Part III Heuristics and Control Strategies

Although planning systems have become much more efficient over the years, they still suffer from combinatorial complexity. As described in Chapter 3 on the complexity of planning, even relatively restricted planning domains can be intractable in the worst case.

The four chapters in this part of the book are devoted to several approaches for improving the efficiency of planning. To provide a foundation for presenting these approaches, we now describe an abstract common framework for the planning procedures presented earlier in the book.

Nearly all of the planning procedures in Parts I and II of this book can be viewed as instances of the Abstract-search procedure shown in Figure III.1. The objective of Abstract-search is to find at least one solution, if a solution exists, without enumerating all of the set explicitly.

Abstract-search nondeterministically searches a space in which each node u represents a set of solution plans Π_u, namely, the set of all solutions that are reachable from u. A node u is basically a structured collection of actions and constraints.

- In state-space planning, u is simply a sequence of actions. Every solution reachable from u (i.e., every plan in Π_u) contains this sequence as either a prefix or a suffix, depending on whether we are doing a forward search or a backward search.
- In plan-space planning, u is a set of actions together with causal links, ordering constraints, and binding constraints. Every solution reachable from u contains all the actions in u and meets all the constraints.

- In the Graphplan algorithm, u is a subgraph of a planning graph, i.e., a sequence of sets of actions, together with constraints on preconditions, effects, and mutual exclusions. Each solution reachable from u contains the actions in u corresponding to the solved levels and at least one action from each level of the subgraph that has not been solved yet in u.
- In SAT-based planning, u is a set of assigned literals and remaining clauses, each of which is a disjunction of literals that describe actions and states. Each solution reachable from u corresponds to an assignment of truth values to the unassigned literals such that all of the remaining clauses are satisfied.
- In CSP-based planning, u is a set of CSP variables and constraints, some variables having already assigned values. Each solution reachable from u includes these assigned variables and assignments to all other CSP variables that satisfy the constraints.

In state-space and plan-space planning; u is a partial plan; every action in u belongs to any solution plan in Π_u. But in the latter three approaches just listed, not every action in u appears in a solution plan in Π_u.[1] For example, several actions may achieve a propositional goal in a planning graph, but perhaps only one of them will be needed in a solution plan in Π_u.

The Abstract-search procedure involves three main steps in addition to a termination step.

1. A *refinement step* consists of modifying the collection of actions and/or constraints associated with a node u. In a refinement of u, the set Π_u of solutions corresponding to u remains *unchanged.* For example, if we find out that there is only one action a that meets a particular constraint in u, then we might make a an explicit part of u and remove the constraint.
2. A *branching step* generates one or more children of u; these nodes will be the *candidates* for the next node to visit. Each child v of the node u represents a *subset* of solution $\Pi_v \subseteq \Pi_u$. For example, in a forward state-space search, each child would correspond to appending a different action to the end of a partial plan. Branching does not need to generate explicitly all the child nodes of u.
3. A *pruning step* consists of removing from the set of candidate nodes $\{u_1, \ldots, u_k\}$ some nodes that appear to be unpromising for the search. For example, we might consider a node to be unpromising if we have a record of having already visited that node or if we have some domain-specific reasons to believe that it is unlikely to contain any desirable solutions.[2]

1. Because of this property, the latter three approaches have sometimes been called *disjunctive-refinement approaches* [306].
2. What constitutes a "desirable" solution depends largely on the planning domain. Sometimes a planning problem can be viewed as an optimization problem involving a numeric objective function such as the "cost" or "utility" of a plan. In other cases, it may be a Pareto optimization problem [209] that

```
Abstract-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
    if $C = \emptyset$ then return(failure)
    nondeterministically choose $v \in C$
    return(Abstract-search($v$))
end
```

Figure III.1 An abstract search procedure that provides a general framework for describing classical and neoclassical planning procedures.

In some planning procedures, the order of these three steps may be different. For example, a procedure might do branching, then pruning, then refinement. Also, some planning procedures may extend Abstract-search to use a control mechanism such as heuristic search or iterative deepening (see Appendix A).

Here is a brief discussion of how the refinement, branching, pruning, and termination steps of Abstract-search relate to previous planning procedures.

In plan-space planning, branching consists of selecting a flaw and finding its resolvers; refinement consists of applying such a resolver to the partial plan; and there is no pruning step. Termination occurs when no flaws are left in the plan. Since paths in the plan space are likely to be infinite, a control strategy such as best-first search or iterative deepening should be used.

