# Chapter 7

# Greedy Algorithms for the Shortest Common Superstring Problem

Anton Nesterov

This paper is based on a article by A. Frieze and W.Szpankowski. It presents some greedy algorithms that solve Shortest Common Superstring Problem and their analysis in probabylistic framework. Also graph algorithms for equivalent problems are presented.

Various versions of the Shortest common superstring (in short SCS) problem play important role in data compression and DNA sequensing.
Problem Formulation. Given a collection of strings, say $x^1$, $x^2, \ldots, x^n$ over an alphabet $\Sigma$, find the shortest string $z$ such that each of $x^i$ appears as substring (a consecutive block) of $z$. In the DNA seqquencing another formulation of the problem may be of even greater interest. We call it an approximate SCS and one asks for a superstring that contains approximately (e.g in the Hamming distance sense) the original strings as $x^1$, $x^2, \ldots, x^n$ as substrings.
Our results are about some greedy approximations of the SCS but in a probabilistic framefork. We prove that several greedy algorithms for the SCS problem are asymptotically optimal in the sence that thay produce a total overlap of SCS that differs from the optimal (maximum) overlap by a quantity that is an order of magnutude smaller than the leading term of the overlap.
We assume, that the strings are generated independently. We first consider the so-called Bernoulli model in which symbols of the alphabet $\Sigma$ are generating independently within a string. Later we extends our results to other models: Markovian model and Mixing Model, which is generalization of previous ones.

## 7.1 Definitions

Before presenting some results, we introduce some notation and a framefork for describing our algorithms.
Suppose $x = x_1x_2 \ldots x_3$ and $y = y_1y_2 \ldots y_3$ are strings over the same finite alphabet $\Sigma = (\omega_1, \omega_2, ..., \omega_M)$ where $M$ is the size of the alphabet. We define their overlap

$$o(x, y) = \max\{j : y_i = x_{r-j+i}, 1 \leq i \leq j\}.$$

If $x \neq y$ and $k = o(x, y)$, then

$$x \oplus y = x_1 x_2 ... y_{k+1} y_{k+2} ... y_s.$$

Let $S$ be a set of all superstrings built over strings $x^1, .., x^n$. Then

$$O_n^{opt} = \sum_{i=1}^{n} |x|^i - \min_{z \in S} |z|.$$

We assume that the input strings are independently generated. We analyse the Bernoully model, that is, each $x = x^j = x_1 x_2 ... x_{i-1}$ is the same length $l$ and and $x_i$ is generated independently of $x_1$, $x_2 \ldots x_{i-1}$. Futhermore, $P(x_i = \omega_j) = p_j > 0$ for $1 \leq j \leq M$. Let

$$H = \sum_{i=1}^{m} p_i \log p_i$$

be the associated entropy for the Bernoully model (i.e., memoryless source).

## 7.2 Greedy algorithms

We study the following algorithm: its input is the strings $x^1, x^2, .., x^n$ over $\Sigma$. It outputs a string $z$ which is a superstring of the input.

**Generic greedy algorithm.**

```
1. $I \leftarrow \{x^1, x^2, ...x^n\}; O_n^{gr} \leftarrow 0;$
2. **repeat**
3. choose $x, y \in I$; $z = x \oplus y$
4. $I \leftarrow (I \setminus \{x, y\})$
5. $O^g r_n \leftarrow O_n^{gr} + o(x, y)$
6. **until** $|I| = 1$
```

We consider two variants:
GREEDY. In Step 3 choose $x \neq y$ in order to maximize $o(x, y)$.
RGREEDY. In Step 3 $x$ is the string $z$ produced in the previous iteration, while $y$ is chosen in order to maximize $o(x, y) = o(z, y)$. Our initial choice for $x$ is $x^1$. Thus in RGREEDY we have one "long" string $z$ grows by addition of strings at the right-hand end.

