INFO8006 Introduction to Artificial Intelligence

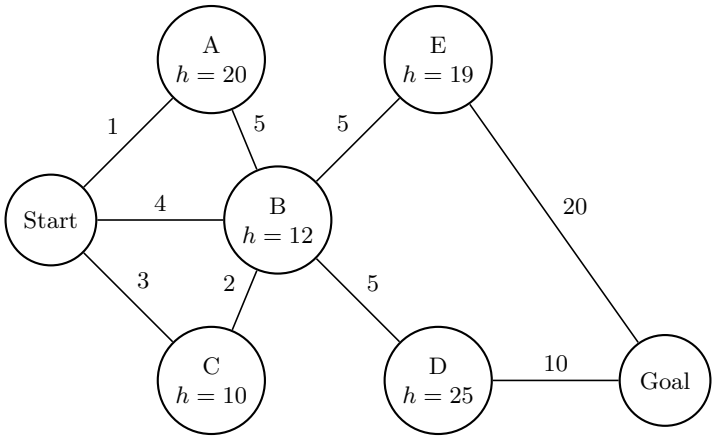
Exercises 1: Solving problems by searching

Learning outcomes

At the end of this exercise session you should be able to:

- formulate search problems rigorously.
- theoretically analyse the algorithms to perform uninformed search (depth-first, breadth-first, uniform-cost) and informed search (greedy-search, A*).
- apply each of these algorithms on search problems defined in fully observable and deterministic environments.

Exercise 1: Search algorithms



For each of the following search algorithms, give the order in which states are expanded as well as the final path returned by the algorithm. If two nodes are in competition to be expanded, the conflict is resolved by alphabetical order.
Here follows a search algorithm which is valid for non-consistent (but admissible) heuristics.

Algorithm 1 Graph-Search Algorithm

Input: fringe: The fringe, which is a priority queue for nodes, **initialised with the starting node**.
closed: A list of closed nodes, **initially empty**.
Output: p: A list of nodes which represents the optimal path.

```
1:  procedure GRAPH-SEARCH(fringe, closed)
2:      while True do
3:          n ← fringe.pop()
4:          if n is nil then
5:              return nil
6:          if n.isGoal() then
7:              return n.getPath()
8:          closed.append(n)
9:          for n' ∈ n.children do
10:             p ← n'.computePriority(n)
11:             if n' ∉ closed  &&  n' ∉ fringe then
12:                 n'.setPriority(p)
13:                 n'.setParent(n)
14:                 fringe.push(n')
15:             else if n' ∈ fringe  &&  p < fringe.get(n').getPriority() then
16:                 fringe.remove(n')
17:                 fringe.push(n')
18:             else if n' ∈ closed  &&  p < closed.get(n').getPriority() then
19:                 closed.remove(n')
20:                 fringe.push(n')
```

1. Depth-First search
   Expansion: S, A, B, D, Goal
   Path: S → A → B → D → G
   We use a LIFO heap to perform the search. In order to prevent cycles, when we add a node into the heap we replace its previous version if there is one and we never add a node that has already been expanded.
   Step-by-step:

   - Heap = {S()}; Expand(S); Visited = {S()}
   - Heap = {A(S), B(S), C(S)}; Expand(A(S)); Visited = {S(), A(S)}
   - Heap = {B(A), C(S)}; Expand(B(A)); Visited = {S(), A(S), B(A)}
   - Heap = {C(B), D(B), E(B)}; Expand(C(B)); Visited = {S(), A(S), B(A), C(B)}
   - Heap = {D(B), E(B)}; Expand(D(B)); Visited = {S(), A(S), B(A), C(B), D(B)}
   - Heap = {Goal(D), E(B)}; Expand(Goal(D)); Visited = {S(), A(S), B(A), C(B), D(B), Goal(D)}

2. Breadth-First search
   Expansion: S, A, B, C, D, E, Goal
   Path: S → B → D → Goal
   We use FIFO queue to perform the search. In order to prevent cycles we never add a node that is already in the queue or has already been expanded.
   Step-by-step:

   - Queue = {S()}; Expand(S()); Visited = S
   - Queue = {A(S), B(S), C(S)}; Expand(A(S)); Visited = {S(), A(S)}
   - Queue = {B(S), C(S)}; Expand(B(S)); Visited = {S(), A(S), B(S)}
   - Queue = {C(S), D(B), E(B)}; Expand(C(S)); Visited = {S(), A(S), B(S), C(S)}
   - Queue = {D(B), E(B)}; Expand(D(B)); Visited = {S(), A(S), B(S), C(S), D(B)}
   - Queue = {E(B), Goal}; Expand(E(B)); Visited = {S(), A(S), B(S), C(S), D(B), E(B)}
   - Queue = {Goal(D)}; Expand(Goal(D)); Visited = {S(), A(S), B(S), C(S), D(B), E(B), Goal(D)}

3. Uniform-Cost search
   We use the algorithm presented above to perform the search. The priority of a node is equal to its total path cost (g(N)).
   Expansion: S, A, C, B, D, E, Goal
   Path: S → B → D → Goal
   Step-by-step, in the following we will denote a node N as N(g(N), Parent(N)):

   - fringe = {S(0, None)}; Expand(S); closed = {S(0, None)}
   - fringe = {A(1, S), C(3, S), B(4, S)}; Expand(A); closed = {S(0, None), A(1, S)}
   - fringe = {C(3, S), B(4, S)}; Expand(C); closed = {S(0, None), A(1, S), C(3, S)}
   - fringe = {B(4, S)}; Expand(B); closed = {S(0, None), A(1, S), C(3, S), B(4, S)}
   - fringe = {D(9, B), E(9, B)}; Expand(D); closed = {S(0, None), A(1, S), C(3, S), B(4, S), D(9, B)}
   - fringe = {E(9, B), Goal(19, D)}; Expand(E); closed = {S(0, None), A(1, S), C(3, S), B(4, S), D(9, B), E(9, B)}
   - fringe = {Goal(19, D)}; Expand(Goal); closed = {S(0, None), A(1, S), C(3, S), B(4, S), D(9, B), E(9, B), Goal(19, D)}

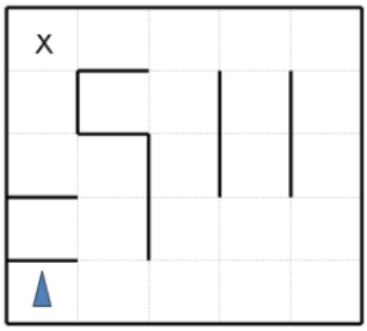
4. Greedy search
   We use the algorithm presented above to perform the search. The priority of a node is equal to the heuristic cost (h(N)).
   Expansion: S, C, B, E, Goal
   Path: S → B → E → Goal
   Step-by-step, in the following we will denote a node N as N(g(N), Parent(N)):

   - fringe = {S(inf, None)}; Expand(S); closed = {S(inf, None)}
   - fringe = {C(10, S), B(12, S), A(20, S)}; Expand(C); closed = {S(inf, None), C(10, S)}
   - fringe = {B(12, S), A(20, S)}; Expand(B); closed = {S(inf, None), C(10, S), B(12, S)}
   - fringe = {E(19, B), A(20, S), D(25, B)}; Expand(E); closed = {S(inf, None), C(10, S), B(12, S), E(19, B)}
   - fringe = {Goal(0, E), A(20, S), D(25, B)}; Expand(Goal); closed = {S(inf, None), C(10, S), B(12, S), E(19, B), Goal(0, E)}

