Lecture slides for
Automated Planning: Theory and Practice

Part III
Heuristics and Control Strategies

Dana S. Nau
University of Maryland

1:32 PM February 29, 2012

Motivation for Part 3 of the Book

- Domain-independent planners suffer from combinatorial complexity
  - Planning is in the worst case intractable
  - Need ways to control the search

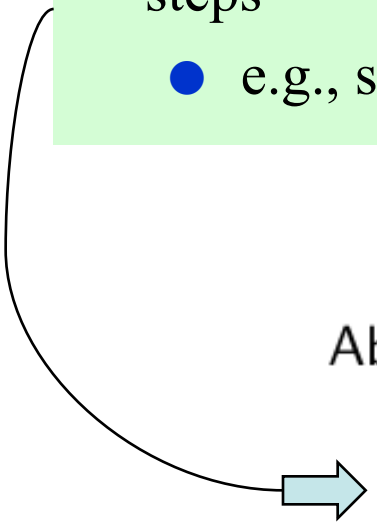
Abstract Search Procedure

- Here is a general framework for describing classical and neoclassical planners
- The planning algorithms we've discussed all fit into the framework, if we vary the details
  - e.g., the steps don't have to be in this order

```
Abstract-search(u)
    if Terminal(u) then return(u)
    u ← Refine(u)          ;;  refinement step
    B ← Branch(u)          ;;  branching step
    B' ← Prune(B)          ;;  pruning step
    if B' = ∅ then return(failure)
    nondeterministically choose v ∈ B'
    return(Abstract-search(v))
end
```

Abstract Search Procedure

- Compute information that may affect how we do some of the other steps
  - e.g., select a flaw to work on next, or compute a planning graph



```
Abstract-search(u)
    if Terminal(u) then return(u)
    u ← Refine(u)          ;;  refinement step
    B ← Branch(u)          ;;  branching step
    B' ← Prune(B)          ;;  pruning step
    if B' = ∅ then return(failure)
    nondeterministically choose v ∈ B'
    return(Abstract-search(v))
end
```

Abstract Search Procedure

- Divide current set of solutions into several sets to be explored in parallel
  - e.g., $B' \leftarrow \{\pi.a \mid a \text{ is applicable to } \gamma(s_0,\pi)\}$

```
Abstract-search(u)
    if Terminal(u) then return(u)
    u ← Refine(u)          ;;  refinement step
    B ← Branch(u)          ;;  branching step
    B' ← Prune(B)          ;;  pruning step
    if B' = ∅ then return(failure)
    nondeterministically choose v ∈ B'
    return(Abstract-search(v))
end
```

Abstract Search Procedure

- Remove some unpromising members of B
  - e.g., loop detection, constraint violation

```
Abstract-search(u)
    if Terminal(u) then return(u)
    u ← Refine(u)          ;;  refinement step
    B ← Branch(u)          ;;  branching step
    B' ← Prune(B)          ;;  pruning step
    if B' = ∅ then return(failure)
    nondeterministically choose v ∈ B'
    return(Abstract-search(v))
end
```

Plan-Space Planning

- Refinement: select which flaw to work on next
- Branching: {the flaw's resolvers}
- Pruning: loop detection
  - recall this is weak for plan-space planning

```
Abstract-search(u)
    if Terminal(u) then return(u)
    u ← Refine(u)          ;;  refinement step
    B ← Branch(u)          ;;  branching step
    B' ← Prune(B)          ;;  pruning step
    if B' = ∅ then return(failure)
    nondeterministically choose v ∈ B'
    return(Abstract-search(v))
end
```

State-Space Planning

- Refinement: none
- Branching: {applicable or relevant actions}
- Pruning: loop detection
  - Other branching & pruning techniques in Chapters 10 & 11

```
Abstract-search(u)
    if Terminal(u) then return(u)
    u ← Refine(u)          ;;  refinement step
    B ← Branch(u)          ;;  branching step
    B' ← Prune(B)          ;;  pruning step
    if B' = ∅ then return(failure)
    nondeterministically choose v ∈ B'
    return(Abstract-search(v))
end
```

Planning-Graph Planning

- Wrap iterative deepening around Abstract-search
- Refinement: generate the planning graph, compute mutex info
- Branching: {sets of actions in action-level i that achieve goals at state-level i}
- Pruning: prune sets of actions that are mutex

for *number of levels* = 0, 1, 2, …
    if Terminal(u) then return(u)
    $u \leftarrow$ Refine(u) ;; *refinement step*
    $B \leftarrow$ Branch(u) ;; *branching step*
    $B' \leftarrow$ Prune(B) ;; *pruning step*
    if $B' = \emptyset$ then return(failure)
    nondeterministically choose $v \in B'$
    return(Abstract-search(v))
end

Search Heuristics

- Chapter 9: Heuristics in Planning
  - Heuristics for choosing where to search next
  - The heuristics in this chapter are *domain-independent* within classical planning

```
Abstract-search(u)
    if Terminal(u) then return(u)
    u ← Refine(u)          ;;  refinement step
    B ← Branch(u)          ;;  branching step
    B' ← Prune(B)          ;;  pruning step
    if B' = ∅ then return(failure)
    nondeterministically choose v ∈ B'
    return(Abstract-search(v))
end
```

Chapter 9

Chapter 9

Branching and Pruning Techniques

- Chapter 10: pruning via search-control rules
- Chapter 11: branching via hierarchical task decomposition
- These chapters discuss *domain-configurable* state-space planners
  - Domain-independent planning engine
  - Domain-specific information to control the search

```
Abstract-search(u)
    if Terminal(u) then return(u)
    u ← Refine(u)          ;;  refinement step
    B ← Branch(u)          ;;  branching step
    B' ← Prune(B)          ;;  pruning step
    if B' = ∅ then return(failure)
    nondeterministically choose v ∈ B'
    return(Abstract-search(v))
end
```

Chapter 11

Chapter 10

Branching Versus Pruning

- Two equivalent approaches:
  - Generate all possible branches, then prune some of them
  - Just don't bother generating the ones that would be pruned
- Example:
  - Domain-configurable implementations of the block-stacking algorithm from Chapter 4
  - Separate branching and pruning (Chapter 10)
    - *Branch:* generate all applicable actions
    - *Prune:* prune actions that build up "bad" stacks or tear down "good" ones
  - Combined branching and pruning (Chapter 11)
    - Only generate actions that don't build up "bad" stacks and don't tear down "good" ones