# 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. Total cost assigned to words $[w_1, w_2...w_n]$ is defined as $\sum_i^n u(w_1)$, where

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

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 is the same as described in 1a above, that it chooses a path which has minimum cost for that moment, and doesn't consider future costs. Consider a corpus "He booked ticket. He baked cookies. He baked sandwiches.". Let the possible insertions be defined as an object with key as query and value being the different possible words with vowels:

$$\text{possible insertions} = \{h : [\prime he\prime], bkd : [\prime baked\prime, \prime booked\prime], cks : [\prime cookies\prime], ...\}$$

And 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 $\infty$. Let the input list be: $[\prime h\prime, \prime bkd\prime, \prime cks\prime]$. The different paths are: $[\prime he\prime, \prime booked\prime, \prime cookies\prime]$ and $[\prime he\prime, \prime baked\prime, \prime cookies\prime]$. For the greedy algorithm, since the count of ["he", "booked"] is higher, the cost is lesser, therefore it takes this path. But this will cause the algorithm

to add up infinitely large cost when evaluating ["booked", "cookies"] bigram, since this doesn't exist in the bigrams for the above corpus. Whereas, an optimum solution would consider taking ['he', 'baked'], in order to reduce the future cost (['baked', 'cookies'] bigram exists in corpus).

# Problem 3

(a) Since the cost function is a bi-gram model, our state should contain just enough information to create the current word, since this current word will be used along with possible vowel insertions to determine the next state. Therefore, the current state would have three elements, namely:

$state = (j : \text{end index of query}, i : \text{initial index of query}, n : \text{index of possible vowel fills})$

Using this state, we can get the current word by finding substring of query from start index to end index and then finding the $n^{th}$ index of this substring.
Start state would be where our start and end indexes of query are 0 along with index of possible vowel fills.

$$startState = 0, 0, 0$$

Our actions would be words formed by insertions of both spaces and vowels, thus will be the next possible word that can be added after the current word. Cost at any point will be computed through bigram function which takes currentStateWord and nextStateWord as the input.
Finally, the end state would be when our end index has visited the end of the string, therefore:

$$endFn(state) => state[0] == len(query)$$

(c) Since we need to define $u_b(w)$ as a cost function for the relaxed problem 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 $w' \epsilon$all possible states/words that can be at current d and $w$ is the next word.

$$u_b^d(w) = min\{b(state_0, w), b(state_1, w)...b(state_{d'}, w)\}$$

where $state_0, state_1....state_{d'}$ are all the possible states at d, which is the "end index of the query string".

Our state will only consist of end index of the query string, action will be next vowel inserted word, cost will be $u_b^d(action)$, start state will be 0 and end state would be when the end index is equal to the length of the query.

$h(s)$ consistency condition are two:

1. $Cost(s, a) + h(Succ(s, a)) > h(s)$
For this problem, $h(s)$ will have the lowest cost to reach the endpoint from any given state, since we are summing the minimum costs while taking actions. For the overall problem, the cost from this point can be minimum or can be more than that. Therefore the above inequality holds true.

2. $h(s_{end}) = 0$ This is true since if we are already at the end node, there won't be any cost for the next state.

(d) No, it's the other way round. $A^*$ is a special case of UCS, where we add a bias(future) to the UCS cost function to force it to follow paths which have lower future costs, thereby increasing the chances of finding the lowest cost solution in lesser time.
Yes, BFS is a special case of UCS as just like UCS, it tends to explore the states which take same costs to reach, but unlike UCS, BFS assumes that the cost for each action is the same for every state. UCS is more generic and is used where the cost of actions are not equal. Both assume that the costs are positive.