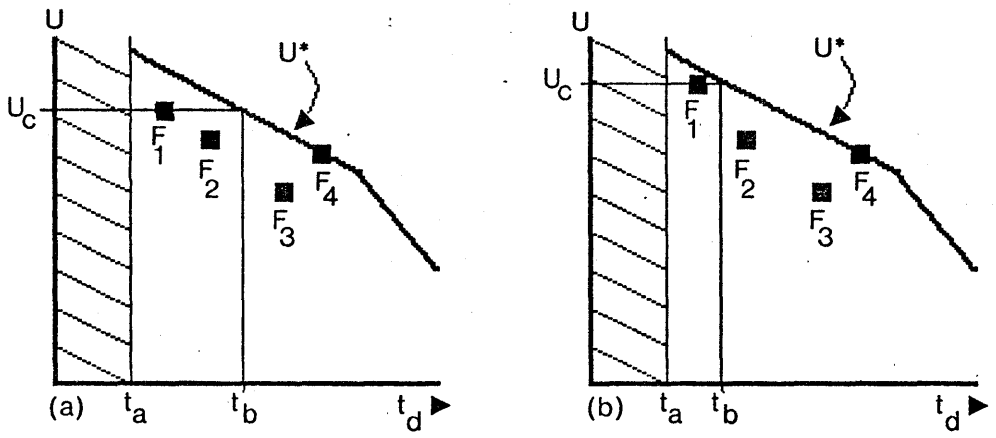

Figure 4-4: Bounded enumeration. Shown are answers (labeled F) as a function of a departure time. Increasingly later departure times (t_d) are considered until the upper bound U^* falls below the utility of the best solution (F_1) at time t_b. Consequently, a wider range of times (t_a to t_b) must be considered in (a) than in (b) because F_1's utility in (a) is considerably below the upper bound.

3. Bounded enumeration. This technique is similar to enumeration in that a collection of solutions is generated and evaluated, but it differs in one respect: the search has the potential to generate a large number of solutions, but the enumeration is arrested by a bound calculation. Example of bounded enumeration: INSTANTIATE. The INSTANTIATE search was designed to enumerate trips with departure times within certain limits; the limits are widened until the upper bound on all possible trip departures outside the limits is below the calculated utility of a chosen trip. Figure 4-4a shows the situation schematically: t_d is the range of possible departure times at a connecting point; t_d cannot be earlier than t_a, for that represents the time of arrival at the point. The solid line represents the upper bound U^* on all trips departing *after* the corresponding time, and shows an early effect due to stress of waiting time and a later effect due to violation of an arrival constraint. F_1 represents the utility of an actual connection; it is less than U^* because the cheapest fare is not available on F_1. (The fare is available on F_4, which accounts for a utility greater than that of its neighbors.) We see from the illustration that we need not consider connections with $t_d > t_b$, for their utilities are guaranteed to be less than F_1. We can continue this search by lowering the cutoff U_c, thereby increasing t_b and enumerating more connections.

Figure 4-4b shows a similar situation in which the cheapest fare is available on all flights. Because the upper bound is tighter, i.e., the difference between it and the utilities of feasible plans is less, the search is narrowed. This is just another example of search limiting by information available in the bounds.

4. Subgoals. Decomposing a goal into subgoals whose solution will achieve the original goal is a common problem-solving technique. Subgoal pursuit is often a feature of searching; AND-OR trees are one example, in which disjunctive subgoals are sprouted at odd levels in a search tree, and conjunctive subgoals at even levels. The techniques mentioned above (enumeration, shortest path, bounded enumeration) were all used in PEGASUS to generate alternatives, that is, to search implicit disjunctive subgoals: the whole program is organized to foster alternatives.

Dealing with conjunctive subgoals is somewhat more difficult, and interacts with the choice of utility function. This interaction arises because we wish

to explore the conjunction in such a way that solutions of high utility are generated first. Our discussion will use an example from PEGASUS: the FILLIN level is responsible for elaborating the ASSUME steps, and must build solutions of the form ((solution to ASSUMEd step) AND (existing itinerary)). For example, FILLIN might be given a plan ASSUME (HOME to SJC), then WAIT (at SJC), then AIR UA 356 (SJC to ORD), etc. We can characterize this plan as ASSUME AND **p**, where **p** represents the portion of the itinerary that, within FILLIN, requires no subgoal expansions.

In order to generate solutions to ASSUME AND **p** in order of decreasing utility, we might at first assume we need only generate solutions to ASSUME with decreasing utilities and append **p** to each solution. This does not work in all cases, because the utility function can be non-linear.

> Counter-example: Let $\mathbf{r}_p$ be the resource vector for part **p** of the plan; let $\mathbf{r}_i$ (or $\mathbf{r}_j$) be the resource vector for the ith solution to ASSUME. We want to find j that maximizes total utility:
>
> $$U(\mathbf{r}_p+\mathbf{r}_j) \geq U(\mathbf{r}_p+\mathbf{r}_i) \quad \forall i \qquad \text{(a)}$$
>
> Suppose we do this by finding the j that maximizes the utility of the ASSUME step, i.e.,
>
> $$U(\mathbf{r}_j) \geq U(\mathbf{r}_i) \quad \forall i \qquad \text{(b)}$$
>
> Let $U(r)=|r|^2=(r[1])^2 + (r[2])^2$, choose $\mathbf{r}_p$=[4 1], $\mathbf{r}_1$=[0 1.1], $\mathbf{r}_2$=[1 0]. By (a), j=2; by (b), j=1.

For an intuitive example of this effect, consider a data-structure selection application (Low, 1974; Rovner, 1976). Suppose the utility function is -st, the familiar space-time product used in charging for computer services. If we choose one data-structure to maximize $-s_1t_1$, and a second to maximize $-s_2t_2$, the conjunction of the two choices, $-(s_1+s_2)(t_1+t_2)$, may not be the optimal combination.

We can constrain the utility function so that maximizing the utility of the subgoal search also maximizes the utility of the total plan. For example, if the utility function is K-linear, i.e., of the form U(r)=K(L(r)) (notation of section 4.1.4; L is a linear dot product; K is an arbitrary monotonic function that allows expression of risk), it has the desired property.

Proof: We need to show (b) implies (a). (b) implies $L(r_j) \geq L(r_i)$ $\forall i$ because K is monotonic. Hence $L(r_p+r_j) \geq L(r_p+r_i)$ $\forall i$ because L is linear. Therefore $U(r_p+r_j) \geq U(r_p+r_i)$ $\forall i$ because K is monotonic. This calculation works because the *interaction* among the elements of the resource vector is linear.

This idea can be extended easily to handle more than one subgoal search, such as $ASSUME_1$ AND p AND $ASSUME_2$. Let the sequence a_1, a_2, ... represent the solutions returned from the first search ($U(a_1) \geq U(a_2) \geq$ ...), and b_1, b_2, ... from the other ($U(b_1) \geq U(b_2) \geq$ ...). Solutions to the full problem are represented by the sequence c_1, c_2, ... where $c_i = c(j,k) = [\ a_j\ p\ b_k\ ]$, where c_i is the plan formed by concatenating the plan a_j, the fixed (already planned) part p, and the plan b_k. Because we require $U(c_1) \geq U(c_2) \geq$ ..., the choices of j and k are constrained. Furthermore, we want to devote effort to the subgoal searches only when necessary. The algorithm to achieve this, used in FILLIN, is:

0. Set jMax ← jReturned ← 0. Set kMax ← kReturned ← 0. Set S to null.

1. If jReturned ≠ jMax then go to step 2. jMax←jMax+1. Generate the new solution a_{jMax}. Add c(jMax,k) to S for $1 \leq k \leq kMax$.

2. If kReturned ≠ kMax then go to step 2. kMax←kMax+1. Generate the new solution b_{kMax}. Add c(j,kMax) to S for $1 \leq j \leq jMax$.

3. Remove from S the plan with greatest utility, say c(x,y), and declare it the next solution to the conjunction. Set jReturned←x, kReturned←y, and go to step 1.

The idea behind this algorithm is that whenever c(j,k) is returned as best, we must be sure that c(j+1,k) and c(j,k+1) become available for comparison before returning the next solution.

A general solution that avoids the dependency on K-linearity might be to pass r_p to the procedures for solving the ASSUME step; they can simply add r_p to all resource vectors before calculating the utility. Unfortunately, this technique will fail if more than one subgoal is involved, e.g., $ASSUME_1$ AND p AND $ASSUME_2$. It is impossible to pass to $ASSUME_1$ a resource vector

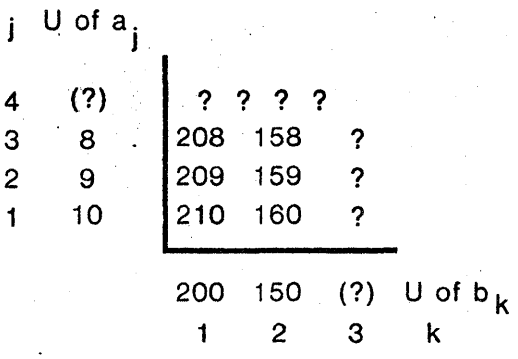

Figure 4-5: Pursuit of conjunctive subgoals. The situation depicted has jMax=3, kMax=2, and c(3,1) with a utility of 208 will be returned as the next solution. Before another solution can be returned, it will be necessary to generate a_4 so that c(4,1) can be constructed.

that does not depend on the results of search in $ASSUME_2$ and vice-versa.[5] Knowledge of the utility function may allow a bounds calculation that can be used (as in bounded enumeration) to calculate cutoff utilities for the $ASSUME_1$ and $ASSUME_2$ searches such that the optimal solution to the entire problem cannot involve solutions to $ASSUME_1$ below its cutoff or solutions to $ASSUME_2$ below its cutoff.

Heuristics

Heuristic techniques can be used to circumvent exhaustive searches, often by applying specialized knowledge of the problem domain. A common consequence of these techniques is that solutions are not guaranteed to have the highest utility of all possible solutions. We shall see that hierarchical planning and cost-of-planning techniques can be used to apply heuristic techniques sensibly.

As an example of a heuristic method, PEGASUS includes special scripts for common itineraries for connecting airline flights: west coast and east coast cities not linked by direct routes are often linked by connections in Chicago or Dallas. Although solutions to SP problems to which this heuristic applies are generated without search, they are not optimal:

[5]This is akin to the shortcomings of the linear theory of conjunctions: satisfy each subgoal independently. Sussman, 1973, explores this problem in the symbolic domain.

Solutions generated by SP in order (without heuristic):
SJC-ORD-ROC U=694.7
SFO-DTW-ROC U=693.6
SFO-ORD-ROC U=692.4
OAK-ORD-ROC U=690.7

Heuristic solution:
SFO-ORD-ROC U=692.4

SP combines these methods: first, it returns heuristic solutions if it has applicable scripts. If the planning algorithm requests further solutions, SP reverts to its shortest path search. Note that generating a heuristic solution will not decrease the upper bound; only the shortest path search offers proof that all remaining solutions lie below some bound. In the example, the upper bound remains at 774 even though a heuristic solution with utility 692 has been generated.

This use of "heuristic solutions" is similar to the current thrust in AI work to use modest amounts of reasoning to devise very good solutions to goals or subgoals, and to do the reasoning in advance of proposing the solution. This reduces search to a minimum, and places less reliance on backtrack techniques to resolve difficulties encountered deep in the search tree. The approach generates solutions without exhaustive search, but the process is insensitive to optimality. Because these methods have been explored in other AI ventures, their use was deemphasized in PEGASUS.

Solution Lookup

In some cases, it may be feasible to retrieve a solution from a data base. The cross-country heuristic of the previous section is an example. Another instance of this technique occurs in INSTANTIATE: a simple problem of the form "what is the next airplane flying from ORD to ROC leaving soon after July 14 16:00?" is answered by referring to a data base similar to the airlines guide. This retrieval does not guarantee optimality, because no knowledge of the utility function is involved.

We may even be able to retrieve optimal solutions by considering all the relevant parameters. Ideally, a strategy could be labeled with a set of conditions under which it is optimal; the optimal strategy can be later retrieved by examining the necessary conditions. Such conditions might take the form of

rules. For example, a decision rule for the monkey and bananas problem with one box is (see Figure 2-6 to identify the strategies $A\alpha$, $A\beta$, and $B\alpha$):

if $d_m < 50$ then (if $d_b < 3$ then $A\alpha$ else $A\beta$)
else (if $d_b < 8 - d_m/10$ then $A\alpha$ else $B\alpha$)
Parameters: C_w, C_p, C_b, C_c, C_{tf}, C_{tn}, U_e, U_{F_i},
P_{fw}, P_{fc}, P_{nw}, P_{nc}, P_o.

The variables in the rule are d_m, the distance from the monkey to the box, and d_b, the distance from the box to the bananas; all other parameters of the problem constitute conditions under which this rule applies (e.g., costs, utility function). The rule does not attempt to compare eating strategies to plans for pursuing other goals. Rather, it is a convenient way to retrieve a good strategy based on a small number of symbolic requirements and some parameters. After retrieving the best eating strategy, its utility can be compared with that of other plans.

Unfortunately, generating concise rules to cover a wide variety of situations is not a trivial task. We can, of course, always resort to planning and searching decision trees if a pre-computed strategy is lacking. Furthermore, the results of each search could be stored for ready reference in the future. But a deeper problem makes this hard: if small changes to any parameter result in different strategies, the number of rules could grow unreasonably large. Alternatively, we can use a partitioning technique such as the one described above for probability models.

Different forms of solution lookup can cover a wide range of capabilities. A single proper action can be extracted from a data base, using relevant constraints as the keys. Alternatively an entire sequence of actions, or script, can be retrieved.[6] The script may not offer a final solution, but rather a template for further processing. It might require elaboration, or it might simply be an upper bound for a plan alternative: Chapter 2 presents an instance of this technique.

Local Calculations

Various kinds of local calculations can sometimes be used to avoid the combinatorial explosion that a search technique would experience.

Often the optimal value of a numerical parameter can be found by closed form

[6]Schank and Abelson, 1977, describe an entire framework based on scripts for processing cognitive information.

calculations or by limited search. The PROB level in PEGASUS uses such a technique to compute the best departure time for an unscheduled first step of a trip (e.g., a TAXI or RENTED-CAR going to an airport). PEGASUS does this by reevaluating the plan with a succession of departure times, using a hill-climbing procedure to locate the departure time that maximizes utility. Although the form of the utility function is known, this optimization is not expressed in closed form because the evaluation of failure probabilities is so complicated. In any case, a variety of numerical optimization procedures (e.g., dynamic programming) are available to generate local solutions.

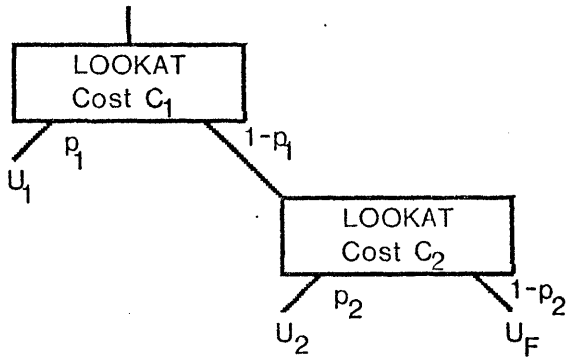

Figure 4-6: Example of closed-form calculation of the expected utility of a sequence of LOOKAT operators.

Another example arises from Chapter 2: how to determine where to look for a box (section 2.3; Figure 2-7). We are asked to find an appropriate ordering of LOOKAT operators that maximizes utility; a search of all possible orderings of hundreds of (x,y) locations is out of the question. But we note that if the *a priori* Pr{box at (x,y)} values are independent, we simply sort the LOOKAT(x,y) operators by the value of $U_{box,x,y}+C_{x,y}/Pr\{box\ at\ (x,y)\}$.

