

## Planning and Reinforcement Learning (046203)

### Homework 1, due 20/4/2020

#### Question 1 (Longest Increasing Subsequence)

In the lectures we saw the *longest increasing subsequence problem*, in which we are given a sequence  $A_n$  of  $n$  numbers  $\{a_i\}_{i=1}^n$ , and we defined the *longest increasing subsequence* (LIS) of  $A_n$  as the longest *strictly* increasing subset of  $\{a_i\}_{i=1}^n$ . For example, the LIS of (9, 2, 6, 4, 8) is (2, 6, 8) or (2, 4, 8). In the lectures we formulated a DP approach to find the length of the LIS of  $A_n$ .

Now, we define the *longest increasing odd even subsequence* (LIOES) of a sequence  $A_n$ , as the longest increasing subsequence of  $A_n$  whose terms alternate between odd and even numbers. For example, the LIOES of (4, 5, 8, 3, 6, 7, 9) is (4, 5, 6, 7) or (4, 5, 6, 9).

- Efficient solution – formulate a DP approach to find the size of the LIOES of a sequence of length  $n$ . Write the algorithm down in pseudocode.
- How does the complexity of the algorithm you suggested compare to that of an exhaustive solution (checking all possibilities)?

#### Question 2 (Counting)

Given an integer number  $X$ , we would like to count in how many ways it can be represented as a sum of  $N$  natural numbers (with relevance to order) marked by  $\Psi_N(X)$ . For example,  $\Psi_2(3) = 2$  since:  $3 = 1 + 2 = 2 + 1$ .

- Find the following:  $\Psi_1(2), \Psi_2(4), \Psi_3(2), \Psi_N(N)$  and  $\Psi_1(X)$ .
- Find a recursive equation for  $\Psi_N(X)$ .
- Write a code in for finding  $\Psi_N(X)$  using dynamic programming.
  - What is the time and memory complexity of your algorithm?
  - Find  $\Psi_{12}(800)$ .
- Now assume each natural number  $i$  is associated with some cost  $c_i$ . For a given  $X, N$  we are interested in finding the lowest cost combination of natural numbers  $\{x_i\}_{i=1}^N$  satisfying  $\sum_{i=1}^N x_i = X$ .
  - Formulate the problem as a finite horizon decision problem: Define the state space, the action space and the cumulative cost function.
  - Bound the complexity of finding the best combination.

### Question 3 (Language model)

In the city of Bokoboko the locals use a language with only 3 letters ('B','K','O'). After careful inspection of this language, researchers have reached two conclusions:

- I. Every word starts with the letter 'B'.
- II. Every consecutive letter is distributed only according to the previous letter as follows:

$$P(l_{t+1} | l_t) = \begin{matrix} & \begin{matrix} B & K & O & - \end{matrix} \\ \begin{matrix} B \\ K \\ O \\ - \end{matrix} & \begin{bmatrix} 0.1 & 0.325 & 0.25 & 0.325 \\ 0.4 & 0 & 0.4 & 0.2 \\ 0.2 & 0.2 & 0.2 & 0.4 \\ 1 & 0 & 0 & 0 \end{bmatrix} \end{matrix}$$

Where '-' represents the end of a word. For example, the probability of the word 'bko' is given by  $0.325 \cdot 0.4 \cdot 0.4 = 0.052$ .

- a. Find the probability of the following words: 'Bob', 'Koko', 'B', 'Bokk', 'Boooo'.
- b. We wish to find the most probable word in the language of length  $K$ .
  1. Formulate the problem as a finite horizon decision problem: Define the state space, the action space and the **multiplicative** cost function.
  2. Bound the complexity of finding the best combination.
  3. Find a reduction from the given problem to an analogous problem with **additive** cost function instead.
  4. Explain when each approach (multiplicative vs. additive) is preferable. Hint: Think of the consequence of applying the reduction on the memory representation of a number in a standard operating system.
  5. Write a code which finds the most probable word of a given size using dynamic programming. What is the most probable word of size 5?

### Question 4 (LCS of Three Strings)

Consider the problem of finding a longest subsequence of three strings. For example, given the strings  $X = \text{'ABCBD'}$ ,  $Y = \text{'ACD'}$ ,  $Z = \text{'AAACD'}$  the outputs should be 'ACD'.

In tutorial 1, question 2, we solved the two-strings LCS problem. In this question you will use a similar principle to solve the three-strings LCS problem.

- i. Suggest a brute force algorithm to solve the three-strings LCS problem. What is its computational complexity?
- ii. Generalize the theorem from section (c), tutorial 1, to the three strings LCS problem.
- iii. Building on the generalization, suggest a dynamic programming algorithm to solve the three strings LCS problem. Your algorithm should return the actual string, not just the length of the LCS. Write your algorithm in a pseudocode form.

### **Question 5 (MinMax dynamic programming)**

In this problem we consider an adversarial version of finite horizon dynamic programming, which is suitable for solving 2-player games. In this setting, at time  $k \in \{0, 1, 2, \dots, N-1\}$  the system is at state  $s_k \in S_k$ , the agent chooses an action  $a_k \in A_k(s_k)$  according to the agent policy  $\pi_k^a(s_k)$ , and subsequently the opponent chooses an action  $b_k$  from the set of allowable opponent actions  $B_k(s_k, a_k)$ , according to the opponent policy  $\pi_k^b(s_k, a_k)$ .

The system then transitions to a new state according to:

$$s_{k+1} = f_k(s_k, a_k, b_k), \quad k = 0, 1, 2, \dots, N-1.$$

The instantaneous reward is denoted by  $r(s_k, a_k, b_k)$ , and, for an N-stage path  $h_N = (s_0, a_0, b_0, \dots, s_{N-1}, a_{N-1}, b_{N-1}, s_N)$  the cumulative reward is

$$R_N(h_N) = \sum_{k=0}^{N-1} r_k(s_k, a_k, b_k) + r_N(s_N).$$

Given  $s_0$ , the agent's goal is to find a policy  $\pi_a^*$  that maximizes the *worst-case* cumulative reward:

$$\pi_a^* \in \arg \max_{\pi_a} \min_{\pi_b} R_N(h_N)$$

- Formulate a dynamic programming algorithm for this problem. Explain what the value function represents in this case.
- What is the computational complexity of your algorithm?
- Could this approach be used to solve the game of tic-tac-toe? Explain what are the states, actions and rewards for this game.
- Could this approach be used to solve the game of chess? Explain.