Shortest Superstring Problem

Vahan Mkrtchyan

Lane Department of Computer Science and Electrical Engineering
West Virginia University

February 24, 2014

Shortest Superstring Problem

The Formulation of the Problem

Given a finite alphabet Σ,

Shortest Superstring Problem

The Formulation of the Problem

Given a finite alphabet Σ, and a set of n strings $S = \{s_1, ..., s_n\}$.

Shortest Superstring Problem

The Formulation of the Problem

Given a finite alphabet Σ, and a set of n strings $S = \{s_1, ..., s_n\}$. The objective is to find a shortest string that contains each s_i as a substring.

Shortest Superstring Problem

The Formulation of the Problem

Given a finite alphabet Σ, and a set of n strings $S = \{s_1, ..., s_n\}$. The objective is to find a shortest string that contains each s_i as a substring.

Remark

Observe that without loss of generality, we can assume that no string s_i is a substring of s_j for $i \neq j$.

Shortest Superstring Problem

The Formulation of the Problem

Given a finite alphabet Σ, and a set of n strings $S = \{s_1, ..., s_n\}$. The objective is to find a shortest string that contains each s_i as a substring.

Remark

Observe that without loss of generality, we can assume that no string s_i is a substring of s_j for $i \neq j$.

Example

Observe that if $s_1 = ab$ and $s_2 = ba$, then the concatenation gives $s = abba$, while the shortest superstring is $s' = aba$.

Shortest Superstring Problem

The Formulation of the Problem

Given a finite alphabet Σ, and a set of n strings $S = \{s_1, ..., s_n\}$. The objective is to find a shortest string that contains each s_i as a substring.

Remark

Observe that without loss of generality, we can assume that no string s_i is a substring of s_j for $i \neq j$.

Example

Observe that if $s_1 = ab$ and $s_2 = ba$, then the concatenation gives $s = abba$, while the shortest superstring is $s' = aba$.

The complexity of Shortest Superstring Problem

The Shortest Superstring Problem is **NP-hard**, in great contrast with Largest Common Substring problem which can be solved in polynomial time with the help of dynamic programming.

A possible 2-approximation algorithm

The definition of overlap

The overlap of two strings s and t denoted by $overlap(s, t)$ is the suffix of maximum length of s, which is also a prefix of t.

A possible 2-approximation algorithm

The definition of overlap

The overlap of two strings s and t denoted by $overlap(s,t)$ is the suffix of maximum length of s, which is also a prefix of t.

The Algorithm

- Choose two strings with maximum overlap, and replace them with one obtained by overlapping them as much as possible.

A possible 2-approximation algorithm

The definition of overlap

The overlap of two strings s and t denoted by $overlap(s, t)$ is the suffix of maximum length of s, which is also a prefix of t.

The Algorithm

- Choose two strings with maximum overlap, and replace them with one obtained by overlapping them as much as possible.
- Repeat the above step until one string is obtained.

A possible 2-approximation algorithm

The definition of overlap

The overlap of two strings s and t denoted by $overlap(s, t)$ is the suffix of maximum length of s, which is also a prefix of t.

The Algorithm

- Choose two strings with maximum overlap, and replace them with one obtained by overlapping them as much as possible.
- Repeat the above step until one string is obtained.

The Algorithm

This algorithm is conjectured to have performance ratio 2.

A possible 2-approximation algorithm

The definition of overlap

The overlap of two strings s and t denoted by $overlap(s, t)$ is the suffix of maximum length of s, which is also a prefix of t.

The Algorithm

- Choose two strings with maximum overlap, and replace them with one obtained by overlapping them as much as possible.
- Repeat the above step until one string is obtained.

The Algorithm

This algorithm is conjectured to have performance ratio 2. The following example shows that the coefficient cannot be improved: ab^k, $b^k c$, and b^{k+1}.

A possible 2-approximation algorithm

The definition of overlap

The overlap of two strings s and t denoted by $overlap(s, t)$ is the suffix of maximum length of s, which is also a prefix of t.

