Goal Achievement 13

Goal Achievement and Vision

Goals and plans are important for visual processing.

- Some skilled vision actually is like problem solving.
- Vision for information gathering can be part of a planned sequence of actions.
- Planning can be a useful and efficient way to guide many visual computations, even those that are not meant to imply "conscious" cognitive activity.

The artificial intelligence activity often called *planning* traditionally has dealt with "robots" (real or modeled) performing actions in the real world. Planning has several aspects.

- Avoid nasty "subgoal interactions" such as getting painted into a corner.
- Find the plan with optimal properties (least risk, least cost, maximized "goodness" of some variety).
- Derive a sequence of steps that will achieve the goal from the starting situation.
- Remember effective action sequences so that they may be applied in new situations.
- Apply planning techniques to giving advice, presumably by simulating the advisee's actions and making the next step from the point they left off.
- Recover from errors or changes in conditions that occur in the middle of a plan.

Traditional planning research has not concentrated on plans with information gathering steps, such as vision. The main interest in planning research has been the expensive and sometimes irrevocable nature of actions in the world. Our goal is to give a flavor of the issues that are pursued in much more detail in the planning

literature [Nilsson 1980; Tate 1977; Fahlman 1974; Fikes and Nilsson 1971; Fikes et al. 1972a; 1972b; Warren 1974; Sacerdoti 1974; 1977; Sussman 1975].

Planning concerns an active agent and its interaction with the world. This conception does not fit with the idea of vision as a passive activity. However, one claim of this book is that much of vision is a constructive, active, goal-oriented process, replete with uncertainty. Then a model of vision as a sequence of decisions punctuated by more or less costly information gathering steps becomes more compelling. Vision often is a sequential (recursive, cyclical) process of alternating information gathering and decision making. This paradigm is quite common in computer vision [Shirai 1975; Ballard 1978; Mackworth 1978; Ambler et al. 1975]. However, the formalization of the process in terms of minimizing cost or maximizing utility is not so common [Feldman and Sproull 1977; Ballard 1978; Garvey 1976]. This section examines the paradigms of planning, evaluating plans with costs and utilities, and how plans may be applied to vision processing.

13.1 SYMBOLIC PLANNING

In artificial intelligence, planning is usually a form of problem-solving activity involving a formal "simulation" of a physical world. (Planning, theorem proving, and state-space problem solving are all closely related.) There is an agent (the "robot") who can perform actions that transform the state of the simulated world. The robot planner is confronted with an initial world state and a set of goals to be achieved. Planning explores world states resulting from actions, and tries to find a sequence of actions that achieves the goals. The states can be arranged in a tree with initial state as the root, and branches resulting from applying different actions in a state. Planning is a search through this tree, resulting in a path or sequence of actions, from the root to a state in which the goals are achieved. Usually there is a metric over action sequences; the simplest is that there be as few actions as possible. More generally (Section 13.2), actions may be assigned some cost which the planner should minimize.

13.1.1 Representing the World

This section illustrates planning briefly with a classical example—block stacking. In one simple form there are three blocks initially stacked as shown on the left in Fig. 13.1, to be stacked as shown.

This task may be "formalized" [Bundy 1978] using only the symbolic objects Floor, A, B, and C. (A formalization suitable for a real automated planner must be much more careful about details than we shall be). Assume that only a single block can be picked up at a time. Necessary predicates are CLEAR(X) which is true if a block may be put directly on X and which must be true before X may be picked up, and ON(X, Y), which is true if X is resting directly on Y. Let us stipulate that the Floor is always CLEAR, but otherwise if ON(X, Y) is true, Y is not CLEAR. Then the initial situation in Fig. 13.1 is characterized by the following assertions.

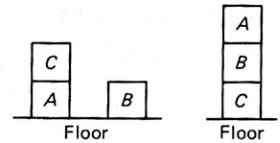

Fig. 13.1 A simple block stacking task.

INITIAL STATE: ON(C,A), ON(A, Floor), ON(B, Floor),
CLEAR(C), CLEAR(B), CLEAR(Floor)

The goal state is one in which the following two assertions are true.

GOAL ASSERTIONS: ON(A,B), ON(B,C)

With only these rules, the formalization of the block stacking world yields a very "loose" semantics. (The task easily translates to sorting integers with some restrictions on operations, or to the "seriation" task of arranging blocks horizontally in order of size, or a host of others.)

