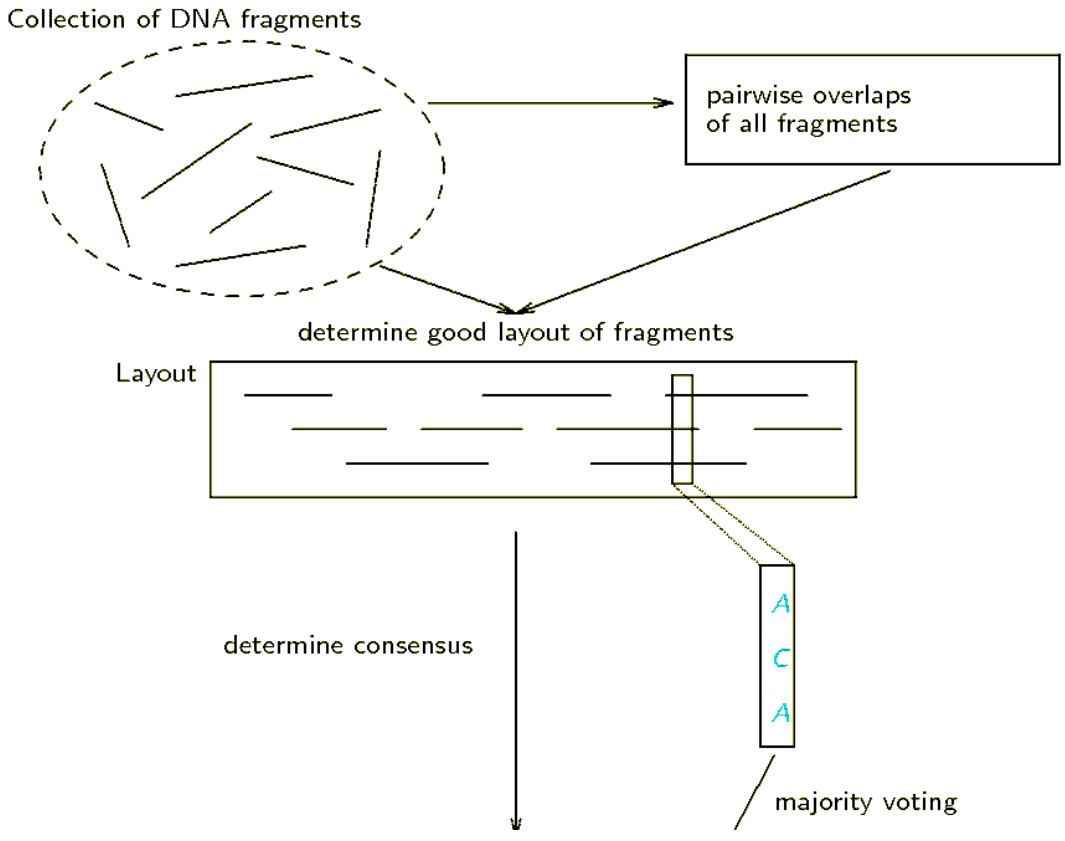
## Shotgun sequencing and fragment assembly:

### Definition

*Let $D$ be a DNA molecule to be sequenced and $S = \{s_1, \ldots, s_n\}$ the set of words (fragment sequences), observed at a shotgun sequencing of $D$. Then the* fragment assembly problem *is to determine (algorithmically) the arrangement of the words from $S$ corresponding to their original positions in $D$.*

Solution of the fragment assembly problem:

1. **Overlap:** In this phase the possible overlaps of pairs of words from $S$ are determined. They do not have to be exact prefix-suffix pairs but alignments with great similarity may also be used.
2. **Layout:** From the result of the first phase now an arrangement (similar to a semiglobal alignment) of the words from $S$ is designed, representing the arrangement of the fragments in $D$. The structure resulting from the overlaps is called *Layout*.
3. **Consensus:** As the layout usually contains several fragments overlapping at a given position of $D$ in the final step it has to be decided, which symbol is chosen. This can e.g. be done by majority voting.




## Sources of errors and problems:

- sequencing errors,
- creation of chimeras,
- uncovered parts (toxic effect of a fragments wrt. host),
- orientation ($s_i$ or complement reverse to $s_i$?),
- repeats (of (almost) identical substrings) (overlap or not?).

## Shortest superstrings

We create as layout an overlapping of the fragments that implies a sequence for $D$ as short as possible.
Formally we are looking for a word $w_S$ containing all $s_i \in S$ (superstring for $S$) for $S = \{s_1, \ldots, s_n\}$ a subword free set of words. We call this problem the *shortest common superstring problem* (SCSP).

Another view at the problem is given by the notion of compression:

### Definition

*Let $w_S$ a superstring for set $S = \{s_1, \ldots, s_n\}$. The* compression *of $w_S$ is defined by*

$$comp(w_S) := \left( \sum_{1 \leq i \leq n} |s_i| \right) - |w_S|.$$

Intuitively $comp(w_S)$ is the number of symbols that $w_S$ saves compared to the trivial superstring $s_1 \cdot s_2 \cdots s_n$.

