Algorithms for Three Versions of the Shortest Common Superstring Problem

Maxime Crochemore, Marek Cygan, Costas Iliopoulos, Marcin Kubica, Jakub Radoszewski, Wojciech Rytter, Tomasz Walen

King's College London and University of Warsaw

CPM 2010, June 23, 2010, New York

Classical SCS problem

SCS problem:
Input: words $s_1, s_2, \ldots, s_k \in \Sigma^*$, $|s_1| + \ldots + |s_k| = n$.
Output: the shortest word s containing each s_i as a factor.

Example:
$s_1 = \mathtt{abaab}$, $s_2 = \mathtt{baba}$, $s_3 = \mathtt{aabbb}$, $s_4 = \mathtt{bbab}$.
$s = \mathtt{bbabaabbb}$, since:

Classical SCS problem

SCS problem:
Input: words $s_1, s_2, \ldots, s_k \in \Sigma^*$, $|s_1| + \ldots + |s_k| = n$.
Output: the shortest word s containing each s_i as a factor.

Example:
$s_1 = \mathtt{abaab}$, $s_2 = \mathtt{baba}$, $s_3 = \mathtt{aabbb}$, $s_4 = \mathtt{bbab}$.
$s = \mathtt{bbabaabbb}$, since:

bb abaab bb

Classical SCS problem

SCS problem:
Input: words $s_1, s_2, \ldots, s_k \in \Sigma^*$, $|s_1| + \ldots + |s_k| = n$.
Output: the shortest word s containing each s_i as a factor.

Example:
$s_1 = \mathtt{abaab}$, $s_2 = \mathtt{baba}$, $s_3 = \mathtt{aabbb}$, $s_4 = \mathtt{bbab}$.
$s = \mathtt{bbabaabbb}$, since:

bb abaab bb
b baba abbb

Classical SCS problem

SCS problem:
Input: words $s_1, s_2, \ldots, s_k \in \Sigma^*$, $|s_1| + \ldots + |s_k| = n$.
Output: the shortest word s containing each s_i as a factor.

Example:
$s_1 = \mathtt{abaab}$, $s_2 = \mathtt{baba}$, $s_3 = \mathtt{aabbb}$, $s_4 = \mathtt{bbab}$.
$s = \mathtt{bbabaabbb}$, since:

bb abaab bb
b baba abbb
bbab aabbb

Classical SCS problem

SCS problem:
Input: words $s_1, s_2, \ldots, s_k \in \Sigma^*$, $|s_1| + \ldots + |s_k| = n$.
Output: the shortest word s containing each s_i as a factor.

Example:
$s_1 = \mathtt{abaab}$, $s_2 = \mathtt{baba}$, $s_3 = \mathtt{aabbb}$, $s_4 = \mathtt{bbab}$.
$s = \mathtt{bbabaabbb}$, since:

bb abaab bb
b baba abbb
bbab aabbb

A standard assumption: the set $\{s_1, \ldots, s_k\}$ is *factor-free*.

Known results for SCS

- SCS problem is NP-complete.

Known results for SCS

- SCS problem is NP-complete.
- Exact algorithms:
  - $O^*(2^k)$ time, $O^*(2^k)$ space (Held & Karp, 1962, via TSP)
  - $O^*(2^k)$ time, $O^*(1)$ space (Karp, 1982, via TSP)

Known results for SCS

- SCS problem is NP-complete.
- Exact algorithms:
  - $O^*(2^k)$ time, $O^*(2^k)$ space (Held & Karp, 1962, via TSP)
  - $O^*(2^k)$ time, $O^*(1)$ space (Karp, 1982, via TSP)
- Greedy approximation algorithm:
  - 4-approximation (Blum et al., 1991)
  - 3.5-approximation (Kaplan & Shafrir, 2005)
  - 2-approximation? (conjecture, Blum et al.)

