10 Sequence Assembly

The exposition was prepared by Daniel Huson, Knut Reinert, and (a few bits) Clemens GrÃPpl. It is based on the following sources, which are all recommended reading:

1. Daniel Huson, Knut Reinert: Bioinfomatics support for genome sequencing projects, *in:* Thomas Lengauer (ed.), Bioinformatics - From Genomes to Therapies, Wiley-VCH, Weinheim, 2007. ISBN: 978-3-527-31278-8
2. Michael S. Waterman, Introduction to computational biology, Chapman and Hall, 1995. (Chapter 7)
3. Eugene W. (Gene) Myers *et al.*: A Whole-Genome Assembly of *Drosophila*, Science, 287:2196-2204, 24 March 2000.
4. Venter *et al.*: The sequence of the Human Genome, Science, 291:1304-1351, 16 February 2001.
5. Daniel Huson, Knut Reinert and Eugene Myers: The Greedy-Path Merging Algorithm for Sequence Assembly, RECOMB 2001, 157-163, 2001.

10.1 Genome Sequencing

Current sequencing technologies can only determine short consecutive pieces of DNA (Depending on the method $20-60$, $150-250$, and $700-1000$). 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, an assembly of the $130Mb$ Drosophila genome was published.

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

Since then many genomes have been sequenced using the whole shotgun method.

10.2 The technologies

We give now three short animations to illustrate how the still most commonly used method (capillary gel electrophoresis) and two new, quite mature technologies (454 sequencing and Solexa sequencing) work.

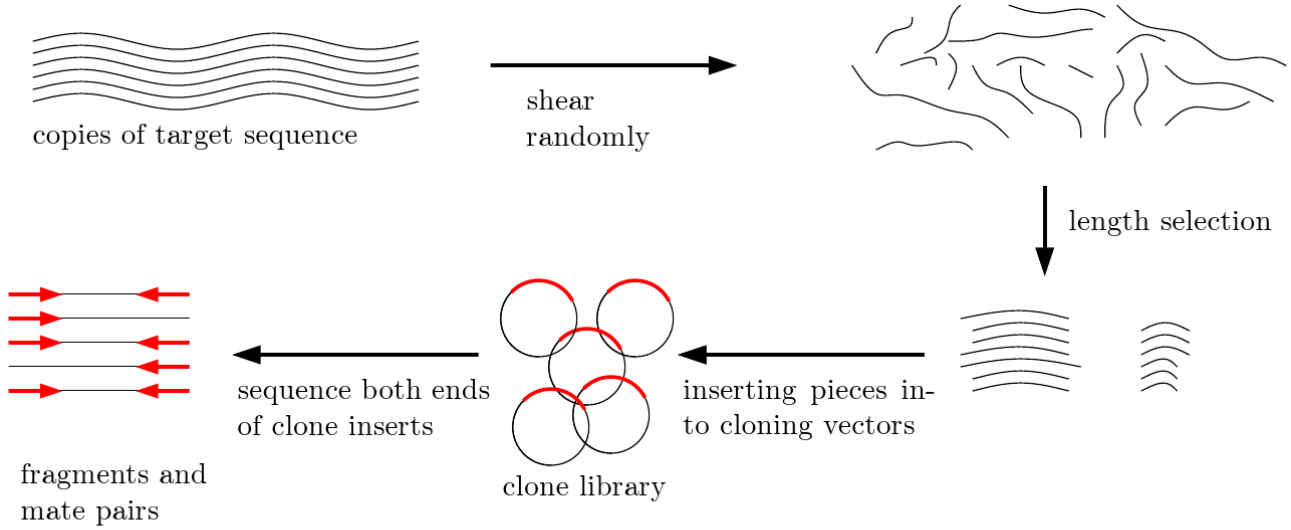


Figure 2.1: Experimental protocol of shotgun sequencing

10.3 The technologies

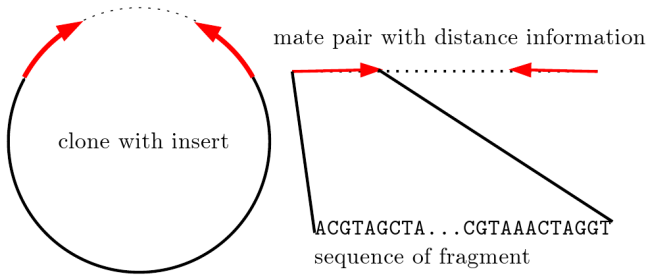


Figure 2.2: Fragments and mate-pairs

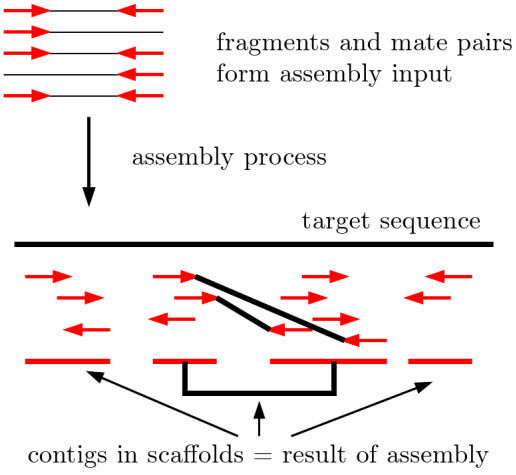
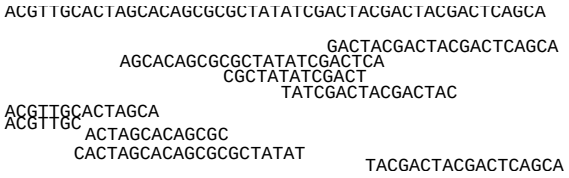
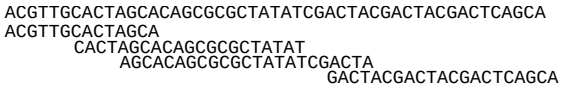
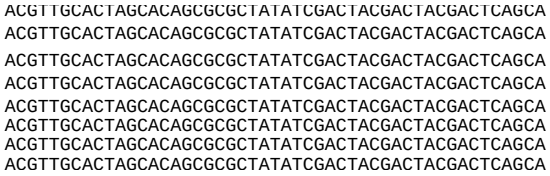
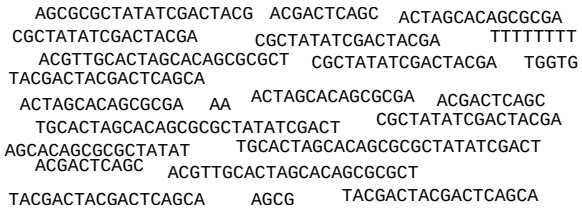
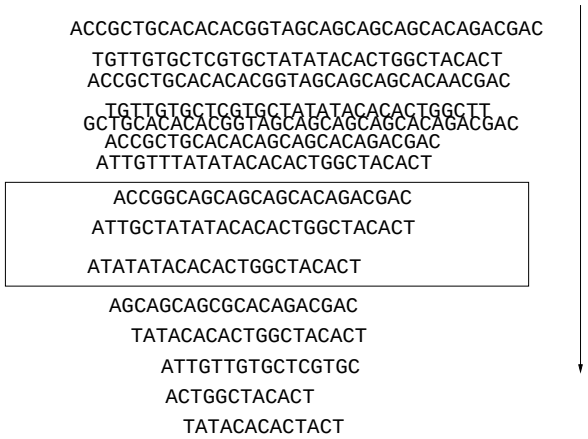


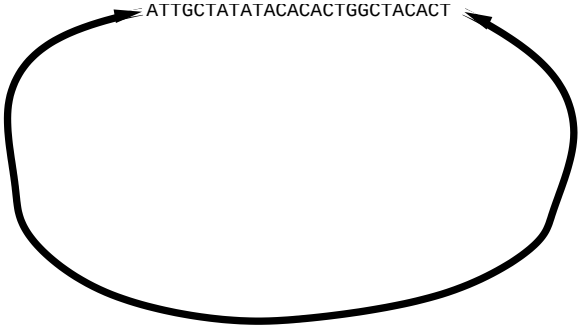
Figure 2.3: Assembly process

10.4 The big picture – From molecule to sequence

Whole genome shotgun sequencing (WGS)	**Illustration**	**Clone by clone sequencing (CBC)**
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 to tile the target (minumum tiling least costly but most difficult) ⇒ *Physical mapping*
Big source sequence is copied many times. . .		*all* source sequences (e.g. *40000* for human) are copied many times
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*

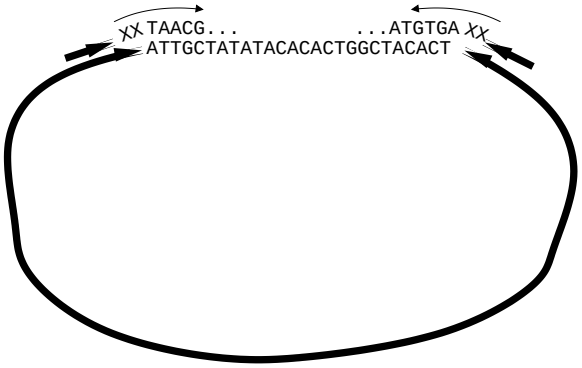
[0] `http://www.biology-online.org/dictionary/sonication` : The process of disrupting biologic materials by use of sound wave energy.

and inserted into *cloning vectors*.



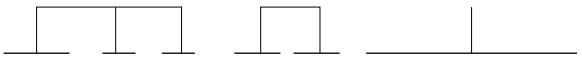
and *all* inserted into *cloning vectors*.

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 ≈ 1% error.



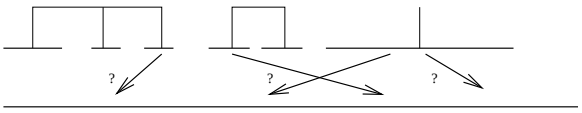
first approaches did not use double barrel, later they did.

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



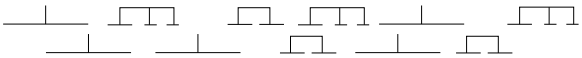
Each clone is a collection of *scaffolds*.

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.

10.5 Shotgun sequencing data

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

Shotgun sequencing of a produces a set of reads

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

of average length 550 (at present).

Essential characteristics of the data:

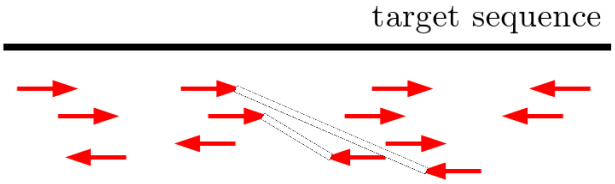
- Incomplete coverage of the source sequences.
- Sequencing introduces errors at a rate of about %1 for the first 500 bases, if carefully performed.
- The reads are sampled from both strands of the source sequence and thus the orientation of any given read is unknown.