Correspondingly the maximum compression common superstring problem (MCCSP) is the problem of finding algorithmically a superstrings for $S$ with maximal compression.

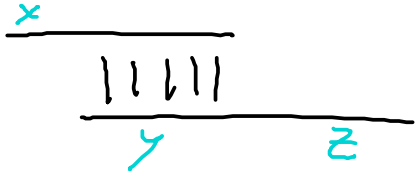
**Obviously:** Optimal solutions for SCSP and MCCSP are the same.

**But:** Performance guarantees can not be exchanged between the problems.

## Definition

*Given a set $S := \{s_1, s_2, \ldots s_n\}$, $n > 0$, of words. If there are decompositions of $s_i, s_j \in S$ satisfying*

- $s_i = xy$,
- $s_j = yz$,
- $x \neq \varepsilon$ *and* $z \neq \varepsilon$,
- $|y|$ *maximal with these properties,*




*$y$ is called* overlap *of $s_i$ and $s_j$ (notation $Ov(s_i, s_j)$). The* Merge *$< s_i, s_j >$ of $s_i$ and $s_j$ then is the word*

$$< s_i, s_j >:= xyz,$$

*and we call $x$ prefix of the merge $< s_i, s_j >$ (notation $Pref(s_i, s_j)$).*

## Definition

*Let $S = \{s_1, \ldots, s_n\}$ be a set of words and $G$ the digraph with vertices $S$ and edges $S \times S$. The* overlap graph *$OG(S)$ of $S$ is the digraph we get from marking $G$ according to*

$$ov : (S \times S) \to \mathbb{N}_0 : ov(s_i, s_j) := |Ov(s_i, s_j)|.$$

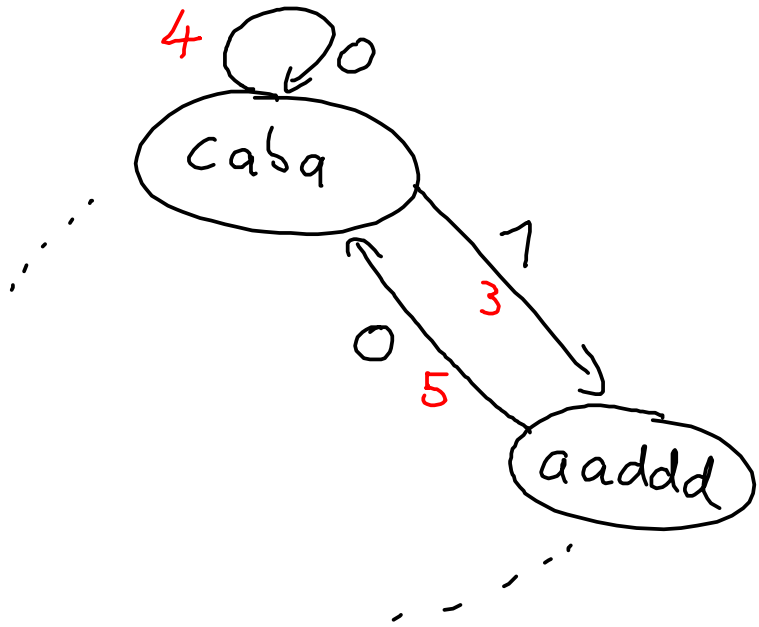
*If we mark $G$ according to*

$$pr : (S \times S) \to \mathbb{N}_0 : pr(s_i, s_j) := |Pref(s_i, s_j)|,$$

*we get the* distance graph *of $S$ (notation $DG(S)$).*

**Example:** $S = \{aabca, aacab, aaddd, ababaa, caba\}$, table shows edge weights $ov(x, y) \mid pr(x, y)$:

| | $y$ | | | | |
|---|---|---|---|---|---|
| $x$ | $aabca$ | $aacab$ | $aaddd$ | $ababaa$ | $caba$ |
| $aabca$ | 1 \| 4 | 1 \| 4 | 1 \| 4 | 1 \| 4 | 2 \| 3 |
| $aacab$ | 0 \| 5 | 0 \| 5 | 0 \| 5 | 2 \| 3 | 3 \| 2 |
| $aaddd$ | 0 \| 5 | 0 \| 5 | 0 \| 5 | 0 \| 5 | 0 \| 5 |
| $ababaa$ | 2 \| 4 | 2 \| 4 | 2 \| 4 | 1 \| 5 | 0 \| 6 |
| $caba$ | 1 \| 3 | 1 \| 3 | 1 \| 3 | 3 \| 1 | 0 \| 4 |




X ≙ overlap
Y ≙ prefix

Using our method from the exercises (9. Task) to find all pairwise overlaps of words in $S$ we can create the overlap graph for $S$ in time $\mathcal{O}(N \cdot (\log(N) + |\Sigma| + n))$, assuming $S$ is a set of words over $\Sigma$ and $N := \sum_{1 \leq i \leq n} |s_i|$.

