CS3243 - Introduction to Artificial Intelligence

Tutorial 3

Theodore Leebrant

Tutorial Group 3

Key Concepts

Overview

- **Modelling a search problem**
  - Define state, start/goal states, set of actions, transition model
  - Compute size of state space
- **Uninformed search space**
  - Tracing
  - Completeness
  - Optimality
  - Space and Time Complexity

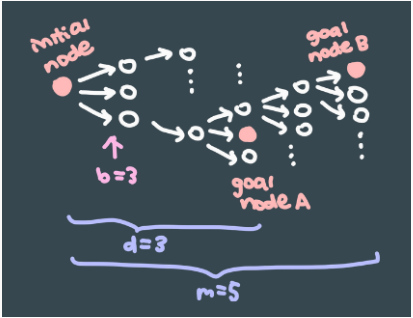
Modelling a Search Problem

- **State**: include essential variables (+ Initial / Goal state(s))
- **Actions**: possible actions that the agent can do
- **Transition model**: description of what each action does (specified by a function that returns a state).
  `RESULT(STATE, ACTION) => RESULT_STATE`
- **Goal test**: determine whether the state is a goal state
- **Cost function**: a function that assigns a cost to each action

Specifying Search Space

The search space is implicitly represented by a graph, with nodes and edges.

- **Branching factor (b)**: how many actions are available at each state.
- **Depth**: Depth of shallowest goal node **(d)**, maximum depth of any path in the state space **(m)**.



Properties of cost

The confusing lecture part - path cost

Usually, $g(n)$ is the current path cost from the initial state to node n (as defined in AIMA)

In lecture, Prof defined:
$g(n)$ as the *optimal* path cost from initial state to n
$\hat{g}(n)$ as the current (minimum) path cost to n
$\hat{g}_{pop}(n)$ as the minimum path to n when we pop

Step cost - $c(s, a, s')$: the cost from s to s' when taking action a.

Uninformed Search Algorithms

All the algorithms you learn are in the family of whatever-first search:

```
put s into data_structure
while data_structure is not empty:
    take node from data_structure
    for each neighbour:
        if goal_test:
            return true
        else:
            put neighbour into data_structure
return false
```

This is a tree-based implementation without memory of searched space.
For DFS the data structure is a stack; BFS: queue; UCS: Priority queue

Uninformed Search Algorithms

```
put s into data_structure
while data_structure is not empty:
    take node from data_structure
    mark node
    for each neighbour:
        if goal_test:
            return true
        else if neighbour unmarked:
            put neighbour into data_structure
return false
```

This is the graph-based implementation.

The term whatever-first search is taken from Jeff Erickson's Algorithm (and is not a term you should use in exam)

Uninformed Search Algorithms

BFS: Expands shallowest node first. Use when you know solution is near root or ree is deep but solutions are rare.

UCS: Expand the least-path-cost unexpanded node (explore cheaper nodes by current path cost first). Equivalent to BFS if all step costs are equal. It is a special case of Dijkstra's.

DFS: Expand the deepest unexpanded node. Use when you don't care how you reach a node, you just want to reach it, and when the solution/goal node is very deep, or when all solutions are at same depth

Completeness and Optimality

- **Complete:** if there is a path from start to the goal node, the algorithm will find it.
- **Optimal:** always able to find the least cost solution. Implies completeness.

BFS: Complete if b is finite, optimal if all step costs identical.
UCS: Complete and optimal if b is finite and all step costs $\geq \varepsilon$ for some $\varepsilon > 0$.