CHAPTER 9 Heuristics in Planning

9.1 Introduction

The introduction to this part described a conceptual view of planning as an abstract search problem in which the steps include refinement, branching, and pruning. The introduction pointed out that in order to implement the search deterministically, a node-selection function Select(C) is needed to choose which node u to visit next from a set of candidate nodes C. Often, although not always, the deterministic search is done in a depth-first manner; Figure 9.1 gives an abstract example. This chapter focuses on heuristic techniques for node selection.

The outline of the chapter is as follows. We first discuss the notion of relaxation as a general principle for designing heuristics (Section 9.2). We then detail several heuristics for state-space planning and relate them to the planning-graph techniques (Section 9.3). We finally present heuristics for flaw selection and resolver selection in plan-space planning (Section 9.4). The chapter ends with a discussion section and exercises.

9.2 Design Principle for Heuristics: Relaxation

A *node-selection heuristic* is any way of ranking a set of nodes in order of their relative desirability. We will model this heuristic as a function h that can be used to compute a numeric evaluation $h(u)$ for each candidate node $u \in C$, with a convention that says that the preferred node $u \in C$ is the one that has the smallest $h(u)$, i.e., Select$(C) = \text{argmin}\{h(u) \mid u \in C\}$.

Node-selection heuristics are used for resolving nondeterministic choices. If there is a known deterministic technique for choosing at each choice point the right node for solving the problem, one would just integrate that way into a deterministic algorithm and we would not call it a heuristic. Hence, a node-selection heuristic is usually not foolproof, in the sense that the node recommended by the heuristic is not always guaranteed to be the best choice: this node may not always lead to the best

```
Depth-first-search($u$)
    if Terminal($u$) then return($u$)
    $u \leftarrow$ Refine($u$)          ;; refinement step
    $B \leftarrow$ Branch($u$)          ;; branching step
    $C \leftarrow$ Prune($B$)           ;; pruning step
    while $C \neq \emptyset$ do
        $v \leftarrow$ Select($C$)      ;; node-selection step
        $C \leftarrow C - \{v\}$
        $\pi \leftarrow$ Depth-first-search($v$)
        if $\pi \neq$ failure then return($\pi$)
    return(failure)
end
```

Figure 9.1 Abstract version of a depth-first search procedure.

solution or even to a solution at all. However, we would like the heuristic to be as *informative* as possible, i.e., to be as close as possible to an *oracle* that knows the right choice. The smaller the number of incorrect choices a node-selection heuristic h makes, the more informative it is. We also want h to be easily computable, so that there will be a clear benefit in computing it and using it for making choices rather than, say, making random choices or doing a brute-force search that tries all alternatives. There is usually a trade-off between how informative h is and how easy it is to compute.

Node-selection heuristics are often based on the following *relaxation principle*: in order to assess how desirable a node u is, one considers a simpler problem that is obtained from the original one by making simplifying assumptions and by relaxing constraints. One estimates how desirable u is by using u to solve the simpler relaxed problem and using that solution as an estimate of the solution one would get if one used u to solve the original problem. The closer the relaxed problem is to the original one, the more informative the estimate will be. On the other hand, the more simplified the relaxed problem is, the easier it will be to compute the heuristic. Unfortunately, most of the time it is not easy to find the best trade-off between relaxation and informativeness of the heuristic: to do that requires a knowledge of the structure of the problem.

We close this section with another property of node-selection heuristics that is desirable if one seeks a solution that is *optimal* with respect to some cost criterion. In this case, a candidate node u is preferred if it leads to a lower-cost solution, and $h(u)$ is an estimate of the minimum possible cost $h^*(u)$ of any solution reachable from u (with $h^*(u) = \infty$ if no solution is reachable from u). In this case, a heuristic function h is *admissible* if it is a lower bound estimate of the cost of a minimal solution reachable from u, i.e., if $h(u) \leq h^*(u)$. Heuristic search algorithms, such as the iterative-deepening scheme, are usually able to guarantee an optimal solution when guided with an admissible node-selection heuristic. But quite often, one may have to trade admissibility for informativeness.

9.3 Heuristics for State-Space Planning

This section presents several node-selection heuristics for state-space planning. We start with a simple heuristic function and present its use in guiding a forward-search backtracking algorithm, then a backward-search procedure. We then develop several more elaborate heuristics and discuss their properties. Finally, the relationship between these heuristics and the planning-graph techniques is discussed.

9.3.1 State Reachability Relaxation

Let us consider a planning domain in the set-theoretic representation (Section 2.2), and let us focus on a forward-search procedure in the state space. At some point the candidate nodes are the successor states of the current state s, for the actions applicable to s. In order to choose the most preferable one, we need to assess how close each action may bring us to the goal.

For any action a applicable to a state s, the next state is given by the transition function: $\gamma(s, a) = (s - \text{effects}^-(a)) \cup \text{effects}^+(a)$. Although this is a very simple set operation for each elementary transition, $\gamma(s, a)$ does not allow an easy prediction of distances to a goal: even if $\text{effects}^+(a)$ brings us apparently closer by adding some of the goal propositions to s, we may still be getting off track with an action a because of its $\text{effects}^-(a)$.

A very intuitive relaxation idea for the classical planning search is to neglect $\text{effects}^-(a)$. This simplified $\gamma(s, a)$ involves only a *monotonic* increase in the number of propositions from s to $\gamma(s, a)$. Hence, it is easier to compute distances to goals with such a simplified γ. The following heuristic functions are based on this relaxation idea.

