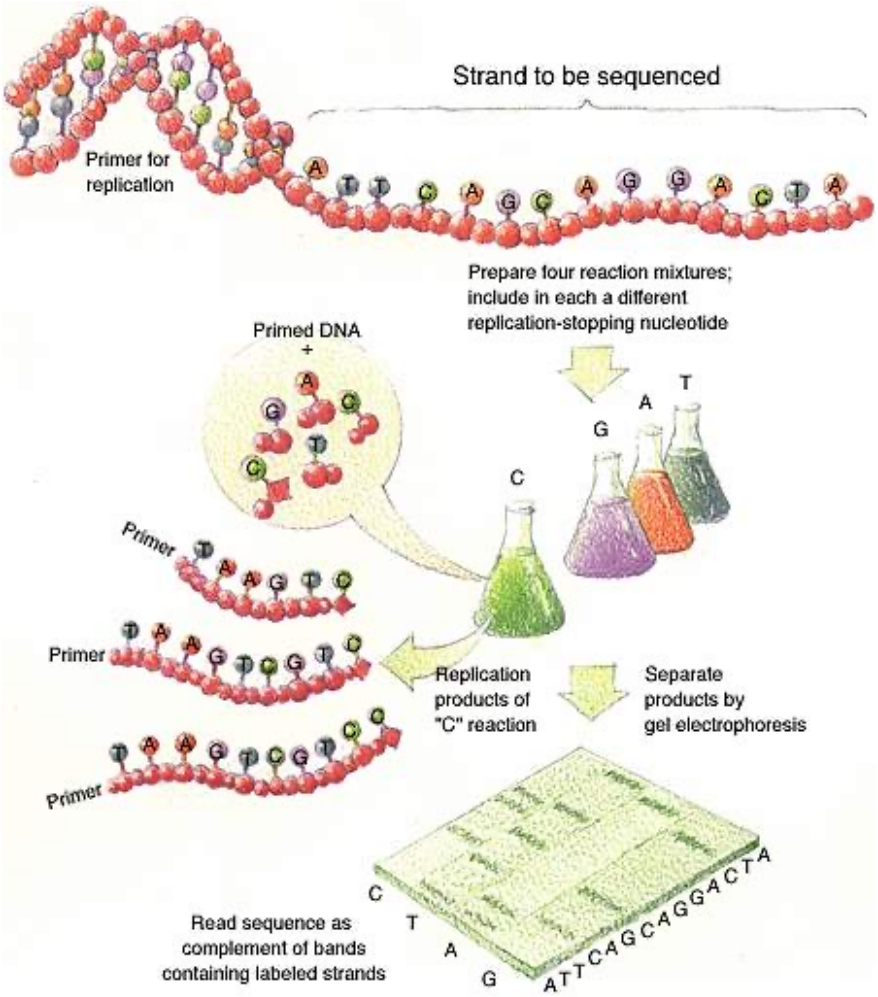
Eulerian graph

DNA Sequencing

- Shear DNA into millions of small fragments
- Read 500 – 700 nucleotides at a time from the small fragments (Sanger method)



Fragment Assembly

- **Computational Challenge:** assemble individual short fragments (reads) into a single genomic sequence (“superstring”)

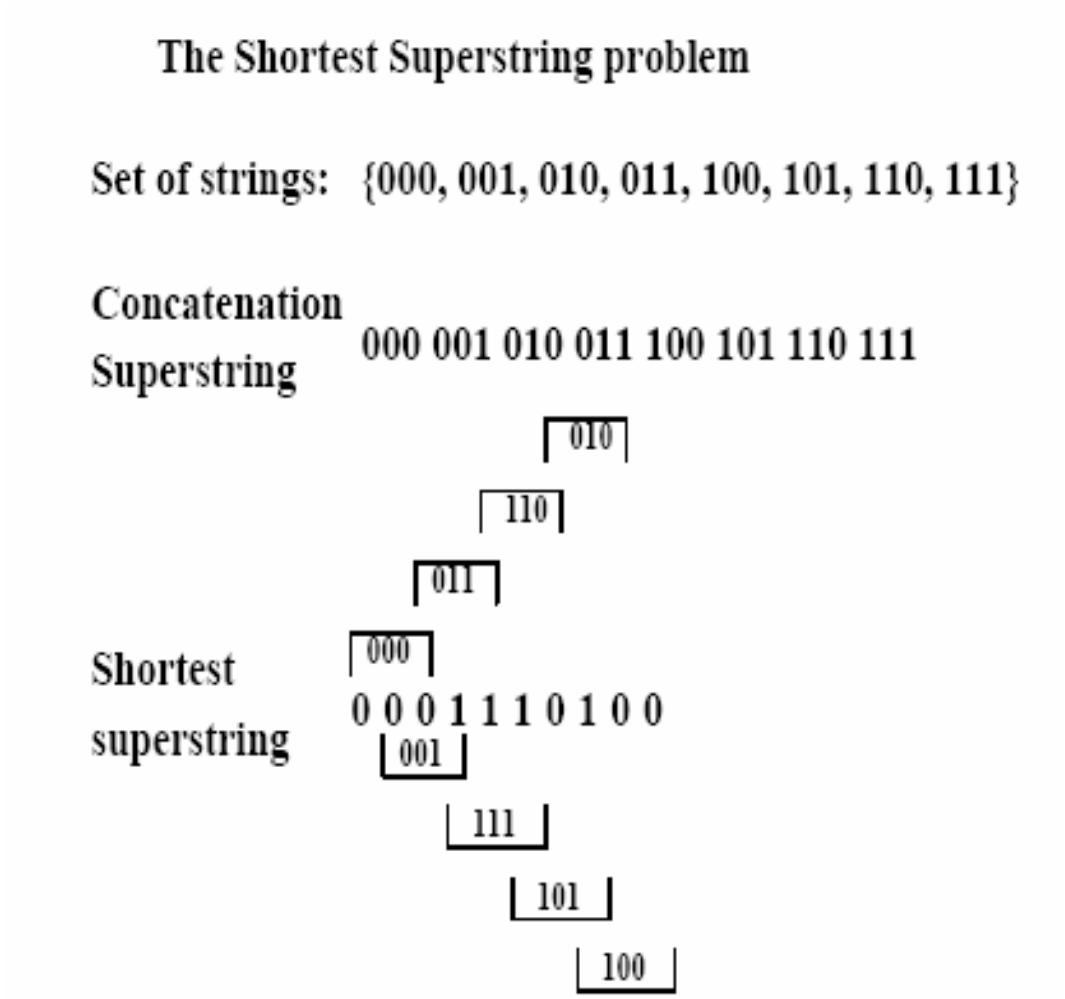
- Until late 1990s the shotgun fragment assembly of human genome was viewed as intractable problem

Shortest superstring problem

- Problem: Given a set of strings, find a shortest string that contains all of them
- Input: Strings $s_1, s_2, \ldots, s_n$
- Output: A string s that contains all strings $s_1, s_2, \ldots, s_n$ as substrings, such that the length of s is minimized

- **Complexity:** NP–hard
- **Note:** this formulation does not take into account sequencing errors

Shortest superstring problem: example

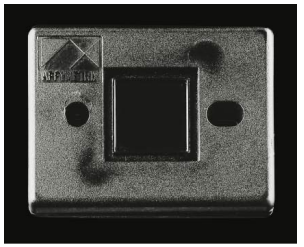


Sequencing by Hybridization (SBH): History

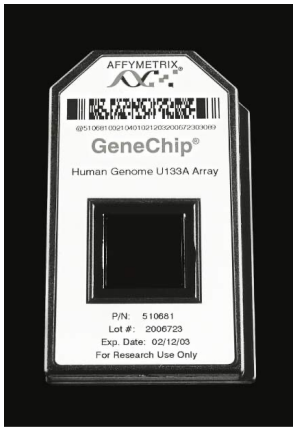
- **1988:** SBH is suggested as an an alternative sequencing method.
- **1991:** Light directed polymer synthesis developed by Steve Fodor and colleagues.
- **1994:** Affymetrix develops first 64-kb DNA microarray

