1 Approximating Set Cover

1.1 Definition

An Instance (X, F) of the set-covering problem consists of a finite set X and a family F of subset of X, such that every elemennt of X belongs to at least one subset of F :

$$X = \bigcup_{S \in F} S$$

We say that a subset $S \in F$ covers all elements in X. Our goal is to find a minimum size subset $C \subseteq F$ whose members cover all of X.

$$X = \bigcup_{S \in C} S \tag{1}$$

The cost of the set-covering is the size of C, which defines as the number of sets it contains, and we want $|C|$ to be minimum. An example of set-covering is shown in Figure 1. In this Figure, the minimum size set cover is $C = \{T_3, T_4, T_5\}$ and it has the size of 3.

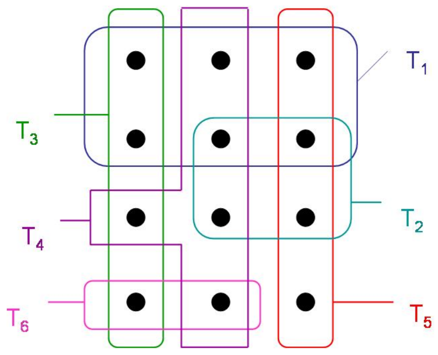

Figure 1: An instance (X, F) of set-covering problem. Here, X consists of 12 vertices and $F = \{T_1, T_2, T_3, T_4, T_5, T_6\}$. A minimum-size set cover is $C = \{S_3, S_4, S_5\}$. The greedy algorithm produces set cover of size 4 by selecting the sets S_1, S_4, S_5, S_3 in order.

Set-covering problem is a model for many resource covering problems. As mentioned earlier in the previous lecture, set-covering is an NP-Hard problem. We will now examine a greedy algorithm that gives logarithmic approximation solution.

1.2 A Greedy Approximation Algorithm

Idea: At each stage, the greedy algorithm picks the set $S \in F$ that covers the greatest numbers of elements not yet covered.
For the example in Figure 1, the greedy algorithm will first pick T_1 because T_1 covers the maximum number of uncovered elements, which is 6. Then, it will pick T_4 since it covers maximum number uncovered elements, which is 3, leaving 3 more elements uncovered. Then it will select T_5 and T_3 , which cover 2 and 1 uncovered elements, respectively. At this point, every element in X will be covered.
By greedy algorithm, $C = \{T_1, T_4, T_5, T_3\} \;\Rightarrow\; cost = |C| = 4$

Where Optimum solution, $C = \{T_3, T_4, T_5\} \;\Rightarrow\; cost \;=\; |C| \;=3$

Algorithm 1: GREEDY-SET-COVER (X, F)

```
1 $U \leftarrow X$
2 $C \leftarrow \emptyset$
3 While $U \neq 0$
4   do select an $S \in F$ that maximizes $|S \cap U|$
5     $U \leftarrow U - S$
6     $C \leftarrow C \cup \{S\}$
7 return $C$
```

The description of this algorithm are as following. First, start with an empty set C. Let C contains a cover being constructed. Let U contain, at each stage, the set of remaining uncovered elements. While there exists remaining uncovered elements, choose the set S from F that covers as many uncovered elements as possible, put that set in C and remove these covered elements from U. When all element are covered, C contains a subfamily of F that covers X and the algorithm terminates.

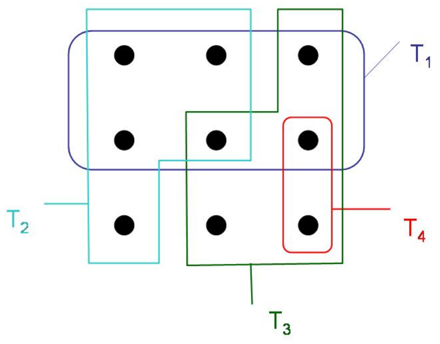

Figure 2: Another instance (X, F) of set-covering problem. Here, X consists of 9 vertices and $F = \{T_1, T_2, T_3, T_4\}$. The greedy algorithm produces set cover of size 3 by selecting the sets T_1, T_3 and T_2 in order.

