CS221, Spring 2019, PS2 Reconstruct

Haiyuan Mei (hmei0411@stanford.edu)

Problem 1: word segmentation

a. Show that the greedy search is suboptimal

- The following example input string on greedy approach fails to find the lowest-cost segmentation of the input (generated by unigramCost function). Example: whatisyouname
  Should be 'what is your name' in UCS algorithm, cost for each words are with UCS algorithm:
  - what 5.47877310482
  - is 5.08040259845
  - your 6.46401383481
  - name 8.32568529323
    totalCost = 25.34887483131

  But the greedy algorithm ends up with: what i s you r name.
  - what 5.47877310482
  - i 4.83525677784
  - s 5.08040259845
  - you 5.00015324078
  - r 6.46401383481
  - name 8.32568529323
    totalCost = 35.18428484993
- The reason is because in the whole search tree, the greedy approach is completely ignorance of future costs; it makes decision merely according to the next state. Mathematically speaking, suppose $j = \arg\min \ell(s_{0-j})$ is the index that gives the minimum cost in all sub strings of s, starting from index 0 to j but exclude index j, n is the total length of s; and the unigram cost is denoted as $\ell_{uni}(w)$:
  $$\begin{aligned}\ell(s) &= \min_i(\ell_{uni}(s_{0-i} + \ell(s_{i-n})) \\ &\leq \ell_{uni}(s_{0-j}) + \ell(s_{j-n})\end{aligned}$$
  The above tells us that even the first sub string gives the least cost, it cannot guarrantee the cost of total cost of the whole string.

b. Coding of the state-space search problem

Problem 2: vowel insertion

a. Show that the greedy search is suboptimal

- The example below shows that the greedy approach fails to find the lowest-cost vowel insertion (cost function generated by bigramModel in makeLanguageModels).
  Example: ['ths', 'ppl', 'wrkd', 'hpply']
  Should be 'these people worked happily', cost for each bigram with UCS algorithm:
  - (-BEGIN, these) 9.23381883315
  - (these, people) 10.0867919595
  - (people, worked) 13.3055145899
  - (worked, happily) 13.304704934
    totalCost = 45.93083031655

  But the greedy approach ends up with: this appeal worked happily
  - (-BEGIN-, this) 7.88331357988
  - (this, appeal) 13.3083299497
  - (appeal, worked) 13.3047082673
  - (worked, happily) 13.304704934
    totalCost = 47.80105673088
- mathematical explaination. Suppose the vowel free string list is s, previous word is denoted as p, the current vowel free string index is i, and the possible fills for s_i is list f_i; and suppose k is the index of f_i such that $f_i[k]$ gives the minimum next bigram cost. The bigram cost is denoted as $\ell_{bi}(w1, w2)$ and the cost function satisfies the following:
  $$\begin{aligned}\ell(p, s_i) &= \min_j(\ell_{bi}(p, f_i[j]) + \ell(f_i[j], s_{i+1})) \\ &\leq \ell_{bi}(p, f_i[k]) + \ell(f_i[k], s_{i+1})\end{aligned}$$
  The above inquality explains that the greedy approach cannot supass the UCS algorithm, it can only reach suboptimal.

Problem 3: putting it together

a. Find a minimal representation of the states.

Suppose the current remaining string is s, i is a index such that s[:i] is used for possibleFills to generate possible words.

- States: (prevWord, s[i:]), prevWord is one of possibleFills(s[:i]), and s[i:] is the remaining vowel free space free sub string, $1 \leq i < len(s)$
- s_{start}: (wordsegUtil.SENTENCE_BEGIN, s[0:]) tuple.
- Actions(s): one word chosen from possibleFills(s[:i]), $1 \leq i < len(s)$
- Cost(s; a): bigramCost(prevWord, a), a is chosen from possibleFills(s[:i]), $1 \leq i < len(s)$.
- Succ(s; a): (a, s[i:]), a is chosen from possibleFills(s[:i]), $1 \leq i < len(s)$
- IsEnd(s): s== ''?

Since bigram cost needs to have two consequent words as input, the states should keep track of previous word and current remaining string in order to compare different costs, thus the above (prevWord, s[i:]) is the minimal representation.

c. Speed up joint space and vowel insertion with A*

For the relaxed problem:

- We can apply 'Easier Search' trick as introduced in class, simplify the state space to include only the current remaining string. That way the state space only contains n elements;
- The actions will become the index i used to retrive next word w from 'possibleFills(s[:i])'.
- The cost for each state is is simply $u_b(w) = \min_i(b(w_i, w))$, here w is chosen from 'possibleFills(s[:i])'
- The end state is just an empty string.

The modified cost of relaxed problem $u_b(w) = \min_i(b(w_i, w))$ basically says,

$$\begin{aligned} \text{textrmCost}_{rel}(s, a) &= u_b(w) \\ &= \min_i(b(w_i, w)) \\ &\leq b(w', w) \\ &= \text{textrmCost}(s, a) \end{aligned}$$

Now prove the consistency of the relaxed heuristic $h(s)$:

$$\begin{aligned} h(s) &= \text{textrmFutureCost}_{rel}(s) \\ &\leq \text{textrmCost}_{rel}(s, a) + h(\text{Succ}(s, a)) \\ &\leq \text{textrmCost}(s, a) + h(\text{Succ}(s, a)) \end{aligned}$$

The first $\leq$ is because of triangle inequality, the second $\leq$ is from relaxation; the above says the heuristic is consistent.

d. Relationship between different search algorithms

- Is UCS a special case of A*? The answer is yes, UCS is a special case of A* with h(s) = 0.
- Is BFS a special case of UCS? BFS requires the cost of each edge to be constant. In this sense BFS behaves the same as UCS and can be considered a special case of UCS. The difference is that since BFS assumes constant edge cost, it uses an FIFO queue as opposed to a priority queue used by UCS.