Massachusetts Institute of Technology
18.434: Seminar in Theoretical Computer Science

Lecturer: Lele Yu
February 16, 2006

Lecture notes on Shortest Superstring Problem

So far we have studied the set covering problem, but not looked at any real life applications. The shortest superstring problem takes as input, several strings of different lengths and finds the shortest string that contains all the input strings as substrings. This is helpful in the genome project since it will allow researchers to determine entire coding regions from a collection of fragmented sections. We will look at the set covering problem as a way to solve the shortest superstring problem.

1 The Shortest Superstring Problem

Given an alphabet $\sum$ and a set of n strings, $S = \{s_1, ..., s_n\} \subseteq \sum$, we want to find the shortest string s that contains $s_i \forall i$, as a substring. We assume that no $s_i \in S$ is a substring of $s_j \in S$. This problem is NP-hard.

The Greedy Algorithm Approach Let T be equal to the the set of strings in the input. Take the overlap of two strings s and t to be the longest suffix of s that is the same as a prefix of t. This algorithm iterates through T, selecting two strings with maximum overlap and replaces them with a combined string. After n-1 iterations, T will contain only one string, the superstring of the strings previously in T.

The problem with this estimator is that it has an approximation factor of 2 at best. This can be demonstrated through looking at the strings 124242, 242425 and 2424242. There is the same amount of overlap between all the strings. Say the algorithm first picked the first two strings to merge, then the resulting superstring would be 12424252424242. However, the best answer would have been 124242425.

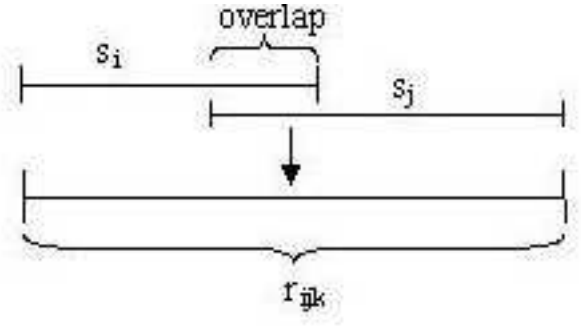

Figure 1: A combined string, taking the largest overlap of s_i with s_j.

The Set Cover Algorithm Approach Using the set cover method, we obtain a $2H_n$ factor approximation algorithm. Given input, $S = \{s_1, ..., s_n\}$, we construct a string r_{ijk} for all possible combinations s_i and $s_j \in S$ (where k is the maximum overlap between the two). The figure above shows a possible combination. Now, let's call the set of all such r, R. Now let v be $\in \sum$, then let sub$(v) = \{s \in S|$ s is a substring of $v\}$. All possible subsets of S is sub(v) for all $v \in S \cup R$.

Algorithm 1 (Set cover)

1. Use the greedy set cover algorithm to find a cover for the instance C.
2. Backwards construct $v_1, ...v_k$ from the sets selected by the algorithm so that sub$(v_1) \cup ... \cup$ sub(v_k) is the cover for C.
3. Uniting the strings $v_1, ...v_k$ gives the shortest superstring via set cover.

Lemma 1 $OPT \leq OPT_C \leq 2 \cdot OPT$

Proof: In this case, OPT_C is the cost of a set cover solution. To show the first part of the inequality, assume that we have an optimal set cover $\{sub(v_{i_j})|1 \leq j \leq l\}$. Then, since every string in S is a substring of some v_{i_j}, $1 \leq j \leq l$, every string is in S is also a substring of the result from the set cover algorithm. Therefore, $OPT_C \geq OPT$.

To prove the second part, we keep in mind that each string in the set cover must start and end in different places from the rest of the strings since none of the strings are substrings of another string. We can then partition the strings of the set cover into groups that are contained by $v_1, ..., v_k$. The strings are split such that the last string in group v_i will be the furthest overlapping string to the first string the group v_i. This system will ensure that r_i cannot overlap with v_{i+2}. If a string in v_{i+2} overlaps with v_i, then it would definitely overlap with the first string in v_{i+1} and thus should be in v_{i+1}. Hence, in the worst case, every other group completely overlaps and we have that $OPT_C \leq 2 \cdot OPT$. A diagram of this is shown on the next page.

