Genome Assembly

J. Christian Somody

6 December 2011

BCB410

Sanger Sequencing

DNA Polymerase reads the template strand and synthesizes a new second strand to match:

IF 5% of the T nucleotides are actually dideoxy T, then each strand will terminate when it gets a ddT on its growing end:

5' - TACGCGGTAACGGTATGTTCGACCGTTTAGCTACCGAT•

5' - TACGCGGTAACGGTATGTTCGACCGTTTAGCT•

5' - TACGCGGTAACGGTATGTTCGACCGTTT•

5' - TACGCGGTAACGGTATGTTCGACCGTT•

5' - TACGCGGTAACGGTATGTTCGACCGT•

5' - TACGCGGTAACGGTATGTT•

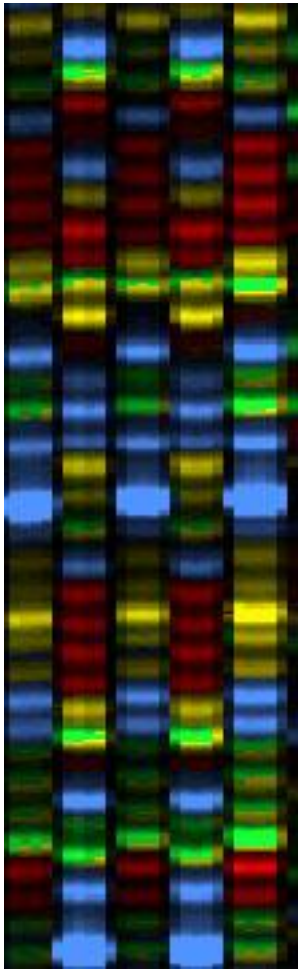
5' - TACGCGGTAACGGTATGT•

5' - TACGCGGTAACGGTAT•

5' - TACGCGGTAACGGT•

5' - TACGCGGT•

Sanger Sequencing

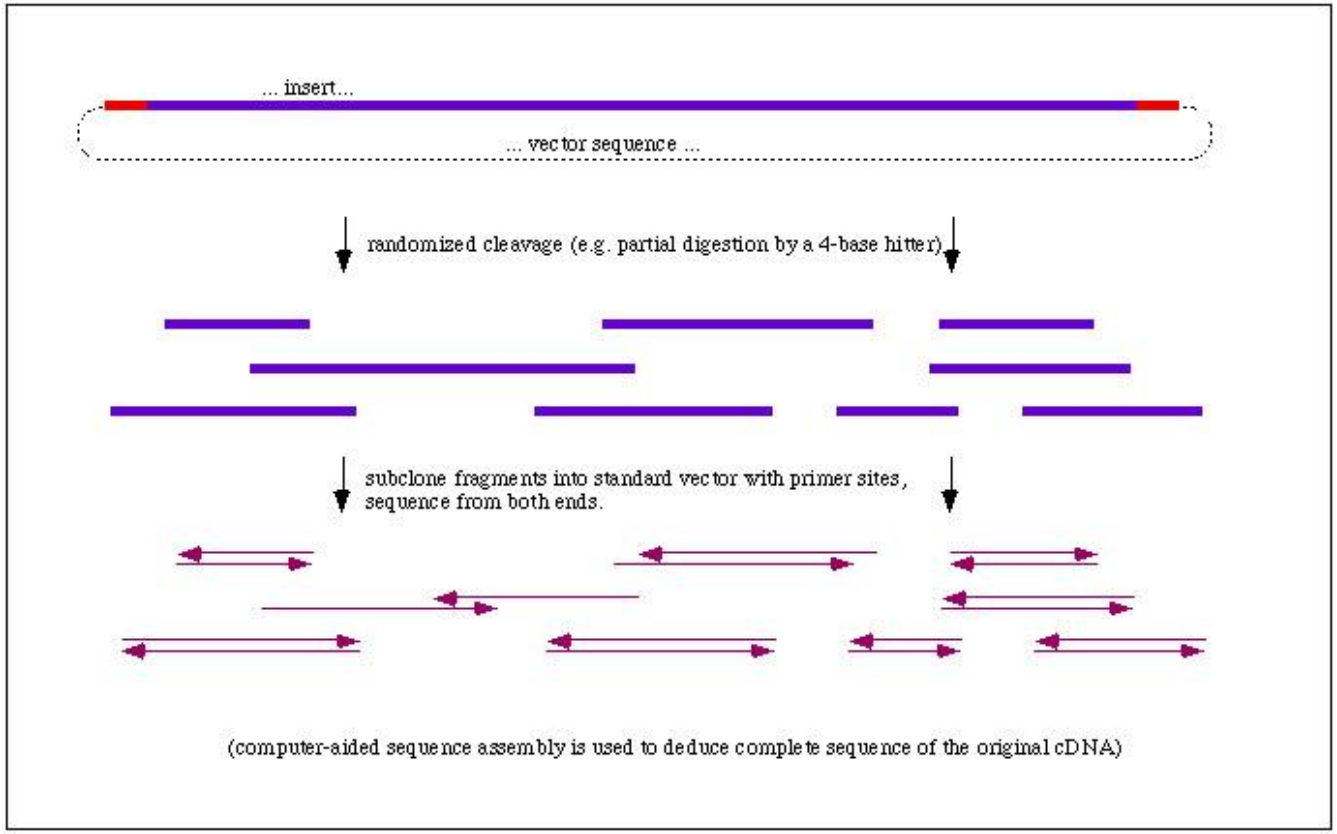


Next, run a gel of the terminated fragments, differently coloured depending on the final nucleotide.

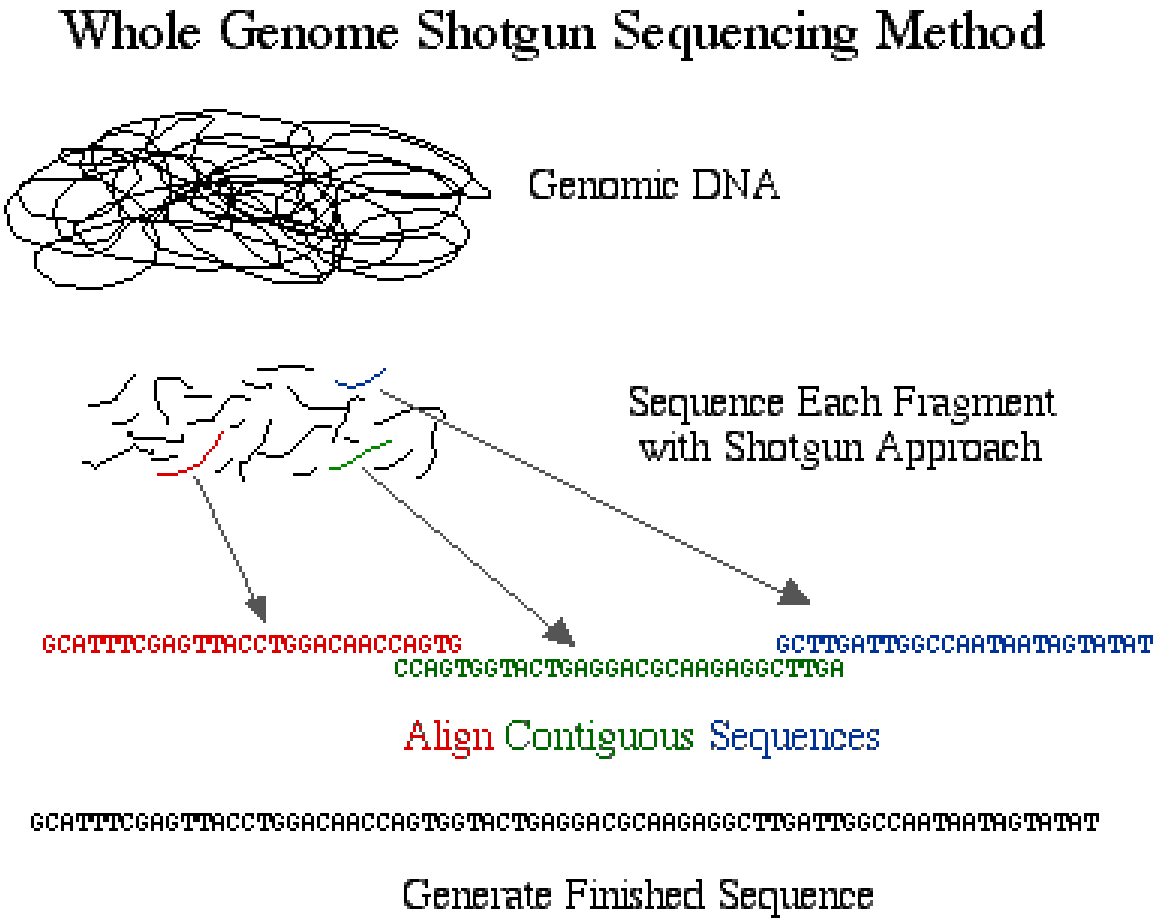
Read the sequence from the gel from the bottom to the top.

