

CS262 Computational Genomics
Lecture 10: Fragment Assembly
Professor Serafim Batzoglou

Brief Review of DNA Sequencing from last lecture:

Terminology

insert - a fragment that was incorporated in a circular genome, and can be copied many times, or cloned

vector - the circular genome such as a plasmid or cosmid (host) that incorporated the fragment

BAC - Bacterial Artificial Chromosome, a type of insert–vector combination, typically of length 100-200 kb

read - a 500-900 long word that comes out of a sequencing machine

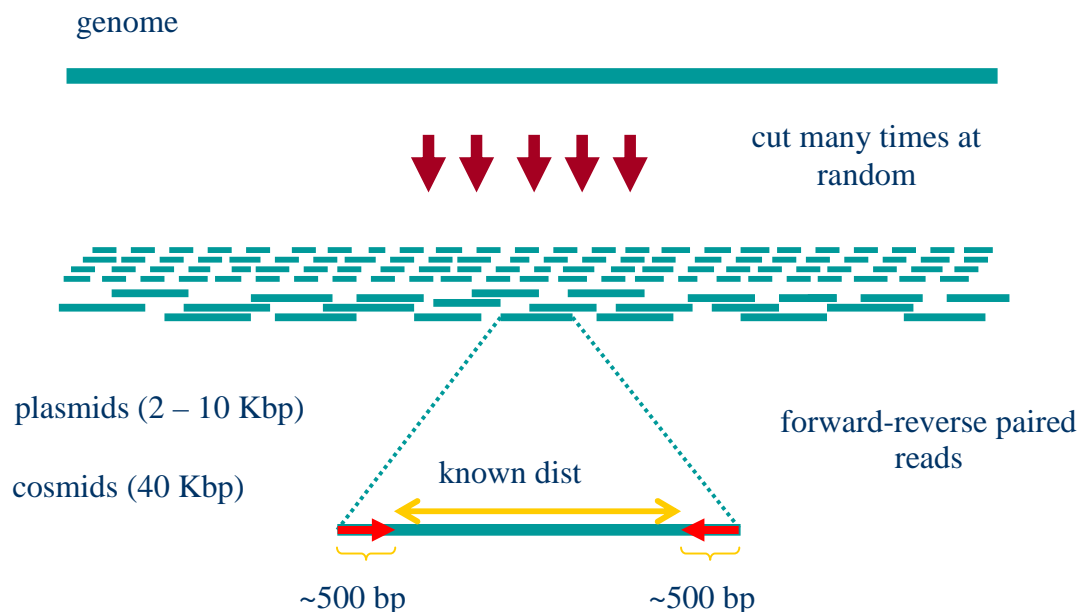
coverage - the average number of reads (or inserts) that cover a position in the target DNA piece

shotgun sequencing - the process of obtaining many reads from random locations in DNA, to detect overlaps and assemble

Whole Genome Shotgun Sequencing

We start out by breaking up the genome into many smaller sized pieces. To do so, starting from the genome, we shake it, obtaining many inserts of various sizes. These could be incorporated into plasmids or cosmids, or BACs (150kb).

We then typically sequence two reads from each insert, one from the left end, one from the right (double barreled sequencing).



Fragment Assembly

(in whole-genome shotgun sequencing)

We now move on to talk about what we do with the millions of reads that we've acquired in the previous step. What we want to do is to take these pieces and put them together in a coherent order to form the genome in question.

Task: We have millions of pieces to put together—pieces consisting of a pair of reads whose distance we know within $\pm 20\%$ standard deviation and whose sequence we know within 1% error rate. What we want to end up with is a sequence for the entire genome, so we need to reconstruct the genome from these fragments by putting them together, in order.

Since we have millions of these pieces, in order to perform the assembly efficiently, we need to find a linear-time algorithm!

