CS221 Problem Workout Solutions

Week 3

1) [CA session] 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 $\text{Cost}((x, y, A)) = 1$ 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) [Breakouts] 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:

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