*First microarray prototype **(1989)***

*First commercial DNA microarray prototype w/16,000 features **(1994)***



*500,000 features per chip **(2002)***



How SBH works

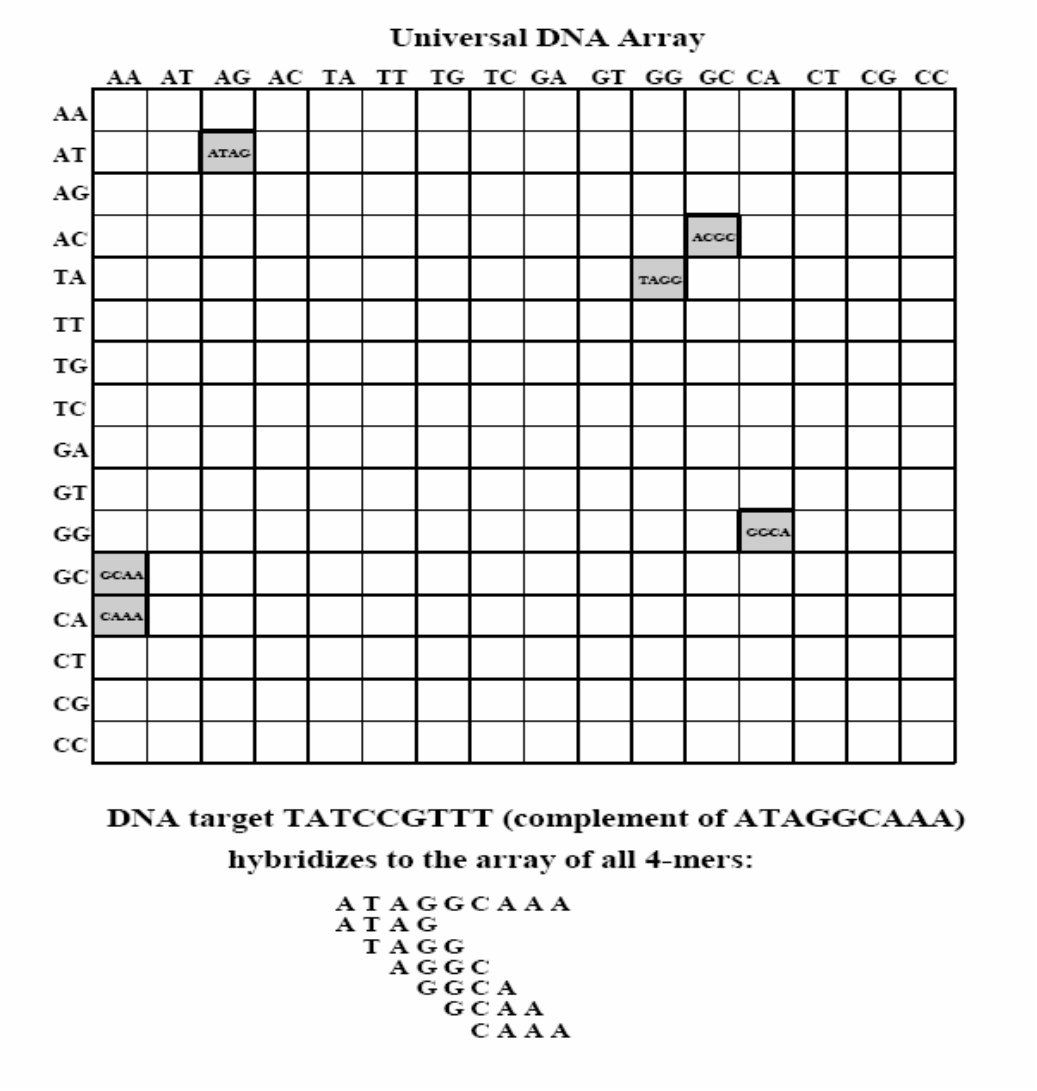
- Attach all possible DNA probes of length *l* to a flat surface, each probe at a distinct and known location. This set of probes is called the DNA array.
- Apply a solution containing fluorescently labeled DNA fragment to the array.
- The DNA fragment hybridizes with those probes that are complementary to substrings of length *l* of the fragment.

How SBH works

- Using a spectroscopic detector, determine which probes hybridize to the DNA fragment to obtain the *l*–mer composition of the target DNA fragment.

- Apply the combinatorial algorithm (below) to reconstruct the sequence of the target DNA fragment from the *l* – mer composition.

Hybridization on DNA Array



l-mer composition

- ***Spectrum (s, l)*** - *unordered* multiset of all possible *(n – l + 1)* *l*-mers in a string *s* of length *n*
- The order of individual elements in *Spectrum (s, l)* does not matter
- For *s* = TATGGTGC all of the following are equivalent representations of *Spectrum (s, 3):*

  {TAT, ATG, TGG, GGT, GTG, TGC}

  {ATG, GGT, GTG, TAT, TGC, TGG}

  {TGG, TGC, TAT, GTG, GGT, ATG}

l-mer composition

- ***Spectrum (s, l)*** - *unordered* multiset of all possible *(n – l + 1)* *l*-mers in a string *s* of length *n*
- The order of individual elements in *Spectrum (s, l)* does not matter
- For *s* = TATGGTGC all of the following are equivalent representations of *Spectrum (s, 3):*

  {TAT, ATG, TGG, GGT, GTG, TGC}

  {ATG, GGT, GTG, TAT, TGC, TGG}

  {TGG, TGC, TAT, GTG, GGT, ATG}
- We usually choose the lexicographically maximal representation as the canonical one.

Observations: different sequences may have the same spectrum

- Different sequences may have the same spectrum:

  Spectrum(GTATCT,2)=

  Spectrum(GTCTAT,2)=

  {AT, CT, GT, TA, TC}

The SBH problem

- Goal: Reconstruct a string from its *l*-mer composition

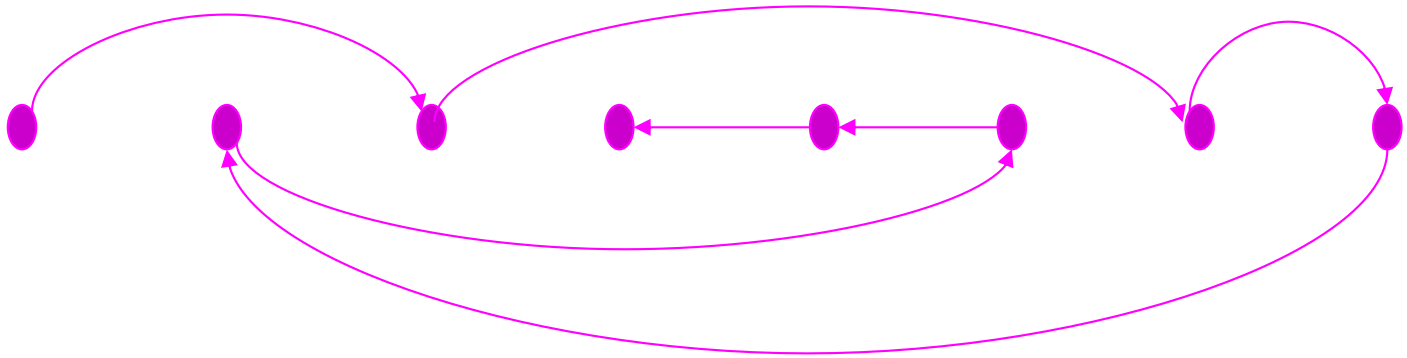
- Input: A set *S*, representing all *l*-mers from an (unknown) string *s*

- Output: String *s* such that *Spectrum (s,l)* = *S*

SBH: Hamiltonian path approach

S = { ATG AGG TGC TCC GTC GGT GCA CAG }

H ATG AGG TGC TCC GTC GGT GCA CAG



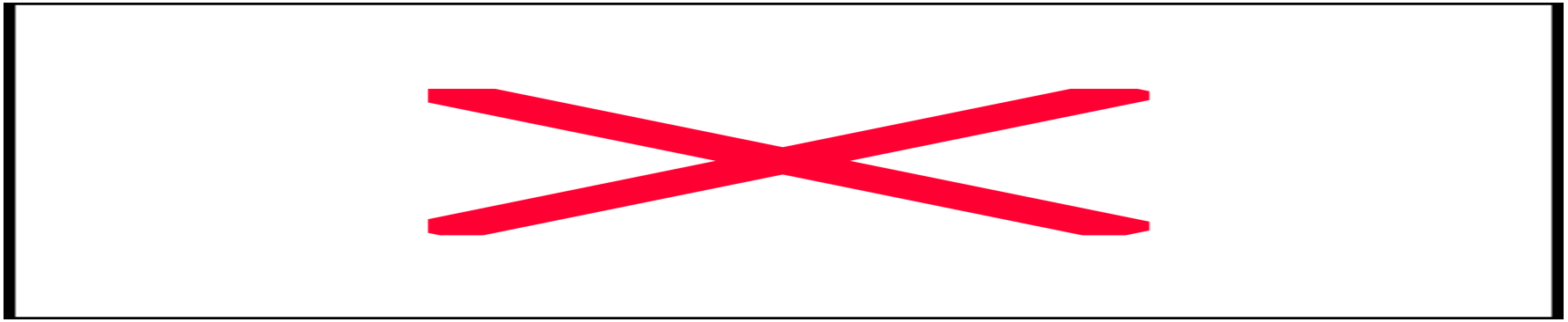
ATGCAGGTCC

Path visited every VERTEX once

SBH: Hamiltonian path approach

A more complicated graph:

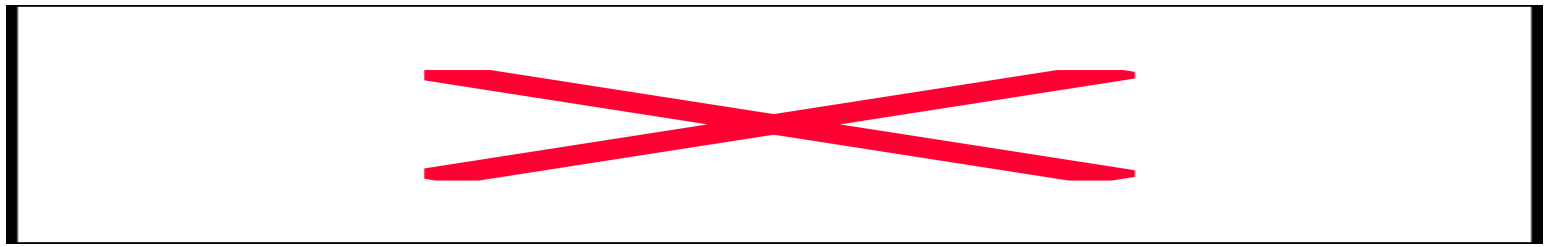
S = { ATG TGG TGC GTG GGC GCA GCG CGT }



SBH: Hamiltonian path approach

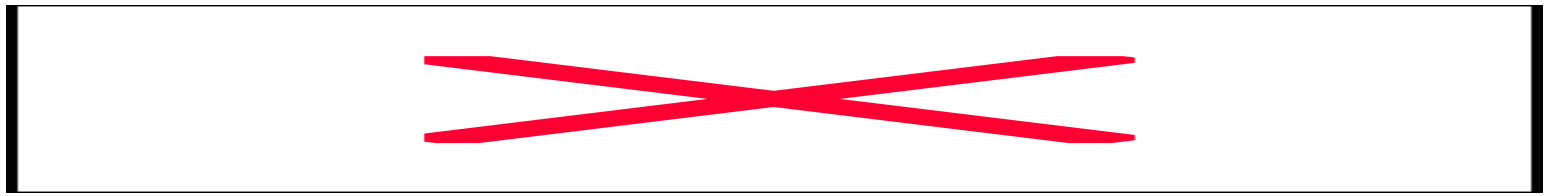
S = { ATG TGG TGC GTG GGC GCA GCG CGT }

Path



ATGCGTGGCA

Path 2:

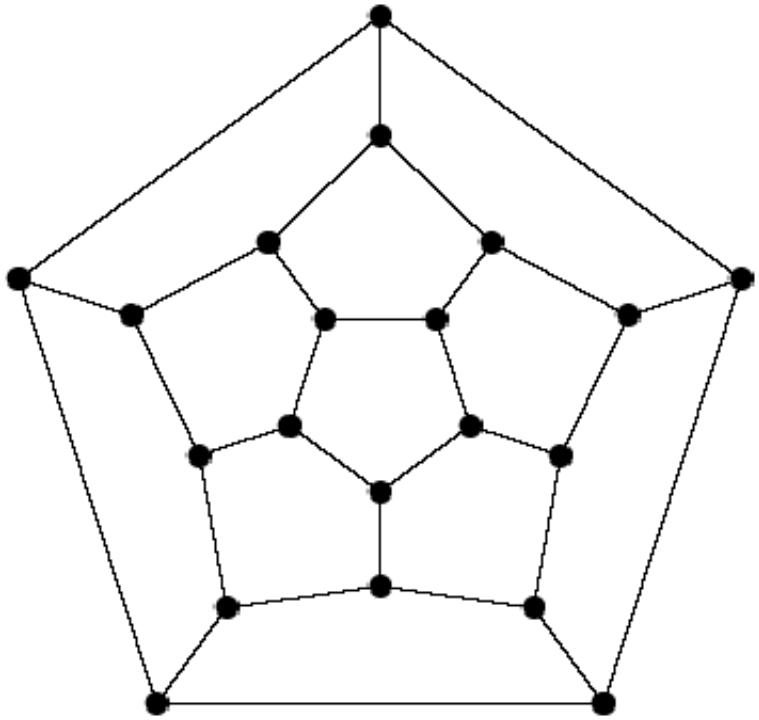


ATGGCGTGCA

Hamiltonian cycle problem

- Find a cycle that visits every ***vertex*** exactly once

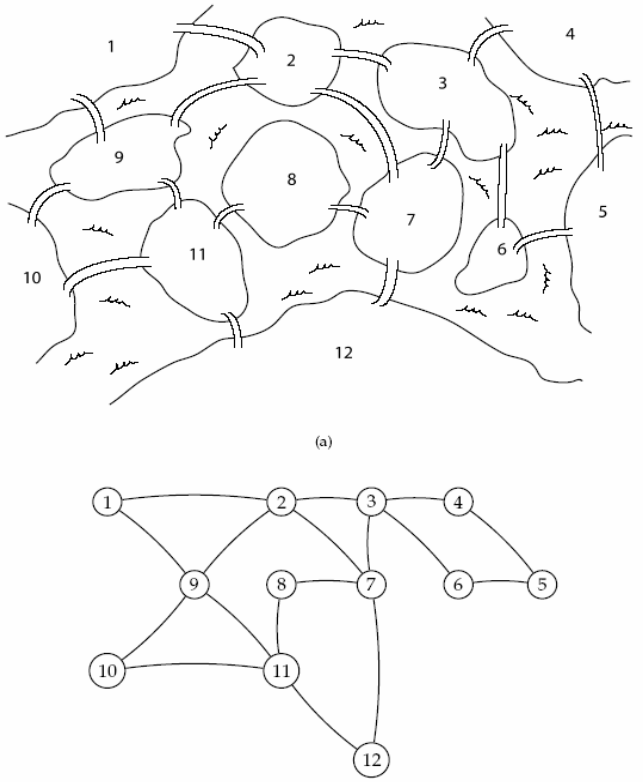
- NP–complete



Game invented by Sir William Hamilton in 1857

Eulerian cycle problem

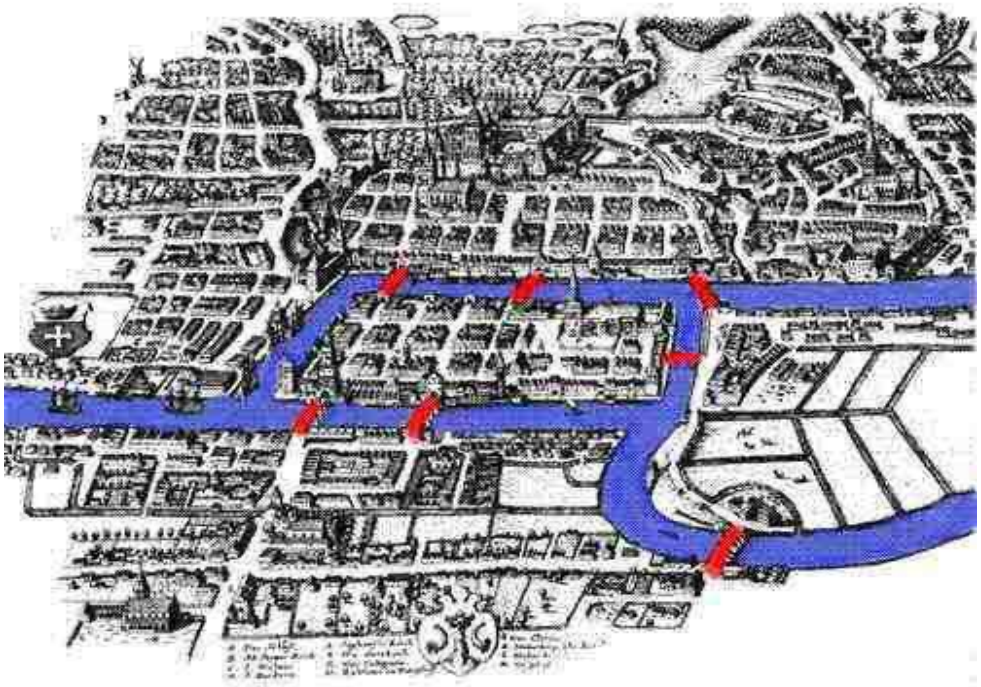
- Find a cycle that visits every ***edge*** exactly once
- Linear time



More complicated Königsberg

The Bridge Obsession Problem

Find a tour crossing every bridge just once
Leonhard Euler, 1735



Bridges of Königsberg

Euler Theorem

- A graph is balanced if for every vertex the number of incoming edges equals to the number of outgoing edges:

$$in(v)=out(v)$$

- **Theorem**: *A connected graph is Eulerian if and only if each of its vertices is balanced.*

Euler Theorem: proof

- Eulerian → balanced

  for every edge entering *v* (incoming edge) there exists an edge leaving *v* (outgoing edge). Therefore

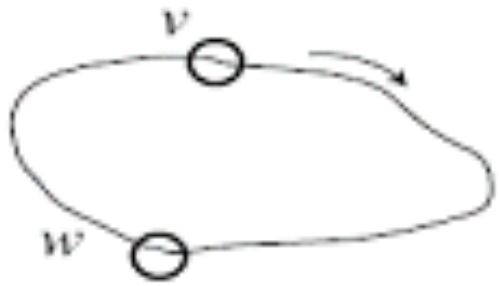
  $$in(v)=out(v)$$

- Balanced → Eulerian

  ???

