Stochastic modeling: How good do randomly chosen fragments cover a molecule?

Definition

Let $A : \mathbb{R}_0 \rightarrow \mathbb{N}$ a nondecreasing function satisfying $A(0) = 0$, where $A(t)$ describes the number of events until time t. Then we have a poisson process with rate λ, if

(1) $\Pr[A(s+t) - A(s) = n] = \exp(-\lambda \cdot t)\frac{(\lambda \cdot t)^n}{n!}$ *and*

(2) *the distribution of the number of events is stationary, i.e. it depends only on the length but not on the position of a given interval.*

Theorem

Let A be a poisson process.

a) *The expected number $\mathbb{E}[A(t)]$ of events in an interval of length t satisfies $\mathbb{E}[A(t)] = \lambda \cdot t$.*

b) *Let T_n be the time between the $(n-1)$-th and the n-th event. Then*

$$\Pr[T_1 > t] = \Pr[T_n > t] = \exp(-\lambda \cdot t).$$

Model: Assuming fragments of length L are cut from multiple copies of a DNA molecule of length C randomly and independently. Then for i any position of the molecule

$$\Pr[i \text{ is covered by randomly chosen fragment}] = \frac{L}{C}$$

holds and thus

$$\left(1 - \frac{L}{C}\right)^N \approx \exp\left(-\frac{L \cdot N}{C}\right) \qquad (1)$$

is the probability that i is not covered by any of the N fragments.

$$\exp(x) = \lim_{n \to \infty} \left(1 + \frac{x}{n}\right)^n$$

Poisson process?

We let $\lambda := \frac{L}{C}$ and ask for the probability of exactly n of the N fragments covering position i. This probability is given by

$$\begin{aligned}\binom{N}{n}\cdot\left(\frac{L}{C}\right)^{n}\cdot\left(1-\frac{L}{C}\right)^{N-n} &\approx \frac{N!}{n!\cdot(N-n)!}\lambda^{n}\cdot\exp(-\lambda\cdot N)\\ &= \frac{N!}{N^{n}\cdot(N-n)!}\exp(-\lambda\cdot N)\cdot\frac{(\lambda\cdot N)^{n}}{n!}\\ &\approx \exp(-\lambda\cdot N)\cdot\frac{(\lambda\cdot N)^{n}}{n!}.\end{aligned}$$

So the number of fragments covering a fixed position is approximately poisson distributed with rate $\lambda = \frac{L}{C}$. (This statement holds for $n \ll N \ll C$ and $L \ll C$.)

The expected number of fragments covering position i is thus $R := \lambda \cdot N = \frac{L\cdot N}{C}$. R is called *redundancy* of the fragment set.

Corollary

The expected number of positions not covered is (approximately) given by

$$\exp\left(-\frac{L\cdot N}{C}\right)\cdot C = \exp(-R)\cdot C.$$

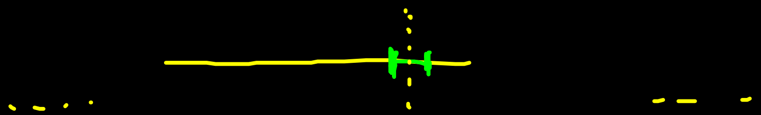
□

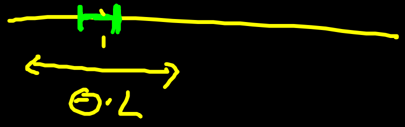
Definition

Let $\mathcal{F}$ be a set of fragments of length L and $\Theta \in [0,1]$. We take $\mathcal{F}$ as vertices of a graph and connect two fragments $f_1, f_2 \in \mathcal{F}$ by an indirected edge, if a suffix (prefix) of f_1 of length at least $\Theta \cdot L$ is a prefix (suffix) of f_2 (overlap). We get an undirected graph whose connected components are called Θ-islands.

Intuition: Fragments with only small overlap should not be considered overlapping.

How many Θ-islands are to be expected?





Lemma

Let $\Theta \in [0,1]$ and redundancy R be given and let N be the number of randomly chosen fragments. Then

$$N \cdot \exp(-R \cdot (1-\Theta))$$

is (approximately) the expected number of Θ-islands.