10.6 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

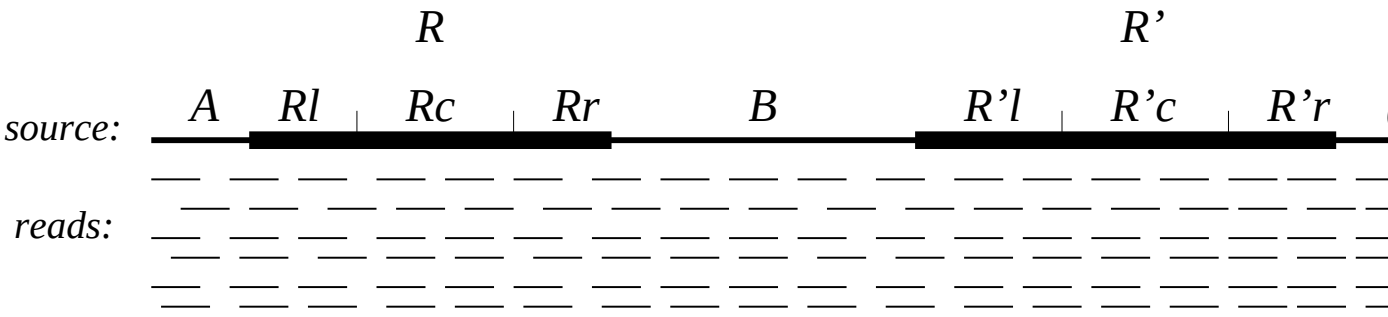
- if $s_j < e_j$ then f_j can be aligned to the substring $S[s_j, e_j]$ with less than $\epsilon \cdot \mid f_j \mid$ differences, and
- if $s_j > e_j$ then f_j can be aligned to the substring $\overline{S[e_j, s_j]}$ with less than $\epsilon \cdot \mid f_j \mid$ differences, then
- $\bigcup_{j=1}^{R}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



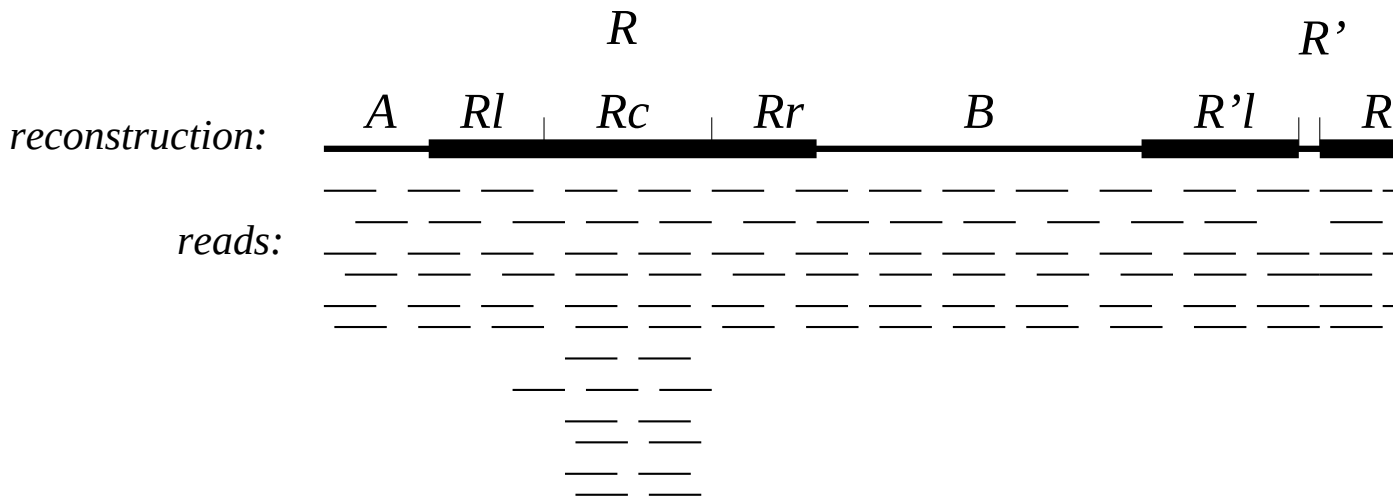
The string S is the reconstructed source string. The integer pairs indicate where the reads are placed and the order of s_i and e_i indicate the orientation of the read f_i, i.e. whether f_i was sampled from S or its complement $\overline{S}$.

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*:



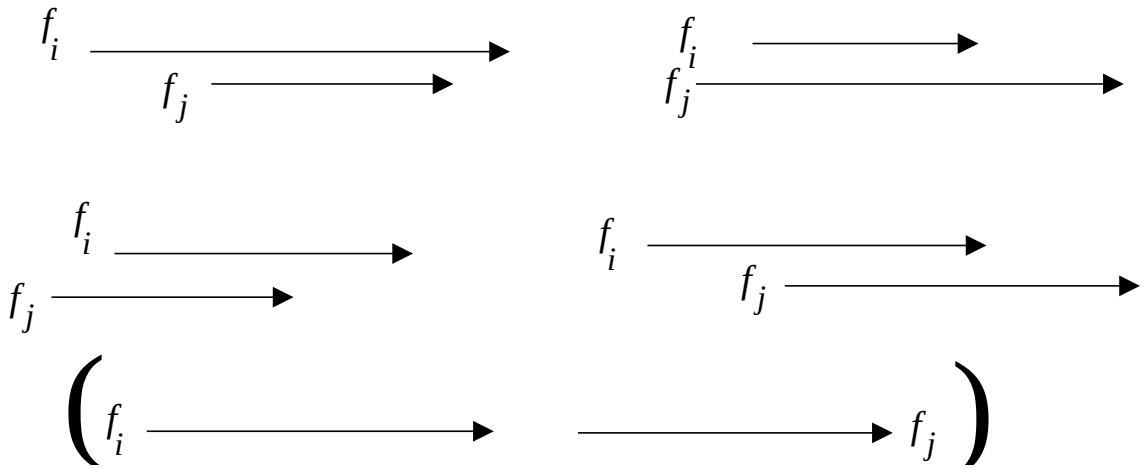
10.7 Sequence assembly in three stages

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.

10.8 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 (or not!):



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

$$2 \cdot \binom{27000000}{2} \approx 1458000000000000 \approx 1.5 \cdot 10^{15}$$

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 $a = a_1a_2 \dots a_m$ and $b = b_1b_2 \dots b_n$; then we want to compute:

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

10.9 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: $a = a_1a_2 \dots a_n$ and $b = b_1b_2 \dots b_m$, $s(\cdot, \cdot)$ 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

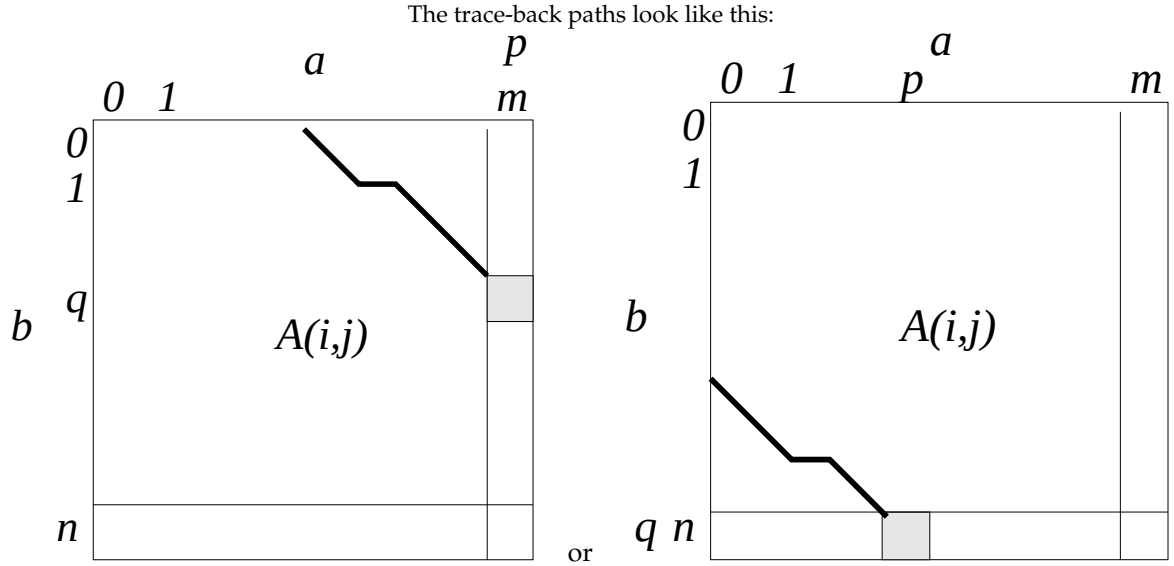
Runtime is $O(nm)$.

Given two reads $a = a_1a_2 \dots a_m$ and $b = b_1b_2 \dots b_n$. For the matrix $A(i, j)$ computed as above, set $(p, q) := \arg\max\{A(i, j) \mid i = m \text{ or } j = n\}$.

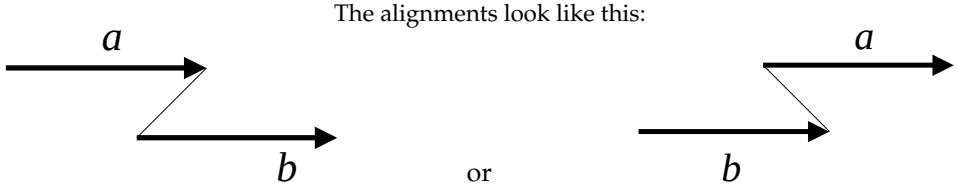
There are two cases:

$p = m$ or $q = n$

The trace-back paths look like this:



The alignments look like this:



10.10 Faster overlap detection

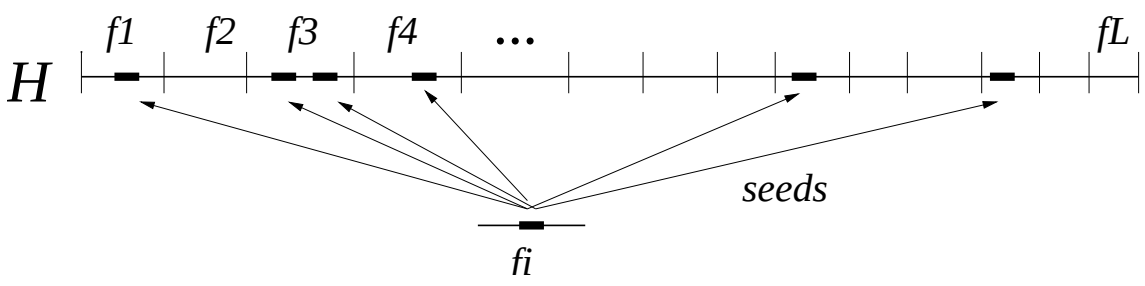
Dynamic programming is too slow for large sequencing projects. Indeed, it is wasteful, as in assembly, only high scoring overlaps with more than e.g. 96% identity play a role.

One can use a *seed and extend* approach (as used e. g. 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)

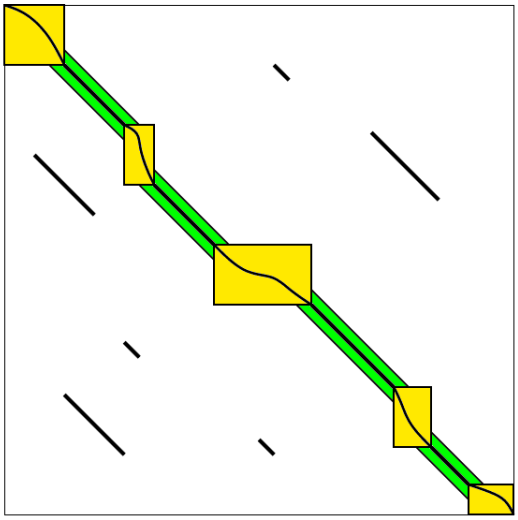


Figure 2.6: Seed and extend paradigm. A banded alignment is conducted that explores narrow bands around the seeds and possibly larger regions between the seeds.

10.11 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.

The figure shows a true overlap between f_i and f_j and a repeat induced overlap between f_k and f_l.

10.12 Avoiding repeat-induced overlaps

We want to avoid the computation of repeat-induced overlaps. One strategy is to only consider those 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 the program *Repeatmasker*. Portions of reads that match a known repeat are labeled as "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.

