Algorithmische Bioinformatik

Teile der Vorlesung von Prof. Dr. Martin Vingron
Wintersemester 2003, FU-Berlin

Prof. Dr. Knut Reinert
reinert@inf.fu-berlin.de

Skript mit Beiträgen von:
Prof. Dr. Daniel Huson (Tübingen)
Prof. Dr. Knut Reinert (Berlin)

0.1 Organisatorisches

Die Vorlesung wird zum größten Teil auf Folien in englischer Sprache formuliert. Dabei sollten Sie folgende Punkte beachten:

- Sie bekommen einen Ausdruck (und/oder Quellcode) aller Folien. Eine Mitschrift des Folieninhaltes ist **nicht nötig**.
- Die Folien sind **kein** vollständiges Skript. Bitte machen Sie sich Notizen und **ergänzen** Sie die Ausdrucke oder Quellcode.
- Die Vorlesung wird auf Deutsch gehalten, es sei denn, Englisch ist von allen erwünscht.

Sprechstunden: Freitag 15:30–16:30 Fabeckstrasse 15 Tel.: 75221

Mail: reinert@inf.fu-berlin.de, uschild@inf.fu-berlin.de

0.2 Content

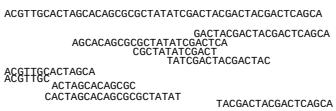
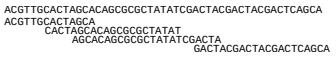
1. Physical Mapping
2. (Whole) Genome Shotgun Sequence Assembly
3. Sequencing by Hybridization (if time allows)
4. Multiple sequence alignment

1 Physical Mapping (Knut Reinert)

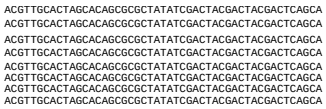
This exposition is based on the following sources, which are all recommended reading:

1. Pevzner, Computational Molecular Biology, MIT Press, 2000, chapter 2,3.
2. Setubal und Meidanis, Introduction to Computational Molecular Biology, PWS Publishing, 1997, chapter 5.
3. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge University Press, 1997, chapter 16.
4. Michael S. Waterman, Introduction to computational biology, Chapman and Hall, 1995, chapters 2,3,6.

1.1 The big picture – From molecule to sequence

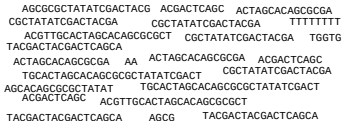
Whole Genome Shotgun sequencing	illustration	Clone by clone sequencing
Source sequence (target) (≈ 3000 Mbp for human)		Source sequence (target)
Not done in WGS		is broken into smaller pieces (150–1000kbp)
Not done in WGS		Big pieces are selected using different protocols to tile the target (minum tiling least costly but most difficult) ⇒ Physical mapping

Big source sequence is copied many times...



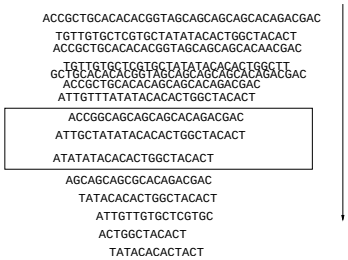
all source sequences are copied many times (e.g. 40000 for human)

and randomly broken into fragments, e.g. using *sonication* or *nebulation*, ...



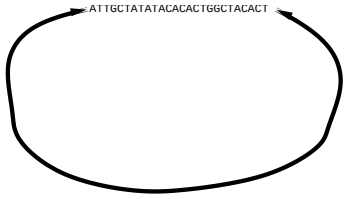
each sequence is randomly broken into fragments

that are then size selected, size e.g. 2kb, 10kb, 50kb or 150kb, ...

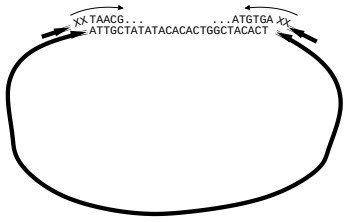


that are then size selected

and inserted into cloning vectors.



and *all* inserted into cloning vectors.



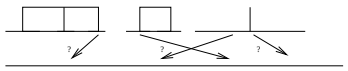
In *double barrel shotgun sequencing*, each clone is sequenced from both ends, to obtain a *mate-pair* of reads, each read of average length 550 with $\approx 1\%$ error

first approaches did not use double barrel, later they did.

Result of assembly is a collection of *scaffolds* for the *whole genome*.

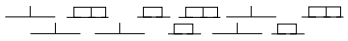
Each clone is a collection of scaffolds.

Ordering is quite difficult, since small pieces are hard to *map* back to the genomic axis



Local ordering is relatively easy.

Not done in WGS



The sequence of *all* clones has to be asssembled according to the physical map and sequence overlaps. Due to repeats and assembly errors this is hard.

1.2 Physical mapping

A *physical map* of a target tells us the location of certain *markers* along the genome. The markers are used for navigating in the genome. For example if I have a piece of DNA with some known markers, I can pinpoint it on a chromosome giving me its genomic context for further exploration.

Generally we use *fingerprinting* techniques such as restriction enzymes and hybridization experiments to known sequence to determine overlaps of clones or to order non-overlapping pieces.

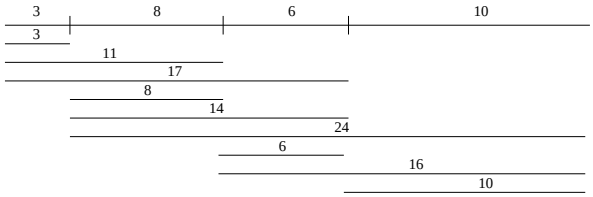
1.3 Restriction maps

To build a restriction map, different *biochemical* techniques are used to derive information about the map and *combinatorial* methods are used to reonstruct the map from that data.

Most restriction mapping problems correspond to the following problem: If X is a set of points on the line, let ΔX denote the multiset of *all* pairwise distances between points in X that means $\Delta X = \{\mid x_1 - x_2 \mid : x_1, x_2 \in X\}$. In restriction mapping some subset $E \subseteq \Delta X$ corresponding to the experimental data about fragment lengths is given and the problem is to reconstruct X from E.

1.4 Partial digest problem

For the *partial digest problem* (PDP), the experiment provides data about *all* pairwise distances between restriction sites i.e. $E = \Delta X$

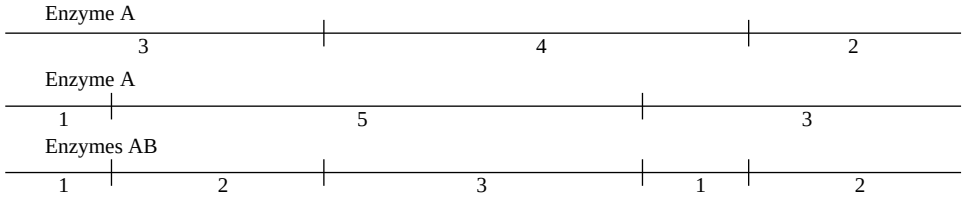


For example the above PDP problem has $\Delta X = \{3, 6, 8, 10, 11, 14, 16, 17, 24\}$

- No polynomial time algorithm for the PDP is known.
- Skiena devised a simple backtracking algorithm that performs well in practice but might require exponential time
- PDP is not the favorite mapping method since it is difficult to produce *every* pairwise distance.

1.5 Double digest problem

For the *double digest problem* (DDP), the experiment provides data about the *complete* digest, i.e. *all consecutive* restriction sites for two different restriction enzymes A and B applied alone and in combination yielding $\Delta A, \Delta B$ and ΔAB.



The above DDP problem has a unique solution (but only given ΔAB, Try $\Delta AB = \{1, 1, 1, 2, 4\}$). In the exercises you are asked to construct an instance that does not have a unique solution.

- DDP is NP-complete.
- All algorithms have problems with more than 10 restriction sites for each enzyme.
- Solution is not unique and number of solutions grows exponentially.
- DDP is the favorite mapping method since the experiments are easy to conduct.

1.6 Hybridization mapping

Hybridization mapping makes use of the fact, that small genomic sequences can be tested for presence in a clone (e.g. using an hyridization experiment or PCR).

There are two sorts of probes. Unique probes like STS (Sequence Tagged Site) or non-unique probes. Both give rise to different algorithmic problems. We will concentrate on unique probes.

Given unique probes, two protocols are commonly used, *STS content mapping* and *radiation hybrid mapping.*

1.7 STS content mapping

Given a set U of probes and a collection of subsets $\Phi = \{S_1, \ldots, S_n\}, \forall i, S_I \subseteq U$.

Problem:
Find a permutation $\Pi(\Phi)$ of U along which every S_i is contiguous.

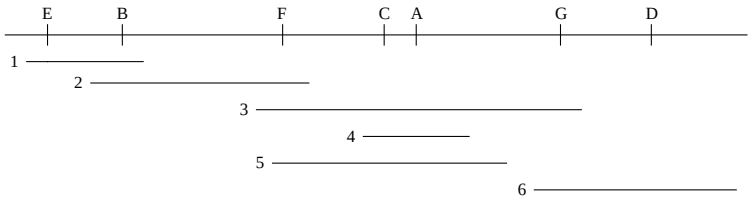
For example, assume the following incidence matrix is given. An entry in line i and row j is 1 if clone i hybridizes to probe j.

c/p	A	B	C	D	E	F	G
1	0	1	0	0	1	0	0
2	0	1	0	0	0	1	0
3	1	0	1	0	0	1	1
4	1	0	1	0	0	0	0
5	1	0	1	0	0	1	0
6	0	0	0	1	0	0	1

The probes $A, \ldots, G$ can be permuted as follows:

c/p	E	B	F	C	A	G	D
1	1	1	0	0	0	0	0
2	0	1	1	0	0	0	0
3	0	0	1	1	1	1	0
4	0	0	0	1	1	0	0
5	0	0	1	1	1	0	0
6	0	0	0	0	0	1	1

g This implies the following layout:



Now all probes are consecutive for each clone. The matrix has the *consecutive ones property*. The solution(s) can be computed in linear time and represented in a data structure called $PQ-tree$. Not only we have ordered all clones, but we determined also an ordering of the STSs.

Unfortunately the hybridization experiments are very likely not error-free. The following errors can occur:

- *false positives*. A false positive reports a clone has probe when in fact it does not.
- *false negatives*. A false negative reports a clone has *not* a probe when in fact it does.
- *chimeras*. Chimeric clones join pieces of DNA together that are far away on the genomic axis and hence bring unrelated probes near together.

The below matrix depicts a correctly ordered probe set with a false negative in clone 3, a false positive in clone 1, and a possible chimeric clone 6.

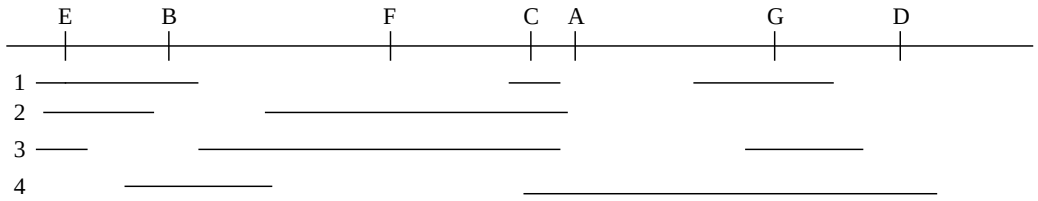
c/p	E	B	F	C	A	G	D
1	1	1	0	0	1	0	0
2	0	1	1	0	0	0	0
3	0	0	1	0	1	1	0
4	0	0	0	1	1	0	0
5	0	0	1	1	1	0	0
6	1	0	0	0	0	1	1

Before we discuss how to handle this errors, we describe the second method, radiation hybrid mapping, since it yields similar data.

1.8 radiation hybrid mapping

In radiation hybrid mapping the target chromosome is irradiated and broken into a small number of fragments. These non-overlapping fragments are merged into a rodent (e.g. hamster) cell.

In successive replications each hamster cell replicates less and less of the foreign DNA. In the end, each hamster cell contains 5 to 10 large, disconnected, non-overlapping fragments of a single human chromosome. This is done several times with different random irradiation results.



Now it can be determined which cell hybridizes with which probes (see figure). This is very similar to STS-content mapping, except that we do not know in which fragment the probe is, nor how many fragments there are.

The following matrix show the data from the above radiation hybrid experiment:

c/p	E	B	F	C	A	G	D
1	1	1	0	0	1	0	1
2	0	1	0	1	1	1	0
3	0	1	0	1	1	0	0
4	1	1	1	0	0	0	1

What is a sensible objective function to find the correct permutation of probes?

Since there are still some probes likely to be consecutive on a fragment, we try to minimize the total number of blocks of consecutive ones.

1.9 TSP solution for consecutive ones with gaps

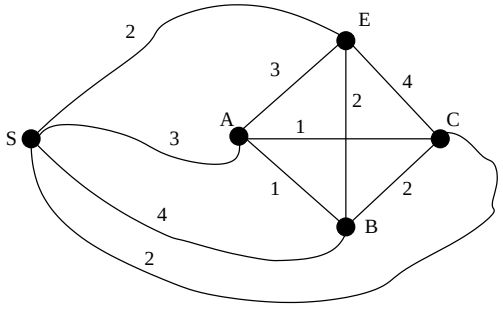
We reduce the problem of finding the probe permutation with the minimum number of consecutive ones to the travelling salesman problem as follows:

1. Define a weighted graph $G = (V, E)$ with $V = \{s, p_1, \ldots, p_k\}$ where p_i is a node for each probe i and s is a special node.
2. E contains an edge from s to each p_i and an edge for each pair of probes.
3. the weight of the edges from s to the p_i is the number of ones in the corresponding column of the matrix. The weight of any other edge (p_i, p_j) is the *Hamming distance* bwetween the columns corresponding to p_i and p_j.