> Proof: Consider two LOOKAT operators (Figure 4-6) and a common failure outcome. We have two possible orderings: (a) Utility U_a received from the plan $LOOKAT_1$; $LOOKAT_2$; FAIL or (b) U_b received from the plan $LOOKAT_2$; $LOOKAT_1$; FAIL. Now $U_a \geq U_b$ implies
>
> $$C_1+P_1U_1+(1-P_1)[C_2+P_2U_2+(1-P_2)U_F] \geq$$
> $$C_2+P_2U_2+(1-P_2)[C_1+P_1U_1+(1-P_1)U_F]$$

which, after some manipulation, comes to $U_1+C_1/P_1 \geq U_2+C_2/P_2$. This simple case can be extended by induction to show that we simply sort the operators by U_i+C_i/P_i. We can also "sort in" alternatives besides LOOKAT's (e.g., giving up, U_{db}) by assuming C=0, P=1, and therefore just sorting based on their utility! This cost/probability ratio is sometimes used as an *ad hoc* figure of merit (Slagle, 1971 p. 99, and Garvey 1976); but as we see, it derives very easily from simple decision-theoretic considerations.

Elaboration

Elaboration techniques transform crude plans into more detailed ones, using utility measures as a guide. PEGASUS performs most elaborations as a consequence of its hierarchical planning discipline, which is considered in the next section.

The monkey and bananas example, however, suggests a different model of elaboration: the problem-solver has a number of plan-improving techniques it can apply. Hypothesized improvements are accepted if they cause the plan's utility to rise. Following are examples of elaboration:

1. Fixing failures. Paths in the plan outline that end in failure are expanded to recover. Often, this involves pursuing another top-level alternative plan. In this case, an estimate of the utility of fixing up the failure is the current utility assessment of the top-level alternative. This is, of course, not completely correct, because the state of the world used to compute the top-level utility is not the same as that after a failure. For example, in the monkey and bananas problem, failure F_3 (Figure 2-6) leaves the monkey under the bananas, and the ruins of a cardboard box under the monkey. Later, if the search for good strategies indicates effort should be devoted to this plan, the failure elaboration may be improved from an estimate to an explicit plan.

2. Inserting steps. The insertion of tests was considered in Chapter 2; the location of such insertions is governed by preconditions on the test and state information provided in the plan outline. The elaboration process considers inserting only those tests that will affect the outcome of subsequent steps, i.e., tests that modify a parameter that is used to

calculate the cost or outcome probabilities of a subsequent step. Determining how to insert tests that change the basic plan outline by changing the state of the world (e.g., tests that require the monkey to make additional moves for a better view) is very hard.

3. Changing operators. The model may provide several operators that accomplish the same operation (from the standpoint of the symbolic model) but with different costs, or different reliabilities, etc. Thus we might have two operators: WALKTO and WALKTO-AVOIDING-OBSTACLES. The choice of operator (or sub-graph of the plan outline) is controlled by utility appraisals.

4. Moving operators. If a plan outline is a sequential union of several outlines (e.g., stacking two blocks, the QA4 "buy groceries and mail a letter" problem (Rulifson, Derksen and Waldinger, 1972), or certain assembly problems (Taylor, 1976)), it may be advantageous to re-order some of the steps. A simple case, the grouping of vision operations, was discussed above. In general, however, this is a very hard problem. The decision-theory techniques provide a useful way to decide if progress is being made, but they do not obviate a considerable amount of symbolic reasoning to decide whether the plan outline remains legal.

The elaboration process has a strong parallel with trial evaluation: a plan modification is tentatively made, the utility of the new plan is computed, and the modification is saved if the utility rises. Thus a test will be inserted if its "value of information" is greater than its cost because the recalculation of the utility automatically incorporates both of these influences.

Elaboration should not be confused with a general plan-construction task. It is a local technique, and assumes that the utility space is well behaved: elaboration is a hill-climbing technique. It is a mechanism for improving a plan incrementally, and can be likened to debugging almost-correct programs (Sussman, 1973).

4.2.2 Hierarchical Planning

The organizing concept of PEGASUS, and a source of considerable power for its problem-solving, is the hierarchical planning discipline. The prime motivation for hierarchy is to prevent "blind" one-step-at-a-time planning that is likely to

explore many solution paths that are dead ends; they do not lead to solutions. This awesome possibility should be particularly evident in the travel domain: to mount a depth-first search in which the initial actions are specified in great detail, such as "Walk from HOME to the California Street train station, leaving at 7:10," is to disappear into a combinatorial jungle.

The key to efficient hierarchical processing is to make use of increasingly detailed constraints as the detail of the solution alternatives increases. In PEGASUS, we introduce the time constraints when the flexibilities of scheduling are introduced (INSTANTIATE). We introduce the stress of time model as soon as alternative transportation modes are being considered.

Although hierarchical planning using symbolic techniques alone is quite powerful, the additional information provided by the utility function adds considerable power to the method. A purely symbolic approach depends on each level to select a good solution according to criteria available to it, and then to hope that the plan will survive scrutiny at more detailed levels. The utility function represents one criterion that will be applied consistently at all levels of detail, and thus directs high-level planning toward plans that are likely to receive favorable treatment when analyzed in detail.

We shall see that the utility function has additional uses in a hierarchical organization: allocating planning resources.

Hierarchical Constraints

A solution at one level of detail constrains pursuit of solutions at the next more detailed level in several ways:

1. Symbolic constraint. A solution is, in effect, a symbolic template for a more detailed solution. TOP reports AIR*, SP searches the AIR graph. SP reports a particular route; INSTANTIATE finds pertinent transportation along the route.

2. Utility constraint. A solution can also act as a utility template. When a lower level works on a task and needs to bound the utility contributions of a part of the itinerary not yet subjected to the additional detail, the resource estimates generated at the previous level of detail can be used.

An interesting relationship between the levels is revealed in the correspondence between the answer from one level and the search in the next. For example, SP uses the "idealized conveyance" to bound unplanned parts of partial paths. This is precisely the same analysis used to *solve* the TOP level. Similarly, INSTANTIATE uses the route parameters calculated by SP as a bound on unplanned parts of the instantiated itinerary. Thus *the criteria used to evaluate a plan at one level are the search guides, the heuristic, at the next level.*

As plans are propagated into levels of increasing detail, conflicts with the original, top-level decisions may arise. In a purely symbolic setting, the detailed analysis may impose constraints that violate constraints at higher levels. This simply means that the abstract solution did not lead to a valid detailed solution. The planner can backtrack (as in GPS (see Newell and Simon, 1971) or ABSTRIPS (Sacerdoti, 1974)) or can attempt to repair the derivation tree of plans (as in NOAH (Sacerdoti, 1975)).

The constraints added by PEGASUS levels are rarely of this sort, but instead represent gradual lessening of the utility calculated at higher levels. Thus as constraints are added, a plan may cease to appear as attractive a solution as it once did, but it is not usually demonstrated to be simply impossible.

However, it occasionally happens that the choice of best solution at one level may present the next or subsequent levels with a problem that has no reasonable solution. For example, the crude reasoning in TOP may suggest taking a train from home to work; the SP level will discover that no train transportation goes near these places. This situation is detected when SP finds that the ASSUMEd ground transportation to get to the goal is preferable to taking a train headed toward the goal. This means that some other planning task elsewhere in PEGASUS dominates all plans using a train--in particular, there is somewhere a plan worked out in some detail that specifies the very ground transportation shown to dominate. The inferior plan is marked as "dominated" so that further planning will not be attempted. If the utility function or some other piece of the model is altered, the mark is removed; the changes may have caused the dominance to disappear.[7]

---

[7]The plan believed to dominate may also encounter a similar domination when it is considered in detail, and so forth. If this happens, the original domination "proof" is of course no longer valid. For this reason, the domination marks are used only when doing depth-first planning, in order to assure that the depth-first search will generate a feasible plan.

Ordering of the Hierarchical Levels

Why are the levels ordered as they are? What are the criteria for ordering? In some sense, any order would do, for all details and constraints would eventually be considered. The key to making the program efficient, however, is to perform first the planning that most firmly constrains further processing. The order of processing in PEGASUS was chosen by arguments based on this principle: "Instantiation will be more tightly constrained when a route has already been chosen," and so forth. Unfortunately, the best order may vary from problem to problem. If, for example, the hard problem is to find a way to the airport, then delaying the FILLIN processing is unattractive. In some cases, exceptional information (e.g., that ORD airport is closed due to snow) is a strong indication of the sort of analysis that should be used to constrain the search. Because the U^* measure is used to constrain searches from exploring clearly inferior areas, it is a key measure of how effectively the problem is being constrained as it is processed. Consequently, *the best processing strategy is one that causes the U^* measure to decrease as rapidly as possible.*

The ingredients of a good high-level technique are therefore the combination of low computation cost and quick discovery of flaws. Reasoning based on both of these properties led to doing probability-related processing last (PROB level in PEGASUS). More powerful methods are available for searching deterministic graphs than decision trees (Hart, 1969), and can therefore be applied first.[8] Also, it can be argued that probability will have a small effect on most plans, as travel is fairly reliable--the processing at higher levels has been induced to find reasonable plans, and not to generate proposals that have low probability of success. This design requires that only the PROB level need deal with decision trees; all other levels deal with sequential plans.

Although the PEGASUS fixed ordering works satisfactorily in most cases, it is appallingly inefficient in certain situations. Suppose we know that the weather in Chicago is poor. PEGASUS's upper bound calculations do not use this knowledge; only when expectations are calculated in PROB is the difficulty revealed. All planning tasks in progress that have not yet been analyzed by PROB will continue to attract planning attention because they have high upper

[8]One must be somewhat careful about this argument. The deterministic graph we are searching is an *upper bound* on the probabilistic graph. Consequently, although it can be searched efficiently, it yields only limited information about the true (probabilistic) graph.

bounds. All sorts of plans involving Chicago may tumble down to PROB for processing, only to have their prospects diminish. In such cases, it *may* be possible to calculate an adjustment to the resource assignments of all high-level plans than anticipate using Chicago that will not exceed the change in resource assignment that such plans would experience in the PROB level. This is an example of how a piece of exceptional information thwarts the pre-established hierarchy, but can be included in the more abstract levels *simply because much of its effect can be expressed numerically.*

A pre-established hierarchy is certainly not necessary: we could mass all planning tasks in a single heap, and try to apply to the task with highest current utility precisely the right planning step. This step would deftly identify the proper constraint argument to apply in order to further lessen the upper bound on the utility of the plan or find the proper expansion of the plan with more detail. The most effective argument would vary from problem to problem, and would of course change as plans become more fully analyzed. The difficulty with this approach, however, is to design representations for plans that indeed make the application of constraints in arbitrary orders practical.

4.3 Control of Planning

We have repeatedly claimed that the utility information can be used to control planning effort. The simplest forms of control are exactly analogous to the control over a search exerted by a numerical evaluation function:

Breadth-first (uniform) planning: Find the task in the labyrinth of levels that reports the greatest upper bound on utility. Apply planning effort to this task until its bound is no longer the highest. Repeat until the problem with the highest utility is one that has undergone all levels of processing. This method reveals the optimal solution.

The uniform planning approach demonstrates the main idea of the planning algorithm: the job of a planning routine is to find constraints on a plan that cause its utility bound to fall sharply and consequently to inhibit further planning effort on this plan. Uniform planning is a search process: we are looking for the plan of highest utility, and are improving the utility estimate as we go along. This perspective is most readily seen if we arrange all problems in a single list, sorted by plan upper bounds. These bounds

calculations have varying precisions: plans in which only crude analysis has been done have correspondingly crude utility estimates--only the first few bits of the bound value are significant. When planning effort is devoted to a problem, a more precise utility estimate is developed; and the altered plan is re-sorted into the list. This is similar to a digital search for the largest value in which only certain digits of the keys need to be examined (Knuth, 1973, section 6.3). In our case, the digits are not generated unless and until they are needed by the search.

Depth-first planning: When the first solution emerges from a level, planning at that level is suspended. The solution spawns a new problem at a new level, adding more detail. Effort is now devoted to the new level. Repeat until a plan is processed at the finest level of detail. If, during the search, a case of utility dominance (described above) is encountered, the problem is marked as dominated, and a new solution is sought from the next higher level. This method does not guarantee finding the optimal plan.

Depth-first planning is powerful because each level of analysis can propose the "best" solution it is capable of devising. If, by good fortune, the "best" solution proposed by a level remains the absolutely best (in the utility sense) solution after analysis by more detailed levels, depth-first planning would be infallible. This approach is increasingly practiced in AI: "smart" analysis is used to develop good solutions that will not cause a depth-first search of alternatives to backtrack. Such a principle may, however, devote more computing resources to deriving the "best" answer to a level than are really worthwhile.

Both of these planning techniques seem to call for solution methods within the hierarchical levels that return the "best" solution first. This is an unnecessary restriction: levels can have the freedom to use other criteria for returning solutions. A level might attempt to return first a solution that it expects will be *globally* best. For example, if ORD is known to be closed, the SP level might report a route SJC-DAL-CMH rather than SJC-ORD-CMH even though SJC-ORD-CMH has a higher utility measured by SP criteria. Or a level might try to report a first solution that requires little computing--the SP heuristics for cross-country flights are an example of this strategy. In either case because the solution is not the "best," there remain solutions that are locally better (i.e., according to the evaluation at this level) and the emergence of this solution

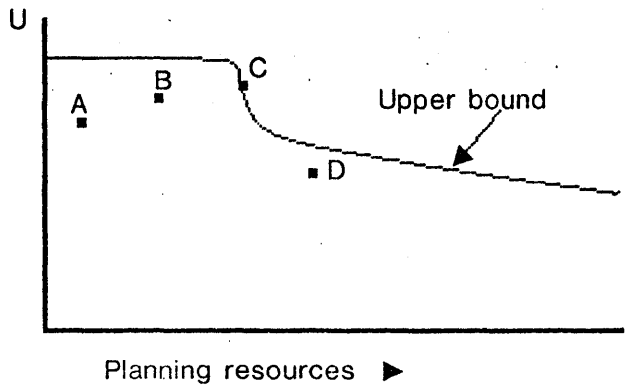

Figure 4-7: Profile of a typical planning process. Early solutions (A and B) are heuristic solutions generated with little processing, and do not cause the upper bound to fall. Later answers are emitted in the presence of a falling upper bound.

does not cause the upper bound for the entire task to fall. This property is shown schematically in Figure 4-7. This sort of behavior is advantageous for depth-first planning--the whole idea is to promote a plan that has global promise.

The uniform planning approach is still valid in this situation, although it cannot take advantage of these early solutions. The reason is that the problem with largest upper bound gets resources; the task shown in Figure 4-7 will therefore be given resources until solution C emerges and is passed on to a lower level. Solution A (or B) will be ignored until upper bounds at all levels have fallen below the utility recorded for A (or B).

The relaxation of the "best-first" requirement shows how heuristic techniques can be applied in our framework. Solutions A and B might have been generated by heuristics--they require relatively little computation to derive, but do not reduce the upper bound. These inexpensive solutions are available for processing at lower levels, but the undiminished upper bound of the original problem will invite more processing to be applied to the task should the heuristic solutions prove poor.

Clearly, neither the uniform approach nor the depth-first approach is adequate by itself: the depth-first approach cannot guarantee a good plan; the uniform approach uses too much planning time. What we need is a way to pursue a planning strategy intermediate between these extremes: analyzing the "cost of planning" reveals a solution.

4.3.1 The Cost of Planning

Planning is not free. If resources consumed in problem-solving were of no concern, we could build a perfect chess-playing program: it could examine *all* possible future games derivable from a given board position (~ 10^{160} nodes) and choose a best move. Both PEGASUS and the Monkey and Bananas example could be formulated as dynamic programming problems; the exploration of the huge state space would reveal the optimal solution.

But these large problems are not solved with exhaustive techniques: we implicitly acknowledge that planning is not free, and formulate solution techniques that attenuate the processing required.

