CS221 Problem Workout Solutions

Week 3

1) Problem 1

Sabina has just moved to a new town, which is represented as a grid of locations (see below). She needs to visit various shops $S_1, \ldots, S_k$. From a location on the grid, Sabina can move to the location that is immediately north, south, east, or west, but certain locations have been blocked off and she cannot enter them. It takes one unit of time to move between adjacent locations. Here is an example layout of the town:

	(2,5)	(3,5)	(4,5)	
(1,4)	**S1** (2,4)	(3,4)	**S2** (4,4)	(5,4)
(1,3)	(2,3)		(4,3)	(5,3)
	(2,2)	(3,2)	(4,2)	**S3** (5,2)
House (1,1)	(2,1)	**S4** (3,1)	(4,1)	(5,1)

Sabina lives at $(1, 1)$, and no location contains more than one building (Sabina's house or a shop).

(a) Sabina wants to start at her house, visit the shops $S_1, \ldots, S_k$ **in any order**, and then return to her house as quickly as possible. We will construct a search problem to find the fastest route for Sabina. Each state is modeled as a tuple $s = (x, y, A)$, where (x, y) is Sabina's current position, and A is some auxiliary information that you need to choose. If an action is invalid from a given state, set its cost to infinity. Let V be the set of valid (non-blocked) locations; use this to define your search problem. You may assume that the locations of the k shops are known. You must choose a minimal representation of A and solve this problem for general k. Be precise!

- Describe A: _____
- $s_{\text{start}} =$ _____
- $\text{Actions}((x, y, A)) = \{\text{N}, \text{S}, \text{E}, \text{W}\}$
- $\text{Succ}((x, y, A), a) =$

  _____
- $\text{Cost}((x, y, A), a) =$

  _____
- $\text{IsGoal}((x, y, A)) =$

  _____

Solution

- $A = (A_1, A_2, \ldots A_k)$ where $A_i \in \{0, 1\}$ is a boolean representing whether S_i has been visited or not.
- $s_{\text{start}} = (1, 1, [0, \ldots, 0])$
- $$\text{Succ}((x, y, A), a) = \begin{cases} (x, y+1, A') & \text{if } a = \text{N} \\ (x, y-1, A') & \text{if } a = \text{S} \\ (x+1, y, A') & \text{if } a = \text{E} \\ (x-1, y, A') & \text{if } a = \text{W}, \end{cases}$$

  where A' is defined as follows: $A'_i = 1$ if Sabina's new location contains shop i; otherwise, $A'_i = A_i$.
- Let $(x', y', A') = \text{Succ}((x, y, A), a)$. Then $\text{Cost}((x, y, A), a) = 1$ if $(x', y') \in V$ and ∞ otherwise.
- $\text{IsGoal}((x, y, A)) = [x = 1 \wedge y = 1 \wedge A = [1, \ldots, 1]]$.

(b) Sabina is considering a few different methods to visit the shops in as few steps as possible. For each of the following, state whether the algorithm will be able to find a path to visit all shops in as few steps as possible, and if so, provide a running time assuming an $N \times N$ grid.

- Depth-First Search (DFS)
- Backtracking search

Solution

- DFS: Will not be able to find the min-cost path, since the version as defined in lecture stops after finding *any* path.
- Backtracking search: Will be able to find the min-cost path. From lecture, we know the runtime is $O(b^D)$ where b is the branching factor, and D is the maximum depth of the search tree. Here, we can take at most 4 actions at any location (in the cardinal directions), so $b = 4$. There are N^2 positions on a grid of size $N \times N$, and there are 2^k different descriptors A that tell us which of the k shops have been visited. Therefore, there are $O(2^k N^2)$ states. Backtracking search, in the worst case, finds a path through each of these $O(2^k N^2)$ states. At each state, we can take 1 of 4 possible actions. Therefore, the running time is $O(4^{2^k N^2})$ in the worst case (yikes!).