The relationship between Abstract-search and state-space planning is also very direct: branches are actions; pruning removes candidate nodes corresponding to cycles; and termination occurs when the plan goes all the way from the initial state to a goal.

Graphplan corresponds to using Abstract-search with an iterative-deepening control strategy. Refinement consists of propagating constraints for the actions chosen in the branching step. Branching identifies possible actions that achieve subgoals. Pruning uses the recorded nogood tuples of subgoals that failed in some layer. Termination occurs if the solution-extraction process succeeds.

In SAT planning, refinement is exactly the Unit-Propagate procedure. Branching consists of selecting some proposition and calling Davis-Putnam with both of the possible truth values (true and false) for that proposition. Termination occurs if no clause remains unsatisfied.

involves several competing objective functions (e.g., cost, time, number of goals achieved). In other cases, a plan's desirability may depend on things like how easily it can be modified during execution to accomodate unforeseen events (e.g., unanticipated delays or failures, new goals to achieve).

Finally, CSP-based planning branches over the consistent values in the domain of a CSP variable; the refinement step consist of propagating the chosen value for that variable over remaining constraints.

Note that we may need to extend Abstract-search with some type of iterative-deepening control mechanisms in order to model the expansion phase of Graphplan or the control needed for handling the infinite search space in plan-space planning, for example.

Abstract-search includes a nondeterministic step that chooses among the current node's nonpruned children, if any, to get the next node to visit in the search space. As discussed in Chapter 4 and Appendix A, this essential step is usually implemented as a depth-first backtracking search. Thus, a *node-selection function* Select(C) is needed to choose which child $u \in C$ to visit next, i.e., in place of the nondeterministic choice in Figure III.1 we will have a step such as $v \leftarrow$ Select(C).

In general, the search procedure will perform well if the Refine, Branch, Prune, and Select procedures can do a good job of organizing the search space and guiding the search. A number of heuristic techniques have been developed to try to accomplish this.

Sometimes, the heuristics are *domain-independent*, i.e., intended for use in many different planning domains. Chapter 9 describes several domain-independent heuristics for node selection.

Alternatively, the heuristics may be *domain-specific*, i.e., tailor-made for a particular kind of domain. One way to reduce the effort needed to write these heuristics is to provide a domain-independent planning engine plus a language in which to write the heuristics. Chapter 10 describes how to use *temporal logic* to write node-pruning rules that focus the search. Chapter 11 describes how to use *task decomposition* to write branching rules. Chapter 12 describes how to specify search strategies in two different logic formalisms: situation calculus and dynamic logic. The formalisms described in these chapters have expressive power that goes substantially beyond classical planning: they can accommodate numeric computations, external function calls, axiomatic inference, time durations, and overlapping actions.

Although it can take some effort to write and tune domain-specific heuristics, the case for doing so is quite strong. In the last two AI planning competitions [28, 195], the systems that were fastest, could solve the hardest problems, and could handle the widest variety of planning domains all incorporated domain-specific control of the planning process. Furthermore, nearly every successful planning system for real-world applications incorporates domain-specific heuristics or even domain-specific algorithms.

To end this introduction, here is a brief discussion of other proposed unifying frameworks for planning.

A general formulation of planning as a *refinement search* has been proposed and analyzed in [302, 306]. The notion of refinement used in that work is slightly different from ours, since it includes branching and pruning. However some of the notions described there can be transported easily to Abstract-search. Examples include *completeness*, i.e., the property that every solution in u is included in at least one of the nodes of B', and *systematicity*, i.e., the property that the nodes of B' define a partition of the set of solutions in u. It is important to note that completeness requires that only nodes in B corresponding to an empty set of solutions can be pruned.

Another unifying framework [427] views planning as a *branch-and-bound* search. Branch-and-bound is a widely known technique for *optimization problems*. It maintains an upper bound λ on the cost of the optimal solution, which can simply be the cost of the best solution found so far. When the procedure visits a new node u in its search space, it calls a domain-specific lower bound function $l(u)$ that returns a lower bound on the cost of the best solution in Π_u. If $l(u) > \lambda$, then the procedure prunes the node u; if u is terminal, then the procedure updates λ and decides either to stop or to pursue some promising open node. More generally, branch-and-bound can incorporate an abstract pruning function similar to our Prune function, which can prune nodes based on criteria other than just bounding [415]. Our Abstract-search procedure can be viewed as a version of such a branch-and-bound procedure.