Introduction to Artificial Intelligence (INFO8006)

Exercises 1 – Solving problems by searching

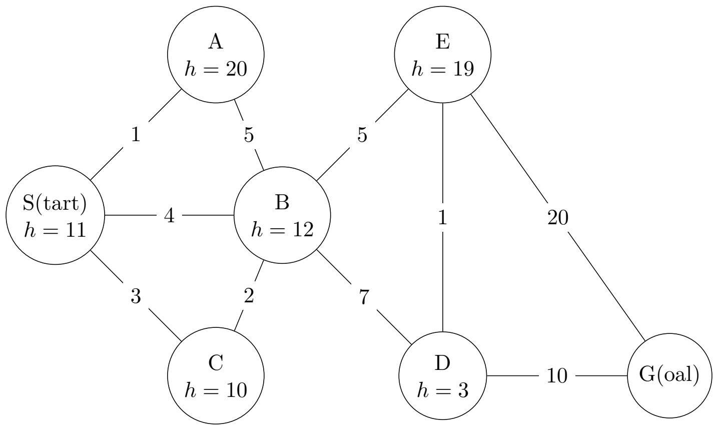
September 16, 2022

Learning outcomes

At the end of this session you should be able to

- formulate search problems rigorously;
- analyze the algorithms to perform uninformed search (depth-first, breadth-first, uniform-cost) and informed search (greedy, A*);
- apply these algorithms in fully observable and deterministic environments.

Exercise 1 Search algorithms



For each of the following search algorithms, apply the Graph-Search algorithm (lecture 2, slide 62) and give the order in which the states, represented by the nodes of this undirected graph, 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 (DFS)

   We use a Last-In First-Out (LIFO) fringe, *i.e.* a *stack*, to perform the search. In this stack, we keep track of the states that are reachable from the visited (closed) states. These states are annotated with the state they were expanded from, which allows to reconstruct the path at the end. At each step, the state at the *top* of the stack is *popped.* If it was not visited previously, it is expanded, which consists in *pushing* its neighbors to the top of the stack.

Fringe (LIFO stack)	Closed	Expand
S		S
C(S), B(S), A(S)	S	A(S)
C(S), B(S), B(A)	S, A	B(A)
C(S), E(B), D(B), C(B)	S, A, B	C(B)
E(B), D(B)	S, A, B, C	D(B)
E(B), G(D), E(D)	S, A, B, C, D	E(D)
G(D), G(E)	S, A, B, C, D, E	G(E)

For readability, we don't write the nodes currently in the fringe that are already in the closed set, since they will never be expanded.

Expansion: S, A, B, C, D, E, G. Path: S, A, B, D, E, G.

2. Breadth-First Search (BFS)

   We use a First-In First-Out (FIFO) fringe, *i.e.* a *queue*, to keep track of the reachable states. At each step, the state at the *front* of the queue is popped. If it was not visited previously, it is expanded, which consists in pushing its neighbors to the *back* of the queue.

Fringe (FIFO queue)	Closed	Expand
S		S
C(S), B(S), A(S)	S	A(S)
B(A), C(S), B(S)	S, A	B(S)
E(B), D(B), C(B), C(S)	S, A, B	C(S)
E(B), D(B)	S, A, B, C	D(B)
G(D), E(D), E(B)	S, A, B, C, D	E(B)
G(E), G(D)	S, A, B, C, D, E	G(D)

   Expansion: S, A, B, C, D, E, G. Path: S, B, D, G.

3. Uniform-Cost search (UCS)

   The fringe is a *priority* queue. The priority of a node n in the queue is its cumulative cost $g(n)$.

Fringe (priority queue)	Closed	Expand
S(0)		S(0)
B(S, 4), C(S, 3), A(S, 1)	S	A(S, 1)
B(A, 6), B(S, 4), C(S, 3)	S, A	C(S, 3)
B(A, 6), B(C, 5), B(S, 4)	S, A, C	B(S, 4)
D(B, 11), E(B, 9)	S, A, C, B	E(B, 9)
D(B, 11), D(E, 10)	S, A, C, B, E	D(E, 10)
G(D, 20)	S, A, C, B, E, D	G(D, 20)

   Expansion: S, A, C, B, E, D, G. Path: S, B, E, D, G.

4. Greedy search

   The fringe is also a *priority* queue, but the priority of a node n in the queue is its *heuristic* cost $h(n)$.