1. Steps to Assemble a Genome (overview)

1. First we find pairs of reads with overlap
2. Choose some subset of those overlaps and merge the “good” pairs of reads into longer contigs (those from similar places in the genome)
3. Link these contigs to form supercontigs (You can put supercontigs together with mapping information to form ultracontigs)
4. Derive consensus sequence

1.1 More terminology

- **read** - a 500-900 long word that comes out of sequencer
- **mate pair** - a pair of reads from two ends of the same insert fragment (we know approx. distance)
- **contig** - a contiguous sequence formed by several overlapping reads with no gaps
- **supercontig** (scaffold) - an ordered and oriented set of contigs, usually by mate pairs
- **consensus sequence** - sequence derived from the multiple alignment of reads in a contig

Note: when reading from the genome, we don't know if it's from the forward or reverse complement of the DNA—so we treat every read as two equivalent words: both the forward and the reverse complement

2. Finding overlapping reads

The goal of the overlapping step is to find, among all the reads we have, all the pairs of reads that are actually overlapping with each other in the genome. In other words, two reads would be overlapping if they're offset from each other only a small amount, so that both reads contain a section of DNA that is common to both. Since we have millions of reads, we cannot afford to use an algorithm that runs in longer than linear time (in the number of reads we have), in order to be efficient. Earlier algorithms for fragment assembly, such as PHRAP, required comparing every pair of reads, which resulted in $O(N^2)$ time.

2.1 Basic Linear-Time algorithm:

1. Using a value of $k \sim 24$, list all the words of length k in all the reads. Each read yields $2 * \text{ReadLen} - (k + 1)$ words. Note: Due to memory limitations, this task must be done in several passes, each time only taking k -mers that start and end in particular sequences
2. The k -mers are sorted according to read, position, word, orientation
3. Sort the k -mers according to word (lexicographically), read, orientation, position. Sorting can be done in $O(N)$ time, using radix sort
4. Read all overlaps off this list; this can be done easily because all overlaps will be adjacent to each other (since the k -mers are sorted lexicographically!)
5. Extend each overlap to full alignment between its two source reads and throw the pair away if not 98% similar (this can be done very quickly)

One problem we encounter here is repeats. Basically, very common k -mers that occur many times in the genome can increase our running time to quadratic from linear, because k -mers that occur N times cause $O(N^2)$ overlaps or read/read comparisons. In addition, they do not offer very useful information, since for overlaps we're interested in overlaps with unique regions, not overlaps stemming from repeats.

The solution to this is to discard all k -mers that occur too frequently. This threshold is variable, as we have to set it to balance the sensitivity/speed tradeoff, according to the genome at hand and the computing resources available. What this means is that we will require that every overlap we do pay attention to include an uncommon k -mer, because otherwise we will not flag it as an overlap (since we've already discarded all the common ones).

2.2 Performing Error Correction

We start by creating local multiple alignments from the overlapping reads, as shown below:



insert A

replace T with C



On the left side, we can see quite clearly that the gap in the bottom sequence should be filled in with an A, according to the other reads, and the T in the middle sequence is a sequencing error that should be a C. We can fix these and move on. However, the correlated errors (on the right) are probably caused by two versions of the same repeat. The solution is to disentangle the overlaps into two sets of the same overlap.

This error correction process can remove up to 98% of the errors.

3. Merging reads into Contigs

Summarize the overlap info into an overlap graph by forming one node for every read, where:

Nodes: reads r_1, \dots, r_n

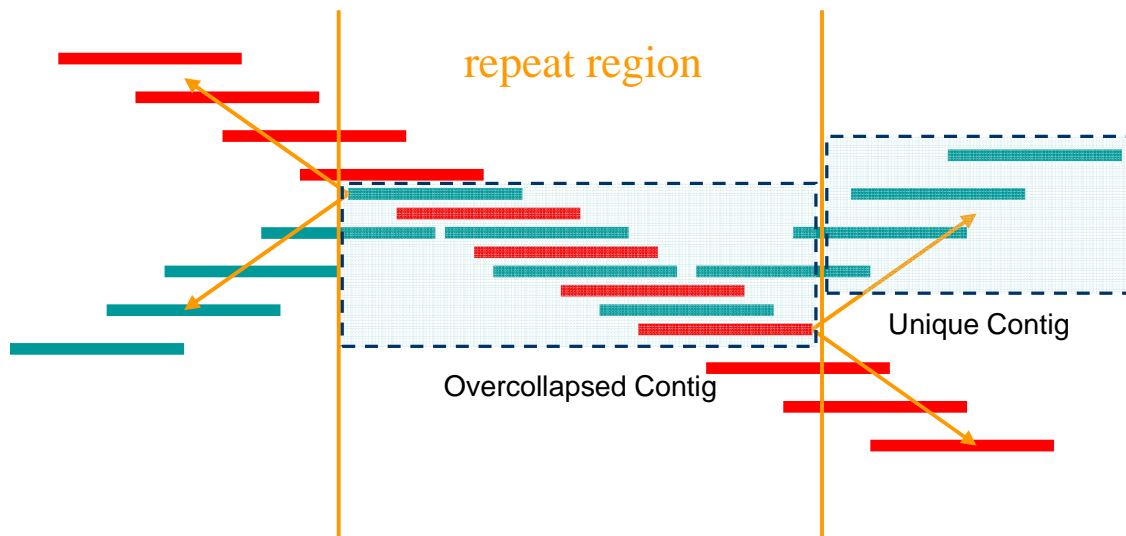
Edges: overlaps $(r_i, r_j, \text{shift}, \text{orientation}, \text{score})$

3.1 Problem: Repeat regions in the overlaps

In this example, we see a set of overlaps that appear to all be related. However, the blue reads and the red reads actually come from different regions of the genome (although we can't tell this!). They appear to all be related because they both have overlaps with the same repeat region.

The problem this causes is that we want all the contigs we form to actually exist in the target genome, but in this case, if we formed a contig that crosses the repeat boundaries, we would be forming a contig that does not really exist!

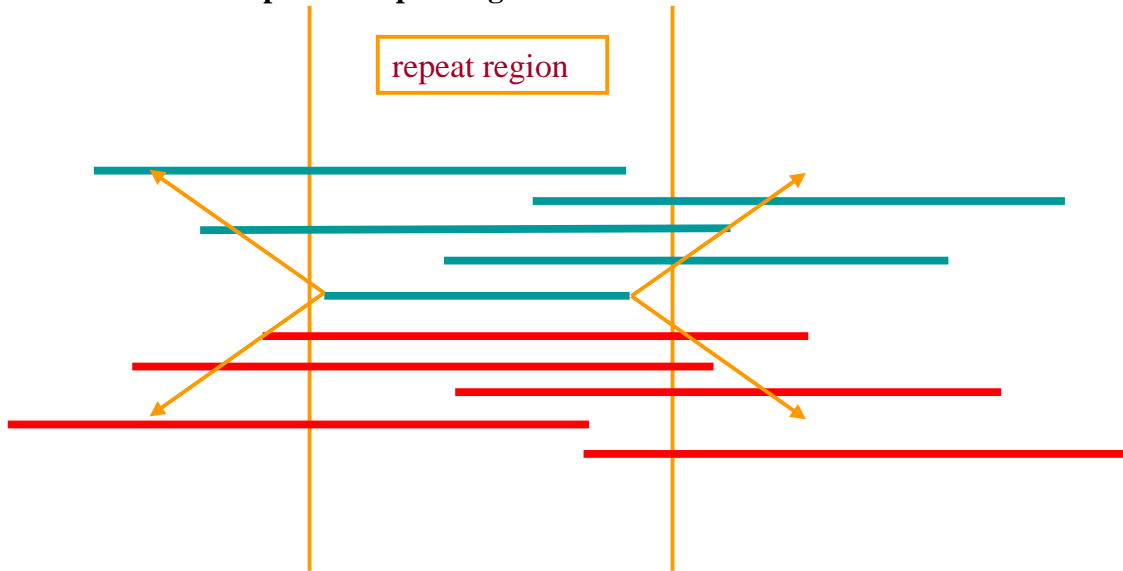
To avoid doing this in this case, we have to only look at reads up to potential repeat boundaries; here, we would form five contigs: four for the flanking regions, one for the repeat region.



3.2 How to find repeat boundaries?

Detecting repeat boundaries like this one relies upon recognizing a read that is overlapping with two other reads that are very dissimilar to each other (like the red read at the bottom right of the repeat region), which is overlapping both with the blue read above it and the red read below it; the non-overlapping regions in those two reads do not match at all.