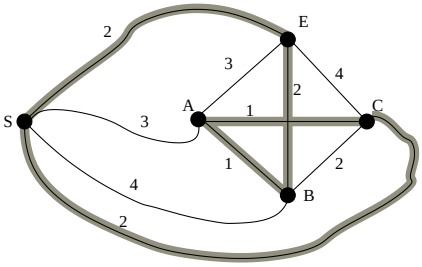
For the sake of exposition we look at the submatrix from the previous example.

c/p	E	B	C	A
1	1	1	0	1
2	0	1	1	1
3	0	1	1	1
4	1	1	0	0

This translates into the following graph G.



The solution unfortunately does not give the correct ordering. The optimal tour s, C, A, B, E contains no gap however.



c/p	C	A	B	A
1	0	1	1	1
2	1	1	1	0
3	1	1	1	0
4	0	0	1	1

Theorem 1. *The TSP tour of weigth w corresponds to a probe permutation that with exactly $w/2$ blocks of consecutive ones.*

Proof: Each TSP tour corresponds to a probe permutation. Except for the edges incident to s, a tour is charged the Hamming distance if it traverses edge (p_i, p_j). For the combination $(0, 1)$ it is charged for the *beginning* of a new block induced by ordering p_i before p_j, and for the combination $(1, 0)$ it is charged for the *ending* of a block. Hence each block is charged 1 for its begin and end. The weights from s to each node counts the number of blocks ending in the rightmost resp. starting in the leftmost column.

1.10 Summary

Physical mapping comes in two flavors:

1. Restriction mapping. Here restriction enzymes are used to digest the target into smaller pieces. Using the partial or double digest protocol certain sets of distances between restriction sites are constructed. The goal is to explain all distances. Sometimes restriction mapping is also used to determine whether two clones overlap.
2. Hybridization mapping. The goal is here to determine the order of *overlapping* clones. The hybridization signature of short (possibly unique) sequences is determined. The goal is to determine a possibly minimal tiling path so that only few clones need to be sequenced.

There are different protocols to determine a map, each suitable in different situations. Each protocol has its associated algorithmic problem. Most of them are in exact form already NP-hard. Errors need to be taken into account.

You should know:

- The partial digest problem
- The double digest problem
- STS content mapping
- Radiation Hybrid mapping

We talked about solving the STS content mapping and radiation hybrid mapping by reducing it to the TSP problem

2 Sequence Assembly (Daniel Huson)

The slides are largely based on Prof. Daniel Huson's lecture at the University of Tübingen. The exposition is based on the following sources, which are all recommended reading:

1. Michael S. Waterman, Introduction to computational biology, Chapman and Hall, 1995. (Chapter 7)
2. Eugene W. (Gene) Myers *et al.*, A Whole-Genome Assembly of *Drosophila*, Science, 287:2196-2204, 24 March 2000.
3. Venter *et al.*, The sequence of the Human Genome, Science, 291:1304-1351, 16 February 2001.
4. Daniel Huson, Knut Reinert and Eugene Myers, The Greedy-Path Merging Algorithm for Sequence Assembly, RECOMB 2001, 157-163, 2001.

2.1 Genome Sequencing

Using a method that was basically invented in 1980 by Sanger, current sequencing technology can only determine $500 - 1000$ consecutive base pairs of DNA in any one read. To sequence a larger piece of DNA, *shotgun sequencing* is used.

Originally, shotgun sequencing was applied to small viral genomes and to $30 - 40kb$ segments of larger genomes.

In 1994, the $1.8Mb$ genome of the bacteria *H.influenzae* was assembled from shotgun data.

At the beginning of 2000, am assembly of the $130Mb$ Drosophila genome was published.

At the beginning of 2001, two initial assemblies of the human genome were published.

2.2 Shotgun sequencing data

Given an unknown DNA sequence $a = a_1 a_2 \dots a_L$.

Shotgun sequencing of a produces a set of reads

$$\mathcal{F} = \{f_1, f_2, \dots, f_R\},$$

of average length 550 (at present).

Essential characteristics of the data:

- Incomplete coverage of the source sequences.

- Sequencing introduces errors at a rate of about %1 for the first 500 bases, if carefully performed.
- The reads are sampled from both strands of the source sequence and thus the orientation of any given read is unknown.

2.3 The fragment assembly problem

The input is a collection of reads (or *fragments*) $\mathcal{F} = \{f_1, f_2, \ldots, f_R\}$, that are sequences over the alphabet $\Sigma = \{A, C, G, T\}$.

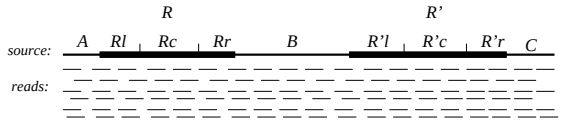
An ϵ-*layout* of $\mathcal{F}$ is a string S over Σ and a collection of R pairs of integers $(s_j, e_j)_{j \in \{1,2,\ldots,R\}}$, such that

- if $s_j < e_j$ then f_j can be aligned to the substring $S[s_j, e_j]$ with less than $\epsilon \cdot |f_j|$ differences, and
- if $s_j > e_j$ then f_j can be aligned to the substring $\overline{S[e_j, s_j]}$ with less than $\epsilon \cdot |f_j|$ differences, then
- $\cup_{j=1}^{R} [\min(s_j, e_j), \max(s_j, e_j)] = [1, |S|]$.

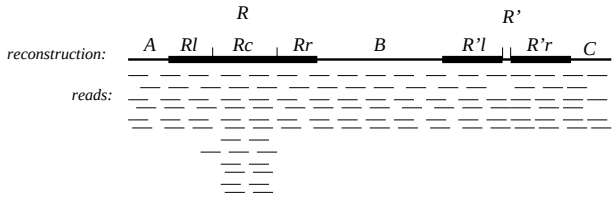
The string S is the reconstructed source string. The integer pairs indicate where the reads are placed and the order of s_i and e_i indicate the orientation of the read f_i, i.e. whether f_i was sampled from S or its complement $\overline{S}$.

The set of all ϵ-layouts models the set of all possible solutions. There are many such solutions and so we want a solution that is in some sense *best*. Traditionally, this has been phrased as the *Shortest Common Superstring Problem (SCS)* of the reads within error rate ϵ. Unfortunately, the SCS Problem often produces overcompressed results.

Consider the following source sequence that contains two instances R, R' of a high fidelity repeat and three stretches of unique sequence A, B and C:



The shortest answer isn't always the best and the interior part $R_c \approx R'_c$ of the repeat region is *overcompressed*:



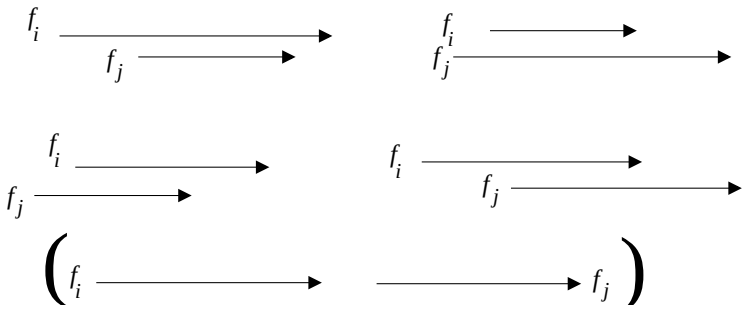
2.4 Sequence assembly in three stages

Traditional approaches to sequence assembly divides the problem into three phases:

1. In the *overlap* phase, every read is compared with every other read, and the overlap graph is computed.
2. In the *layout* phase, the pairs (s_j, e_j) are determined that position every read in the assembly.
3. In the *consensus* phase, a multialignment of all the placed reads is produced to obtain the final sequence.

2.5 The overlap phase

For a read f_i, we must calculate how it overlaps any other read f_j (or its reverse complement, $\overline{f_j}$). Holding f_i fixed in orientation, f_i and f_j can overlap in the following ways:



The number of possible relationships doubles, when we also consider $\overline{f_j}$.

The overlap phase is the computational bottleneck in large assembly projects. For example, assembling all 27 million human reads produced at Celera requires

$$2 \cdot \binom{27000000}{2} \approx 1458000000000000$$

comparisons.

For any two reads a and b (and either orientation of the latter), one searches for the overlap alignment with the highest alignment score, based on a similarity score $s(a, b)$ on Σ and an indel penalty $g(k) = k\delta$.

Let $S(a, b)$ be the maximum score over all alignments of two reads $a = a_1 a_2 \dots a_m$ and $b = b_1 b_2 \dots b_n$, we want to compute:

$$A(a,b) = \max \left\{ S(a_k, a_{k+1} \dots a_i, b_l b_{l+1} \dots b_j) \mid \left\{ \begin{array}{l} 1 \le k \le i \le m, \\ 1 \le l \le j \le n, \\ \text{and } i = m \text{ or } j = n \text{ holds} \end{array} \right\} \right\}.$$

2.6 Overlap alignment

This is a standard pairwise alignment problem (similar to local alignment, except we don't have a 0 in the recursion) and we can use dynamic programming to compute:

$$A(i,j) = \max\{S(a_k, a_{k+1} \dots a_i, b_l b_{l+1} \dots b_j) \mid 1 \le k \le i \text{ and } 1 \le l \le j\}.$$

Algorithm (Overlap alignment)
Input: $a = a_1 a_2 \dots a_n$ and $b = b_1 b_2 \dots b_m$, $s(\cdot,\cdot)$ and δ
Output: $A(i,j)$
begin

$A(0,j) = A(i,0) \leftarrow 0$ for $i = 1, \dots, n$, $j = 1, \dots, m$
for $i = 1, \dots, n$:
for $j = 1, \dots, m$:

$$A(i,j) \leftarrow \max \left\{ \begin{array}{l} A(i-1,j) - \delta, \\ A(i,j-1) - \delta, \\ A(i-1,j-1) + s(a_i, b_i) \end{array} \right\}$$

end

Runtime is $O(nm)$.

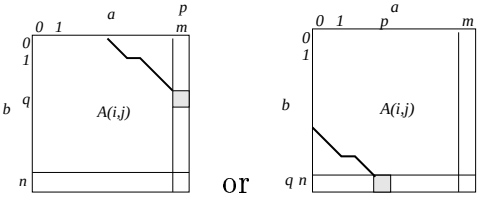
Given two reads $a = a_1 a_2 \dots a_m$ and $b = b_1 b_2 \dots b_n$. For the matrix $A(i,j)$ computed as above, set

$$(p,q) := \arg\max\{A(i,j) \mid i = m \text{ or } j = n\}.$$

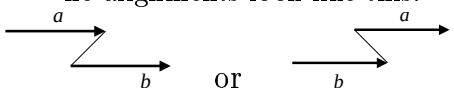
There are two cases:

$p = m$ or $q = n$

The trace-back paths look like this:



The alignments look like this:



2.7 Faster overlap detection

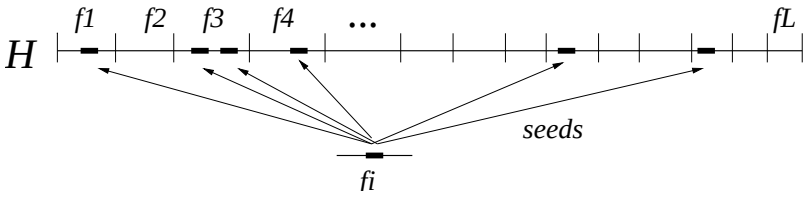
Dynamic programming is too slow for large sequencing projects. Indeed, it is wasteful, as in assembly, only high scoring overlaps with more than e.g. 96% identity, play a role.

One can use a *seed and extend* approach (as used in BLAST):

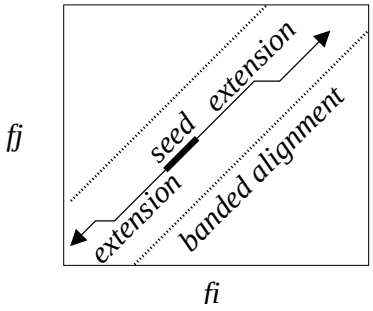
1. Produce the concatenation of all input reads $H = f_1 f_2 \dots f_L$.
2. For each read $f_i \in \mathcal{F}$: Find all *seeds*, i.e. exact matches between k-mers of f_i and the concatenated sequence H. (Merge neighboring seeds.)
3. Compute *extensions*: Attempt to extend each (merged) seed to a high scoring overlap alignment between f_i and the corresponding read f_j.

(A *k-mer* is a string of length k. In this context, $k = 18 \dots 22$)

Computation of seeds:



Extension of seeds using *banded* dynamic programing (running time is linear in the read length):



2.8 True and repeat-induced overlaps

Assume that we have found a high quality overlap o between f_i and f_j. There there are three possible cases:

- The overlap o corresponds to an overlap of f_i and f_j in the source sequence. In this case we call o a *true* overlap.
- The reads f_i and f_j come from different parts of the source sequence and their overlapping portions are contained in different instances of the same repeat, this is called a *repeat-induced* overlap.
- The overlap exists by chance. To avoid short random overlaps, one requires that an overlap is at least 40bp long.

True overlap between f_i and f_j, repeat induced overlap between f_k and f_l.

2.9 Avoiding repeat-induced overlaps

To avoid the computation of repeat-induced overlaps, one strategy is to only consider seeds in the seed-and-extend computation whose k-mers are not contained inside a repeat. In this way we can ensure that any computed overlap has a significant unique part.