Definition 9.1 Let $s \in S$ be a state, p a proposition, and g a set of propositions. The *minimum distance* from s to p, denoted by $\Delta^*(s, p)$, is the minimum number of actions required to reach from s a state containing p. The *minimum distance* from s to g, $\Delta^*(s, g)$, is the minimum number of actions required to reach from s a state containing *all* propositions in g. ■

Let $\Delta(s, p)$ be an estimate of $\Delta^*(s, p)$, and $\Delta(s, g)$ be an estimate of $\Delta^*(s, g)$. In the remaining sections, we will be defining several instances of the estimate Δ, denoted by $\Delta_0, \Delta_1, \ldots, \Delta_k$, and Δ_g, as well as heuristic functions obtained from these estimates, denoted by $h_0, \ldots, h_k$. These estimates will rely on different relaxations of the planning problem. Let us focus on the estimate Δ_0, which ignores effects$^-(a)$ for all a and which approximates the distance to g as the sum of the distances to the propositions in g. This estimate is based on the intuitive relaxation discussed previously. Δ_0 is given by the following equations.

$$\begin{array}{ll}
\Delta_0(s,p) = 0 & \text{if } p \in s \\
\Delta_0(s,p) = \infty & \text{if } \forall a \in A, p \notin \text{effects}^+(a) \\
\Delta_0(s,g) = 0 & \text{if } g \subseteq s \\
\text{otherwise:} & \\
\Delta_0(s,p) = \min_a\{1 + \Delta_0(s, \text{precond}(a)) \mid p \in \text{effects}^+(a)\} & \\
\Delta_0(s,g) = \sum_{p \in g} \Delta_0(s,p) &
\end{array} \tag{9.1}$$

These equations give an estimate of the distance to g in the relaxed problem, and this value is an estimate of the distance to g in the unrelaxed problem. The first two equations are simple termination conditions. The third one says that p is not reachable from s if the domain contains no action that produces p. The fourth equation defines $\Delta_0(s,p)$ recursively with respect to $\Delta_0(s,g)$. It states that if a is applicable to s' and produces p, then it takes just one step to reach p from s', precond$(a) \subseteq s'$, and Δ_0 is the minimum distance to reach any such s'. The last equation states that the distance to g is the sum of the distances to its propositions. It follows the relaxation intuition that each proposition in g can be reached *independently* of the others. This follows the independence relaxation. Note that Formulas 9.1 do ignore the effects$^-$ of actions.

We can now define a heuristic function $h_0(s)$ that gives an estimate of the distance from a node s to a node that satisfies the goal g of a planning problem $\mathcal{P} = (\Sigma, s_0, g)$:

$$h_0(s) = \Delta_0(s,g)$$

Figure 9.2 shows an algorithm that computes $h_0(s)$ by using Formulas 9.1 to compute $\Delta_0(s,p)$ for every proposition p in the domain. It is similar to a minimum-distance graph-searching algorithm: starting from the state s, it proceeds through each action whose preconditions have been reached, until a fixed point is reached. The algorithm is polynomial in the number of propositions and actions.

If we are given a planning problem where each action a has a cost $cost(a)$, then we will define the distance to a proposition p or to a goal g to be the cost of achieving p or g, and we will try to estimate such a distance. This can be done easily when the cost of a sequence of actions is the sum of the individual costs. In that case, in Formulas 9.1 and in the Delta algorithm, we just need to replace the digit "1" with $cost(a)$. The same is true for the other node-selection heuristics that we will define later in Formulas 9.3 through 9.5.

A state-space algorithm that is heuristically guided with h_0 is given in Figure 9.3. This algorithm does a recursive backtracking search. Among the children of each node s, it first goes to the child whose Δ_0 value is best. If backtracking is required, then the algorithm tries the second-best child, then the third-best child,

```
Delta(s)
    for each $p$ do: if $p \in s$ then $\Delta_0(s,p) \leftarrow 0$, else $\Delta_0(s,p) \leftarrow \infty$
    $U \leftarrow \{s\}$
    iterate
        for each $a$ such that $\exists u \in U, \mathrm{precond}(a) \subseteq u$ do
            $U \leftarrow \{u\} \cup \mathrm{effects}^+(a)$
            for each $p \in \mathrm{effects}^+(a)$ do
                $\Delta_0(s,p) \leftarrow \min\{\Delta_0(s,p), 1 + \sum_{q \in \mathrm{precond}(a)} \Delta_0(s,q)\}$
    until no change occurs in the above updates
end
```

Figure 9.2 Algorithm for computing heuristic estimates of distances from a state s to propositions.

```
Heuristic-forward-search($\pi, s, g, A$)
    if $s$ satisfies $g$ then return $\pi$
    $options \leftarrow \{a \in A \mid a$ applicable to $s\}$
    for each $a \in options$ do Delta($\gamma(s,a)$)
    while $options \neq \emptyset$ do
        $a \leftarrow \mathrm{argmin}\{\Delta_0(\gamma(s,a), g) \mid a \in options\}$
        $options \leftarrow options - \{a\}$
        $\pi' \leftarrow$ Heuristic-forward-search($\pi.a, \gamma(s,a), g, A$)
        if $\pi' \neq$ failure then return($\pi'$)
    return(failure)
end
```

Figure 9.3 Heuristically guided forward-search algorithm.

and so forth. The argument π is the current partial plan; thus the initial call is Heuristic-forward-search($\langle\rangle, s_0, g, A$), where $\langle\rangle$ is the empty plan.

One may also use h_0 with a greedy search algorithm or with a best-first search that keeps all candidate nodes and chooses at each step the best one. Variants that are less memory intensive than a best-first search can be a mixture between greedy and best-first (keep a greedy strategy unless its choice is too far from the best candidate node) or an iterative-deepening algorithm.

The generalization of Delta and the above procedure to the classical planning representation with predicate symbols in planning operators is quite easy. In such a representation, a state s is a set of ground atoms, and a goal formula g is a set of ground literals. Along with the relaxation principle, we estimate the distances to the goal by approximating g to g^+, its set of positive atoms. We do not need to compute the distances $\Delta_0(s, p)$ for all ground instances of predicate symbols in the domain but only for those needed to get the sum: $\sum_{p \in g^+} \Delta_0(s, p)$.