Proof: C ≙ length of molecule

N ≙ # of fragments

L ≙ length of fragment

$\lambda := L/C$

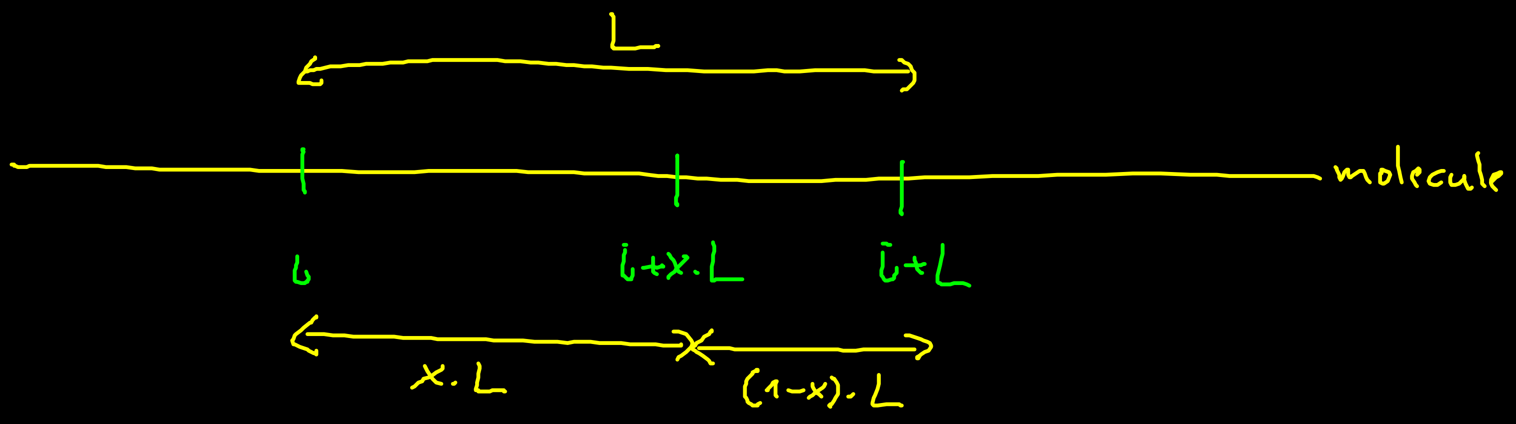
Let i be any position of the molecule

We define

$J(\underline{x}) := \Pr[$positions i and $i+\underline{x}\cdot L$ are covered by randomly chosen fragment$]$

Claim: $$J(x) = \begin{cases} \exp(-R(1-x)) & \text{for } 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases}$$

with $R = \lambda \cdot N = \frac{L \cdot N}{C}$ the redundancy.



$\Longrightarrow$ The number of different locations for a random fragment to cover both positions is given by $(1-x)\cdot L$

$\rightsquigarrow$ $\Pr[i$ and $i+x\cdot L$ to be covered by random fragment$]$

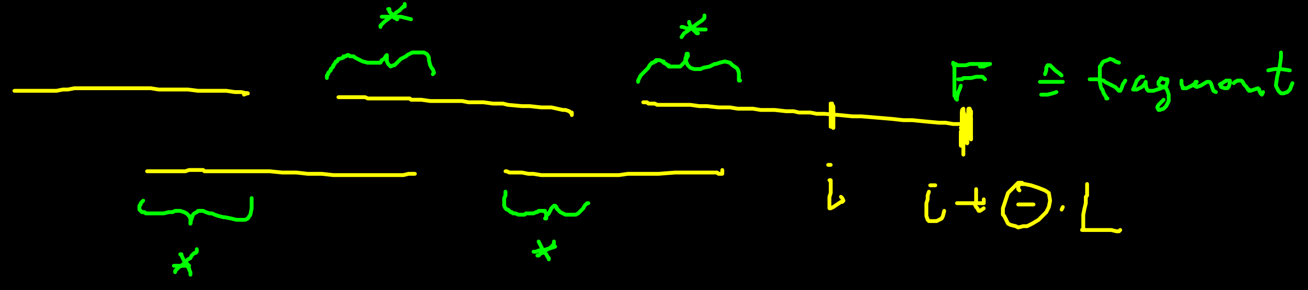
$$= \frac{(1-x)\cdot L}{C}$$

$\rightsquigarrow$ $\Pr[i$ and $i+x\cdot L$ are not covered by any of the N fragments$]$

$$= \left(1 - \frac{(1-x)\cdot L}{C}\right)^{N} \underset{R=\frac{L\cdot N}{C}}{=} \left(1 - \frac{(1-x)\cdot R}{N}\right)^{N}$$

$$\xrightarrow{N\to\infty} \; = \exp\left(-(1-x)\cdot R\right)$$

Islands:



$$* \hat{=} \geq \Theta\cdot L$$

Observation: F is the only fragment which covers positions i and $i+\Theta\cdot L$ since otherwise it would not be the rightmost element of the considered island.

$$\mathbb{E}[\#\Theta\text{-islands}] = \mathbb{E}[\#\text{rightmost fragments}]$$