Shotgun Sequencing

- Break DNA into bacterial artificial chromosomes (BACs).
- Map the BACs to the genome and obtain a tiling path.
- Apply the shotgun method to each BAC.



Shotgun Sequencing



Sequencing

- Since genomes cannot currently be sequenced in one run, they must be sequenced as fragments and reassembled.
- Now that sequencing is becoming fast and accurate, how do we assemble the reads as quickly?
- Algorithms needed to do this.

Mapping-Based Assembly

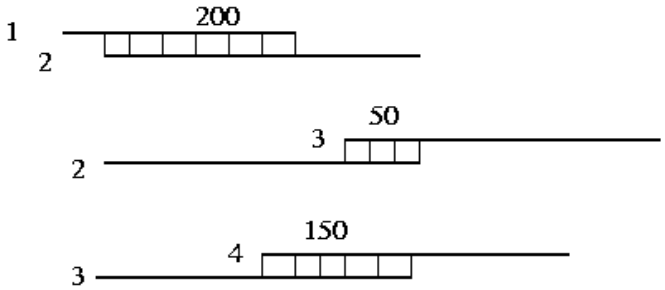
- One of the prereadings gave an overview of mapping ("comparative") genome assembly.
- We will focus on *de novo* assembly in this lecture.

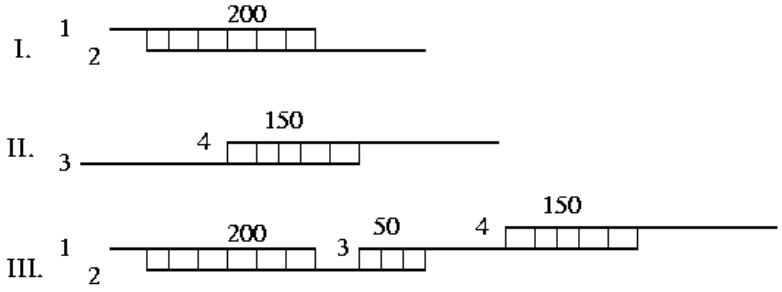
Algorithms

- **Shotgun sequencing assembly problem**
  - Find the shortest common superstring of a set of sequences.
  - Given *n* strings $\{s_1, s_2, \ldots, s_n\}$ find the shortest string T such that every s_i is a substring of T.
  - This is NP-hard, problem becomes unsolvable when implemented with many reads.
  - Approximation algorithm for this: the greedy algorithm.
    - Perform pairwise comparisons on the reads.
    - Pick the highest scoring pair and merge.
    - Repeat until no more merges can be done

Algorithms

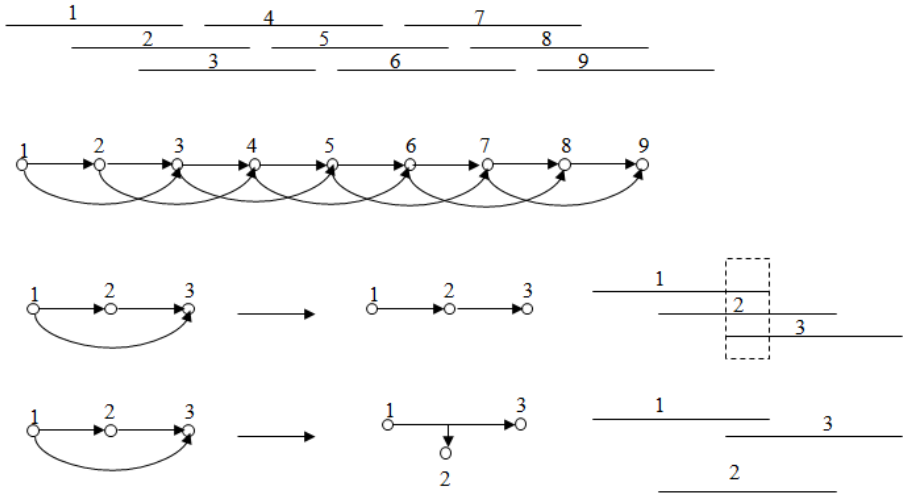
- **Shotgun sequencing assembly problem**
  - Greedy algorithms were the first successful assembly algorithm implemented.
  - Because of the greedy algorithm's limitations, two other algorithms were derived.