A consequence of including costs of planning in the derivation of the "optimal" solution is that the performance of the problem-solver is included in the assessment of optimality. In order to differentiate between optimal plans and optimal planning, we shall introduce two terms:

- S-optimal. The Solution itself is optimal, as defined by the various utility and probability models.
- P-optimal. The Planning is optimal, in the sense that utility of the plan *and* the planning together is optimal.

The economics of planning are just as important as the economics of the solution itself. Thus, a P-optimal solution may be a barely adequate solution to the problem, provided relatively few planning resources are consumed. Or the solution may be itself S-optimal, even though more planning resources were used.

There are, of course, applications in which either the cost of planning or the cost of the plan dominates the total. If the cost of planning is large, as exemplified by chess-playing programs, solutions of widely varying utility may all have nearly identical total costs. In the extreme, planning techniques need not search for the best solution, but must be efficient. If the cost of executing the plan is large, as exemplified by optimization problems on a huge scale (how to refine $100,000,000 of oil), we need techniques that find the S-optimum.

Notions of the cost of planning have application in problem-solving at many levels. The discussion above centers on an external description of the entire

problem-solving process: what planning resources are invested, and what sorts of solutions result? But this same question arises repeatedly in any search: if the search proceeds along a particular path, how much processing will be required to find a solution, and how good will that solution be? The second question is answered by the utility model of the problem, as explained in earlier sections. The first question is the province of a cost-of-planning model.

We can distinguish two sorts of cost-of-planning models: a *recording* model and a *predictive* model. The recording model simply measures expenditure of resources as planning actually progresses, and charges the appropriate plan with the expenditure. A predictive model must compute, on the basis of a partially complete plan and possibly additional information (e.g., a data base that characterizes problem-solving performance in the past), the cost of planning required to complete the plan.

PEGASUS incorporates the cost of planning in its control of the hierarchical planning tasks. The next three sections describe two control models and the PEGASUS implementation that uses both models.

The Search Model

The direction of a problem-solver is controlled jointly by the utility of the solution approach and the utility of the planning. Let us couch this control in the form of an optimal search algorithm. We shall need some notation: consider a plan p that is not yet completely finished. First, we consider resources already assigned to p:

r^*_g — Upper bound resource assignment to p given current detail of processing. The subscript g is intended to evoke the function g(n) in the discussion of A^* (section 2.1.4).

r_{pg} — Planning resources already consumed to generate p. A nasty problem arises in computing this number. In any search, the development of solutions shares a good deal of the processing. Properly assessing each solution for the resources it consumed may require re-assessment when more solutions arise.

Second, we consider upper bound resource assignments to the effects of further processing on p:

r^*_h Upper bound resource assignment on the effects that will arise as p is refined to become a complete plan. That is, if r is the resource assignment of the complete plan, derived from p, then $U(r) \leq U(r^*_g + r^*_h)$.

r^*_{ph} Upper bound resource assignment for processing required to finish the planning of p. Estimating this quantity requires a predictive cost-of-planning model.

Using these estimates, we control planning with an A^* algorithm using the evaluation function $U(r^*_g + r^*_{pg} + r^*_h + r^*_{ph})$. By contrast, what we have called "breadth-first" planning is driven by $U(r^*_g)$. In both cases, planning terminates when the plan selected to work on is already complete to the finest level of detail.

If A^* is to be admissible, i.e., if it is to find the optimal solution, it is essential that the second two resource assignments be upper bounds. If we disregard this requirement, we can include more heuristic information that increases the selectivity of the search, at the expense of missing the P-optimal solution.

The Cutoff Model

Depth-first planning suggests a rather different use of the cost of planning: the cutoff model (first mentioned in section 2.4). Consider the picture of a complete problem-solver shown in Figure 4-8. A plan p_i is presented to the "black box" for processing. The box consumes resources: at time $t>0$, it has consumed $r_p(t)$. The box keeps an upper bound resource assignment for the task given it in $r_u(t)$. At integer times, a plan "solution" emerges: $p_o(1)$ emerges at $t=1$, $p_o(2)$ at $t=2$, etc. Let R be a resource assignment function that characterizes as much detail of plans as the problem-solver is capable. Then we can calculate the utilities of the emerging solutions: $U(R(p_o(j)))$. These plans do not necessarily emerge in order of decreasing utility.

Let us suppose that we have, by some mechanism, determined to allocate planning resources to this problem-solver. When should we stop allocating resources? Clearly, if

$$U(r_u(t)) \leq \max_{j=1,2,\ldots t} U(R(p_o(j)))$$

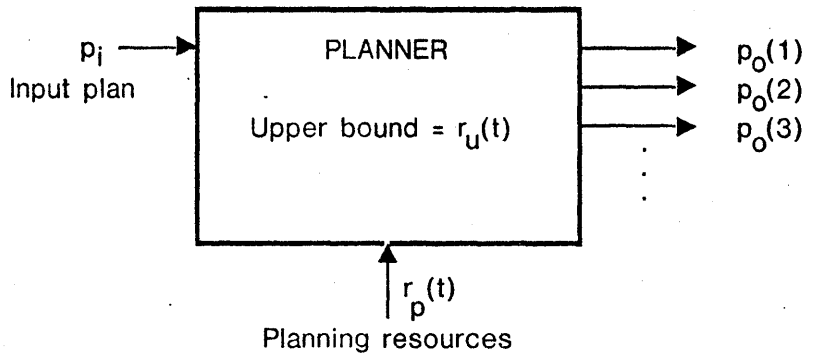

Figure 4-8: A model of a single level of a hierarchical planning process. It is presented with an input plan (p_i), and some planning resources that total $r_p(t)$ at time t. Answers $p_o(i)$ are generated.

the problem-solver has already generated the best solution according to its criteria, and can stop. We can also write down the condition for cutting off planning effort at time t:

$$U(r_u(t) + r_p(t) - r_p(j)) \leq U(R(p_o(j)))$$

The right-hand side is the utility of executing the plan that was generated at time j. The left-hand side represents the current upper-bound, diminished by the planning resources consumed since time j. This inequality, which subsumes the earlier one, curtails planning when more resources have been expended than the difference between the upper bound and the utility of the plan generated. There are several such equations, one for each solution already generated. If the utility function is K-linear, we may write the condition for cutoff:

$$U(r_u(t) + r_p(t)) \leq \max_{i=1,2,\ldots\lfloor t \rfloor} U(R(p_o(i)) + r_p(i))$$

Implementing this cutoff requires only a recording cost-of-planning model. However, if an anticipating cost-of-planning model may be able to determine at time t that $r_p(\lfloor t+1 \rfloor)$ is so large (i.e., that so many resources will be required to generate the next solution) that the inequality can be demonstrated to hold before actually expending the resources.

Note that this method continues to work during execution of a plan, provided we assume that all planning tasks are "advanced" and updated to reflect the new situation. If execution has gone awry, the updates will lower substantially the utilities of existing plans, thus allowing execution of some planning tasks previously cutoff. This is a consistent way that re-planning is accomplished.

Now let us ask whether *any* resources should be devoted to this problem-solver at all, even before $p_o(1)$ is generated. In order to answer this question, we assume the existence of a currently "best" plan p_b, already available--if no planning has been done at all, this is simply the "do nothing" plan, with an appropriate low utility. What information do we have?

$R(p_i)$ We know the resource assignment for p_i, the crude plan presented to our problem-solver. $U(R(p_i))$ is an upper bound on $\max(U(R(p_o(j))))$ if we were to actually generate the $p_o(j)$.

r^*_{ph} Lower bound on the amount of planning resources known to be needed by this problem-solver. The anticipating cost-of-planning model is used to compute this, and it can use any relevant information about p_i. Of course, the computation itself should consume *much* less resources than r^*_{ph}.

r^*_h Lower bound on the growth of plan resources as a result of constraints imposed by this problem-solver.

The problem-solver should not be used at all if

$$U(R(p_b)) \geq U(R(p_i) + r^*_{ph} + r^*_h)$$

Here again, we could disregard the upper bounds requirement, and obtain greater selectivity (i.e., fewer competing problem-solvers will allow resources to be devoted to them) at the expense of admissibility.

These two cutoff criteria can be applied recursively to a problem-solving framework such as used in PEGASUS (see Figure 4-9). Note that the recursion is from the right--i.e., the *outputs* of all nested models are the same, not the inputs. Consequently, the p_o are always completed plans, and are evaluated with R, the most detailed resource assignment available in the problem-solver. This is an essential feature: if, in level 1, we use a corresponding resource assignment R_1 in the cutoff equations (Figure 4-10), the equations will usually stop the planner too early. If this level generates answers best-first (by its own evaluation, R_1), only one answer will be generated, because thereafter the upper bound $r_u(t)$ will represent plans strictly poorer than the first answer: $U(r_u(t)) \leq U(R(p_o(1)))$ for all $t>1$. This error is sometimes called "the one-step planning pitfall:" local optima, viewed under a subset of all problem constraints, do not always turn out to be global solutions as well.

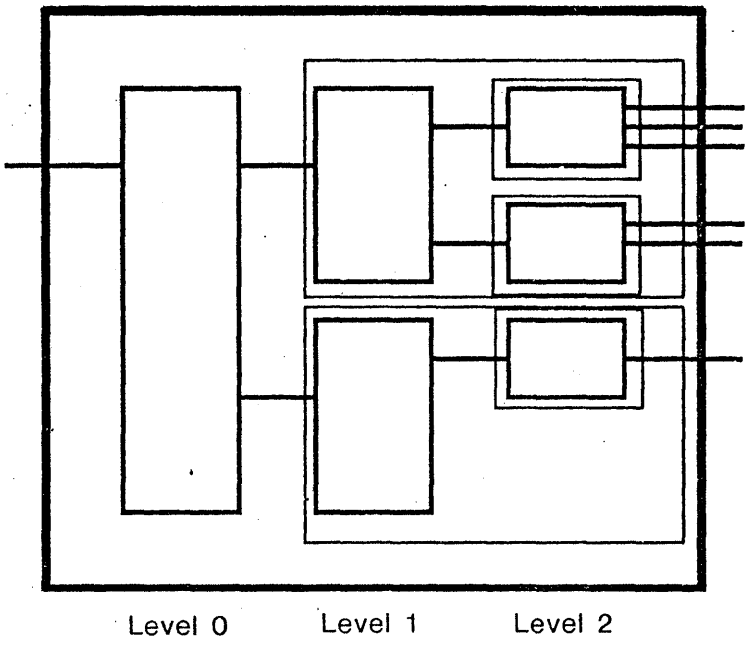

Figure 4-9: Recursive application of the cutoff criterion to a hierarchical planning system. The thinnest lines show boundaries around systems of the form of Figure 4-8. Note that the recursion is from the right, i.e., the p_O answers must have had all possible detailed processing applied to them.

An implementation

PEGASUS makes use of both models. The search model is helpful in deciding which of the many competing tasks should get resources. A simplified form of the cutoff model, applied only to PEGASUS as a whole, is used to curtail planning.

In PEGASUS, the control of the various tasks assumes $r^*_h=0$. This is largely because we express the upper bound on more detailed planning as carefully as possible in the upper bound resource assignment for r^*_g. For r^*_{ph}, we measure the best planning performance of each level, and compute r^*_{ph} as the sum of the minimum requirements for all levels to which p has not yet been subjected. However, r_{pg} is assumed to be zero, because of the difficulty in computing it properly. This is a serious fault that must be addressed in future systems.

By way of example, consider the following table summarizing the performance of

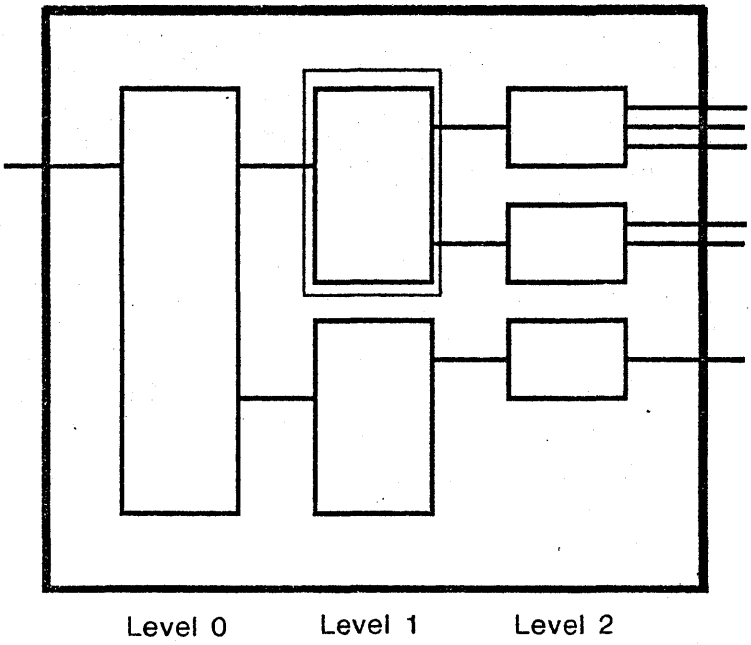

Figure 4-10: Incorrect application of the cutoff criterion. The further drop in utility that level 2 can cause may have the effect of allowing more planning within the level 1 process identified.

PEGASUS on a typical run, using three different planning strategies: uniform, depth-first, and CofP (planning controlled by the cost-of-planning techniques):

Technique	Number of solutions	Utility of best solution	Planning time (sec)	Planning cost (utility units/sec)
Uniform	16	649	672	0
CofP	4	649	242	-.3
CofP	1	633	101	-1
CofP	2	633	59	-3
Depth First	1	633	42	

For this problem, uniform planning found the S-optimum solution, but so did P-optimal planning using all but the highest costs of computer time.

4.4 Summary

Although this chapter has attempted to review many of the aspects of our problem-solving formulation, two points stand out:

- Heuristic methods have a place in problem-solvers that find optimal solutions. Maintaining upper bounds on all solution paths by uniform

application of resource assignments and the utility function permits the value of heuristic solutions to be compared with the potential of more exhaustive techniques. As a consequence, the heuristic solution will be used when it appears good, but may be supplanted with a more exhaustive technique when an upper bound indicates the solution is poor.

- The cost of planning must be viewed as a resource requirement, just as are the resources required to execute a plan. A cost of planning model can be used to help control planning by discounting the utility of a partial plan by the amount of resources required to complete the planning. Consequently, an exhaustive search may, simply because it is computationally expensive, be dominated by a heuristic solution.

Chapter 5
Decision Theory and Current AI Work

This chapter relates the work presented in this dissertation to other work in AI. There are three primary relations: What other work or problems motivated this study? What techniques have been borrowed from previous work? And what contributions can this work make to present or future endeavors?

5.1 The Origin of This Study

The motivation for an exploration of ways to find *good* solutions to symbolic problems and to control allocation of planning resources emerged from shortcomings of a large AI system, an "Instant Insanity" puzzle solver (Feldman, *et al*, 1971). This system, constructed at the Stanford Artificial Intelligence Project in 1970, was designed to build, using a computer-controlled manipulator, a tower of four blocks that was a correct solution to the puzzle (each "side" of the tower must present an example of each of four colors). The blocks were initially scattered on a work table; the computer was expected to derive position and color information from images of the table measured by a vidicon equipped with color filters.

The final system was extremely large, consisting of 8 time-sharing jobs that communicated through shared memory and interprocess communication--altogether more than 300K words of program. The system consisted of a monitor job and seven "specialist" jobs:

- Control and Planning
- TV Camera Model
- Edge Follower (image gathering and analysis)
- Simple Body Recognizer
- Color Finder
- Manipulator trajectory calculator
- Manipulator servo

Each of these programs save the first was the pride of a different graduate student, eager to see his program exploited fully. But the size and complexity of the system immediately generated questions that required comprehensive strategic control of the whole process. For example:

How much effort should be devoted to analyzing a scene for edges? When can the analysis stop? The answers depend on the quality of information needed by the body finder, and how it fares in its task.

