Heuristic Algorithm Notes

Jonathan Pearce

July 3, 2018

Key:
Word Length = k
Sequence Length = L
Number of words = n
Optimal Word = A word comprised of only the most likely nucleotide at each position
Negative Sequence = A sampled sequenced that will not be matched to an element in the word set

1 Algorithm 1: Random Sampling

Algorithm 1 Title

$wordSet \leftarrow$ best n optimal words
while within time limit **do**
  for i in $(0, 1, ..., n-1)$ **do**
    $w_i = eval(wordSet \setminus i)$
  remove word i with probability $\propto w_i$
  $samples \leftarrow x$ negative sequence samples
  $word \leftarrow$ most frequently occuring word in $samples$
  $Add\ word$ to $wordSet$
return wordSet

General Idea: Begin with an easy to find and somewhat successful set of n words. Upon each iteration select one word from the set to remove with probability inversely proportional to how much that word contributes to the set probability (i.e. a word contributes a lot to the set score, remove with low probability and vice versa). Collect x negative sequences and process, find the word and index pair that occurs most often, add this word to the word set.

2 Algorithm 2: Linear Random Sampling

Algorithm 2 Title

1: $wordSet \leftarrow \emptyset$
2: **for** i in $(0, 1, ..., n-1)$ **do**
3: $samples \leftarrow x$ negative sequence samples
4: $word \leftarrow$ most frequently occuring word in $samples$
5: $Add\ word$ to $wordSet$
6: return wordSet

General Idea: This algorithm was designed to speed up and improve Algorithm 1. Start with empty word set, Collect x negative sequences and process, find the word and index pair that occurs most often, add this word to the word set. Repeat n times.

3 Algorithm 3: Dynamic Programming

Algorithm 3 Title

```
dp ← [L − k][n]
for i in (0, 1, ..., n − 1) do
    dp[0, i] = opt(0, i)
for i in (1, 2, ..., L − k − 1) do
    for j in (0, 1, ..., n − 1) do
        maxSet ← 0
        maxProb ← 0
        for k in (0, 1, ..., j) do
            temp = dp(i − 1, k) + opt(i, j − k)
            if probability(temp) > maxProb then
                maxSet = temp
                maxProb = probability(temp)
        dp[i, j] = maxSet
return dp[L − k][n]
```

General Idea: Solve for best n words at each position in sequence. Proceed to fill dynamic programming table according to recurrence relation,

$$dp[i, j] = \begin{cases} opt(i, j), & \text{if } i = 0 \\ \underset{1 \leq k \leq j}{max}\ dp[i-1][k] + opt[i, j-k], & \text{if } i \neq 0 \end{cases}$$