↷ we are looking for

$$\mathbb{E}[\#\text{fragments covering positions } i \text{ and } i+\Theta L \text{ exclusively}]$$

$$= N\cdot J(\Theta) \approx N\cdot \exp(-R(1-\Theta))$$

Since $J(\Theta) \hat{=} \Pr[\text{random fragment does } \underline{\text{not}} \text{ overlap the rightmost fragment}]$

$\hat{=} \Pr[\text{fragment is the last of an island}]$

Example: We consider the case of a molecule with 10^8 bases to be mapped. We assume that a library of 10000 fragments has been created, each around 50000 bases long. In this case $R = \frac{5 \cdot 10^4 \cdot 10^4}{10^8} = 5$ and for Θ small enough $N \cdot \exp(-R \cdot (1 - \Theta)) \approx 10^4 \cdot \exp(-5) = 67.37946999\ldots$ many islands are to be expected.

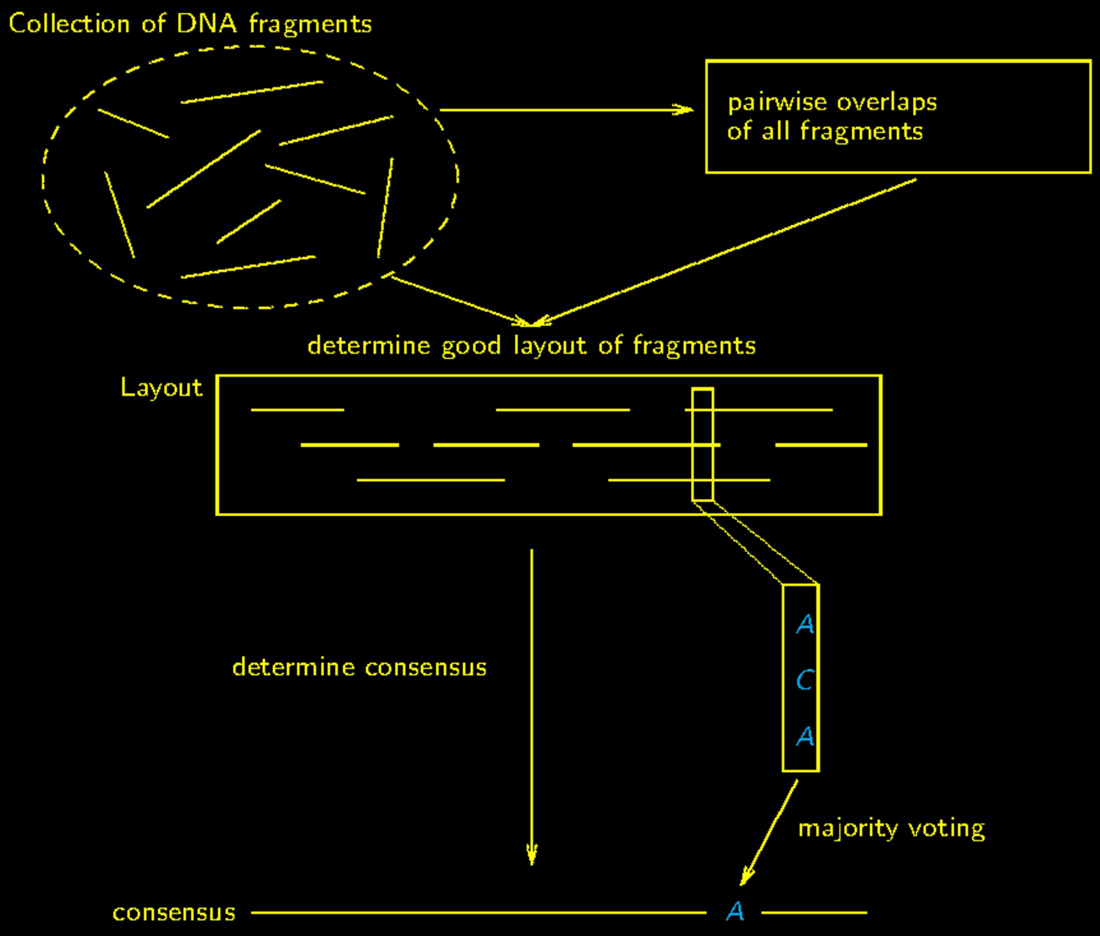
Shotgun sequencing and fragment assembly:

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.


Collection of DNA fragments
pairwise overlaps of all fragments
determine good layout of fragments
Layout
determine consensus
A
C
A
majority voting
consensus
A

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

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

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.