Algorithms

- **Overlap-layout-consensus**
  - Algorithm based on graph theory
  - A graph is constructed:
    - **nodes are reads, edges represent overlapping reads**
  - A contig (contiguous sequence) is a simple path (node visited at most once) in the graph



Algorithms

- **Overlap-layout-consensus**
  - An assembler builds the graph
  - Output is a set of nonintersecting simple paths, each path being a contig.
  - Ideally looking for a traversal of the graph (visits all nodes exactly once), a Hamiltonian path (also NP-hard).
  - We'll come back to a simplification of this.

Repeats

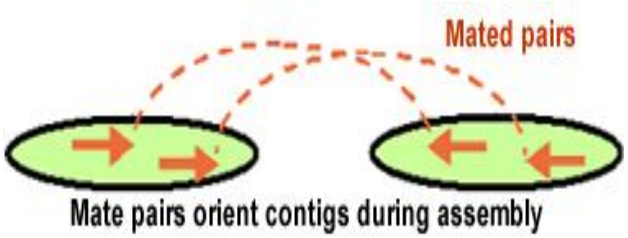
- There are often repeats in the sequence. Assembly algorithms should detect these during assembly, not after, to reduce incorrect reconstruction.
  - avoid "over-collapsing" repeats
- Repeats can be detected statistically or using algorithms.

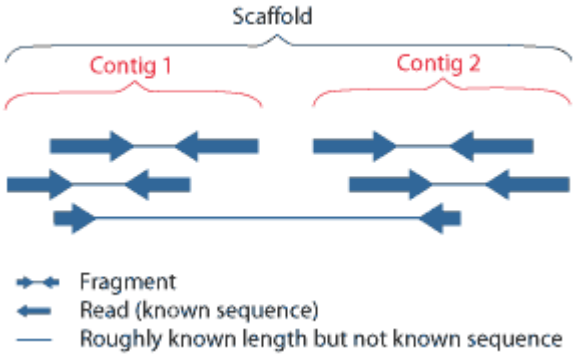
Repeats

- Statistical detection
  - Under the assumption that the genome is sampled equally at random, we see that if a certain sequence comes up more than others, it is likely part of repeat sequence.
  - Not a very good method, samples are not uniformly distributed.
- Graph-based detection
  - Finds repeats in complex parts of the graph constructed during the assembly process.
  - Complex areas investigated and attempted to be resolved.

Scaffolding

- Scaffolding groups contigs into subsets with known order and orientation.
- Nodes are contigs.
- Directed edge is between two nodes when paired-end tags bridge the gap between them.





Scaffolding

- Three basic problems
  - Find all connected components
  - Find a consistent orientation for all nodes in the graph. Nodes have two types of edges
    - Same orientation
    - Different orientation
    - Consistent orientation possible only if all undirected cycles have an even number of reversal edges.
    - Optimization problem: find the smallest number of edges to be removed so that no cycle has an odd number of reversal edges
  - Fit the edges on a line so the least number of constraints is invalidated. (NP-complete)
- Analogy
  - Like taking a map, broken into pieces, and reassembling it using the cities spanning more than one piece to help determine boundaries.