Theorem 2 *The greedy algorithm is an H_n factor approximation algorithm for the minimum set cover problem.*

We learned this theorem in the first lecture of the semester. Together, the lemma and the theorem show that the set cover algorithm gives a $2H_n$ factor algorithm for the shortest superstring problem.

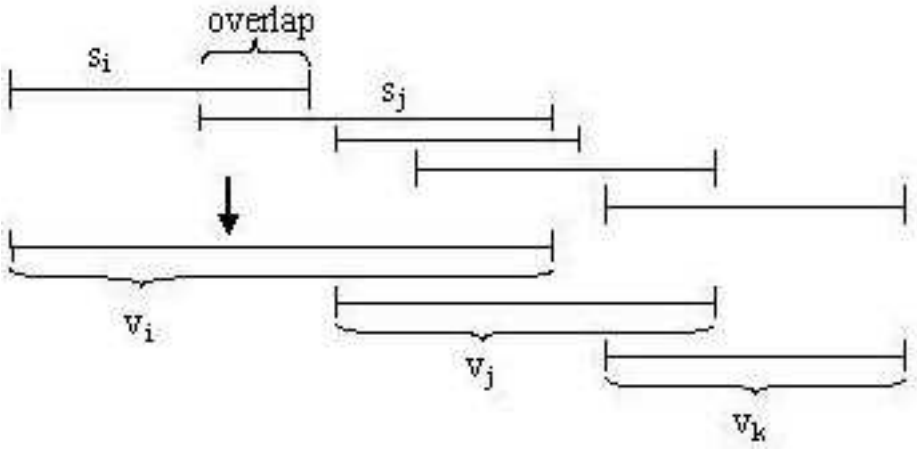

Figure 2: Strings v_i, v_j and v_k backwards constructed by their substrings.

2 A Factor 4 Algorithm

Definition 1 $OPT = |prefix(s_1, s_2)| + |prefix(s_2, s_3)| + ... + |prefix(s_n, s_1)| + |overlap(s_n, s_1)|$

Below is a graphical representation of the shortest superstring in this definition with relation to the input strings. The addition of the overlap at the end in effect just appends all of the last(in terms of occurance in the shortest superstring) string to the shortest superstring.

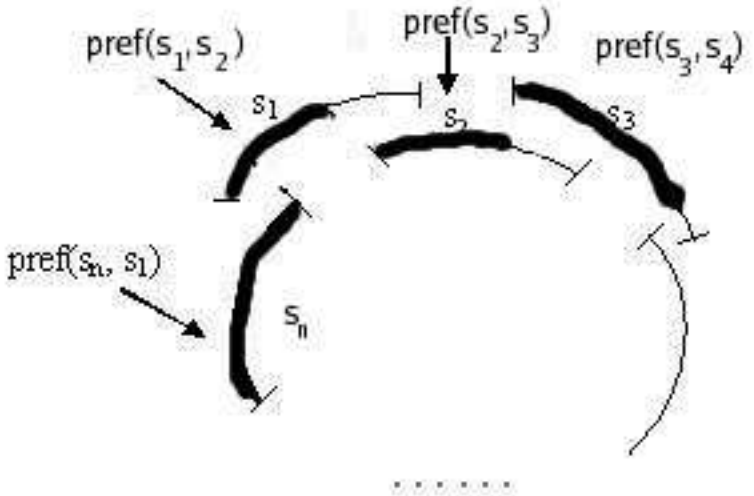

Figure 3: A graphical representation of OPT.

Including the last overlap ensures that OPT covers all the input strings even if all of them share large overlaps with every other one of them. For example, if we started wih the input, $S = \{121212, 212121\}$. Then, the shortest superstring would be $s = 2121212$. However, if we just take the prefix of each string with the next one, we would only get 1 from the prefix of s_1 with s_2 and 2 from the prefix of s_2 with s_1, hence the OPT would only be 12, which obviously does not cover either one of the strings.