There are two strategies for this:

- *Screening known repeats:* Each read is aligned against a database of known repeats, i.e. using Repeatmasker. Portions of reads that match a known repeat are labeled *repetitive*.
- *De novo screening:* For each k-mer contained in H, the concatenation of reads, we determine how many times it occurs in H and then label those k-mers as *repetitive*, whose number of occurrences is unexpectedly high.

2.10 Celera's overlapper

The assembler developed at Celera Genomics employs an overlapper than compares up to 32 million pairs of reads per second.

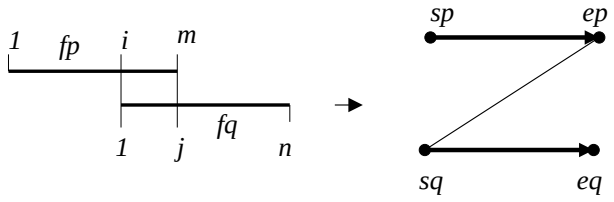
Overlapping all pairs of 27 million reads of human DNA using this program takes about 10 days, running on about 10-20 four processor machines (Compaq ES40), each with $4GB$ of main memory.

The input data file is about $50GB$. To parallelize the overlap compute, each job grabs as many reads as will fit into $4GB$ of memory (minus the memory necessary for doing the computation) and then streams all 27 million reads against the ones in memory.

2.11 The overlap graph

The overlap phase produces an *overlap graph* OG, defined as follows: Each read $f_p \in \mathcal{F}$ is represented by a directed edge (s_p, e_p) from node s_p to e_p, representing the start and end of f_p, respectively. The *length* of such a *read edge* is simply the length of the corresponding read.

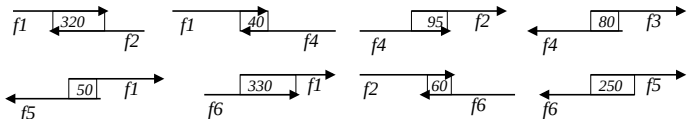
An overlap between $f_p = f_{p_1} f_{p_2} \dots f_{p_m}$ and $f_q = f_{q_1} f_{q_2} \dots f_{q_n}$ gives rise to an undirected *overlap edge* e between s_p, or e_p, and s_q, or e_q, depending on the orientation of the overlap, e.g.:



The label (or *"length"*) of the overlap edge e is defined to be -1 times the overlap length, e.g. $-(\frac{m-i+j-1}{2}+1)$ in the figure.

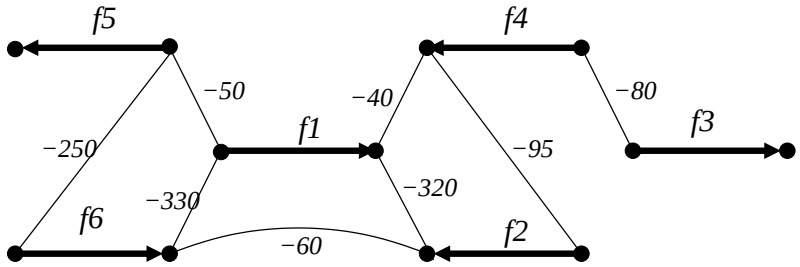
2.12 Example

Assume we are given 6 reads $\mathcal{F} = \{f_1, f_2, \ldots, f_6\}$, each of length 500, together with the following overlaps:



Here, for example, the last 320 bases of read f_1 align to the first 320 bases of the reverse complement $\overline{f_2}$ of f_2, whereas f_1 and $\overline{f_5}$ overlap in the first 50 bases of each.

We obtain the following overlap graph OG:

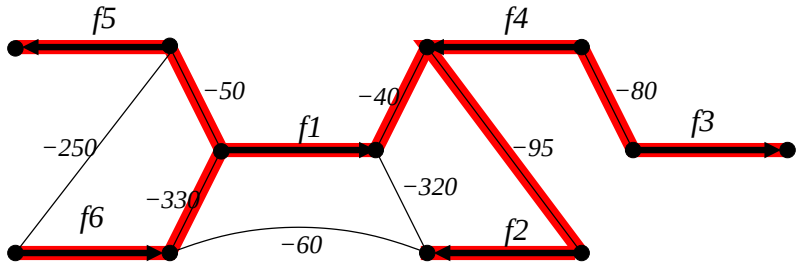


Each read f_p is represented by a read edge (s_p, e_p) of length $|f_p|$. Overlaps off the start s_p, or end e_p, of f_p are represented by overlap edges starting at the node s_p, or e_p, respectively. Each overlap edge is labeled by -1 times the overlap length.

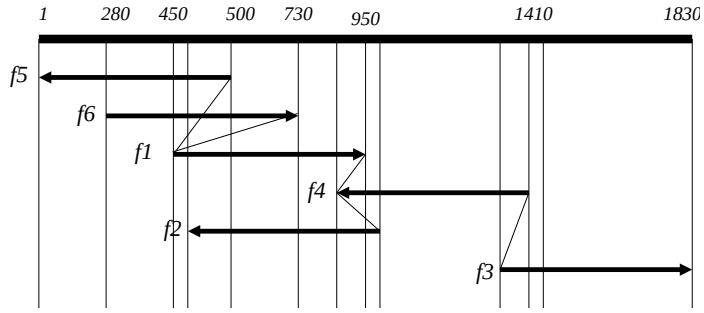
2.13 The layout phase

The goal of the layout phase is to arrange all reads into an approximate multi-alignment. This involves assigning coordinates to all nodes of the overlap graph OG, and thus, determining the value of s_i and e_i for each read f_i.

A simple heuristic is to select a *spanning forest* of the overlap graph OG that contains all read edges. (A spanning forest is a set F of edges such that any two nodes in the same connected component of OG are connected by a unique simple, unoriented path of edges in F.)

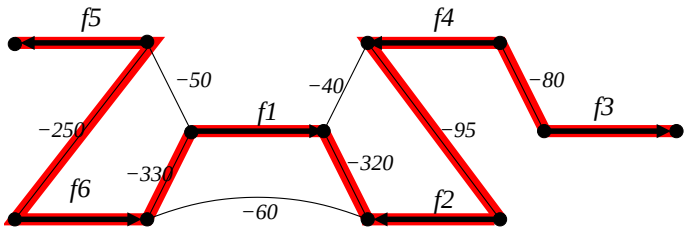


such a subset of edges positions every read with respect to every other, within a given connected component of the graph:



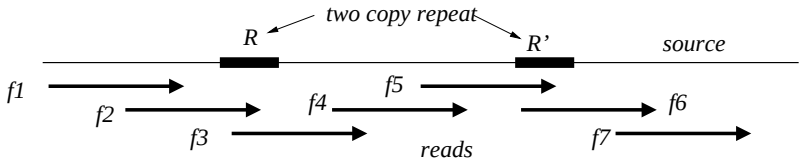
Such a putative alignment of reads is called a *contig*.

The spanning tree is usually constructed using a *greedy heuristic* in which the overlap edges are chosen in decreasing overlap length (i.e., increasing edge "length").

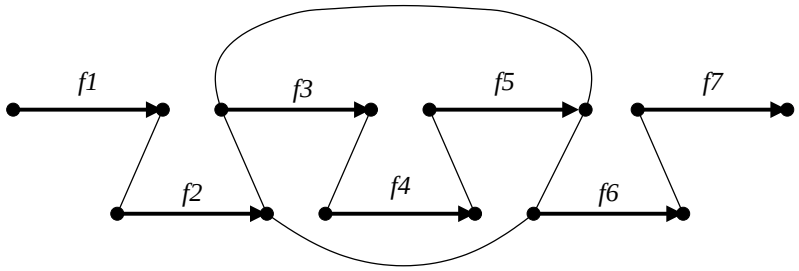


2.14 Repeats and the layout phase

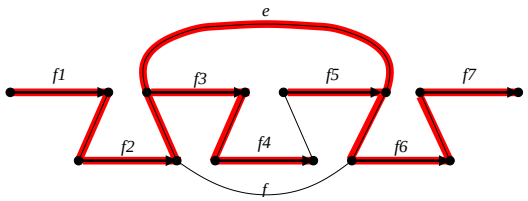
Consider the following situation:



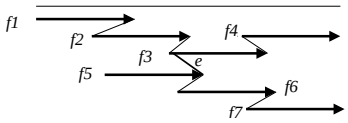
This gives rise to the following overlap graph:

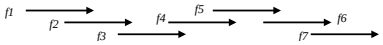


Consider this spanning tree:



A layout produced using the edge e or f does not reflect the true ordering of the reads and the obtained contig is called *misassembled*:



However, avoiding the repeat-induced edges e and f, one obtains a correct layout: 

Note that *neither* of the two layouts is "consistent" with all overlap edges in the graph.

2.15 Unitigging

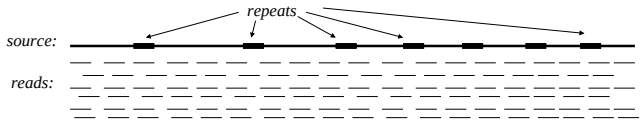
The main difficulty in the layout phase is that we can't distinguish between true overlaps and repeat-induced overlaps. The latter produce "inconsistent" layouts in which the coordinate assignment induces overlaps that are not reflected in the overlap graph (e.g., reads f_4 and f_7 in the example above).

Thus, the layout phase proceeds in two stages:

1. *Unitigging:* First, all uniquely assemblable contigs are produced, as just described. These are called *unitigs*.

2. *Repeat resolution:* Then, at a later stage, one attempts to reconstruct the repetitive sequence that lies between such unitigs.

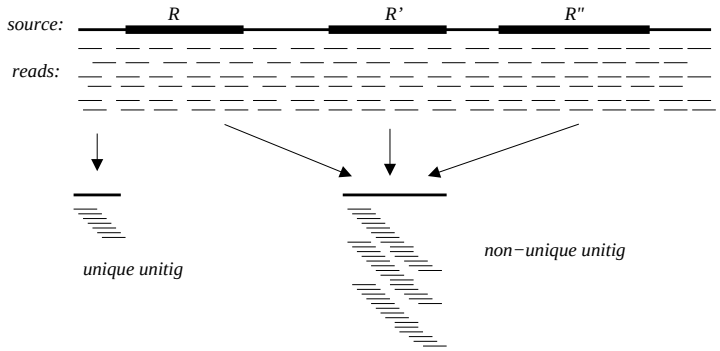
Reads are sampled from a source sequence that contains repeats:



Reads that form consistent chains in the overlap graph are assembled into unitigs and the remaining *"repetitive"* reads are processed later:

2.16 Unique unitigs

As defined above, a "unitig" is obtained as a chain of consistently overlapping reads. However, a unitig does not necessarily represent a segment of unique source sequence. For example, its reads may come from the interior of different instances of a long (many copy) repeat:



Non-unique unitigs can be detected by virtue of the fact that they contain significantly more reads than expected.

2.17 Identifying unique unitigs

Under assumption that the sampling of reads from the target is done uniformly, the *arrival* of the fragments start positions mapped along the target sequence should have constant, low probability. Hence we can modell this process using a Poison distribution.

Let R be the number of reads and G the estimated length of the source sequence. We then expect a on average $\frac{R}{G}$ arrivals of fragments per base.

For a unitig with k reads and approximate length ρ, the probability of seeing the $k-1$ start positions in the interval of length ρ is

$$\frac{e^{-c}c^k}{k!},$$

with $c := \frac{\rho R}{G}$, if the unitig is not oversampled, and

$$\frac{e^{-2c}(2c)^k}{k!},$$

if the unitig is the result of collapsing two repeats.
(see Mike Waterman's book, page 148, for details)
The *arrival statistic* used to identify *unique* unitigs is the (natural) log of the ratio of these two probabilities,

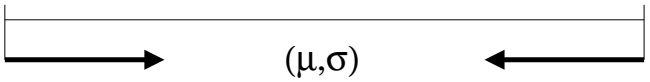
$$c - (\log 2)k.$$

A unitig is called *unique*, if it's arrival statistic has a positive value of 10 or more.

2.18 Mate pairs

Fragment assembly of reads produces contigs, whose relative placement and orientation with respect to each other is unknown.

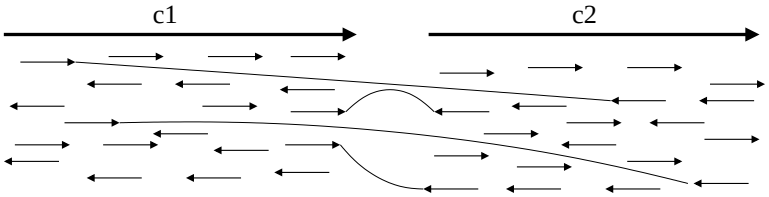
Recall that modern shotgun sequencing protocols employ a so-called *double barreled* shotgun. That is, longer clones of a given fixed length are sequenced from both ends and one obtains a pair of reads, a *mate pair*, whose relative orientation and mean μ (and standard deviation σ of) length are known:



Typical clone lengths are $\mu = 2kb$, $10kb$, $50kb$ or $150kb$. For clean data, $\sigma \approx 10\%$ of μ. Mate pair mismatching is a problem and can effect $10 - 30\%$ of all pairs.

2.19 Scaffolding

Consider two reconstructed contigs. If they correspond to neighboring regions in the source sequence, then we can expect to see mate pairs to span the gap between them:



Such mate pairs determine the relative orientation of both contigs, and we can compute a mean and standard deviation for the gap between them. In this case, the contigs are said to be *scaffolded*:

2.20 Determining the distance between two contigs

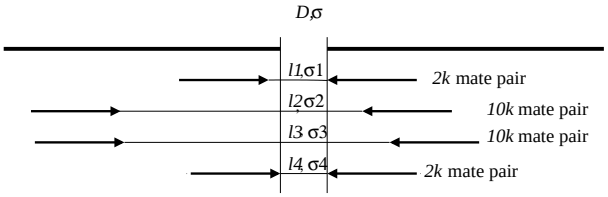
Given two contigs c_1 and c_2 connected by mate pairs $m_1, m_2, \ldots, m_k$. Each mate pair gives an estimation of the distance between the two contigs.