9.3.2 Heuristically Guided Backward Search

Although the Delta algorithm is very efficient, it will be heavily used in the Heuristic-forward-search algorithm because it has to be run for every applicable action of every visited state. By doing a backward search instead of a forward search, we can use the node-selection heuristic more economically in the number of calls to Delta.

A backward search would proceed by regressing the current goal g through an action a that is relevant for g until the initial state s_0 is reached.[1] According to the definition of γ^{-1}, the heuristic distances are with respect to various current goals and a single initial state s_0. We may run Delta initially and compute, *just once*, $\Delta_0(s_0, p)$ for every p. In backward search, the current node is a current goal g. The heuristic estimate of g is given from the precomputed values by $\Delta_0(s_0, g) = \sum_{p \in g} \Delta_0(s_0, p)$.

The guided backward-search procedure is specified in Figure 9.4. It does not include any call to procedure Delta, but it relies on an initial run of Delta(s_0). After this run, the procedure is called initially with Backward-search($\langle\rangle, s_0, g, A$).

We close this section with two remarks. First, a goal regression does not involve effects$^-(a)$ because $\gamma^{-1}(g, a) = (g - \text{effects}^+(a)) \cup \text{precond}(a)$. In a sense, the goal regression achieved by not taking into account effects$^-$ is similar to the relaxation used here in Delta. However, recall that the definition of a *relevant* action

```
Backward-search($\pi, s_0, g, A$)
    if $s_0$ satisfies $g$ then return($\pi$)
    $options \leftarrow \{a \in A \mid a$ relevant for $g\}$
    while $options \neq \emptyset$ do
        $a \leftarrow \text{argmin}\{\Delta_0(s_0, \gamma^{-1}(g, a)) \mid a \in options\}$
        $options \leftarrow options - \{a\}$
        $\pi' \leftarrow$ Backward-search($a.\pi, s_0, \gamma^{-1}(g, a), A$)
        if $\pi' \neq$ failure then return($\pi'$)
    return failure
end
```

Figure 9.4 Heuristically guided backward-search algorithm.

1. Recall that the regression of a goal g for an action a *relevant* to g, as defined in Section 2.2.2, is $\gamma^{-1}(g, a) = (g - \text{effects}^+(a)) \cup \text{precond}(a)$.

(and consequently the set *options* in the above procedure) does involve $\text{effects}^-(a)$ because we defined a to be relevant for g iff $g \cap \text{effects}^+(a) \neq \emptyset$ and $g \cap \text{effects}^-(a) = \emptyset$.

Second, in the classical representation, goals are ground literals, including negated literals. The generalization of this approach to the classical representation requires neglecting the atoms in g^- but also taking into account unground atoms that may appear in γ^{-1} because of $\text{precond}(a)$. In that case, $\Delta_0(s_0, p)$ can be defined as the minimal distance to any ground instance of p. The initial run of algorithm Delta is pursued until $\Delta_0(s_0, g)$ can be computed, for the initial goal g. This run is resumed incrementally whenever a new set of atoms g' requires the value of not yet computed distances $\Delta_0(s_0, p)$.

9.3.3 Admissible State-Space Heuristics

The heuristic h_0 is not admissible. In other words, the estimate $\Delta_0(s, g)$ is not a lower bound on the true minimal distance $\Delta^*(s, g)$. This is easily verified. For example, assume a problem where there is an action a such that $\text{precond}(a) \subseteq s_0$, $\text{effects}^+(a) = g$, and $s_0 \cap g = \emptyset$. In this problem the distance to the goal is 1, but $\Delta_0(s_0, g) = \sum_{p \in g} \Delta_0(s_0, p) = |g|$.

It can be desirable to use admissible heuristic functions for two reasons. We may be interested in getting the shortest plan, or there may be explicit costs associated with actions and we are using a best-first algorithm and requiring an optimal or near optimal plan. Furthermore, admissible heuristics permit a *safe pruning*: if Y is the length (or the cost) of a known plan and if $h(u) > Y$, h being admissible, then we are sure that no solution of length (or cost) smaller than Y can be obtained from the node u. Pruning u is safe in the sense that it does not affect the completeness of the algorithm.

It is very simple to obtain an admissible heuristic by modifying the estimate Δ_0 given earlier. Instead of estimating the distance to a set of propositions g to be the *sum* of the distances to the elements of g, we estimate it to be the *maximum* distance to its propositions. This leads to an estimate, denoted Δ_1, which is defined by changing the last equation of Formulas 9.1 to the following:

$$\Delta_1(s, g) = \max\{\Delta_1(s, p) \mid p \in g\} \tag{9.2}$$

The heuristic function associated with this estimate is $h_1(s) = \Delta_1(s, g)$.

The Delta algorithm is easily changed for computing h_1: the update step for each $p \in \text{effects}^+(a)$ is now:

$$\Delta_1(s, p) \leftarrow \min\{\Delta_1(s, p), 1 + \max\{\Delta_1(s, q) \mid q \in \text{precond}(a)\}\}$$

The heuristic h_1 can be used, like h_0 was used, to guide a state-space search algorithm in a forward or backward manner. In addition, it allows a safe pruning with respect to an a priori given upper bound. However, experience shows that h_1 is not as informative as h_0. It usually leads to a much less focused search.