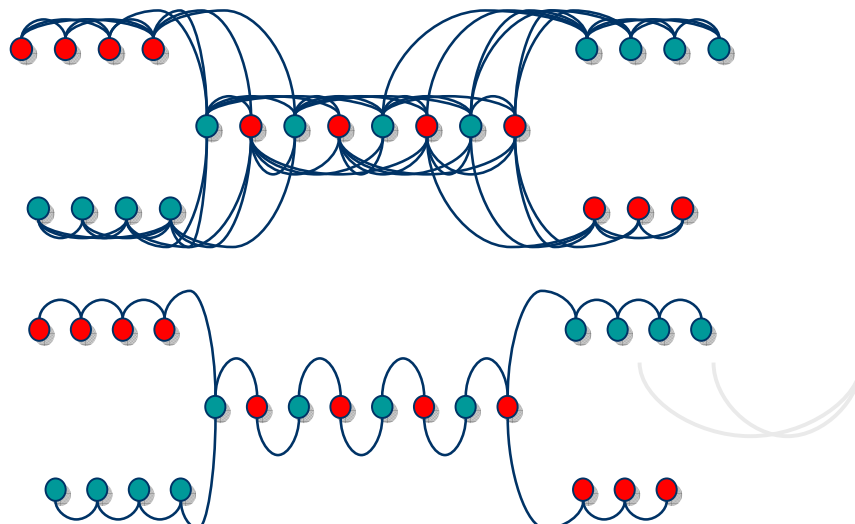
3.2.1 Another example of a repeat region:



Strategies in this case:

- Ignore non-maximal reads
- Merge only maximal reads into contigs

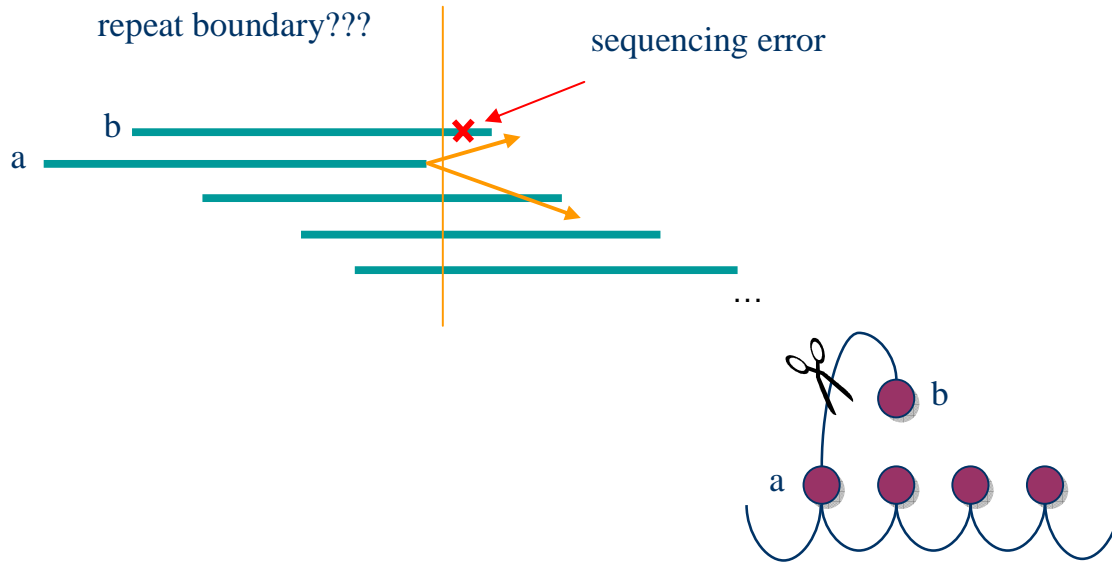
Given this overlap graph, we want to transform it so that repeat regions are much easier to detect. To do so, we remove transitively inferable overlaps-- If read r overlaps to the right reads r_1 , r_2 , and r_1 overlaps r_2 , then (r, r_2) can be inferred by (r, r_1) and (r_1, r_2)



Repeat boundaries are much more obvious in this graph.

