CS221 Fall 2015 Homework [reconstruct]

Problem 1: word segmentation

1a.

Consider the string:

theventuresomeentrepreneurcreatedastartup

The bi-gram cost model with n=2 is defined as:

b(-BEGIN-, the) = .02
b(the, venture) = .4
b(the, venturesome) = .8
b(venture, some) = .5
b(some, entrepreneur) = .5
b(venturesome, entrepreneur) = .2
b(entrepreneur, created) = .5
b(created, a) = .2
b(a, startup) = .5

The greedy algorithm will segment the string with a cost of 2.62 as follows:

the venture some entrepreneur created a startup

However this is not the optimal segmentation. The optimal segmentation would have been at a cost of 2.22 as follows:

the venturesome entrepreneur created a startup

Problem 2: vowel insertion

2a.

Consider the sequence of words with missing vowels:

sh sws sds

The set of possible fills for these words is as follows:
possibleFills('sh') = she
possibleFills('sws') = sews, sows
possibleFills('sds') = seeds

With the following bi-gram cost model:
b(-BEGIN-, she) = .05
b(she, sews) = .2
b(she, sows) = .3
b(sews, seeds) = .9
b(sows, seeds) = .1

The greedy algorithm will perform vowel insertion with a cost of 1.15 as follows:

she sews seeds

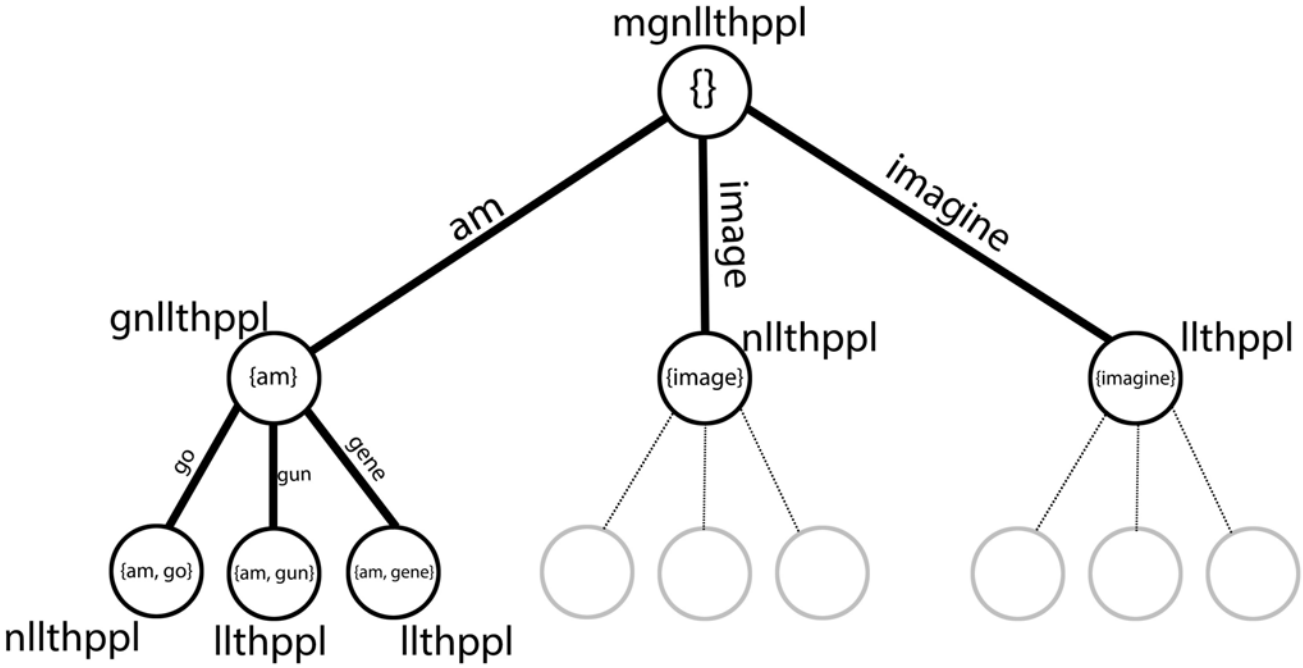
This is not the optimal solution. The optimal solution is with a cost of .45 and is as follows:

she sows seeds

Problem 3: putting it together

3a.

The search tree for the problem can be illustrated as follows:



We start our search using the input string as the initial unparsed string: "mgnllthppl"
Next we will use a dictionary that maps the vowel free words to the original words.
Then at any state we match words from the reduced dictionary to valid prefixes of the unparsed string. When we find a match we remove the corresponding matched characters from the unparsed string of the successor node. For every node, a path can be calculated by following parent pointers to the root node. This path defines a sequence of words, for example the node {am, go} in the above diagram defines the sequence: **-BEGIN- am go**
The bi-gram cost for this node is: **b(-BEGIN-, am) + b(am, go)**
The goal condition would be that the unparsed string is of zero length.

Therefore, using these steps as described, the states, actions, initial state and goal test can be defined as:
states = <sequence_of_matched_valid_words[], unparsed_string>
actions = possibleFills(prefix_of_unparsed_string)
costs =
Let $w_1, w_2, w_3, \ldots w_L$ be equal to: sequence_of_matched_valid_words, and
L= length(sequence_of_matched_valid_words)

$$cost(at\ any\ node) = \sum_{i=0}^{L-1} b(\ w_i, w_{(i+1)}\ ) \quad \text{where} \quad w_0 = \text{-BEGIN-}$$

initial state = < sequence_of_matched_valid_words = None, unparsed_string = input_string >
goal test = if(length(unparsed_string) == 0)

3c.
Given $b(w', w)$ we can define $u(w)$ as follows:

$u(w) = min_{w'} \; b(w', w)$
Now we can define a heuristic for a given search node *s* as follows:

$h_u(s)$ = cost of joint space and vowel insertion assuming the unigram cost model but using the costs $u(w) = min_{w'} \; b(w', w)$

For proof it is consistent, it suffices to prove the triangle inequality:

$h_u(s) \leq h_u(s') + cost(s, s')$ where *s'* is a successor of *s* in the search tree.

Let s = < matched_words , unparsed_string > and s' = <matched_words' , unparsed_string' >
Because *s'* is a successor of *s* then unparsed_string' is a suffix of unparsed_string

Let *w* be the word that annotates the edge from *s* to *s'*
$cost(s, s') = u(w) = min_{w'} \; b(w', w)$

$h_u(s)$ represents the cost of joint space and vowel insertion for unparsed_string in the unigram cost model

$h_u(s') + cost(s, s')$ represents the cost of joint space and vowel insertion for unparsed_string in the unigram cost model conditioned on the first word being *w*

Since the latter is more restrictive than the former, this proves that
$h_u(s) \leq h_u(s') + cost(s, s')$ is true and since we also know that $h_u(s') = 0$ when $s' = goal$ because unparsed_string will be empty, together this proves that the proposed heuristic is consistent.