More Assembly Methods

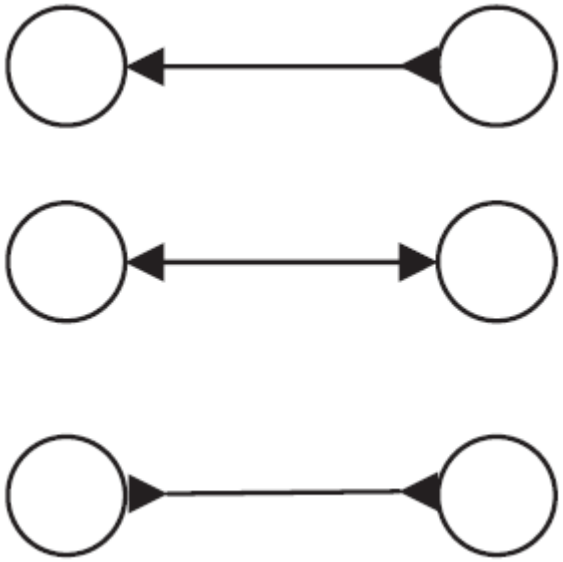
- many graph-based methods exist:
  - EULER assembler: previously discussed
  - string graph: same graph, remove redundant edges, establish edge constraints (must balance flux at each node), find shortest walk… a very complicated algorithm, fails when a repeat is longer than a read
  - all methods minimise size of genome, result in repeat "over-collapsing", assemblies can be improved

Maximum Likelihood Assembly

- Medvedev & Brudno changed the goal of assembly: don't want to minimize size of genome, but maximize its likelihood!
- Take advantage of the high coverage to estimate the copy number of each read
- Maximizing the likelihood can be considered a "mininum-cost bidirected flow" problem

Bidirected Overlap Graph

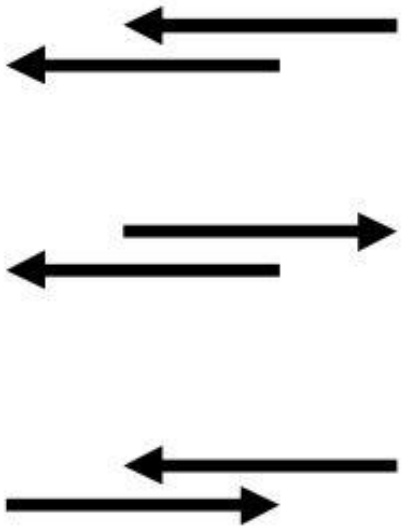
- Edges in bidirected graphs have two orientations: one at each end
- Therefore, three types of edges:



- ***For a walk in a bidirected graph, for each vertex on that walk, the orientation of the edge entering the vertex must be opposite that of the edge leaving the vertex.

Bidirected Overlap Graph

- Each vertex is a double-stranded read, edges represent read overlaps
- Three possible ways that two double-stranded reads can overlap (corresponds to the three types of edges)
  - Given two reads, each read can be oriented to the left or to the right , giving three possible overlaps:



Bidirected Overlap Graph

- A walk along this graph that visits every vertex at least once produces the original double-stranded genome (under the assumptions that the whole genome was covered by the reads, and that the reads are error-free)
- Overlap graph is constructed by placing an edge between two reads if they overlap by a minimum number of characters
- Then perform transitive edge reduction: remove overlaps covered by two shorter overlaps

Chinese Postman Problem

- In a weighted graph, a "tour" is a walk that traverses every edge at least once. A "circuit" is a cyclical tour.
- Chinese postman problem is to find a minimum weight circuit.
- Eulerian circuit is a circuit traversing each edge exactly once (not always possible).

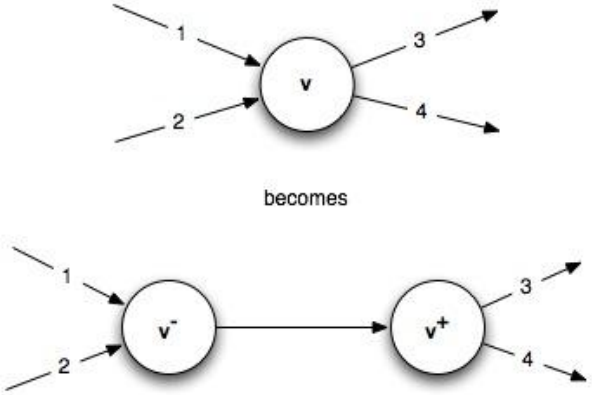
Minimum-Cost Biflow Problem

- Set upper (u) and lower (l) flow bounds on each edge, cost (c) for each edge
- Flow function *f* must obey the constraint for each edge e $l(e) \leq f(e) \leq u(e)$
- For each vertex, the incoming flow is balanced with the outgoing flow
- Objective: Find the flow that minimizes

$$\sum c_e f(e)$$

Adjusted Minimum-Cost Biflow Problem

- Upper and lower flow bounds on vertices as well
- Accomplished by splitting every vertex v into two: v^+ and v^-
- v^- serves as the "incoming" vertex, and inherits v's incoming edges
- v^+ serves as the "outgoing" vertex, and inherits v's outgoing edges
- Finally add one edge between v^- and v^+ and assign it the upper and lower flow bounds for v