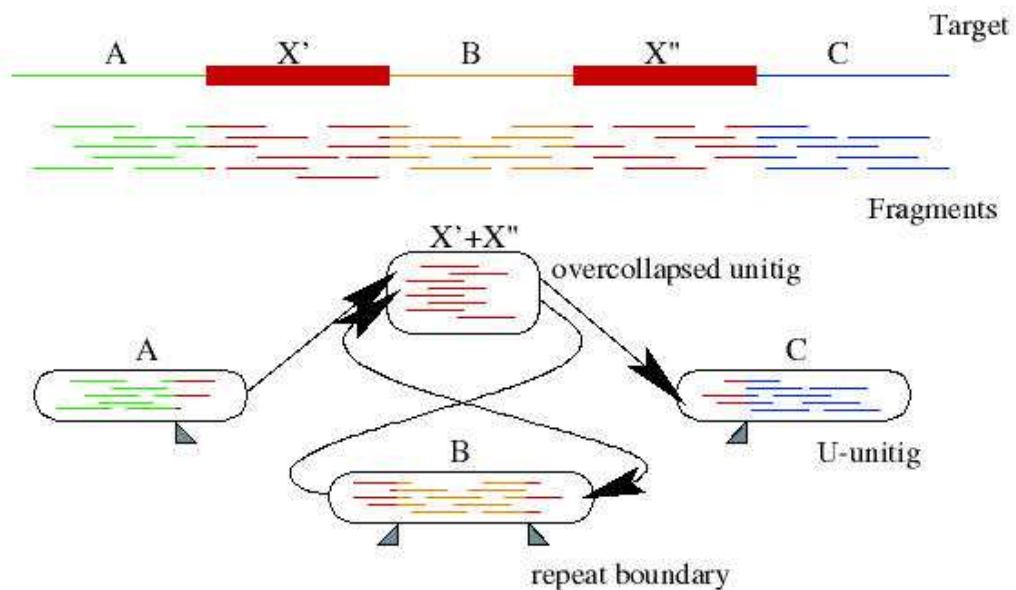
3.2 Recognizing non-repeat boundaries

We also have to be careful not to flag a repeat boundary where one doesn't exist. In this example, if we're not careful, we may see this as a repeat boundary, when in reality, we're dealing with a sequence error on read b that makes it look like it's overlapping with an unrelated region.



The solution is to ignore 'hanging' reads that we see here (the overlap with b) when detecting repeat boundaries, because it does not indicate a true repeat boundary.

This is the overlap graph that results after having formed contigs:
The end of A, beginning of C and start and end of B all overlap with the overcollapsed contig.



3.3 Repeats, Errors and Contig Lengths

It's easy for us to deal with repeats that are shorter than the read lengths, as these are not really repeats and will have overlaps that can be easily disambiguated, since reads that span across the repeat will reveal the order of the flanking regions.

Additionally, repeats that have more base pair differences than the sequencing error rate allows also do not cause problems, because in our overlap creation process, we will reject those overlaps that are from two different copies of the repeat, because they will have too many base pair differences.

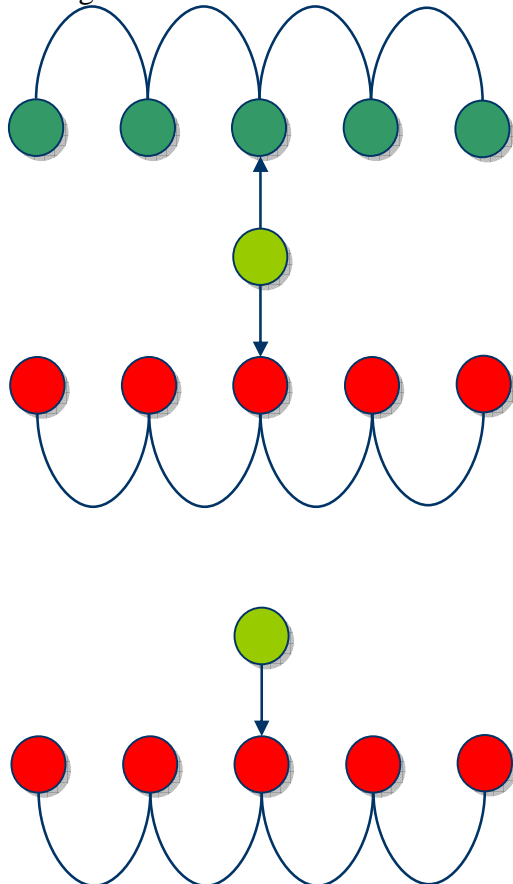
Consequently, if we want to reduce the occurrence of repeats for our assembly process, we can either increase the read length or decrease the sequencing error rate.