10.13 Celera's overlapper

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

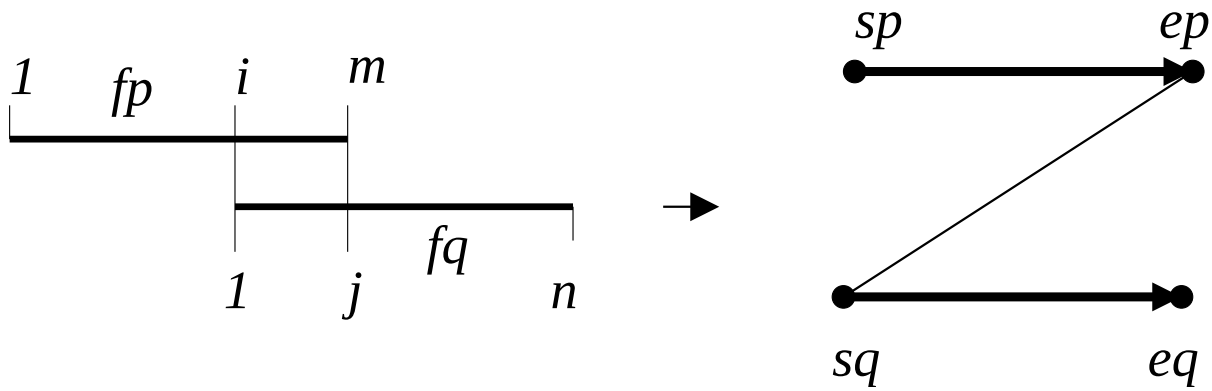
Overlapping all pairs of 27 million reads of human DNA using this program takes about 10 days, running on about 10-20 four-processor machines (Compaq ES40), each with $4GB$ of main memory.

The input data file is about $50GB$. To parallelize the overlap computation, 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 held in main memory.

10.14 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.

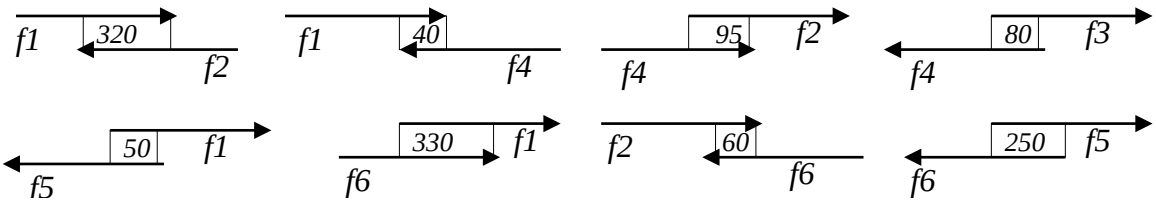
An overlap between $f_p = f_{p_1} f_{p_2} \dots f_{p_m}$ and $f_q = f_{q_1} f_{q_2} \dots f_{q_n}$ gives rise to an undirected *overlap edge* e between s_p, or e_p, and s_q, or e_q, depending on the orientation of the overlap, e.g.:



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.

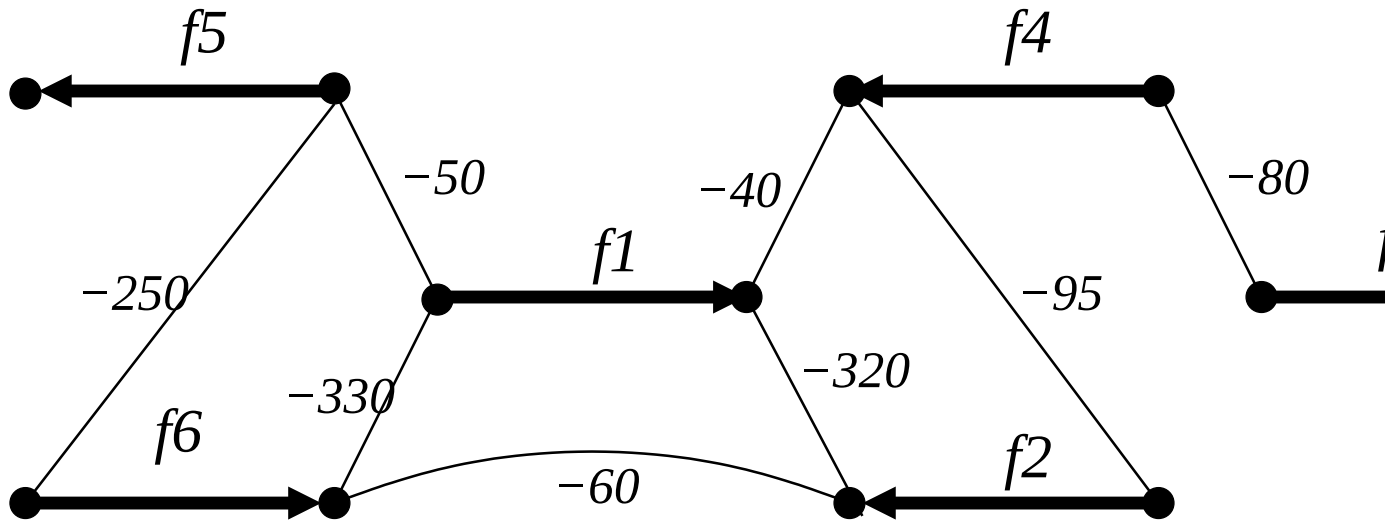
10.15 Example

Assume we are given 6 reads $\mathcal{F} = \{f_1, f_2, \dots, f_6\}$, 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*:



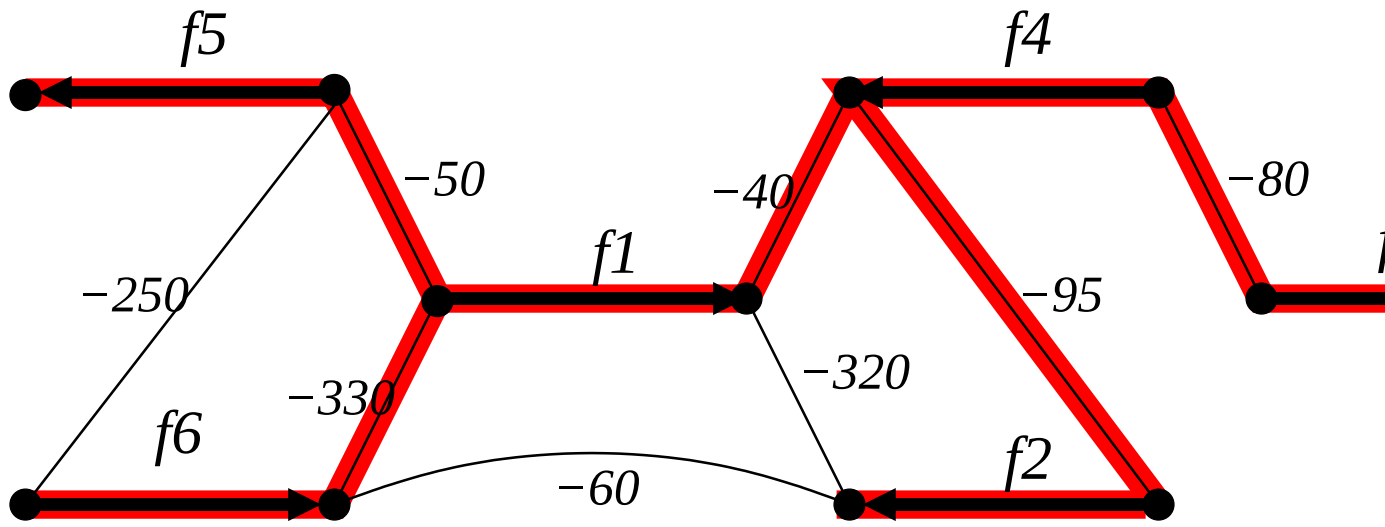
Each read f_p is represented by a read edge (s_p, e_p) of length $| f_p |$. Overlaps off the start s_p, or end e_p, of f_p are represented by overlap edges starting at the node s_p, or e_p, respectively. Each overlap edge is labeled by -1 times the overlap length.

10.16 The layout phase

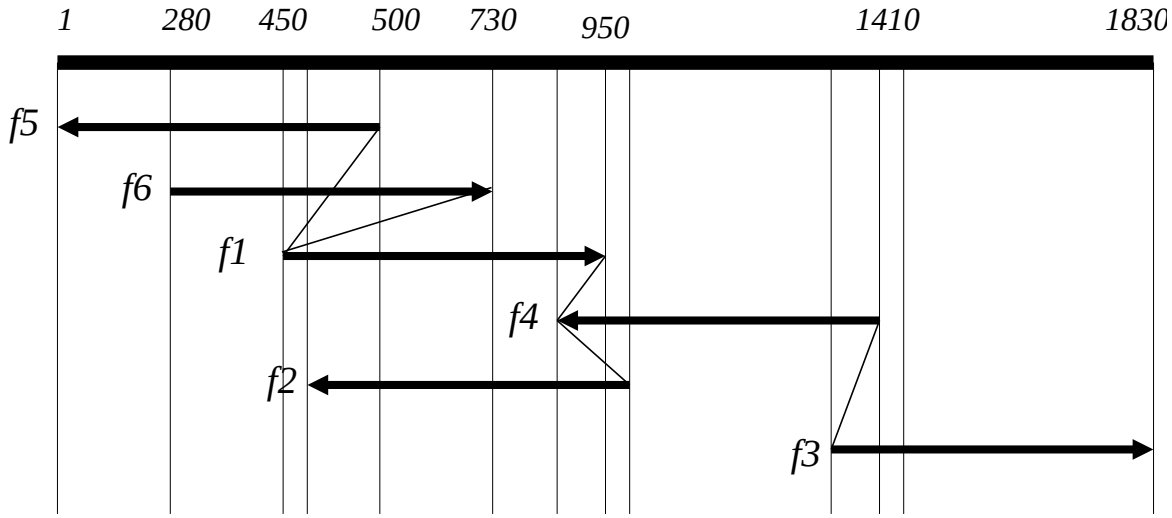
The goal of the layout phase is to arrange all reads into an approximate multi-alignment. This involves assigning coordinates to all nodes of the overlap graph *OG*, and thus, determining the value of s_i and e_i for each read f_i.

A simple heuristic is to select a *spanning forest* of the overlap graph *OG* that contains all read edges. [1]

[1](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.)