Algorithm for Constructing an Eulerian Cycle

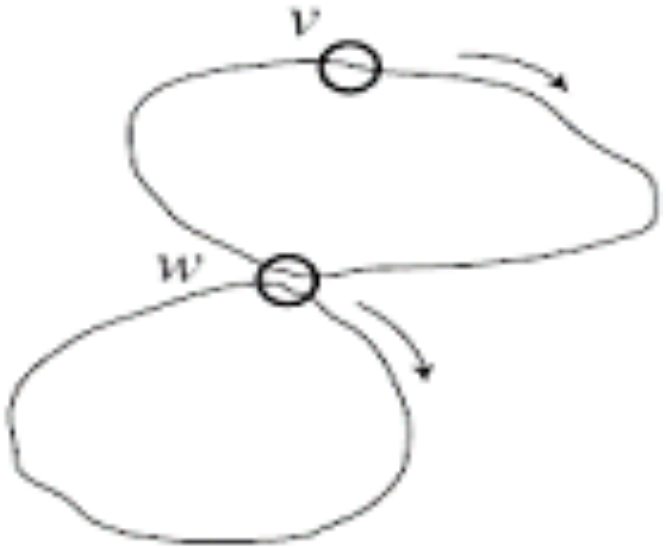
a. Start with an arbitrary vertex *v* and form an arbitrary cycle with unused edges until a dead end is reached. Since the graph is Eulerian this dead end is necessarily the starting point, i.e., vertex *v*.



(a)

Algorithm for Constructing an Eulerian Cycle (cont'd)

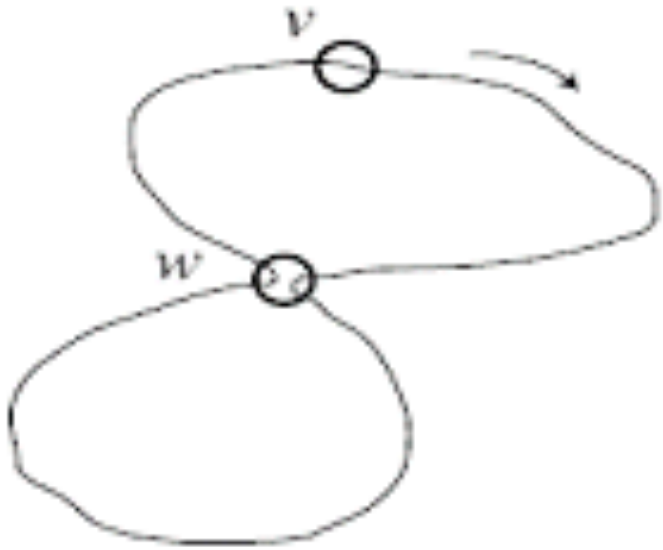
b. If cycle from (a) above is not an Eulerian cycle, it must contain a vertex *w*, which has untraversed edges. Perform step (a) again, using vertex *w* as the starting point. Once again, we will end up in the starting vertex *w.*



(b)

Algorithm for Constructing an Eulerian Cycle

c. Combine the cycles from (a) and (b) into a single cycle and iterate step (b).



(c)

Euler Theorem: extension

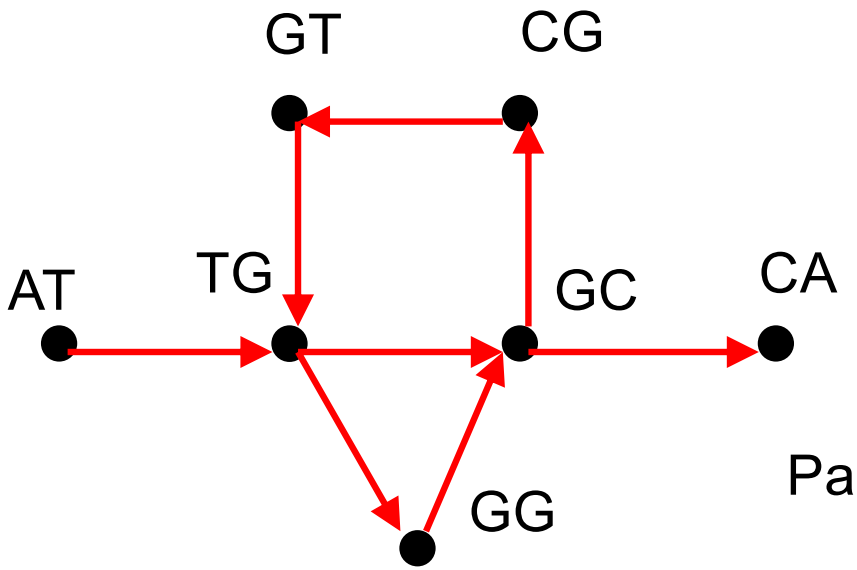
- **Theorem**: *A connected graph has an Eulerian path if and only if it contains at most two semi-balanced vertices and all other vertices are balanced.*

SBH: Eulerian path approach

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

Vertices correspond to (*l* – 1) – mers : { AT, TG, GC, GG, GT, CA, CG }

Edges correspond to *l* – mers from *S*

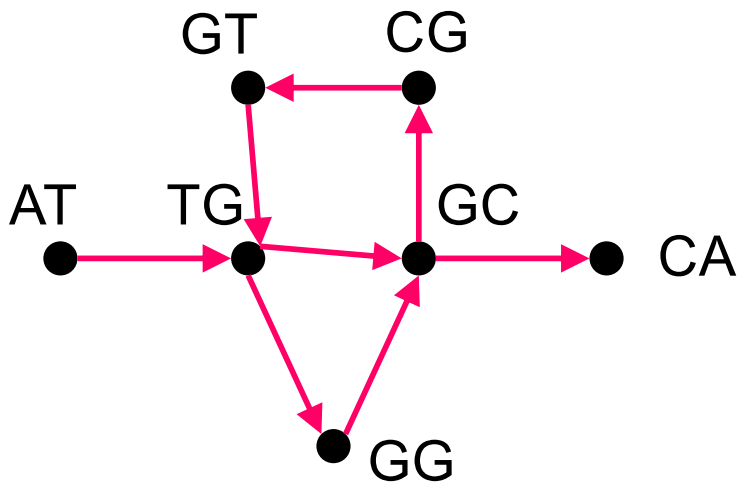


De Bruijn graph

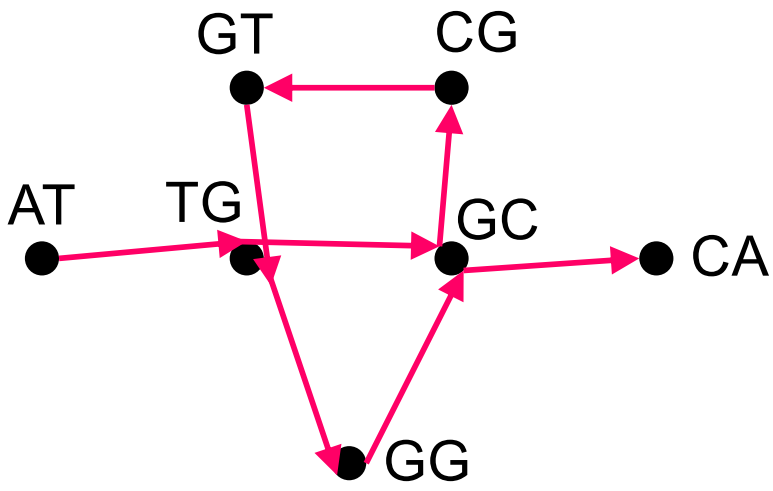
Path visited every EDGE once

SBH: multiple solutions

S = { AT, TG, GC, GG, GT, CA, CG } corresponds to two different paths:



ATGGCGTGCA



ATGCGTGGCA

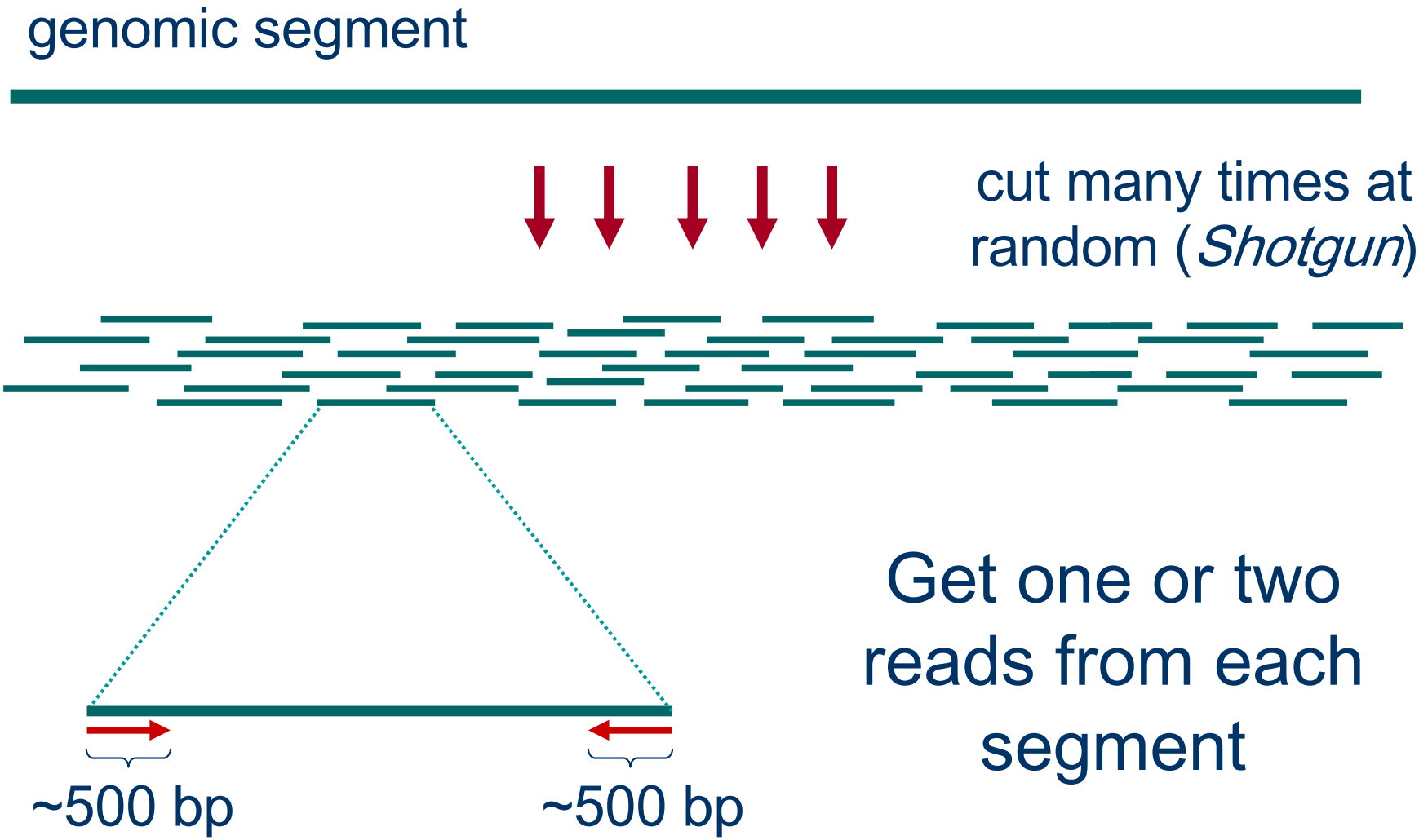
Representing alternative SBH solutions by Eulerian graphs

- # of solutions ←→ # of Eulerian cycles
  - BEST theorem

- Obtains all solutions from one solution → Cassette transformations

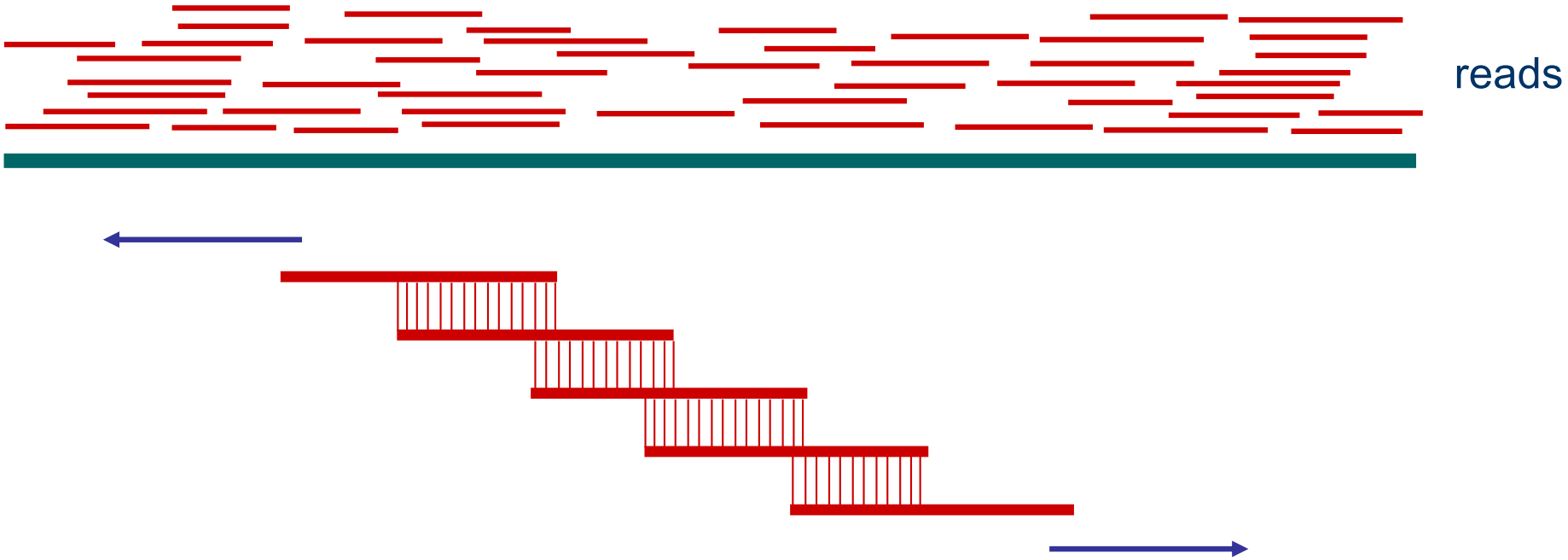
Reads assembly

Shotgun Sequencing



Get one or two reads from each segment

Fragment Assembly



Cover region with ~7-fold redundancy

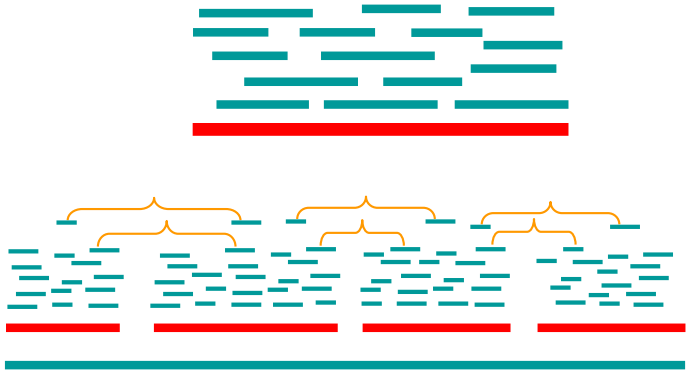
Overlap reads and extend to reconstruct the original genomic region

Overlap-Layout-Consensus

Assemblers: ARACHNE, PHRAP, CAP, TIGR, CELERA

Overlap: find potentially overlapping reads

Layout: merge reads into contigs and contigs into supercontigs



Consensus: derive the DNA sequence and correct read errors

..ACGATTACAATAGGTT..

Overlap

- Find the best match between the suffix of one read and the prefix of another
- Due to sequencing errors, need to use dynamic programming to find the optimal *overlap alignment*
- Apply a filtration method to filter out pairs of fragments that do not share a significantly long common substring

Overlapping Reads

- Sort all *k*-mers in reads (*k* ~ 24)
- Find pairs of reads sharing a k-mer
- Extend to full alignment – throw away if not >95% similar

Overlapping Reads and Repeats

- A *k*-mer that appears N times, initiates N^2 comparisons

- For an *Alu* that appears 10^6 times → 10^{12} comparisons – too much

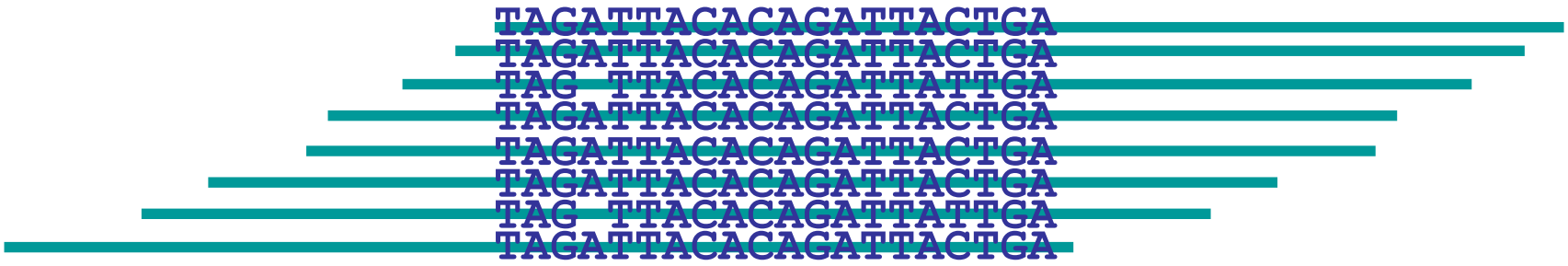
- **<u>Solution:</u>**

  Discard all *k*-mers that appear more than

  $t \times$ Coverage, ($t \sim 10$)