Role of error correction:

- Discards up to 98% of single-letter sequencing errors
- decreases error rate
- ⇒ decreases effective repeat content
- ⇒ increases contig length

We insert non-maximal reads whenever we can (whenever it's part of an unambiguous structure)

Ambiguous:

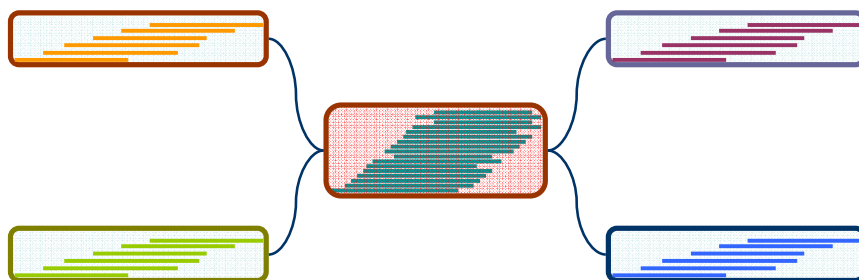


Unambiguous

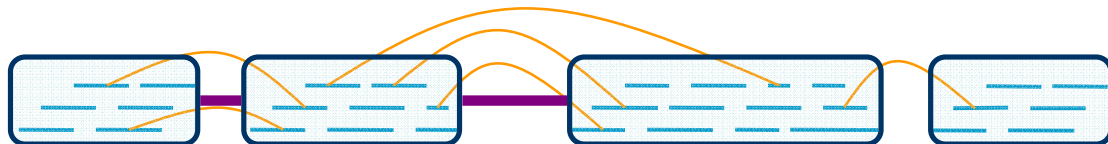
4. Link Contigs into Supercontigs

To differentiate between overcollapsed contigs and unique contigs, we can look at the density of the reads—overcollapsed reads will have a high density, because the number of reads overlapping with them should be proportional to the number of time the region repeats in the genome..

We can also look at the overlap graph to find inconsistent links, where a contig links to two or more contigs on the left or on the right that do not overlap with each other:

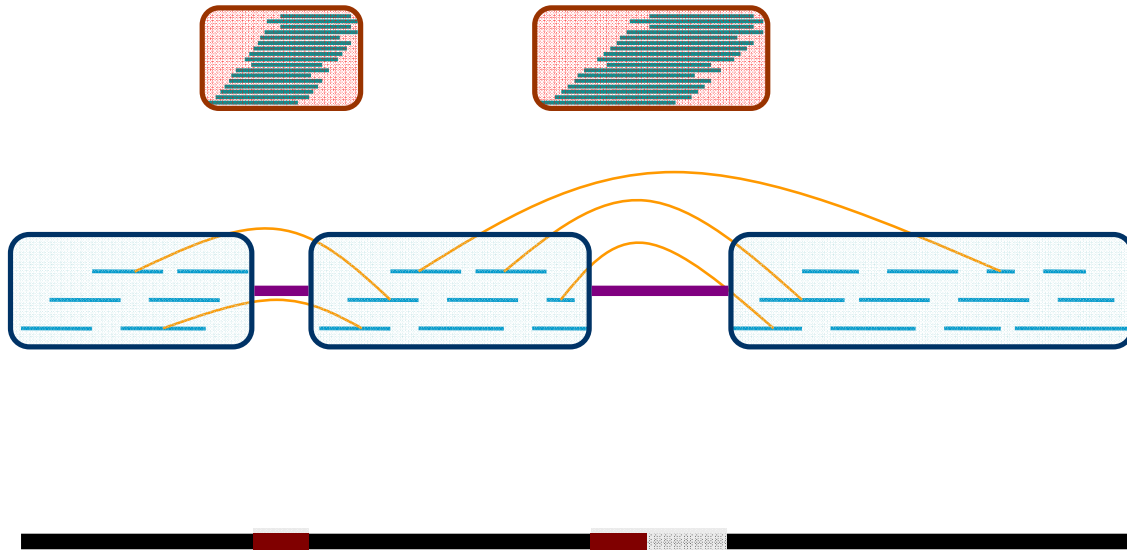


We find all links that exist between unique contigs and connect the contigs incrementally, if there are two or more links.



The missing regions here represent repeat regions that we're not considering yet.

After we've formed the basic structure, we can fill in the gaps in the supercontigs with paths of repeat contigs. This is typically the most difficult and error-prone step in the whole assembly process, as you have to deal with, for instance, palindromic repeats or tandem repetitions, where it's very difficult to avoid mistakes.



At this point we have an ordered list of reads along with precise shifts.

5. Derive Consensus Sequence

In linear time now, we can form multiple alignments from pairwise read alignments. When we encounter discrepancies in the multiple alignment, we derive each consensus base by weighted voting from the reads in the multiple alignment, or we choose to take the maximum-quality letter.

6. General Assembler Information

6.1 Some Assemblers

PHRAP

Early assembler, widely used, good model of read errors
 Overlap $O(n^2)$ → layout (no mate pairs) → consensus

Celera

First assembler to handle large genomes (fly, human, mouse)
 Overlap → layout → consensus
 Made huge impact, helped lead to human genome being sequenced 5 yrs ahead of time

Arachne

Public assembler (mouse, several fungi)
 Overlap → layout → consensus

Phusion

Overlap → clustering → PHRAP → assemblage → consensus

Euler

Indexing → Euler graph → layout by picking paths → consensus

6.2 Terminology:

N50 contig length - If we sort contigs from largest to smallest, and start covering the genome in that order, N50 is the length of the contig that just covers the 50th percentile. You would like N50 length > gene length

Assembler quality - generally determined by how much of the genome is covered (how long the contigs are)

6.3 History of WGA

1982: lambda-virus: 48,502bp

1995: h-influenzae: 1Mbp

2000: fly: 100Mbp

2001-present: human (3Gbp), mouse (2.5Gbp), rat*, chicken, dog, chimpanzee, several fungal genomes

Now: whole genome shotgun assembly is used as the standard

Genomes Sequenced: <http://www.genome.gov/10002154>

7. Molecular Inversion Probes

Molecular inversion probes are a new sequencing technology that should be emerging more in the next 2-3 years.

