Chapter 9
Heuristics in Planning

Dana S. Nau
University of Maryland

3:08 PM March 7, 2012
Planning as Nondeterministic Search

Abstract-search(\(u\))
    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
Making it Deterministic

```
Depth-first-search(u)
    if Terminal(u) then return(u)
    u ← Refine(u) ;; refinement step
    B ← Branch(u) ;; branching step
    C ← Prune(B) ;; pruning step
    while C ≠ ∅ do
        v ← Select(C) ;; node-selection step
        C ← C − {v}
        π ← Depth-first-search(v)
        if π ≠ failure then return(π)
    return(failure)
```

Digression: the A* algorithm (on trees)

- Suppose we’re searching a tree in which each edge \((s,s')\) has a cost \(c(s,s')\)
  - If \(p\) is a path, let \(c(p)\) = sum of the edge costs
  - For classical planning, this is the length of \(p\)

- For every state \(s\), let
  - \(g(s)\) = cost of the path from \(s_0\) to \(s\)
  - \(h^*(s)\) = least cost of all paths from \(s\) to goal nodes
  - \(f^*(s) = g(s) + h^*(s)\) = least cost of all paths from \(s_0\) to goal nodes that go through \(s\)

- Suppose \(h(s)\) is an estimate of \(h^*(s)\)
  - Let \(f(s) = g(s) + h(s)\)
    - \(f(s)\) is an estimate of \(f^*(s)\)
  - \(h\) is admissible if for every state \(s\), \(0 \leq h(s) \leq h^*(s)\)
  - If \(h\) is admissible then \(f\) is a lower bound on \(f^*\)
The A* Algorithm

- A* on trees:
  
  loop
    choose the leaf node \( s \) such that \( f(s) \) is smallest
    if \( s \) is a solution then return it and exit
    expand it (generate its children)

- On graphs, A* is more complicated
  - additional machinery to deal with multiple paths to the same node

- If a solution exists (and certain other conditions are satisfied), then:
  - If \( h(s) \) is admissible, then A* is guaranteed to find an optimal solution
  - The more “informed” the heuristic is (i.e., the closer it is to \( h^* \)), the smaller the number of nodes A* expands
  - If \( h(s) \) is within \( c \) of being admissible, then A* is guaranteed to find a solution that’s within \( c \) of optimal
Hill Climbing

- Use $h$ as a node-selection heuristic
  - Select the node $v$ in $C$ for which $h(v)$ is smallest
- Why not use $f$?
- Do we care whether $h$ is admissible?

```plaintext
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
```
FastForward (FF)

- Depth-first search
- Selection heuristic: *relaxed Graphplan*
  - Let $v$ be a node in $C$
  - Let $P_v$ be the planning problem of getting from $v$ to a goal
  - use Graphplan to find a solution for a relaxation of $P_v$
  - The length of this solution is a lower bound on the length of a solution to $P_v$
Selection Heuristic

- Given a planning problem $P_v$, create a relaxed planning problem $P'_v$ and use GraphPlan to solve it
  - Convert to set-theoretic representation
    - No negative literals; goal is now a set of atoms
  - Remove the delete lists from the actions
  - Construct a planning graph until a layer is found that contains all of the goal atoms
  - The graph will contain no mutexes because the delete lists were removed
  - Extract a plan $\pi'$ from the planning graph
    - No mutexes $\implies$ no backtracking $\implies$ polynomial time
- $|\pi'|$ is a lower bound on the length of the best solution to $P_v$
FastForward

- FF evaluates all the nodes in the set $C$ of $u$’s successors
- If none of them has a better heuristic value than $u$, FF does a breadth-first search for a state with a strictly better evaluation
- The path to the new state is added to the current plan, and the search continues from this state
- Works well because plateaus and local minima tend to be small in many benchmark planning problems

- Can’t guarantee how fast FF will find a solution, or how good a solution it will find
  - However, it works pretty well on many problems
AIPS-2000 Planning Competition

- FastForward did quite well
- In the this competition, all of the planning problems were classical problems
- Two tracks:
  - “Fully automated” and “hand-tailored” planners
  - FastForward participated in the fully automated track
    - It got one of the two “outstanding performance” awards
  - Large variance in how close its plans were to optimal
    - However, it found them very fast compared with the other fully-automated planners
Among the automated planners, FastForward was roughly in the middle.