The idea behind h_1 can be extended in order to produce a more informative heuristic that is still admissible. Instead of considering that the distance to a set g is the maximum distance to a proposition $p \in g$, we estimate it to be the maximum distance to a *pair of propositions* $\{p, q\} \in g$. This new estimate Δ_2 is defined according to the three following recursive equations (in addition to the termination cases that remain as in Formulas 9.1):

$$\begin{aligned}
\Delta_2(s,p) \quad &= \min_a\{1 + \Delta_2(s, \text{precond}(a)) \mid p \in \text{effects}^+(a)\} \\
\Delta_2(s,\{p,q\}) \quad &= \min\{ \\
&\quad \min_a\{1 + \Delta_2(s, \text{precond}(a)) \mid \{p,q\} \subseteq \text{effects}^+(a)\} \\
&\quad \min_a\{1 + \Delta_2(s, \{q\} \cup \text{precond}(a)) \mid p \in \text{effects}^+(a)\} \\
&\quad \min_a\{1 + \Delta_2(s, \{p\} \cup \text{precond}(a)) \mid q \in \text{effects}^+(a)\}\} \\
\Delta_2(s,g) \quad &= \max_{p,q}\{\Delta_2(s,\{p,q\}) \mid \{p,q\} \subseteq g\}
\end{aligned} \tag{9.3}$$

The first equation gives, as before, the value of Δ_2 for a single proposition p. The second equation states that a pair $\{p, q\}$ can be reached in one step from a state s' to which is applicable an action a that produces both p and q, when precond$(a) \subseteq s'$; or it can be reached in one step from a state s'' by an action a that produces either p or q when s'' contains the other proposition q or p and the preconditions of a. The last equation states the relaxation condition: it estimates the distance from s to g as the longest distance to the pairs in g. This new estimate Δ_2 leads to the heuristic function $h_2(s) = \Delta_2(s, g)$.

The heuristic h_2 is admissible (see Exercise 9.3) and is significantly more informative than h_1. It takes longer to compute. However, the backward-search procedure devised with h_0, which relies on precomputed distances from s_0, is applicable to h_2. In this scheme, we modify the Delta algorithm to compute distances from s_0 to pairs of propositions, and we either run it just once to reach every pair or run it incrementally when the backward search reaches a pair whose distance from s_0 is not already precomputed (see Exercise 9.4).

It is interesting to note that the relaxation idea in the heuristic h_2 can be generalized. Instead of estimating the distance from s to g through the pairs in g, we may rely on the triples of g or on its k-tuples for any k.

Recall that $\Delta^*(s, g)$ is the true minimal distance from a state s to a goal g. Δ^* can be computed (albeit at great computational cost) according to the following equations:

$$\Delta^*(s,g) = \begin{cases} 0 & \text{if } g \subseteq s, \\ \infty & \text{if } \forall a \in A, a \text{ is not relevant for } g, \text{ and} \\ \min_a\{1 + \Delta^*(s, \gamma^{-1}(g,a)) \mid a \text{ relevant for } g\} \\ & \text{otherwise.} \end{cases} \tag{9.4}$$

From Δ^*, let us define the following family Δ_k, for $k \geq 1$, of heuristic estimates:

$$\Delta_k(s,g) = \begin{cases} 0 & \text{if } g \subseteq s, \\ \infty & \text{if } \forall a \in A, a \text{ is not relevant for } g, \\ \min_a\{1 + \Delta^*(s, \gamma^{-1}(g,a)) \mid a \text{ relevant for } g\} & \\ \quad \text{if } |g| \leq k, & \\ \max_{g'}\{\Delta_k(s,g') \mid g' \subseteq g \text{ and } |g'| = k\} & \\ \quad \text{otherwise.} & \end{cases} \tag{9.5}$$

The third equation in Formula 9.5 says that when g has at most k propositions, we take the exact value Δ^*; otherwise, the relaxation approximates the distance Δ_k to g as the maximum distance to its k-tuple. The corresponding heuristic function is $h_k(s) = \Delta_k(s,g)$.

The heuristic h_k is very informative, but it also incurs a high computational cost, having a complexity of at least $O(n^k)$ for a problem with n propositions. When $k = \max_a\{|g|, |\text{precond}(a)|\}$, with g being the goal of the problem, getting the heuristic values requires solving the planning problem first! For the family of admissible heuristics $h_1, h_2, \ldots, h_k$, the quality of the heuristic estimate and its computational cost increase with larger values of k. The natural question is how to find the best trade-off. Most published results and experimental data are for low values of k, typically $k \leq 2$; beyond that the overhead seems to be too high.

9.3.4 Graphplan as a Heuristic Search Planner

The Backward-search procedure of Section 9.3.2 relies on an initial run of the algorithm Delta that computes, in a forward manner, the distances from state s_0; Backward-search then uses these distances in a backward-search manner to find a solution. The Graphplan procedure (Chapter 6) follows a similar approach. Indeed, Graphplan also runs a forward expansion of polynomial complexity, namely the Expand algorithm (Figure 6.5), before starting a backward search with the procedure GP- Search (Figure 6.8) that relies on the result of the first phase.

The two algorithms Delta and Expand both perform a reachability analysis. The main difference is that Expand builds a data structure, the planning graph, which provides more information attached to propositions than just the distances to s_0.

Indeed, these distance estimates are directly explicit in the planning graph. Let $\Delta^G(s_0, p)$ be the level i of the first layer P_i such that $p \in P_i$. Then $\Delta^G(s_0, p)$ is an estimate of $\Delta^*(s_0, p)$ that is very similar to previously defined estimates. Their differences are clarified in the next step.

The planning graph provides not only these distance estimates from s_0 to every reachable proposition but also the mutual exclusion relations μP_i. Recall that the GP- Search procedure of Graphplan selects a set of propositions g in a layer only if no pair of elements in g is mutex. Furthermore, we showed that if a pair is not mutex in a layer, then it remains nonmutex in the following layers: if p and q are in P_{i-1} and $(p,q) \notin \mu P_{i-1}$, then $(p,q) \notin \mu P_i$. Consequently, we can consider that the planning-graph structure approximates the distance $\Delta^*(s_0, g)$ with an estimate $\Delta^G(s_0, g)$ that is the level of the first layer of the graph such that $g \subseteq P_i$ and no pair of g is in μP_i. This is very close to Δ_2, as defined in Formulas 9.3. The difference between $\Delta^G(s_0, p)$ and Δ_0 is that Δ_2 counts each action individually with a cost of 1, because it is a state-space distance, whereas $\Delta^G(s_0, g)$ counts all the independent actions of a layer with a total cost of 1.

