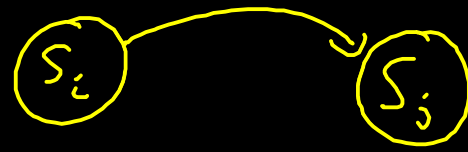
Theorem

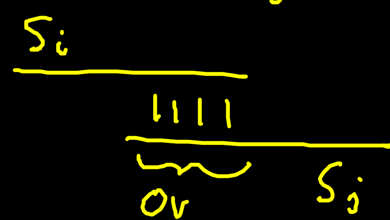
Assuming a constant size of the alphabet above greedy algorithm given input $S = \{s_1, \ldots, s_n\}$ takes running time in $\mathcal{O}(N \cdot (n + \log(N)))$, $N = \sum_{1 \le i \le n} |s_i|$.

Proof:

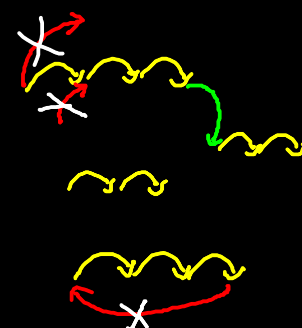


Under all remaining edges find one with maximal weight which „can be used":

$\langle S_i, S_j \rangle$



a) Row of start vertex or column of goal vertex labeled (constant time)



b) FIND on incident vertices gives same partition. (constant time)

a) $\vee$ b) $\rightsquigarrow$ edge not used.

At most $2n-1$ edges cannot be used (per iteration)

$\Longrightarrow$ all iterations together in time $O(n^2) = O(n \cdot N)$

If we „insert" the edge $S_i \rightarrow S_j$ we do the following:

- label row i and column j of adjacency matrix

→ excluding $2n-2$ edges (constant time)

- UNION(S_i, S_j)
  - Scan shortest list element by element and update $R[v]$; merge the two lists (time $O(n)$)

$\Longrightarrow$ iteration over all edges selected $O(n^2) = O(n \cdot N)$.

3

Performance guarantees?

Example: Let $S = \{c(ab)^m, (ba)^m, (ab)^m c\}$, $m \in \mathbb{N}$, the input for our greedy algorithm. The pair with maximal overlap is given by $(c(ab)^m, (ab)^m c)$ so the algorithm does merge $< (c(ab)^m, (ab)^m c >= c(ab)^m c$.
The next iteration then processes set $\{(ba)^m, c(ab)^m c\}$.
As both possible pairings do not yield an overlap concatenating is our only choice, resulting in the superstring $(ba)^m c(ab)^m c$ or $c(ab)^m c(ba)^m$ both of length $4m + 2$.

It would have been better to first place the string $(ba)^m$ between the words $c(ab)^m$ and $(ab)^m c$ leading to the optimal superstring $ca(ba)^m bc$ of length $2m + 4$.
So the approximation rate of the greedy method for this example is $\lim_{m\to\infty} \frac{4m+2}{2m+4} = 2$.

We have hence shown that for the greedy algorithm there can be no performance guarantee **better** than 2. It is widely believed that this is the actual performance guarantee but none has yet been able to prove this.

The following is proven:

Theorem

The greedy algorithm to compute a superstring is a 4-approximation algorithm for SCSP.

□

As mentioned above an optimal solution for SCSP is also an optimal solution for MCCSP. However we do not yet know any performance guarantee for the greedy algorithm wrt. maximizing compression.

Theorem

The greedy algorithm to compute a superstring is a 3-approximation algorithm for MCCSP.

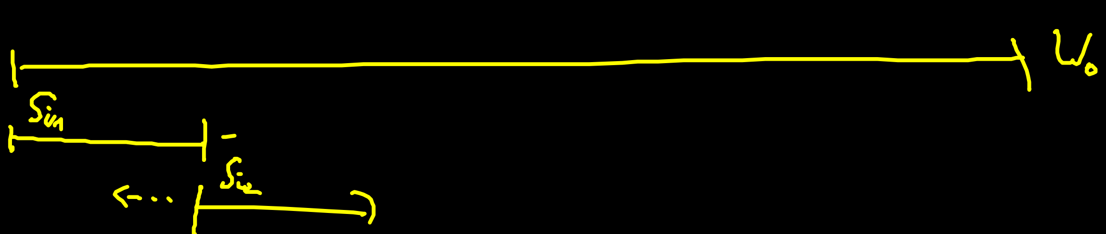
Proof:

w_0 = optimal superstring for input $S = \{s_1, s_2, \dots, s_n\}$

We order the s_i according to their first (left-most) appearance within w_0

$\rightsquigarrow (s_{i_1}, s_{i_2}, \dots, s_{i_n})$, i.e.

$$w_0 = \langle \dots \langle\langle s_{i_1}, s_{i_2}\rangle, s_{i_3}\rangle \dots\rangle, s_{i_n}\rangle$$



$$\mathrm{comp}(w_0) = \sum_{m \in \mathrm{merge}} |ov(m)| = \sum_{1 \le k < n} ov(s_{i_k}, s_{i_{k+1}})$$

Greedy algorithm generates an ordering

$(S_{j_1}, S_{j_2}, \ldots, S_{j_n})$ analoguosly.