How is information about the arm trajectory to be used properly? Some tasks given to the trajectory planner require an intermediate step to deposit and re-grasp the block, a rather expensive and somewhat unreliable operation. If a particular trajectory request requires such an intermediate step, is there an alternative that is less costly? Perhaps the blocks can be stacked in another order to avoid the extra step. Or perhaps there is an "extra" block on the table whose use would avoid the intermediate step. But maybe it is obscured, and will be hard to recognize with the body-finder.

Clearly, what is needed is a method for assessing the relative advantages of different sequences of actions to build the tower. These sequences need to contain primitives that correspond to the controllable aspects of the specialists, such as which solution methods to try. The controller needs to monitor the execution of these sequences because actual execution costs and outcomes differ from estimates used during planning.

These problems were barely new at the time the Instant Insanity demonstration was being fashioned. Indeed, the problem of overall control was rapidly being identified as an area of weakness in robot problem-solvers at the time (Munson, 1971; Fikes, Hart and Nilsson, 1972; Winston, 1972). These issues stimulated this work and also influenced subsequent Stanford efforts (Bolles, 1976; Taylor, 1976).

5.2 Planning and Problem-Solving

PEGASUS is related in diverse ways to previous efforts in problem-solving, and offers a new perspective in some areas. This section examines briefly these relationships by exploring a few selected facets of problem-solving. More complete surveys of modern problem-solving appear in Bobrow (1975) and Fikes (1976).

Mechanisms for exploring alternatives. A key issue throughout the history of problem-solving has been the manner in which alternative solution possibilities are generated and pursued. For example, the renowned "hypothesize and test"

paradigm is one structure for sifting through alternatives. We have seen in both case studies involving decision-theoretic notions a requirement that we explore enough alternative solutions (or partial solutions) to verify that we are pursuing a good one. Because we desire a *good* solution, it is reasonable to generate more alternatives than if we seek only a feasible solution.

A classical method for exploring alternatives is heuristic search (Newell and Simon, 1971, offer an excellent formulation of these techniques). The representation for an alternative is a *node* in a search tree. A node can be *expanded* into one or more successor nodes that can in turn be considered. For example, a node of a game tree is usually a state of the game, such as a board position, and the successors are the collection of legal moves from that position. A resolution theorem-prover might consider a node to be a clause; successors are generated by resolving two existing clauses. These programs struggle to find the best node to expand next to lead to a quick solution, but are usually confronted with large lists of unexpanded nodes among which to choose. The expansion process is very local: virtually no information about the goal is used in the expansion itself.

Frustrations with node-searching techniques led to a competing paradigm for problem-solving, using "procedural embedding" of knowledge (MicroPlanner: Sussman, Winograd and Charniak, 1970; Conniver: McDermott and Sussman, 1972). The basic idea is to use knowledge about kinds of goals and classes of solutions, represented as fragments of computer programs, to constrain the exploration of state-space. A program, in principle embodying arbitrary computation, can be used to choose among the various successor operators applicable in a given state. Once a choice is made, the consequences are typically explored in a depth-first fashion, counting on the excellence of the choice. If, as the solution is explored, a symbolic constraint cannot be satisfied, control is caused to backtrack to one of the choices, and a new alternative is tried. It was soon realized that as much care and computation wanted to be devoted to backtrack choices as to (forward) solution choices (Sussman and McDermott, 1972).

This technique makes it hard to apply uniform problem-solving processes, such as utility evaluation. The decision processes are embedded in computer routines, largely inaccessible to external inquiry. More importantly, the decision *consequences* are disguised in bindings in a control stack that is an inconvenient representation for the current plan.

The poor accessibility of plan knowledge in part motivated a softening of the procedural embedding, shown in the NOAH system (Sacerdoti, 1975) and in KRL (Bobrow and Winograd, 1977). These systems retain the power of careful choice by using programs as the representation for considerable domain knowledge, but rely on separate, accessible representations for plan knowledge: the *procedural net* in NOAH, the *description* in KRL. The division also enables other processing, in addition to the application of certain uniform problem-solving procedures. In NOAH, for example, it permits plan steps to be re-ordered to construct linear plans from "non-linear" symbolic constraints; by contrast, reordering a call-stack representation of a plan is unthinkable.

Despite withdrawal from control structures as plan representations, current problem-solving systems are not intended to consider a large number of alternatives. Rather, they assume that the specificity of heuristics of choice will be matched to the size of the problem space in such a way that the problem-solving system can be expected to explore twos, not hundreds, of partial solutions. That choice algorithms can be made to improve as fast as the problem space expands, both by formalizing problems with greater detail and by toughening solution criteria to include objectives such as optimality, is extremely questionable.

The most recent effort in AI languages (KRL) emphasizes flexibility, so that systems could perhaps be tailored to explore a variable number of alternatives. In KRL, the designer is provided with a framework in which problems of representation, control and processing can be resolved in a variety of ways. The handling of alternatives must be just as flexible as the representation of plan or domain knowledge: the extent of exploration of alternatives is decided by the designer of the particular problem solution.

Processes in planning. It would seem that an obvious way to provide flexibility for the handling of alternatives is to use the techniques of multiprogramming: each alternative pursuit is represented as a process that owns state information relevant to its task and is allocated computing resources in some way. We can examine an AI problem-solving system for opportunities to split computation into parallel or quasi-parallel tasks, perhaps to take advantage of several processors. Such an approach has seen some use in AI (Fennel, 1975; Taylor 1976). However, experimentation with processes has been hampered by the poor support provided by most programming languages. Only recently has there been progress

in this area (Feldman, *et al*, 1972; McDermott and Sussman, 1972; Bobrow and Wegbreit, 1973).

The real payoff from multiprogramming would seem to appear when the process structure is derived from the need to explore various alternative solution possibilities. This is the approach used in PEGASUS, GUS (Bobrow, *et al*, 1977), Hearsay II (Lesser, *et al*, 1975), and SNIFFER (Fikes and Hendrix, 1977). But a conventional process approach may bundle up too much of its state in a control stack, and make it all but impossible to pass into that context various global changes that must affect the processing.

A tool to attack this problem can be found in the class/instance model derived from SIMULA and Smalltalk (Goldberg and Kay, 1976), and evidenced in a crude form in PEGASUS. The idea is to fix on the "instance data," the data structure that must be owned by a particular process. Other data is not of such vital concern: the *class* itself is usually defined as a collection of procedures; individual *activations* of an instance indeed require a control and binding context for local variables, but it is fleeting and can be destroyed after each operation on the instance is complete. Thus the crucial item is the *instance data structure*, which can be viewed during instance processing as a collection of bindings. PEGASUS uses this idea in a primitive form, without proper supporting language primitives. This style will be more ably supported in Plits (Feldman, 1976).

To proper access to processes must be added proper control and allocation of resources. Generally, the few uses of process structures in AI have scheduled processes with the objective of achieving desirable performance from the problem-solver. *Ad hoc* priority schemes that have little or no relation to the problem domain are used to drive the scheduler. At most, the costs of planning are loosely incorporated into the developer's notion of desirable performance. Generally, the uses of numerical models in problem-solving have atrophied as the flexibility of symbolic techniques has increased.

Numerical guides to problem-solving. Numerical guides to problem-solving are as old as AI itself; indeed AI was born of a split with the numerically-oriented worlds of optimization and "signal processors." The uses of numerical measures in AI can usefully be divided into two classes:

- Objectivity. The objective of the problem-solver is stated in part numerically, and depends on a numerical model of the domain. Examples are: to solve a puzzle in the minimum number of moves or to plan an optimum travel itinerary, based on a model of the traveler's utility.

- Selectivity. A numerical method of some form is used to steer problem-solving in a direction where solutions are thought to lie. Numerical controls are attractive for several reasons:

    - The numerical function provides a convenient way to aggregate different sorts of heuristic information, to "weight" information, etc.

    - The tractability of simple mathematical functions (e.g., polynomials) often suggests techniques. For example, differentiation might be used in a hill-climbing procedure.[1]

    - Finally, of course, search techniques are available that use numerical information fully. A numerical score allows the "best" node of a search tree to be expanded, or the "best" problem-solving process to be executed.

Numerical objective functions are not in wide use in problem-solving, although they are attractive in signal-rich applications (e.g., speech and vision). Indeed, one of the aims of this work is to argue that some problem-solving objectives are best stated numerically, and that use of numerical objectives should be encouraged. As we have seen, even if "heuristic" problem-solving techniques are used, a numerical objective function helps determine whether an acceptable solution has been found, or whether additional processing is warranted.

By contrast, numerical functions to provide selectivity have been, and continue to be, widely used in AI (various books provide ample surveys: Nilsson, 1971; Newell and Simon, 1971; Slagle, 1971). A selectivity function is used to imprint a topography onto the solution space that is believed (by the designer of the function) to elevate solution locations so that they may be found by an

---

[1]Unfortunately, there is a converse to this advantage. The reverse problem is enticing: what results will a particular neat mathematical function have on a problem-solver? Samuel, for instance, tried briefly an evaluation function for playing checkers that involved the first and higher moments of the white and black pieces about various axes on the board. The results were predictably disappointing: the abstractions of mechanics and of checkers are not likely to have much overlap (Samuel, 1967).

altitude-sensitive search. But such a design may emphasize selectivity rather than precision, and may render a numerically-controlled search incomplete: an erroneously low upper bound on the value of a partial path can irremediably prevent further exploration of that path. Consequently, the designer worries about the magnitude of the error (Pohl, 1973). A common hedge against error in a game-playing evaluation function is to search ahead several moves, and then "back up" the scores: the (static) score of a particular next move is thereby buttressed by a small amount of searching (Slagle, 1971, presents many variations on this theme). Distressingly little work has been done to determine experimentally the precision of a heuristic or evaluation function, although this is feasible only when a large number of points in the space can be examined and evaluated. There are exceptions: Paxton (1977) has explored the performance of various speech-processing techniques over a sample of 11 input utterances. Samuel (1967) used a learning scheme that is at heart an error detector that provides a feedback signal to adjust the evaluation polynomial.

Hierarchies.[2] Hierarchical representations are one method of organizing the application of constraints to the solution space, shrinking it until no more constraints remain, and (one can hope!) leaving a non-empty portion of the space: the solution. By contrast, a single search makes tentative explorations of the solution space in different directions, evaluates them in the light of all constraints, and pursues paths according to the results of the evaluation. Early problem-solvers (e.g., STRIPS in Fikes and Nilsson, 1971) used this second approach.

Hierarchies for problem-solving developed in an attempt to reap more substantial search reduction by using constraints cleverly. The first application of this idea is found in GPS, and is called "planning."[3] For example, as an aid to proving theorems in propositional calculus, solutions are first sought to a more abstract problem that retains only certain aspects of the real problem (differences in connectives, signs and order of symbols are ignored). This solution limits the choice of operators to apply in the search for a solution in the original space.

---

[2]This section uses the word "hierarchy" in a loose way to mean a succession of increasingly detailed abstract representations of a problem. The sequence need not represent a strict containment tree or domination of detailed levels by more abstract levels. I have no desire to take up the hierarchy/heterarchy debate.

[3]GPS thrust a technical definition upon this abstract noun. The emergence of robot problem-solving systems inevitably caused the term to be used in a looser sense to describe the generation of action sequences.

If no solution results, a different solution to the abstract problem is sought, and the process repeated (Newell and Simon, 1971, p. 428).

The next work that strongly evokes hierarchies is the ABSTRIPS problem-solver (Sacerdoti, 1974). This technique, born of the quasi-formal theorem-proving in STRIPS, seeks solutions first to simplified forms of the problem and uses these to guide a means-end analysis solution to the original problem. The simpler problems are constructed by dropping many of the antecedent conditions that must be satisfied in order that an operator may be applied. When a solution to the simpler problem is found, some of the dropped constraints are restored, and a solution to the more constrained problem is sought, constrained to visit the same solution-space points as the simple solution. ABSTRIPS uses a clever, but restricted, method to construct the abstract problems: why should the abstract problem definitions be so closely related to the detailed ones? (See Amarel, 1968, for a beautiful description of alternative non-intuitive abstractions of the missionary and cannibals problem.) Additionally, ABSTRIPS borrowed from STRIPS the limitations of an inaccessible planning control structure.

Sacerdoti remedied both of these difficulties in NOAH (Sacerdoti, 1975). The abstract versions of a problem are "programmed up" in a language called SOUP, similar in many ways to other AI languages. Rather than directly executing all SOUP code, however, the system partly executes code and partly examines, in a global fashion, the interactions among the constraints of the several GOALS attempted in the SOUP code. This analysis often establishes constraints on the order of application of the operators--all this is without intervention by the SOUP code. When the problem is satisfactorily "solved" at one level of detail, NOAH expands the steps of the plan in more detail by invoking SOUP code specified for each step. Thus, the selectivity in choosing planning paths is much the same as, say, MicroPlanner, but the control is entirely different: Sacerdoti saw that a breadth-first approach permits many problems of global constraint to be resolved before detailed analysis is attempted.

Execution monitoring. The need for execution monitoring arises because the states of the world induced during execution of a plan may differ from the states hypothesized during planning, consequently causing further execution to run amok. Generally, execution monitoring is accomplished by endowing the computer system with a complete world model, and by updating that model after every action is performed. Then the actual state of the world, as reflected in

the model, can be compared with the state anticipated by the planning system. Various kinds of processing can take care of differences between reality and anticipation: to detect failures and replan to recover; or to detect surprises that unexpectedly ease the problem solution.

The most renowned example of execution monitoring is the STRIPS PLANEX system (Fikes and Nilsson, 1971), which uses a representation of the plan called a "triangle table." The table records, for each action in the plan, a symbolic "kernel" that is compared with the world model to decide which step to execute next. If all goes "according to plan," these comparisons result in a simple sequential execution of the planned steps. If something goes seriously awry, no kernel will be satisfied, and replanning is indicated.

Hayes, in a travel planning program, also makes use of a perfect world model to invoke appropriate replanning when something is wrong (Hayes, 1975). The replanning effort is limited by redoing only those sections of a plan that have become invalid. The idea is to keep a data structure that records decisions made during the planning process, together with their dependency on other decisions and their relation to planning subgoals. When execution of a subgoal fails, it is therefore possible to identify and remove decisions that are inappropriate to the new situation, and to eliminate from the plan those steps that depend on the invalidated decisions. In a rather different way, this same idea is used in PEGASUS: when new information is available, replanning is limited to those subplans that are directly affected by the information. A dependency structure such as used by Hayes is not kept: it would be massively cumbersome in PEGASUS--consider the dependency of decisions on the numerical values in the utility function![4]

NOAH begins to relax the requirement for a perfect world model, updated after every step. If a particular action fails, NOAH probes around for the discrepancy in the world model, and replans, often by patching an existing plan, using domain semantics (SOUP code) tailored for this kind of error. An interesting aspect of this treatment is that the discrepancy between reality and the execution simulation of reality is allowed to grow quite large; consequently the

---

[4]Hayes actually dealt only with *positive* decision premises becoming invalid, e.g., cancellation of conveyances upon which the itinerary depends. A more difficult problem is dealing with negative premises: a decision to take a particular train depends on the non-existence of trains with better connections as well as on the existence of the chosen train. The PEGASUS updating scheme, although slower, is able to make such adjustments.

world model need not be constantly updated.[5]

An execution monitor may choose to limit its information-gathering requirement by updating certain parts of the world model very frequently, and ignoring the majority of the model. A simple example of such a technique is a servo control, as might be used to effect the positioning of a mechanical arm. A more complex example is the use of feedback around simple assembly strategies such as a spiral search for a hole using axial position feedback (Finkel, *et al*, 1975; Taylor, 1976). Bolles has investigated numerous ways of limiting vision processing required to close such feedback loops (Bolles, 1976).