(c) Recall that Sabina is allowed to visit the shops **in any order**. But she is impatient and doesn't want to wait around for your search algorithm to finish running. In response, you will use the A* algorithm, but you need a heuristic. For each pair of shops (S_i, S_j) where $i \neq j$ and $1 \leq i, j \leq k$, define a **consistent** heuristic $h_{i,j}$ that approximates the time it takes to ensure that shops S_i and S_j are visited and then return home. Computing $h_{i,j}(s)$ should take $O(1)$ time.

Solution We will define $h_{i,j}(x, y, A)$ now: based on A, we will have visited some subset of shops in $\{S_i, S_j\}$. We need to compute the distance to visit the remaining shops and return home. Let $d(p, q)$ refer to the Manhattan distance between points p and q, which can be computed in $O(1)$ time. Note that if we haven't visited either S_i or S_j, we must take the min over visiting either one first, in order to produce a consistent heuristic.

$$h_{i,j}(x, y, A) = \begin{cases} \min\{d((x,y), S_i) + d(S_i, S_j) + d(S_j, (1,1)), & \\ \qquad d((x,y), S_j) + d(S_j, S_i) + d(S_i, (1,1))\} & \text{if } A_i = 0 \wedge A_j = 0, \\ d((x,y), S_j) + d(S_j, (1,1)) & \text{if } A_i = 1 \wedge A_j = 0, \\ d((x,y), S_i) + d(S_i, (1,1)) & \text{if } A_i = 0 \wedge A_j = 1, \\ d((x,y), (1,1)) & \text{if } A_i = 1 \wedge A_j = 1. \end{cases}$$

It is clear that computing $h_{i,j}(x, y, A)$ is $O(1)$ it makes $O(1)$ calls to $d(\cdot, \cdot)$.

2) Extra: Problem 2

In 16th century England, there were a set of $N+1$ cities $C = \{0, 1, 2, \ldots, N\}$. Connecting these cities were a set of bidirectional roads R: $(i, j) \in R$ means that there is a road between city i and city j. Assume there is at most one road between any pair of cities, and that all the cities are connected. If a road exists between i and j, then it takes $T(i, j)$ hours to go from i to j.

Romeo lives in city 0 and wants to travel along the roads to meet Juliet, who lives in city N. They want to meet.

(a) Fast-forward 400 years and now our star-crossed lovers now have iPhones to coordinate their actions. To reduce the commute time, they will both travel at the same time, Romeo from city 0 and Juliet from city N.

To reduce confusion, they will reconnect after each traveling a road. For example, if Romeo travels from city 3 to city 5 in 10 hours at the same time that Juliet travels from city 9 to city 7 in 8 hours, then Juliet will wait 2 hours. Once they reconnect, they will both traverse the next road (neither is allowed to remain in the same city). Furthermore, they must meet in the end in a city, not in the middle of a road. Assume it is always possible for them to meet in a city.

Help them find the best plan for meeting in the least amount of time by formulating the task as a (single-agent) search problem. Fill out the rest of the specification:

- Each state is a pair $s = (r, j)$ where $r \in C$ and $j \in C$ are the cities Romeo and Juliet are currently in, respectively.
- Actions$((r, j)) =$ _____
- Cost$((r, j), a) =$ _____
- Succ$((r, j), a) =$ _____
- $s_{\text{start}} = (0, N)$
- IsGoal$((r, j)) = \mathbb{I}[r = j]$ (whether the two are in the same city).

Solution

- Each state $s = (r, j)$ is the pair of cities that Romeo and Juliet are currently in, respectively.

- $\text{Actions}((r, j)) = \{(r', j') : (r, r') \in R, (j, j') \in R\}$ corresponds to both traveling to a connected city
- $\text{Cost}((r, j), (r', j')) = \max(T(r, r'), T(j, j'))$ is the maximum over the two times.
- $\text{Succ}((r, j), (r', j')) = (r', j')$: just go to the desired city

(b) Assume that Romeo and Juliet have done their CS221 homework and used Uniform Cost Search to compute $M(i, k)$, the minimum time it takes one person to travel from city i to city k for all pairs of cities $i, k \in C$.