The Algorithm

- Choose two strings with maximum overlap, and replace them with one obtained by overlapping them as much as possible.
- Repeat the above step until one string is obtained.

The Algorithm

This algorithm is conjectured to have performance ratio 2. The following example shows that the coefficient cannot be improved: ab^k, $b^k c$, and b^{k+1}. $OPT = k + 3$, $ALG = 2k + 3$.

$2 \cdot H_n$-approximation algorithm via Set Cover

Constructing the Instance of Set Cover

- For $s_i, s_j \in S$ and $k > 0$, if the k last symbols of s_i are the same as the first k symbols of s_j, then let $\sigma_{i,j,k}$ denote the string obtained by overlapping these k positions of s_i and s_j.

$2 \cdot H_n$-approximation algorithm via Set Cover

Constructing the Instance of Set Cover

- For $s_i, s_j \in S$ and $k > 0$, if the k last symbols of s_i are the same as the first k symbols of s_j, then let $\sigma_{i,j,k}$ denote the string obtained by overlapping these k positions of s_i and s_j.
- Let M be the set of all strings $\sigma_{i,j,k}$ for all valid choices of i, j, k.

$2 \cdot H_n$-approximation algorithm via Set Cover

Constructing the Instance of Set Cover

- For $s_i, s_j \in S$ and $k > 0$, if the k last symbols of s_i are the same as the first k symbols of s_j, then let $\sigma_{i,j,k}$ denote the string obtained by overlapping these k positions of s_i and s_j.
- Let M be the set of all strings $\sigma_{i,j,k}$ for all valid choices of i, j, k. For a string π let $set(\pi)$ be the set of all strings of S that are a substring of π.

$2 \cdot H_n$-approximation algorithm via Set Cover

Constructing the Instance of Set Cover

- For $s_i, s_j \in S$ and $k > 0$, if the k last symbols of s_i are the same as the first k symbols of s_j, then let $\sigma_{i,j,k}$ denote the string obtained by overlapping these k positions of s_i and s_j.
- Let M be the set of all strings $\sigma_{i,j,k}$ for all valid choices of i, j, k. For a string π let $set(\pi)$ be the set of all strings of S that are a substring of π.
- The instance $\mathcal{T}$ of set cover is constructed as follows:

$2 \cdot H_n$-approximation algorithm via Set Cover

Constructing the Instance of Set Cover

- For $s_i, s_j \in S$ and $k > 0$, if the k last symbols of s_i are the same as the first k symbols of s_j, then let $\sigma_{i,j,k}$ denote the string obtained by overlapping these k positions of s_i and s_j.
- Let M be the set of all strings $\sigma_{i,j,k}$ for all valid choices of i, j, k. For a string π let $set(\pi)$ be the set of all strings of S that are a substring of π.
- The instance $\mathcal{T}$ of set cover is constructed as follows: the universal set of $\mathcal{T}$ is S, and the specified subsets of S are the sets $set(\pi)$ which correspond to all sets $\pi \in S \cup M$.

2 · H_n-approximation algorithm via Set Cover

Constructing the Instance of Set Cover

- For $s_i, s_j \in S$ and $k > 0$, if the k last symbols of s_i are the same as the first k symbols of s_j, then let $\sigma_{i,j,k}$ denote the string obtained by overlapping these k positions of s_i and s_j.
- Let M be the set of all strings $\sigma_{i,j,k}$ for all valid choices of i, j, k. For a string π let $set(\pi)$ be the set of all strings of S that are a substring of π.
- The instance $\mathcal{T}$ of set cover is constructed as follows: the universal set of $\mathcal{T}$ is S, and the specified subsets of S are the sets $set(\pi)$ which correspond to all sets $\pi \in S \cup M$. The cost of $set(\pi)$ is the length of π.