5.3 Decision Theory in Robotics

This dissertation is by no means the first work that observes the problems of incorporating cost and reliability into planning systems and seeks relief using ideas from decision theory. Munson (1971), in a speculative paper on robot problem-solving, thought that utility would be a powerful aid in developing strategies, but the ideas were never put into practice. Piper (1972) experimented with searching a homogeneous probabilistic graph (decision tree) to find plans of action. This work was an outgrowth of experiments with the Graph Traverser, itself an experiment in general-purpose problem-solving. Hart (1969) has pointed out, however, that searches of probabilistic trees have rather poor performance properties (admissible searches, guaranteed to find the optimum path, are nearly exhaustive). This is one of the reasons PEGASUS uses a hierarchical organization to search deterministic graphs first, thus bounding search in a probabilistic space.

The most effective uses of decision theory in robotics have all appeared in recent vision systems. It is not surprising that decision theory has been applied to these problems, partly because vision problems are often attacked with numerical techniques, partly because of similarities between vision and statistical decision problems, and partly because of the appeal of allocating sensibly the large amount of computation these systems require. Three vision developments are especially noteworthy:

[5]NOAH's actions are intended to be interpreted and executed by a human. To an action that he believes he has executed successfully, the operator responds "OK." To one that has failed, he responds "CAN'T." Of course, the "OK" answers have the effect of updating the model so that the following step is executed next, but the right way to look at these answers is as shorthand for "I was able to carry out the command you gave me, as I understood it," *not* "I have verified that all consequences of my action match the expectations of the planner."

- Yakimovsky (Yakimovsky and Feldman, 1974) developed a system for region-growing that works from a model of the class of images presented to the system. The model provides estimates of the liklihood of competing region-growing decisions. The system searches the space of possible region shapes and interpretations by trying to maximize the liklihood that a particular solution is the proper one. The key to this system is that the model could express "semantic" relationships, e.g., "a region labeled *sky* is most likely to be above a region labeled *grass*."

- Garvey and Tenenbaum (1974) expanded these ideas to apply to scene analysis and to the reduction of effort in locating specific objects in a scene. Likely positions of objects (e.g., pictures are high on walls; telephones are on tables) are used, together with estimates of the cost of making certain visual discriminations (e.g., color, orientation of a surface), to plan a sequence of vision operators to find the requested object.

- Bolles (1976), in a most ambitious system for "verification vision," has made extensive use of statistical detection criteria. His job is to reduce to a given level the uncertainty of some crude measurement; he selects vision operators based on their cost and their empirically observed power to reduce uncertainty. He specifically copes with the powerful geometric constraints available in most verification: the number of degrees of freedom is usually much smaller than would be computed assuming all features and objects are independent. This system is a fine example of the combination of symbolic (geometric) constraints with decision-theoretic criteria.

5.4 Potential Applications

It is worthwhile to speculate briefly on how some of the techniques presented in this dissertation might benefit current AI systems. The observations divide into two main categories: better allocation of planning effort, with attention to the cost of planning, and more use of utility or other objective functions.

- Hearsay II (Lesser, *et al*, 1975). The Carnegie-Mellon speech understanding system drives a search to analyze a speech wave with a partially numerical process. Hypotheses from various sources are recorded in a global "blackboard," linked together by dependency relations, and tagged

with "attention focussing" numbers. A uniform procedure propagates focus information among the hypotheses, according to dependencies. Processing of the hypotheses is scheduled by a multi-process scheduler, driven by the attention-focussing tags and capable of allocating one of several processors to a task. The attention-focussing markers are apparently used both to express the validity of a hypothesis and to direct processing effort. No attempt is made to combine the cost of processing with the validity, but simply to use validity as a way of imposing selectivity in the search.

- SRI/SDC speech system (Paxton, 1977). Paxton has made several empirical studies of the effectiveness of different heuristics for controlling the effort in this speech analysis system. He was concerned with issues such as: Should processing be focussed by inhibiting alternative choices? The answer turns out be depend on the false-alarm rate of acoustic matching processes. These measurements contain the germ of a cost-of-planning model for the system that could apply the information uniformly: should effort be put in reducing false alarms to permit focus? Because of the large variance in input utterances, it is unwise to encode the results of these studies as heuristics that say. in effect, "in situation a use method b;" the characterization of situations is too coarse to permit unequivocal decisions. Once again, we find an application for numerical tradeoff among several approaches.

- MYCIN (Shortliffe, 1974). MYCIN is a decision-making framework applied to diagnosing bacteremia. Semantics of the domain, painstakingly acquired from physicians, are encoded in a data base and used to form implications (e.g., if a and ~b then c), with a numerical qualification on the confidence of the implication (a "certainty factor," CF). The confidence measure, only loosely related to probability, is used as the basis for comparing two or more implications of the available evidence, and consequently for suggesting the "best" diagnosis and treatment. The CF attempts to combine several effects (e.g., empirical probability, expressions of caution by physicians because of the severity of a missed diagnosis, etc.). What is entirely absent from MYCIN is any notion of *value*: the relative consequences, to the patient, of various diagnoses, the costs of laboratory tests, etc. Ginsberg (1969) investigated a utility function that reflects patient's preferences in similar situations.

- NOAH (Sacerdoti, 1975). The NOAH framework offers an excellent opportunity to experiment with some of the ideas presented here in a general-purpose setting. NOAH has well-developed structures for hierarchical planning and for invoking procedures that expand crude plans into more detailed ones. Adding a mechanism to report resource assignments and therefore to calculate utility information would be straightforward. A more substantial modification will be needed to permit concurrent exploration of a number of alternative plans.

Other applications in Computer Science

Computer systems are getting bigger. This is true not only of AI programs, but also of such apparently trivial applications as text editors. Some of the growth can be ascribed to growing requirements: an AI program may attempt to solve an intricate problem; a text editor may acquire complexity because it attempts to offer a very pleasant experience to its user; or a command and control system may grow as requirements increasingly integrate diverse aspects of the problem into one system.

But an additional generator of growth is particularly interesting: as the repertoire of computer solution techniques grows, designers attempt to devise general-purpose programs that can cope efficiently with ever broader classes of input problems. For example:

- "Hidden surface elimination" is a problem in computational geometry often associated with computer graphics (Sutherland, *et al*, 1974). There is no solution that is "best" for all geometrical and topological situations that may arise. Even the details of the solution techniques, particularly the selection of certain sorting and computational geometry algorithms, may depend on the statistics of the problem presented for solution and on the exact form of output desired.

- Computer-controlled manipulators require an intricate program of control signals to accomplish an assembly task. Many aspects of the task can be planned in advance, given specifications of the assembly and of the geometry of the pieces. Some aspects of planning must await more precise information about locations of the parts. Some of the procedures cannot be planned in detail at all, because of cumulative manipulator

errors; local iterative procedures, such as a search to insert a screw in a hole, must be used (Taylor, 1976).

- How should a function be integrated numerically? There are many techniques, of varying applicability, speed, and precision.

- The range of computer language translators, extending from interpreters to compilers to very sophisticated optimizers, encompasses a myriad of techniques. Much is known about optimization tricks, but an optimizing compiler may be expensive. At the other end, an interactive user demands flexibility, but expects adequate efficiency.

The interesting aspect of these examples is that solution techniques abound; the problem is to build systems that plan a proper application of the techniques to achieve an efficient solution. All of the considerations explored in this dissertation are relevant: We need to plan in the presence of uncertainty about many of the problem details. We need to assess tradeoffs among different techniques with differing reliabilities and costs. We need to balance the effort invested in planning with that devoted to achieving the solution itself. We need to devote a judicious amount of effort to monitoring execution progress and to replanning if necessary. As we learn more about the individual solution techniques and about planning techniques, we can expect to build computer systems that plan the proper combination of techniques to solve classes of problems.

Chapter 6
Conclusions

6.1 Summary

This dissertation has explored a conjunction of AI and decision-theoretic techniques, especially as applied to two case studies. Our aim has been to use a judicious amount of mechanism to achieve *good* solutions to problems. The utility function proves useful for:

- Comparing the value of alternative plans, including expressing tradeoffs among multiple goals. The plans may be plans of action, plans to acquire information, or plans to engage in planning activities.
- Expressing the effects of uncertainty and risk consistently for all plans, so that plan evaluation includes these effects.
- Exposing to the designer or user crucial parameters that affect the system's decisions.

The utility functions we devised in the two case studies exhibit certain properties tailor for application in a problem-solver:

- Additivity of a "resource vector" that represents the plan-specific parameters for the utility calculation permits inexpensive incremental updates to the utility as a plan is built and modified.
- Monotonicity simplifies arguments used to construct resource vectors that give rise to upper bounds on the utility of partially complete plans.

Used in a problem-solver, the utility function on plans and partial plans helps apply various solution techniques:

- Search. The utility function provides a numerical guide to control search. Calculation of upper bounds on partial plans is important for this use.
- Elaboration. The utility of a plan can be used to assess the benefits of modifications or extensions to a plan that are thought to improve it.
- Hierarchies. The utility function provides a way to organize a hierarchy. Solutions at one level of detail furnish symbolic and utility "templates" to work on more detailed solutions. The utility of a solution at one level becomes the guide at the next level, used to bound incomplete plans and to control search.

- Heuristics and specialists. The utility function allows the peaceful cooperation of different solution techniques by subjecting the efforts of all to a uniform evaluation. A low value of the utility of a plan generated by a heuristic that "is not good enough" will suggest invoking another available solution technique.

The utility function also helps to allocate resources to planning, acting and information-gathering. In particular, a model of the cost of planning is used to:

- Allocate planning resources to alternative processing techniques with regard for their efficiency: their ability to generate high-utility solutions with low expenditures of planning resources.

- Engage in only enough planning to find the P-optimal solution, i.e., the optimal solution to the problem, given that planning costs must be charged against the final solution.

If we characterize problem-solving as a massive topological sort through a problem-space, the utility evaluation emerges as one technique for computing the sorting order. Symbolic constraints will cause great sections of the space to be excised from further consideration. Extremely low utility bounds will similarly discard other areas. In the remaining regions, the utility and symbolic "sorting keys" are developed to greater and greater precision as attention focusses on the solution.

6.2 Suggestions for Further Work

Systems. The approach presented in this work could influence the design of problem-solving systems able to use utility information. Current AI practice would encourage these techniques to be designed into a *language.* As mentioned in Chapter 5, a receptive host language might be Sacerdoti's NOAH. If this is to be attempted, a number of needs revealed by PEGASUS must be addressed; not all require innovation:

1. Many individual problems are being pursued at once. As increasing attention has been placed on procedural methods for reducing search, or for representing search implicitly in a control stack, facilities for conventional searching have atrophied. The whole thrust of attempts to optimize requires examining several solutions; these are often alternative

instantiations of plan outlines. The need to keep several solution tasks in progress is also responsible for several other requirements in this list.

2. Means must be provided to allocate computing resources to the various plans being pursued. Each plan could be represented as a process; a scheduler could be used to control the planning.

3. Access to the plan processes is needed for various purposes. A task scheduler will ask for current upper bound evaluations in order to allocate effort. Re-evaluation requires enumerating plans and applying a plan-specific procedure to effect updates. Some planning techniques may require access to current plans in order to locate an alternative (e.g., for a failure recovery).

4. The massaging of plans as a consequence of execution is especially taxing. Regardless of the suspended state of planning on each problem, the problem itself must be transformed into a new one, in which the first, now executed, step is discarded (or, at more abstract levels, simply altered). As initial parts of plans are removed, it is possible that remaining tails of two or more plans become identical: duplicate plans must be removed from the processing schedule. The exact notion of "identical" will vary from abstraction to abstraction--plans are identical at level SP if they use the same route; at level INSTANTIATE the same trip choices are required as well.

5. Incremental changes. One lesson of the PEGASUS experiment is the chaos that results when clean mechanisms for effecting incremental changes are absent. Whereas most AI planning programs have limited the "changeable" parts of the state of nature to the direct effects of planned actions, PEGASUS attempts to withstand changes in utility model (i.e., goal structure), in available conveyances, in weather conditions, etc. This problem is so widespread that the "demons" of AI languages or "continuously evaluating expressions" offer solutions that are too bulky. Part of the answer lies in designing searching and planning procedures that can cope reasonably with new data (e.g., "an extra section of flight 103 is being added"), regardless of the progress of the search, and then expressing these with uniform conventions, such as the class/instance paradigm.

6. Sharing. PEGASUS uses fairly crude and replicative data structures for describing plans that often have common sub-pieces. Although a good deal of sharing is practical within levels (e.g., SP), as plans are packaged for shipment to other levels, they are copied. It is tempting to design a canonical plan representation to achieve more sharing (as in, for example, QA4, Rulifson, *et al*, 1972). There are at least two problems with such a design: the equality checks mentioned above are not made significantly easier, and, more importantly, there is no canonical plan-building direction (unlike the building of lists by CONS) -- some solution techniques work forward, some backward, some in both directions.

Cost of Planning. PEGASUS makes only beginning attempts to model planning costs. One class of extensions is trivial: adding to the repertoire of planning resources *measured*: computer time, access to data bases (suppose each access to a travel guide cost $.10), storage utilization, etc. Refinements here can lead to more precise models of the cost of computation, but not to more effective uses of the information.

However, power lies in being able to estimate future effects of planning. This information was used in PEGASUS to curtail planning, but not to choose among alternative problem-solving techniques. More effective *predictive* cost-of-planning models can certainly be built. Generally, these are statistical models that map certain features of the plan and problem-solving system into estimated costs. The features may include aspects of the plan (e.g., number of steps, current level of detail) and aspects of the planner (e.g., estimated branching factors in searches, enumeration of as yet untried specialized planning tricks, etc.)

Admissibility. The framework we have developed here requires a notion of admissibility that is broader than normally given: it must include the effects of the cost of planning. Corresponding to searches for P-optimal plans, there is a criterion of P-admissibility. A solution technique will be P-admissible if it finds a solution with maximum utility of execution and planning combined. This criterion permits more than purely numerical tradeoff of solution and planning costs: in cases of large planning costs, we can even use *incomplete symbolic methods*. Allowing incomplete methods may permit approximate solutions to some of the "very hard" problems (e.g., NP-complete problems) to be P-optimal. Of course, computers generate only approximate solutions to many problems presented to them (e.g., finite precision arithmetic represents an approximation in

many cases), but the notion of P-admissibility will determine what sorts of approximations are satisfactory.

P-admissibility may yield solutions at varying levels of detail. A P-optimal solution from a hierarchy of abstractions might not necessarily have completed processing at the most detailed level. This is another sort of approximation: in some cases we will obtain more abstract solutions than in others.

The problem, of course, is that we have no techniques for making effective use of the P-admissibility notion in the most general sense: to justify approximate symbolic or abstract solution techniques in the presence of planning costs. P-admissibility has some non-intuitive implications: Should a technique that is executed on a faster or less expensive computer be required to find a better solution?

In many ways, P-admissibility is what AI is all about. It recognizes that optimal solutions to computationally outrageous problems are not useful. AI has, in effect, loosely defined P-admissibility by using techniques that attempt to get good solutions most of the time for modest computational investments. AI is concerned with the *design* of methods to achieve these solutions: heuristics, hierarchies, bounds arguments, and so forth.

6.3 Decision Theory and Artificial Intelligence

Some readers will have already objected that our suggestions do not increase the range of problems solvable by decision theory or symbolic processing, that each is a powerful and complete paradigm, and that our remarks bear on efficiency considerations alone. The pure symbolic processor claims that he can achieve optimization effects by dividing numeric ranges into a small number of "symbolic values" (e.g., temperature into COLD, COOL, WARM and HOT) that suffice for a given problem. Information about tradeoffs can be encoded as a set of symbolic preferences: (FED and WALKED-A-LONG-DISTANCE) is preferred to ((not FED) and WALKED-A-SHORT-DISTANCE). Or he will assess tradeoffs numerically by instantiating theorems of number theory, analysis and algebra. This gives rise to crude and awkward models in cases where a small amount of numerical processing is more natural and accurate.