These estimations can viewed as independent measurements (l_1, σ_1), (l_2, σ_2), $\ldots (l_k, \sigma_k)$ of the distance D between the two contigs c_1 and c_2. Following standard statistical practice, they can be combined as follows:

Define $p := \sum \frac{l_i}{\sigma_i^2}$ and $q = \sum \frac{1}{\sigma_i^2}$. We set the distance between c_1 and c_2 to

$$D := \frac{p}{q}, \text{ with standard deviation } \sigma := \frac{1}{\sqrt{q}}.$$

Here is an example:



It is possible that the mate pairs between two contigs c_1 and c_2 lead to significantly different estimations of the distance from c_1 and c_2. In practice, only mate pairs that *confirm* each other, i.e. whose estimations are within 3σ of each other are considered together in a gap estimation.

2.21 The significance of mate pairs

Given two contigs c_1 and c_2. If there is only one mate pair between the two contigs, then due to the high error rates associated with mate pairs, this is not significant.

If, however, c_1 and c_2 are *unique unitigs*, and if there exist two different mate pairs between the two that give rise to the same relative orientation and similar estimations of the gap size between c_1 and c_2, then this the scaffolding of c_1 and c_2 is highly reliable.

This is because that probability that two false mate pairs occur that confirm each other, is extremely small.

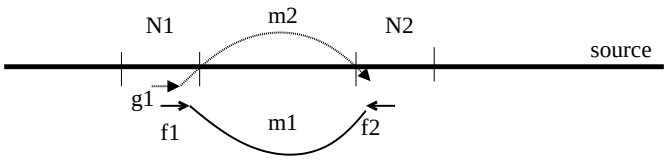
2.22 Example

Let the sequence length be $G = 120MB$, for example (Drosophila). For simplicity, assume we have 5-fold coverage of mate pairs, with a mean length of $\mu = 10kb$ and standard deviation of $\sigma = 1kb$.

Consider a false mate pair $m_1 = (f_1, f_2)$ with reads f_1 and f_2. Let N_1 and N_2 denote the two intervals (in the source sequence) of length 3σ centered at the starts of f_1 and f_2, respectively. Both have length $6kb$.

Consider a second false mate $m_2 = (g_1, g_2)$ with g_1 inside N_1. The probability that g_2 lies in N_2 is roughly

$$\frac{6kb}{120MB} = \frac{1}{20000}.$$



Assume that the reads have length 600. Assume that 10% of all mate pairs are false. At 5-fold coverage, the interval N_1 is covered by about $5 \cdot \frac{6000}{600} = 50$ reads, of which ≈ 5 have false mates.

Hence, the probability that m_1 is confirmed by some second false mate pair m_2 is

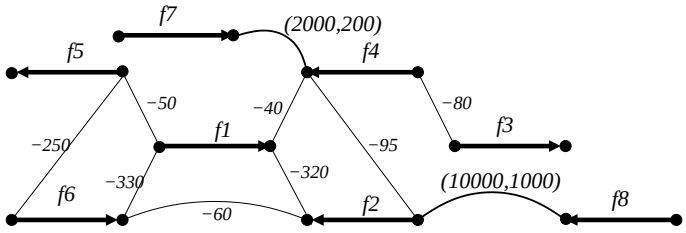
$$\approx 5 \cdot \frac{1}{20000} = \frac{1}{4000} = 0.00025.$$

2.23 The overlap-mate graph

Given a set of reads $\mathcal{F} = \{f_1, f_2, \ldots, f_R\}$ and let G denote the overlap graph associated with $\mathcal{F}$.

Given one set (or more) $M_{\mu,\sigma} = \{m_1, \ldots, m_k\}$ of mate pairs $m_k = (f_i, f_j)$, with mean μ and standard deviation σ.

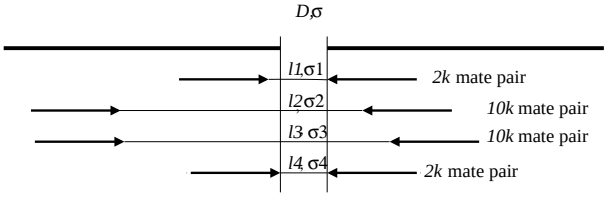
Let f_i and f_j be two mated reads represented by the edges (s_i, e_i) and (s_j, e_j) in G. We add an undirected *mate* edge between e_i and e_j, labeled (μ, σ), to indicate that f_i and f_j are mates and thus obtain the *overlap-mate graph*:



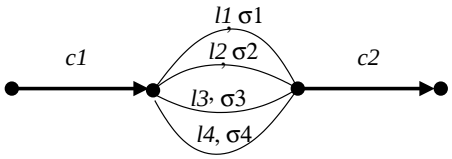
2.24 The contig-mate graph

Given a set of $\mathcal{F}$ of fragments and a set of assembled contigs $\mathcal{C} = \{c_1, c_2, \ldots, c_t\}$. A more useful graph is obtained as follows:

Represent each assembled contig c_i by a *contig edge* with nodes s_i and e_i. Then, add *mate edges* between such nodes to indicate that the corresponding contigs contain fragments that are mates:



Leads to:



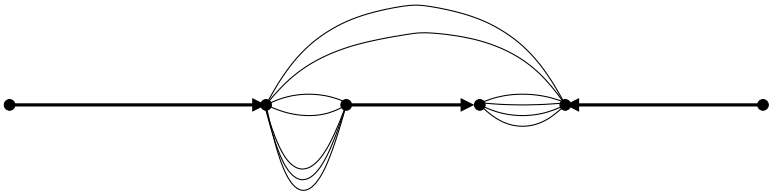
2.25 Edge bundling

Consider two contigs c_1 and c_2, joined by mate pair edges $m_1, \ldots, m_k$ between node e_1 and s_2. Every maximal subset of mutually confirming mate edges is replaced by a single *bundled mate edge* e, whose mean length μ and standard deviation σ are computed as discussed above. Any such bundled edge is labeled (μ, σ).

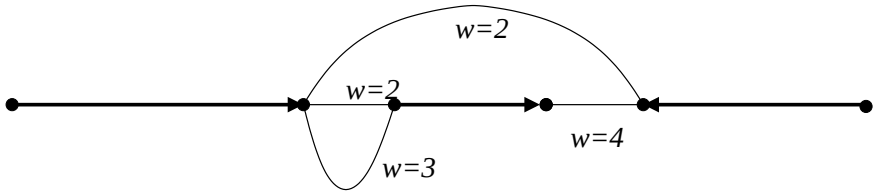
(A heuristic used to compute these subsets is to repeatedly bundle the median-length simple mate edge with all mate edges within three standard deviations of it, until all simple mate edges have been bundled.)

Additionally, we set the weight $w(e)$ of any mate edge to 1, if it is a simple mate edge, and to $\sum_{i=1}^{k} w(e_i)$, if it was obtained by bundling edges $e_1, \ldots, e_k$.

Consider the following graph:

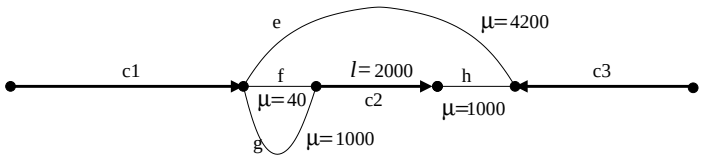


Assuming that mate edges drawn together have similar lengths and large enough standard deviation, edge bundling will produce the following graph:



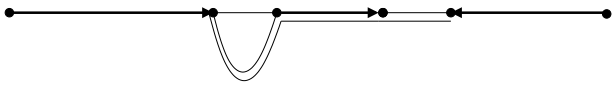
2.26 Transitive edge reduction

Consider the previous graph with some specific edge lengths:

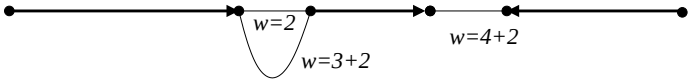


The mate edge e gives rise to estimation of the distance from the right node of contig c_1 to

the left node of c_3 that is similar to the one obtained by following the path P=(g, c_2, h). Based on this *transitivity* property we can *reduce* the edge e on to the path p:



to obtain:



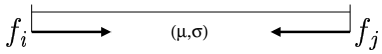
Consider two nodes v and w that are connected by an alternating path $P = (m_1, b_1, m_2, \ldots, m_k)$ of mate-edges $(m_1, m_2, \ldots)$ and contig edges $(c_1, c_2, \ldots)$ from v to w, beginning and ending with a mate-edge. We obtain a mean length and standard deviation for P by setting $l(P) := \sum_{m_i} \mu(m_i) + \sum_{c_i} l(c_i)$ and $\sigma(P) := \sqrt{\sum_{m_i} \sigma(m_i)^2}$.

We say that a mate-edge e from v to w can be *transitively reduced* on to the path P, if e and P approximately have the same length, i.e. if $|\mu(e) - l(P)| \leq C \cdot \max\{\sigma(e), \sigma(P)\}$ for some constant C, typically 3. If this is the case, then we can *reduce* e by removing e from the graph and incrementing the weight of every mate-edge m_i in P by $w(e)$.

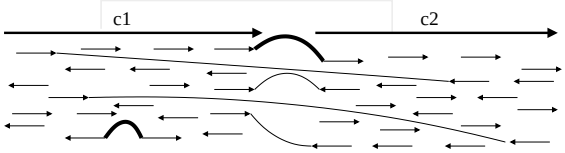
In the following, we will assume that any contig-mate graph considered has been edge-bundled and perhaps also transitively reduced to some degree.

2.27 Happy mate pairs

Consider a mate pair m of two reads f_i and f_j, obtained from a clone of mean length μ and standard deviation σ:



Assume that f_i and f_j are contained in the same contig or scaffold of an assembly. We call m *happy*, if f_i and f_j have the correct relative orientation (i.e., are facing each other) and are at approximately the right distance, i.e., $|\mu - |s_i - s_j|| \leq 3\sigma$. Otherwise, m is *unhappy*. Two unhappy mates are highlighted here:



2.28 Ordering and orientation of the contig-mate graph

Given a collection of contigs $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ constructed from a set of reads $\mathcal{F} = \{f_1, f_2, \ldots, f_R\}$, together with the corresponding mate pair information M. Let $G = (V, E)$

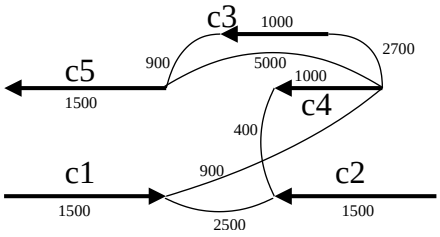
denote the associated contig-mate graph.

An *ordering (and orientation) of G (or $\mathcal{C}$)* is a map $\phi : V \to \mathbb{N}$ such that $|\phi(b_i) - \phi(e_i)| = l(c_i)$ for all contigs $c_i \in \mathcal{C}$, in other words, an assignment of coordinates to all nodes that preserves contig lengths.

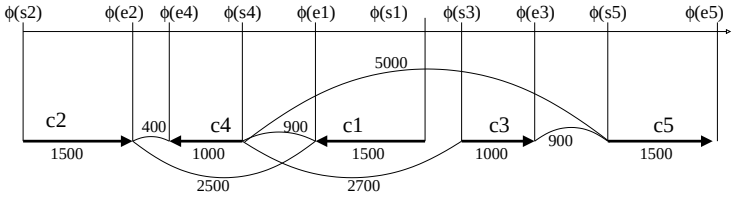
Additionally, we require $\{\phi(b_i), \phi(e_i)\} \neq \{\phi(b_j), \phi(e_j)\}$ for any two distinct contigs c_i and c_j.

2.29 Example

Given the following contig-mate graph:



An ordering ϕ assigns coordinates $\phi(v)$ to all nodes v and thus determines a layout of the contigs:



2.30 Happiness of mate edges

Let $G = (V, E)$ be a contig-mate graph and ϕ an ordering of G.

Consider a mate-edge e with nodes v and w. Let c_i denote the contig edge incident to v and let c_j denote the contig edge incident to w. Let v' and w' denote the other two nodes of c_i and c_j, respectively. We call e *happy* (with respect to ϕ), if c_i and c_j have the correct relative orientation, and if the distance between v and w is approximately correct, in other words, we require that either

1. $\phi(v') \leq \phi(v)$ & $|\phi(w) - \phi(v) - \mu(e)| \leq 3\sigma(e)$ & $\phi(w) \leq \phi(w')$, or
2. $\phi(w') \leq \phi(w)$ & $|\phi(v) - \phi(w) - \mu(e)| \leq 3\sigma(e)$ & $\phi(v) \leq \phi(v')$.

Otherwise, e is *unhappy*.

2.31 The Contig Ordering Problem

Given a collection of contigs $\mathcal{C} = \{c_1, c_2, \ldots, c_k\}$ constructed from a set of reads $\mathcal{F} = \{f_1, f_2, \ldots, f_R\}$, together with the corresponding mate pair information M. Let $G = (V, E)$ denote the associated contig-mate graph.

Problem The *Contig Ordering Problem* is to find an ordering of G that maximizes the sum of weights of happy mate edges.

Theorem The corresponding decision problem is NP-complete.