1.3 Analysis Of Greedy-Set-Cover

Theorem: GREEDY-SET-COVER is a polynomial time $\alpha - approximation$ algorithm, where

$$\alpha \;=\; H\,(\,max\,\{|S|\,:\,S \in F\}) \tag{2}$$

and $H(d) =$ the d^{th} harmonic number, which is equal to $1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{d} = \sum_{i=1}^{d} \frac{1}{i} = log\,d + O(1)$ (from equation (A.7) in Appendix A).
From the previous lectures, we have

$$\frac{C(T)}{C(OPT)} = \alpha \tag{3}$$

where $C(T) =$ the solution of GREEDY-SET-COVER algorithm $= |C|$
$C(OPT) =$ the optimal solution $= |C^*|$

So we have

$$\frac{|C|}{|C^*|} = \alpha \tag{4}$$

this says that the Greedy-Set-Cover produces the set cover C that has the size not more than α times of that of the optimal solution. Combining equation (2) and (4), we get

$$\frac{|C|}{|C^*|} = H\,(\,max\,\{|S|\,:\,S \in F\}) \tag{5}$$

Equation (2) and (5) are equal. So if we prove equation (5), we prove the theorem.

Proof: From now on, we refer to Greedy-Set-Cover algorithm as "the algorithm". To prove this algorithm, we assign a *price* of 1 to each set $S \in F$ selected by the algorithm and distribute this price over the elements covered for the first time. Let S_i denote the i^{th} subset selected by the algorithm at i^{th} iteration. Let c_x be the price allocated to element $x \in X$, that is covered for the first time at i^{th} iteration.

$$c_x \;=\; \frac{1}{|\,S_i - (\,S_1 \cup S_2 \cup ... \cup S_{i-1})\,|} \tag{6}$$

Notice that $|\,S_i - (\,S_1 \cup S_2 \cup ... \cup S_{i-1})\,|$ is the number of elements covered for the first time by S_i at i^{th} iteration. For example, from Figure 2, the price allocated to each element is calculated as follows. At the 1^{st} iteration, there are 6 elements covered for the first time by T_1. Therefore, the price of each of those 6 elements is $c_x = \frac{1}{|T_1 - \emptyset|} = \frac{1}{6}$. In 2^{nd} iteration, T_3 covers 2 new elements, so the price for each of those elements is $c_x = \frac{1}{|T_3 - T_1|} = \frac{1}{2}$. Finally, in 3^{rd} iteration, T_2 covers 1 more new element, so the price for this is obviously 1 and can be calculated from $c_x = \frac{1}{|T_2 - (T_1 \cup T_3)|} = \frac{1}{1}$
At each iteration of the algorithm, 1 unit price is assigned and one of set $S \in F$ is also added to the cover C, so the cost of the algorithm equals to the price of universe, $\sum_{x \in X} c_x$.

$$|C| \;=\; \sum_{x \in X} c_x \tag{7}$$

Notice that the optimal solution is intractable, so in order to find out how our greedy algorithm does in terms of performance compare to the optimal solution we need to make indirect comparisons by using some reference quantities. Here, the reference quantity is the price of universe, $\sum_{x \in X} c_x$.
The price assigned to the optimal cover is defined as

$$\sum_{S \in C^*} \sum_{x \in S} c_x$$

We need to find a relationship of the price of universe and the price optimal cover. From the optimal cover price defined above, if some sets in the optimal solution overlap each other, the price of the overlapped elements in those sets will be double counted. So we have

$$\sum_{S \in C^*} \sum_{x \in S} c_x \;\geq\; \sum_{x \in X} c_x \tag{8}$$

From equation (7) and (8), we get

$$|C| \quad \leq \quad \sum_{S \in C^*} \sum_{x \in S} c_x \tag{9}$$

Claim:

$$\sum_{x \in S} c_x \quad \leq \quad H\,(\,|S|\,) \tag{10}$$

If the claim is true, from equation (3) we will get

$$\begin{aligned} |C| \quad &\leq \quad \sum_{x \in C*} H\,(\,|S|\,) \\ &\leq \quad |C^*| \,\times\, H\,(\,max\,\{|S| \,:\, S \in F\}) \end{aligned} \tag{11}$$

which is the same as equation (5) and this proves the theorem.
Here, It remains to be shown that our claim in equation (10) is true.
Let S be any set $S \in F$ and $i = 1,\, 2,\, ...,\, |C|$, and let

$$u_i(S) = |S - (S_1 \cup S_2 \cup ... \cup S_i)|$$

be number of elements of S remaining uncovered after the algorithm adding $S_1,\, S_2, ...,\, S_i$ to C at i^{th} iteration. We define

$$u_0(S) = |S|$$

to be the number of all elements in S, which is initially uncovered. Let k be the least index such that

$$u_k(S) = 0$$

so that at k^{th} iteration all element in S are covered for the first time by the sets $S_1,\, S_2, ...,\, S_i$ chosen by the algorithm. Notice that $u_0(S) \geq u_1(S) \geq ... \geq u_{k-1}(S) \geq u_k(S)$. So we have $u_{i-1}(S) \geq u_i(S)$ and $u_{i-1}(S) - u_i(S)$ is the number of elements in S that are covered for the first time by S_i at i^{th}iteration, for $i = i = 1,\, 2,\, ...,\, k$. Thus,

$$\sum_{x \in S} c_x = \sum_{i=1}^{k} (u_{i-1}(S) - u_i(S)) \cdot \frac{1}{|S - (S_1 \cup S_2 \cup ... \cup S_{i-1})|} \tag{12}$$

which is equal to the sum of number of elements of S covered by $S_i$times the price assigned to each element at i^{th} iteration. Observe that

$$\begin{aligned} |S_i - (S_1 \cup S_2 \cup ... \cup S_{i-1})| \quad &\geq \quad |S - (S_1 \cup S_2 \cup ... \cup S_{i-1})| &(13) \\ &= \quad u_{i-1} &(14) \end{aligned}$$

because S_i is the gready choice at i^{th} iteration, so S cannot cover more new elements than S_i does (otherwise, S should be chosen by the algorithm instead of S_i). So by equation (12) and (14), we obtain

$$\sum_{x \in S} c_x \quad \leq \quad \sum_{i=1}^{k} (u_{i-1} - u_i) \cdot \frac{1}{u_{i-1}}$$

We now bound this quantity as follow:

$$\begin{aligned}
\sum_{x\in S} c_x &\le \sum_{i=1}^{k}(u_{i-1}-u_i)\cdot\frac{1}{u_{i-1}} \\
&= \sum_{i=1}^{k}\sum_{j=u_i+1}^{u_i-1}\frac{1}{u_{i-1}} \\
&\le \sum_{i=1}^{k}\sum_{j=u_i+1}^{u_i-1}\frac{1}{j} \\
&= \sum_{i=1}^{k}(\sum_{j=1}^{u_i-1}\frac{1}{j}-\sum_{j=1}^{u_i}\frac{1}{j}) \\
&= \sum_{i=1}^{k}(H(u_{i-1})-H(u_i)) \\
&= H(u_0)-H(u_1)+(H(u_1)-H(u_2)+...+H(u_{k-1})-H(u_k) \\
&= H(u_0)-H(u_k) \\
&= H(u_0)-H(0) \\
&= H(u_0) \\
&= H(|S|)
\end{aligned}$$

which is the same as our claim in equation (10) and so this completes our proof. ■