Known results for SCS

- SCS problem is NP-complete.
- Exact algorithms:
  - $O^*(2^k)$ time, $O^*(2^k)$ space (Held & Karp, 1962, via TSP)
  - $O^*(2^k)$ time, $O^*(1)$ space (Karp, 1982, via TSP)
- Greedy approximation algorithm:
  - 4-approximation (Blum et al., 1991)
  - 3.5-approximation (Kaplan & Shafrir, 2005)
  - 2-approximation? (conjecture, Blum et al.)
- 2.59-approximation (Breslauer et al., 1997)
  2.5-approximation (Sweedyk, 1999 + Kaplan et al., 2005)

Multiple occurrences

Notation: $\#occ(u, v)$ – the number of occurrences of the word u as a factor of v.

Input: words $s_1, s_2, \ldots, s_k \in \Sigma^*$, $|s_1| + \ldots + |s_k| = n$.

SUM-SCS(k) problem:
Input: words s_i and positive integer m.
Output: The word shortest u such that $\sum_{i=1}^{k} \#occ(s_i, u) \geq m$.

MULTI-SCS(k) problem:
Input: words s_i and positive integers m_i.
Output: the shortest word u such that $\#occ(s_i, u) \geq m_i$ for $i = 1, 2, \ldots, k$.

MULTI-SCS$_2$(k) problem: for words of length 2.

Assumptions

We assume that $m = O(2^n)$, $m_i = O(2^n)$ and such numbers fit in a single memory cell.
Note that the size of input is $O(n)$.

We wish to find a compressed representation of the result of $O(poly(n))$ size that can be used to reconstruct the actual word in a straightforward manner and in $O(\ell)$ time, where ℓ is the length of the word.

MULTI-SCS(k) problem

MULTI-SCS(k) problem:
Input: words s_i and positive integers m_i.
Output: the shortest word u such that $\#occ(s_i, u) \geq m_i$ for $i = 1, 2, \ldots, k$.

We assume that $k = O(1)$.

We design a $O(poly(n))$ time solution.

MULTI-SCS(k) problem

MULTI-SCS(k) problem:
Input: words s_i and positive integers m_i.
Output: the shortest word u such that $\#occ(s_i, u) \geq m_i$ for $i = 1, 2, \ldots, k$.

We assume that $k = O(1)$.

We design a $O(poly(n))$ time solution.

Sketch of the solution:

1. words $\longrightarrow$ a graph
2. a graph $\longrightarrow$ a deterministic finite automaton
3. a DFA $\longrightarrow$ a regular expression
4. a regexp $\longrightarrow$ a regexp in a normal form
5. a regexp in a normal form $\longrightarrow$ integer program.

Step 1: Prefix graph

$ov(s,t)$ – the largest overlap of words s and t
$pr(s,t)$: $s = pr(s,t)ov(s,t)$

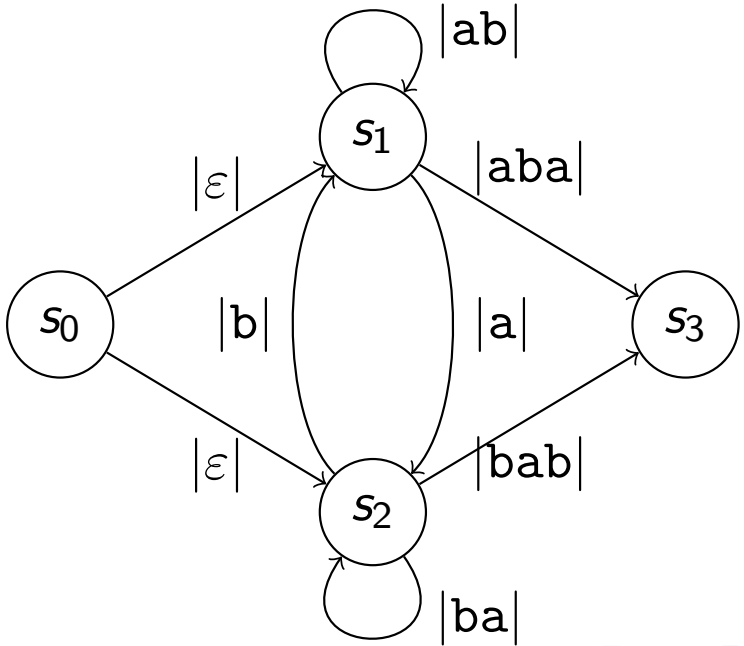
Example: $s = \mathtt{baba}$, $t = \mathtt{abaab}$, $pr(s,t) = \mathtt{b}$, $ov(s,t) = \mathtt{aba}$.