By this definition, the problem of the shortest superstring becomes an instance of the traveling salesman problem with prefix(s_i,s_j) as the weight between vertices i and j. As we recall, the travelling salesman problem is one of finding the lowest cost cycle travelling through all the vertices in a graph. However, since TSP is not easily computable, we consider travelling through all the vertices using multiple cycles, i.e., finding the cycle cover of this prefix graph. Since the tour above in the figure is a cycle cover, the minimum weight of a cycle cover is the lower bound of OPT.

We can compute a minimum weight cycle cover in polynomial time by constructing a bipartite graph with $U = \{u_1, ..., u_n\}$ and $V = \{v_1, ..., v_n\}$ as the two vertex sets. For every two vertices s_i and s_j, we add an edge with weight $|prefix(s_i, s_j)|$. A cycle cover of the graph corresponds to a perfect matching of H.

If c_i is a cycle in the prefix graph, than we can define wt(c_i), the weight of c_i to be:

$$wt(c_i) = prefix(s_{i_1},s_{i_2}) \circ ... \circ prefix(s_{i_l},s_{i_1})$$

Also, let

$$\sigma(c_i) = wt(c_i) \circ s_{i_1}$$

Together, we know that each string, s_{i_1},...,s_{i_l} is a substring of both $(wt(c_i))^\infty$ and $\sigma(c_i)$.

Algorithm 2

1. Construct a prefix graph and find its minimum weight cycle cover, $C = \{c_1, ..., c_k\}$.
2. Return $\sigma(c_1) \circ \sigma(c_2) \circ ... \circ \sigma(c_k)$.

Theorem 3 *Algorithm 2 achieves an approximation factor of 4 for the shortest superstring problem*

Proof: Let $wt(C) = \sum_{i=1}^{k} wt(c_i)$, then

$$\sum_{i=1}^{k} |\sigma(c_i)| = wt(C) + \sum_{i=1}^{k} |s_i|$$

where s_i is the representative string that is concatenated to $wt(c_i)$ in $\sigma(c_i)$. We already know that $wt(C) \leq OPT$ and now we rearrange to find $\sum_{i=1}^{k} |s_i|$.Using lemma 4, we find:

$$OPT \geq \sum_{i=1}^{k} |s_i| - \sum_{i=1}^{k-1} |overlap(s_i, s_{i+1})| \geq \sum_{i=1}^{k} |s_i| - 2\sum_{i=1}^{k} wt(c_i)$$

The leftmost part of this inequality is valid because the right side of the inequality has the same issue as $wt(c)$ from before; it does not take into account parts of the shortest superstring that is shared amongst all the s_i. The right-most equation is true due to Lemma 4.

Hence,

$$\sum_{i=1}^{k} |r_i| \leq OPT + 2\sum_{i=1}^{k} wt(c_i) \leq 3 \cdot OPT$$

When we substitute this into the original equation, we get that:

$$\sum_{i=1}^{k} |\sigma(c_i)| = 4 \cdot OPT$$

Lemma 4 *Take C, the minimum weight cycle cover of S. Let c and c' be two cycles in C, and let r and r' be representative strings from those cycles. Then $|overlap(r, r') \leq wt(c) + wt(c')$*

Proof: By contradiction. Assume that $|overlap(r, r')| \geq wt(c)+wt(c')$. Then $overlap(r, r')$ is a prefix of α^∞ and α'^∞. In addition, since α is $\in overlap(r, r')$, α is a prefix of $(\alpha')^\infty$ and vice versa. Hence, $\alpha \circ \alpha' = \alpha' \circ \alpha$. This means that $(\alpha)^k(\alpha')^k = (\alpha)^{k-1}(\alpha')(\alpha)(\alpha')^{k-1}$ and $(\alpha)^k(\alpha')^k = (\alpha')^k(\alpha)^k$. Hence, $\alpha^\infty = \alpha'^\infty$. By lemma 5, there exists a cycle of weight wt(c) that covers all strings in c and c' Hence, C cannot be a minimum weight cycle.

Lemma 5 *If each string in S' is a substring of t^∞ then there is a cycle of weight at most $|t|$ in the prefix graph covering all the vertices corresponding to strings in S'*

Proof: Align the strings of S' along t^∞. They should all be contained within t and the weight of the cycle in the prefix graph visiting them all in this order should be at most $|t|$.