It is simple to modify Formulas 9.3 to get exactly the distance estimates $\Delta^G(s_0, g)$ defined from the planning-graph structure. We modify the second equation to state that a pair (p, q) is reachable in *one* step if either there is a single action achieving both propositions or there are two *independent* actions achieving each one. This modification gives a more informed node-selection heuristic in a domain where independence between actions is important.

Consequently, Graphplan can be viewed as a heuristic search planner that first computes the distance estimates $\Delta^G(s_0, g)$ in a forward propagation manner and then searches backward from the goal using an iterative-deepening approach (i.e., the planning graph is deepened if needed) augmented with a learning mechanism corresponding to the nogood hash table of previously failed sets of propositions.

9.4 Heuristics for Plan-Space Planning

Plan-space planning (e.g., the PSP procedure described in Chapter 5) searches a search space whose nodes are partial plans and finds solutions by repairing flaws in these plans. This search space can be viewed as a search in an AND/OR tree (see Figure 9.5). The flaws in a plan correspond to an AND-branch because all of them must eventually be resolved in order to find a solution. For each flaw, the possible resolvers correspond to an OR-branch because only one of them is needed in order to find a solution. Each solution corresponds to a *solution tree* that starts at the root of the tree and includes one edge at each OR-branch and edges at each AND-branch. As an example, Figure 9.5 shows one of the solution trees in boldface.

Thus, plan-space planning leaves room for two types of node-selection heuristics: heuristics for selecting the next flaw, and heuristics for choosing the resolver of the selected flaw. Section 9.4.1 discusses the former type of heuristic, while the latter is discussed in Section 9.4.2.

9.4.1 Flaw-Selection Heuristics

Since a plan-space planner such as PSP repairs flaws one at a time, this means that in effect it searches a *serialization* of the AND/OR tree. The flaws that are not explored immediately do not appear at the top of the serialized tree but instead appear lower in that tree. For example, consider the AND-branch at the top of Figure 9.5. This branch represents three flaws in the partial plan π. Suppose PSP chooses to resolve

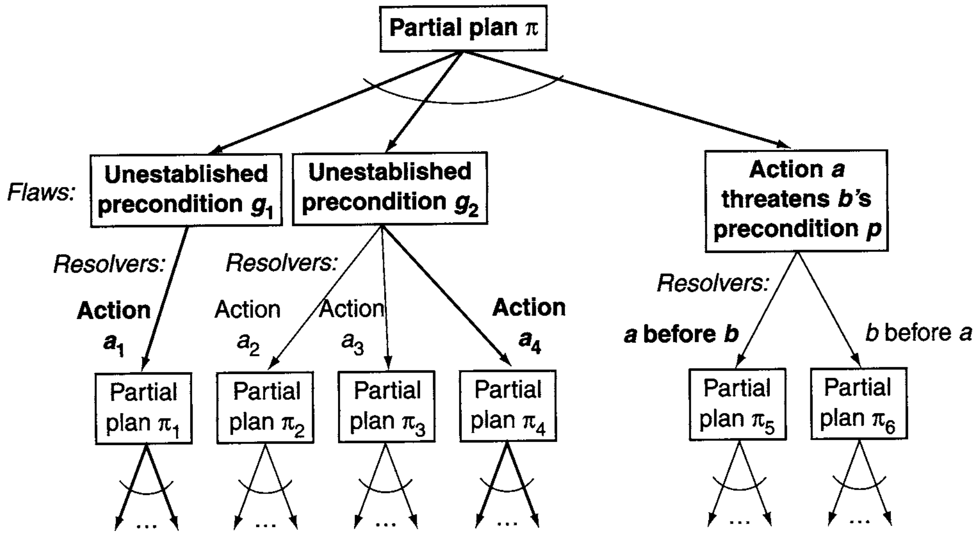


Figure 9.5 An AND/OR tree for flaw repair. Each set of flaws corresponds to an AND-branch, and each set of resolvers corresponds to an OR-branch. A solution tree is shown in boldface.

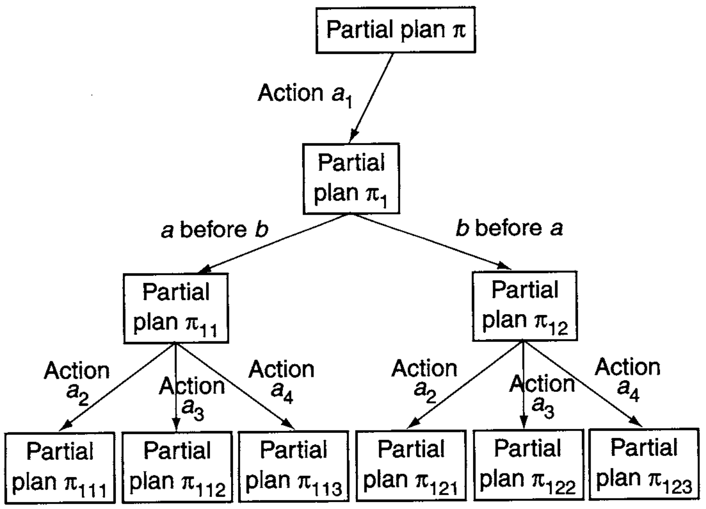


Figure 9.6 A serialization of the AND/OR tree shown in Figure 9.5.

these flaws in the following order: first find an establisher for g_1, then resolve the threat, and then find an establisher for g_2. Then PSP will develop the serialized tree shown in Figure 9.6. However, if PSP chooses to resolve these flaws in the opposite order, then it will explore the tree shown in Figure 9.7. Both of these trees are serializations of the AND/OR tree in Figure 9.5.

