COMP3702/COMP7702
Artificial Intelligence

Module 1: Search

Dr Archie Chapman

Semester 2, 2020

The University of Queensland
School of Information Technology and Electrical Engineering

Thanks to Dr Alina Bialkowski and Dr Hanna Kurniawati

Table of contents

Review

Recall our goal: To build a useful, intelligent agent

To start with:

- Computers perceive the world using sensors.
- Agents maintain models/representations of the world and use them for reasoning.
- Computers can learn from data.

So, to achieve our goal, we need to define our "agent" in a way that we can program it:

- The problem of constructing an agent is usually called the **agent design problem**
- Simply, it's about defining the **components** of the agent, so that when the agent acts rationally, it will accomplish the task it is supposed to perform, and do it well.

Agent design components

The following **components** are required to solve an agent design problem:

- **Action Space** (A): The set of all possible actions the agent can perform.
- **Percept Space** (P): The set of all possible things the agent can perceive.
- **State Space** (S): The set of all possible configurations of the world the agent is operating in.
- **World Dynamics/Transition Function** ($T : S \times A \rightarrow S'$): A function that specifies how the configuration of the world changes when the agent performs actions in it.
- **Perception Function** ($Z : S \rightarrow P$): A function that maps a state to a perception.
- **Utility Function** ($U : S \rightarrow \mathbb{R}$): A function that maps a state (or a sequence of states) to a real number, indicating how desirable it is for the agent to occupy that state/sequence of states.

Graph searching

When to apply search methods? When we have:

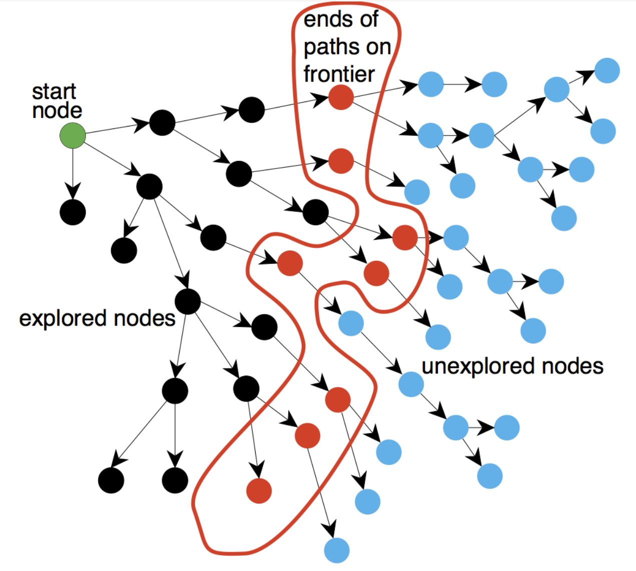
- Sensing uncertainty: **fully observable**
- Effect uncertainty: **deterministic**
- Number of agents: **single agent**

The we can drop the *percept space* and *perception function* from our design considerations.

Generic search algorithm

- Given a graph, start and goal nodes, incrementally explore paths from the start nodes.
- Maintain a **frontier** (fringe) of paths from the start node that have been explored.
- As search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered.
- The way in which the frontier is expanded defines the **search algorithm** or search strategy.

Generic search algorithm (P&M Figure 3.3)



Types of search methods

Two types:

- **Blind search**: Do not use any additional information to “guess” cost of moving from current node to goal
- DFS, BFS, iterative-deepening DFS, and uniform cost search
- **Informed search**: Use additional information to “guess” the cost of moving from current node to goal and decide where to explore next using this information
- Greedy best-first search and A* search

Performance measures for search algorithms

Completeness

- **Complete**: The algorithm will find the solution whenever one exists.

Optimality

- **Optimal**: Return a minimum cost path whenever one exists.

Complexity

- **Time** (#steps) and **space** (memory) complexity
- Complexity analysis informs us of how the required time and memory needed to solve the problem increase as the input size increases
- **Input size**: Size of the state and action spaces of the search problem
- In state graph representation: Size of the graph
- Use computational complexity notation (e.g. Big-O)

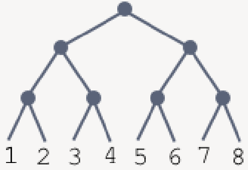
Performance measures for search algorithms

