Grad AI. 15-780 Fall, 2006

Homework 5 Solutions

- *Homework deadline: 10:30am on Dec. 6*
- *Please do one of the following two problems (Computational Game Theory or Computational Biology).* (Students who do both problems will receive 40% bonus credit for the problem on which they achieve a lower score).
- *Please print your code and hand it in with the hard copy of your homework. Also send a copy of your code by e-mail to both TAs.*

1. **Computational Game Theory [50 pts]**. Note: you only need to do one of the two problems on this homework (see title). In this problem you will implement and test the support-enumeration algorithm for finding a Nash equilibrium in a general sum two-player normal form game (NFG). Recall that a two-player NFG with m row player strategies and n column player strategies is described by two $m \times n$ matrices (or equivalently one $m \times n$ matrix with two entries per cell). A probability vector p is a mixed row (column) player strategy if it specifies the probability with which the row (column) player will play each of his strategies. A Nash equilibrium is described by a mapping of mixed strategies to players such that no player has an incentive to change his mixture.

   One technique for finding a Nash equilibrium is the enumeration and checking of possible supports using a simple feasibility program (the program is linear for two players). The supports of the equilibrium are the pure strategies for both players with positive probability. The *size* of the equilibrium supports is considered the total number strategies in the supports of both players.

   (a) **[30 pts]** Programming problem. A few students mistakenly thought that this required solving two linear programs for each set of supports, one for the row player and one for the column player. Note that the equations given in lecture for formalizing the linear feasibility problem involved solving a single program for both players simultaneously.

   (b) **[5 pts]** Also programming. This should be as simple as removing a break statement in your code from part (a).

   (c) **[5 pts]**

   - **Game 1:** Two equilibria of size $k = 4$ where players mix over their inside two and outside two strategies evenly.
   - **Game 2:** Only one equilibrium of size $k = 6$ where both players mix evenly over their first three strategies.

(d) **[10 pts]** Your histogram should reveal that nearly 70% of all random games have pure-strategy equilibria and equilibria of size 4. You should also notice that the games tend to have balanced equilibria with even support sizes. This makes the support-enumeration a very good choice for finding a single NE in random games (since it will likely only require enumeration up to size k = 4).

2. **Computational Biology [50 pts]** Note: you only need to do one of the two problems on this homework (see title). In class we discussed algorithms for finding the best global alignment between two DNA sequences. In this problem we ask you to consider the task of finding the best *local alignment* between two sequences.

   We will use a scoring function that assigns +2 points when two nucleotides match, -3 when they are aligned even though they do not match and -2 for a gap (d). The best local alignment between two sequences is the two subsequences (continuous subsections for each of the two sequences) with the highest alignment score (or the empty sequence if the two sequences share no nucleotides).

   For example consider the following two sequences:

   AATGCCG
   CGCCATG

   The best local alignment between these two sequences is "GCC" from positions $4-6$ in the first string and $2-4$ in the second, which scores 6 points.

   (a) **[5 pts]** What is the best local alignment and its score for the following two sequeces:

   AATT**AGCTCCTA**C
   CC**AGCACCTA**TG

   The best local alignment is AGC[T,A]CCTA with score 11 (note that [T,A] indicates a mis-match in the explanation) from places 5 - 13 for the first sequence and 3 - 10 for the second.

   (b) **[20 pts]** The Needham-Wuncsh algorithm can be extended to find local alignments using the following modifications:

   - The value of cell i,j in the dynamic programming array should be allowed to take on the value of 0 if it would otherwise become negative. The zero value is symbolizes starting the sub-sequence at the next nucleotide:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + s(x_i, x_j) \\ F(i-1,j) - d \\ F(i,j-1) - d \\ 0 \end{cases}$$

- After filling in the values of all cells you must search through the entire table (rather than only the outer row and column) to find the largest score value. This will indicate the end of the best local alignment match.
- Finally, to recover the actual alignment you must retrace backward from the largest score value to just before the first zero encountered or the beginning of the table (which ever happens first).

(c) **[3 pts]** A good quantity is the ratio between the prior probability and the model's probability prediction of the sequence:

$$\rho(S) \quad = \quad \frac{P(S \mid M)}{P(S)}$$

(d) **[22 pts]** To compute the best local alignment of a particular sub-sequence s of any length we can modify our HMM to properly compute $P(s \mid M)$ for any local alignment to the family represented by M by adding the following to M:

- An equal probability transition from the start state to all insertion and consensus states (unless you have reason to believe that some are more likely than others). Note that we do not allow transitions from the start state to deletion states since they do not require observations and we need to enforce that at least one observation is considered.
- An end state that emits an end character with probability 1.
- Transitions from every consensus and insert state to the end state (with arbitrary but equal probability).
- All transitions other than those to the end state should be scaled down appropriately in order to ensure all states have transitions with probabilities that sum to 1.

Once we have modified the HMM for each family we take our sub-sequence s and append the end character. For each modified family HMM, M_i, we use the Viterbi algorithm to compute the maximum likelihood of any local alignment for S given M_i, $P(S \mid M_i)$. The model that gives us the highest likelihood provides the family with the best local match to the sub-sequence.

Now we can take a full sequence S and try all possible subsequences (there are $O(n^2)$ such sub-sequences) and use the technique above for each of them. However, before comparing the likelihood of sub-sequences of different length we must normalize them with the background model as we did in part (c). The final sub-sequence and family with the best local alignment out of all will be given by,

$$(s^*, i^*) = \arg\max_{s \in S, i} \frac{P(s \mid M_i)}{P(s)}$$