The pure mathematical programmer, on the other hand, will mathematize all constraints or move complexity into value or reward functions. He will

formulate any search as a shortest path problem with appropriate arc weights and propose dynamic programming to calculate a solution. The result is often a huge state space for very simple problems, making numerical solution simply infeasible.

Practitioners of either field adopt more moderate approaches: the AI designer finds many problems suited to partially-numerical approaches. Similarly, the decision theorist engages in a substantial amount of symbolic reasoning to formulate his model and to apply it intelligently to the situation; he may also use "heuristic" solution techniques on large problems. A human analyst will perform the reasoning required to build a decision tree intelligently, one that represents sensible plans. From the point of view of AI, this construction process is itself an endeavor of interest.

From the point of view of decision theory, our formulation aims to permit a computer program to emulate a good decision analyst. Such an analyst combines formulating plans and searching decision trees to arrive at a solution. A good analyst will monitor the implementation of the decisions, keeping abreast of exogenous changes in the utilities on which his solution was based, formulating additional plans, etc. This is in contrast to conventional computer programs used to search one static tree exhaustively.

From the point of view of AI, the advantage of decision theory is the ability to find solutions that are "optimal" in some model. Although the approach requires a certain amount of search to find solutions, we have shown several powerful methods to limit the search:

- The symbolic problem solver constrains the search later undertaken to perfect a strategy. A rough plan generated in simplified, abstract space, can be used to constrain the more careful planning. These are basic search-limiting methods of AI not practiced in decision analysis programs.

- A number of decision-theoretic techniques limit search. Branch-and-bound methods limit search based on bounds derived from the utility models. In addition, one can *prove* that the failure F_1 in Figure 2-6 should not include paths that persist in using the same box (i.e., paths that disregard the outcome of the test): every such strategy is dominated by one that simply does not perform the test at all. Such "utility theorems" limit search.

Another example of search limiting occurs when the plan outline specifies a loop. Any paths that involve loops continue to incur increased costs as they are expanded, but the ultimate utility is fixed. The loop thus expends effort without approaching the goal; such paths will be cut off by the branch-and-bound algorithm.

- Domain-independent heuristics can be applied to limit search. One such heuristic is to explore paths of high probability first, and perhaps be willing to bound pessimistically those paths of low probability. Although pure decision theory looks dimly on this technique because even paths of low probability may have unbounded utilities, in many cases we can meaningfully assign bounds to the utilities.

  Certain of the recovery mechanisms, e.g., using another top-level alternative, are domain-independent, as is the method of approximating the utility of such an alternative plan.

- Domain-dependent heuristics can limit search. Although these techniques may require a certain amount of reprogramming for each new domain, they are probably far more powerful than domain-independent methods. The current AI trend toward knowledge-based systems (Nilsson, 1974; Bobrow and Collins, 1975; Fikes, 1976) is due in part to benefits of distributing domain knowledge throughout systems.

Search-limiting heuristics are not without drawback -- the resulting search may not guarantee finding the optimal solution, i.e., it is not admissible. However, the utility measure still allows us to extract the best plan among those developed in the search.

The Combination

What the two fields of decision theory and artificial intelligence offer is a collection of techniques that can be applied judiciously to solve problems. There are cases when decision theory applied to the problem domain adds little to AI techniques, but may still offer help in allocating planning resources:

- *Insignificant Costs.* The benefit of optimal planning may simply be too low if the costs of the planning and execution are themselves insignificant or if the planning costs greatly exceed the execution costs.

- *Identical Values.* A problem may give rise to solutions of identical preference. A theorem-proving program may not be at all concerned with finding the shortest proof or with the expense of the search. A program that attempts to "understand" a paragraph of natural language in order to answer questions about it is likewise not concerned with optimization but with capturing a conceptual structure. In these cases, the utility function on outcomes is nearly constant, and gives no information to the search.

  Both of these examples are characterized by the intuition that the domain is inherently symbolic: the understanding problem is to build a conceptual structure that is communicated as a string of words; the theorem-proving task, even as practiced by humans, is primarily symbolic manipulation. There are notions of "best" solutions in both cases, but they are second-order considerations.

  However, these examples are confronted with problems in allocating solution effort, and we can still hope to find the "best derivation" of the solution. Humans too may find processing resources limited when performing these chores, and must settle for approximate solutions (Norman and Bobrow, 1974).

- *Modeling Difficulties.* It may be very difficult to construct a utility and probability model that applies to the problem. Although the central theorem of decision theory shows that any choice of a "best" plan is an implied assessment of utilities and probabilities, it still may be difficult to cast the model in numerical terms.

  A particularly painful aspect of this problem is presented by Bayes' rule: if we use the rule to calculate the probability distribution resulting from a sequence of tests, a potentially huge number of conditional probabilities (or probability distributions) is required. This difficulty, coupled with that of extracting probability information from humans, has led to several alternative "rules of inference" for computing likelihood information based on test outcomes (e.g., Shortliffe, 1974). This is an important current research topic.

But there are also ways in which decision theory adds considerable power:

- *Convenient Representation.* Utility and probability models are often

convenient ways of representing parameters of a problem; they thereby ease parameter modification by a designer or by a user with a slightly different problem. For example, if a vision operator is modified to use a faster algorithm and therefore less computer time, a small modification to the utility model will suffice to alter the performance of an entire vision system correctly. It would be less obvious how to modify a set of symbolic heuristics that governs the application of the operator.

A simple utility function may express the tradeoffs among the various resources the system consumes (money, elapsed time, etc.). The information that governs the tradeoffs the system actually makes is thus localized and easily modifiable. Some such modifications can be made by the system itself in reaction to complaints about its behavior; the changes could require only simple numerical calculations to compute new parameters for the utility model. It is less obvious how a program should itself "learn" heuristics.

Finally, because decision theory is continually being applied to real-world problems, new models are built, refined and used. For example, efforts are underway to provide doctors with decision-theory models to help plan the diagnosis and treatment of various diseases (Pauker and Kassirer, 1975; Ginsberg, 1969). Computer aids to such decision-making can take advantage of the models.

- *Ubiquity of Planning.* Such models are not limited to application in traditional "AI" domains. For example, an optimizing compiler embarks upon substantial symbolic reasoning to plan efficient object code for a program; sophisticated optimizers measure or estimate how often a section of code is executed and use this as an estimate of the utility of an optimization. An extended utility structure would permit trading off different forms of optimization and including the user's utility function. Automatic programming, and in particular automatic coding (Low, 1974; Rovner, 1976), seem to involve the same kinds of planning and elaboration mechanisms presented here.

- *Optimal Planning.* A decision-theoretic model of a planning process itself can be used to make planning decisions and thus to control allocation of effort to planning tasks. Many AI programs such as planners, problem-

solvers, parsers, and "understanders" require such guidance in the application of available methods: Is it more important to plan further ahead or to investigate detail of the current plan? (Sacerdoti, 1974) How far should consequences of a situation be investigated? (Rieger, 1975) Increasingly, this problem becomes one of controlling a number of processes, which are "triggered" by various changes in the world model, and which are responsible for exploring consequences of the change (Bobrow and Winograd, 1977). If two alternative parsings of a sentence appear similar in a crude analysis, should one be examined in detail, or should both be explored uniformly? (Paxton and Robinson, 1973) How are alternative hypotheses pursued? (Woods, 1974)

Even if the plans themselves have nearly constant utility, optimal planning is useful. For example, in a theorem prover, we are given a set of clauses and must decide which of several resolutions to make; if we can calculate the cost of planning a solution from a given set of clauses, we choose the resolution that gives rise to the lowest planning cost. Thus although the space of outcome utilities is constant, the utilities of various alternative planning approaches are not. This second space has been important to the development of search programs; it corresponds for example to the evaluation functions in game-playing programs.

When the costs, uncertainties, and outcomes of the planning process itself are considered in controlling a planning and execution system, the system does "optimal planning." Although the plans generated may not be optimal, the entire process, including planning, is optimal. This suggests an extended notion of admissibility that includes consideration of planning costs.

- *Detection Problems.* AI has embraced a number of problems that have significant *detection* components: speech understanding and machine vision are the most obvious examples. The problems of efficient detection, and especially of uncertainty in the results, are at the heart of decision theory. In an AI setting, the knowledge gained from detection operations must be incorporated into higher-level reasoning that has significant symbolic components. It is perhaps in these problems that the approach we propose is most advantageous, for it unifies inherently numerical computation (detection) with symbolic reasoning (understanding). Indeed it

is these areas that gave rise to the approach and saw early applications (Yakimovsky and Feldman, 1974; Garvey and Tenenbaum, 1974; Tenenbaum, 1973; Bolles, 1976).

As computer systems generally, and AI programs particularly, become larger and more complex, they make more internal choices among available methods to attack the stated problem. This is a natural consequence of increased understanding of computer algorithms and of desires to make programs more general in purpose. Even if we are reluctant to impose numerical models on the solution space, we cannot neglect opportunities to measure and guide the choices these systems make to generate solutions to the problem.

Bibliography

Abbreviations

IJCAI: International Joint Conference on Artificial Intelligence.

FJCC: Fall Joint Computer Conference.

Amarel, S., "On Representations of Problems of Reasoning About Actions," *Machine Intelligence 3*, American Elsevier, New York, 1968.

Arrow, K. J., *Social Choice and Individual Value*, Wiley, New York, 1951.

Bellman, R., *Dynamic Programming*, Princeton Univ. Press, Princeton, N.J., 1957.

Bobrow, D. G., "Dimensions of Representation," Xerox Palo Alto Research Center, in Bobrow and Collins, 1975.

Bobrow, D. G. and Collins, A. (eds) *Representation and Understanding*, Academic Press, New York, 1975.

Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., and Winograd, T., "GUS, a Frame-Driven Dialog System," *Artificial Intelligence*, Vol. 8, No. 1, Spring 1977.

Bobrow, D. G. and Raphael, B., "New programming languages for AI research," *Computing Surveys*, Vol. 6, No. 3 (September 1974).

Bobrow, D. G., and Wegbreit, B., "A Model and Stack Implementation for Multiple Environments, "*Communications of the ACM*, Vol. 16, No. 10, October 1973.

Bobrow, D. G. and Winograd, T., "An overview of KRL, a Knowledge Representation Language," *Cognitive Science*, Vol. 1, No. 1, 1977.

Bolles, R., "Verification Vision with a Programmable Assembly System," Computer Science Dept., Stanford University, 1976.

Buchanan, B. G., Sutherland, G., and Feigenbaun, E. A., "Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry," *Machine Intelligence 4*, American Elsevier, New York, 1969.

Chernoff, H. and Moses, L., *Elementary Decision Theory*, Wiley, N.Y., 1959.

Cozzolino, J. M., "Search for an unknown number of objects of nonuniform size," *Operations Research*, Vol. 20, No. 2 (March-April 1972), p. 293.

Fahlman, S., "A planning system for robot construction tasks," *Artificial Intelligence*, Vol. 5, No. 1 (Spring 1974).

Feldman, J. A., "A Programming Methodology for Distributed Computing (among other things)," Computer Science Dept., University of Rochester, TR-9, 1976.

Feldman, J. A., Pingle, K., Binford, T., Falk, G., Kay, A., Paul, R., Sproull, R., and Tenenbaum, J., "The Use of Vision and Manipulation to Solve the Instant Insanity Puzzle," *Proceedings Second IJCAI*, 1971.

Feldman, J. A., Low, J., Taylor, R., and Swinehart, D., "Recent developments in SAIL -- An ALGOL-based language for artificial intelligence," *Proceedings FJCC*, 1972.

Feldman, J. A. and Sproull, R. F., "System Support for the Stanford Hand-Eye Project," *Proceedings Second IJCAI*, 1971.

Feldman, J. A. and Sproull, R. F., "Decision Theory in Artificial Intelligence II: The Hungry Monkey," Computer Science Department, University of Rochester, TR-2, 1974. To appear in *Cognitive Science*.

Fennel, R. D., "Multiprocess Software Architecture for AI Problem-Solving," Dept. of Computer Science, Carnegie-Mellon University, 1975.

Fikes, R. E., "Knowledge Representation in Automatic Planning Systems," Tech. Node 119, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, 1976.

Fikes, R. E., Hart, P. E., and Nilsson, N. J., "Some New Directions in Robot Problem-Solving," *Machine Intelligence 7*, Edinburgh, University Press, 1972.

Fikes, R. E., and Hendrix, G., "A Network-Based Knowledge Representation and its Natural Deduction System," to appear in *Proceedings Fifth IJCAI*.

Fikes, R. E. and Nilsson, N. J., "STRIPS: A new approach to the application of theorem proving in problem solving," *Artificial Intelligence*, Vol. 2 (1971), p. 189. Use of A^* was revealed in a private communication.

Finkel, R., Taylor, R., Bolles, R., Paul, R., and Feldman, J., "An Overview of AL, A Programming System for Automation," *Proceedings Fourth IJCAI*, 1975.

Fishburn, P. C., "Modern Utility Theory: 1940-1970," Research Analysis Corporation, 1970. Also CFSTI AD 625047.

Garvey, T., "Perceptual Strategies for Purposive Vision," Stanford Research Institute, Technical Note 117, September 1976.

Garvey, T. and Tenenbaum, J. M., "On the automatic generation of programs for locating objects in office scenes," *Proceedings Second Intern. Joint Conf. on Pattern Recognition*, IEEE 74-CH0885-4C (August 1974), p. 162.

Ginsberg, A. S., "Decision analysis in clinical patient management with an application to the pleural effusion problem. PhD Thesis, Stanford University, 1969.

Goldberg, A. and Kay, A., "Smalltalk-72 Instruction Manual," Xerox Palo Alto Research Center, Report SSL 76-6, 1976.

Grape, G., "Model based (intermediate level) computer vision," PhD Thesis, Computer Science Dept., Stanford University, Stanford, California, 1973.

Green, C. C., "The Application of Theorem Proving to Question-Answering Systems," Computer Science Dept., Stanford University, AIM-96, Stanford, California, 1969.

Hart, P. E., "Searching Probabilistic Decision Trees," Tech. Note 2, AI Group, Stanford Research Institute, Menlo Park, California, 1969.

Hart, P., Nilsson, N. J. and Raphael, B., "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Sys. Sci. Cybernetics*, Vol. SSC-4, No. 2 (July 1968), p. 100.

Hayes, P., "A Representation for Robot Plans," *Proceedings Fourth IJCAI*, 1975.

Knuth, D. E., *The Art of Computer Programming; Sorting and Searching*, Vol. 3, Addison-Wesley, 1973.

Lawler, E. and Wood, D., "Branch and bound methods: a survey," *Operations Research*, Vol. 14, No. 4 (July 1966), p. 699.

Lesser, V. R., Fennel, R. D., Erman, L. D., and Reddy, D. R., "Organization of the Hearsay II Speech Understanding System," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1975, ASSP-23.

Low, J. R., "Automatic coding: choice of data structures," Stanford Artificial Intelligence Project, AIM 242, Stanford, California, 1974.

Matheson, J. E., "The Economic Value of Analysis and Computation," *IEEE Transactions on Systems Science and Cybernetics*, SSC-4, 3, 325 (September 1968).

McDermott, D. V., and Sussman, G. J., "The Conniver Reference Manual," AI Memo 259, MIT Project MAC, 1972.

Minsky, M., "Steps Toward Artificial Intelligence," 1961. Reprinted in Feigenbaum, E. and Feldman, J. (Eds.) *Computers and Thought*, McGraw-Hill, N.Y., 1963.

Munson, J. H., "Robot planning, execution and monitoring in an uncertain environment," *Proceedings Second IJCAI* (September 1971), p. 338.