Big-O notation

- Suppose $f(n)$ is the required time/space required to solve the problem if the input size is n
- Then, we say $f(n)$ is of complexity $O(g(n))$ if there is a constant k and n_0 such that:

$$0 \geq f(n) \leq k\, g(n) \quad \text{for all } n_0$$

Branching factor: used to characterise graph topologies



$b = 2 \qquad n = 8$

$b = 8 \qquad n = 8$

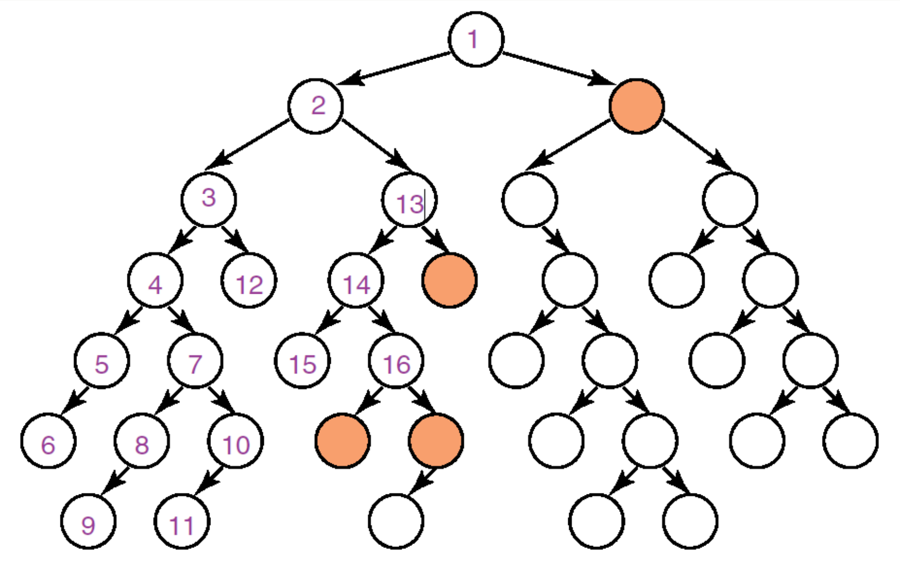
Blind search: DFS and BFS

Depth-first search

Depth-first search

- **Depth-first search** treats the frontier as a stack, or a last-in first-out queue.
- It always selects one of the last elements added to the frontier.
- If the list of paths on the frontier is $[p_1, p_2, \ldots]$
  - p_1 is selected. Paths that extend p_1 are added to the front of the stack (in front of p_2).
  - p_2 is only selected when all paths from p_1 have been explored.

Illustrative Graph — Depth-first Search



Depth-first search: Properties and analysis

Parameters: b, branching factor; m, maximum depth; d, depth of shallowest goal node

Complete? Will DFS find a solution?

- Complete, if m and b are finite and nodes are not revisited

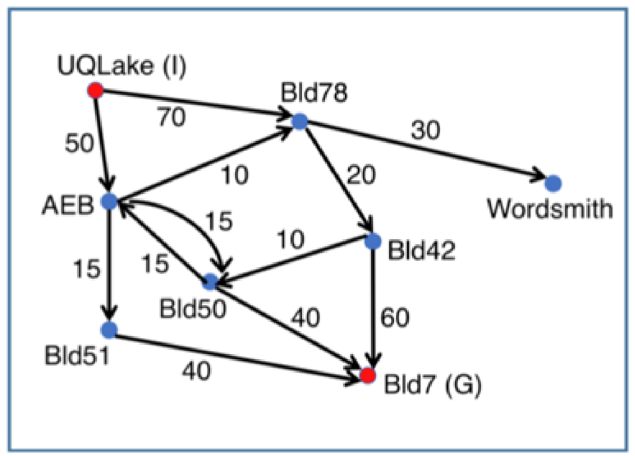
Generate optimal solution? Does DFS guarantee to find the path with fewest edges?