LPG (graphplan + local search) did much better, and got a “distinguished performance of the first order” award.

It’s interesting to see how FastForward did in problems that went beyond classical planning:
- Numbers, optimization

Example: Satellite domain, numeric version
- A domain inspired by the Hubble space telescope (a lot simpler than the real domain, of course)
  - A satellite needs to take observations of stars
  - Gather as much data as possible before running out of fuel
- Any amount of data gathered is a solution
  - Thus, FastForward always returned the null plan.
2004 International Planning Competition

- FastForward’s author was one of the competition chairs
  - Thus FastForward did not participate
Plan-Space Planning

Abstract-search($u$)
  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

- **Refine** = select next flaw to work on
- **Branch** = generate resolvers
- **Prune** = remove some of the resolvers
- **nondeterministic choice** = resolver selection
Abstract-search($u$)
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

Flaw Selection

- Must eventually resolve all of the flaws, regardless of which one we choose first
  - an “AND” branch
Serializing and AND/OR Tree

- The search space is an AND/OR tree

- Deciding what flaw to work on next = serializing this tree (turning it into a state-space tree)
  - at each AND branch, choose a child to expand next, and delay expanding the other children
One Serialization

partial plan $\pi$

action $a_1$

partial plan $\pi_1$

$a$ before $b$

$b$ before $a$

partial plan $\pi_{11}$

action $a_2$

partial plan $\pi_{11}$

action $a_3$

partial plan $\pi_{11}$

action $a_4$

partial plan $\pi_{11}$

partial plan $\pi_{12}$

action $a_2$

partial plan $\pi_{12}$

action $a_3$

partial plan $\pi_{12}$

action $a_4$

partial plan $\pi_{12}$
Another Serialization

```
partial plan π

action a_2

partial plan π_1
  a before
  b before

  partial plan π_{11}
  partial plan π_{12}

action a_4

partial plan π_3
  a before
  b before

  partial plan π_{31}
  partial plan π_{32}

action a_3

partial plan π_2
  a before
  b before

  partial plan π_{21}
  partial plan π_{22}

action a_1

partial plan π_{111}
partial plan π_{121}
partial plan π_{211}
partial plan π_{221}
partial plan π_{311}
partial plan π_{321}
```
Why Does This Matter?

- Different refinement strategies produce different serializations
  - the search spaces have different numbers of nodes
- In the worst case, the planner will search the entire serialized search space
- The smaller the serialization, the more likely that the planner will be efficient

- One pretty good heuristic: fewest alternatives first
A Pretty Good Heuristic

- Fewest Alternatives First (FAF)
  - Choose the flaw that has the smallest number of alternatives
  - In this case, unestablished precondition $g_1$

![Diagram showing flaws and resolvers](image)
How Much Difference Can the Refinement Strategy Make?

- Case study: build an AND/OR graph from repeated occurrences of this pattern:

- Example:
  - number of levels $k = 3$
  - branching factor $b = 2$

- Analysis:
  - Total number of nodes in the AND/OR graph is $n = \Theta(b^k)$
  - How many nodes in the best and worst serializations?
Case Study, Continued

- The best serialization contains \( \Theta(b^{2^k}) \) nodes
- The worst serialization contains \( \Theta(2^k b^{2^k}) \) nodes
  - The size differs by an exponential factor
  - But both serializations are *doubly* exponentially large
- This limits how good *any* flaw-selection heuristic can do
  - To do better, need good ways to do node selection, branching, pruning
Resolver Selection

- This is an “or” branch

```
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
```

Diagram showing the resolver selection process with flaws and resolvers for unwritten partial plans and actions.