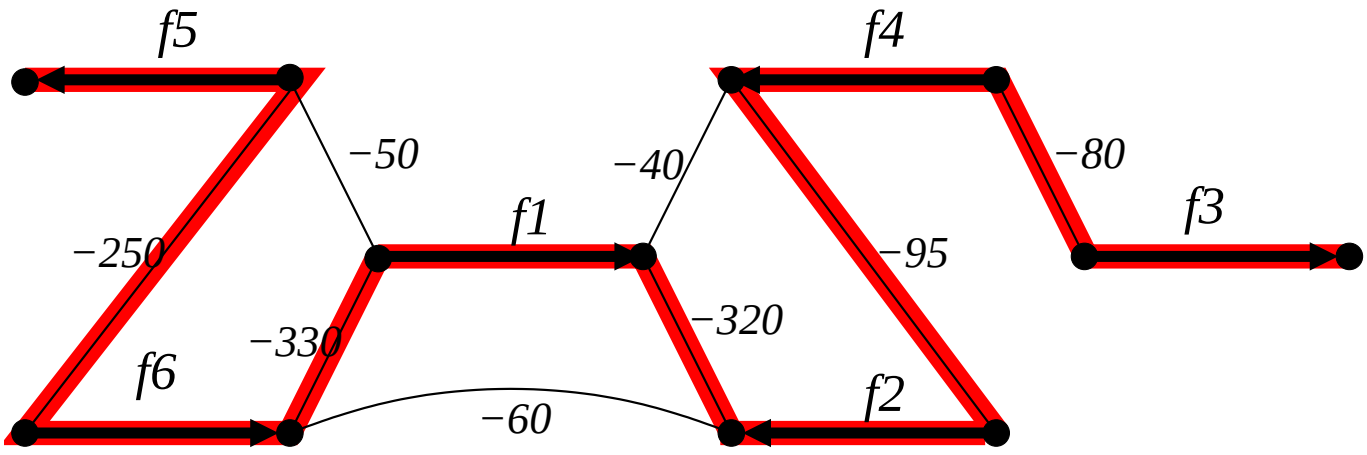


Such a subset of edges positions every read with respect to every other, within a given connected component of the graph:



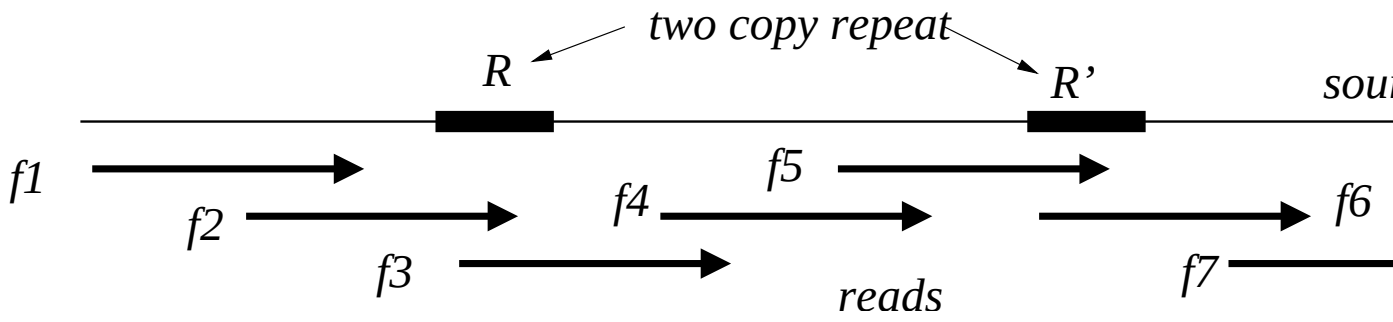
Such a putative alignment of reads is called a *contig*.

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

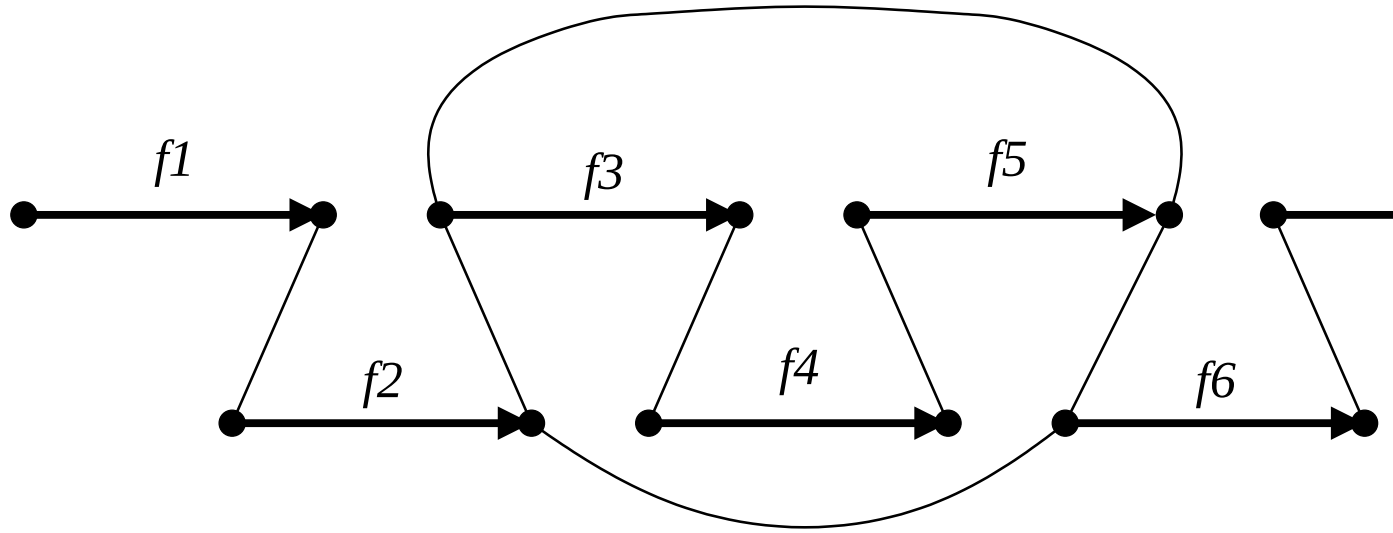


10.17 Repeats and the layout phase

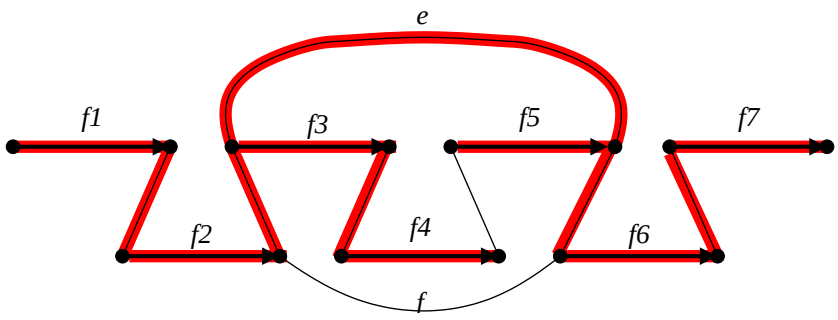
Consider the following situation:



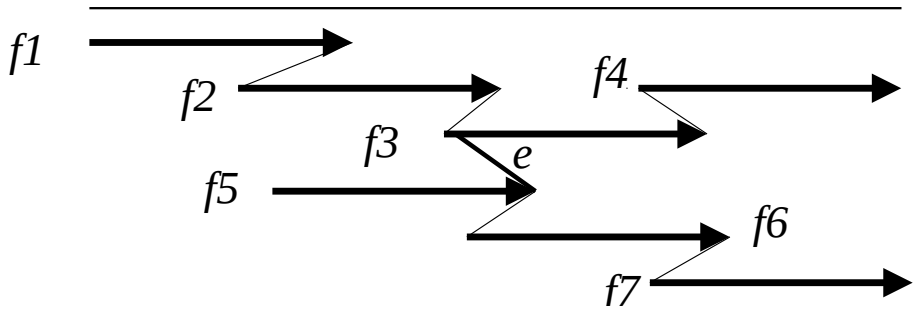
This gives rise to the following overlap graph:



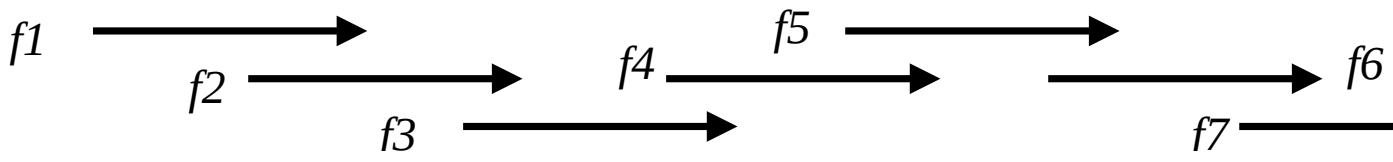
Consider this spanning tree:



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.

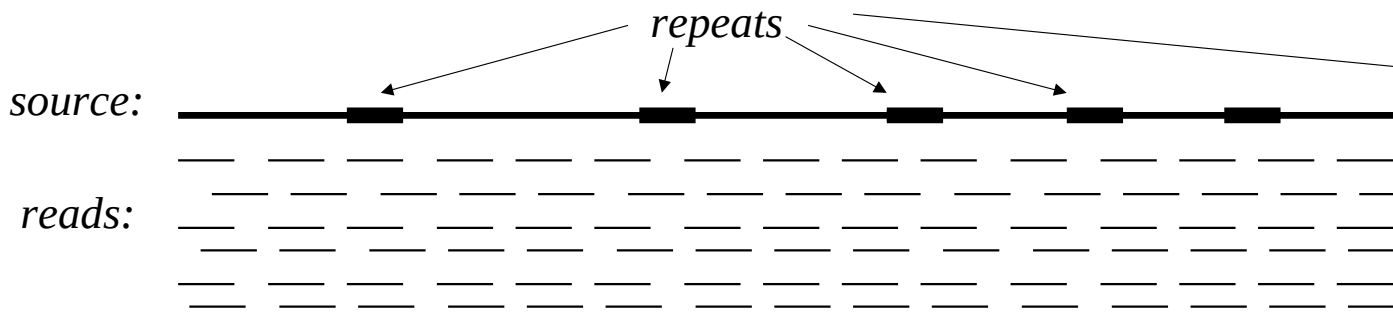
10.18 Unitigging

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).

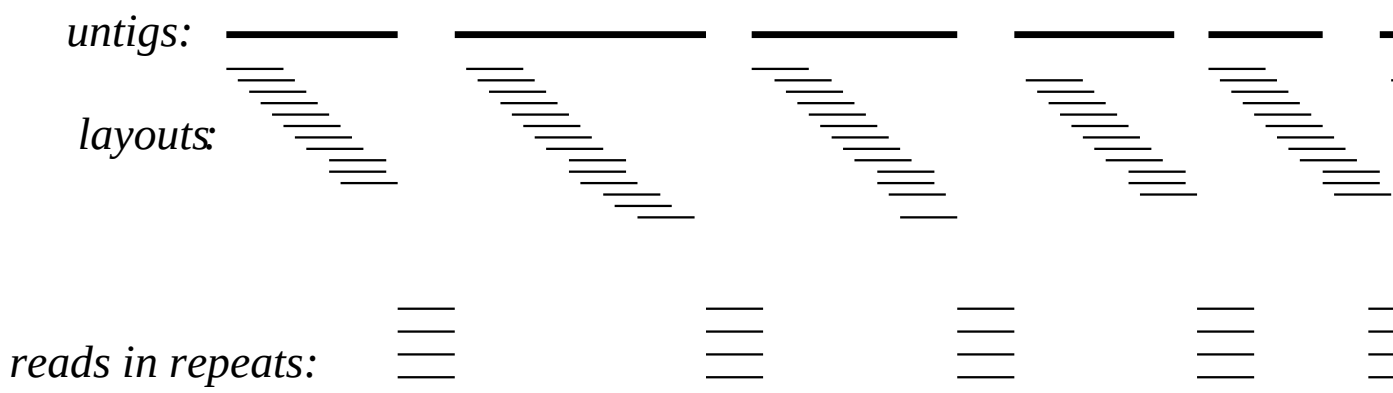
Thus, the layout phase proceeds in two stages:

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:

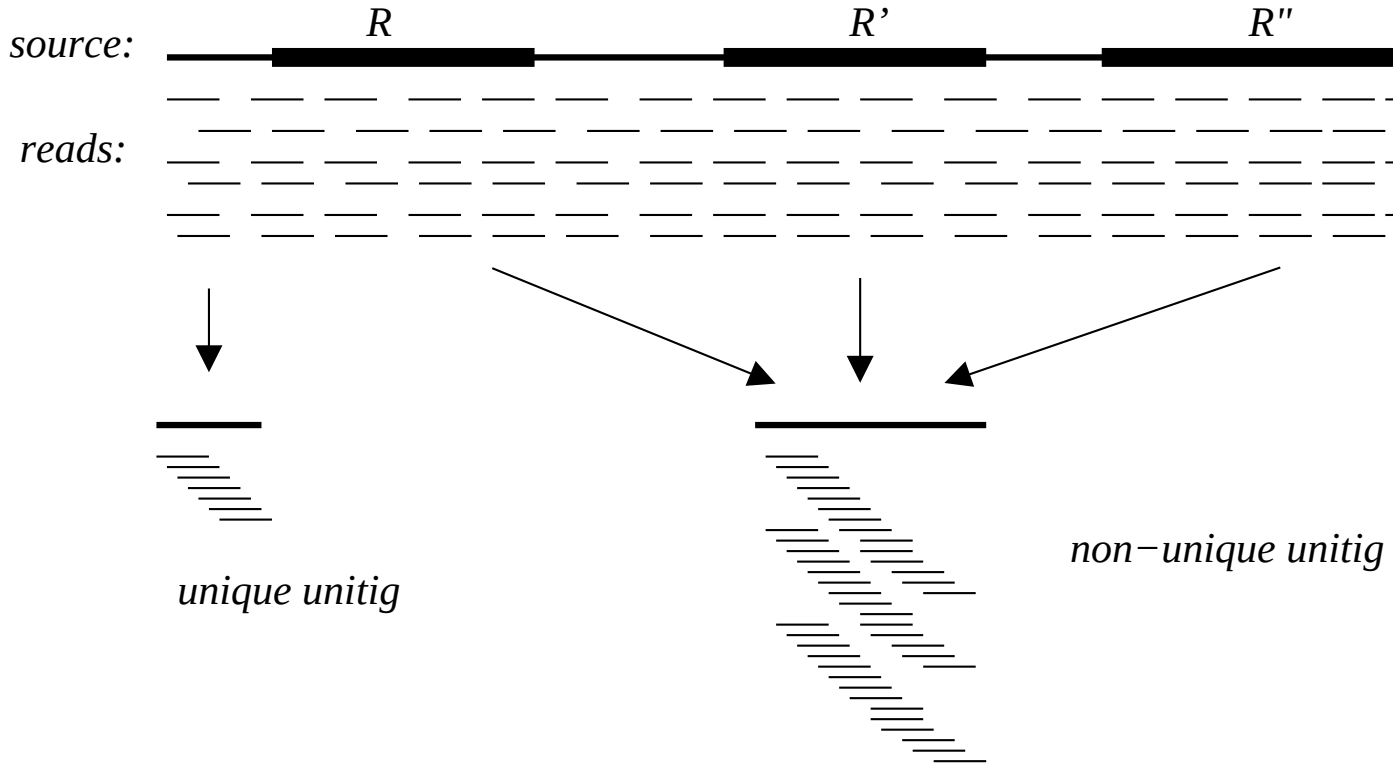


Reads that form consistent chains in the overlap graph are assembled into unitigs and the remaining *"repetitive"* reads are processed later:



10.19 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*.

10.20 The Poisson distribution

In probability theory and statistics, the Poisson distribution (pronounced, after Simeon-Denis Poisson (1781-1840)) is a discrete probability distribution that expresses the probability of a number of events occurring in a fixed

period of time if these events occur with a known average rate and independently of the time since the last event. The Poisson distribution can also be used for the number of events in other specified intervals such as distance, area or volume.

If the expected number of occurrences in this interval is λ, then the probability that there are exactly k occurrences (k being a non-negative integer, $k = 0, 1, 2, \ldots$) is equal to

$$f(k;\ \lambda) = \frac{\lambda^k e^{-\lambda}}{k!},$$

where

- e is the base of the natural logarithm ($e \doteq 2.71828$)
- k is the number of occurrences of an event - the probability of which is given by the function
- $k!$ is the factorial of k

- λ is a positive real number, equal to the expected number of occurrences that occur during the given interval. For instance, if the events occur on average 4 times per minute, and you are interested in the number of events occurring in a 10 minute interval, you would use as your model a Poisson distribution with $\lambda = 10 \cdot 4 = 40$.

As a function of k, this is the probability mass function. The Poisson distribution can be derived as a limiting case of the binomial distribution. The Poisson distribution can be applied to systems with a large number of possible events, each of which is rare. A classic example is the nuclear decay of atoms.

(Cited from `http://en.wikipedia.org/wiki/Poisson_distribution`, with modifications.)

10.21 Identifying unique unitigs

Under assumption that the sampling of reads from the target is done uniformly, we can model the arrival of the fragments start positions along the target sequence by a *Poisson process*.

Let R be the number of reads and G the estimated length of the source sequence. Then we expect on average R/G arrivals of fragments per base. This is called the *rate* of the Poisson process.

In a Poisson process with rate λ, the distances between the sites are independent exponentially distributed random variables with mean $1/\lambda$; and the probability that we have k sites in an interval $[s, s+t]$ is $e^{-\lambda t}(\lambda t)^k/k!$. [Waterman, p. 89]

Let ρ be the length of fragments and assume $\rho \ll G$. One can show that the fraction of G covered by k fragments is $e^{-c}c^k/k!$, where $c = R\rho/G$.

For a unitig with k reads and approximate length ρ, the probability of seeing the k start positions in the interval of length ρ is (neglecting border effects)

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

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,

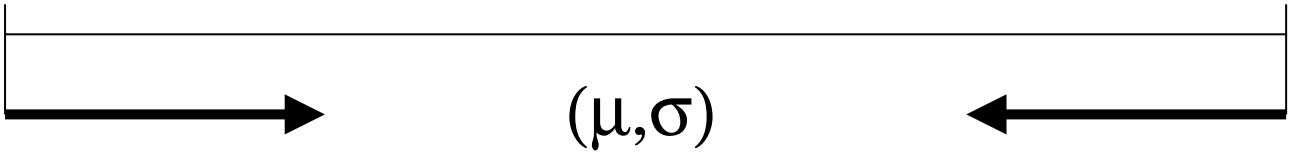
$$c - (\log 2)k.$$

The sign of the arrival statistic tells which of the two cases is more likely. However, for the purpose of unitigging, we want to be really sure, thus a unitig is deemed *unique* only if its arrival statistic has a positive value of 10 or more.

10.22 Mate pairs

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

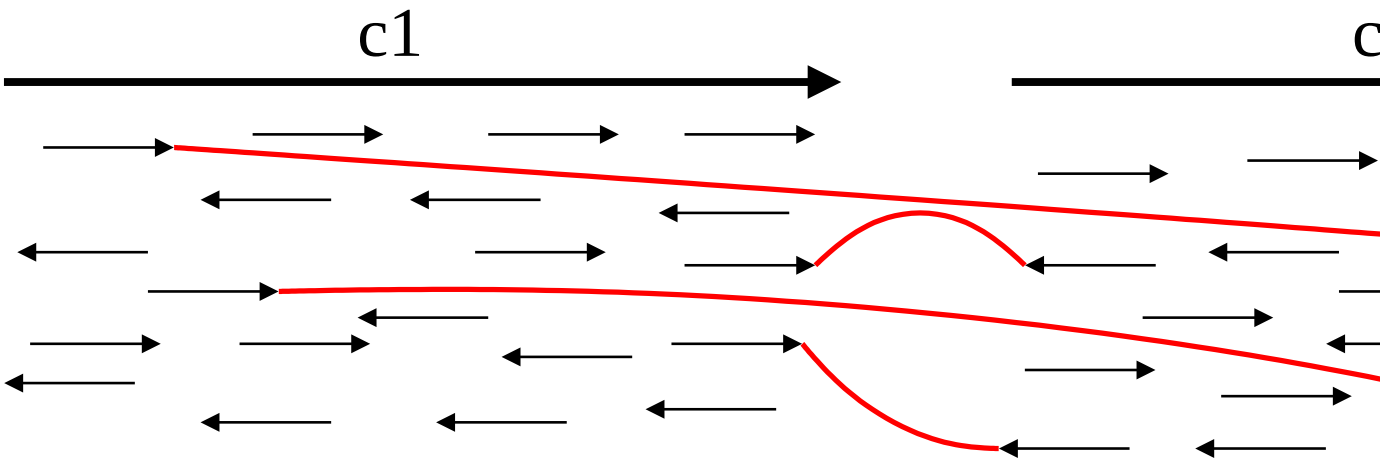
Recall that modern shotgun sequencing protocols employ a so-called *double barreled* shotgun. That is, longer clones of a given fixed length are sequenced from both ends and one obtains a pair of reads, a *mate pair*, whose relative orientation and mean μ (and standard deviation σ of) length are known:



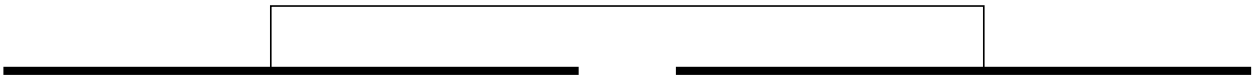
Typical clone lengths are $\mu = 2kb, 10kb, 50kb$ or $150kb$. For clean data, $\sigma \approx 10\%$ of μ. Mate pair mismatching is a problem and can effect $10 - 30\%$ of all pairs.

10.23 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]:



10.24 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 as an independent estimate (μ, σ) for the true distance between the two contigs.

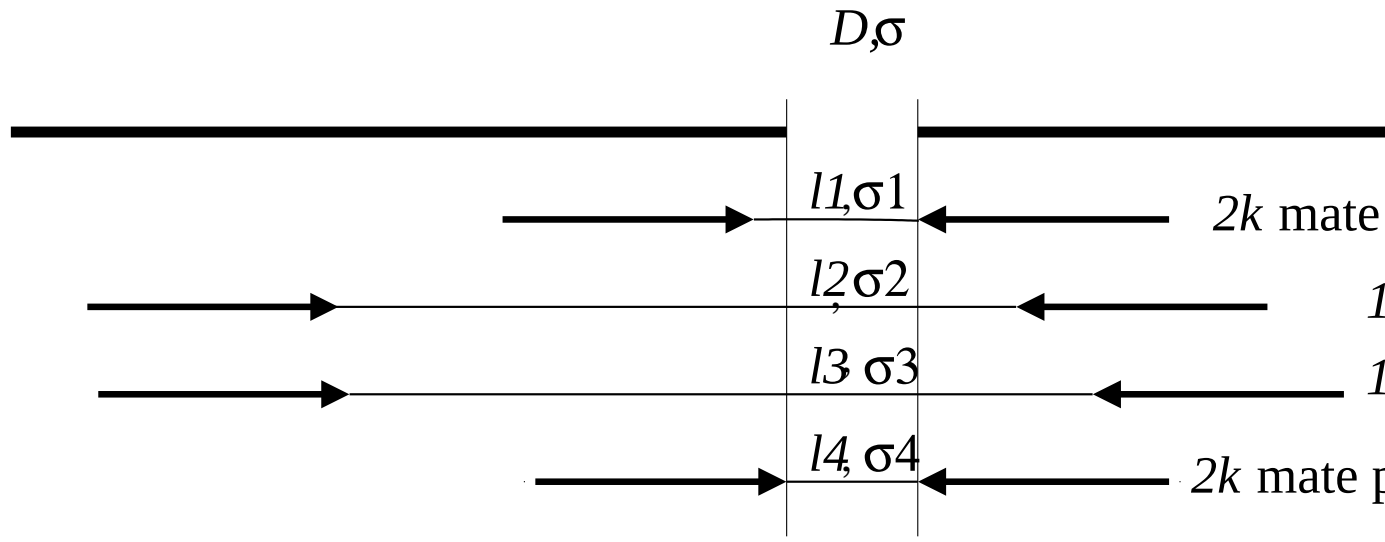
Following standard statistical practice, these measurements (μ_1, σ_1), (μ_2, σ_2), $\ldots$, (μ_k, σ_k) of the distanc between the two contigs c_1 and c_2 can be combined by taking a weighted average, using the reciprocal variances as weights, as follows:

[2]engl. scaffold = dt. GerÃ¼st

Let $p := \sum \frac{l_i}{\sigma_i^2}$ and $q = \sum \frac{1}{\sigma_i^2}$. Then the distance between c_1 and c_2 is estimated as

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

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. (The "3" is a magic constant.)