Prefix graph:

Vertices: s_i, the source ($s_0 = \varepsilon$), the sink ($s_{k+1} = \varepsilon$).

Edges: $s_i \xrightarrow{pr(s_i,s_j)} s_j$.

Example for $s_1 = \mathtt{aba}$, $s_2 = \mathtt{bab}$:

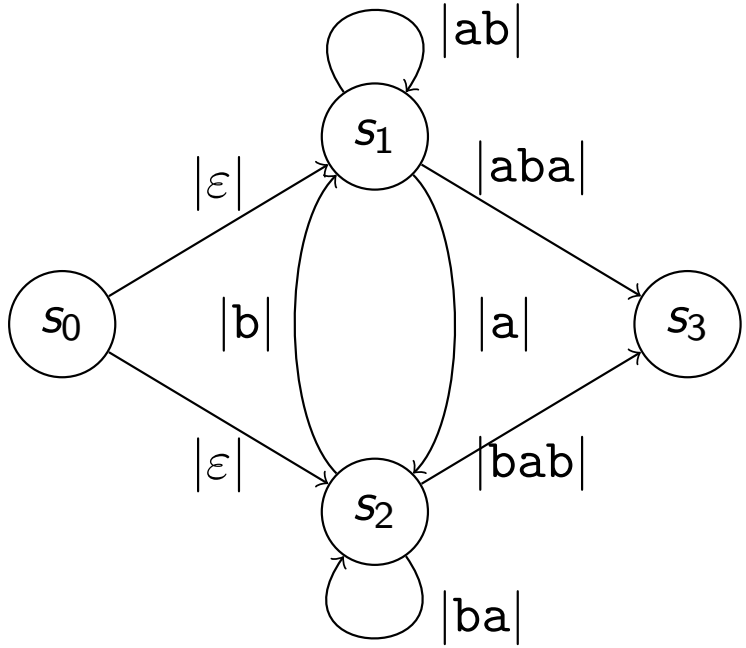


Prefix graph – construction and usage

Vertices: s_i, the source ($s_0 = \varepsilon$), the sink ($s_{k+1} = \varepsilon$).

Edges: $s_i \xrightarrow{pr(s_i, s_j)} s_j$.

Example for $s_1 = \mathtt{aba}$, $s_2 = \mathtt{bab}$:



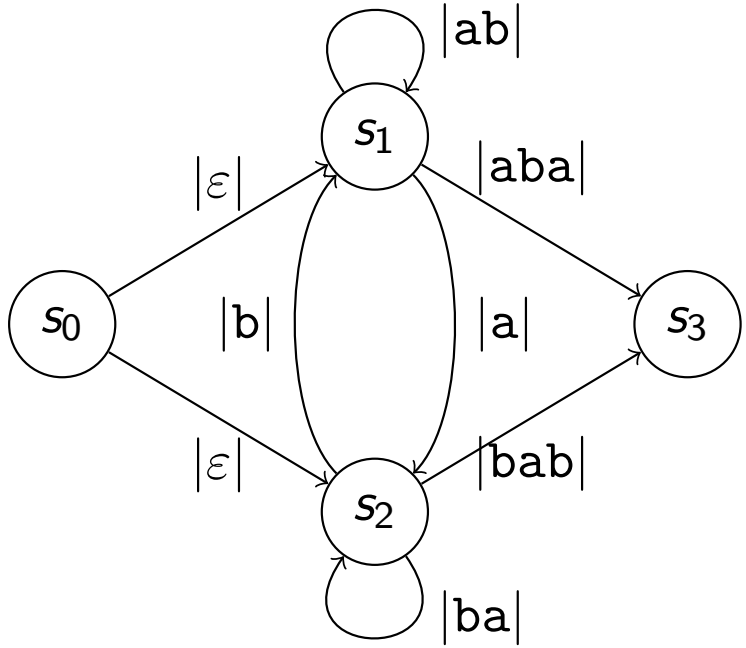
Construction: an optimal $O(n + k^2)$ time algorithm (Gusfield, Landau, Schieber, 1992).