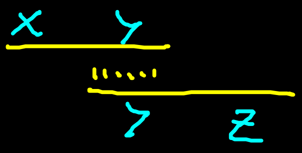
Definition

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

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

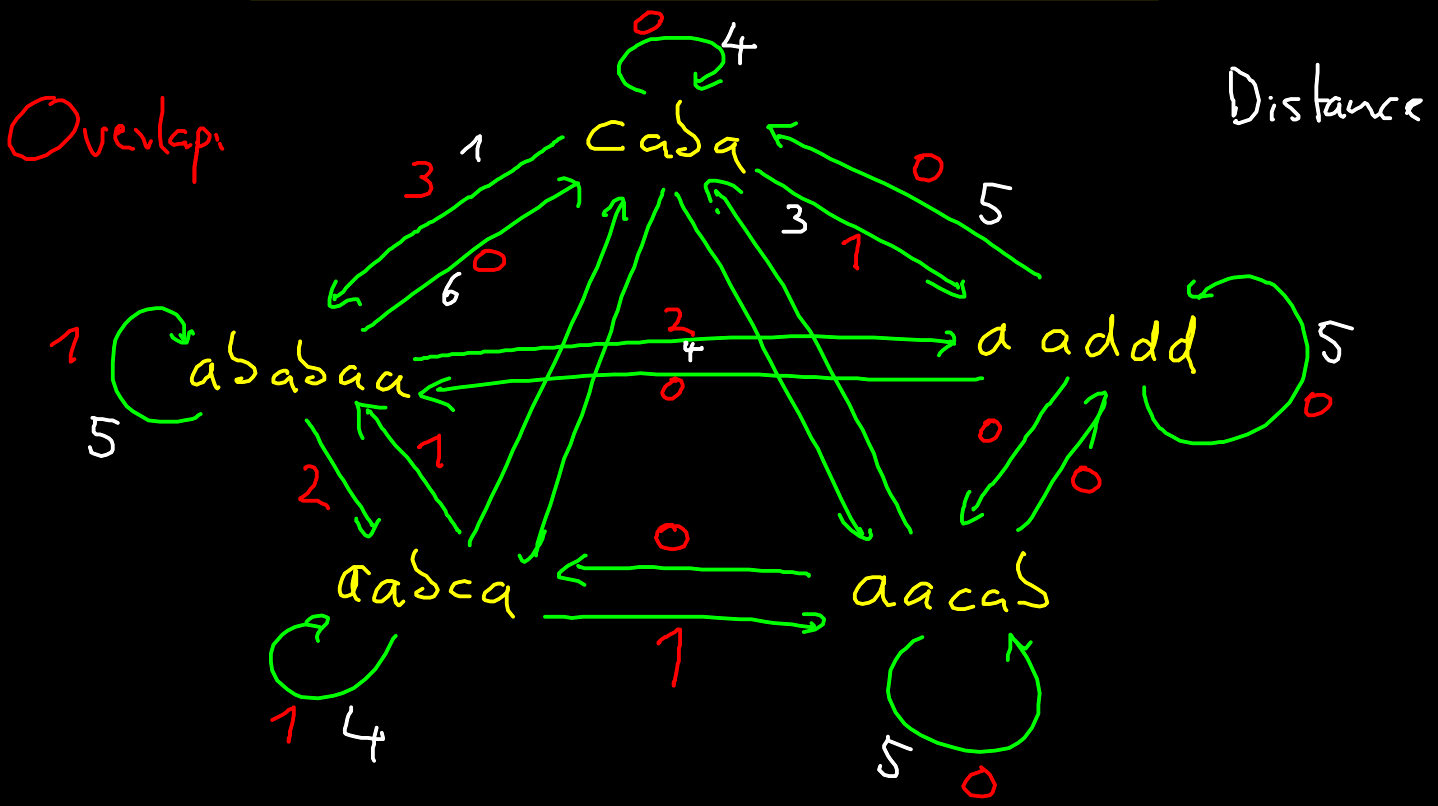
we get the distance graph *of S (notation $DG(S)$).*

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

Greedy:

aacaba

	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



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 Σ 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 ε being a substring of every word can't be an element of S.

Edge $s_i \longrightarrow s_j$ 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 $s_1, s_2, \ldots, s_k$

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

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:

Constant alphabet size → compute all pairwise overlaps in time $O(N \cdot (n + \log(N)))$ [suffix trees]

Sort the n^2 edges in time $O(n^2 + N)$ $= O(n \cdot N)$ (according to label) using distribution counting (labels are bounded above by length of the longest input sequence and (l-s by N)

Graph: time $O(n^2)$ adjacency matrix.

UNION-FIND (to prevent cycles):

Array $R[1:n]$ with $R[i]$ = name of partition i belongs to

Partition: Linear List (storing its length)

Initialize: $\Theta(n)$; n singletons $\{s_i\}$

$\Longrightarrow$ initialization $O(N \cdot (n + \log(N)))$