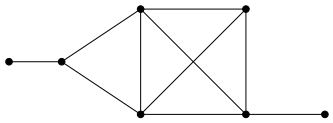
(The decision problem is: Given a contig-mate graph G, does there exist an ordering of G such that the total weight of all happy edges $\geq K$?)

2.32 Proof of NP-completeness

Recall: to prove that a problem X is NP-complete one must reduce a known NP-complete problem N to X. In other words, one must show that any instance I of N can be translated into an instance J of X in polynomial time such that I has the answer *true* iff J does.

We will use the following NP-complete problem:

BANDWIDTH: For a given graph $G = (V, E)$ with node set $V = \{v_1, v_2, \ldots, v_n\}$ and number K, does there exist a permutation ϕ of $\{1, 2, \ldots, n\}$ such that for all edges $\{v_i, v_j\} \in E$ we have $|\phi(i) - \phi(j)| \leq K$? (See Garey and Johnson 1979 for details.)

A graph with bandwidth 4: 

Problem is in NP: For a given ordering ϕ, we can determine whether the number of happy mate-edges exceeds the given threshold K in polynomial time by simple inspection of all mate edges.

Reduction of BANDWIDTH: Given an instance $G = (V, E)$ of this problem, we construct a contig graph $G' = (V', E')$ in polynomial time as follows:

First, set $V' := V$ and $E' := E$, and let these edges be the mate-edges, setting $\mu(e) := 1 + \frac{K-1}{2}$ and $\sigma(e) := \frac{K-1}{6}$ so as to obtain a happy range of $[1, K]$, and $w(e) := 1$, for every mate-edge e.

Then, for each initial node $v \in V$, add a new auxiliary node v' to V' and join v and v' by a contig edge of length 0.

The answer to the BANDWIDTH question is *true*, iff the graph G' has an ordering ϕ such that all mate edges in G' are happy:

$$\text{A graph } G \text{ has BANDWIDTH} \leq K$$
$$\Longleftrightarrow$$
$$\exists \text{ permutation } \phi \text{ such that } (v_i, v_j) \in E \text{ implies } |\phi(i) - \phi(j)| \leq K$$

$$\Longleftrightarrow$$

$$\exists \text{ ordering } \phi \text{ such that } (v_i, v_j) \in E \text{ implies } 1 \leq |\phi(i) - \phi(j)| \leq K$$

$$\Longleftrightarrow$$

$$\exists \text{ ordering } \phi \text{ such that } e = (v_i, v_j) \in E \text{ implies } \mu(e) - 3\sigma(e) \leq |\phi(i) - \phi(j)| \leq \mu(e) + 3\sigma(e)$$

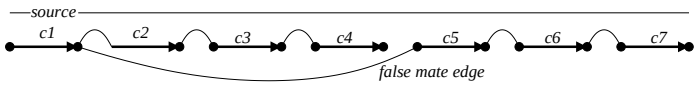
$$\Longleftrightarrow$$

all mate-edges of G' are happy. $\square$

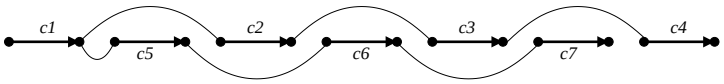
2.33 Spanning tree heuristic for the Contig Ordering Problem

An ordering ϕ that maximizes the number of happy mate edges is a useful scaffolding of the given contigs.

The simplest heuristic for obtaining an ordering is to compute a maximum weight spanning tree for the contig-mate graph and use it to order all contigs, similar to the read layout problem.



Unfortunately, this method does not work well in practice, as false mate edges lead to incorrect *interleaving* of contigs from completely different regions of the source sequence:



2.34 Representing an ordering in the mate-contig graph

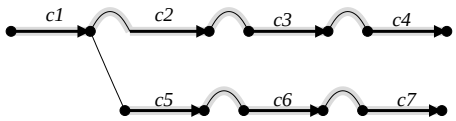
By the definition given above, an ordering is an assignment of coordinates to all nodes of the contig-mate graph that corresponds to a scaffolding of the contigs. When we are not interested in the exact coordinates, then the relative order and orientation of the contigs can be represented as follows:

Given a contig-mate graph $G = (V, E)$. A set $S \subseteq E$ of *selected* edges is called a *scaffolding* of G, if it has the following two properties:

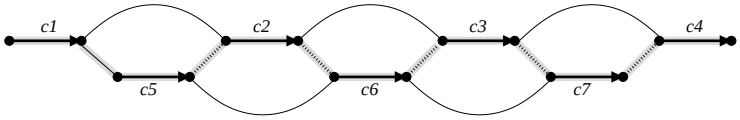
- every contig edge is selected, and
- every node is incident to at most two selected edges.

Thus, a scaffolding of G is a set of non-intersecting *selected paths*, each representing a scaffolding of its contained contigs.

The following example contains two chains of selected edges representing scaffolds $s_1 = (c_1, c_2, c_3, c_4)$ and $s_2 = (c_5, c_6, c_7)$:



However, to be able to represent the interleaved scaffolding discussed earlier, we need to add some *inferred* edges (shown here as dotted lines) to the graph:



2.35 Greedy path-merging

Given a contig-mate graph $G = (V, E)$. The greedy path merging algorithm is a heuristic for solving the Contig Ordering Problem. It proceeds "bottom up" as follows, maintaining a valid scaffolding $S \subseteq E$:

Initially, all contig edges $c_1, c_2, \ldots, c_k$ are selected, and none others. At this stage, the graph consists of k selected paths $P_1 = (c_1), \ldots, P_k = (c_k)$.

Then, in ordering of decreasing weight we consider each mate edge $e = \{v, w\}$: If v and w lie in the same selected path P_i, then e is a chord of P_i and no action is necessary.

If v and w are contained in two different paths P_i and P_j, then we attempt to merge the two paths to obtain a new path P_k and accept such a merge, only if the increase of $H(G)$, the number of happy mate edges, is larger than the increase of $U(G)$, the number of unhappy ones.

2.36 The greedy path-merging algorithm

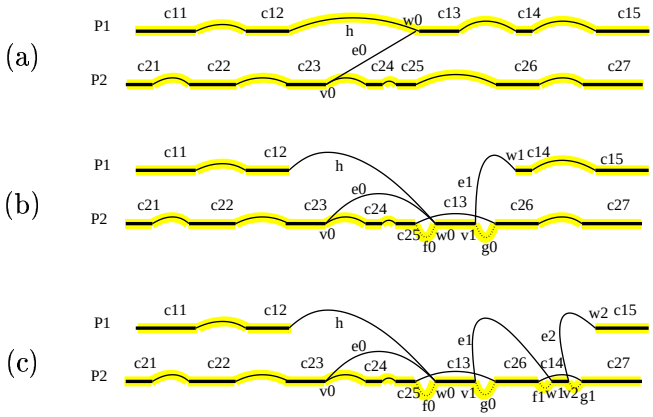
Algorithm Given a contig-mate graph G. The output of this algorithm is a node-disjoint collection of selected paths in G, each one defining an ordering of the contigs whose edges it covers.

begin
Select all contig edges.
for each mate-edge e in descending order of weight:
  if e is not selected:
    Let v, w denote the two nodes connected by e
    Let P_1 be the selected path incident to v
    Let P_2 be the selected path incident to w
    if $P_1 \neq P_2$ **and** we can *merge* P_1 and P_2 (guided by e)

to obtain P:
if $H(P) - (H(P_1) + H(P_2)) \geq U(P) - (U(P_1) + U(P_2))$:
Replace P_1 and P_2 by P
end

2.37 Merging two paths

Given two selected paths P_1 and P_2 and a *guiding* unselected mate-edge e_0 with nodes v_0 (incident to P_1) and w_0 (incident to P_2). Merging of P_1 and P_2 is attempted as follows:

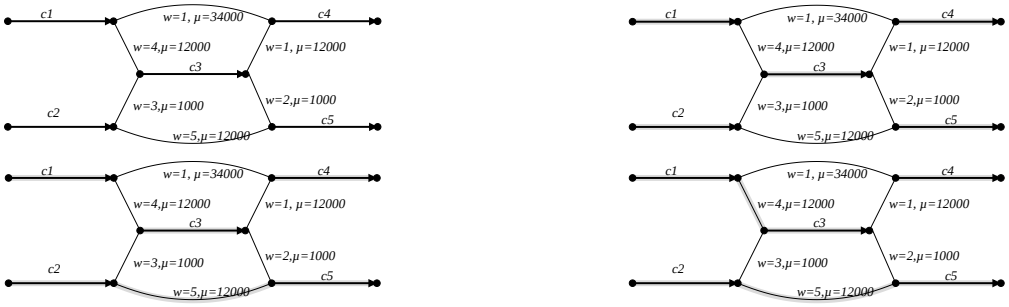


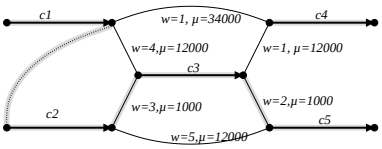
This algorithm returns *true*, if it successfully produced a new selected path P containing all contigs edges in P_1 and P_2, and *false*, if it fails.

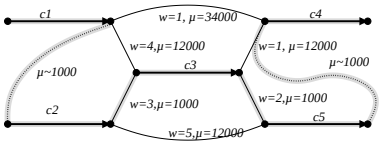
Merging proceeds by "zipping" the two paths P_1 and P_2 together, first starting with e_0 and "zipping" to the right. Then, with the edge labeled h now playing the role of e_0, zipper to the left. Merging is said to *fail*, if the positioning of the "active" contig c_i^1 implies that it must overlap with some contig in P_2 by a significant amount, but no such alignment (of sufficiently high quality) exists.

2.38 Example

Here are we are given 5 contigs $c_1, \ldots, c_5$, each of length $l(c_i) = 10000$:





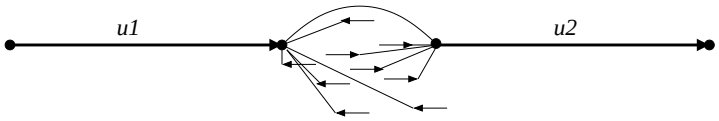


The final scaffolding is $(c_1, c_2, c_3, c_5, c_4)$.

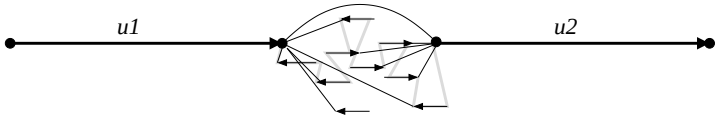
2.39 Repeat resolution

Consider two unique unitigs u_1 and u_2 that are placed next to each other in a scaffolding, due to a heavy mate edge between them:

We consider all non-unique unitigs and singleton reads that potentially can be placed between u_1 and u_2 by mate edges:



Different heuristics are used to explore the corresponding local region of the overlap graph in an attempt to find a chain of overlapping fragments that spans the gap and is compatible with the given mate pair information:



2.40 Multialignment

In a last step we have to compute a consensus sequence for each contig based on the layout of the fragments (this can also be doen right after computing the contigs/unitigs).

```
R1           ACGCTCCAACCGCTAATACG
R2                    ATCGCTAATCCACGCCCGCCCCGC
R3         AAAC-CTCCAACCG
R4                              TGCGCGCCCGCCCCGAAACCGC
Consensus AAAC-CTCCAACCGCTAATGCGCGCCCGCCCCGAAACCGC
```

2.41 Summary

Given a collection $\mathcal{F} = \{f_1, f_2, \ldots, f_R\}$ of reads and mate pair information, sampled from a unknown source DNA sequence. Assembly proceeds in the following steps:

1. compute the overlap graph, e.g. using a seed-and-extend approach,
2. construct all unitigs, e.g. using the minimal spanning tree approach,
3. scaffold the unitigs, e.g. using the greedy-path merging algorithm,
4. attempt to resolve repeats between unitigs, and
5. compute a multi alignment of all reads in a given contig to obtain a consensus sequence for it.

Note that the algorithms for steps (2) and (3) that are used in actual assembly projects are much more sophisticated than ones described in these notes.

2.42 A WGS assembly of human (Celera)

Input: 27 million fragments of av. length $550bp$, 70% paired:

$5m$	pairs of length $2kb$
$4m$	pairs of length $10kb$
$0.9m$	pairs of length $50kb$
$0.35m$	pairs of length $150kb$

Celera's assembler uses approximately the following resources:

Program	CPU hours		Max. memory
Screener	4800	2-3 days on 10-20 computers	$2GB$
Overlapper	12000	10 days on 10-20 computers	$4GB$
Unitigger	120	4-5 days on a single computer	$32GB$
Scaffolder	120	4-5 days on a single computer	$32GB$
RepeatRez	50	Two days on a single computer	$32GB$
Consensus	160	One day on 10-20 computers	$2GB$

Total: $\approx$ 18000 CPU hours.

The size of the human genome is $\approx 3Gb$. An unpublished 2001 assembly of the $27m$ fragments has the following statistics:

- The assembly consists of 6500 scaffolds that span $2776Mb$ of sequence.
- The spanned sequence contains 150000 gaps, making up $148Mb$ in total.
- Of the spanned sequence, 99.0% is contained in scaffolds (or contigs?) of size $30kb$ or more.
- Of the spanned sequence, 98.7% is contained in scaffolds (or contigs?) of size $100kb$ or more.

3 Sequencing by Hybridization (SBH) (Daniel Huson)

1. Pavel Penzer, Computational Molecular biology - an algorithmic approach, MIT Press, 2000, chapter 5.

In the early days of DNA arrays, the inventors of the method suggested them indeed for sequencing. We will shortly review DNA arrays and then talk about the combinatorial problems arising.

