# CS221 Problem Workout Solutions

Week 3

## Key Takeaways from this Week

1. **Machine Learning II**:

    - **Generalization**: The real objective: minimize loss on unseen future examples, not the training loss. Use the test set loss to estimate the performance on unseen examples. Try to minimize training error, but keep the hypothesis class small.
    - **Best Practices**: Follow good data hygiene and always prepare a test set and don't look at it. Use the validation set for tuning and testing. Practice allows you to develop more intuition, learn the best design decisions, and how to tune hyperparameters.
    - **K-means**: simple a widely-used method for discovering cluster structure in data. K means can mean the objective and the algorithm.

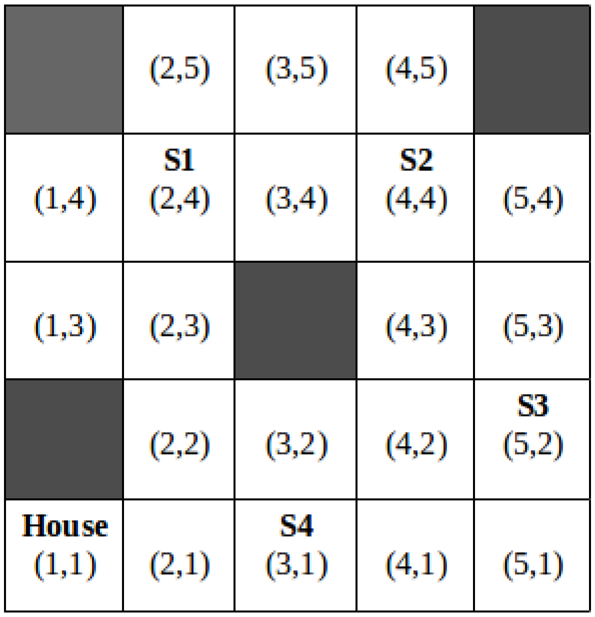
2. **Introduction to Search**: we introduced search problems, our first instance of a state-based model. We formalized how to define search problems, with states, successors, actions, and cost functions.

    - **Tree Search**: Four algorithms to keep in mind: backtracking search, depth-first search, breadth-first search, and depth-first search with iterative deepening.
    - **Dynamic Programming**: an algorithm that is akin to backtracking search with memoization with the benefits of exponential savings. The states in DP contain a summary of past actions sufficient to choose future actions optimally.

# Practice Problems

## 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 $S_1, \dots, S_k$. 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:



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 $S_1, \dots, S_k$ **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**

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

  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 $\infty$ otherwise.
- $\text{IsGoal}((x, y, A)) = [x = 1 \wedge y = 1 \wedge A = [1, \dots, 1]]$.

(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. Depending on $A_i$ and $A_j$, we have the following four cases:

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

Computing $h_{i,j}(x, y, A)$ is $O(1)$ if it makes $O(1)$ calls to $d(\cdot, \cdot)$.
To show that this heuristic is consistent, first note that when we take any action

(taking one step in any direction), the Manhattan distance from any location to the current state and the successor state will differ by exactly 1. That is, for any current state $s$ and action $a$:

$$|d(s,(x,y)) - d(succ(s,a),(x,y))| = 1 \text{ for all } (x,y) \in V$$

We will verify the following:

$$cost'(s,a) = cost(s,a) + h_{ij}(succ(s,a)) - h_{ij}(s) \geq 0 \text{ for all } s,a,i,j.$$

We assume that $cost(s,a) = 1$ (the action is valid) and let $succ(s,a)$ be the successor state's location for the distance calculations. Let's case on $A_i$ and $A_j$.
In the first case,

$$1 + d(succ(s,a),(1,1)) - d(s,(1,1)) \geq 0$$

because $d(succ(s,a),(1,1)) - d(s,(1,1))$ is either 1 or $-1$.
In the second case, we have

$$\begin{aligned}&1 + d(succ(s,a),S_j) + d(S_j,(1,1)) - d(s,S_j) - d(S_j,(1,1))\\ =&1 + d(succ(s,a),S_j) - d(s,S_j)\\ \geq&0\end{aligned}$$

The third case is analogous.
For the fourth case (where 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. This is because $h_{ij}(succ(s,a))$ and $h_{ij}(s)$ will differ by one if they are both visiting $S_i$ and $S_j$ in an order that minimizes the Mahanttan distance. If either heuristic is allowed to visit the shops in an order that does not yield the minimum distance, then we have no such guarantee.

## 2) Problem 2

In 16th century England, there were a set of $N+1$ cities $C = \{0, 1, 2, \dots, N\}$. 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.

- Actions$((r, j)) = \{(r', j') : (r, r') \in R, (j, j') \in R\}$ corresponds to both traveling to a connected city
- Cost$((r, j), (r', j')) = \max(T(r, r'), T(j, j'))$ is the maximum over the two times.
- 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. Explain why it is consistent. Hint: think of constructing a heuristic based on solving a relaxed search problem.

$$h((r, j)) = \underline{\hspace{8cm}} \tag{2}$$

**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 $c$:

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

Now consider the inequality in (1). We need to verify that for every action that takes us from $(r, j)$ to some $(r', j')$:

$$\min_{c \in C} \max\{M(r, c), M(j, c)\} \leq \text{Cost}((r, j), (r', j')) + \min_{c' \in C} \max\{M(r', c'), M(j', c')\}.$$

On the right hand side, we have the travel time when we start from $(r, j)$, go through $(r', j')$ and wait for each other (as specified by Cost), and then meet at some city $c'$. On the left hand side, we have the travel time for Romeo and Juliet to both get to a city (that may be $c'$) without waiting in intermediate cities. Therefore, the inequality holds and the heuristic is consistent.