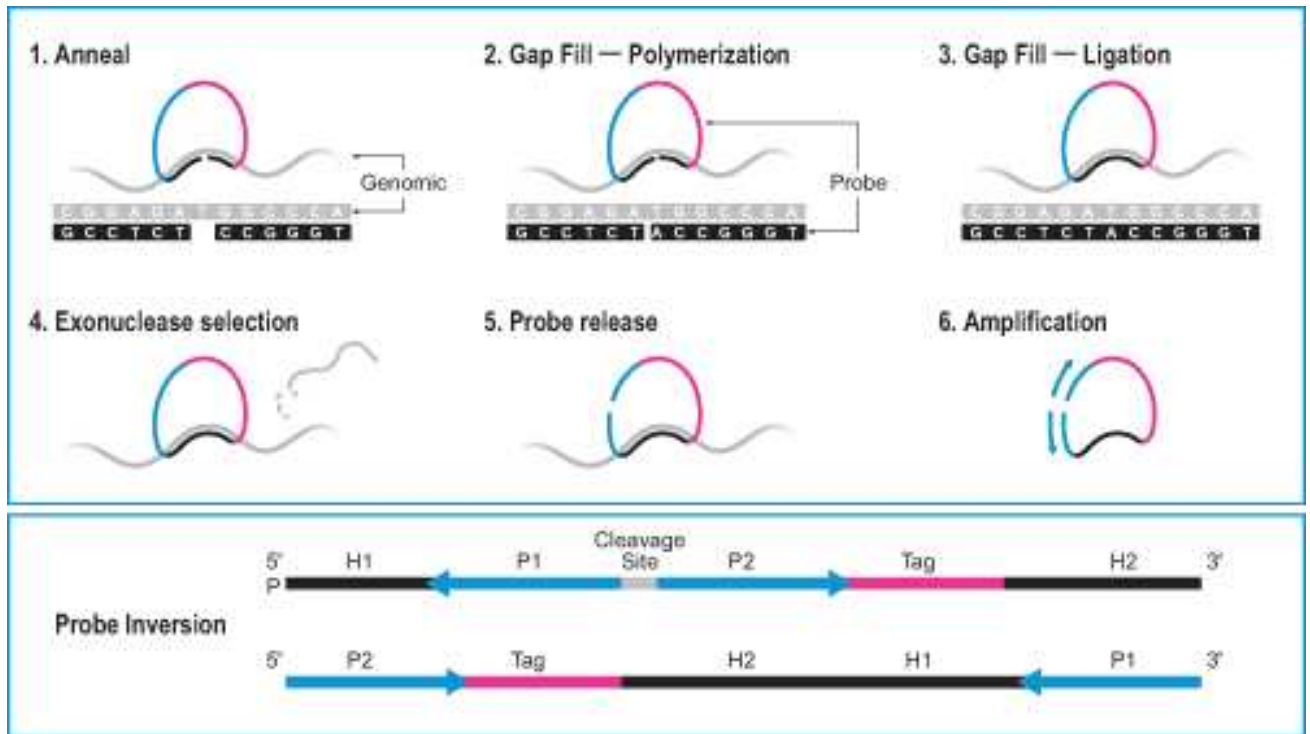
Since we know that each human genome have 1-2 million differences, the goal of these probes is to actually find these differences and see how they affect various phenotypic characteristics (i.e. diseases, lifespan, etc.)

7.1 Requirements:

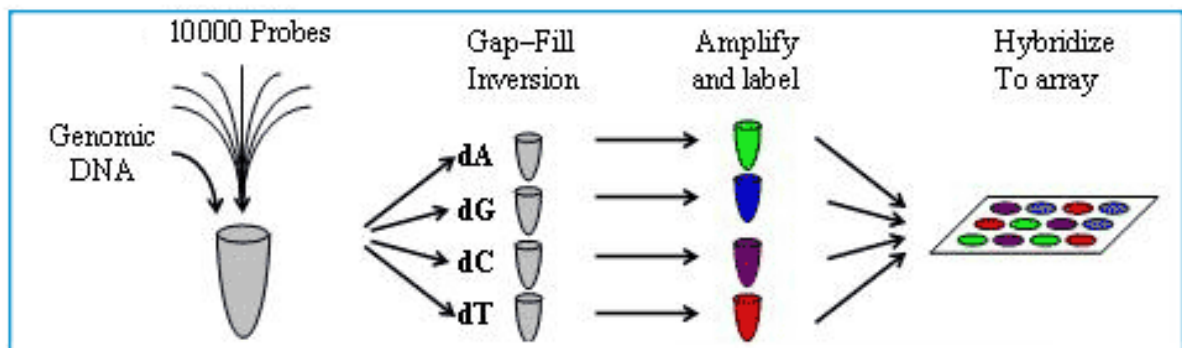
- A map of the genome with locations of where differences occur
- Find the specific alleles that we have that are different

7.2 Procedure

probes - linear pieces of DNA which go and attach in a DNA sequence at a specific location.



1. We ‘program’ a probe to a location in the genome by knowing the sequence that usually surrounds the base pair that frequently differs. Looking at step [1] in the figure above, we use the information about the surrounding sequence to construct the black portion of the probe, leaving a single gap for the nucleotide that frequently differs
2. Then we fill the gap by sending the solution of a particular nucleotide; if that nucleotide is complementary, then it will fill the gap. [2]
3. You then amplify it, and the only probe that should multiply many many times are those whose gap is filled.



4. You do this simultaneously for many locations and use a different colored solution for each nucleotide. In the end, you send the results to a chip where you can read off, according to color, what the differences in the genome were found.