Recall that an A* heuristic $h(s)$ is consistent if

$$h(s) \leq \text{Cost}(s, a) + h(\text{Succ}(s, a)). \tag{1}$$

Give a consistent A* heuristic for the search problem in (a). Your heuristic should take $O(N)$ time to compute, assuming that looking up $M(i, k)$ takes $O(1)$ time. In one sentence, explain why it is consistent. Hint: think of constructing a heuristic based on solving a relaxed search problem.

$$h((r, j)) = \underline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} \tag{2}$$

Solution Consider the relaxed search problem of giving Romeo and Juliet the option to not wait for each other at every city, but still allowing the waiting at meeting point. Then if Romeo and Juliet are in (r, j), then traveling to some city c in this fashion takes $\max(M(r, c), M(j, c))$. We just need to minimize over all possible cities c:

$$h((r, j)) = \min_{c \in C} \max\{M(r, c), M(j, c)\}. \tag{3}$$

2. Karel Loves Cookies! (*30 points*)

We revisit a homework question and check in on Karel, our favorite cookie-loving robot. Recall that Karel is a robot placed in an $m \times n$ grid and wishes to reach the cell with the cookie using the least cost. There are also pitfalls scattered on the grid – be careful not to step into them!

At each step, Karel may use one of the following commands:

1. `turnLeft()`: rotate the current direction 90 degrees counter-clockwise (costs 1);
2. `turnRight()`: rotate the current direction 90 degrees clockwise (costs 1);
3. `move()`: shift by 1 cell along the current direction (costs 1);
4. `leap()`: shift by 2 cells along the current direction (costs 3).

Note that moving or leaping onto a pitfall has an additional cost of 100 and terminates the game, but leaping over a pitfall doesn't have such penalty.

Initially, Karel starts at the top-left corner of the grid (1, 1) and faces east. You may assume that the cookie is always placed at the bottom-right corner (m, n) unless otherwise specified.

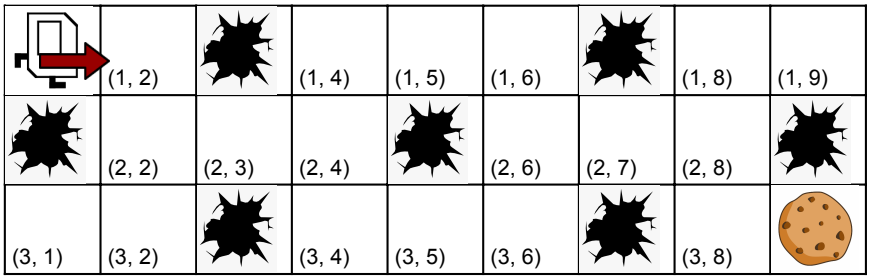


Figure 1: An example 3×9 grid with 7 pitfalls and 1 cookie. Karel starts at (1, 1) and faces east. An example trajectory to the cookie: `move()` into (1, 2), → `leap()` into (1, 4) since there's a pitfall in the middle → `move()` twice into (1, 6) → `leap()` into (1, 8) → `move()` into (1, 9) → `turnRight()`→ `leap()` into the cookie!

a. (*6 points*) Fill out the components of the search problem corresponding to the above problem setting. Each state consists of the cell coordinate (i, j) and the direction $d \in \{E,\ S,\ W,\ N\}$.

In your answer write up, you may use helper functions `counter-clockwise`(d), `clockwise`(d), $\triangle i(d)$, $\triangle j(d)$ whose values are defined in Table 1:

- $s_{\text{start}} = (1, 1, E)$.
- $\text{Actions}(i, j, d) = \{a \in \{\texttt{turnLeft()},\ \texttt{turnRight()},\ \texttt{move()},\ \texttt{leap()}\} :$ $\text{Succ}((i, j, d), a)$ doesn't exceed the boundary$\}$
- $\text{IsEnd}(i, j, d) =$

d	counter-clockwise(d)	clockwise(d)	$\triangle i(d)$	$\triangle j(d)$
E	N	S	0	+1
N	W	E	-1	0
W	S	N	0	-1
S	E	W	+1	0

Table 1: Helper functions for answer write up.

