# Uninformed Search

Lecture 2

# Outline

# Outline

1. Problem Formulation
2. Search Trees
3. Uninformed Search

# Problems

- A problem is defined as a 5-tuple:
  1. A set of operators, or actions, available to the agent.
  2. An initial state.
  3. A state space: the set of states reachable from the initial state by any sequence of actions.
  4. A goal test, which the agent applies to a state to determine if it is a goal state.
  5. A path cost function: a function that assigns cost to a sequence of actions. Typically, it is the sum of the costs of individual actions in the sequence.

# Solutions

- A search algorithm takes a problem as input and returns a solution as output.
- The solution is a sequence of actions from the initial state to a state satisfying the goal test.
- A solution with smaller path cost is preferable.

# Outline

# The Essence of Search

- Recall: Starting at the initial state, we need to know what to do next, taking into account a goal we want to achieve.
- We expand the current state by, hypothetically, applying the various operators.
- We choose one of the, yet un-expanded, states and expand it.
- We continue until we attempt to expand a node satisfying the goal test.
- The order in which nodes are chosen for expansion is determined by the search strategy.

# The Search Tree

- The search process may be thought of as building a tree through the state space.
- The root of the search tree is the initial state of the problem.
- A leaf of the search tree is a node which has either
  1. not yet been expanded, or
  2. been expanded, but has no successors.

# The State Space and the Search Tree

- The state space is *not* the search tree.
  1. Topologically, the state space is a directed graph; the search tree is a tree.
  2. Nodes of the search tree are not mere states; they typically carry more information.
  3. The state space is finite; the search tree may be infinite.

# Search Tree Nodes

- Following Russell and Norvig, we shall consider nodes to be 5-tuples:
  1. The state of the state space that this node corresponds to.
  2. The parent node.
  3. The operator applied to generate this node.
  4. The depth of the node in the tree.
  5. The path cost from the root.

# General Search

**function** GENERAL-SEARCH(*problem*, QING-FUN)
  **returns** a solution, or failure
 *nodes* ⟵ MAKE-Q(MAKE-NODE(INIT-STATE(*problem*)))
 **loop do**
  **If** *nodes* is empty **then return** failure
  *node* ⟵ REMOVE-FRONT(*nodes*)
  **If** GOAL-TEST(*problem*)(STATE(*node*)) **then return** *node*
  *nodes* ⟵ QING-FUN(*nodes*, EXPAND(node, OPER(*problem*)))
 **end**

# Evaluation Criteria

- Search strategies are evaluated according to four criteria:
  1. Completeness: Is it guaranteed to find a solution if there is one?
  2. Time Complexity: How long does it take to find a solution?
  3. Space Complexity: How much memory is needed?
  4. Optimality: Is the best solution found?

# Outline

1. Problem Formulation
2. Search Trees
3. Uninformed Search

# Breadth-First Search

**function** BF-SEARCH(*problem*)
    **returns** a solution, or failure
  **return** GENERAL-SEARCH(*problem*, ENQUEUE-AT-END)

- Expands all nodes of depth $d$ before those of depth $d + 1$.

# Evaluation

- Complete.
- Optimal only if some conditions are satisfied by the path cost function.
  - If path cost to a node is a non-decreasing function of the depth of the node, plus possibly other conditions.
- Time complexity: $O(b^d)$, where $b$ is the branching factor and $d$ is the depth of the shallowest solution.
- Space complexity: $O(b^d)$.

# Uniform Cost Search

**function** UC-SEARCH(*problem*)
    **returns** a solution, or failure
  **return** GENERAL-SEARCH(*problem*, ORDERED-INSERT)

- Expands nodes with lowest path cost first.
- Identical to BFS when path cost is identical to path length.

# Evaluation

- Complete, provided that the cost of node is less than the cost of successors.
- Optimal, provided that cost of node is less than or equal to cost of successors.
- Time and space complexity: $O(b^d)$.
    - In the worst-case, this is $O(b^{C^*/\epsilon})$
    - $C^*$ is the cost of the optimal solution and $\epsilon$ is the minimum operator cost.

# Depth-First Search

**function** DF-SEARCH(*problem*)
    **returns** a solution, or failure
  **return** GENERAL-SEARCH(*problem*, ENQUEUE-AT-FRONT)

- Expands deeper nodes first.
- Backtracks if search hits a dead-end.
- Identical to UCS with operator costs of $-1$.

# Evaluation

- Incomplete.
- Not optimal, in general.
- Time complexity: $O(b^m)$, where $m$ is the maximum depth of the tree.
- Space complexity: $O(bm)$. (Why?)

# Depth-Limited Search

- Similar to depth-first search, but imposes a cut-off, $l$, on the maximum depth of a path.
- Complete, provided $l \geq d$.
- Not optimal, in general.
- Time complexity: $O(b^l)$.
- Space complexity: $O(bl)$.

# Iterative Deepening Search

**function** ID-SEARCH(*problem*) **returns** a solution
  **for** *depth* $\longleftarrow$ 0 **to** $\infty$ **do**
    **If** DEPTH-LIMITED-SEARCH(*problem*, *depth*) succeeds
    **then return** its result
  **end**

- Tries all depths, starting with 0.

# Evaluation

- Complete.
- Optimal, provided the path cost is a non-decreasing function in the depth of the node.
- Time complexity: $O(b^d)$.
- Space complexity: $O(bd)$.

# Repeated States

- Remember that we need to avoid repeated states.
- In some problems, no states may possibly be repeated.
  - Typically, when you cannot undo an action (Tic-Tac-Toe, for example).
- Whenever operators are reversible, repeated states are a possibility.
- There are, in general, three strategies to avoid repeating states.

# Repeated States: Strategies

1. Do not return to the parent state.
   - No major overhead.
2. Do not return to an ancestor state.
   - Possibly $O(d)$ time for checking.
3. Do not return to a state that was ever generated.
   - Potentially, a time complexity of $O(b^d)$.
4. Define states and operators wisely.
   - Does not always work.