Prefix graph – construction and usage

Vertices: s_i, the source ($s_0 = \varepsilon$), the sink ($s_{k+1} = \varepsilon$).

Edges: $s_i \xrightarrow{pr(s_i, s_j)} s_j$.

Example for $s_1 = \mathtt{aba}$, $s_2 = \mathtt{bab}$:



Construction: an optimal $O(n + k^2)$ time algorithm (Gusfield, Landau, Schieber, 1992).

Usage: paths from the source s_0 to the sink s_{k+1} correspond to common superstrings.

Steps 2 and 3: DFA and regexp

Deterministic finite automaton

We treat the prefix graph as a deterministic finite automaton A:

- states: vertices of the graph
- start state: the source (s_0)
- accept state: the sink (s_{k+1})
- alphabet Γ: triples $(i, j, |pr(s_i, s_j)|)$ for edges (s_i, s_j)
- transition function: defined by edges of the graph.

Note that the size of A is $O(poly(k)) = O(1)$.

Regular expression

Let $\alpha(A)$ be a regular expression corresponding to the language accepted by A. Note that its size is $O(1)$ and it can be computed in $O(1)$ time.

Step 4: Normal form

We transform $\alpha(A)$ into a *simpler* regular expression $\beta(A)$ such that:

- $\mathcal{L}(\beta(A)) \subseteq \mathcal{L}(\alpha(A))$
- for every $u \in \alpha(A)$ there exists $v \in \beta(A)$ which is an anagram of u.

Step 4: Normal form

We transform $\alpha(A)$ into a *simpler* regular expression $\beta(A)$ such that:

- $\mathcal{L}(\beta(A)) \subseteq \mathcal{L}(\alpha(A))$
- for every $u \in \alpha(A)$ there exists $v \in \beta(A)$ which is an anagram of u.

using the transformations (first 1 and 2 repetitively, then 3):

1. $\big(\gamma(\delta+\sigma)\rho\big)^* \to (\gamma\delta\rho)^*(\gamma\sigma\rho)^*$
2. $(\gamma\delta^*\sigma)^* \to \big(\gamma\delta^*\sigma(\gamma\sigma)^*\big) + \varepsilon$
3. $\gamma(\delta+\sigma)\rho \to (\gamma\delta\rho) + (\gamma\sigma\rho)$

Step 4: Normal form

We transform $\alpha(A)$ into a *simpler* regular expression $\beta(A)$ such that:

- $\mathcal{L}(\beta(A)) \subseteq \mathcal{L}(\alpha(A))$
- for every $u \in \alpha(A)$ there exists $v \in \beta(A)$ which is an anagram of u.

using the transformations (first 1 and 2 repetitively, then 3):

1. $\left(\gamma(\delta+\sigma)\rho\right)^* \to (\gamma\delta\rho)^*(\gamma\sigma\rho)^*$
2. $(\gamma\delta^*\sigma)^* \to \left(\gamma\delta^*\sigma(\gamma\sigma)^*\right) + \varepsilon$
3. $\gamma(\delta+\sigma)\rho \to (\gamma\delta\rho) + (\gamma\sigma\rho)$

We obtain the following normal form:

$$\beta(A) = B_1 + B_2 + \ldots + B_k$$

where $B_i = C_{i,1}C_{i,2}\ldots C_{i,l_i}$ and each $C_{i,j}$ is:

- either $a \in \Gamma$
- or $(a_1 a_2 \ldots a_p)^*$, where $a_r \in \Gamma$.

The size of $\beta(A)$ and its computation time is $O(1)$.