Solution IsEnd$(i, j, d) = \mathbf{1}$[cell (i, j) contains cookie or is a pitfall].

Forgetting to include the pitfall condition was a common mistake.

- Succ$((i, j, d), a) =$

Solution

$$\begin{cases} (i, j, \texttt{counter-clockwise}(d)) & a = \texttt{turnLeft()} \\ (i, j, \texttt{clockwise}(d)) & a = \texttt{turnRight()} \\ (i + \triangle i(d), j + \triangle j(d), d) & a = \texttt{move()} \\ (i + 2\triangle i(d), j + 2\triangle j(d), d) & a = \texttt{leap()} \end{cases}$$

A number of students forgot to include the original i and j values when writing out the successor states for `move()` and `leap()`.

- Cost$((i, j, d), a) =$

Solution

$$C_1 + C_2$$

where

$$C_1 = \begin{cases} 1 & a \in \{\texttt{turnLeft()}, \texttt{turnRight()}, \texttt{move()}\} \\ 3 & a = \texttt{leap()} \end{cases}$$

$$C_2 = \begin{cases} 100 & \text{Succ((i, j, d), a) is a pitfall} \\ 0 & \textit{otherwise} \end{cases}$$

Mistakes the teaching staff saw included using 2 instead of 3 for the cost of `leap()`, forgetting to add the cost of moving into a pitfall to the original `move()` or `leap()` action, and forgetting to account for the pitfall cost altogether.

b. (*6 points*) You decide to use the A* algorithm to help Karel find the cookie with the least cost. Define a consistent heuristic function $h(i, j, d)$ for this problem. Briefly explain why your heuristic function is consistent.

[Hint: Think about a relaxed version of the original problem.]

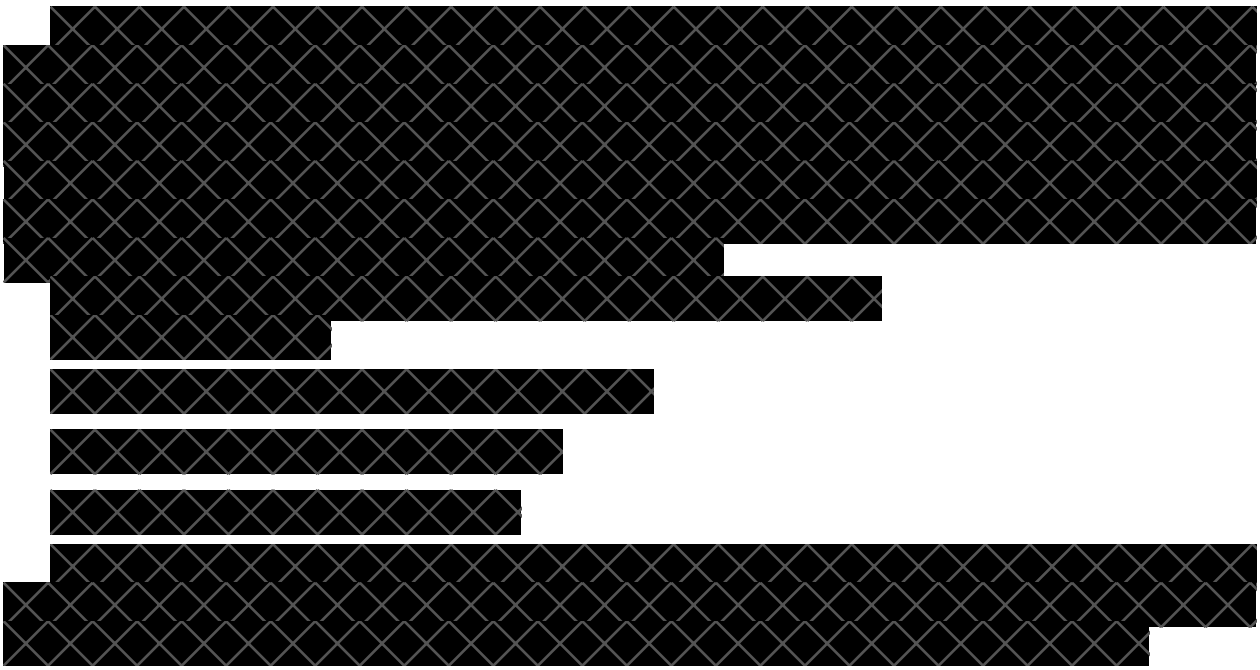
Solution To construct a consistent heuristic, one convenient method is to build a relaxed problem and compute its FutureCost. One possible solution is the Manhattan distance from each cell to the cookie. This is because (1) When all the pitfalls are removed and the direction is ignored, we construct a relaxed version of the original problem, and (2) The FutureCost of this relaxed problem is the Manhattan distance, because the cost of one `leap()` is larger than two `move()`s.