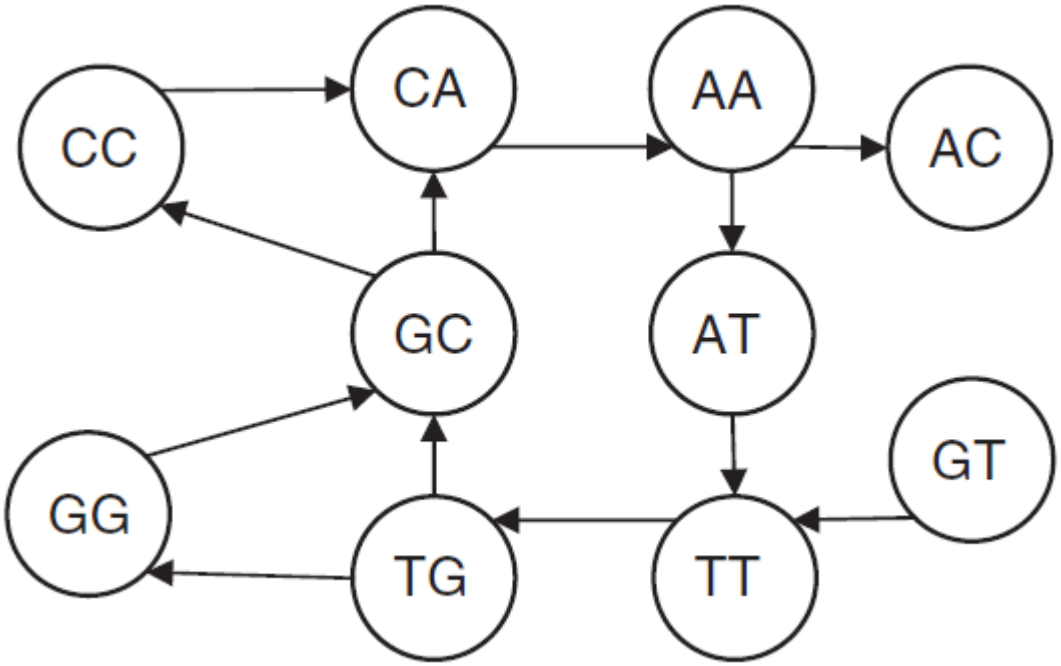
Build Graph Given Spectrum

- Nodes are all (*k*-1)-molecule sequences present in *k*-molecule-spectrum (two per *k*-molecule).
- Directed edges connect 5'-heavy (*k*-1)-mer to 3'-heavy one in positive strands, opposite in negative strands.
- Edges unweighted (all have weight of 1).