Newell, A. and Simon, H., *Human Problem-Solving*, Prentice-Hall, Englewood Cliffs, 1971.

Newell, A. Shaw, J. and Simon, H., "Chess playing programs and the problem of complexity," *IBM J. Res. and Develop.*, Vol. 2 (October 1958), p. 320. Reprinted in Feigenbaum, E. and Feldman, J. (Eds.) *Computers and Thought*, McGraw-Hill, N.Y., 1963.

Nilsson, N. J., "Artificial Intelligence," *Proceedings IFIP Congress 1974*, p. 778.

Nilsson, N. J. *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, N.Y., 1971.

Norman, D. A., and Bobrow, D. G., "On Data-Limited and Resource-Limited Processes," Xerox Palo Alto Research Center, Report CSL 74-2, Palo Alto, California, 1974.

Pauker, S. G. and Kassirer, J. P., "Therapeutic Decision Making: A Cost-Benefit Analysis," *N Engl J Med*, 239:229-234, 1975.

Paxton, W. H., forthcoming PhD thesis, Stanford University, 1977.

Paxton, W. H. and Robinson, A. E., "A Parser for a Speech Understanding System," *Prceedings Third IJCAI* (August 1973), p. 216.

Piper, J., "Integrated planning and acting in a stochastic environment," MIP-R-94, School of AI, Univ. of Edinburgh, 1972.

Pohl, I., "Bi-directional and Heuristic Search in Path Problems," SLAC Report 104, Stanford University, Stanford, California, 1969.

Pohl, I., "The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving," *Proceedings Third IJCAI* (August 1973), p. 12.

Raiffa, H., *Decision Analysis, Introductory Lectures on Choices under Uncertainty*, Addison Wesley, Reading, 1970.

Rieger, C. J., "Conceptual Memory," in R. Schank (ed) *Conceptual Information Processing*, North-Holland, Amsterdam, 1975.

Rovner, P. D., "Automatic Representation Selection for Associative Data Structures," TR10, Univ. of Rochester, Computer Science Dept., September 1976.

Rulifson, J. F. Derksen, J. A. and Waldinger, R. J., "QA4: A procedural calculus for intuitive reasoning," Tech. Note 73, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, 1972.

Sacerdoti, E., "Planning in a hierarchy of abstraction spaces," *Artificial Intelligence*, Vol. 5, No. 2 (Summer 1974), 115-135.

Sacerdoti, E., "A Structure for Plans and Behavior," Tech. Note 109, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, 1975.

Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers, II, Recent Progress," *IBM J. Res. Develop.*, vol. XI, no. 6, November, 1967.

Schank, R. C., and Abelson, R. P., *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*, Erlbaum Associates, Hillsdale, New Jersey, 1977 (in press).

Shortliffe, E. H., "MYCIN: A rule-based computer program for advising physicians regarding anti-microbial therapy selection," AI Memo 251, Stanford Artificial Intelligence Project, Stanford, Calif. (October 1974).

Slagle, J. R., *Artificial Intelligence: The Heuristic Programming Approach*, McGraw-Hill, New York, 1971.

Slagle, J. R. and Lee, R. C. T. Application of game tree searching techniques to sequential pattern recognition. *Communications of the ACM*, Vol. 14, No. 2 (February 1971), p. 103.

Sussman, G. J., "A Computation Model of Skill Acquisition," AI TR-297, Massachusetts Institute of Technology, AI Lab, 1973.

Sussman, G. J., and McDermott, D. V., "From PLANNER to CONNIVER--a genetic approach," *Proceedings FJCC*, AFIPS Press, Montvale, 1972.

Sussman, G. J., and Winograd, T., and Charniak, E., "MicroPlanner Reference Manual," AI Memo 203, MIT Project MAC, 1970.

Sutherland, I. E., Sproull, R. F., and Schumacker, R. A., "A Characterization of Ten Hidden-Surface Algorithms," *Computing Surveys*, Vol. 6, No. 1, April 1974.

Taylor, R. H., "A Synthesis of Manipulator Control Programs from Task-Level Specifications," AIM-282, Computer Science Dept., Stanford University, 1976.

Teitelman, W., "InterLisp Reference Manual," Xerox Palo Alto Research Center, Palo Alto, California, 1975.

Tenenbaum, J. M., "On locating objects by their distinguishing features in multisensory images," *Computer Graphics and Image Processing*, Vol. 2, No. 3/4 (December 1973).

Tversky, A. and Kahneman, D., "Judgement under uncertainty: heuristics and biases," *Science*, Vol. 185, No. 4157 (27 September 1974), p. 1124.

Winograd, T., *Understanding Natural Language*, Academic Press, New York, 1972.

Winston, P. H., "The MIT Robot," *Machine Intelligence 7*, Edinburgh, University Press, 1972.

Woods, W. A., "Motivation and overview of BBN Speechlis: An experimental prototype for speech understanding research," *IEEE Symposium on Speech Recognition*, IEEE 74-CH0878-9 AE (April 1974).

Yakimovsky, Y. and Feldman, J., "Decision Theory and Artificial Intelligence: I. A Semantics-based Region Analyzer," *Artificial Intelligence*, Vol. 5, 349-371 (1974).

Appendix
Trace of PEGASUS Execution

This appendix presents a trace of the execution of PEGASUS on a simple problem. The trace is intended to elucidate several points about the operation of the planner:

- How the utility measures applied to all levels are used to control the planning.
- How the cost of planning changes the outcome of the planner, and how the information is used.
- How many alternatives are considered, at least in part.

PEGASUS is given the problem of designing an itinerary to travel from HOME (Palo Alto, California) to UR (University of Rochester, Rochester, N.Y.), given the constraint to leave home after 7:00 June 14. The salient features of the model of the client are:

Utility function = 1000 - Money -.33 Time -.1 Stress - Misc
Cost of planning = -.3 * ComputerTime (measured in seconds)
Quality of time contributions to stress:
Home: 0
all other spots: .03/minute
Air: .05/minute
Train: .1/minute
Bus: .1/minute

The itineraries planned by PEGASUS are:

Itinerary 1. (Tasks 0,1,2,3,4,5,6) Utility: 633.2546
Main path: probability=.5939, U=676.16
TAXI*. From HOME to SFO. Departure JUN 14 8:13 transit time 0:22 ($12.96).
Layover at SFO starting JUN 14 8:35 lasting 0:10.
AA 92. From SFO to DTW. Departure JUN 14 8:45 transit time 4:03 ($139.0).
Layover at DTW starting JUN 14 15:48 lasting 1:02.
AA 92. From DTW to ROC. Departure JUN 14 16:50 transit time 0:57 ($17.0).
Layover at ROC starting JUN 14 17:47 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 14 18:02 transit time 0:01 ($1.42).
Arriving at destination JUN 14 18:03.

Backup: probability = .1684, U=616.07
Layover at SFO starting JUN 14 8:35 lasting 3:25.
UA 86. From SFO to DTW. Departure JUN 14 12:00 transit time 4:03 ($139.0).
Layover at DTW starting JUN 14 19:03 lasting 0:02.
AL 736. From DTW to ROC. Departure JUN 14 19:05 transit time 1:10 ($17.0).
Layover at ROC starting JUN 14 20:15 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 14 20:30 transit time 0:01 ($1.42).
Arriving at destination JUN 14 20:31.

Backup: probability = .2498, U=623.27
Layover at DTW starting JUN 14 15:48 lasting 3:17.
AL 736. From DTW to ROC. Departure JUN 14 19:05 transit time 1:10 ($17.0).
Layover at ROC starting JUN 14 20:15 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 14 20:30 transit time 0:01 ($1.42).
Arriving at destination JUN 14 20:31.

Itinerary 2. (Tasks 0,1,8,14,15,16,17) Utility: 638.697
Main path: probability = .7391, U=679.81
TAXI*. From HOME to SJC. Departure JUN 14 7:16 transit time 0:14 ($8.28).
Layover at SJC starting JUN 14 7:30 lasting 0:15.
UA 464. From SJC to ORD. Departure JUN 14 7:45 transit time 3:45 ($126.0).
Layover at ORD starting JUN 14 13:30 lasting 0:55.
UA 362. From ORD to ROC. Departure JUN 14 14:25 transit time 1:27 ($30.0).
Layover at ROC starting JUN 14 16:52 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 14 17:07 transit time 0:01 ($1.42).
Arriving at destination JUN 14 17:08.

Backup: probability = .1032, U=565.57
Layover at SJC starting JUN 14 7:30 lasting 5:35.
UA 356. From SJC to ORD. Departure JUN 14 13:05 transit time 3:55 ($126.0).
Layover at ORD starting JUN 14 19:00 lasting 0:30.
AA 214. From ORD to ROC. Departure JUN 14 19:30 transit time 1:25 ($30.0).
Layover at ROC starting JUN 14 21:55 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 14 22:10 transit time 0:01 ($1.42).
Arriving at destination JUN 14 22:11.

Backup: probability = .1381, U=635.80
Layover at ORD starting JUN 14 13:30 lasting 3:00.
AA 500. From ORD to ROC. Departure JUN 14 16:30 transit time 1:24 ($30.0).
Layover at ROC starting JUN 14 18:54 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 14 19:09 transit time 0:01 ($1.42).
Arriving at destination JUN 14 19:10.

Itinerary 3. (Tasks 0,1,7,10,12,20,21) Utility: 649.1374
Main path: probability = .6101, U=682.48
TAXI*. From HOME to SFO. Departure JUN 14 9:38 transit time 0:22 ($12.96).
Layover at SFO starting JUN 14 10:00 lasting 0:05.
AA 182. From SFO to ORD. Departure JUN 14 10:05 transit time 3:44 ($126.0).
Layover at ORD starting JUN 14 15:49 lasting 0:41.
AA 500. From ORD to ROC. Departure JUN 14 16:30 transit time 1:24 ($30.0).
Layover at ROC starting JUN 14 18:54 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 14 19:09 transit time 0:01 ($1.42).
Arriving at destination JUN 14 19:10.

Backup: probability = .1852, U=646.63
Layover at SFO starting JUN 14 10:00 lasting 0:15.
UA 126. From SFO to ORD. Departure JUN 14 10:15 transit time 3:55 ($126.0).
Layover at ORD starting JUN 14 16:10 lasting 1:30.
UA 794. From ORD to ROC. Departure JUN 14 17:40 transit time 1:30 ($30.0).
Layover at ROC starting JUN 14 20:10 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 14 20:25 transit time 0:01 ($1.42).
Arriving at destination JUN 14 20:26.

Backup: probability = .2142, U=655.30
Layover at ORD starting JUN 14 15:49 lasting 1:51.
UA 794. From ORD to ROC. Departure JUN 14 17:40 transit time 1:30 ($30.0).
Layover at ROC starting JUN 14 20:10 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 14 20:25 transit time 0:01 ($1.42).
Arriving at destination JUN 14 20:26.

Itinerary 4. (Tasks 0,1,7,11,13,22,23) Utility: 595.3527
Main path: probability = .6893, U=675.94
TAXI*. From HOME to SFO. Departure JUN 14 14:18 transit time 0:22 ($12.96).
Layover at SFO starting JUN 14 14:40 lasting 0:20.

AA 222. From SFO to ORD. Departure JUN 14 15:00 transit time 3:49 ($126.0).
Layover at ORD starting JUN 14 20:49 lasting 0:41.
AA 524. From ORD to ROC. Departure JUN 14 21:30 transit time 1:22 ($30.0).
Layover at ROC starting JUN 14 23:52 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 15 0:07 transit time 0:01 ($1.42).
Arriving at destination JUN 15 0:08.

Backup: probability = .0779, U=467.32
Layover at SFO starting JUN 14 14:40 lasting 0:50.
UA 130. From SFO to ORD. Departure JUN 14 15:30 transit time 3:55 ($126.0).
Layover at ORD starting JUN 14 21:25 lasting 9:35.
AA 196. From ORD to ROC. Departure JUN 15 7:00 transit time 1:22 ($30.0).
Layover at ROC starting JUN 15 9:22 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 15 9:37 transit time 0:01 ($1.42).
Arriving at destination JUN 15 9:38.

Backup: probability = .2166, U=470.74
Layover at ORD starting JUN 14 20:49 lasting 10:11.
AA 196. From ORD to ROC. Departure JUN 15 7:00 transit time 1:22 ($30.0).
Layover at ROC starting JUN 15 9:22 lasting 0:15.
TAXI*. From ROC to UR. Departure JUN 15 9:37 transit time 0:01 ($1.42).
Arriving at destination JUN 15 9:38.

A chart of the tasks undertaken is presented in Figure A-1. Each task is identified by a number that describes the order in which the tasks were started. Task 0 is created when the problem statement is ingested. The first solution to this task initiates task 1, which later produced solutions numbered 2, 7, 8, and 9. The arrows symbolize the passing of a solution to a new level; in parentheses near each arrow is a pair: the utility of the solution generated, and the time (in seconds) since the previous solution to the task was generated.

It is instructive to make a table showing where each solution path "lost" its utility:

Itinerary	SP	INSTANTIATE	DOLLARS	FILLIN	PROB
1	-81	-14	-0	-11	-35
2	-80	-13	-0	-7	-36
3	-82	-9	-0	-10	-24
4	-82	-10	-0	-10	-79

We notice that the differences among the plans are largely "explained" by the analysis done in the PROB level. Itinerary 4 is poor because there is very poor backup at ORD--AA 524 is the last flight of the day. Itinerary 3, on the other hand, has exceptionally good backup (short waiting times before the next flight on the same route). Itineraries 1 and 2 have intermediate appeal. It will not always be the case that PROB processing is the most significant. In this case, because transportation to the airports is similar and because planes fly often

enough on the route to make scheduling easy, the INSTANTIATE and FILLIN contributions are almost identical.

Let us examine the processing of PEGASUS in somewhat more detail. Until the time task 6 is generated, PEGASUS is using its depth-first approach, and is simply allocating effort to the most recently initiated task. Thereafter, however, the processing skips around, governed by the upper bounds (and discounted by the cost of planning). Here is a sample (note that when a solution is generated, its true utility is cited, whereas control of planning is based on Up, the upper bound of the task, discounted by the expected costs of completing a solution that would emerge from the task):

```
Work on task 7 (INSTANTIATE) until Up drops below 679.4.
 This generates a new solution of U=684.0, and spawns task 10.
Work on task 7 (INSTANTIATE) until Up drops below 679.4.
 This generates a new solution of U=682.8, and spawns task 11.
Work on task 10 (DOLLARS) until Up drops below 679.0.
 This generates a new solution of U=684.0, and spawns task 12.
Work on task 12 (FILLIN) until Up drops below 679.0.
Work on task 1 (SP) until Up drops below 679.4.
Work on task 7 (INSTANTIATE) until Up drops below 678.3.
Work on task 11 (DOLLARS) until Up drops below 677.8.
 This generates a new solution of U=682.8, and spawns task 13.
Work on task 13 (FILLIN) until Up drops below 677.8.
Work on task 8 (INSTANTIATE) until Up drops below 677.0.
 This generates a new solution of U=681.5, and spawns task 14.
```

Each FILLIN process sets up a recursive call on the entire system in order to flesh out the ASSUMEd steps of the original plan. Here is a sample sequence from task 4:

```
Work on task a (TOP; HOME-SFO) until Up drops below -10E10.
 This generates a new solution (TRAIN*) of U=988, and spawns task b.
Work on task b (SP) until Up drops below 983.
 This generates a "utility dominance," and causes b to be suspended.
Work on task a (TOP) until Up drops below -10E10.
 This generates a new solution (BUS*) of U=988, and spawns task c.
Work on task c (SP) until Up drops below 979.
 This generates a "utility dominance," and causes c to be suspended.
Work on task a (TOP) until Up drops below -10E10.
 This generates a new solution (TAXI*) of U=979, and spawns task d.
 (Note: TAXI has no transportation graph, and proceeds to INSTANTIATE)
Work on task d (INSTANTIATE) until Up drops below 974.
 This generates a new solution ...

 ...TAXI from HOME to SFO continues through the processing.

Work on task i (TOP; ROC-UR) until Up drops below -10E10.
 This generates a new solution (TRAIN*) of U=999, and spawns task j.
Work on task j (SP) until Up drops below 994.
 This generates a "utility dominance," and causes j to be suspended.
Work on task i (TOP) until Up drops below -10E10.
 This generates a new solution (TAXI*) of U=998, and spawns task k.

 ...TAXI from ROC to UR continues through the processing.
```

FILLIN (task 4) returns a joint solution of U=668.