2 · H_n-approximation algorithm via Set Cover

Constructing the Instance of Set Cover

- For $s_i, s_j \in S$ and $k > 0$, if the k last symbols of s_i are the same as the first k symbols of s_j, then let $\sigma_{i,j,k}$ denote the string obtained by overlapping these k positions of s_i and s_j.
- Let M be the set of all strings $\sigma_{i,j,k}$ for all valid choices of i, j, k. For a string π let $set(\pi)$ be the set of all strings of S that are a substring of π.
- The instance $\mathcal{T}$ of set cover is constructed as follows: the universal set of $\mathcal{T}$ is S, and the specified subsets of S are the sets $set(\pi)$ which correspond to all sets $\pi \in S \cup M$. The cost of $set(\pi)$ is the length of π.
- Let OPT and $OPT_{\mathcal{T}}$ denote the length of the shortest superstring of S and the cost of an optimal solution to $\mathcal{T}$.

2 · H_n-approximation algorithm via Set Cover

The description of the algorithm

- Construct the instance $\mathcal{I}$ and solve it by the greedy algorithm.

2 · H_n-approximation algorithm via Set Cover

The description of the algorithm

- Construct the instance $\mathcal{T}$ and solve it by the greedy algorithm. Let $set(\pi_1), \ldots, set(\pi_k)$ be the sets picked by this cover.

$2 \cdot H_n$-approximation algorithm via Set Cover

The description of the algorithm

- Construct the instance $\mathcal{T}$ and solve it by the greedy algorithm. Let $set(\pi_1), \ldots, set(\pi_k)$ be the sets picked by this cover.
- Concatenate the strings $\pi_1, \ldots, \pi_k$ in any order.

$2 \cdot H_n$-approximation algorithm via Set Cover

The description of the algorithm

- Construct the instance $\mathcal{T}$ and solve it by the greedy algorithm. Let $set(\pi_1), \ldots, set(\pi_k)$ be the sets picked by this cover.
- Concatenate the strings $\pi_1, \ldots, \pi_k$ in any order.
- Output the resulting string s.

$2 \cdot H_n$-approximation algorithm via Set Cover

The description of the algorithm

- Construct the instance $\mathcal{T}$ and solve it by the greedy algorithm. Let $set(\pi_1), \ldots, set(\pi_k)$ be the sets picked by this cover.
- Concatenate the strings $\pi_1, \ldots, \pi_k$ in any order.
- Output the resulting string s.

Theorem

$OPT \leq OPT_{\mathcal{T}} \leq 2 \cdot OPT$.

$2 \cdot H_n$-approximation algorithm via Set Cover

The description of the algorithm

- Construct the instance $\mathcal{T}$ and solve it by the greedy algorithm. Let $set(\pi_1), \ldots, set(\pi_k)$ be the sets picked by this cover.
- Concatenate the strings $\pi_1, \ldots, \pi_k$ in any order.
- Output the resulting string s.

Theorem

$OPT \leq OPT_{\mathcal{T}} \leq 2 \cdot OPT$.

Corollary

The above algorithm has performance ratio $2 \cdot H_n$.

A bound for *OPT* via Cycle Cover

The definition of prefix

If $overlap(s_i, s_j)$ denotes the maximum overlap between s_i and s_j, then let $prefix(s_i, s_j)$ be the prefix of s_i obtained by removing its overlap with s_j.

A bound for *OPT* via Cycle Cover

The definition of prefix

If $overlap(s_i, s_j)$ denotes the maximum overlap between s_i and s_j, then let $prefix(s_i, s_j)$ be the prefix of s_i obtained by removing its overlap with s_j.

The definition of prefix graph

The vertex set of the prefix graph is the set $\{1, ..., n\}$, and for each i, j $i \neq j$ the graph contains an arc from i to j whose weight is $|prefix(s_i, s_j)|$.

A bound for *OPT* via Cycle Cover

The definition of prefix

If $overlap(s_i, s_j)$ denotes the maximum overlap between s_i and s_j, then let $prefix(s_i, s_j)$ be the prefix of s_i obtained by removing its overlap with s_j.

The definition of prefix graph

The vertex set of the prefix graph is the set $\{1, ..., n\}$, and for each i, j $i \neq j$ the graph contains an arc from i to j whose weight is $|prefix(s_i, s_j)|$.