- No

Complexity:

- **Time**: $O(b^m) \quad \Leftarrow \quad 1 + b + b^2 + \ldots + b^m = \frac{b^{m+1-1}}{b-1}$
- **Space**: Can be implemented using $O(bm)$, or $O(m)$ using *backtracking* DFS. But be careful of revisiting vertices (states)!
- **Efficient in use of space**

Example — Navigating UQ



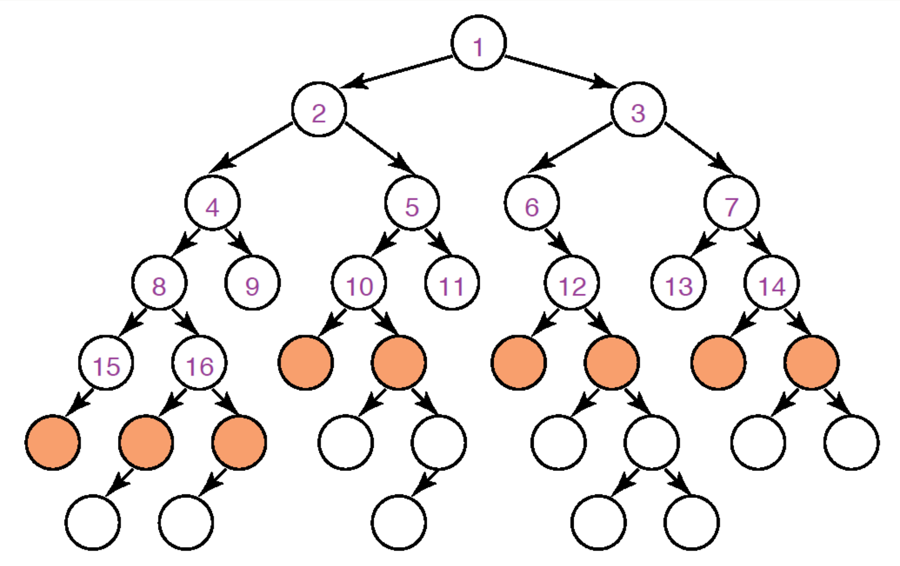
Ignore costs on these edges

Breadth-first search

Breadth-first search

- **Breadth-first search** treats the frontier as a first-in first-out queue.
- It always selects one of the earliest elements added to the frontier.
- If the list of paths on the frontier is $[p_1, p_2, \ldots, p_n]$:
    - p_1 is selected. Its neighbours are added to the end of the queue, after p_n.
    - p_2 is selected next.

Illustrative Graph — Breadth-first Search



Breadth-first search: Properties and analysis

Parameters: b, branching factor; d, depth of shallowest goal node

Complete? Will BFS find a solution?

- Complete if b is finite

Generate optimal solution? Does BFS guarantee to find the path with fewest edges?

- **Yes** in #steps

Complexity:

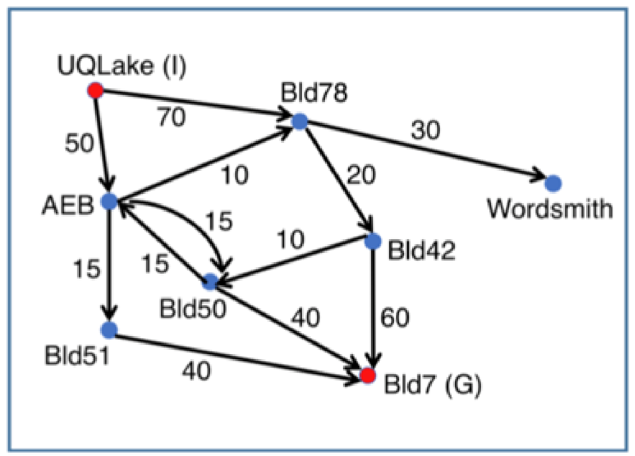
- **Time**: $O(b^d) \quad \Leftarrow \quad 1 + b + b^2 + \ldots + b^d = \frac{b^{d+1-1}}{b-1}$
- **Space**: $O(b^d)$ nodes to remember: $O(b^{d-1})$ explored nodes $+$ $O(b^d)$ unexplored nodes
- **Finds minimum step path, but requires exponential space!**

Let's get some intuition: Practical numbers for BFS

d	# Nodes	Time	Memory
2	110	.11 msec	107 Kbytes
4	11,110	11 msec	10.6 Mbyte
6	$\sim 10^6$	1 sec	1 Gbytes
8	$\sim 10^8$	~2min	103 Gbytes
10	$\sim 10^{10}$	~2.8 hours	10 Tbyte
12	$\sim 10^{12}$	~11.6 days	1 Pbytes
14	$\sim 10^{14}$	~3.2 years	99 Pbytes

Assumptions: b = 10; 1 Kbytes/node; 1million nodes/sec

Example — Navigating UQ



Ignore costs on these edges

Iterative deepening depth-first search

Iterative deepening DFS (IDDFS)

DFS:Efficient in space, but no path length guarantee.

BFS: Finds minimum step path, but requires exponential space.