Demonstration

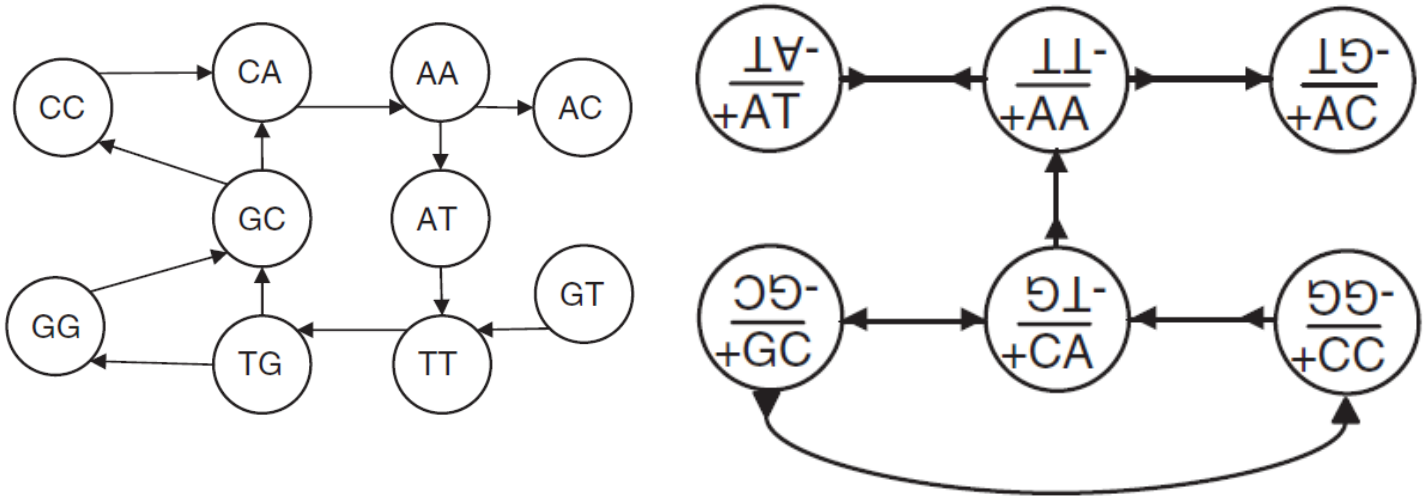
- How to build graph given *k*-molecule-spectrum.

{ATT/AAT, TGC/GCA, GCC/GGC, CCA/TGG, CAA/TTG, AAC/GTT}



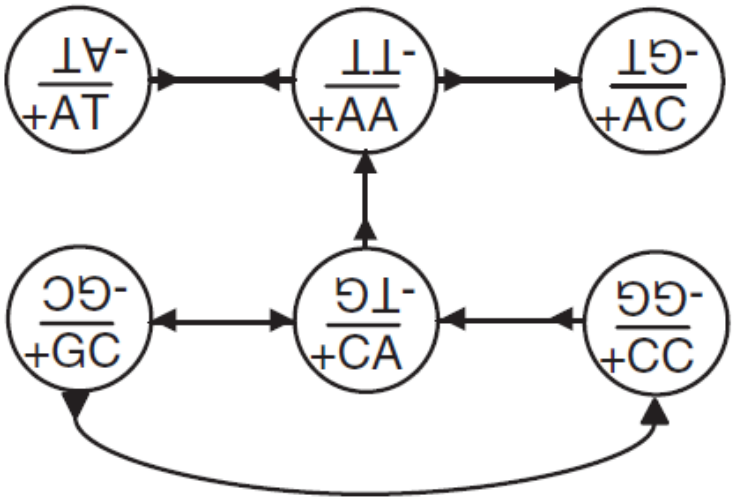
Demonstration

- Convert to bidirected de Bruijn graph:



Demonstration

- Now find circular walk in this graph (includes all edges)
- While walking, append each node minus the overlap to some string.



Direction of entry = which strand to read

ATTGCCAAC

Reverse walk gives reverse complement!

Conversion to Algorithm

- This procedure is trivial to do manually for small graphs, but imagine a graph of tens of thousands of 50-mers, for example. An efficient algorithm is needed.
- We already have algorithms for shortest path, but it needs to be modified for this special case.

Adjustments

- Supersource and supersink added to convert flow problem into circulation problem
- Each vertex has a lower bound of 1, since each read must appear in the finished genome at least once
- Edge bounds are set to 0 (lower bound) and infinity (upper bound)
- Put prohibitively large cost on the edge leading from the supersource and the edge leading to the supersink to ensure that the assembly uses the smallest number of contigs possible
- Flow through each vertex represents number of times it appears in the assembled genome

Methods: Maximizing the Global Read-Count Likelihood

- Start with the probability of a *k*-mer *i* being sampled a certain number of times from a genome *G*
- Let *N*(*G*) be the length of the genome assembly of *G*, and let g_i be the frequency of *i* in *G*
- Under the assumption of uniform sampling, the probability of sampling *i* is $g_i/N(G)$
- Let X_i be the random variable that represents the number of trials whose outcome is *i*
- Each random variable for every possible *k*-mer has a binomial distribution. Their joint distribution is the following multinomial distribution:

$$P\left[X_1 = x_1, X_2 = x_2, \ldots, X_{4^k} = x_{4^k}\right] = \frac{n!}{\prod x_i} \prod_i \left(\frac{g_i}{N(G)}\right)^{x_i}$$

Putting It Together

- Build the bidirected overlap graph.
- Perform transitive edge reduction (remove redundant overlaps).
- Add supersource and supersink and appropriate weightings to the graph.
- Solve biflow problem (using likelihoods).
  - very confusing portion of algorithm
- Solution gives collection of walks, representing contigs.
- Assemble contigs using paired-end read data.