10.25 The significance of mate pairs

Can we really rely on mate pair information? 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.

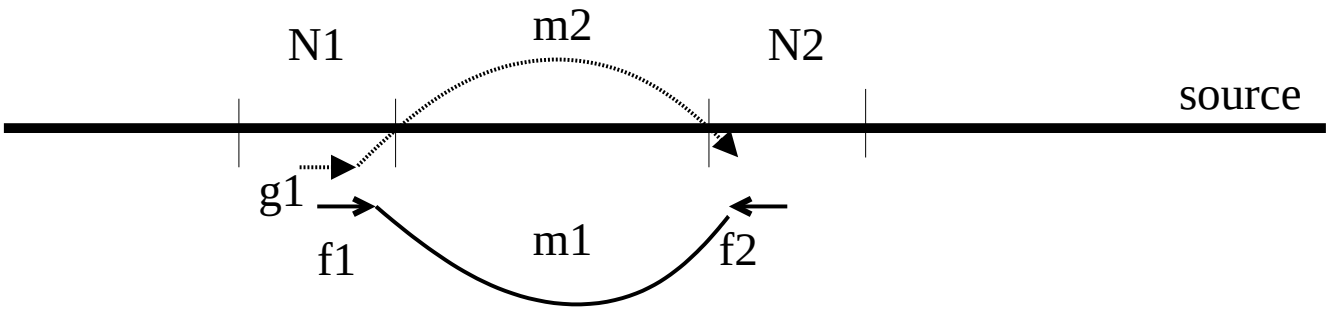
This is because that probability that two false mate pairs occur that confirm each other, is extremely small.

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. Then 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

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

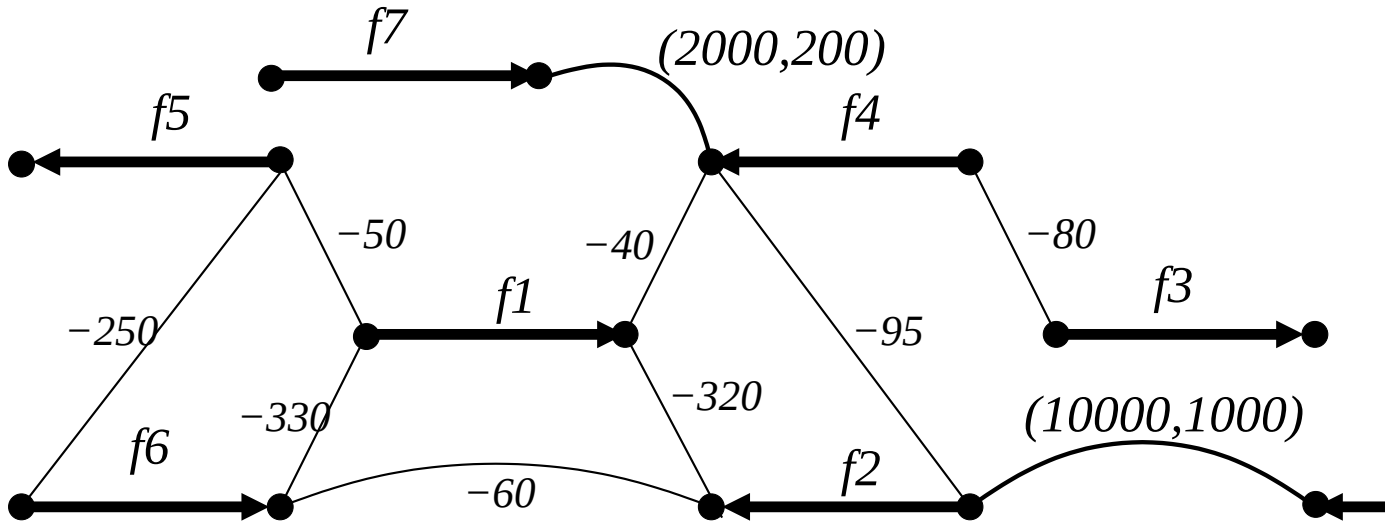
This does not take into account that N_1 certainly contains many reads with correct mate pairs.

10.26 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*:

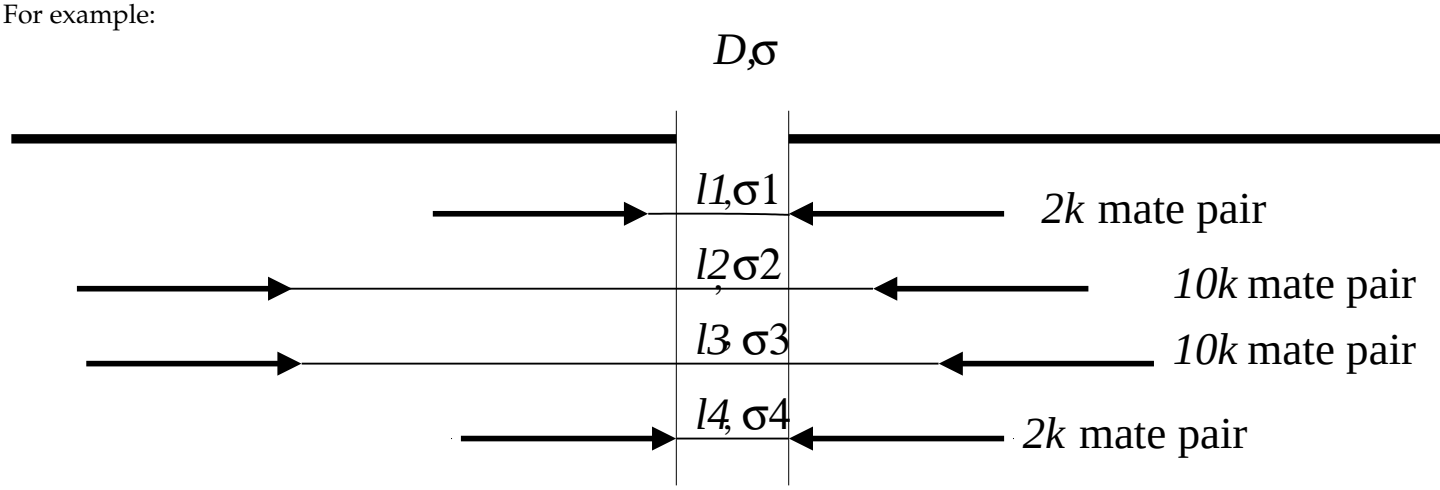


The overlap-mate graph is good for visualization, but it turns out that a more useful concept is obtained by "lifting" the mate pair information to the level of contigs.

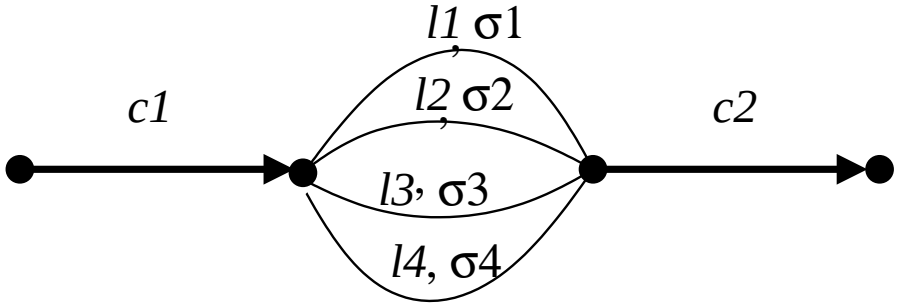
10.27 The contig-mate graph

Given a set of $\mathcal{F}$ of fragments and a set of assembled contigs $C = \{c_1, c_2, \ldots, c_t\}$. 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.

For example:



leads to:



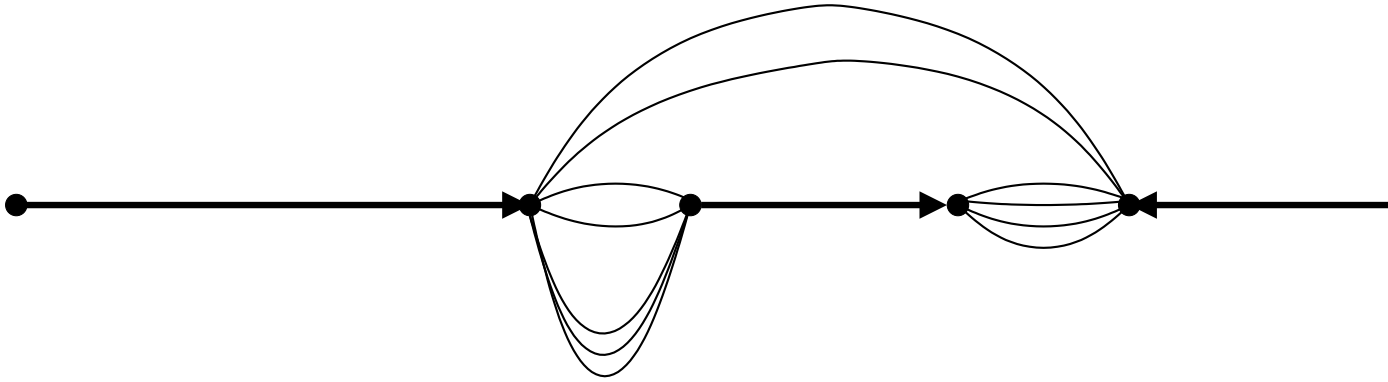
10.28 Edge bundling

The complexity is further reduced by *edge bundling*. Consider two contigs c_1 and c_2, joined by several 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 again labeled by a pair (μ, σ).

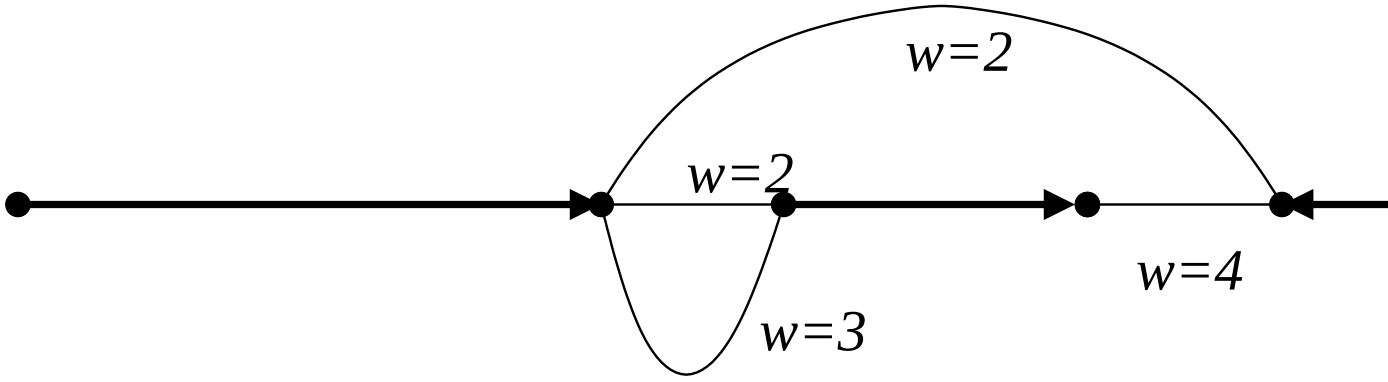
(A heuristic is used to compute these subsets: Repeatedly bundle the median-length simple mate edge with all mate edges within three standard deviations of it, until all simple mate edges have been bundled.)