Actions transform the set of assertions describing the world. For problems of realistic scale, the representation of the tree of world states is a practical problem. The issue is one of maintaining several coexisting "hypothetical worlds" and reasoning about them. This is another version of the frame problem discussed in sec. 12.1.6. One way to solve this problem is to give each assertion an extra argument, naming the hypothetical world (usually called a situation [Nilsson 1980; McCarthy and Hayes 1969]) in which the assertion holds. Then actions map situations to situations as well as introducing and changing assertions.

An equivalent way to think about (and implement) multiple, dependent, hypothetical worlds is with a tree-structured *context-oriented data base.* This idea is a general one that is useful in many artificial intelligence applications, not just symbolic planning. Such data bases are included in many artificial intelligence languages and appear in other more traditional environments as well. A context-oriented data base *acts* like a tree of data bases; at any node of the tree is a set of assertions that makes up the data base. A new data base (context) may be spawned from any context (data base) in the tree. All assertions that are true in the spawning (ancestor) context are initially true in the spawned (descendant) context. However, new assertions added in any context or deleted from it do not affect its ancestor. Thus by going back to the ancestor, all data base changes performed in the descendent context disappear.

Implementing such a data base is an interesting exercise. Copying all assertions to each new context is possible, but very wasteful if only a few changes are made in each context. The following mechanism is much more efficient. The root or initial context has some set of assertions in it, and each descendant context is merely an *add list* of assertions to add to the data base and a *delete list* of assertions to delete. Then to see if an assertion is true in a context, do the following.

1. If the context is the root context, look up "as usual."
2. Otherwise, if the assertion is on the *add list* of this context, return *true.* If the assertion is on the *delete list* of this context, return *false.*

3. Otherwise, recursively apply this procedure to the ancestor of this context.

In a general programming environment, contexts have names, and there is the facility of executing procedures "in" particular contexts, moving around the context tree, and so forth. However, in what follows, only the ability to look up assertions in contexts is relevant.

13.1.2 Representing Actions

Represent an action as a triple.

ACTION ::= [PATTERN, PRECONDITIONS, POSTCONDITIONS].

Here the pattern gives the name of the action and names for the objects with which it deals—its "formal parameters." Preconditions and postconditions may use the formal variables of the pattern. In a sense, the preconditions and postconditions are the "body" of the action, with subroutine-like "variable bindings" taking place when the action is to be performed. The preconditions give the world states in which the action may be applied. Here the preconditions are assumed simply to be a list of assertions all of which must be true. The postconditions describe the world state that results from performing the action. The context-oriented data base of hypothetical worlds can be used to implement the postconditions.

POSTCONDITIONS ::= [ADD-LIST, DELETE-LIST].

An action is then performed as follows.

1. Bind the pattern variables to entities in the world, thus binding the associated variables in the preconditions and postconditions.
2. If the preconditions are met (the bound assertions exist in the data base), do the next step, else exit reporting failure.
3. Delete the assertions in the delete list, add those in the add list, and exit reporting success.

Here is the *Move* action for our block-stacking example.

Move Object X from Y to Z			
PATTERN	*PRECONDITIONS*	*DELETE-LIST*	*ADD-LIST*
Move(X,Y,Z)	CLEAR(X) CLEAR(Z) ON(X,Y)	ON(X,Y) CLEAR(Z)	ON(X,Z) CLEAR(Y)

Here X, Y, and Z are all variables bound to world entities. In the initial state of Fig. 13.1, Move(C,A,Floor) binds X to C, Y to A, Z to Floor, and the preconditions are satisfied; the action may proceed.

However, notice two things.

1. The action given above deletes the CLEAR(Floor) assertion that always should be true. One must fix this somehow; putting CLEAR(Floor) in the add-list does the job, but is a little inelegant.
2. What about an action like Move(C,A,C)? It meets the preconditions, but causes trouble when the add and delete lists are applied. One fix here is to keep in the data base ("world model") a set of assertions such as Different (A,B), Different(A,Floor), . . . , and to add assertions such as Different (X,Z) to the preconditions of Move.

Such housekeeping chores and details of axiomatization are inherent in applying basically syntactic, formal solution methods to problem solving. For now, let us assume that CLEAR(Floor) is never deleted, and that Move(X,Y,Z) is applied only if Z is different from X and Y.

13.1.3 Stacking Blocks

In the block-stacking example, the goal is two simultaneous assertions, ON(A,B) and ON(B,C). One solution method proceeds by repeatedly picking a goal to work on, finding an operator that moves closer to the goal, and applying it. In this case of only one action the question is how to apply it—what to move where. This is answered by looking at the postconditions of the action in the light of the goal. The reasoning might go like this: ON(B,C) can be made true if X is B and Z is C. That is possible in this state if Y is A; all preconditions are satisfied, and the goal ON(B,C) can be achieved with one action.