Iterative deepening DFS

- Multiple DFS with increasing depth-cutoff until the goal is found.
- For $k = 1, 2, \ldots$: Perform DFS with depth cutoff k.
- Only generates nodes with depth $\leq k$.

IDDFS: Properties and analysis

Parameters: b, branching factor; m, maximum depth; d, depth of shallowest goal node

Complete? Will IDDFS find a solution?

- Complete if b is finite

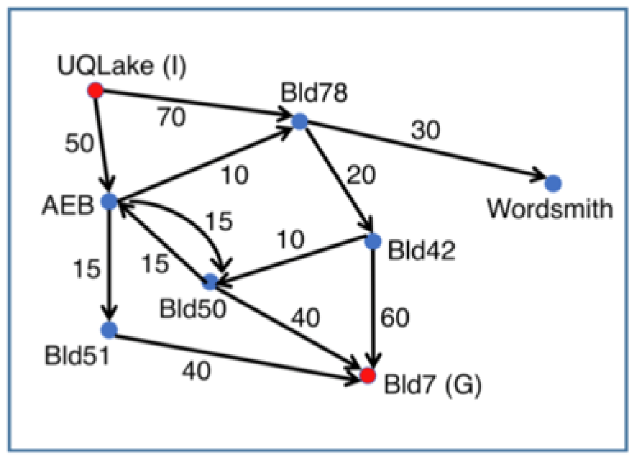
Generate optimal solution? Does IDDFS guarantee to find the path with fewest edges?

- **Yes** in #steps

Complexity:

- **Time**: $O(b^d) \quad \Leftarrow \quad db + (d-1)b^2 + \ldots + (1)b^d$
- **Space**: $O(bd)$
- **Finds minimum step path, and doesn't require exponential space!**

Example — Navigating UQ



Ignore costs on these edges

Search with edge costs: Uniform cost search

Search with edge costs: Uniform cost search

- Sometimes there are costs associated with edges.
- The **cost** of a path is the sum of the costs of its edges:

$$cost(n_0, \ldots, n_k) = \sum_{i=1}^{k} cost(n_{i-1}, n_i)$$

- An optimal solution is one with minimum cost.

Uniform cost search

- At each stage, uniform-cost search selects a path on the frontier with lowest cost.
- The first path to a goal is a least-cost path to a goal node.
- When edge costs are equal $\Rightarrow$ breadth-first search.

Uniform cost Search

- **Uniform cost search** treats the frontier as a **priority queue** ordered by path cost.
- It always selects one of the highest-priority vertices added to the frontier.
- If the list of paths on the frontier is $[p_1, p_2, \ldots]$:
  - p_1 is selected to be expanded.
  - Its successors are inserted into the priority queue.
  - The highest-priority vertex is selected next (and it might be a newly expanded vertex).

UCS: Properties and analysis

Parameters: b, branching factor; m, maximum depth; d, depth of shallowest goal node

$C*$: Cost of optimal solution, ϵ: minimum cost of a step

Complete? Will UCS find a solution?

- Complete if b is finite and all edges have a cost $\geq \epsilon > 0$

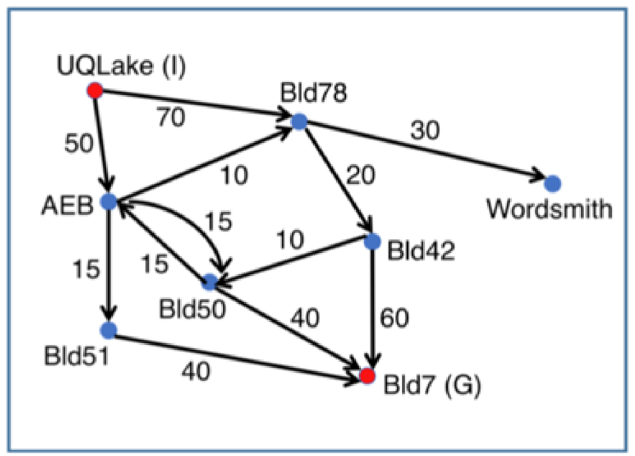
Generate optimal solution? Does UCS guarantee to find the path with **the lowest cost**?

