A key idea in defining heuristics is decomposition: dividing a problem into parts, solving each part independently, and then combining the parts. The subgoal independence assumption is that the cost of solving a conjunction of subgoals is approximated by the sum of the costs of solving each subgoal independently. The subgoal independence assumption can be optimistic or pessimistic. It is optimistic when there are negative interactions between the subplans for each subgoal—for example, when an action in one subplan deletes a goal achieved by another subplan. It is pessimistic, and therefore inadmissible, when subplans contain redundant actions—for instance, two actions that could be replaced by a single action in the merged plan.

Suppose the goal is a set of fluents \( G \), which we divide into disjoint subsets \( G_1, \ldots, G_m \). We then find plans \( P_1, \ldots, P_m \) that solve the respective subgoals. What is an estimate of the cost of the plan for achieving all of \( G \)? We can think of each \( \text{Cost}(P_i) \) as a heuristic estimate, and we know that if we combine estimates by taking their maximum value, we always get an admissible heuristic. So \( \max_i \text{Cost}(P_i) \) is admissible, and sometimes it is exactly correct: it could be that \( P_1 \) serendipitously achieves all the \( G_i \). But in most cases, in practice the estimate is too low. Could we sum the costs instead? For many problems that is a reasonable estimate, but it is not admissible. The best case is when we can determine that \( G_i \) and \( G_j \) are independent. If the effects of \( l \) leave all the preconditions and goals of \( P_i \) unchanged, then the estimate \( \text{Cost}(P_i) + \text{Cost}(P_j) \) is admissible, and more accurate than the max estimate.

We show in Section 10.3.1 that planning graphs can help provide better heuristic estimates.

It is clear that there is great potential for cutting down the search space by forming abstractions. The trick is choosing the right abstractions and using them in a way that makes the total cost—defining an abstraction, doing an abstract search, and mapping the abstraction back to the original problem—less than the cost of solving the original problem. The tech-
piques of pattern databases from Section 3.6.3 can be useful, because the cost of creating the pattern database can be amortized over multiple problem instances.

An example of a system that makes use of effective heuristics is FF, or FASTFORWARD (Hoffmann, 2005), a forward state-space searcher that uses the ignore-delete-lists heuristic, estimating the heuristic with the help of a planning graph (see Section 10.3). FF then uses hill-climbing search (modified to keep track of the plan) with the heuristic to find a solution. When it hits a plateau or local maximum—when no action leads to a state with better heuristic score—then FF uses iterative deepening search until it finds a state that is better, or it gives up and restarts hill-climbing.

10.3 PLANNING GRAPHS

All of the heuristics we have suggested can suffer from inaccuracies. This section shows how a special data structure called a planning graph can be used to give better heuristic estimates. These heuristics can be applied to any of the search techniques we have seen so far. Alternatively, we can search for a solution over the space formed by the planning graph, using an algorithm called GRAPHPLAN.

A planning problem asks if we can reach a goal state from the initial state. Suppose we are given a tree of all possible actions from the initial state to successor states, and their successors, and so on. If we indexed this tree appropriately, we could answer the planning question "can we reach state $G$ from state $S_0$" immediately, just by looking it up. Of course, the tree is of exponential size, so this approach is impractical. A planning graph is polynomial-size approximation to this tree that can be constructed quickly. The planning graph can't answer definitively whether $G$ is reachable from $S_0$, but it can estimate how many steps it takes to reach $G$. The estimate is always correct when it reports the goal is not reachable, and it never overestimates the number of steps, so it is an admissible heuristic.

A planning graph is a directed graph organized into levels: first a level $S_0$ for the initial state, consisting of nodes representing each fluent that holds in $S_0$; then a level $A_0$ consisting of nodes for each ground action that might be applicable in $S_0$; then alternating levels $S$, followed by $A$, until we reach a termination condition (to be discussed later).

Roughly speaking, $S_i$ contains all the literals that could hold at time $i$, depending on the actions executed at preceding time steps. If it is possible that either $P$ or $\neg P$ could hold, then both will be represented in $S_i$. Also roughly speaking, $A_i$ contains all the actions that could have their preconditions satisfied at time $i$. We say "roughly speaking" because the planning graph records only a restricted subset of the possible negative interactions among actions; therefore, a literal might show up at level $S_j$ when actually it could not be true until a later level, if at all. (A literal will never show up too late.) Despite the possible error, the level $j$ at which a literal first appears is a good estimate of how difficult it is to achieve the literal from the initial state.

Planning graphs work only for propositional planning problems—ones with no variables. As we mentioned on page 358, it is straightforward to propositionalize a set of ac-