3.1 DNA arrays

- Also known as: biochips, DNA chips, oligo arrays, DNA microarrays or gene arrays.
- An array is an orderly arrangement of (spots of) samples.
- Samples are either DNA or DNA products.
- Each spot in the array contains many copies of the sample.
- Array provides a medium for matching known and unknown DNA samples based on base-pairing (*hybridization*) rules and for automating the process of identifying the unknowns.
- Sample spot size in microarray less than 200 microns and an array contains thousands of spots.
- Microarrays require specialized robotics and imaging equipment.
- High-throughput biology: a single DNA chip can provide information on thousands of genes simultaneously.

3.2 General principle

We are given an unknown *target* nucleic acid sample and the goal is to detect the identity and/or abundance of its constituents using known *probe* sequences. Single stranded DNA probes are called *oligo-nucleotides* or *oligos*.

Modern chips work as follows: An array of probes is produced either *in situ* or by attachment. The array is then exposed to sample DNA. Examples are *oligo-arrays* and *cDNA* microarrays.

The free sequence is fluorescently or radioactively labeled and hybridization is used to determine the identity/abundance of complementary sequences.

3.3 Oligo arrays $C(l)$

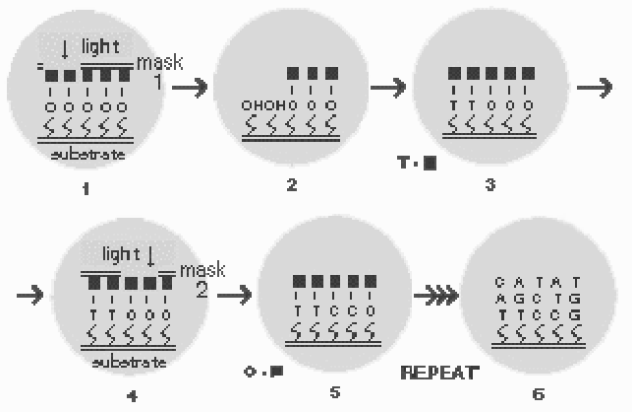
The simplest oligo array $C(l)$ consists of all possible oligos of length l and is used e.g. in *sequencing by hybridization* (SBH).

$C(4)$	AA	AT	AG	AC	TA	TT	TG	TC	GA	GT	GG	GC	CA	CT	CG	CC
AA																
AT																
AG																
AC																
TA																
TT																
TG																
TC									□							
GA																
GT																
GG																
GC																
CA																
CT																
CG																
CC																

Example: oligo at □: $TCGA$

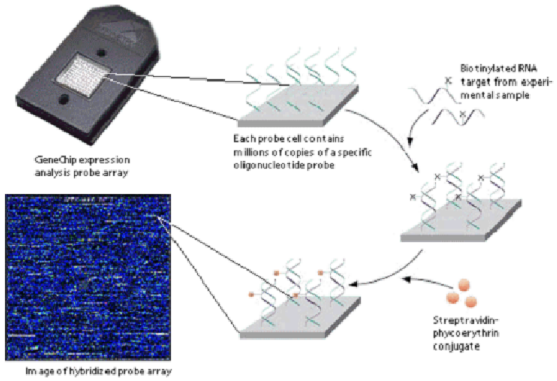
3.4 Manufacturing oligo arrays

1. Start with a matrix created over a glass substrate.
2. Each cell contains a growing "chain" of nucleotides that ends with a *terminator* that prevents chain extension.
3. Cover the substrate with a mask and then illuminate the uncovered cells, breaking the bonds between the chains and their terminators.
4. Expose the substrate to a solution of many copies a specific nucleotide base so that each of the unterminated chains is extended by one copy of the nucleotide base and a new terminator.
5. Repeat using different masks.



Exposure to light replaces the terminators by hydrogen bonds (1–2), and (3) bonds forms with nucleotide bases provided in a solution, and then the process is repeated with a different base (4–6).

3.5 Experiment with a DNA chip



Labeled RNA molecules are applied to the probes on the chip, creating a fluorescent spot where hybridization has occurred.

3.6 Sequencing by Hybridization (SBH)

Originally, the hope was that one can use DNA chips to sequence large unknown DNA fragments using a large array of short probes:

1. Produce a chip $C(l)$ spotted with all possible probes of length l ($l = 8$ in the first SBH papers),
2. Apply a solution containing many copies of a fluorescently labeled DNA target fragment to the array.

3. The DNA fragments hybridize to those probes that are complementary to substrings of length l of the fragment

4. Detect probes that hybridize with the DNA fragment and obtain the l-tuple composition of the DNA fragment

5. Apply a combinatorial algorithm to reconstruct the sequence of the DNA target from the l-tuple composition

3.7 The Shortest Superstring Problem

SBH provides information of the l-tuples present in a target DNA sequence, but not their positions. Suppose we are given the *spectrum* S of all l-tuples of a target DNA sequence, how do we construct the sequence?

This is a special case of the *Shortest Common Superstring Problem (SCS)*: A *superstring* for a given set of strings $s_1, s_2, \ldots, s_m$ is a string that contains each s_i as a substring. Given a set of strings, finding the shortest superstring is NP-complete.

Define $overlap(s_i, s_j)$ as the length of a maximal prefix of s_j that matches a suffix of s_i. The SCS problem can be cast as a Traveling Salesman Problem in a complete directed graph G with m vertices $s_1, s_2, \ldots, s_m$ and edges (s_i, s_j) of length $-overlap(s_i, s_j)$.

3.8 The SBH graph

SBH corresponds to the special case when all substrings have the same length l. We say that two SBH probes p and q *overlap*, if the last $l-1$ letters of p coincide with the first $l-1$ of q.

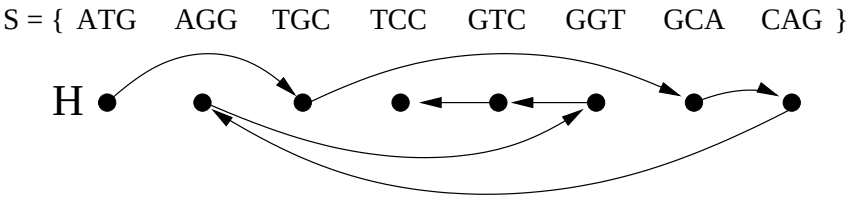
Given the spectrum S of a DNA fragment, construct the directed graph H with vertex set S and edge set

$$E = \{(p, q) \mid p \text{ and } q \text{ overlap}\}.$$

There exists a one-to-one correspondence between paths that visit each vertex of H at least once and the DNA fragments with the spectrum S.

3.9 Example of the SBH graph

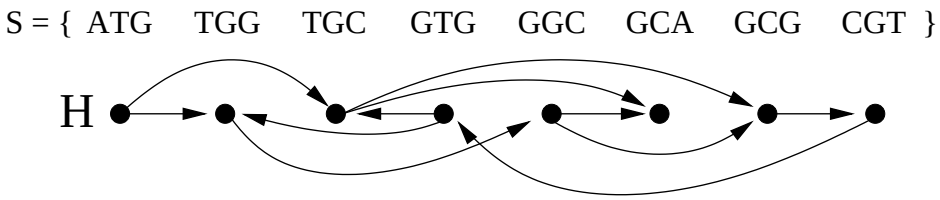
Vertices: l tuples of the spectrum S, edges: overlapping l-tuples:



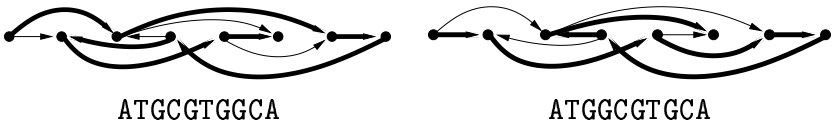
The path visiting all vertices corresponds to the sequence reconstruction `ATGCAGGTCC`.

A path that visits all nodes of a graph exactly once is called a *Hamiltonian* path. Unfortunately, the Hamiltonian Path Problem is NP-complete, so for larger graphs we cannot hope to find such paths.

3.10 Second example of the SBH graph

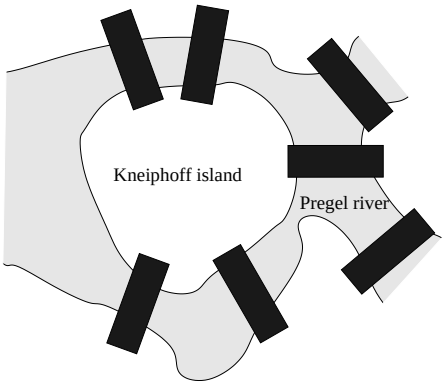


This example has two different Hamiltonian paths and thus two different reconstructed sequences:



3.11 Euler Path

Leonard Euler wanted to know whether there exists a path that uses all seven bridges in Königsberg exactly once:



Birth of graph theory...

3.12 SBH and the Eulerian Path Problem

Let S be the spectrum of a DNA fragment. We define a graph G whose set of nodes consists of *all possible* $(l-1)$*-tuples*.

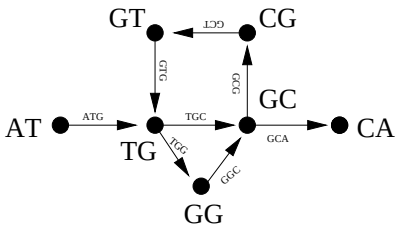
We connect one $l-1$-tuple $v = v_1 \dots v_{l-1}$ to another $w = w_1 \dots w_{l-1}$ by a directed edge (v, w), if the spectrum S contains an l-tuple u with prefix v and suffix w, i.e. such that $u = v_1 \dots v_{l-1} w_1 = v_{l-1} w_1 \dots w_{l-1}$.

Hence, in this graph the probes correspond to edges and the problem is to find a path that visits all *edges* exactly once, i.e., an *Eulerian path*.

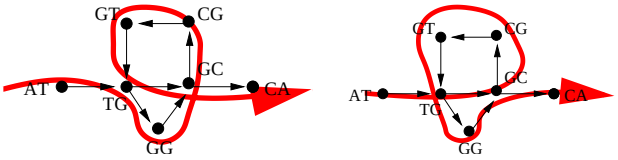
Finding all Eulerian paths is simple to solve.

(To be precise, the Chinese Postman Problem (visit all edges at least once in a minimal tour) can be efficiently solved for directed or undirected graphs, but not in a graph that contains both directed and undirected edges.)

S = { ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT }



Vertices represent $(l-1)$-tuples, edges correspond to l-tuples of the spectrum. There are two different solutions:



3.13 Eulerian graphs

A directed graph G is called *Eulerian*, if it contains a cycle that traverses every edge of G exactly once.

A vertex v is called *balanced*, if the number of edges entering v equals the number of edges leaving v, i.e. $indegree(v) = outdegree(v)$. We call v *semi-balanced*, if $|indegree(v) - outdegree(v)| = 1$.

Theorem A directed graph is Eulerian, iff it is connected and each of its vertices is balanced.

Lemma A connected graph has an Eulerian path, iff it contains at most two semi-balanced nodes.

3.14 When does a unique solution exist?

Problem: Given a spectrum (l-tuple composition) S. Does it possess a unique sequence reconstruction?

Consider the corresponding graph G. If the graph G is Eulerian, then we can decompose it into *simple* cycles $C_1, \ldots, C_t$, that is, cycles without self-intersections. Each edge of G is used in exactly one cycle, although nodes can be used in many cycles. Define the *intersection graph* G_I on t vertices $C_1, \ldots, C_t$, where C_i and C_j are connected by k edges, iff they have precisely k original vertices in common.

Lemma Assume G is Eulerian. Then G has only one Eulerian cycle iff the intersection graph G_I is a tree.

3.15 Probability of unique sequence reconstruction

What is the probability that a randomly generated DNA fragment of n can be uniquely reconstructed using a DNA array $C(l)$? In other words, how large must l be so that a random sequence of length n can be uniquely reconstructed from its l-tuples?

We assume that the bases at each position are chosen independently, each with probability $p = \frac{1}{4}$.

Note that a repeat of length $\geq l$ will always lead to a non-unique reconstruction. We expect about $\binom{n}{2}p^l$ repeats of length $\geq l$. Note that $\binom{n}{2}p^l = 1$ implies $l = \log_{\frac{1}{p}} \binom{n}{2}$.

$\Longrightarrow$ For a given l one should choose $n \approx \sqrt{2 \cdot 4^l}$, but not larger. (However, this is a very loose bound and a much tighter bound is known.)

3.16 Summary

- SBH can be formulated as a Hamiltonian path problem, which is NP-hard.
- SBH can bes formulated as Eulerian path problem, which can be solved in linear time.
- The Eulerian path approach to SBH is currently infeasible due to two problems:
  - Errors in the data
    - False positives arise, when the the target DNA hybridizes to a probe even though an exact match is not present
    - False negatives arise, when an exact match goes undetected
  - Repeats make the reconstruction impossible, as soon as the length of the repeated sequence is longer than the word length l

- Nevertheless, ideas developed here are employed in a new approach to sequence assembly that uses sequenced reads and a Eulerian path representation of the data (Pavel Pevzner, Recomb'2001).

4 Multiple Sequence alignment (Knut Reinert)

This exposition is based on the following sources, which are all recommended reading:

1. Mount, Bioinformatics, CSHL Press, 2001, chapter 4.
2. Setubal und Meidanis, Introduction to Computational Molecular Biology, PWS Publishing, 1997, chapter 5.
3. Gusfield, Algorithms on Strings, Trees, and Sequences, Cambridge University Press, 1997, chapter 14.
4. Reinert, Stoye, Will, An iterative method for faster sum-of-pairs multiple sequence alignment, Bioinformatics, 2000, Vol 16, no 9, pages 808-814.

Remark: This topic will be treated in depth in the lecture in the spring semester and will have several master thesis topics.

