CS221 Fall 2018 Homework [Reconstruct]

SUNet ID: prabhjot
Name: Prabhjot Singh Rai

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Problem 1

(a) Greedy algorithm, since it's inclined to make a choice that seems best for that particular moment, focuses on choosing the minimum cost path for the given model at the every state it visits. Since it does not consider future costs, therefore, the greedy algorithm is suboptimal for such problems. Consider a corpus "In the last test, antioxidants were found to be present. This result is anti intuitive, since for earlier cases, oxidants were not present". Assuming that the language model is a uni-gram model. The cost assigned to words $[w_1, w_2...w_n]$ can be defined as the inversely proportional to the count of the word present in the corpus.

$$u(w) = \begin{cases} \frac{c}{\text{count of } w \text{ in corpus}}, & \text{if w in given corpus, } c > 0 \\ P, & \text{if w not given corpus, } P \text{ significantly larger than } c \end{cases}$$

Running greedy algorithm on "antioxidantswerepresent" will pick up words occuring in the corpus one by one. The exact result will be "anti", "oxidants", "were" and "present". This will cost $4c$, whereas an optimal solution of lower cost is $3c$ ("antioxidants", "were" and "present").

Problem 2

(a) The approach of greedy algorithm that it chooses a path which has minimum cost for that node, and doesn't consider future costs while calculating the minimum cost to reach the end. Consider a corpus "He booked ticket. He baked cookies. He booked hotel. She baked cookies.". Let the possible insertions be defined as an object with key as query and value being the different possible words with vowels, and is depicted in **Table 1**.

Let the bigram cost be defined as inverse of all the counts of different bigrams in the given text, and for those which do not exist, let's assume the cost to be way higher than C, example $1000C$ as depicted in **Table 2**.

Let the input list be: $['h','bkd','cks']$. The different paths are: $['he','booked','cookies']$ and $['he','baked','cookies']$.

Table 1: Table depicting possible insertions

Word without vowel	**Possible Insertions**
h	he
bkd	baked, booked
cks	cookies
htl	hotel
sh	she

Table 2: Table depicting bigram costs based on count

Bigram	**Bigram Cost**
he booked	C
he baked	2C
booked ticket	2C
baked cookies	C
booked hotel	2C
she baked	2C
Any other bigram	1000C

For the greedy algorithm, since the cost of ["he", "booked"](C) is lower than ["he", "baked"]($2C$), therefore the greedy algorithm chooses "he booked". But this will add up very high cost while evaluating ["booked", "cookies"] bigram, since this doesn't exist in the bigrams for the above corpus which will be assigned a value of $1000C$. Overall cost of this path will be $1001C$. Whereas, an optimum solution would consider taking ['he', 'baked'] as the first choice even if it's costlier ($2C$) to minimise the overall cost, since the future cost of ['baked', 'cookies'] (C) is smaller than ['booked', 'cookies'].

Problem 3

(a) Since the cost function is a bi-gram model, our state should contain just enough information to find the cost of going to the next state (here in cost of next possible fill given the previous fill). Therefore, it should contain the previous fill, and the begin index of the next string.

$$state = \textbf{PREVIOUS WORD}, \textbf{BEGIN INDEX}$$

Start state would be previous word as sentence begin word, and begin index of the next string to be zero .

$$startState = \textbf{SENTENCE BEGIN}, 0$$

Each of the actions would be the next word from the possible fills of vowels for the next substring. Cost at any point will be computed through bigram cost function which takes **PREVIOUS WORD** and next possible fill as the input.
Finally, the end state would be when our **BEGIN INDEX** has visited the end of the query string, therefore:

$$endFn(state) => state[1] == len(query)$$

(c) Since we need to define $u_b(w)$ as a cost function for the relaxed problem(P_{rel}) which is also a heuristic for the overall problem, it should return a cost less than or equal to the bi-gram cost function $b(w', w)$. Therefore, for every input w, if u_b returns the minimum value of all the possible costs returned by $b(w', w)$, when w' belongs to all possible words(states) that can occur prior to the word and w is the next word. Assuming that all the possible words occurring prior to the next word on any state won't be any word outside the given corpus, therefore

$$u_b(w) = min\{b(w'_1, w), b(w'_2, w)...b(w'_n, w)\}$$

where $w'_1, w'_2....w'_n$ are distinct words in the given corpus.

Our state will only consist of begin index for the next word, action will be next possible insertion for each combination of characters without spaces starting from the begin index, cost will be $u_b(action)$, start state will be 0 and end state would be when the begin index is equal to the length of the query.

$h(s)$ consistency condition are two:

1. $Cost(s, a) + h(Succ(s, a)) - h(s) \geq 0$ or $Cost(s, a) + h(Succ(s, a)) \geq h(s)$

Since our heuristic is the relaxed problem defined above (P_{rel}), $h(s)$ depicts lowest cost to reach the endpoint from any given state s, since on every action, our cost is minimum of all the possible bigram costs at that state ($C_{rel} \leq C$). Therefore, actual cost of performing an action a, which is depicted as sum of the past cost ($Cost(s, a)$) and future cost ($h(Succ(s, a))$ will be greater than or equal to the heuristic at state $s(h(s))$.

2. $h(s_{end}) = 0$

This is true since if we are already at the end node, P_{rel} is zero.

(d) Yes, UCS is a special case of A^* search. Since A^* explores in the order of sum of past and future costs, and UCS explores states in the order of past costs, therefore when the future cost (or heuristic value) is zero for every state, A^* is same as UCS.

Yes, BFS is a special case of UCS as just like UCS, it tends to explore the states in the order of increasing costs, but unlike UCS, BFS assumes that the cost for each action is the same for every state. Therefore, BFS is UCS with an assumption that all the costs are equal and positive. UCS is more flexible and can be used where the cost of actions are not equal. Both assume that the costs are positive.