Remark

Minimum weight cycle cover can be found in polynomial time.

A bound for *OPT* via Cycle Cover

The definition of prefix

If $overlap(s_i, s_j)$ denotes the maximum overlap between s_i and s_j, then let $prefix(s_i, s_j)$ be the prefix of s_i obtained by removing its overlap with s_j.

The definition of prefix graph

The vertex set of the prefix graph is the set $\{1, ..., n\}$, and for each i, j $i \neq j$ the graph contains an arc from i to j whose weight is $|prefix(s_i, s_j)|$.

Remark

Minimum weight cycle cover can be found in polynomial time.

The actual bound

If s_1,..., s_n are numbered in order of leftmost occurrence in the shortest superstring s, then

$$\begin{aligned} OPT &= |s| \\ &= |prefix(s_1, s_2)| + ... + |prefix(s_{n-1}, s_n)| + |prefix(s_n, s_1)| + |overlap(s_n, s_1)| \\ &\geq |prefix(s_1, s_2)| + ... + |prefix(s_{n-1}, s_n)| + |prefix(s_n, s_1)| \\ &\geq \text{ the length of the shortest cycle cover in prefix graph.} \end{aligned}$$

Some definitions and notations

A notation

If $c = (i_1 \rightarrow i_2 \rightarrow ... \rightarrow i_l \rightarrow i_1)$ is a cycle in the prefix graph, then let $\alpha(c) = prefix(s_1, s_2) \cdot ... \cdot prefix(s_{i_{l-1}}, s_{i_l}) \cdot prefix(s_{i_l}, s_{i_1})$.

Some definitions and notations

A notation

If $c = (i_1 \to i_2 \to \ldots \to i_l \to i_1)$ is a cycle in the prefix graph, then let $\alpha(c) = prefix(s_1, s_2) \cdot \ldots \cdot prefix(s_{i_{l-1}}, s_{i_l}) \cdot prefix(s_{i_l}, s_{i_1})$. Define the weight of cycle c, to be the length of $\alpha(c)$.

Some definitions and notations

A notation

If $c = (i_1 \to i_2 \to \ldots \to i_l \to i_1)$ is a cycle in the prefix graph, then let $\alpha(c) = prefix(s_1, s_2) \cdot \ldots \cdot prefix(s_{i_{l-1}}, s_{i_l}) \cdot prefix(s_{i_l}, s_{i_1})$. Define the weight of cycle c, to be the length of $\alpha(c)$.

Remark

Notice that each string s_{i_1},..., s_{i_l} is a substring of $(\alpha(c))^{\infty}$.

Some definitions and notations

A notation

If $c = (i_1 \to i_2 \to \dots \to i_l \to i_1)$ is a cycle in the prefix graph, then let $\alpha(c) = prefix(s_1, s_2) \cdot \dots \cdot prefix(s_{i_{l-1}}, s_{i_l}) \cdot prefix(s_{i_l}, s_{i_1})$. Define the weight of cycle c, to be the length of $\alpha(c)$.

Remark

Notice that each string s_{i_1},..., s_{i_l} is a substring of $(\alpha(c))^\infty$.

One more notation

$\sigma(c) = \alpha(c) \cdot s_{i_1}$.

Some definitions and notations

A notation

If $c = (i_1 \to i_2 \to \dots \to i_l \to i_1)$ is a cycle in the prefix graph, then let $\alpha(c) = prefix(s_1, s_2) \cdot \dots \cdot prefix(s_{i_{l-1}}, s_{i_l}) \cdot prefix(s_{i_l}, s_{i_1})$. Define the weight of cycle c, to be the length of $\alpha(c)$.

Remark

Notice that each string $s_{i_1}, \dots, s_{i_l}$ is a substring of $(\alpha(c))^\infty$.

One more notation

$\sigma(c) = \alpha(c) \cdot s_{i_1}$.