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 $e_1, \ldots, e_k$.

For example, consider the following graph:

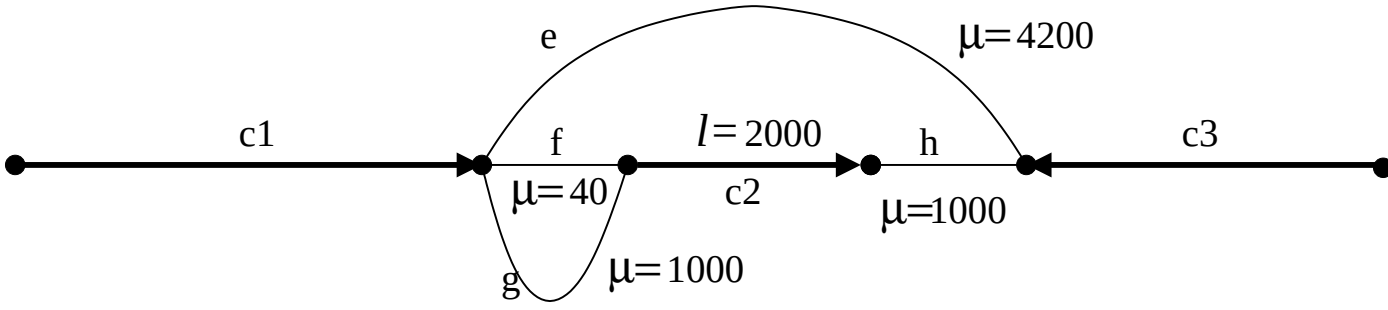


Assuming that mate edges drawn together have similar lengths and large enough standard deviation, edge bundling will produce the following graph:

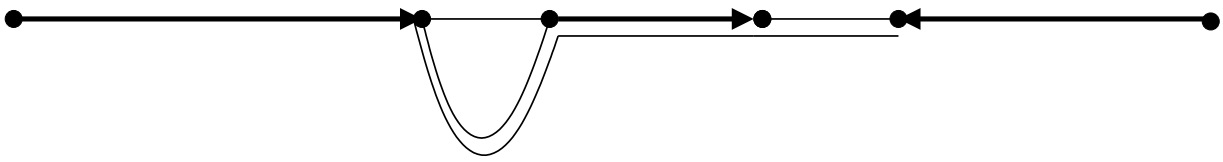


10.29 Transitive edge reduction

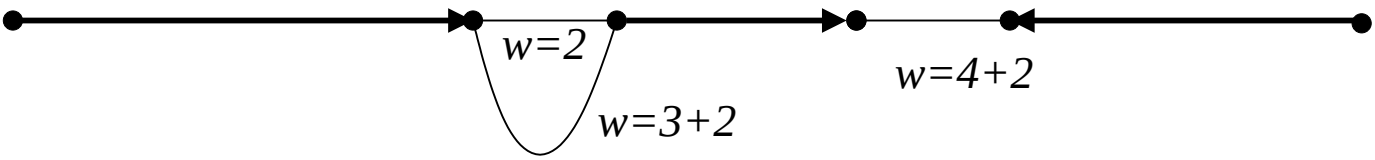
Yet another trick used for simplification is *transitive edge reduction*. Consider the previous graph with some specific edge lengths:



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

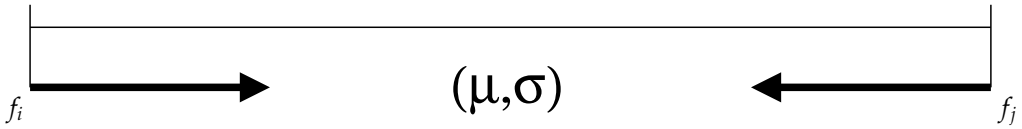
$$\sigma(P) := \sqrt{\sum_{m_i} \sigma(m_i)^2}\,.$$

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 $\mid \mu(e) - l(P) \mid \leq C \cdot \max\{\sigma(e), \sigma(P)\}$ 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 $w(e)$.

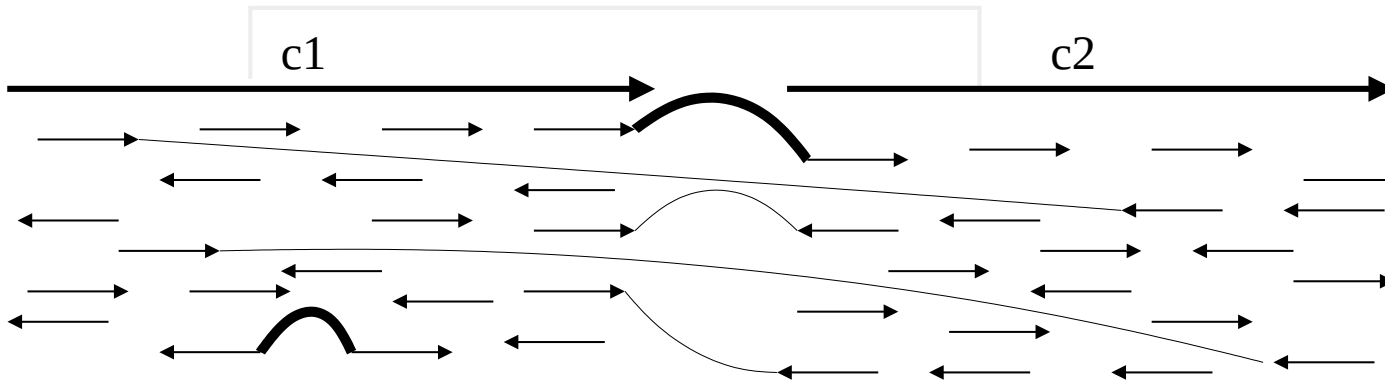
In the following, we will assume that any contig-mate graph considered has been edge-bundled and perhaps also transitively reduced to some degree.

10.30 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. the arrows are facing each other, and both are at approximately the *right distance*, i.e., $\left|\mu - |s_i - s_j|\right| \leq 3\sigma$. Otherwise, m is *unhappy*. Two unhappy mates (due to their orientation) are highlighted here:



10.31 Ordering and orientation of the contig-mate graph

Given a collection of contigs $C = \{c_1, c_2, \ldots, c_k\}$ 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 C)* is a map $\phi : V \to \mathbb{N}$ such that $\mid \phi(b_i) - \phi(e_i) \mid = l(c_i)$ for all contigs $c_i \in C$. In other words, it is 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.

10.32 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

1. c_i and c_j have the correct relative orientation, and
2. the distance between v and w is approximately correct.

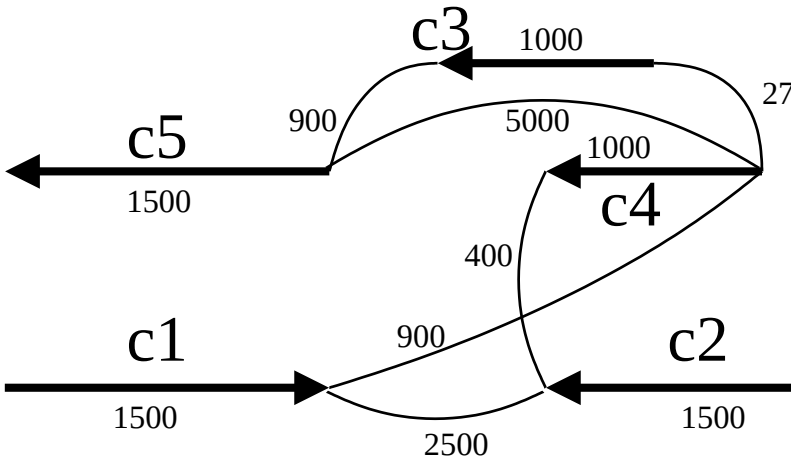
In other words, we require that either

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

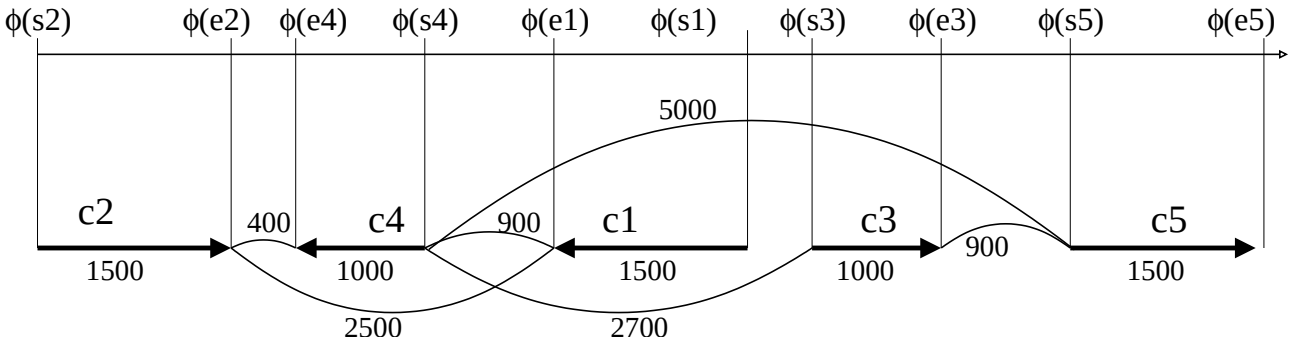
Otherwise, e is *unhappy*.

10.33 Example

Given the following contig-mate graph:



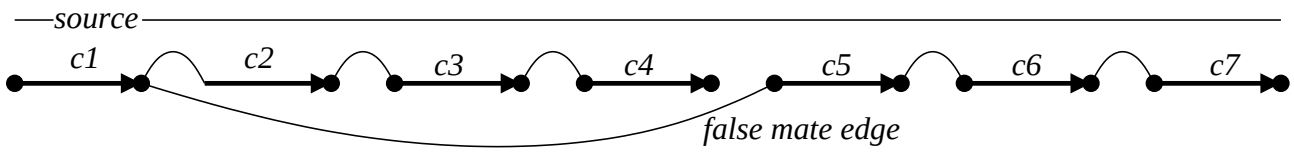
An ordering ϕ assigns coordinates $\phi(v)$ to all nodes v and thus determines a layout of the contigs:



10.34 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, because a single false mate edge can lead to incorrect interleaving of contigs from completely different regions of the source sequence:

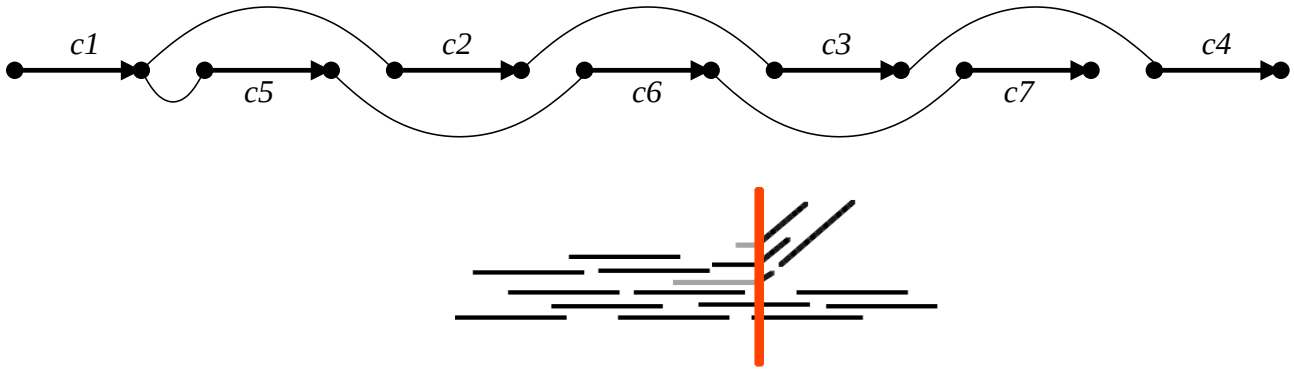


Figure 2.10: From left to right, the reads overlap consistently, until we reach the "branch-point" at the position indicated by a dotted line. From this position onward, the data partitions into two mutually incompatible chains of overlapping reads. Here, the reads to the left of the branch-point lie in the interior of a repeat whereas the reads that span the branch-point overlap with unique flanking sequence.

