1 Recap: Minimum Set Cover

Recall the (Weighted) Set Cover problem, defined as follows.

Set Cover Problem (SC): Given a universe X of elements, and a collection F of subsets $S \subset X$, where each $S \in F$ has an associated non-negative cost, find a subcollection $C \subset F$ of minimum total cost that covers X, assuming one exists.

Valika showed us last time that although this problem is NP-hard to solve exactly, it can be approximated to a factor of $H_d = \log d + O(1)$, where $d = \max\{|S| : S \in F\}$, at least for the special case where $\text{cost}(S) = 1 \ \forall S \in F$. In fact, the algorithm and analysis generalizes quite naturally to the weighted problem, but I will give an alternate, perhaps more intuitive, proof of the approximation factor anyway. First, the algorithm is as follows.

Algorithm 1: GreedySetCover(X, F)

```
1 C ⟵ ∅
2 U ⟵ X
3 while U ≠ ∅ do
4 |   Find set S ∈ F \ C that minimizes α := cost(S)/(S∩U).
5 |   for each x ∈ S ∩ U do
6 |   |   price(x) ⟵ α
7 |   C ⟵ C ∪ {S}
8 |   U ⟵ U \ S
9 return C
```

Note the only 2 modifications in this algorithm from the one Valika presented yesterday, namely, the minimized quantity α in each iteration is now $\frac{\text{cost}(S)}{S \cap U}$ instead of simply $\frac{1}{S \cap U}$, and we incorporate this price associated with each element x covered for the first time into the algorithm (strictly to aid in the analysis).

Now, we will show the slightly weaker bound than yesterday, that is, that GreedySetCover is an H_n-approximation for Set Cover, where $n = |X|$. Observe that the cost of the returned solution C is precisely the total assigned price of each element in X, $\sum_{x \in X} \text{price}(x)$. If we order the elements of X as $x_1, x_2, \ldots, x_n$ by the order in which they were covered by the algorithm, breaking ties arbitrarily, then we can write this total cost as $\sum_{k=1}^{n} \text{price}(x_k)$. In order to show $\sum_{k=1}^{n} \text{price}(x_k) \leq H_n \, \text{OPT}$, it thus suffices to prove the following lemma.

Lemma: For each $k \in \{1, 2, \ldots, n\}$, $\text{price}(x_k) \leq \frac{\text{OPT}}{n-k+1}$, where OPT is the cost of the optimal cover.

Proof: Consider the iteration during which x_k is covered. At the beginning of the iteration, U contains all the elements as yet uncovered, of which there are at least $n - k + 1$. Now, the optimal cover covers all of X, so in particular it certainly covers U. This implies that there exists a set that achieves $\alpha \leq \frac{\text{OPT}}{|U|}$. Why is this?

Imagine, for this and future iterations, we choose sets from the optimal cover instead of minimizing α. If we maintain element prices as usual, then 1 of the elements $x_0 \in U$ must have

price$(x_0) \leq \frac{\text{OPT}}{|U|}$, since otherwise the total tally over the optimal cover will end up being > OPT, which is absurd. But then the set S that covered x_0 is precisely the one we're looking for, since its α can only increase over iterations as fewer and fewer elements become available over which it can distribute its cost.

So coming back to our original algorithm, the existence of a set with $\alpha \leq \frac{\text{OPT}}{|U|}$ means that since we take the set that *minimizes* α, the set we end up selecting during the current iteration must also have $\alpha \leq \frac{\text{OPT}}{|U|}$. But α is the value we assign to price(x_k), so we have price$(x_k) \leq \frac{\text{OPT}}{|U|}$, where $|U| \geq n - k + 1$, which gives us our lemma. ■

This gives us the following theorem.

Theorem: GreedySetCover is an H_n-approximation algorithm for the Set Cover problem.

2 Today: Shortest Superstring

So now we move on to our main topic of today. We will see an application of the Vertex Cover approximation to in turn approximate a seemingly unrelated problem, namely Shortest Superstring (SS). This is not the best approximation known for SS, but it is nonetheless an interesting reduction.

Applications of SS include DNA analysis and data compression. A strand of human DNA can be viewed as a long string over a 4-letter alphabet. Typically, only short substrings at a time can be read from arbitrary and unknown positions in the long strand, many of which may overlap, and it is conjectured that the shortest DNA string that contains all the read segments as substrings is a good approximation of the actual DNA strand. In data compression, instead of sending/storing a lot of strings independently, we can store a single shortest superstring, together with beginning and ending positions in it for each substring.