While many students initially proposed the Manhattan distance as a heuristic, a good percentage did not fully explain why it is consistent. In particular, it's important to note that the cost of taking a `leap()` is more expensive than taking two `move()`s. If this were not the case, then the Manhattan distance would not underestimate the minimum cost to the goal state, and hence would not be an admissible (and therefore consistent) heuristic for A*.

c. (*2 points*) Would dynamic programming be an appropriate algorithm for solving this search problem as well? Explain why or why not.

Solution Dynamic programming would not be applicable in this case, as the associated graph is not acyclic. For example, Karel could `turnLeft()` 4 times and wind up facing the same direction in the same location.

One common mistake was to claim dynamic programming is applicable because Karel's *optimal* path does not contain cycles. Unfortunately, while it is true that Karel would not take a cycle on the optimal path, the entire graph must be acyclic for dynamic programming to work. It's a good exercise to think about how even a single cycle in the graph would cause DP to break down and infinitely loop.



2. Maze (*50 points*)

One day, you wake up to find yourself in the middle of a corn field holding an axe and a map (Figure 1). The corn field consists of an $n \times n$ grid of cells, where some adjacent cells are blocked by walls of corn stalks; specifically, for any two adjacent cells (i, j) and (i', j'), let $W((i, j), (i', j')) = 1$ if there is a wall between the two cells and 0 otherwise. For example, in Figure 1, $W((1, 1), (1, 2)) = 0$ and $W((1, 2), (1, 3)) = 1$.

You can either move to an adjacent cell if there's no intervening wall with cost 1, or you can use the axe to cut down a wall with cost c without changing your position. Your axe can be used to break down at most b_0 walls, and your goal is to get from your starting point (i_0, j_0) to the exit at (n, n) with the minimum cost.

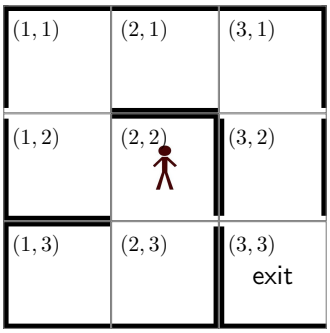


Figure 1: An example of a corn maze. The goal is to go from the initial location $(i_0, j_0) = (2, 2)$ to the exit $(n, n) = (3, 3)$ with the minimum cost.

a. (*15 points*)
(i) [10 points] Fill out the components of the search problem corresponding to the above maze.