Figures A-2 through A-4 illustrate various aspects of the progress of the solution in the presence of cost of planning. They plot the lessening in utility as planning proceeds; the numbers indicate the point at which the task of that number was started. For example, Figure A-2 shows only task 1: it was spawned with a solution that had U=774; the task itself generated solutions that spawned tasks 2, 7, 8 and 9. Figure A-3 overlays all possible solution paths in an attempt to compare the various different possibilities; note that this is not the order in which PEGASUS actually pursued alternatives. The sequence 1-2-3-4-5-6, the depth-first solution, is shown darkened. Notice that the optimal solution (task 21) would have actually required slightly less computing than did the depth-first solution. Figure A-4 shows the actual planning sequence used by PEGASUS. The numbers near the vectors label the task whose execution resumed at this point; the numbers in a straight row indicate the points at which new tasks were started. Overlaid on the drawing is a line whose slope represents the cost of planning.

PEGASUS eventually stopped planning when the planning resources consumed since the emergence of the depth-first solution exceeded the cutoff (U=59 in this case). When PEGASUS suspended execution, the tasks were in the following states:

	Task number	Excess time	U	Up, with cost of planning
TOP				
	0	0	-593.2	-600.2
SP				
	1	4.0	656.1	652.3
INSTANTIATE				
	2	.6	666.8	663.6
	7	0	666.2	663.2
	8	.4	655.3	649.8
	9	1.7	658.5	652.8
DOLLARS				
	3	0	654.4	649.2
	10	0	652.0	646.7
	11	0	650.8	645.6
	14	0	649.5	644.3
	18	0	644.0	639.8
	24	0	647.2	644.3
	26	0	638.8	635.8
	27	0	665.8	662.9
FILLIN				
	4	.9	663.6	663.6
	12	2.2	663.0	663.0
	13	1.9	661.9	661.9

15	2.2	660.4	660.4
19	2.5	666.6	663.1
25	2.5	662.8	660.7
28	2.4	661.3	659.2

PROB

(no residuals here)

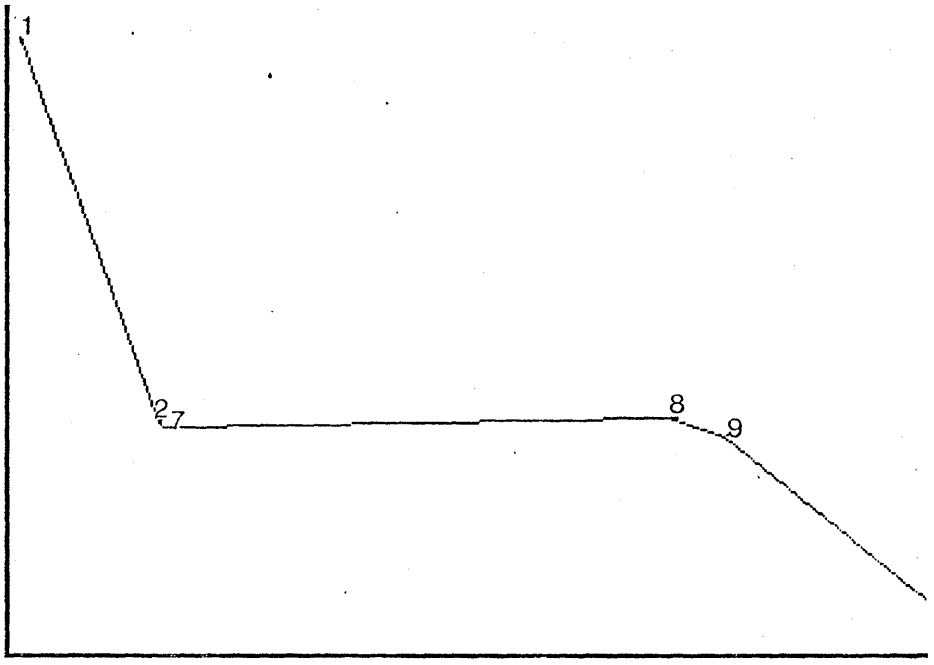

Figure A-2: History of solutions to task 1, as more planning resources are applied. Solutions started tasks 2, 7, 8 and 9. Notice that solutions 2 and 7 were generated with a heuristic method, as they have slightly lower utility than solution 8, which emerged later.

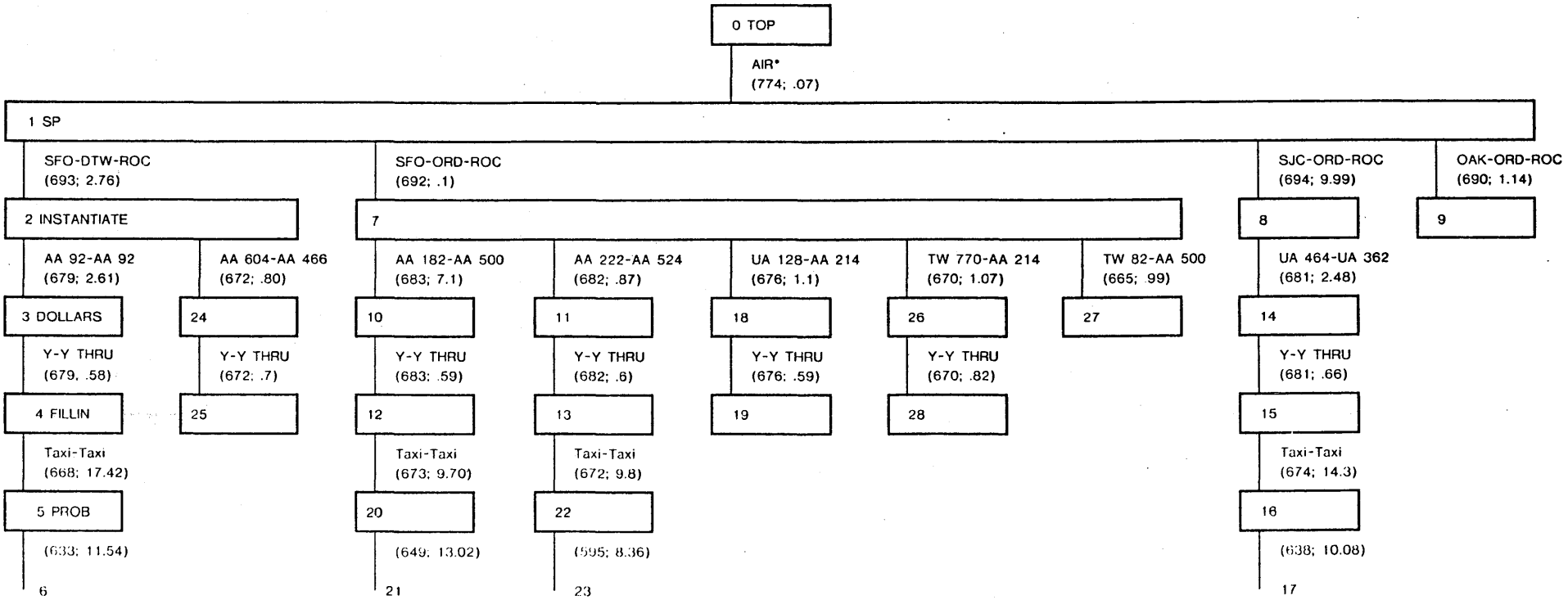

Figure A-1: Search tree undertaken by PEGASUS in the example. Each box corresponds to a single planning task; the numbers inside the boxes indicate the order in which the task was started. Solutions exit the bottom of a box, and are labeled with a pair (utility; processing time in seconds since the previous solution emerged from the task).

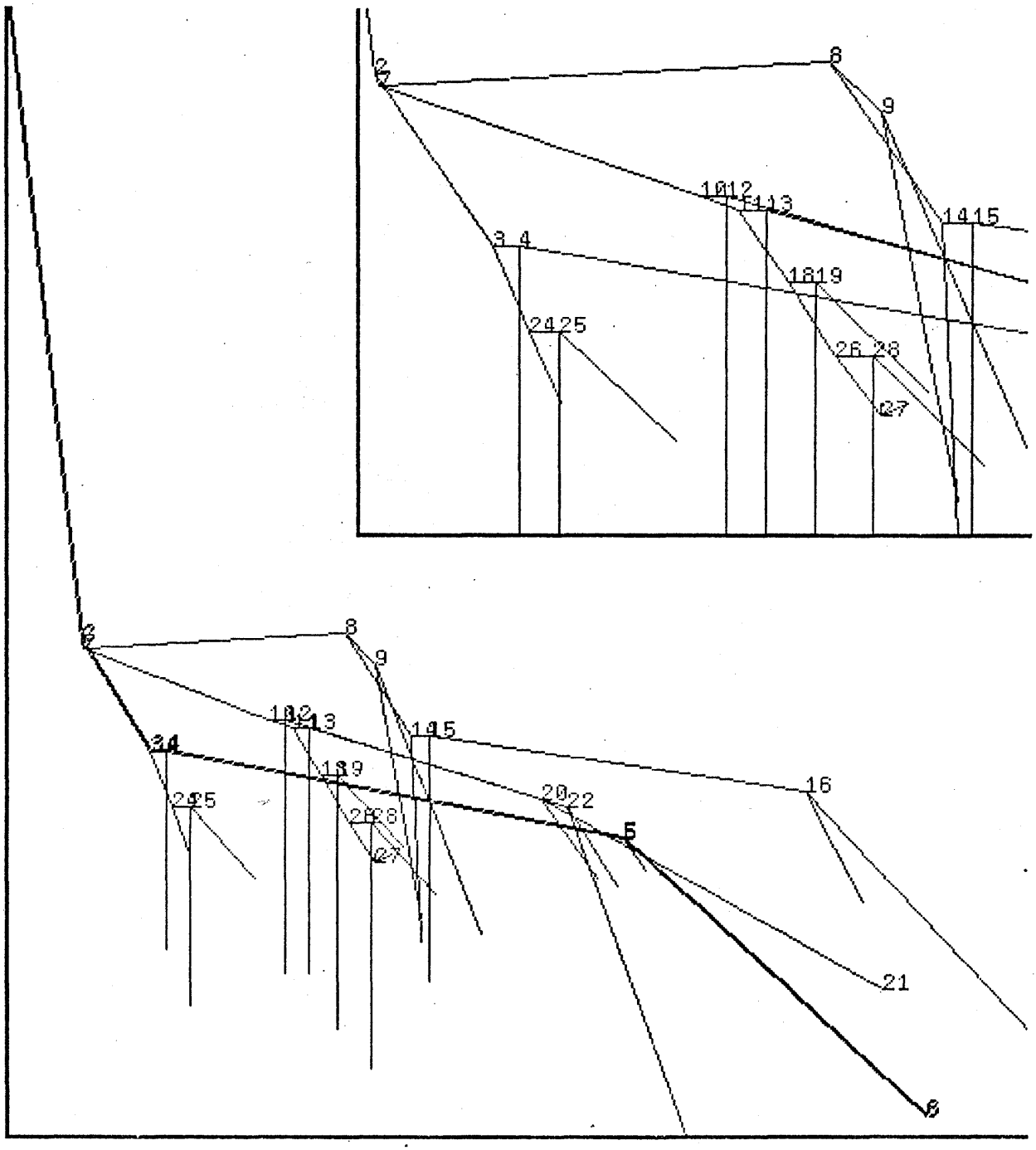

Figure A-3: Overlay of all possible solution trajectories. This plot attempts to compare the depth-first path actually taken first (shown darker) with other possibilities. Note that the depth-first path terminates at a point labeled 6; the optimum solution is labeled 21. An enlargement of the center area is also shown.

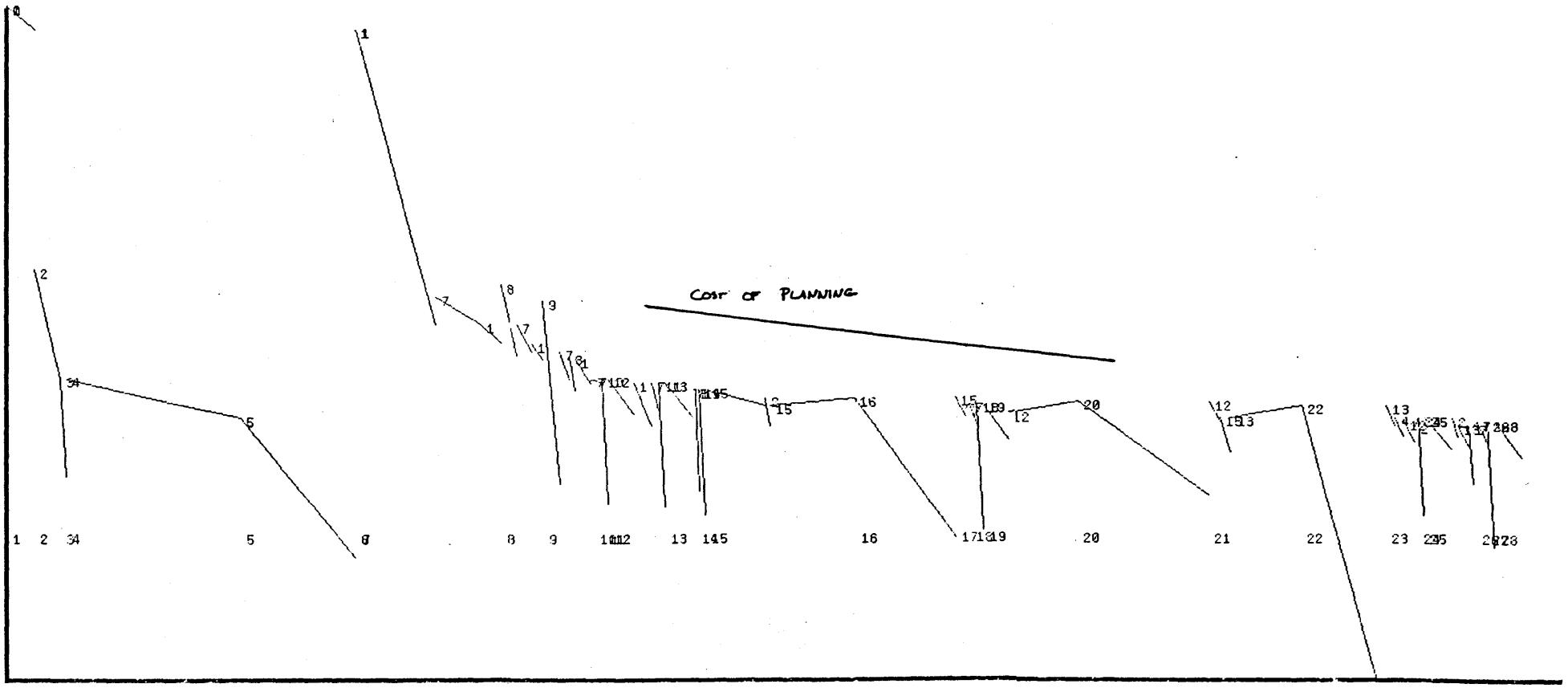

Figure A-4: Actual planning trajectory of PEGASUS. The sequence of solution utilities is shown an a function of time. Numbers near the vectors label the task resumed at that point. Numbers in a row indicate points at which new tasks were started.

XEROX

Xerox Corporation
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, California 94304

Printed in U.S.A.