4.1 Introduction and definition

Multiple sequence alignment is a generalization of pairwise sequence alignment. While this seems trivial, multiple alignments are much more than a technical exercise:

They are used for

- detecting faint similarities in sequences that are not detected by pairwise sequence comparison
- for grouping proteins into families
- for computing the consensus in assembly projects
- and many more ...

Since there are so many different versions of this problem, it is important to use the appropriate implementation. In this section we

- define multiple sequence alignment
- look at scoring schemes
- survey different algorithmic techniques
- describe an optimal approach

- describe a progressive method

Definition 2. *Let Σ be a finite alphabet without the letter '$-$' and $\Sigma' = \Sigma \cup \{'-'\}$. Further, let $S_1, \ldots, S_k$ be k sequences over Σ with lengths $n_1, \ldots, n_k$. A* (global) multiple alignment *A of $S_1, \ldots, S_k$ is a matrix of dimension $k \times l$ with the following properties:*

- $\max\{n_1, \ldots, n_k\} \leq l \leq \sum_{i=1}^{k} n_i$.
- $A[i][j] \in \Sigma' \quad \forall\ 1 \leq i \leq k, 1 \leq j \leq l$.
- *The i-th row without blanks equals S_i.*
- *There is no column consisting entirely of blanks.*

Example 3.

```
S1 = - G C T G A T A T A G C T
S2 = G G G T G A T - T A G C T
S3 = - G C T - A T - - C G C -
S4 = A G C G G A - A C A C C T
```

4.2 Cost functions

Generally one can define similarity measures or distance measures as it is the case for pairwise alignment. In what follows we will use distance measures and try to minimize the cost if not noted otherwise.

In what follows we need the definition of multiple alignments of subsets of the k strings.

Definition 4. *Let A be a multiple alignment for the k strings $S_1, \ldots, S_k$ and $I \subseteq \{1, \ldots, k\}$ be a set of indices defining a subset of the k strings. Then we define A_I as the alignment resulting from first removing all rows $i \notin I$ from A and then deleting all columns consisting entirely of blanks.We call A_I the* projection *of A to I. If I is given explicitely, we simplify notation and write $A_{i,j,k}$ instead of $A_{\{i,j,k\}}$.*

Example 5.

```
S1 = - G C T G A T A T A G C T
S2 = G G G T G A T - T A G C T
S3 = - G C T - A T - - C G C -
S4 = A G C G G A - A C A C C T
```

The projection $A_{2,3}$ is given by first taking the second and third row of the alignment:

```
S2 = G G G T G A T - T A G C T
S3 = - G C T - A T - - C G C -
```

and then deleting the column consisting only of blanks.

```
S2 = G G G T G A T T A G C T
S3 = - G C T - A T - C G C -
```

What is the quality, or the score of a multiple alignment? Let $c : \mathcal{A} \to \mathbb{R}$ be a function the maps each possible alignment in the set of all alignments $\mathcal{A}$ to a real number. If we index an optimal alignment with a $*$, then our goal is to find an optimal multiple alignment A^* with $c(A^*) = \min_{A \in \mathcal{A}} c(A)$

In the following we describe a few common cost functions, namely

- weighted sum of pairs (wsop) score (with linear gap costs)
- phylogenetic score
- consensus score
- profile score

4.3 WSOP

One of the most common cost function is the *(weighted) sum of pairs* which is the (weighted) sum of the costs of all pairwise projections, namely

$$c(A) = \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} w_{i,j} \cdot c(A_{i,j})$$

or, setting $d(-,-) = 0$:

$$c(A) = \sum_{h=1}^{l} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} w_{i,j} \cdot d(A[h,i], A[h,j])$$

Example 6. *Let* $d(A,B) = 2$ *for* $A \neq B$, $d(A,-) = d(-,A) = 1$ *for* $A \neq' -'$, 0 *else. All the weight factors are* 1.

```
            S1 = - G C T G A T A T A A C T
            S2 = G G G T G A T - T A G C T
            S3 = A G C G G A - A C A C C T
                 ------------------------
cost of column:  4 0 4 4 0 0 2 2 4 0 6 0 0 = 26
```

The sum of pair score has the advantage to take all pairwise information into account, however it is easily biased by overrepresentation of sequences from the same family. This can be dealt with by weighting overrepresented sequences accordingly (see later in the description of ClustalW).

4.4 Phylogenetic score

Sometimes a *phylogenetic* tree T is given, implying the ancestral relationships of the sequences. Then $c(A)$ can be defined as follows:

$$c(A) = \sum_{(i,j) \in T} c(A_{i,j})$$

Example 7. *Let* $d(A, B) = 2$ *for* $A \neq B$, $d(A, -) = d(-, A) = 1$ *for* $A \neq' -'$, 0 *else.*

```
                 S1 = - G C T G A T A T A A C T
 T= S1--S2--S3   S2 = G G G T G A T - T A G C T
                 S3 = A G C G G A - A C A C C T
                      ------------------------
cost of column:       3 0 4 2 0 0 1 2 2 0 4 0 0 = 18
```

4.5 Consensus Score

Let $A[][i]$ be any column of a multiple alignment A. Then the letter x_i is called the i-th *consensus*-letter, if the *consensus*-error ($\sum_{j=1}^{k} d(x_i, A[j][i])$) is minimal. The concatenation of the consensus letters yields the *consensus*-string. Hence the goal is to find the alignment A^*, which minimizes the consensus error summed over all columns. $c(A)$ is defined as follows:

$$c(A) = \sum_{i=1}^{l} \sum_{j=1}^{k} d(x_i, A[j][i]) \quad x_i \text{ is the } i\text{-th consensus letter}$$

4.6 Consensus Score

Example 8. *Let* $d(A, B) = 2$ *for* $A \neq B$, $d(A, -) = d(-, A) = 1$ *for* $A \neq' -'$, 0 *else.*

```
              S1 = - G C T G A T A T A A C T
              S2 = G G G T G A T - T A G C T
              S3 = A G C G G A - A C A C C T
                   ------------------------
consensus   :      - G C T G A T A T A X C T
column value:      2 0 2 2 0 0 1 1 2 0 3 0 0 = 13
```

A X in the consensus string symbolizes that the consensus can be any letter.

4.7 Profile Alignment

Let A be a multiple alignment. Then the *profile* of A gives the relative frequency of each letter in each column. Aligning a string S to a profile accounts to computing the weighted sum of the letters S to the columns of the profile.

Example 9. *(similarity alignment) Let* $s(A,B) = s(X,-) = s(-,X) = -1$, $s(A,C) = -3$, $s(B,C) = -2$ *and* $s(A,A) = s(B,B) = s(C,C) = 2$.

```
S1 = A B C - A     Profile: C1   C2   C3   C4   C5
S2 = A B A B A           A: .75       .25       .50
S3 = A C C B -           B:      .75  .50  .75
S4 = C B - B C           C: .25  .25            .25
                         -:           .25  .25  .25
-------------------------------------------------

      column  column value
   A    1   =  0.75*2 - 0.25*3          =  0.75
   A        = -1.0 *1                  = -1.0
   B    2   =  0.75*2 - 0.25*2          =  1.0
   -    3   = -0.25*1 - 0.50*1 - 0.25*1 = -1.0
   B    4   =  0.75*2 - 0.25*1          =  1.25
   C    5   =  0.25*2 - 0.5 *3 - 0.25*1 = -1.25
                                        -------------
                                           -0.25
```

4.8 Methods for MSA

Most optimization problems using the above cost functions are NP-hard. Hence exact solutions cannot be expected for more than 7-15 sequences and one has to resort to heuristic approaches. Most multiple alignment methods can be classified as follows:

Iterative algorithms (Realigner,PRRP)
Progressive algorithms (ClustalW)
Motif searching algorithms (Dialign)
Stochastic algorithms (HMM, Gibbs-Sampling)
Divide-and-Conquer algorithms (DCA,OMA)
Exact algorithms (MSA,Combinatorial Sequence Alignment)

We will now in detail describe an advanced exact method for optimizing the WSOP cost function with linear gap costs, and then the most commonly used iterative algorithm implemented in the program `ClustalW`.

4.9 An exact method

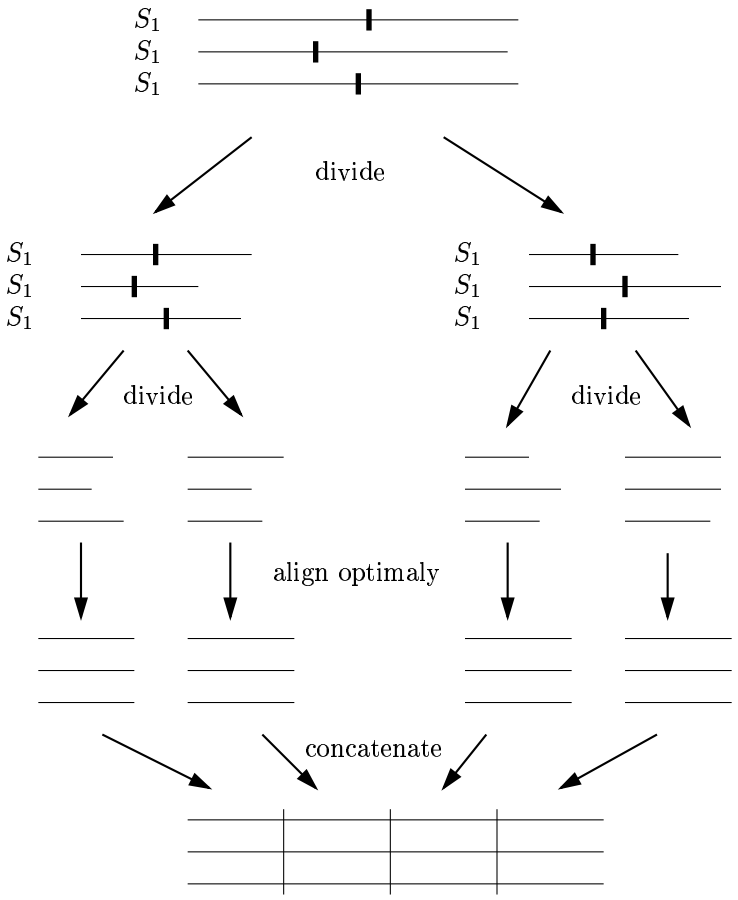
In the following description we will combine three basic algorithmic techniques:

1. divide-and-conquer
2. dynamic programming
3. branch-and-bound

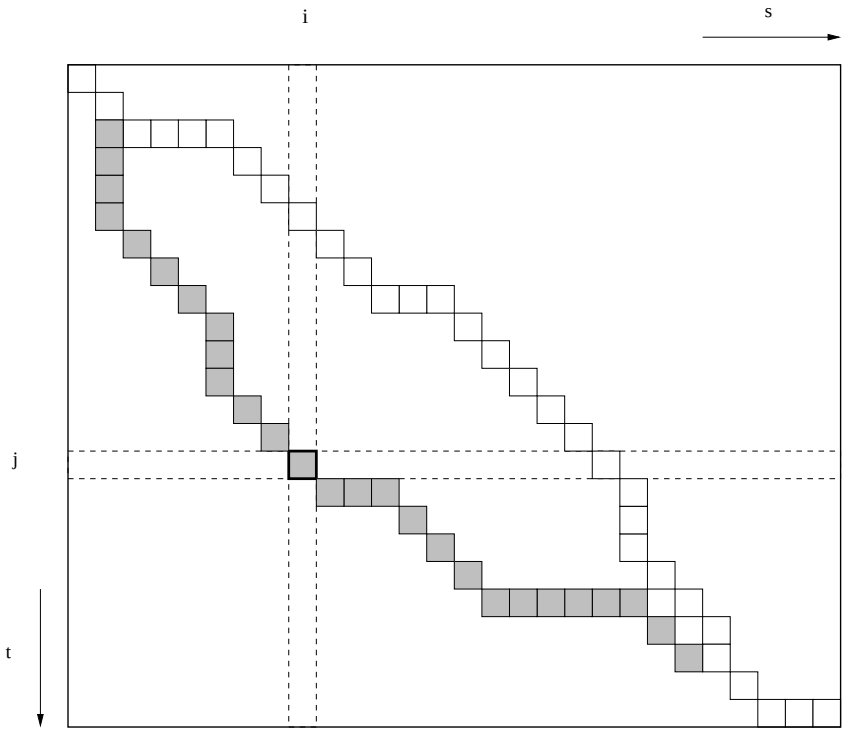
4.10 Divide-and-conquer

The idea is straightforward: cut the k sequences and solve the resulting subproblems recursively. The solutions can be combined trivially. If the problem size is small enough an exact algorithm can be employed.

On the one hand it is clear that optimal cut positions exist, on the other hand it is clear that it is NP-hard to find them. Trivial ideas for determining the cut positions are not very succesful. Stoye showed how to compute relatively good cut positions.



The idea is to compute so called *additional cost* matrices for each pair of sequences and each possible cut position. The additional costs are best illustrated as the "length" difference between the optimal global alignment and the best alignment going through position i and j.



Then Stoye determines *C-optimal* families of slicing positions, such that the (weighted) sum of pairwise additional costs is minimized. (more on that in the spring lecture).

4.11 Dynamic programming

The straightforward extension to compute an optimale (W)SOP-cost alignment A^* using dynamic programming takes time $O(k^2 2^k N)$ with $N = \prod_i n_i$. Obviously this is only practical for very few, short sequences ($k = 3, 4$).

In the following we switch to the edit graph representation of a (multiple) alignment. In this representation finding an optimal alignment corresponds to finding a shortest path in a directed acyclic graph (DAG).