10.35 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

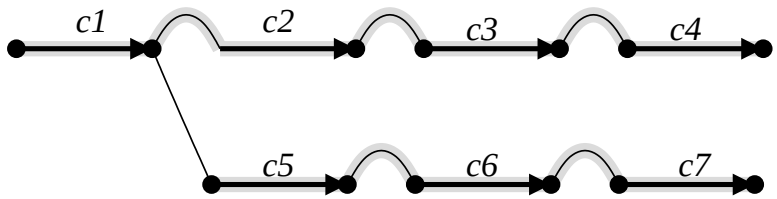
be represented as follows:

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

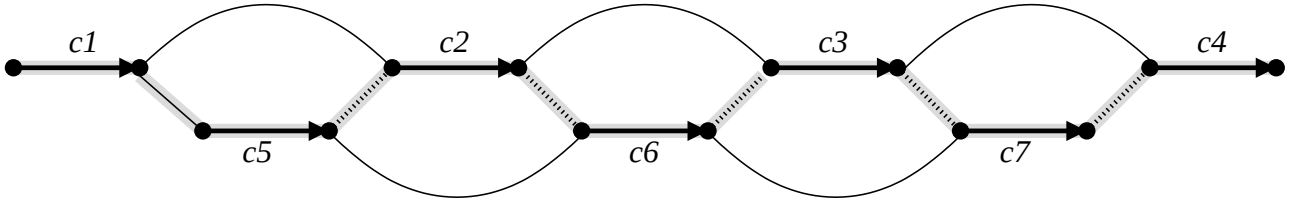
- every contig edge is selected, and
- every node is incident to at most two selected edges.

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:



10.36 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", maintaining a valid *scaffolding* $S \subseteq E$, as follows:

Initially, all contig edges $c_1, c_2, \ldots, c_k$ are selected, and no other edges. At this stage, the graph consists of k *selected paths* $P_1 = (c_1), \ldots, P_k = (c_k)$.

Then, in order 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:

1. We *attempt* to merge the two paths (as will be described soon) to obtain a new path P_k, but
2. We *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.

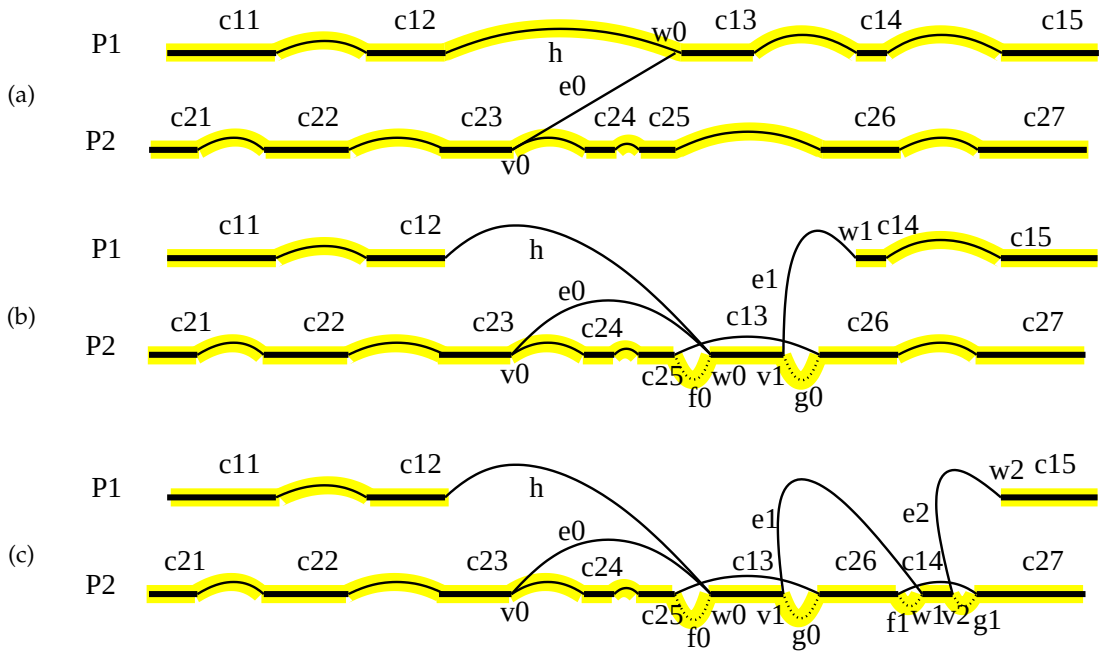
10.37 The greedy path-merging algorithm

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

10.38 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_2) and w_0 (incident to P_1). Merging of P_1 and P_2 is attempted as follows:

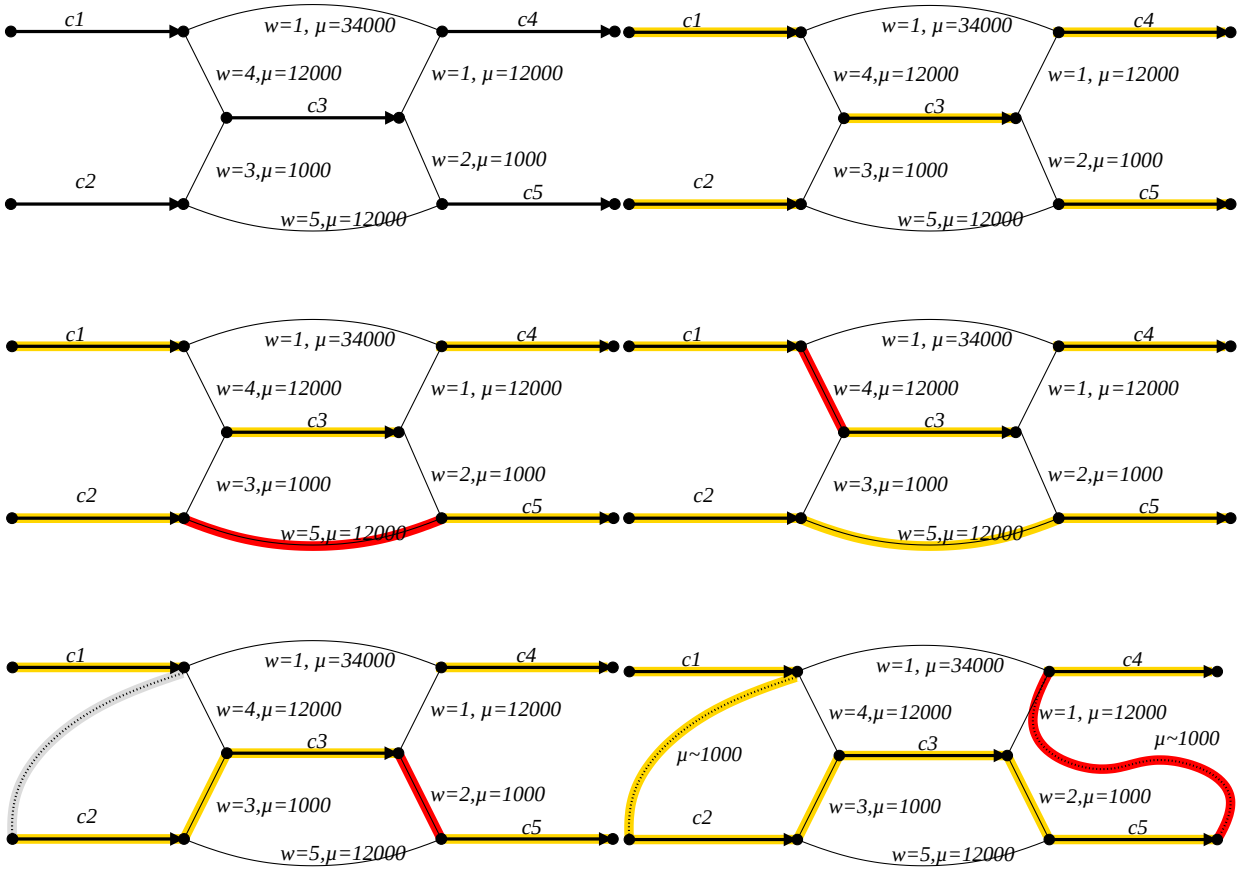


This algorithm returns *true*, if it successfully produced a new selected path P containing all contigs edges in P_1 and P_2, and *false*, if it fails.

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.

10.39 Example

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

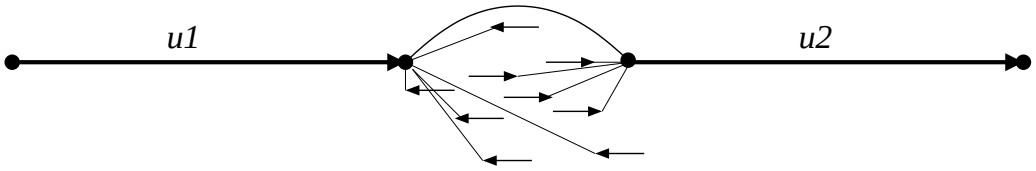


The final scaffolding is $(c_1, c_2, c_3, c_5, c_4)$.

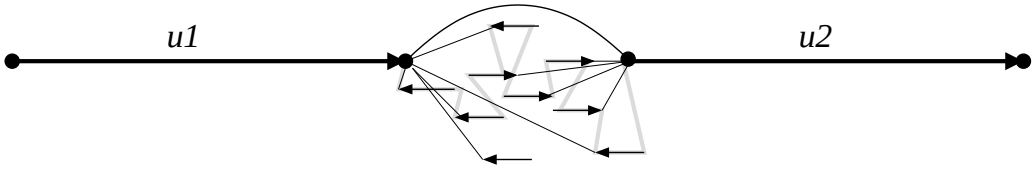
10.40 Repeat resolution

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 (and manual inspection by experts, for the remaining cases) 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:



10.41 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 done right after computing the contigs/unitigs).

```
Read 1:              ACGCTCCAACCGCTAATACG
Read 2:                       ATCGCTAATCCACGCCCGCCCCGC
Read 3:           AAAC-CTCCAACCG
Read 4:                                TGCGCGCCCGCCCCGAAACCGC
Consensus:        AAAC-CTCCAACCGCTAATGCGCGCCCGCCCCGAAACCGC
```

10.42 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.

10.43 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.