Algorithms Seminar Session 2

Scribe: Sichen Zhong

9/5/14

Prefix Matching Problem (Continued)

Claim: Prefix Matching is NP-Complete.

Proof: We reduce Vertex Cover to Prefix Matching. Given a simply connected graph $G(V, E)$, first map each vertex to a different letter. For example, if $V = \{v_1, v_2, v_3, v_4\}$, then map $v_1 \to A$, $v_2 \to B$, $v_3 \to C$, $v_4 \to D$. If there exists an edge between 2 vertices v_i and v_j, then this edge corresponds to the length 2 string the 2 vertices are mapped to. This completes our initial transformation of a vertex cover problem to a prefix matching problem.

We note the following equation:

$$|PrefixMatchings| + |VertexCover| = |E| \tag{1}$$

Hence, if there exists a polynomial time algorithm to solve the Vertex Cover problem, then we can also find the number of prefix matchings in polynomial time. It follows that Prefix Matching is NP-Complete.

Prefix Matching 2-approximation

Algorithm:

1) For a given set of n strings, construct the following complete graph K_n. Each node represents the forward and reverse orientations of a particular string. The edge weight between any pair of nodes is the greatest prefix matching between all 4 permutations of the reverse and forward orientations of the 2 strings.

2) Use Edmonds (or any other polynomial time matching algorithm) algorithm to find a maximum weight matching for K_n.

Notes:

- For the above approximation, we do not care about maximizing the prefix matchings for strings in lines 2 and 3, lines 4 and 5, etc.... In the optimal solution, the total prefix matching for lines 2 and 3, 4 and 5, etc... cannot exceed the total prefix matching of lines 1 and 2, 3, and 4 etc... in our approximation. Hence, our algorithm is a 2-approximation.

- The above method captures the best pairs of strings with the most prefix matchings in lines 1 and 2, lines 3 and 4, etc...

Other Potential Approximation Methods

Algorithm: Minimizing unmatched characters

- For a given set of n strings, construct a complete graph K_n for the **1 of a set TSP** problem. The 1 of a set TSP problem is the standard TSP problem except that each node is a set consisting of the forward and reverse orientations of a string. The edge weights change depending on which orientation we choose for a node. In this case, the edge weights are the costs of non-overlap of the orientations we choose for 2 nodes.

- This transformation preserves distance properties. Namely, the edge weights satisfy the properties of a metric (symmetry, triangle inequality, $d(x, y) = 0$ iff $x = y$ for any 2 vertices x and y)

- The 1 of a set TSP problem with metric-preserving properties has a 3-approximation due to LP-Rounding techniques.

Open Question 1 For a finite small alphabet, what is the complexity of the Prefix Matching Problem?

Open Question 2 For the above 2-approximation, can we improve the approximation constant by recursing the algorithm on matched pairs of strings?