All considerations can also be done with affine gap costs, but for the sake of exposition we use linear gap costs.

The graph has nodes

$$V = \{v = (v[1], v[2], \ldots, v[k]) : v \in \mathbb{N}^k$$
$$\text{and } 0 \leq v[i] \leq n_i, \quad 1 \leq i \leq k\}$$

and edges

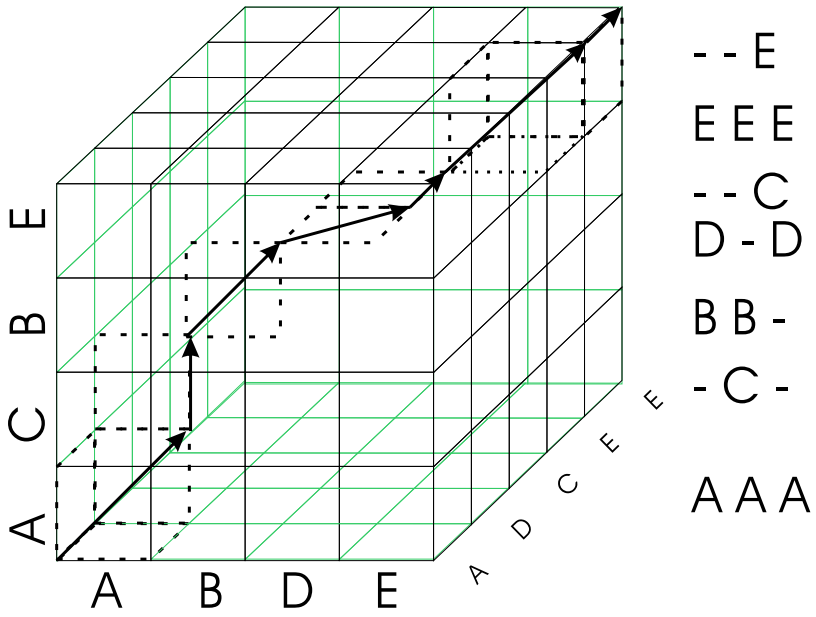
$$E = \{(p, q) : p, q \in V,\ p \neq q \text{ und } q - p \in \{0, 1\}^k \setminus \{0\}^k\}$$

We denote with $p \to q$ the set of all paths from a node p to a node q

$$p \to q := \left\{(p{=}v_0,\ v_1,\ \ldots\ ,\ v_n{=}q) : (v_i, v_{i+1}) \in E,\ 0 \leq i < n\right\}$$

Analogously we write $p \to q \to r$ for the set of all paths from p to r, which run through node q.

The graph contains two special nodes the *source* $s = (0, 0, \ldots, 0)$, and the sink $t = (n_1, n_2, \ldots, n_k)$.



A path π of length l from $s = (0, \ldots, 0)$ to $t = (n_1, \ldots, n_k)$ corresponds to an alignment defined through the following matrix:

$$A_{ij} = \begin{cases} - & \text{if } v_j[i] - v_{j-1}[i] = 0 \\ S_i[v_j[i]] & \text{if } v_j[i] - v_{j-1}[i] = 1 \end{cases} \qquad \text{for } 1 \leq i \leq k,\ 1 \leq j \leq l$$

The cost of an alignment is the sum of the cost of all edges:

$$c(\pi) := \sum_{i=0}^{l-1} c(v_i, v_{i+1})$$

with

$$c(v, w) := \sum_{1 \leq i < j \leq k} d(a_{i*}, a_{j*})$$

We overload the notation of c, since an edge in the grid graph corresponds exactly to a column in a multiple alignment. We denote the shortest path in $p \to q$ with $p \to^* q$ and its length with $c(p \to^* q)$.

A node r in the grid cuts each sequence S_i into a prefix α_i^r and a suffix σ_i^r.

4.12 Dynamic programming and branch-and-bound

Computing the shortest path in the DAG(directed acyclic graph) can be done by traversing the DAG in any topolocial order, which in turn corresponds exactly to the natural extension of dynamic programming.

However, this would imply *constructing* the whole graph which is a major obstacle. Hence we resort to an in theory more costly algorithm, namely Dijkstra's algorithm for graphs with non-negative edge costs.

```
void DIJKSTRA(const graph& G, node s, const edge_array⟨int⟩& cost,
    node_array⟨int⟩& dist, node_array⟨edge⟩& pred ){
  node_pq⟨int⟩ Q(G); int c; node u, v; edge e;
  forall_nodes(v, G)
    { pred[v] = 0;
      dist[v] = infinity;
      Q.insert(v, dist[v]); }
  dist[s] = 0;
  Q.decrease_inf(s, 0);
  while ( ! Q.empty())
    { u = Q.del_min();
      forall_adj_edges(e, u)
        { v = G.target(e);
          c = dist[u] + cost[e];
          if ( c < dist[v])
            { dist[v] = c;
              pred[v] = e;
              Q.decrease_inf(v, c);
            }
        } /* forall_adj_edges */
    } /* while */ }
```

As noted before, constructing the complete graph is much too expensive. Hence we have to adapt Dijstra's algorithm and construct it as needed starting from the source s.

In the priority queue Q we store the values of the shortest paths for the respective prefixes of the sequences. In each step we delete the node q with the minimal value k from Q. This node is then *expanded* which means that all neighboring nodes r of q, that are not already in Q, are inserted into Q with the value $k + c(q, r)$. In case r is already in Q, we relax the triangle inequality.

Dijkstra's algorithm guarantees, that the value of the node u that is removed from the priority queue Q equals the value of the shortest path from s to u.

We discuss now three strategies to reduce the number of nodes that are inserted into Q during the expansion of a node.

Assume we have for each node r of the grid graph a lower bound l for the cost of the shortest path in $r \to t$. For example

$$L(r \to t) := \sum_{1 \le i < j \le k} c(A^*(\sigma_i^r, \sigma_j^r))$$

defines a lower bound, since the projection of an optimal alignment A^* to i and j certainly has a cost bigger than the optimal alignment of σ_i and σ_j. That means

$$\sum_{1 \le i < j \le k} c(A^*(\sigma_i^r, \sigma_j^r)) \le c(r \to^* t)$$

Hence, in the expansion of a node q, we do not need to insert a neighbor r into Q if the following holds:

$$c(s \to^* q) + c(q, r) + L(r \to t) > U$$

where U is an upper bound for $c(A^*)$.

That means if the sum of the shortest path to q *plus* the cost of the edge from q to r *plus* a lower bound for a shortest path from r to t is already greater than an upper bound for an optimal alignment, then there cannot exist any optimal alignment through q and r.

Hence there is no need to insert r into Q.

4.13 Carillo-Lipman Bounding

The idea of Carillo und Lipman is subsumed in the following theorem which is valid for any optimal alignment A^*.

Theorem 10 (Carillo,Lipman). *Let A^* be an optimal alignment of the k sequences $S_1, \ldots, S_k$, $L = L(s \to t)$ be a lower bound for $c(A^*)$ and $U = c(A^{heur})$ be an upper bound for $c(A^*)$. Then the following inequality holds for each projection of a pair i, j of sequences:*

$$c(A^*_{i,j}) \le c(A^*(S_i, S_j)) + U - L$$

Proof.

$$\begin{aligned} U - L &\ge \sum_{1 \le i < j \le k} (c(A^*_{i,j}) - c(A^*(S_i, S_j))) \\ &\ge c(A^*_{i,j}) - c(A^*(S_i, S_j)),\ \forall\ 1 \le i < j \le k \\ \Rightarrow c(A^*_{i,j}) &\le c(A^*(S_i, S_j)) + U - L \end{aligned}$$

□

Applying the above theorem a shortest path cannot go through r if for any pair i, j holds:

$$c(A^*(\alpha_i^r, \alpha_j^r)) + c(A^*(\sigma_i^r, \sigma_j^r)) - c(A^*(S_i, S_j)) + L(s \to t) > U$$

Define $CL_{i,j}(q)$ as the left side of above inequality and call a node r *CL-valid*, if $CL_{i,j}(q) \leq U$ for all pairs i, j. Otherwise r is called *CL-invalid*.

The Carillo-Lipman bound was introduced 1988, but we will show that a combination of Standard-Bounding togehter with a different edge weighting performs provably better.

4.14 GDUS-Bounding

The so-called Goal-directed-unidirectional-search, also known as the $\mathcal{A}^*$-algorithm tries to direct the computation of the shortest path more into the direction of the sink t.

This happens using a lower bound $l(q \to t)$. First the cost of an edge (q, r) is redefined as follows:

$$c'(q, r) := c(q, r) - l(q \to t) + l(r \to t)$$

where $l(a \to b)$ is alower bound for the cost $c(a \to^* b)$ of a shortest path from a nach b ist. If $l()$ fullfills the *consistency*-condition:

$$c(q, r) \geq l(q \to t) - l(r \to t), \ \forall (q, r) \in E$$

then it is easy to show (exercise) that the redefinition of the edge costs does not change the optimal path and that the new edge weights are still positiv.

We will use $L(q \to t)$ as a lower bound. L fullfills the consistency condition::

$$\begin{aligned} c(q, r) + L(r \to t) &= \sum_{1 \leq i < j \leq k} (c(A^*(\sigma_i^r, \sigma_j^r)) + d(a_{i,*}, a_{j,*}) \\ &\geq \sum_{1 \leq i < j \leq k} (c(A^*(\sigma_i^q, \sigma_j^q)) \\ &= L(q \to t) \end{aligned}$$

The better the lower bound, the more directed the search. In the extrem case, if L is tight, then we extract only the nodes on the optimal path from the pqriority queue Q. In the other extreme if $L = 0$, we have standard bounding.

Now apply standard bounding on the grid graph with redefined edge costs. With the new edges costs a shortest path from s to q has costs

$$c'(s \to^* q) = c(s \to^* q) + L(q \to t) - L(s \to t).$$

Since $L(s \to t)$ is constant we can omit it and insert a neighboring node r with the value $\textsc{Prio}_q(r)$ into the priority queue Q, where

$$\textsc{Prio}_q(r) := c(s \to^* q) + c(q, r) + L(r \to t)$$

If we apply the standard-bounding technique, we will not insert a node r into Q whenever

$$\textsc{Prio}_q(r) > U$$

where U is an upper bound for $c(s \to^* t)$.

This does not exclude that r might be inserted later into Q, if it is visisted during the expansion of another neighboring node q'. However, if r is never inserted into Q, i.e. if

$$\textsc{Prio}(r) := \min_{(q,r)\in E} \textsc{Prio}_q(r) > U$$

then we call r *U-invalid*, otherwise *U-valid*.

The following theorem shows, that the standard bounding together with the redefined edge costs always exclude at least as many nodes as the Carrillo-Lipmann bound.

Theorem 11. *CL-ivalidity implies U-invalidity.*

Proof.

$$\begin{aligned}
\mathrm{CL}_{i,j}(q) &= c(A^*(\alpha_i^q, \alpha_j^q)) + c(A^*(\sigma_i^q, \sigma_j^q)) - c(A^*(S_i, S_j)) + L(s \to t) \\
&= c(A^*(\alpha_i^q, \alpha_j^q)) + c(A^*(\sigma_i^q, \sigma_j^q)) - c(A^*(S_i, S_j)) + \sum_{1\le m<n\le k} c(A^*(S_m, S_n)) \\
&= c(A^*(\alpha_i^q, \alpha_j^q)) + c(A^*(\sigma_i^q, \sigma_j^q)) + \sum_{\substack{1\le m<n\le k \\ (m,n)\neq(i,j)}} c(A^*(S_m, S_n)) \\
&\le c(A^*(\alpha_i^q, \alpha_j^q)) + c(A^*(\sigma_i^q, \sigma_j^q)) + \sum_{\substack{1\le m<n\le k \\ (m,n)\neq(i,j)}} \Big(c(A^*(\alpha_m^q, \alpha_n^q)) + c(A^*(\sigma_m^q, \sigma_n^q))\Big) \\
&= \sum_{1\le m<n\le k} c(A^*(\alpha_m^q, \alpha_n^q)) + \sum_{1\le m<n\le k} c(A^*(\sigma_m^q, \sigma_n^q)) \\
&= \sum_{1\le m<n\le k} c(A^*(\alpha_m^q, \alpha_n^q)) + L(q \to t) \\
&\le c(s \to^* q) + L(q \to t) \\
&=: \textsc{Prio}(q)
\end{aligned}$$

□

This means whenever $\mathrm{CL}_{i,j}(q) > U$ for all i, j then also $\textsc{Prio}(q) > U$, which means that q is always U-invalid, if it is CL-invalid.

The proposed version of the algorithm was implemented and the new bounding achieved a speedup of up to twenty times over the Carrillo Lipman bounding. An affine implementation combined with the divide-and-conquer strategy would be a masters project.

4.15 Combining it all

The-divide-and-conquer Algorithm (DCA) is called with a stop length Z at which the recursion stops. Obviously each DCA alignment is a heuristic alignment giving an upper bound U. The better the upper bound U the better our branch-and-bound algorithm works.

The following observation gets us further: DCA adheres a time versus quality tradeoff; the larger one chooses the parameter Z, the (provably) better is the alignment one gets, while the computation time increases due to the larger optimal alignments to be computed.

This motivates an iterative combination of both DCA and the optimal alignment procedure: Successively, we call DCA with increasing values of Z where, at each step, we can use the corresponding partial alignments from the previous step as upper bounds in the $\mathcal{A}^*$ algorithm.

Moreover, we can stop at any point of this procedure and have a heuristic alignment. The longer we wait, the better is the alignment – up to optimal.

Note that for iteratively computing better DCA alignments with larger Z values one only has to compute the cut positions once for the smallest values of Z.