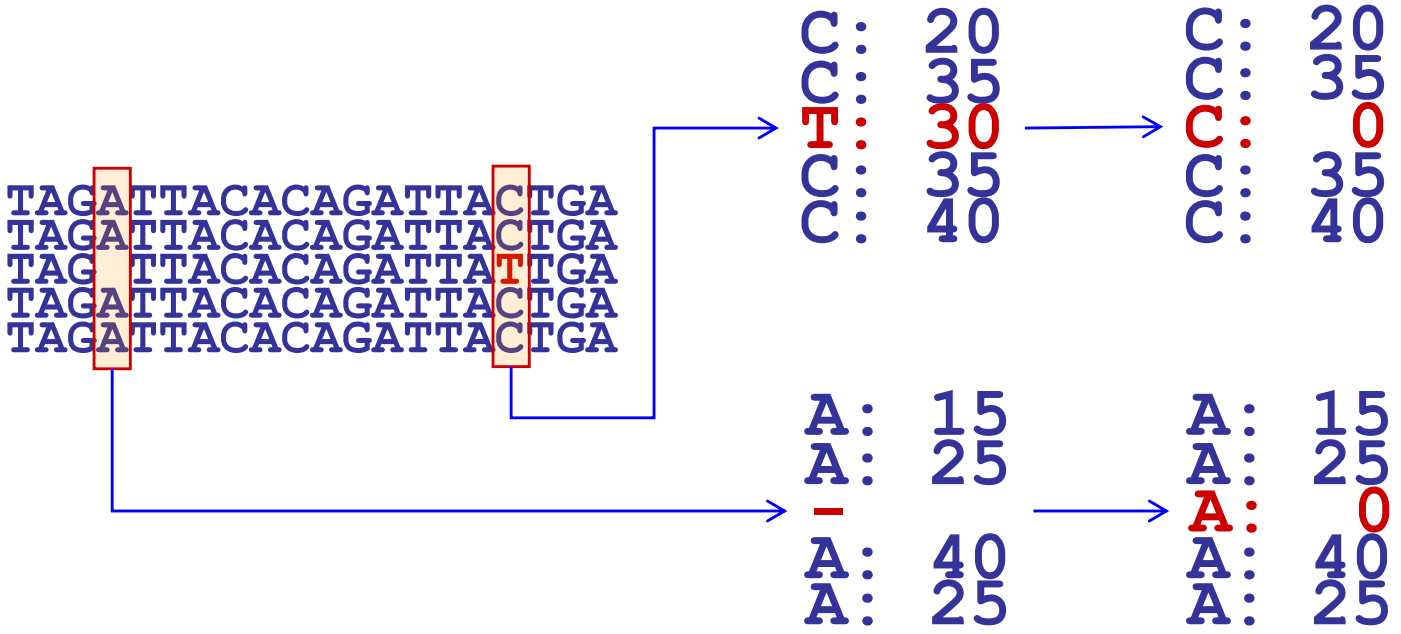
Finding Overlapping Reads

Create local multiple alignments from the overlapping reads



Finding Overlapping Reads
(cont'd)

- Correct errors using multiple alignment

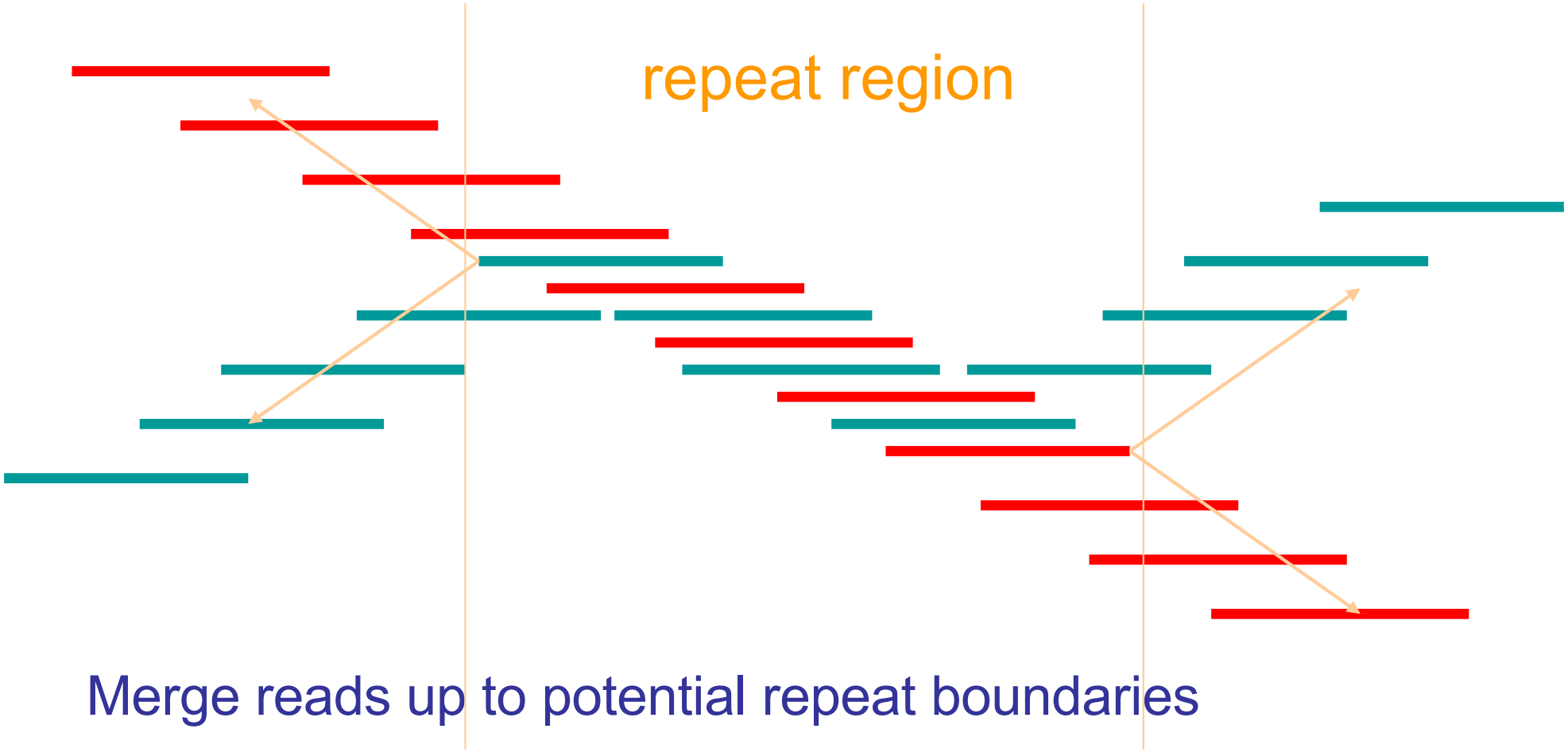


- Score alignments
- Accept alignments with good scores

Layout

- Repeats are a major challenge
- Do two aligned fragments really overlap, or are they from two copies of a repeat?
- Solution: repeat masking – hide the repeats!!!
- Masking results in high rate of misassembly (up to 20%)
- Misassembly means alot more work at the finishing step

Merge Reads into Contigs



Merge reads up to potential repeat boundaries

Repeats, Errors, and Contig Lengths

- Repeats shorter than read length are OK
- Repeats with more base pair differencess than sequencing error rate are OK
- To make a smaller portion of the genome **appear** repetitive, try to:
  - Increase read length
  - Decrease sequencing error rate

Error Correction

Role of error correction:

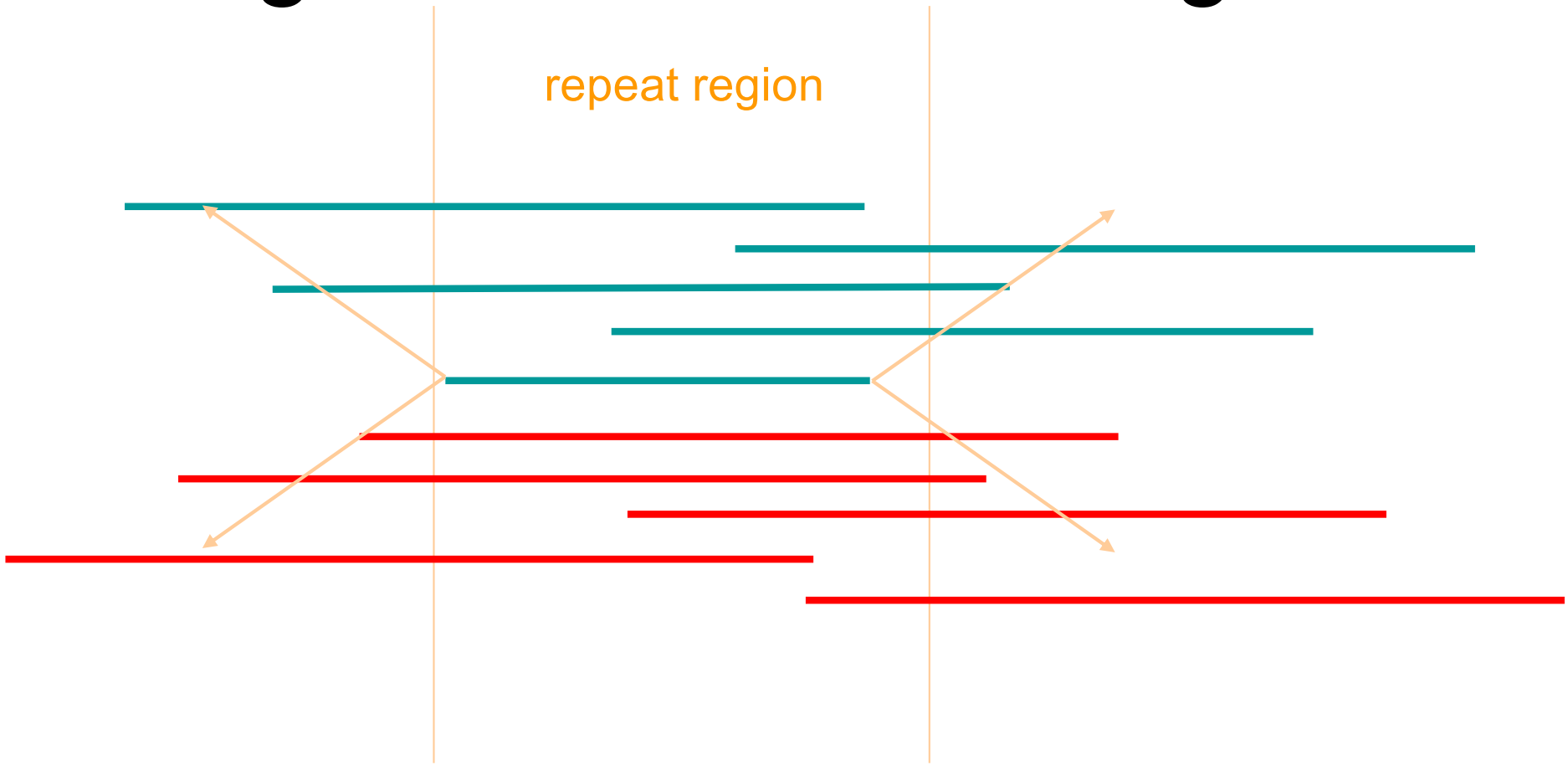
Discards ~90% of single-letter sequencing errors