We show: merge $\langle S_{j_k}, S_{j_{k+1}} \rangle$ can make at most three merges of optimal solution impossible.

Case 1: $\langle S_{j_k}, S_{j_{k+1}} \rangle = \langle S_{i_\ell}, S_{i_{\ell+1}} \rangle$, $\ell \in [1:n-1]$

$\rightsquigarrow$ both merges identical, i.e. no merge of optimal solution becomes impossible

Case 2: $\langle S_{j_k}, S_{j_{k+1}} \rangle = \langle S_{i_\ell}, S_{i_{\ell+m}} \rangle$, $m > 1$

$\rightsquigarrow$ at most 2 merges become impossible, namely

$$\langle S_{i_\ell}, S_{i_{\ell+1}} \rangle \ \& \ \langle S_{i_{\ell+m-1}}, S_{i_{\ell+m}} \rangle$$

Case 3: $\langle S_{j_k}, S_{j_{k+1}} \rangle = \langle S_{i_{\ell+m}}, S_{i_\ell} \rangle$, $m \geq 1$

$\rightsquigarrow$ at most 3 merges become impossible

$\langle S_{i_{\ell+m}}, S_{i_{\ell+m+1}} \rangle$, $\langle S_{i_{\ell-1}}, S_{i_\ell} \rangle$ and one of $\langle S_{i_\ell}, S_{i_{\ell+1}}\rangle, \dots, \langle S_{i_{\ell+k-1}}, S_{i_{\ell+k}}\rangle$

Since the use of all of them would give rise to a cycle which is broken by not using one.

Notation: $M_x \mathrel{\hat{=}}$ set of merges used by algorithm $x \in \{o, g\}$.

$V(m) \subseteq M_o \mathrel{\hat{=}}$ set of merges being made impossible by merge $m = \langle S_{j_k}, S_{j_{k+1}}\rangle \in M_g$

$\leadsto (\forall m \in M_g)(|V(m)| \leq 3)$.

For $ov(m)$, $m \in M_g$, the length of the overlap use within merge m we have

$$(\forall m \in M_g) : \left(ov(m) \geq \frac{1}{3} \Big(\underbrace{\sum_{m' \in V(m)} ov(m')}_{\text{at most 3 summands}} \Big) \right) \qquad (*)$$

Reason: greedy chooses among the remaining (possible) merges the one of maximal overlap

$\leadsto$ a merge made impossible by m never has overlap $> ov(m)$.

Furthermore:

$$(\forall m' \in M_o) : (m' \in M_g) \vee \left((\exists m \in \overbrace{M_g \setminus M_o}^{\checkmark}) : (m' \in V(m)) \right) \quad (**)$$

Reason: For $m' = \langle s, \bar{s} \rangle \notin M_g$ we must have merges with M_g which merge s resp. $\bar{s}$ with other strings. This however make m' impossible

Consider

$$\sum_{m \in M_g} ov(m) = \sum_{m \in M_g \cap M_o} ov(m) + \sum_{m \in M_g \setminus M_o} ov(m)$$

We have

$$\sum_{m \in M_g \cap M_o} ov(m) \geq \frac{1}{3} \sum_{m \in M_g \cap M_o} ov(m) \qquad \text{and}$$

$$\sum_{m \in \underbrace{M_g \setminus M_o}_{\subseteq M_g}} ov(m) \overset{(*)}{\geq} \sum_{m \in M_g \setminus M_o} \frac{1}{3} \cdot \sum_{m' \in V(m)} ov(m')$$

$$\overset{(**)}{\geq} \frac{1}{3} \cdot \sum_{m' \in M_0 \setminus M_g} ov(m')$$

$m' \notin M_g$

$\rightsquigarrow \exists m \in M_g \setminus M_0 : m' \in V(m)$

$$\Longrightarrow \frac{comp(w_0)}{comp(w_g)} = \frac{\sum_{m' \in M_0} ov(m')}{\sum_{m \in M_g} ov(m)} \leq \frac{\sum_{m' \in M_0} ov(m')}{\frac{1}{3} \cdot \sum_{m' \in M_0} ov(m')}$$

$$= 3$$

Using more elaborate reasoning one can show:

Theorem

The greedy algorithm to compute a superstring is a 2-approximation algorithm for MCCSP. □

Alternative Procedure

Idea: Do not use a single cycle to cover all vertices but a set of cycles

$\rightsquigarrow$ cycle-cover for which each vertex must be part of exactly one cycle.

Here: Cycle-cover of minimal cost (according to distance graph).

Remark: A minimal cycle-cover can be computed in time $\mathcal{O}(n^3)$ (whereas the restriction to edges to be covered by at least one cycle leads to an $\mathcal{NP}$-complete problem).