- $s_{\text{start}} = ((i_0, j_0), b_0)$.
- $\text{Actions}(((i,j),b)) = \{a \in \{(-1,0),(+1,0),(0,-1),(0,+1)\} :$
  $(i,j)+a$ is in bounds and $(W((i,j),(i,j)+a) = 0$ or $b > 0)\}$.
- $\text{IsEnd}(((i,j),b)) =$

  Solution $\text{IsEnd}(((i,j),b)) = [i = n \text{ and } j = n]$.
- $\text{Succ}(((i,j),b),a) =$

  Solution The new location is $(i,j)+a$, and we need to decrement our axe budget b whenever $W((i,j),(i,j)+a) = 1$.

$$\text{Succ}(((i,j),b),a) = ((i,j)+a, b - W((i,j),(i,j)+a)) \tag{1}$$

  During exam we made clarifications that $c > 0$ and the cost c should be rephrased as the cost to break down a wall in a move (if there is a wall during a move, we take it down with extra cost c). Some students might not get our clarification during exam, so we also accept this solution in this and following questions:

$$\text{Succ}(((i,j),b),a) = ((i,j)+(1-W((i,j),(i,j)+a))a, b - W((i,j),(i,j)+a))$$
$$W((i,j),(i,j)+a) = 0$$

- $\text{Cost}(((i,j),b),a) =$

  Solution We always pay cost 1 for moving one cell and additionally, we pay c if we have to break down a wall.

$$\text{Cost}(((i,j),n),a) = 1 + cW((i,j),(i,j)+a). \tag{2}$$

  We also accept this solution:

$$\text{Cost}(((i,j),n),a) = (1 - W((i,j),(i,j)+a)) + cW((i,j),(i,j)+a). \tag{3}$$

(ii) [5 points] When you use an axe to take down a wall, the wall stays down but the set of walls which have been taken down are not tracked in the state. Why does our choice of state still guarantee the minimum cost solution to the problem?

Solution Under the current formulation, one needs to pay each time we go through a wall. However, the minimum cost path will never visit the same location twice (otherwise, there's a cycle which can be cut out to reduce the cost of the path). Therefore, there is no difference between paying only for the first time and paying for each time.

b. (*10 points*)

Solving the search problem above is taking forever and you don't want to be stuck in the corn maze all day long. So you decide to use A*.

(i) [5 points] Define a consistent heuristic function $h(((i,j),b))$ based on finding the minimum cost path using the relaxed state (i,j) where we assume we have an infinite axe budget and therefore do not need to track it. Show why your choice of h is consistent and what you would precompute so that evaluating any $h(((i,j),b))$ takes $O(1)$ time and precomputation takes $O(n^2 \log n)$ time.

Solution Define $h(((i,j),b))$ to be the minimum cost path from (i,j) to (n,n), where we have no budget constraint on the number of times we can use the axe. We can solve the relaxed search problem from the exit (n,n) to all locations (i,j). This can be done using UCS in $O(n^2 \log n)$ time. Then, we just store the lookup table for each of the $O(n^2)$ states to be queried during the search.

This problem only relaxes axe budget constraint, so any heuristic relaxing wall or cost related constraint (such as Manhattan distance to the end, which just remove all walls or set $c = 0$) will not be accepted.

(ii) [5 points] Noticing that sometimes h is the true future cost of the original search problem, you wonder when this holds more generally. For what ranges of b_0 and c would this hold? Assume for this part that there is a path that doesn't require breaking down any walls.

$\_\_\_\_\_ \leq b_0$ $\_\_\_\_\_ \leq c$

Your lower bounds need not be tight, but you need to formally justify why they hold.

Solution The heuristic is exactly the future cost when the number of axe uses of the original and relaxed search problems are identical. This happens in two cases:

1. When the budget constraint b_0 is large enough, then there's effectively no constraint on the number of axe uses. This happens when b_0 is at least the largest number of walls that need to be broken down. In worse case we only need to break down $2n$ walls (otherwise we can directly head to the exit with less cost).

2. When c is large enough, then it is not worth breaking down any walls at all. This happens when $c \geq n^2$, an upper bound on the length of the minimum cost path without breaking down walls (which exists by assumption).