Fringe (priority queue)	Closed	Expand
S(11)		S(11)
A(S, 20), B(S, 12), C(S, 10)	S	C(S, 10)
A(S, 20), B(C, 12), B(S, 12)	S, C	B(S, 12)
A(S, 20), E(B, 19), D(B, 3)	S, C, B	D(B, 3)
A(S, 20), E(D, 19), E(B, 19), G(D, 0)	S, C, B, D	G(D, 0)

   Expansion: S, C, B, D, G. Path: S, B, D, G.

5. A*

The priority of a node n is the *estimated* cost $f(n) = g(n) + h(n)$ of the cheapest solution through n.

(a) Is the heuristic admissible ? If not, make it admissible.

A heuristic h is admissible if $0 \leq h(n) \leq h^*(n)$, where $h^*(n)$ is the true forward cost of n. In the graph, $h(\mathrm{E}) = 19$ is larger than $h^*(\mathrm{E}) = 11$, making h non-admissible. This is problematic as it means E (and G) would be expanded before D.

Setting $h(\mathrm{E}) = 11$ is sufficient to make h admissible.

(b) Is the heuristic consistent ? If not, can we apply Graph-Search ?

In a graph, a heuristic h is consistent if $h(n) \leq c(n, n') + h(n')$ for all node n' reachable from n, where $c(n, n')$ is the step cost from n to n'. Because this graph is undirected, unless $c(n, n') \geq |h(n') - h(n)|$ for all pair of neighbors (n, n') (which is not the case), it cannot be consistent.

This is problematic as it could create situations where a node has been expanded within a path that is not optimal, and cannot be re-expanded later on within the optimal one. Hence, the algorithm loses its optimality.

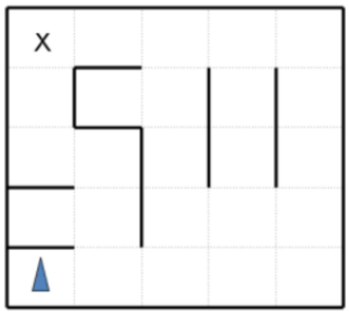
Fortunately, there is a modification of Graph-Search that can handle inconsistent heuristics. If we annotate the nodes in the closed set with the priority at which they were expanded, we obtain the capacity to re-expand a node if it has a better (lower) priority than before. The priority in the closed set is then updated.

We apply the modified algorithm with the admissible heuristic.

Fringe (priority queue)	Closed	Expand
S(11)		S(11)
A(S, 21), B(S, 16), C(S, 13)	S(11)	C(S, 13)
A(S, 21), B(C, 17), B(S, 16)	S(11), C(13)	B(S, 16)
A(B, 29), A(S, 21), E(B, 20), D(B, 14)	S(11), C(13), B(16)	D(B, 14)
A(B, 29), E(D, 23), G(D, 21), A(S, 21), E(B, 20)	S(11), C(13), B(16), D(14)	E(B, 20)
A(B, 29), G(D, 21), A(S, 21), D(E, 13)	S(11), C(13), B(16), D(14), E(20)	D(E, 13)
A(B, 29), G(D, 21), A(S, 21), G(D, 20)	S(11), C(13), B(16), D(13), E(20)	G(D, 20)

Expansion: S, C, B, D, E, D, G. Path: S, B, E, D, G.

Exercise 2 Maze Car (UC Berkeley CS188, Spring 2014)



Consider a car agent which has to exit a maze. At all time-steps, 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 $v \in [0, V]$ 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 `faster` and `slower`. The action `faster` increments the velocity by 1 and `slower` decrements the velocity by 1; in both cases the agent then moves a number of squares equal to its *new* velocity. Any action that would result in a collision with a wall is illegal. Any action that would reduce v below 0 or above a maximum speed V 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, in the hereabove maze, if the agent is initially stationary, it might first turn to the east using `right`, then move one square east using `faster`, then two more squares east using `faster` again. However, the agent will have to slow to take the turn.

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.

   $N \times M \times 4 \times (V + 1)$

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: `faster`, `left`, `right`.

3. Is the Manhattan distance from the agent's location to the exit's location admissible?

   No because it does not take the velocity of the agent into account. For instance, in a long straight path, the number of steps could be smaller than the number of squares.

4. If we used an inadmissible heuristic in A* tree search, could it change the completeness of the search? And the optimality ?