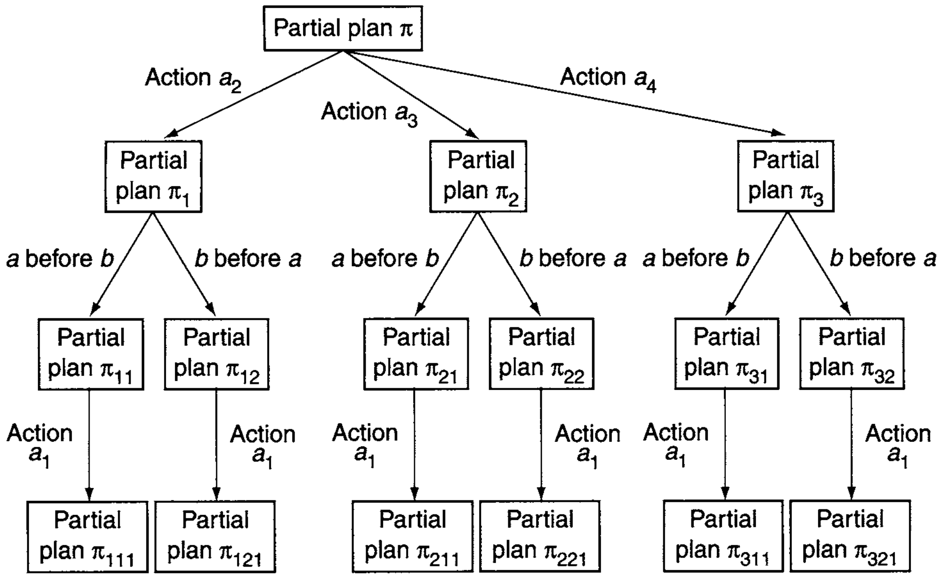


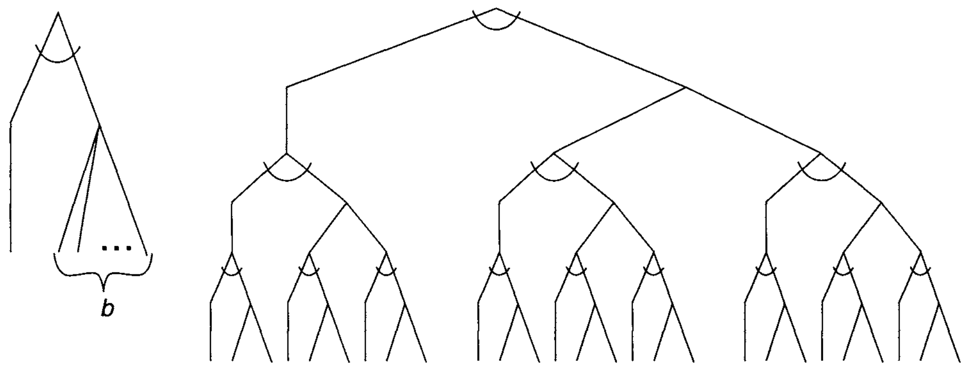
Figure 9.7 Another serialization of the AND/OR tree.

Each serialization of an AND/OR tree leads to exactly the same set of solutions, but different serializations contain different numbers of nodes. The speed of PSP varies significantly depending on the order in which it selects flaws to resolve.

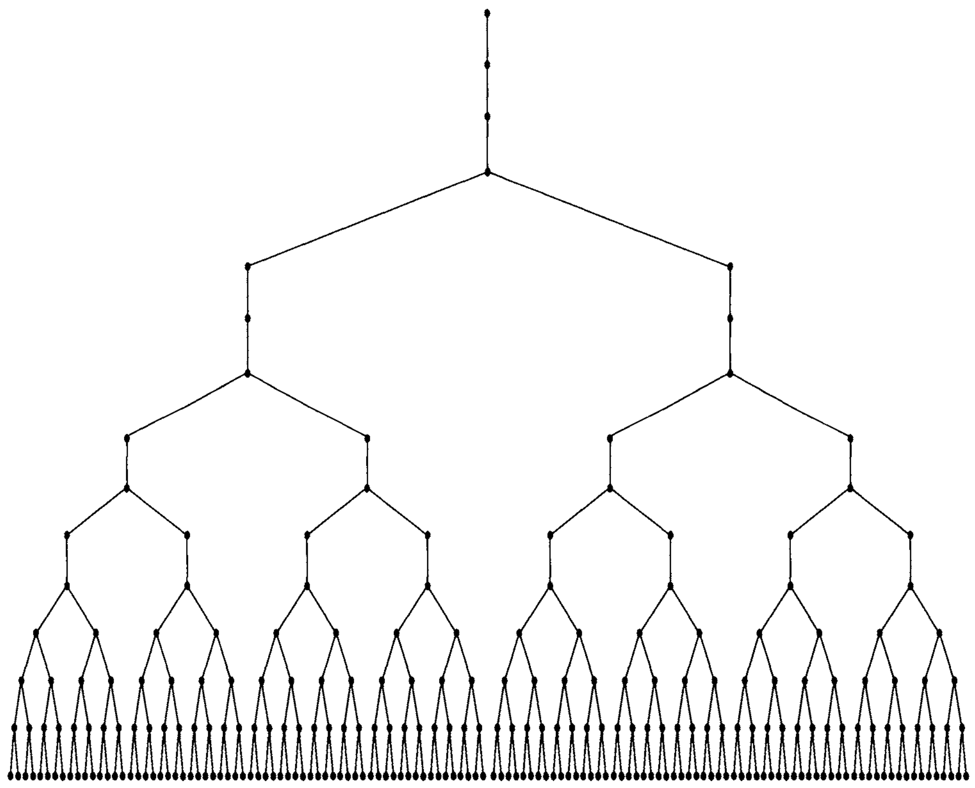
The fewest alternatives first (FAF) heuristic is to choose the flaw having the smallest number of resolvers. The rational for that is to work on flaws with the smallest branching factor as early as possible in order to limit the cost of eventual backtracks. The extreme case is a flaw with just one resolver: such a flaw does not involve any branching but only a refinement of the current node. For example, in Figure 9.5, there is one resolver for the unestablished precondition g_1, three for the unestablished precondition g_2, and two for the threat. Thus, the FAF heuristic would choose to resolve g_1 first, the threat next, and g_2 last. In this case, PSP would develop the tree shown in Figure 9.6. The FAF heuristic is easy to compute: the computation takes time $\Theta(n)$, where n is the number of flaws in a partial plan. Furthermore, experimental comparisons [288, 290] have shown that FAF works relatively well compared with other flaw-selection heuristics.