5. A* (Is the heuristic admissible?)
   The heuristic is not admissible, by changing the heuristic at D by $h = 9$ is enough to make it so.
   Expansion: S, C, B, D, Goal
   Path: S $\rightarrow$ B $\rightarrow$ D $\rightarrow$ Goal
   Step-by-step, in the following we will denote a node N as N(g(N), Parent(N)):
   - fringe = {S(inf, None)}; Expand(S); closed = {S}
   - fringe = {C(13, S), B(16, S), A(21, S)}; Expand(C); closed = {S, C(13, S)}
   - fringe = {B(16, S), A(21, S)}; Expand(B); closed = {S, C(13, S), B(16, S)}
   - fringe = {D(18, B), E(28, B), A(21, S)}; Expand(D); closed = {S, C(13, S), B(12, S), D(18, B)}
   - fringe = {Goal(19, D), E(28, B), A(21, S)}; Expand(Goal); closed = {S, C(13, S), B(12, S), D(18, B), Goal(19, D)}

Exercise 2: Maze Car

This exercice is taken from CS188 Spring 2014.
Consider a car event which has to exit the following maze:



At all timesteps, the agent points at direction $d \in$ (N, S, E, W). With a single action, the agent can either move forward at an adjustable velocity v or turn. The turning actions are left and right, which change the agent's direction by 90 degrees. Turning is only permitted when the velocity is zero (and leaves it at zero). The moving actions are fast and slow. Fast increments the velocity by 1 and slow decrements the velocity by 1; in both cases the agent then moves a number of squares equal to its *new* adjusted velocity. Any action that would result in a collision with a wall crashes the agent and is illegal. Any action that would reduce v below 0 or above a maximum speed V_{max} is also illegal. The agent's goal is to find a plan which parks it (stationary) on the exit square using as few actions (time steps) as possible.

As an example: if the shown agent is initially stationary, it might first turn to the east using (right), then move one square east using fast, then two more squares east using fast again. The agent will of course have to slow to turn.

- Quiz:
  1. For a grid of size $M \times N$, what is the size of the state space? Assume that all configurations are reachable from the starting state.
     $N \times M \times 4 \times (V_{max} + 1)$
  2. What is the maximum branching factor of this problem? Assume that illegal actions are simply not returned by the successor function.
     The maximum branching factor is 3, and this happens when the agent is stationary. While stationary it can take the following 3 actions - fast, left, right.
  3. Is the Manhattan distance from the agent's location to the exit's location admissible?
     No because it could overestimate the true cost in the case where the average car speed is greater than 1.
  4. If we used an inadmissible heuristic in A* tree search, could it change the completeness of the search?
     No, A star will still find a final state if there exists a reachable one.
  5. If we used an inadmissible heuristic in A* tree search, could it change the optimality of the search?
     Yes, we could stop the search whereas there exists a better solution.
- Discussion:
  1. State and motivate a non-trivial admissible heuristic for this problem.
     Some examples of admissible heuristics are:
     - The Manhattan distance divided by the speed max.

– We can improve the above relaxation by accounting for the deceleration dynamics. In this case the agent will have to slow down to 0 when it is about to reach the goal. Note that this heuristic will always return a greater value than the previous one, but is still not an overestimate of the true cost to reach the goal. We can say that this heuristic dominates the previous one

2. Give a general advantage that an inadmissible heuristic might have over an admissible one.
The time to solve an A* tree search problem is a function of two factors: the number of nodes expanded, and the time spent per node. An inadmissible heuristic may be faster to compute, leading to a solution that is obtained faster due to less time spent per node. It can also be a closer estimate to the actual cost function (even though at times it will overestimate!), thus expanding less nodes. We lose the guarantee of optimality by using an inadmissible heuristic. But sometimes we may be okay with finding a suboptimal solution to a search problem.

Exercise 3: Heuristic ⋆

Consider a search problem where all edges have a unit cost and the optimal solution has cost C. Let h be a heuristic which is $\max\{h^\star - k, 0\}$, where $h^\star$ is the actual cost to the closest goal and k is a non-negative constant.