Part of the world state (or context) tree the planner must search is shown in Fig. 13.2, where states are shown diagrammatically instead of through sets of assertions. Notice the following things in Fig. 13.2.

1. Trying to achieve ON(B,C) first is a mistake (Branch 1).
2. Trying to achieve ON(A,B) first is also a mistake for less obvious reasons (Branch 2).
3. Branches 1 and 2 show "subgoal interaction." The goals as stated are not independent. Branch 3 must be generated somehow, either through backtracking or some intelligent way of coping with interaction. It will never be found by the single-minded approach of (1) and (2). However, if ON(C,Floor) were one of the goal assertions, Branch 3 could be found.

Clearly, representing world and actions is not the whole story in planning. Intelligent search of the context is also necessary. This search involves subgoal selection, action selection, and action argument selection. Bad choices anywhere can mean inefficient or looping action sequences, or the generation of impossible subgoals. "Intelligent" search implies a meta-level capability: the ability of a program to reason about its own plans. "Plan critics" are often a part of sophisticated planners; one of their main jobs is to isolate and rectify unwanted subgoal interaction [Sussman 1975].

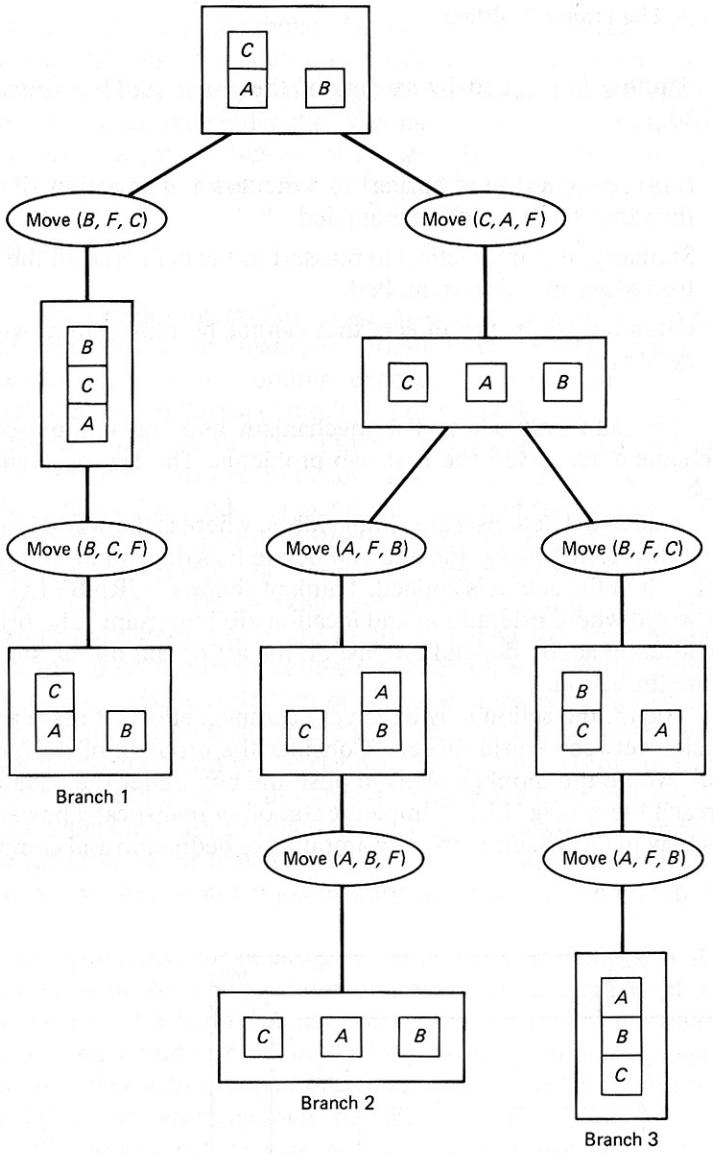

Fig. 13.2 A state tree generated in planning how to stack three blocks.

Intelligent choice of actions is the crux of planning, and is a major research issue. Several avenues have been and are being tried. Perhaps subgoals may be ordered by difficulty and achieved in that order. Perhaps planning should proceed at various levels of detail (like multiresolution image understanding), where the strategic skeleton of a plan is derived without details, then the details are filled in by applying the planner in more detail to the subgoals in the low-resolution plan.

13.1.4 The Frame Problem

All planning is plagued by aspects of the *frame problem* (introduced in Section 12.1.6).