There is a limitation on how good a job FAF or any other flaw-selection heuristic can do: if $n_{\min}$ and $n_{\max}$ are, respectively, the sizes of the smallest and largest serializations of an AND/OR graph, then no resolver-selection heuristic can reduce the size of the search space by more than $n_{\max} - n_{\min}$. To get an idea of this limitation, suppose we take the pattern shown in Figure 9.8(a), and use it repeatedly to form an AND/OR tree $G_{b,k}$ of height $2k$, as shown in Figure 9.8(b). In $G_{b,k}$, the number of occurrences of the pattern is

$$c_{b,k} = 1 + (b+1) + (b+1)^2 + \ldots + (b+1)^{k-1} = \Theta(bk),$$



(a) Basic pattern, with parameter b (b) AND/OR tree $G_{2,3}$ produced by the pattern if $b = 2$ and $k = 3$



(c) The smallest possible serialization $T_{2,3}^{\text{min}}$ of $G_{2,3}$

Figure 9.8 A basic pattern (a), an AND/OR tree formed by repetitions of the pattern (b), and the smallest possible serialization of the AND/OR tree (c).

so the total number of nodes in $G_{b,k}$ is

$$n_{b,k} = 1 + (b+3)c_{b,k} = \Theta(bk).$$

Let $T_{b,k}^{\min}$ and $T_{b,k}^{\max}$ be the serializations of $G_{b,k}$ that have the smallest and largest numbers of nodes, respectively. Both of these trees have the same height $h_{b,k}$, which can be calculated recursively as follows:

$$\begin{aligned} h_{b,k} &= \begin{cases} 2 & \text{if } k = 1 \\ 2h_{b,k-1} + 2 & \text{otherwise} \end{cases} \\ &= \sum_{i=1}^{k} 2^i = 2^{k+1} - 2 \end{aligned}$$

$T_{b,k}^{\min}$ and $T_{b,k}^{\max}$ each have 2^{k-1} levels of unary OR-branches interspersed with 2^{k-1} levels of b-ary OR-branches. However, the unary OR-branches of $T_{b,k}^{\min}$ are as near to the top as possible, and the unary OR-branches of $T_{b,k}^{\min}$ are as near to the bottom as possible. As shown in Figure 9.8(c), the branches at the top k levels of $T_{b,k}^{\min}$ are unary and the branches at the bottom 2^{k-1} levels are all b-ary; the reverse is true for $T_{b,k}^{\max}$. It can be shown that the number of nodes in $T_{b,k}^{\min}$ is $n_{b,k}^{\min} = \Theta(b^{2^k})$ nodes and the number of nodes in $T_{b,k}^{\max}$ is $n_{b,k}^{\max} = \Theta(2^k b^{2^k})$.

In the above analysis, the good news is that if we can find the best possible serialization, we can potentially divide the number of nodes by an exponential amount, namely, $\Theta(2^k)$. However, the bad news is that this still leaves a doubly exponential number of nodes in the search space, namely, $\Theta(b^{2^k})$. Thus, in order for PSP to be efficient, it needs not just a good flaw-selection heuristic but also a good resolver-selection heuristic. Some resolver-selection heuristics are described in the next subsection.

9.4.2 Resolver-Selection Heuristics

This section focuses on the heuristic choice of a resolver for a subgoal. The techniques presented in Section 9.3 do not apply directly here because they rely on relaxed distances between states, while states are not explicit in the plan space (except for s_0). Hence, we have to come up with other means to rank the candidate nodes at a search point, which are here partial plans.

Let $\Theta = \{\pi_1, \ldots, \pi_m\}$ be the set of partial plans corresponding to the open nodes at some search point, and let us use $\eta(\pi)$ to denote a heuristic estimate of a partial plan π. Several such heuristics $\eta_0(\pi), \eta_1(\pi), \ldots$, will be introduced. Let g_π be the set of subgoals in partial plan $\pi \in \Theta$. Remember that g_π is the set of propositions in π without causal links. A very simple and intuitive heuristic is to choose in Θ the partial plan π that has the smallest set g_π, i.e., $\eta_1(\pi) = |g_\pi|$. However, this heuristic is not very informative.

A more informed heuristic involves building from each g_π an AND/OR graph, along regression steps defined by γ^{-1} down to some fixed level k. Let $\eta_k(g_\pi)$ be the weighted sum of: (1) the number of actions in this graph that are not in π, and (2) the number of subgoals remaining in its leaves that are not in the initial state s_0. Then $\eta_k(g_\pi)$ gives an estimate of how far g_π is from s_0, given π. The procedure then chooses the following partial plan: $\pi \leftarrow \text{argmin}\{\eta_k(\pi) \mid \pi \in \Theta\}$.

The heuristic η_k is more informative than η_0. It can be tuned for various values of the depth k of the AND/OR graph explored to assess g_π. However, η_k incurs a significant computational overhead.