Towards step 5

Recall the normal form:

$$\beta(A) = B_1 + B_2 + \ldots + B_k$$

where $B_i = C_{i,1} C_{i,2} \ldots C_{i,l_i}$ and each $C_{i,j}$ is:

- either $a \in \Gamma$
- or $(a_1 a_2 \ldots a_p)^*$, where $a_r \in \Gamma$.

Towards step 5...

- We seek for the "best" word $u \in \beta(A)$.

Towards step 5

Recall the normal form:

$$\beta(A) = B_1 + B_2 + \ldots + B_k$$

where $B_i = C_{i,1} C_{i,2} \ldots C_{i,l_i}$ and each $C_{i,j}$ is:

- either $a \in \Gamma$
- or $(a_1 a_2 \ldots a_p)^*$, where $a_r \in \Gamma$.

Towards step 5...

- We seek for the "best" word $u \in \beta(A)$.
- We process each B_i separately and take the minimum.

Towards step 5

Recall the normal form:

$$\beta(A) = B_1 + B_2 + \ldots + B_k$$

where $B_i = C_{i,1} C_{i,2} \ldots C_{i,l_i}$ and each $C_{i,j}$ is:

- either $a \in \Gamma$
- or $(a_1 a_2 \ldots a_p)^*$, where $a_r \in \Gamma$.

Towards step 5...

- We seek for the "best" word $u \in \beta(A)$.
- We process each B_i separately and take the minimum.
- Within B_i the only choice is given by a Kleene's star.

Step 5: integer program

Example.

Input words: s_1 and s_2 with $m_1 = 2010$ and $m_2 = 30$

The alphabet: $\Gamma \subseteq \{0, \ldots, 3\} \times \{0, \ldots, 3\} \times (\mathbb{Z}_+ \cup \{0\})$

Consider $B_i = (0,1,0)(1,1,7)^*(1,2,3)\big((2,1,2)(1,1,7)(1,2,3)\big)^*(2,3,5)$

Step 5: integer program

Example.

Input words: s_1 and s_2 with $m_1 = 2010$ and $m_2 = 30$

The alphabet: $\Gamma \subseteq \{0, \ldots, 3\} \times \{0, \ldots, 3\} \times (\mathbb{Z}_+ \cup \{0\})$

Consider $B_i = (0,1,0)(1,1,7)^*(1,2,3)\big((2,1,2)(1,1,7)(1,2,3)\big)^*(2,3,5)$

Let us introduce variables instead of Kleene's stars:

$$(0,1,0)(1,1,7)^{x_1}(1,2,3)\big((2,1,2)(1,1,7)(1,2,3)\big)^{x_2}(2,3,5)$$

Step 5: integer program

Example.

Input words: s_1 and s_2 with $m_1 = 2010$ and $m_2 = 30$

The alphabet: $\Gamma \subseteq \{0, \ldots, 3\} \times \{0, \ldots, 3\} \times (\mathbb{Z}_+ \cup \{0\})$

Consider $B_i = (0,1,0)(1,1,7)^*(1,2,3)\big((2,1,2)(1,1,7)(1,2,3)\big)^*(2,3,5)$

Let us introduce variables instead of Kleene's stars:

$$(0,1,0)(1,1,7)^{x_1}(1,2,3)\big((2,1,2)(1,1,7)(1,2,3)\big)^{x_2}(2,3,5)$$

We wish to minimize the expression:

$$len(B_i) = 0 + 7x_1 + 3 + 12x_2 + 5,$$

under the conditions:

$$\begin{aligned} x_1, x_2 &\geq 0 \\ 1 + x_1 + 2x_2 = \#(1, B_i) &\geq m_1 = 2010 \\ 1 + x_2 = \#(2, B_i) &\geq m_2 = 30 \end{aligned}$$

Concluding **MULTI-SCS(k)**

Recall the steps:

1. words $\longrightarrow$ a graph
2. a graph $\longrightarrow$ a deterministic finite automaton
3. a DFA $\longrightarrow$ a regular expression
4. a regexp $\longrightarrow$ a regexp in a normal form
5. a regexp in a normal form $\longrightarrow$ integer program.