- **Yes** if all edges have positive cost

Complexity:

- **Time** and **Space**: $O(b^{1+\lfloor \frac{C*}{\epsilon} \rfloor})$

Example — Navigating UQ



Blind search: Summary

- Depth-first search
- Breadth-first search
- Iterative-deepening depth-first search
- Uniform cost search

Informed search

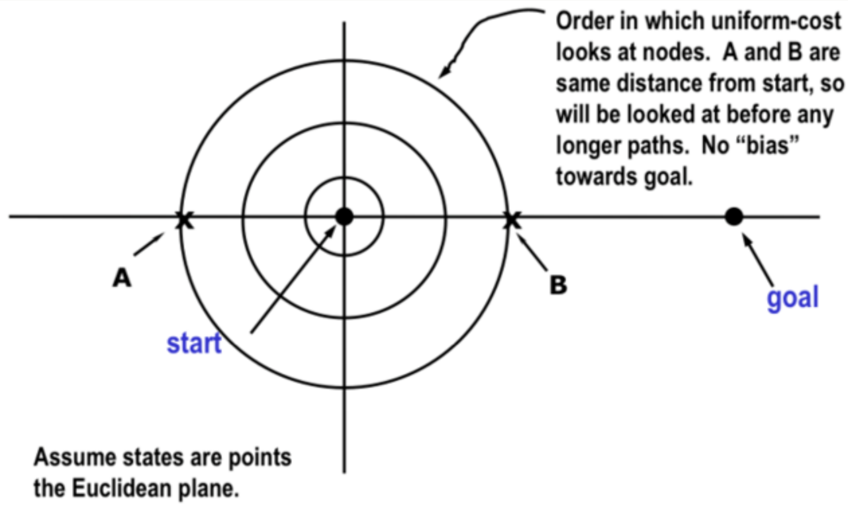
Blind vs informed search

- Blind search algorithms (e.g. UCS) use the cost from the root to the current node, $g(n)$ to prioritise which nodes to search.
- Informed search algorithms rely on **heuristics**, $h(n)$ that give an estimated cost from the current node to the **goal** to prioritise search.
- In general, informed search is faster than blind search
- However, it is more difficult to prove properties of informed search algorithms, as their performance highly depends on the heuristics used

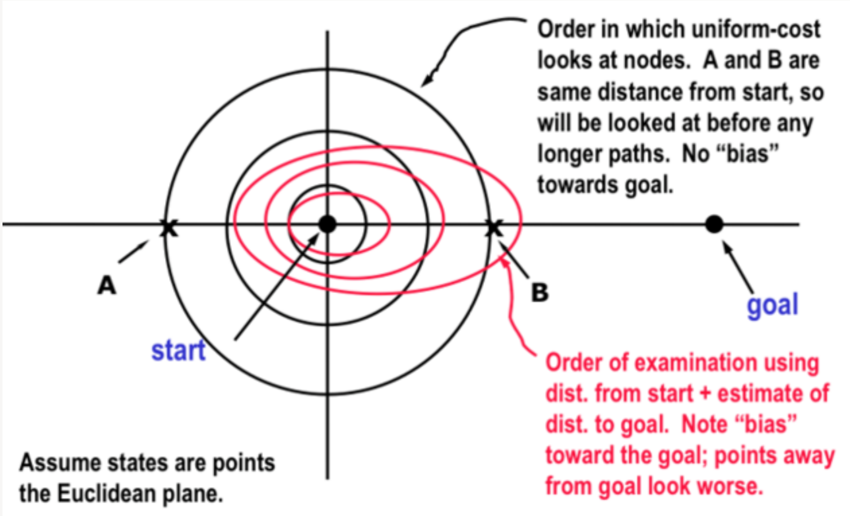
Informed search algorithms

- Greedy best-first search
- A* search

Blind vs informed search



Blind vs informed search



Informed search using heuristics

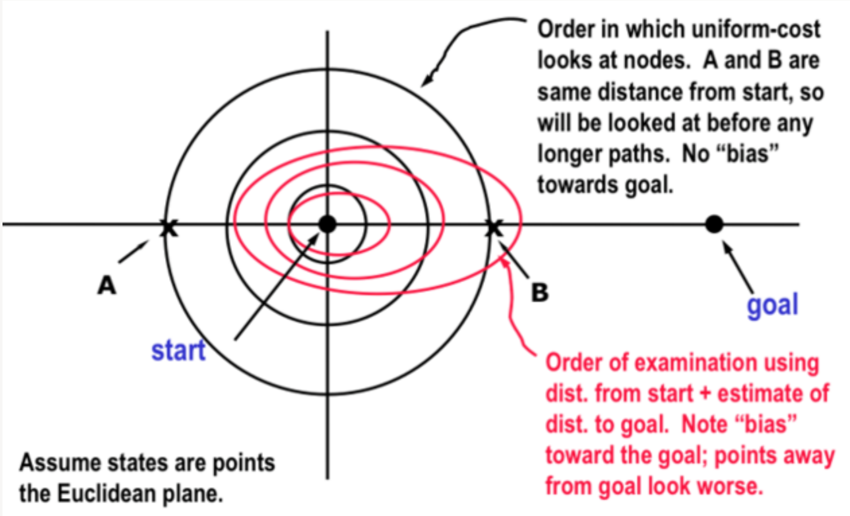
Idea: Don't ignore the goal when selecting paths.