2.1 Definition

Shortest Superstring Problem (SS): Given a finite alphabet Σ and a set of n strings $S = \{s_1, s_2, \ldots, s_n\} \subset \Sigma^*$, find a shortest string $s \in \Sigma^*$ that contains s_i as a substring for each $i = 1, 2, \ldots, n$. WLOG, assume no s_i is a substring of s_j, for $i \neq j$.

This problem is NP-hard, and a simple greedy algorithm for it (which I nevertheless don't have time/space to describe) is conjectured to be a 2-approximation. I guess that means this is still an open problem, at least at the time the book was written. We will instead use the H_n-approximation of SC above to obtain a $2H_n$-approximation of SS.

2.2 The Algorithm

Given an instance $S \subset \Sigma^*$ of SS, we wish to construct a corresponding instance (X, F) of SC. In the SS problem, the set we need to 'cover,' in some sense, is the set S of strings. So let our universe X of elements that need to be covered in the SC problem be S. Now, how do we associate a set set$(\sigma) \in F$ with a string $\sigma \in \Sigma^*$ so that set(σ) covers a string $\tau \in X = S$ if and

only if τ is a substring of σ? We could define it to be the set of all substrings of σ, but since we want to limit our sets to subsets of $X = S$, we will define it as follows:

$$\text{set}(\sigma) := \{\tau \in S : \tau \text{ is a substring of } \sigma\} \tag{1}$$

A set cover, then, will be a collection of such sets set(σ), from which we derive a superstring of S by concatenating all the σ's together. However, we can't define F to be the collection of set(σ)'s for all $\sigma \in \Sigma^*$, since F needs to be finite. On the other hand, we can't limit the σ's to just S, since the only superstring we would then get is the concatenation of all strings in S, a not very useful solution. To strike a balance, we wish the set of σ's to include various superstrings of every *pair* of strings in S. To be precise, let us pick an arbitrary order $\{s_1, s_2, \ldots, s_n\}$ of S. Then for strings $s_i, s_j \in S$, if the last $k > 0$ symbols of s_i are the same as the first k symbols of s_j, let σ_{ijk} denote the string obtained by overlapping these k symbols of s_i and s_j. Let I then be the set of σ_{ijk}'s for all valid choices of i, j, k, that is, the set of all 'good' superstrings of pairs of strings in S. We can now define F as $\{\text{set}(\sigma) : \sigma \in S \cup I\}$, and the associated cost of each set set(σ) is simply the length of σ, that is, $|\sigma|$. Based on this, we can now write down the algorithm for SS as follows.

Algorithm 2: ShortestSuperstring(S)

1 Compute the instance (X, F) of SC as described above.
2 Let $\{\text{set}(\sigma_1), \text{set}(\sigma_2), \ldots, \text{set}(\sigma_k)\}$ be the collection of sets returned by GreedySetCover(X, F).
3 **return** $s := \sigma_1 \cdot \sigma_2 \cdot \cdots \cdot \sigma_k$

2.3 Example

The following is a simple example of the reduction. Consider the simplest alphabet $\Sigma = \{0, 1\}$, over which we have the SS problem instance $S = \{s_1 = 001, s_2 = 01101, s_3 = 010\}$. Then for the string $11010010 \in \Sigma^*$, for instance, we have set$(11010010) = \{s_1 = 001, s_3 = 010\}$. Next, we find I to be

$$I = \{\sigma_{122} = 001101, \sigma_{132} = 0010, \sigma_{232} = 011010, \sigma_{311} = 01001, \sigma_{321} = 0101101\} \tag{2}$$

And finally, we have the SC instance (X, F), with $X = S$,

$$F = \{\{s_1\}, \{s_2\}, \{s_3\}, \text{set}(\sigma_{122}), \text{set}(\sigma_{132}), \text{set}(\sigma_{232}), \text{set}(\sigma_{311}), \text{set}(\sigma_{321})\} \tag{3}$$

and set costs cost$(\{s_1\}) = |s_1| = 3$ and cost$(\text{set}(\sigma_{122})) = |\sigma_{122}| = 6$ as representative examples.

2.4 The Analysis

It is clear that this is a polynomial time reduction, and that ShortestSuperstring gives *some* superstring of S. Since we know that GreedySetCover is an H_n-approximation for SC, in order to show that ShortestSuperstring is a $2H_n$-approximation for SS it suffices to prove the following lemma.

Lemma: Let OPT_{SC} denote the cost of an optimal solution to the SS instance (X, F), and OPT_{SS} denote the length of the shortest superstring of S. Then $\text{OPT}_{SC} \leq 2 \times \text{OPT}_{SS}$.