Likelihood Algorithm

- From this, derive the **global read-count likelihood**, the likelihood of *k*-mer distributions (g_i) given the sampling outcomes (x_i):

$$L\left[g_1,\ldots,g_{4^k} \mid x_1,\ldots,x_{4^k}\right] = \frac{n!}{\prod x_i} \prod \left(\frac{g_i}{N(G)}\right)^{x_i}$$

- Goal is to maximize *L*, or, equivalently, minimize the negative log of *L*
- To translate this problem into a convex min-cost biflow problem, we need convex functions for each *k*-mer c_i s.t. $-\log L = \sum c_i(g_i)$
- Problem: the X_i random variables are not independent…

Likelihood Algorithm

- … unless the number of trials approaches infinity
- The number of trials is usually large enough to warrant the approximation of the multinomial distribution as the product of the binomial distributions for each X_i
- In this binomial approximation, genome length $N(G)$ is constant, and independent of the sampling frequencies
- Therefore, use N instead, which is the actual length of the genome G

Likelihood Algorithm

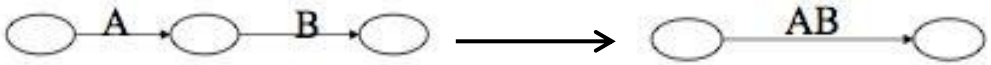
- New approximation of *L*:

$$L[g_1,\ldots,g_{4^k} \mid x_1,\ldots,x_{4^k}] \approx \prod P[X_i = x_i] = \prod \binom{n}{x_i} \left(\frac{g_i}{N}\right)^{x_i} \left(1-\frac{g_i}{N}\right)^{n-x_i}$$

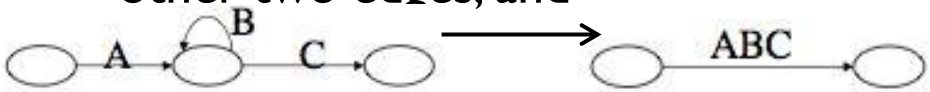
- Now $-\log L = K \sum c_i(g_i)$
- And $c_i(g_i) = -(x_i \log g_i) - (n - x_i)\log(N - g_i)$
- c_i is used as the convex functions for the vertices of the min-cost biflow graph described earlier

Convert Flow to Contigs

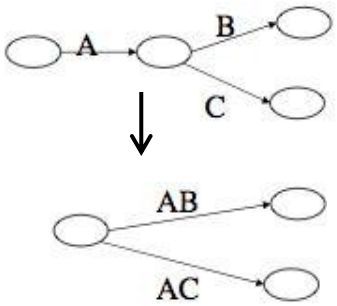
- Decompose flow into a collection of walks, which translates into assembled contigs
- Graph is first simplified by removing all edges with a flow of zero
- Additional simplifications possible by removing vertices *v* where:
  - There is exactly one edge going into *v* and one edge leading out of *v*, and the flow on both edges is the same



  - Vertices where there is also a loop with the same flow as the other two edges, and



  - Split and join vertices, where the flow on the in edges is the same as those of the out-edges



(Potentially Problematic) Assumptions

- **First major assumption:** Reads are error-free
  - can be overcome with higher coverage
- **Second major assumption:** Uniform sampling of all genomic regions
  - certain portions of the genome are actually easier to sample than others

Assembly Quality

- Assessing Assembly Quality
  - misassembly correction is expensive
  - some assemblers have a simple quality-control method that does not capture larger errors
  - test assembly software if we know a complete sequence (artificial or real)
  - measures of quality: number and sizes of contigs
    - under assumption that having few large contigs is better than many small ones.
    - only partially true, because there are less gaps in the former, but, does not account for the possibility of misassemblies

Questions?