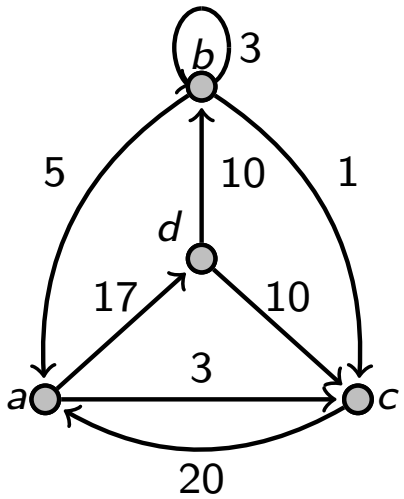
The integer program has $O(1)$ variables and conditions and thus can be solved in polynomial time in the size of input (Lenstra, 1983).

We obtain $O(poly(n))$ time solution for the **MULTI-SCS(k)** problem.

MULTI-SCS$_2(k)$ problem

MULTI-SCS$_2(k)$ problem:
Input: words s_i of length 2 and positive integers m_i.
Output: the shortest word u such that $\#occ(s_i, u) \geq m_i$ for $i = 1, 2, \ldots, k$.

We show an algorithm that computes the length of the word u in $O(n)$ time and computes its compressed representation of size $O(n^2)$ in $O(n^2)$ time (a representation of an Eulerian cycle in a multigraph).



A multigraph for a set of words

$S = \{ad, ac, ba, bb, bc, ca, db, dc\}$

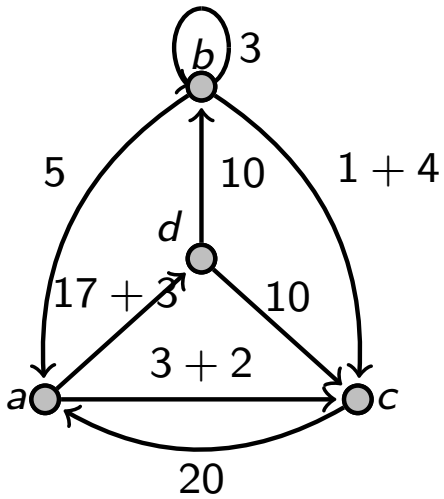
with a sequence of multiplicities

$(m_i)_{i=1}^{8} = (17, 3, 5, 3, 1, 20, 10, 10)$

MULTI-SCS$_2(k)$ problem

MULTI-SCS$_2(k)$ problem:
Input: words s_i of length 2 and positive integers m_i.
Output: the shortest word u such that $\#occ(s_i, u) \geq m_i$ for $i = 1, 2, \ldots, k$.

We show an algorithm that computes the length of the word u in $O(n)$ time and computes its compressed representation of size $O(n^2)$ in $O(n^2)$ time (a representation of an Eulerian cycle in a multigraph).



A multigraph for a set of words

$S = \{ad, ac, ba, bb, bc, ca, db, dc\}$

with a sequence of multiplicities

$(m_i)_{i=1}^{8} = (17, 3, 5, 3, 1, 20, 10, 10)$

SUM-SCS(k) problem

SUM-SCS(k) problem:
Input: words s_i and positive integer m.
Output: The word shortest u such that $\sum_{i=1}^{k} \#occ(s_i, u) \geq m$.

We present an algorithm with $O(n + k^3 \log m)$ time and $O(n + k^2 \log m)$ space complexity.

We find a shortest path in the prefix graph from s_0 to s_{k+1} containing exactly $m + 1$ vertices. For this we use min-plus product of matrices and repeated squaring starting from the adjacency matrix of the prefix graph.

The compressed representation of the SCS is given by a context-free grammar.

Conclusion

- Contribution to the SCS problem for a fixed number of words
- Assembly of a fixed number of fragments: their number is likely to become huge with Next Generation Sequencing, but their length is limited
- Next step: good approximation for a more general problem