Midterm Review

Prateek Tandon, John Dickerson
## Basic Uninformed Search (Summary)

<table>
<thead>
<tr>
<th>Criterion</th>
<th>Breadth-First</th>
<th>Uniform-Cost</th>
<th>Depth-First</th>
<th>Depth-Limited</th>
<th>Iterative Deepening</th>
<th>Bidirectional (if applicable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>$b^d$</td>
<td>$b^d$</td>
<td>$b^m$</td>
<td>$b^l$</td>
<td>$b^d$</td>
<td>$b^{d/2}$</td>
</tr>
<tr>
<td>Space</td>
<td>$b^d$</td>
<td>$b^d$</td>
<td>$bm$</td>
<td>$bl$</td>
<td>$bd$</td>
<td>$b^{d/2}$</td>
</tr>
<tr>
<td>Optimal?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Complete?</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes, if $l \geq d$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

$b = $ branching factor  
$d = $ depth of shallowest goal state  
$m = $ depth of the search space  
$l = $ depth limit of the algorithm
CSP Solving - Backtracking search

• Depth-first search for CSPs with single-variable assignments is called backtracking search

• Improvements:
  – Most constrained variable/Minimum Remaining Values – choose the variable with the fewest legal values
  – Least constraining variable – choose the variable that rules out the fewest values in the remaining value
  – Forward checking – Keep track of remaining legal values and terminate when a variable has no remaining legal values
  – Arc Consistency (AC3) – propagate information across arcs
  – Conflict-Directed Backjumping – maintain a conflict set and backjump to a variable that might help resolve the conflict
A* Search

function A*-SEARCH (problem) returns a solution or failure
return BEST-FIRST-SEARCH (problem, g+h)

\[ f(n) = \text{estimated cost of the cheapest solution through } n \]
\[ = g(n) + h(n) \]
In a minimization problem, an admissible heuristic $h(n)$ never overestimates the real value.

(In a maximization problem, $h(n)$ is admissible if it never underestimates)

Best-first search using $f(n) = g(n) + h(n)$ and an admissible $h(n)$ is known as $A^*$ search.

$A^*$ tree search is complete & optimal.
Iterative Deepening A* (IDA*)

**function** IDA*(problem) **returns** a solution sequence

**inputs:** problem, a problem

**static:** f-limit, the current f-COST limit

   root, a node

root ← MAKE-NODE(INITIAL-STATE[problem])  
f-limit ← f-COST(root)

**loop** do
   solution, f-limit ← DFS-CONTOUR(root,f-limit)
   if solution is non-null **then return** solution
   if f-limit = ∞ **then return** failure; end

**function** DFS-CONTOUR(node,f-limit) **returns** a solution sequence and a new f-COST limit

**inputs:** node, a node

   f-limit, the current f-COST limit

**static:** next-f, the f-COST limit for the next contour, initially ∞

if f-COST[node] > f-limit **then return** null, f-COST[node]  
if GOAL-TEST[problem](STATE[node]) **then return** node, f-limit  
for each node s in SUCCESSOR(node) do
   solution, new-f ← DFS-CONTOUR(s,f-limit)
   if solution is non-null **then return** solution, f-limit  
   next-f ← MIN(next-f, new-f); end

return null, next-f

\[
f-COST[node] = g[node] + h[node]\]
Map of Romania showing contours at $f = 380$, $f = 400$ and $f = 420$, with Arad as the start state. Nodes inside a given contour have $f$-costs lower than the contour value.
LP, IP, MIP, WDP, etc ...

- Topics you should know at a high level:
  - LP: visual representation of simplex
  - (M)IP: Branch and cut (what are cuts? Why do we use them?)
    - Cuts should separate LP optimum from integer points
  - Gomory cuts:

- Topics you should know well:
  - Formulating a combinatorial search problem as an IP/MIP (think HW2, P2)
  - (M)IP: Branch and bound (upper bounds, lower bounds, proving optimality)
  - Principle of least commitment (stay flexible)
Planning Review

• STRIPS – basic representation
• Linear Planning – work on one goal at a time. Solve goal completely before moving onto the next one.
  – Reduces search space since goals are solved one at time.
  – But this leads to incompleteness [Sussman Anomaly]
  – Planner’s efficiency is sensitive to goal orderings
  – Concrete implementation as an algorithm: GPS [look over example in slides]
• Partial-Order Planning – only constrain the ordering in the problem only as much as you need to at the current moment.
  – Sound and complete whereas Linear Planning is only sound
• Graph plan – try to “preprocess” the search using a planning graph
• SatPlan – generate boolean SAT formula for plan
  – What was the limitation?
Planning Graph

Adds a level until either a solution is found by EXTRACT-SOLUTION [either CSP or backwards search] or no solution exists.
Mutex Rules for Actions

• Mutex between two actions at a given level:
  – Inconsistent effects: One action negates the effect of the other
  – Interference: One of the effects of an action is the negation of a precondition of the other
  – Competing needs: One of the preconditions of one action is mutually exclusive with a precondition of the other.
Mutex Rules for Literals

- Literals negation of the other [easy]
- Inconsistent support – if each possible pair of actions from the prior action graph level that could achieve the two literals is mutually exclusive.
  - Check to see if pairs of actions that produce literals are mutex on the past action level.
  - Look at Book example