1. It is impractical (and boring) to write down in an action all the things that stay the same when an action is applied.
2. Similarly, it is impractical to reassert in the data base all the things that remain true when an action is implied.
3. Often an action has effects that cannot be represented with simple add and delete lists.

The add and delete list mechanism and the context-oriented data base mechanism addressed the first two problems. The last problem is more troublesome.

Add and delete lists are simple ideas, whereas the world is a complex place. In many interesting cases, the add and delete lists depend on the current state of the world when the action is applied. Think of actions TURNBY(X) and MOVEBY(Z) in a world where orientation and location are important. The orientation and location after an action depend not just on the action but on the state of the world just before the action.

Again, the action may have very complex effects if there are complex dependencies between world objects. Consider the problem of the "monkey and bananas," where the monkey plans to push the box under the bananas and climb on it to reach them (Fig. 13.3). Implementation of realistically powerful add and delete lists may in fact require arbitrary amounts of deduction and computation.

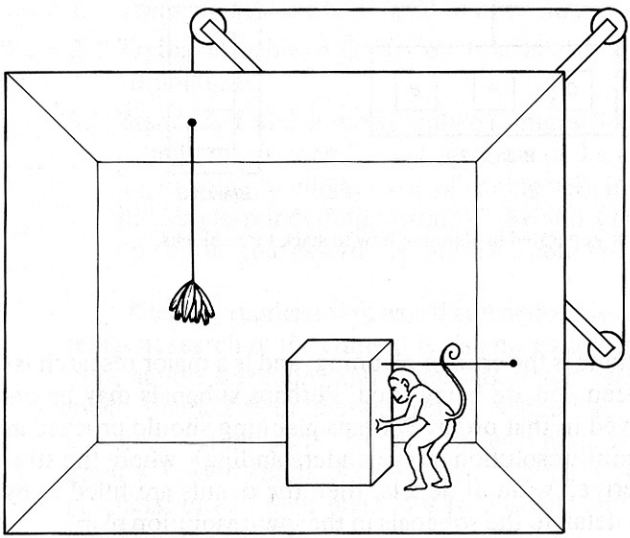

Fig. 13.3 Actions may have complex effects.

This quick précis of symbolic planning does not address many "classical" topics, such as learning or remembering useful plans. Also not discussed are: planning at varying levels of abstraction, plans with uncertain information, or plans with costs. The interested reader should consult the References for more information. The next section addresses plans with costs since they are particularly relevant to vision; some of the other issues appear in the Exercises.

13.2 PLANNING WITH COSTS

Decision making under uncertainty is an important topic in its own right, being of interest to policymakers and managers [Raiffa 1968]. Analytic techniques that can derive the strategy with the "optimal expected outcome" or "maximal expected utility" can be based on Bayesian models of probability.

In [Feldman and Sproull 1977] these techniques are explored in the context of action planning for real-world actions and vision. As an example of the techniques, they are used to model an extended version of the "monkey and bananas" problem of the last section, with multiple boxes but without the maddening pulley arrangement. In the extended problem, there are boxes of different weights which may or may not support the monkey, and he can apply tests (e.g., vision) at some cost to determine whether they are usable. Pushing weighted boxes costs some effort, and the gratification of eating the bananas is "worth" only some finite amount of effort. This extended set of considerations is more like everyday decision making in the number of factors that need balancing, in the uncertainty inherent in the universe, and in the richness of applicable tests. In fact, one might make the claim that human beings always "maximize their expected utility," and if one knew a person's utility functions, his behavior would become predictable. The more intuitive claim that humans beings plan only as far as "sufficient expected utility" can be cast as a maximization operation with nonzero "cost of planning."

The sequential decision-making model of planning with the goal of maximizing the goodness of the expected outcome was used in a travel planner [Sproull 1977]. Knowledge of schedules and costs of various modes of transportation and the attendant risks could be combined with personal prejudices and preferences to produce an itinerary with the maximum expected utility. If unexpected situations (canceled flights, say) arose *en route,* replanning could be initiated; this incremental plan ramification is a natural extension of sequential decision making.

This section is concerned with measuring the expected performance of plans using a single number. Although one might expect one number to be inadequate, the central theorem of decision theory [DeGroot 1970] shows essentially that one number is enough. Using a numerical measure of goodness allows comparisons between normally incomparable concepts to be made easily. Quite frequently numerical scores are directly relevant to the issues at stake in planning, so they are not obnoxiously reductionistic. Decision theory can also help in the process of applying a plan—the basic plan may be simple, but its application to the world may be complex, in terms of when to declare a result established or an action unsuccessful. The decision-theoretic approach has been used in several artificial intelligence and