1. Is h admissible?
Yes, indeed if $h^\star \geq k : h^\star - k \leq h^\star$ else $0 \leq h^\star$.

2. Which of the following is the most reasonable description of how much more work will be done (= how many more nodes will be expanded) with heuristic h compared to $h^\star$, as a function of k?

   (a) Constant in k

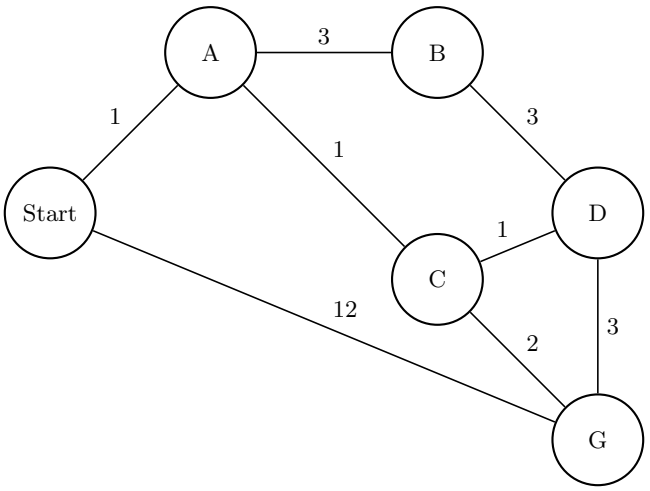
   (b) Linear in k

   (c) Exponential in k

   (d) Unbounded

   Exponential in k. At k = 0, only the d nodes on an optimal path to the closest goal are expanded for search depth (= optimal path length) d. At k = max(h), the problem reduces to uninformed search and BFS expands $b \times d$ nodes for branching factor b. In general, all nodes within distance k of the closest goal will have heuristic h = 0 and uninformed search may expand them. Note that search reduces to BFS since A* with h = 0 is UCS and in this search problem all edges have cost 1 so path cost = path length

Exercise 4: Search algorithms ⋆



For each of the following search algorithms, give the order in which states are expanded as well as the final path returned by the algorithm. If two nodes are in competition to be expanded, the conflict is resolved by alphabetical order.

1. Depth-First search
Expansion: S-A-B-D-C-G
Path: S→A→B→D→C→G

2. Breadth-First search
Expansion: S-A-G
Path: S→G

3. Uniform-Cost search
Expansion: S-A-C-D-B-G
Path: S→A→C→G

State	h_1	h_2
S	5	4
A	3	2
B	6	6
C	2	1
D	3	3
G	0	0

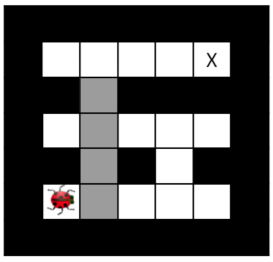
Table 1: 2 possible heuristics

4. Consider the following heuristics: which one is not admissible? Why?
   h1 is not admissible because the real cost from S to G is 4 which is smaller than 5.

5. Use the admissible heuristic to apply A* algorithm.
   Expansion: S-A-C-G Path: S→A→C→G

Exercise 5: The hive ⋆

The hive of insects needs your help. You control an insect in a rectangular maze-like environment of size $M \times N$, as shown on the Figure below. At each time-step, the insect can move into a free adjacent cell or stay in its current location. All actions have a unit cost.
In this particular case, the insect must pass through a series of partially flooded tunnels, as illustrated by the grey cells on the map. The insect can hold its breath for A time-steps in a row. Moving into a flooded cell requires your insect to consume 1 unit of air, while moving into a free cell refills its air supply.



- Give a minimal state space for this problem (i.e., do not include extra information). You should answer for a general instance of the problem, not the specific map above.
  The position of the insect as well as the number of unit of air it has left. $s = (x, y, u_a) \in \{1, \dots, M\} \times \{1, \dots N\} \times \{1, \dots, A\}$.

- Give the size of your state space.
  $M \times N \times A$

Supplementary materials

Berkeley 1
Berkeley 2
Chapter 3 of reference book.