## 7.3 Results

Consider the SCS problem under the Bernoulli model. Let $P = \sum_{j=1}^{M} p_i^2$. Then, with high probability,

$$\lim_{n\to\infty} \frac{O_n^{opt}}{n \log n} = \frac{1}{H}$$

$$\lim_{n\to\infty} \frac{O_n^{gr}}{n \log n} = \frac{1}{H}$$

provided

$$|x| > -\frac{4}{\log P} \log n$$

for all $1 \leq i \leq n$
In many applications, notably for data compression and the DNA recombination problem, the Bernoully model assumption is too unrealistic. Therefore, we extend this theorem to the case when there is some dependency among symbols withing a string. However we still assume that strings $x^1$,$x^2, \ldots, x^n$ are statically independent. But we restrict somewhat the dependency among symbols of each string, that is, we desribe a main ideas of the mixing model.
Mixing Model. During the generation of string, each symbol depends on all previous ones, but the farther the symbol the lesser the dependence on it.

## 7.4 Compression

The SCS can be used to compress strings. Indeed, instead of storing all strings of total length $nl$ we can store the SCS and n pointers indicating the beginning of an original string plus length of all strings. However, this does not provide optimal compression (which is known to be the entropy $H$). Show this, compute the compressionn ratio $C_n$ which is defined as the ratio of the number of bits needed to transmit the compression code to the length of the original set of strings. It is easy to see that

$$C_n = \frac{nl - (1/H)n \log n + n \log_2(nl - (1/H)n \log n)}{nl}$$

where the first term of the numerator represents the length of the Shortest superstring and the second term corresponds to the number of bits needed to encode the pointers. Observe that when the length of a string $l$ grows faster than $\log n$, then $C_n \to 1$, that means no compression. When $l = O(\log n)$ some compression might take place. The fact that SCS does not compress well is hardly surprising: in the construction of

SCS we do not use all available redundancy of all strings but only that contained in suffixes/preffixes of original strings.

## 7.5 Graph processes

In this section some graph algorithms, that correspond to GREEDY and RGREEDY are presented. But before them,
First, we show that a pair $i, j$ such that $o(x^i, x^j) \geq l/2$ unlikely exists. Let $\epsilon$ denote the event that there is no such pair. If $l = K \log n$, then

$$P(\neg\epsilon) \leq \frac{n(n-1)}{2} \sum_{k=l/2}^{l} P^k = O(n^{2+(K \log P)/2)}) = o(1)$$

provided $K \geq -4/\log P$.

### 7.5.1 RGREEDY

Consider a tree process that is equal to RGREEDY. Tree T be an infinite rooted M-ary tree. M (size of an alphabet) edges leading down from each vertex will be labeled with $\omega_1, \omega_2, ...\omega_M$. Thus, each vertex of depth $d$ is identified with string of length $d$. Also, label each vertex $v$ with an integer $v(v)$, number of strings that have the prefix associated with this vertex.
We model the process of RGEEDY in the following way: particle Z starts at the root. Then at a vertex $v$ it moves to $v$'s $\omega_j$ descendent with probability $p_j$. The particle stops at depth $l/2$. Let $\omega = s_k s_{k-1}...s_1$be the lowest vertex on the path traversed that has a nonzero $v$. This process models the computation of the largest suffix $s_k s_{k-1}...s_1$ of $z$ which can be merged with a prefix of $a^i$.
Then we model the deletion of $a^t = a_1 a_2 ... a_{l/2}$ which has the prefix $a_1 a_2 ... a_k$. Let $\omega_i = a_1 a_2 ... a_i$. Put $v(\omega_i) = \max\{0, v(\omega_i) - 1\}$ for $1 \leq i \leq l/2$.
We iterate the above process $n - 1$ times.

### 7.5.2 GREEDY

Let $D$ be the digraph $([n], A)$ with edge weights $\omega_{i,j} = o(b^i, a^j)$ for $i, j \in [n]$.
Sort the edges $A$ into $e_1, e_2, ..., e_N$, where $N = n^2$, so that $\omega(e_i) \geq \omega(e_{i+1})$;
$S_G \leftarrow \emptyset$;
**For** $i = 1$ **to** $N$ **do**: **if** $S_G \cup \{e_i\}$ contains **in** $D$ neither a vertex of outdegree or indegree at least 2 in $S_G$, nor a directed cycle, **then** $S_G \leftarrow S_G \cup \{e_i\}$.
After termination $S_G$ contains $n - 1$ edges of a Hamilton path of $D$ and corresponds to superstring $x^1$,$x^2, \ldots, x^n$. The selection of an edge weight $(b^i, a^j)$ corresponds to overlaping $x^i$ to the left of $x^j$.