2 Approximating Shortest Superstring via Set Cover

2.1 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 2: GreedySetCover(X, F)

```
1 $C \longleftarrow \emptyset$
2 $U \longleftarrow X$
3 while $U \neq \emptyset$ do
4     Find set $S \in F \setminus C$ that minimizes $\alpha := \frac{\text{cost}(S)}{S \cap U}$.
5     for each $x \in S \cap U$ do
6         price($x$) $\longleftarrow \alpha$
7     $C \longleftarrow C \cup \{S\}$
8     $U \longleftarrow U \setminus S$
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 $\text{price}(x_0) \leq \frac{\text{OPT}}{|U|}$, since otherwise the total tally over the optimal cover will end up being $> \text{OPT}$, 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 $\text{price}(x_k) \leq \frac{\text{OPT}}{|U|}$, 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.

2.2 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.

2.3 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.

2.4 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 $\text{set}(\sigma) \in F$ with a string $\sigma \in \Sigma^*$ so that $\text{set}(\sigma)$ 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:

$$\text{set}(\sigma) := \{\tau \in S : \tau \text{ is a substring of } \sigma\} \tag{15}$$

A set cover, then, will be a collection of such sets $\text{set}(\sigma)$, 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 $\text{set}(\sigma)$'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 3: 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 $s := \sigma_1 \cdot \sigma_2 \cdot \cdots \cdot \sigma_k$
```

2.5 An 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

$$I = \{\sigma_{122} = 001101, \sigma_{132} = 0010, \sigma_{232} = 011010, \sigma_{311} = 01001, \sigma_{321} = 0101101\} \qquad (16)$$

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

$$F = \{\{s_1\}, \{s_2\}, \{s_3\}, \text{set}(\sigma_{122}), \text{set}(\sigma_{132}), \text{set}(\sigma_{232}), \text{set}(\sigma_{311}), \text{set}(\sigma_{321})\} \qquad (17)$$

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

2.6 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 OPT$_{SC} \leq 2 \times$ 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 3 for illustration.
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 $e_t = n$.

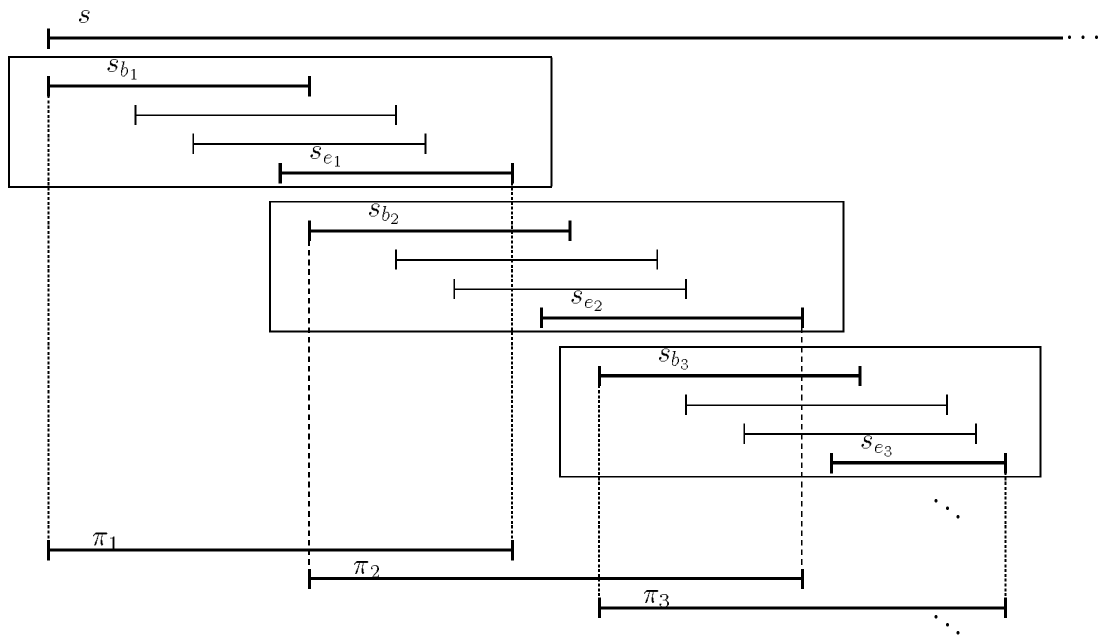

Figure 3: Partitioning and recovering of strings in S within shortest superstring.

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.