- Often there is extra knowledge that can be used to guide the search: **heuristics.**
- $h(n)$ is an estimate of the cost of the shortest path from node n to a goal node.
- $h(n)$ needs to be efficient to compute.
- h can be extended to paths: $h(n_0, \ldots, n_k) = h(n_k)$.
- $h(n)$ is an **underestimate** if there is no path from n to a goal with cost less than $h(n)$.
- An **admissible heuristic** is a nonnegative (≥ 0) heuristic function that is an underestimate of the actual cost of a path to a goal.

Example heuristic functions

- If the nodes are points on a Euclidean plane and the cost is the distance, $h(n)$ can be the straight-line distance from n to the closest goal (i.e. ignoring obstacles).
- If the nodes are locations and cost is *time*, we can use the distance to a goal divided by the maximum speed (underestimate).
- If the goal is complicated, simple decision rules that return an approximate solution and that are easy to compute can make for good heuristics
- A heuristic function can be found by solving a simpler (less constrained) version of the problem.

Blind vs informed search



Greedy best-first search

Greedy best-first search

Greedy Best-First search is *almost* the same as UCS, with some key differences:

- Uses a **priority queue** to order expansion of fringe nodes
- The highest priority in priority queue for greedy best-first search is the node with the smallest **estimated cost** from the current node to the goal
- The estimated cost-to-goal is given by the heuristic function, $h(n)$
- If the list of paths on the frontier is $[p_1, p_2, \ldots]$:
  - p_1 is selected to be expanded.
  - Its successors are inserted into the priority queue.
  - The highest-priority vertex is selected next (and it might be a newly expanded vertex).

Greedy best-first search: Properties and analysis

Complete? Will greedy best-first search find a solution?

- **No** (it depends on the heurisitc)

Generate optimal solution? Is greedy best-first search guaranteed to find the lowest cost path to the goal?

- **No**

Complexity:

- Depends highly on the heuristic
- Worst case if the tree depth is finite: $O(b^m)$ where b is branching factor and m is maximum depth of the tree (i.e. it can be exponentially bad).

Example — Navigating UQ

Heuristic values (to Bld7)

h(UQLake) = 100

h(Bld78) = 50

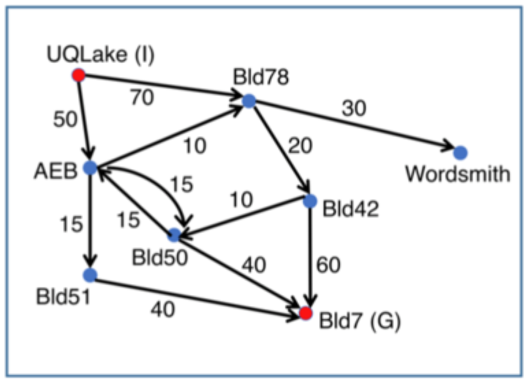
h(AEB) = 53

h(Wordsmith) = 1000 h(Bld42) = 50

h(Bld50) = 38

h(Bld51) = 30

h(Bld7) = 0



A* Search

A* Search

A* search uses both path cost and heuristic values

- It is a mix of uniform-cost and best-first search
- $g(p)$ is the cost of path p from initial state to a node (UCS)
- $h(p)$ estimates the cost from the end of p to a goal (GBFS)
- A* uses: $f(p) = g(p) + h(p)$

$f(p)$ estimates the **total** path cost of going from a start node to a goal via p

$$\underbrace{\underbrace{\mathit{start} \xrightarrow{\text{path } p} n}_{g(p)} \; \underbrace{\xrightarrow{\text{estimate}} \mathit{goal}}_{h(p)}}_{f(p)}$$