Remark

Notice that $\sigma(c)$ is a superstring of $s_{i_1}, \dots, s_{i_l}$.

Some definitions and notations

A notation

If $c = (i_1 \to i_2 \to \ldots \to i_l \to i_1)$ is a cycle in the prefix graph, then let $\alpha(c) = prefix(s_1, s_2) \cdot \ldots \cdot prefix(s_{i_{l-1}}, s_{i_l}) \cdot prefix(s_{i_l}, s_{i_1})$. Define the weight of cycle c, to be the length of $\alpha(c)$.

Remark

Notice that each string $s_{i_1}, \ldots, s_{i_l}$ is a substring of $(\alpha(c))^\infty$.

One more notation

$\sigma(c) = \alpha(c) \cdot s_{i_1}$.

Remark

Notice that $\sigma(c)$ is a superstring of $s_{i_1}, \ldots, s_{i_l}$.

Definition

s_{i_1} is called the representative string for c.

The description of 4-approximation algorithm

The Algorithm

- Construct the prefix graph corresponding to strings in S.

The description of 4-approximation algorithm

The Algorithm

- Construct the prefix graph corresponding to strings in S.
- Find a minimum cycle cover of the prefix graph $\mathcal{C} = \{c_1, ..., c_k\}$.

The description of 4-approximation algorithm

The Algorithm

- Construct the prefix graph corresponding to strings in S.
- Find a minimum cycle cover of the prefix graph $\mathcal{C} = \{c_1, ..., c_k\}$.
- Output the string $\sigma(c_1) \cdot ... \cdot \sigma(c_k)$.

The description of 4-approximation algorithm

The Algorithm

- Construct the prefix graph corresponding to strings in S.
- Find a minimum cycle cover of the prefix graph $\mathcal{C} = \{c_1, ..., c_k\}$.
- Output the string $\sigma(c_1) \cdot ... \cdot \sigma(c_k)$.

Remark

Observe that the algorithm constructs a superstring of strings of S.

The analyses of the algorithm

Lemma

If each string in $S' \subseteq S$ is a substring of t^{∞} for a string t, then there is a cycle of weight at most length of t in the prefix graph covering all the vertices corresponding to strings in S'.

The analyses of the algorithm

Lemma

If each string in $S' \subseteq S$ is a substring of t^{∞} for a string t, then there is a cycle of weight at most length of t in the prefix graph covering all the vertices corresponding to strings in S'.

Lemma

Let c and c' be two cycles in $\mathcal{C}$, and let r, r' be representative strings from these cycles.

The analyses of the algorithm

Lemma

If each string in $S' \subseteq S$ is a substring of t^∞ for a string t, then there is a cycle of weight at most length of t in the prefix graph covering all the vertices corresponding to strings in S'.

Lemma

Let c and c' be two cycles in $\mathcal{C}$, and let r, r' be representative strings from these cycles. Then $|overlap(r, r')| < weight(c) + weight(c')$.

The analyses of the algorithm

Theorem

The algorithm achieves an approximation factor 4 for the shortest superstring problem.

The analyses of the algorithm

Theorem

The algorithm achieves an approximation factor 4 for the shortest superstring problem.

Proof.

Let $weight(\mathcal{C}) = \sum_{i=1}^{k} weight(c_i)$.

The analyses of the algorithm

Theorem

The algorithm achieves an approximation factor 4 for the shortest superstring problem.

Proof.

Let $weight(\mathcal{C}) = \sum_{i=1}^{k} weight(c_i)$. Then

$$\begin{aligned} ALG &= \sum_{i=1}^{k} |\sigma(c_i)| \\ &= weight(\mathcal{C}) + \sum_{i=1}^{k} |r_i| \\ &\leq OPT + \sum_{i=1}^{k} |r_i|. \end{aligned}$$

$\square$

The analyses of the algorithm

Proof.

The analyses of the algorithm

Proof.