Using $pr(s_i, s_j) = |Pref(s_i, s_j)| = |s_i| - |Ov(s_i, s_j)|$ we get the same running time for creating the distance graph.

Note that $n^2 = \mathcal{O}(N \cdot n)$ and even $n^2 \leq n \cdot N$ holds as by assumption $\varepsilon$ being a substring of every word can't be an element of $S$.

Edge $s_i \rightarrow s_j$ in the overlap or distance graph can be identified with merge $< s_i, s_j >$.
$\Rightarrow$ Path corresponds to series of merges of the words represented by the nodes. E.g. for path $s_1, s_2, \ldots, s_k$

$$< s_1, s_2, \ldots, s_k >:= Pref(s_1, s_2) \cdot Pref(s_2, s_3) \cdots Pref(s_{k-1}, s_k) \cdot s_k.$$

A superstring for $S$ then corresponds to a path through the graph visiting each node exactly once, a minimal superstring (i.e. a solution for SCSP or MCCSP) to a path with optimal weight (for the distance graph this is the minimal weight).

$\Rightarrow$ Correspondence of our problems and TSP!

**Theorem**
*Dec-SCSP is NP-complete.*

□

**Approximation algorithm:**

**while** |S|>1 **do**
    Determine $s_i, s_j \in S$, $s_i \neq s_j$, **with** ...
    ...maximal overlap **of** all pairs **in** $S$.
    **Set** $s' :=< s_i, s_j >$ **and** $S = S \setminus \{s_i, s_j\} \cup \{s'\}$.
return $(s \in S)$ //the only word remaining **in** $S$

This way the algorithm creates a hamiltonian cycle in the overlap graph.

**Example:** $S = \{aabca, aacab, aaddd, ababaa, caba\}$ (Graph see previous example)
First choice: Two alternatives ($(caba, ababaa)$ and $(aacab, caba)$).
We take $(caba, ababaa) \Rightarrow$ merge $< caba, ababaa >= cababaa$ and $S = \{cababaa, aabca, aaddd, aacab\}$.

Second step: Combine $aacab$ and $cababaa$ to get $aacababaa$ and $S = \{aacababaa, aabca, aaddd\}$.

Third step: Pair $(aacababaa, aaddd)$ has maximal overlap.
$\Rightarrow S = \{aacababaaddd, aabca\}$

Last step: Output *aabcaacababaaddd* of length 16 (being a shortest superstring for the input considered).

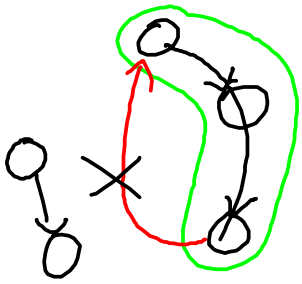
## Theorem

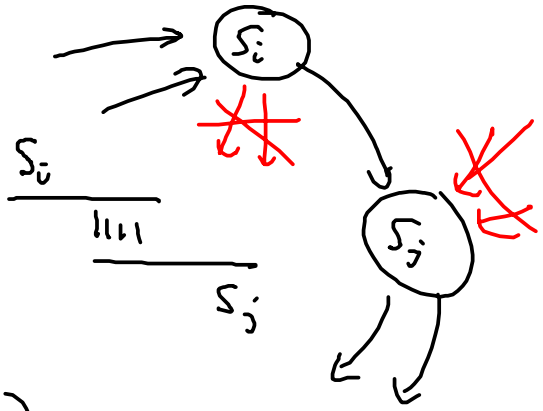
*Assuming a constant size of the alphabet above greedy algorithm given input* $S = \{s_1, \ldots, s_n\}$ *takes running time in* $\mathcal{O}(N \cdot (n + \log(N)))$, $N = \sum_{1 \le i \le n} |s_i|$.

**Proof:**

- Konstruktion Overlap Graphen: $\mathcal{O}(N(n + \log(N)))$
- Sortiere Kanten gem. Gewichte durch distribution Counting (möglich da Gewichte mit bekannter oberer Schranke)

$$\mathcal{O}(n^2 + N) = \mathcal{O}(n \cdot N) \checkmark$$






- UNION-FIND zum Verhindern von Kreisen
- Markierung von Kanten die nicht weiter verwendet

werden dürfen (2n-1 viele pro Iteration) in Adjazenzmatrix.

## 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:**

$W_o \triangleq$ sei optimale Lösung

1. Beobachtung: Jedes $S_i$ kommt in $W_o$ mind. einmal als Teilwort vor.

$\Longrightarrow$ Reihenfolge $(S_{i_1}, \ldots S_{i_n})$

d.h. $W_o = \langle \ldots \langle\langle\langle S_{i_1}, S_{i_2}\rangle, S_{i_3}\rangle, S_{i_4}\rangle \rightarrow$

$comp(W_o) = \sum$ zug. Overlaps

Analog für Ausgabe von Greedy: $(S_{j_1}, \ldots S_{j_n})$

mit zug. $Comp(W_g)$.