Proof: It suffices to exhibit *some* set cover of cost $\leq 2 \times \text{OPT}_{SS}$. Let s be a shortest superstring of S, that is, one of length OPT_{SS}, and let $S = \{s_1, s_2, \ldots, s_n\}$ be ordered by each string's leftmost occurrence in s. For the rest of the proof, when we talk about strings in S we will be referring to this ordering and to that particular leftmost occurrence. It helps to follow the proof with Figure 1 for illustration.

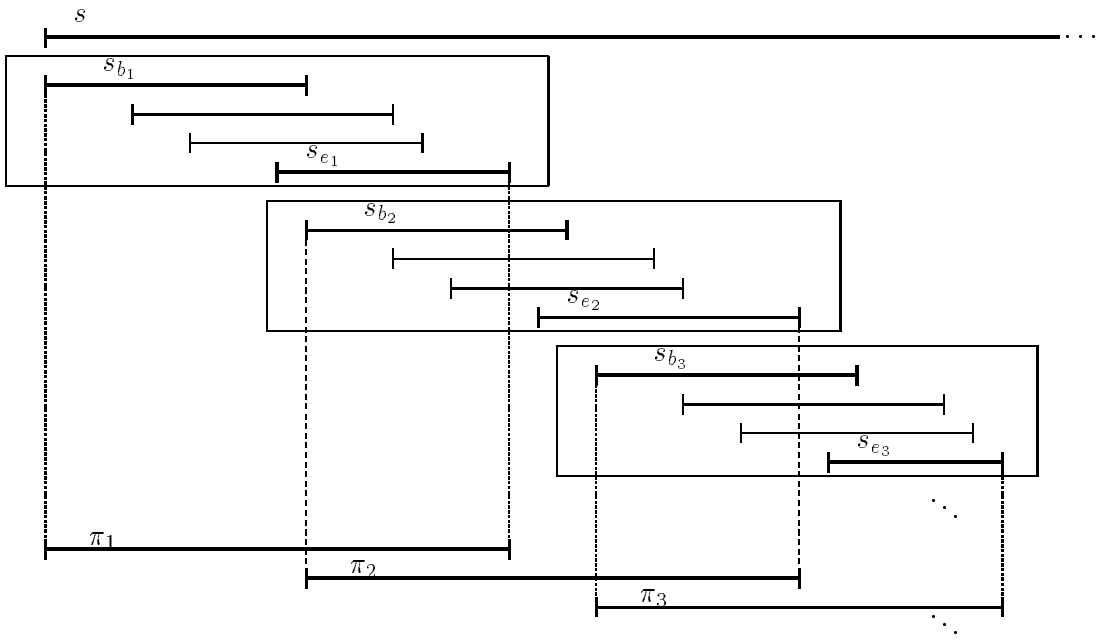

Figure 1: Partitioning and recovering of strings in S within shortest superstring.

Note that since no string in S is a substring of another, for all $i < j$, s_i must start before and end before s_j. We will partition the ordered list $s_1, \ldots, s_n$ into groups as follows. Denote by b_i and e_i the indices of the first and last string in the i^{th} group. We let $b_1 = 1$ and e_1 be the highest index such that s_{e_1} still overlaps with s_{b_1}. Then $b_2 = e_1 + 1$, and e_2 is the highest index such that s_{e_2} still overlaps with s_{b_2}, and so on, until we eventually have $e_t = n$.

Now, for each $i \in \{1, \ldots, t\}$, by definition s_{b_i} must overlap s_{e_i} by some k_i number of symbols. So let $\pi_i = \sigma_{b_i e_i k_i}$. Clearly, π_i 'covers' s_j for $b_i \leq j \leq e_i$, so that $\{\text{set}(\pi_i) : i \in \{1, \ldots, t\}\}$ is a set cover of the SS instance (X, F). We now make the final and key observation that each symbol in s is 'covered' by at most 2 of the π_i's. Why is this? Consider any $i \in \{1, \ldots, t-2\}$ and we will show that π_i cannot overlap π_{i+2}. This is equivalent to saying s_{e_i} does not overlap $s_{b_{i+2}}$. But we know that s_{e_i} must *end* before $s_{b_{i+1}}$ *ends*, and by construction $s_{b_{i+2}}$ must *start* after $s_{b_{i+2}}$ *ends*, so s_{e_i} certainly cannot overlap $s_{b_{i+2}}$. It follows that this set cover that we just found has a cost of $\sum_{1 \leq i \leq t} |\pi_i| \leq 2 \times \text{OPT}_{SS}$, completing our proof. ■

This gives us the following theorem.

Theorem: ShortestSuperstring is a $2H_n$-approximation algorithm for the Shortest Superstring problem.