On the other hand,

$$OPT \geq \sum_{i=1}^{k} |r_i| - \sum_{i=1}^{k-1} |overlap(r_i, r_{i+1})|$$
$$\geq \sum_{i=1}^{k} |r_i| - 2 \cdot weight(\mathcal{C})$$

The analyses of the algorithm

Proof.

On the other hand,

$$OPT \geq \sum_{i=1}^{k} |r_i| - \sum_{i=1}^{k-1} |overlap(r_i, r_{i+1})|$$
$$\geq \sum_{i=1}^{k} |r_i| - 2 \cdot weight(\mathcal{C})$$

which implies that

$$\sum_{i=1}^{k} |r_i| \leq 2 \cdot weight(\mathcal{C}) + OPT$$
$$\leq 3 \cdot OPT.$$

□

Some definitions and notations

A notation

If X is a set of strings, then let $||X||$ denote the sum of lengths of strings in X.

Some definitions and notations

A notation

If X is a set of strings, then let $||X||$ denote the sum of lengths of strings in X.

Definition

Compression achieved by a superstring s of the set of strings S is defined as $||S|| - |s|$.

Some definitions and notations

A notation

If X is a set of strings, then let $||X||$ denote the sum of lengths of strings in X.

Definition

Compression achieved by a superstring s of the set of strings S is defined as $||S|| - |s|$.

Remark

Clearly, maximum compression is achieved by the shortest superstring.

The description of 3-approximation algorithm

The Algorithm

- Construct the prefix graph corresponding to strings in S.

The description of 3-approximation algorithm

The Algorithm

- Construct the prefix graph corresponding to strings in S.
- Find a minimum cycle cover of the prefix graph $\mathcal{C} = \{c_1, ..., c_k\}$.

The description of 3-approximation algorithm

The Algorithm

- Construct the prefix graph corresponding to strings in S.
- Find a minimum cycle cover of the prefix graph $\mathcal{C} = \{c_1, ..., c_k\}$.
- Run any half-optimal-compression achieving algorithm on $\{\sigma(c_1), ..., \sigma(c_k)\}$ and output the resulting string, say τ.

The description of 3-approximation algorithm

The Algorithm

- Construct the prefix graph corresponding to strings in S.
- Find a minimum cycle cover of the prefix graph $\mathcal{C} = \{c_1, ..., c_k\}$.
- Run any half-optimal-compression achieving algorithm on $\{\sigma(c_1), ..., \sigma(c_k)\}$ and output the resulting string, say τ.

Remark

Observe that the algorithm constructs a superstring of strings of S.

The analyses of the algorithm

A notation

Let OPT_σ denote the length of the shortest superstring of the strings in $S_\sigma = \{\sigma(c_1), ..., \sigma(c_k)\}$

The analyses of the algorithm

A notation

Let OPT_σ denote the length of the shortest superstring of the strings in $S_\sigma = \{\sigma(c_1), ..., \sigma(c_k)\}$ and let r_i be the representatives string of c_i.

The analyses of the algorithm

A notation

Let OPT_σ denote the length of the shortest superstring of the strings in $S_\sigma = \{\sigma(c_1), ..., \sigma(c_k)\}$ and let r_i be the representatives string of c_i.

Lemma

$|\tau| \leq OPT_\sigma + weight(\mathcal{C})$.

The analyses of the algorithm

A notation

Let OPT_σ denote the length of the shortest superstring of the strings in $S_\sigma = \{\sigma(c_1), ..., \sigma(c_k)\}$ and let r_i be the representatives string of c_i.

Lemma

$|\tau| \leq OPT_\sigma + weight(\mathcal{C})$.

Proof.

Suppose that $\sigma(c_1), ..., \sigma(c_k)$ appear in this order in the shortest superstring.

The analyses of the algorithm

A notation

Let OPT_σ denote the length of the shortest superstring of the strings in $S_\sigma = \{\sigma(c_1), ..., \sigma(c_k)\}$ and let r_i be the representatives string of c_i.

Lemma

$|\tau| \leq OPT_\sigma + weight(\mathcal{C})$.

Proof.