decreases error rate

$\Rightarrow$ decreases effective repeat content

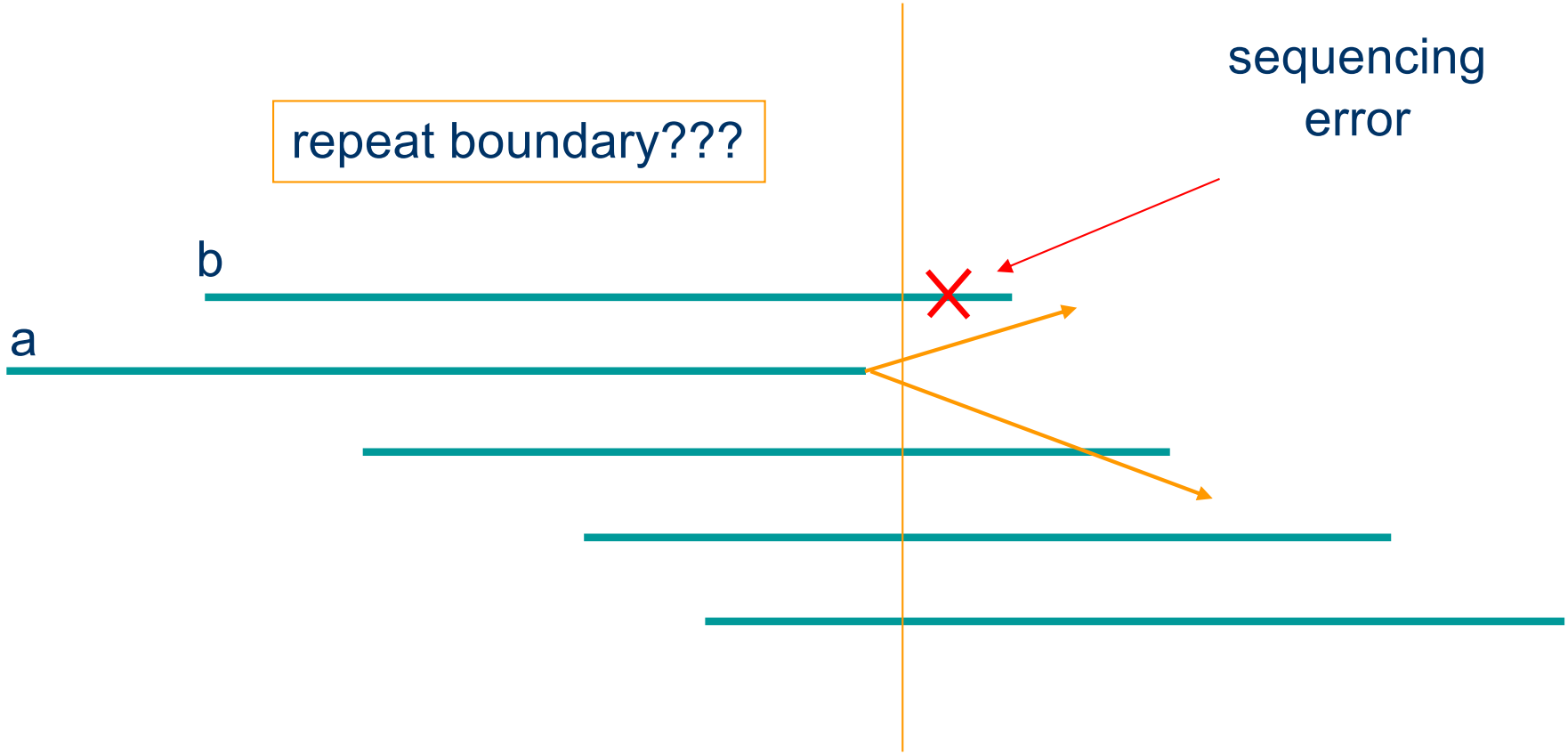
$\Rightarrow$ increases contig length

Merge Reads into Contigs (cont'd)



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

Merge Reads into Contigs (cont'd)



- Ignore "hanging" reads, when detecting repeat boundaries

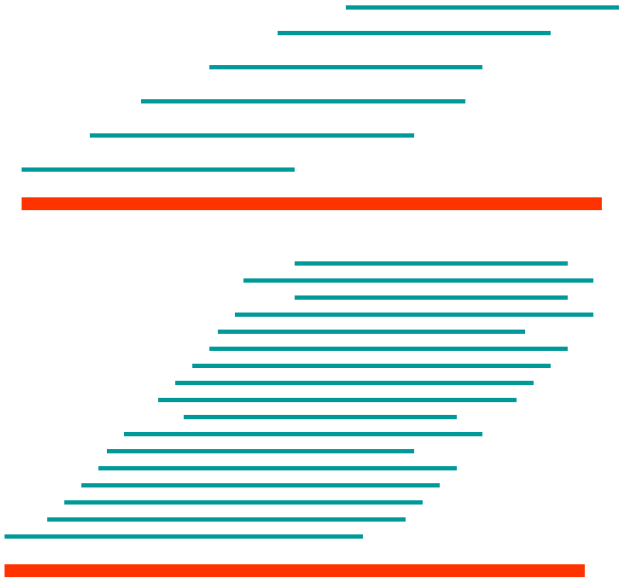
Merge Reads into Contigs (cont'd)

?????

Unambiguous

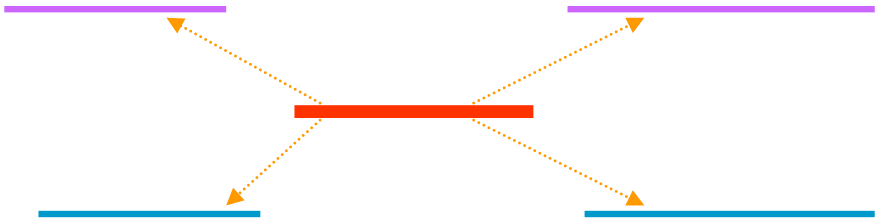
- Insert non-maximal reads whenever unambiguous

Link Contigs into Supercontigs



Normal density

Too dense:
Overcollapsed?

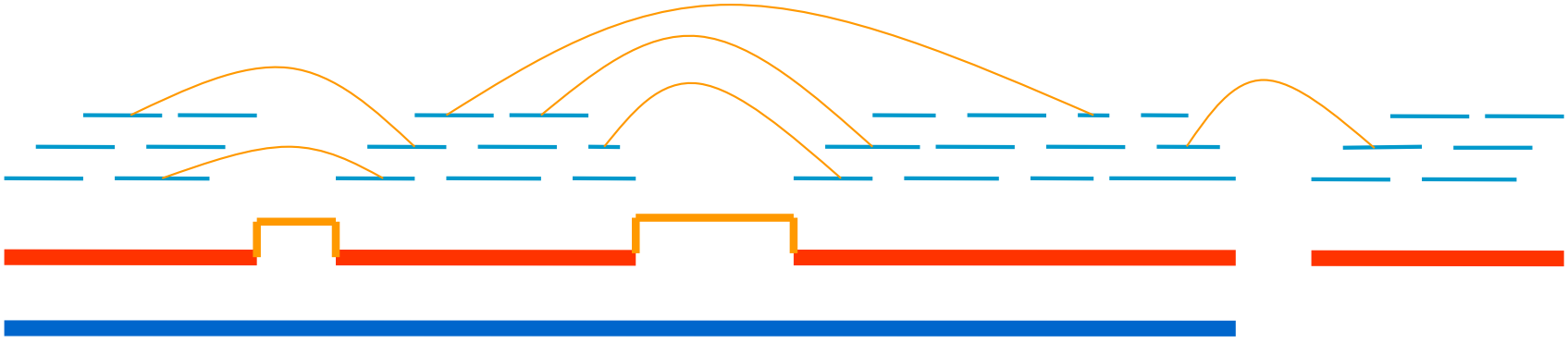


Inconsistent links:
Overcollapsed?