   With an inadmissible heuristic, A* will still find a final state if there exists a reachable one (complete), but it could stop the search before finding the optimal one (non optimal).

5. State and motivate a non-trivial admissible heuristic for this problem.

   Some examples of admissible heuristics are:

   - The Manhattan distance divided by the max speed V.
   - We can improve the above heuristic by adding the current velocity v, as the agent has to slow down to 0 before reaching its goal. This heuristic *dominates* the first one as it

is always larger (or equal), but is still admissible as it never overestimates the true forward cost.

6. 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.
   - Admissible heuristics are sometimes too conservative (well below h^*). An inadmissible heuristic could be a better estimate of the true forward cost h^* (although it will overestimate at times), thus expanding less nodes.

   We lose the guarantee of optimality by using an inadmissible heuristic. However, sometimes, we may be okay with finding a suboptimal solution faster.

Exercise 3 Heuristics (UC Berkeley CS188, Spring 2019)

Consider a graph search problem where all edges have a unit cost and the optimal solution has cost C^*. Let $h(n)$ be a heuristic which is $\max\{h^*(n) - k, 0\}$, where $h^*(n)$ is the true forward cost of n and $k \leq C^*$ is a non-negative constant.

1. Is h admissible?

   Yes. If $h^* \geq k$, $h^* - k \leq h^*$, otherwise $0 \leq h^*$.

2. Which of the following is the most reasonable description of how much more work will be done, *i.e.* how many more nodes will be expanded, with heuristic h compared to h^*, as a function of k?

   (a) Constant in k

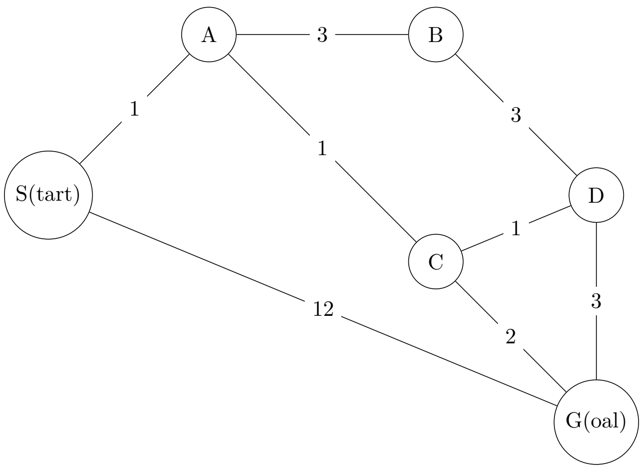
   (b) Linear in k

   (c) Exponential in k

   (d) Unbounded

   With this heuristic, all nodes n such that $h^*(n) \leq k$, *i.e.* within distance k from the objective, will have have a null heuristic. Assuming a node can have up to p parents, there is at most p^k such node and the algorithm will expand all of them, in the worst case. Thus, the additional work is exponential in k.

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 (DFS)

   Expansion: S, A, B, D, C, G. Path: S, A, B, D, C, G.

2. Breadth-First Search (BFS)

   Expansion: S, A, G. Path: S, G.

3. Uniform-Cost search (UCS)

   Expansion: S, A, C, D, B, G. Path: S, A, C, G.

4. Consider the following heuristics: which one is not admissible? Why?

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

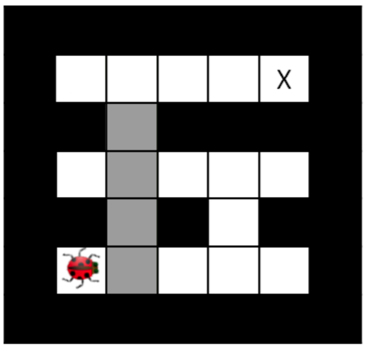
   The heuristic h_1 is not admissible because the real cost from S to G is 4 which is smaller than 5.

5. A* (with the admissible heuristic)

   Expansion: S, A, C, G. Path: S, A, C, G.

Exercise 5 The hive (UC Berkeley CS188, Spring 2019)

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 gray 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 (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. $s = (x, y, u_a) \in [1, M] \times [1, N] \times [1, A]$.

- Give the size of your state space.

  $M \times N \times A$

Supplementary materials

- Search (UC Berkeley CS188, Spring 2014 Section 0).
- Search (UC Berkeley CS188, Spring 2014 Section 1).
- Chapter 3 of the reference textbook.