A* Search Algorithm

- A* is a mix of uniform-cost and best-first search, algorithmically similar to both
- It treats the frontier as a **priority queue** ordered by $f(p)$
- Highest priority is the node with the lowest f value The function $f(p)$ is the the shortest path length from root to the node ($g(p)$) plus the estimated future reward from node p to the goal ($h(p)$)
- It always selects the node on the frontier with the lowest estimated distance from the start to a goal node constrained to go via that node

Example — Navigating UQ

Heuristic values (to Bld7)

h(UQLake) = 100

h(Bld78) = 50

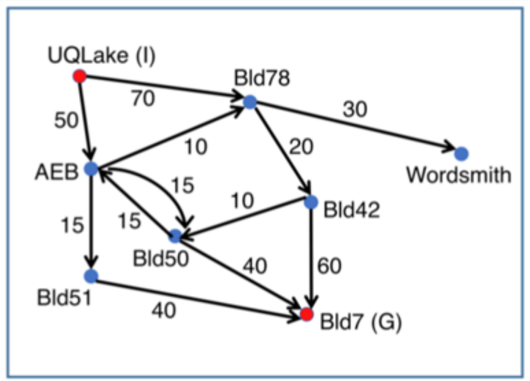
h(AEB) = 53

h(Wordsmith) = 1000 h(Bld42) = 50

h(Bld50) = 38

h(Bld51) = 30

h(Bld7) = 0



A* Search: Properties and analysis

- Will A* search find a solution?
- Is A* search guaranteed to find the shortest path or the path with fewest arcs?
- What is the time complexity as a function of length of the path selected?
- What is the space complexity as a function of length of the path selected?
- How does the goal affect the search?

Admissibility of A*

If there is a solution, A* always finds an optimal solution —as the first path to a goal selected— if the following conditions are met:

1. the search graph branching factor b is finite
2. edge costs are bounded above zero (there is some $\epsilon > 0$ such that all of the edge costs are greater than ϵ), and
3. $h(n)$ is > 0 and an underestimate of the cost of the shortest path from n to a goal node.

. . . *we have seen 2 and 3 before:*

A heuristic is admissible if it never overestimates the cost-to-goal

Why is A* admissible?

If a path p to a goal is selected from a frontier, can there be a shorter path to a goal?

- Suppose path p' is on the frontier.
- Because p was chosen before p', and $h(p) = 0$:

$$g(p) \le g(p') + h(p').$$

- Because h is an underestimate:

$$g(p') + h(p') \le g(p'')$$

  for any path p'' to a goal that extends p'.

So $g(p) \le g(p'')$ for any other path p'' to a goal.

How do good heuristics help?

Suppose c is the cost of an optimal solution. What happens to a path p where

- $g(p) + h(p) < c$
- $g(p) + h(p) = c$
- $g(p) + h(p) > c$

How can a better heuristic function help?

Example — Navigating UQ

Heuristic values (to Bld7)

$h(\text{UQLake}) = 100$

$h(\text{Bld78}) = 50$

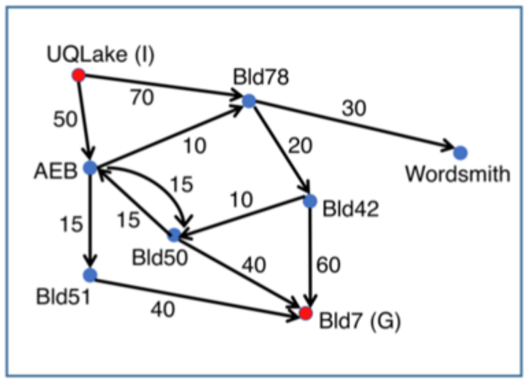
$h(\text{AEB}) = 53$

$h(\text{Wordsmith}) = 1000$ $h(\text{Bld42}) = 50$

$h(\text{Bld50}) = 38$

$h(\text{Bld51}) = 30$

$h(\text{Bld7}) = 0$



Attributions and References

Thanks to Dr Alina Bialkowski and Dr Hanna Kurniawati for their materials.

Many of the frames in Module 1 are adapted from David Poole and Alan Mackworth, *Artificial Intelligence: foundations of computational agents*, 2E, CUP, 2017 `http://https://artint.info/`.

Other materials derived from Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, 3E, Prentice Hall, 2009.

All remaining errors are Archie's — please email if you find any: archie.chapman@uq.edu.au