Link Contigs into Supercontigs
(cont'd)

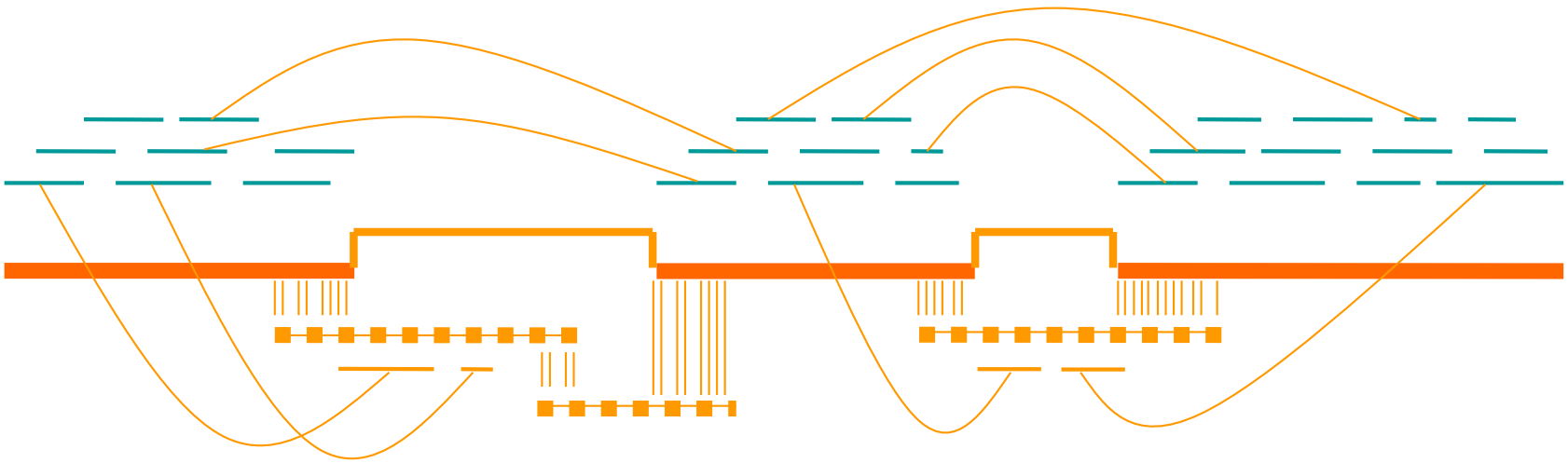
Find all links between unique contigs

Connect contigs incrementally, if ≥ 2 links

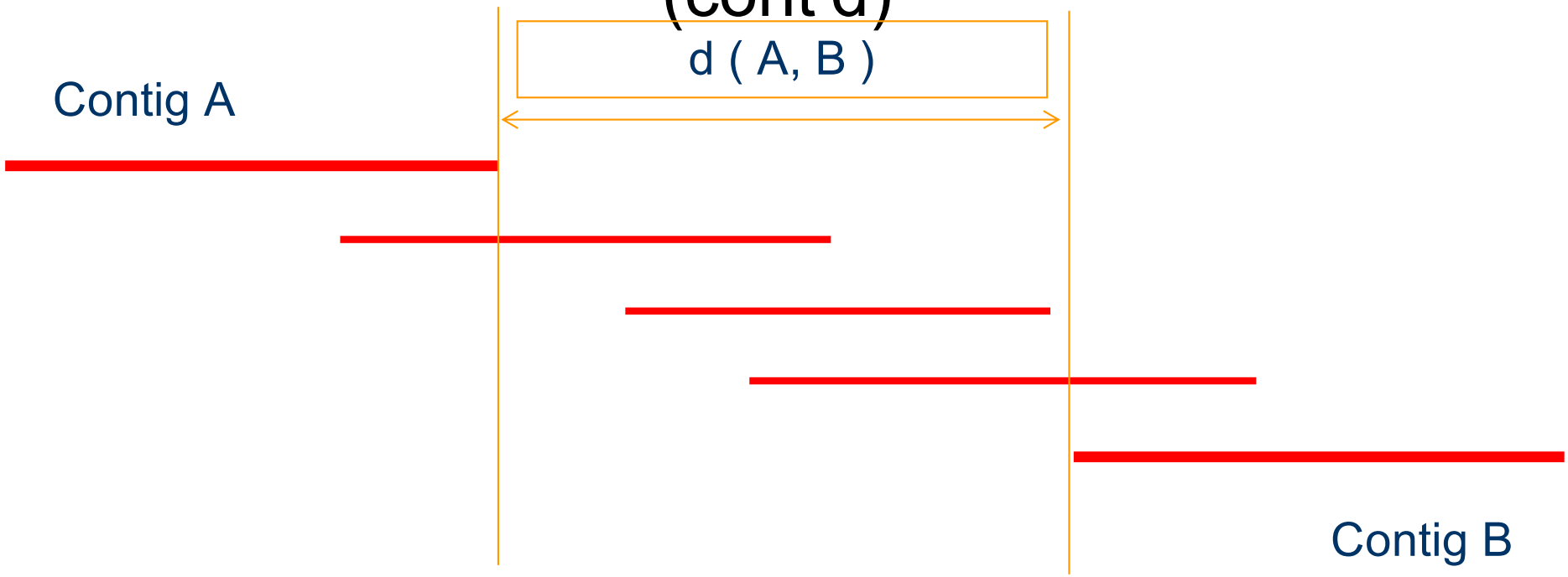


Link Contigs into Supercontigs
(cont'd)

Fill gaps in supercontigs with paths of overcollapsed contigs



Link Contigs into Supercontigs (cont'd)



Define G = (V, E)

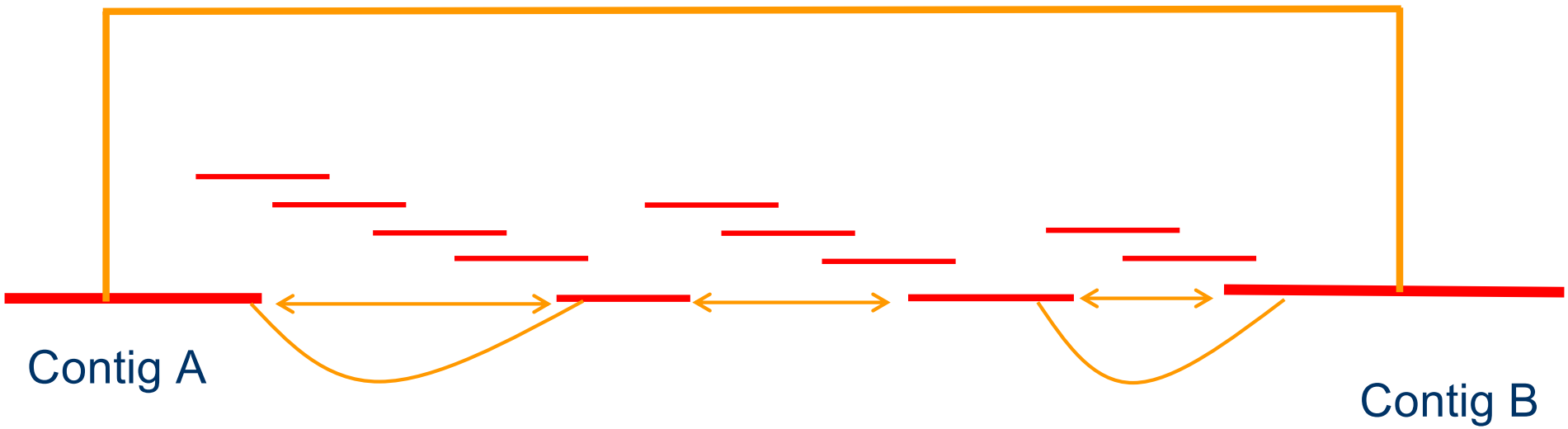
V := contigs

E := (A, B) such that d(A, B) < C

Reason to do so: Efficiency; full shortest paths cannot be computed

Link Contigs into Supercontigs

(cont'd)



Define T: contigs linked to either A or B

Fill gap between A and B if there is a path in G passing only from contigs in T

Consensus

- A consensus sequence is derived from a profile of the assembled fragments
- A sufficient number of reads is required to ensure a statistically significant consensus
- Reading errors are corrected

Derive Consensus Sequence

Derive multiple alignment from pairwise read alignments

Derive each consensus base by weighted voting

Problems with the shotgun approach