A more efficient way to assess the distance in π between g_π and s_0 relies on the planning-graph data structure. Let us define $\delta_\pi(a)$ to be the set of actions *not* in π, i.e., $\delta_\pi(a) = 0$ when a is in π, and $\delta_\pi(a) = 1$ otherwise. This heuristic estimate $\eta(g_\pi)$ can be specified recursively as follows:

$$\eta(g_\pi) = \begin{cases} 0 & \text{if } g_\pi \subseteq s_0, \\ \infty & \text{if } \forall a \in A, a \text{ is not relevant for } g, \\ \max_p\{\delta_\pi(a) + \eta(\gamma^{-1}(g_\pi, a)) \mid p \in g_\pi \cap \text{effects}^+(a) & \\ \qquad \text{and } a \text{ relevant for } g_\pi\} \text{ otherwise.} & \end{cases} \tag{9.6}$$

To compute η, we first run an expansion phase that gives the planning graph for the problem at hand, once and for all, down to a level containing the goal propositions without mutex. Let $l(p)$ be the level of the first layer of the graph containing proposition p. Given g for some partial plan, we find the proposition $p \in g$ that has the highest level $l(p)$. We then apply, backward from p, the recursive definition of η for an action a that provides p, which may be already in π. The heuristic η is quite similar to η_k, but it is computed more efficiently because it relies on a planning graph computed once.

9.5 Discussion and Historical Remarks

Distance-based heuristics were initially proposed for a plan-space temporal planner called IxTeT [224], which introduced flaw-selection as well as resolver-selection heuristics relying on relaxation techniques for estimating the distances to the goal. A more systematic characterizations of the usefulness of distance-based heuristics in planning is due to the work of Bonet and Geffner on the HSP planner [81]. At a time were state-space planning was thought to be too simple to be efficient, this planner, with heuristics h_0 and a best-first search algorithm, outperformed several elaborate planners in the AIPS '99 planning competition. The backward-search improvement and the admissible heuristics [260] brought other improvements, with further extension for dealing with time and resources [261]. The relationship between these heuristics and Graphplan was identified very early [81].

Distance-based goal-ordering heuristics for Graphplan were proposed in several sources [304, 423, 425].

The success of HSP inspired Hoffmann's development of the Fast-Forward planning system [271, 272].[2] Fast-Forward's heuristic estimate is based on the same relaxation as HSP's, but Fast-Forward computes an explicit solution to the relaxed planning problem and uses information about this solution both to compute its heuristic estimate h and to prune some of the current state's successors. Fast-Forward also incorporates some modifications to its hill-climbing algorithm in an effort to prevent getting trapped in local minima [270]. These modifications worked well for Fast-Forward; it outperformed HSP in the AIPS '00 planning competition [28].

For plan-space planning, Pollack *et al.* [444] have done experimental comparisons of several different flaw-selection heuristics. In their experimental results, the FAF heuristic performed well, but no one flaw-selection heuristic consistently dominated the others. Our analysis of the limitations of flaw-selection heuristics for PSP is adapted from the one by Tsuneto *et al.* [517]. Aside from being viewed as a state-space planning algorithm, forward search can alternatively be viewed as a flaw-selection heuristic for plan-space planning: the next flaw to be selected is the one that will occur first in the plan being developed. Although forward search is not particularly effective by itself, it has the advantage that the current state is known at each point in the planning process. This enables the use of the node-selection functions discussed above, as well as the use of some powerful domain-dependent pruning rules, which are discussed in Chapter 10. In HTN planning, which will be introduced in Chapter 11, the flaws to be resolved include unachieved subtasks and constraints that have not yet been enforced. Tsuneto *et al.* have shown that the FAF heuristic works well here but that it is helpful to supplement FAF with an analysis of the preconditions of tasks in the task network [516, 517].

Analogously to forward search, the forward-decomposition approach to HTN planning (see Chapter 11) can be viewed as a flaw-selection heuristic in which the next flaw to be selected is the one that will occur first in the resulting plan. As with forward search, forward decomposition has the advantage that the current state is known at each point in the planning process.

9.6 Exercises

9.1 Compute the values of h_0, h_1, and h_2 at level P_2 of the planning graph in Figure 6.4. Compare these values with each other and with the true distance to a solution.

9.2 Compute the values of h_0, h_1, and h_2 at level P_1 of the planning graph in Figure 6.4. Compare these values with each other and with the true distance to a solution.

9.3 Prove directly that the heuristic functions h_k, for $k \geq 1$, are admissible.

2. See http://www.informatik.uni-freiburg.de/~hoffmann/ff.html.

9.4 Modify the Delta algorithm to compute distances from s_0 to pairs of propositions either systematically to every pair or incrementally. Detail a backward-search procedure with heuristic h_2 that uses this modified Delta.

9.5 Let $\mathcal{P}$ be a planning problem whose state space is a binary tree T whose root node is the initial state s_0. Suppose that every node at depth d is a leaf node, and no other nodes are leaf nodes. Suppose that there is exactly one node s_g that is a goal node and that this node is a leaf of T. Suppose we have a heuristic function h such that at each node s that is an ancestor of s_g, there is a probability of 0.9 that h will recommend the action that takes us toward s_g. Suppose a Heuristic-forward-search algorithm starts at s_0 and uses h to decide which action to choose at each node.

(a) What is the probability that s_g will be the first leaf node visited by the algorithm?

(b) What is the probability that the algorithm will visit all of the other leaf nodes before visiting s_g?

(c) On average, how many other leaf nodes will the algorithm visit before visiting s_g?

9.6 How many different serializations are there for the AND/OR tree in Figure 9.5?

9.7 Let T be a binary AND/OR tree in which the topmost branch is an AND-branch, every node at depth d is a leaf node, and no other nodes are leaf nodes.

(a) How many leaf nodes are there in any serialization of T?

(b) How many different serializations are there for T?

(c) What are the smallest and largest possible numbers of leaf nodes in any serialization of T?

(d) Let e be an edge at the bottom of the tree. Then e will correspond to several different edges in any serialization of T. What are the smallest and largest possible values for this number of edges?