Suppose that $\sigma(c_1), ..., \sigma(c_k)$ appear in this order in the shortest superstring. Then

$$\begin{aligned} ||S_\sigma|| - OPT_\sigma &= \sum_{i=1}^{k-1} |overlap(\sigma(c_i), \sigma(c_{i+1}))| \\ &\leq \sum_{i=1}^{k-1} |overlap(r_i, r_{i+1})| \\ &\leq 2 \cdot weight(\mathcal{C}). \end{aligned}$$

$\square$

The analyses of the algorithm

Proof.

Since the compression is half-optimal, we have $||S_\sigma|| - |\tau| \geq \frac{1}{2}(||S_\sigma|| - OPT_\sigma)$,

The analyses of the algorithm

Proof.

Since the compression is half-optimal, we have $||S_\sigma|| - |\tau| \geq \frac{1}{2}(||S_\sigma|| - OPT_\sigma)$, which implies that

$$\begin{aligned} 2 \cdot (|\tau| - OPT_\sigma) &\leq ||S_\sigma|| - OPT_\sigma \\ &\leq 2 \cdot weight(\mathcal{C}) \end{aligned}$$

The analyses of the algorithm

Proof.

Since the compression is half-optimal, we have $||S_\sigma|| - |\tau| \geq \frac{1}{2}(||S_\sigma|| - OPT_\sigma)$, which implies that

$$\begin{aligned} 2 \cdot (|\tau| - OPT_\sigma) &\leq ||S_\sigma|| - OPT_\sigma \\ &\leq 2 \cdot weight(\mathcal{C}) \end{aligned}$$

or

$$|\tau| \leq OPT_\sigma + weight(\mathcal{C}).$$

$\square$

The analyses of the algorithm

Lemma

$OPT_\sigma \leq OPT + weight(\mathcal{C})$.

The analyses of the algorithm

Lemma

$OPT_\sigma \leq OPT + weight(\mathcal{C})$.

Proof.

Let OPT_r be the length of shortest superstring of $\{r_1, ..., r_k\}$,and assume that they appear in this order in the shortest superstring.

The analyses of the algorithm

Lemma

$OPT_\sigma \leq OPT + weight(\mathcal{C})$.

Proof.

Let OPT_r be the length of shortest superstring of $\{r_1, ..., r_k\}$,and assume that they appear in this order in the shortest superstring. Since $\sigma(c_i)$ begins and ends with r_i, one has

$$||S_\sigma|| - OPT_\sigma \geq ||S_r|| - OPT_r,$$

The analyses of the algorithm

Lemma

$OPT_\sigma \leq OPT + weight(\mathcal{C})$.

Proof.

Let OPT_r be the length of shortest superstring of $\{r_1, ..., r_k\}$,and assume that they appear in this order in the shortest superstring. Since $\sigma(c_i)$ begins and ends with r_i, one has

$$||S_\sigma|| - OPT_\sigma \geq ||S_r|| - OPT_r,$$

But $||S_\sigma|| = ||S_r|| + weight(\mathcal{C})$, which implies that

$$\begin{aligned} OPT_\sigma &\leq OPT_r + weight(\mathcal{C}) \\ &\leq OPT + weight(\mathcal{C}). \end{aligned}$$

□

The analyses of the algorithm

Lemma

$OPT_\sigma \leq OPT + weight(\mathcal{C})$.

Proof.

Let OPT_r be the length of shortest superstring of $\{r_1, ..., r_k\}$,and assume that they appear in this order in the shortest superstring. Since $\sigma(c_i)$ begins and ends with r_i, one has

$$||S_\sigma|| - OPT_\sigma \geq ||S_r|| - OPT_r,$$

But $||S_\sigma|| = ||S_r|| + weight(\mathcal{C})$, which implies that

$$\begin{aligned} OPT_\sigma &\leq OPT_r + weight(\mathcal{C}) \\ &\leq OPT + weight(\mathcal{C}). \end{aligned}$$

□

Theorem

The algorithm achieves an approximation factor 3 for the shortest superstring problem.