

02-251 Recitation 1

Genome Assembly

Wendy & Hongyu
01/18/2019

Outline

- Solving problems on Stepik
- A review of lecture materials
- Biological side of genome assembly
- More algorithms!!!

Solving problems on Stepik

- Authenticate before you solve!!!!
- <https://stepik.org/lesson/201025/step/1?unit=175124>

Review of lecture material

Multiple identical
copies of a genome



Shatter the genome
into reads



Sequence the reads

AGAATATCA

TGAGAATAT

GAGAATATC

Assemble the
genome using
overlapping reads

AGAATATCA
GAGAATATC
TGAGAATAT
...TGAGAATATCA...

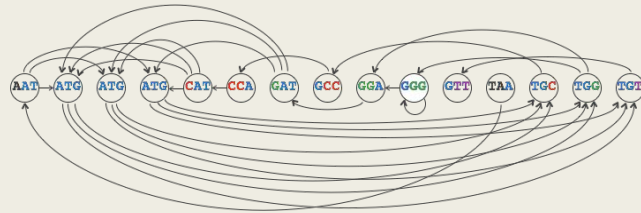
Review of lecture material

String Reconstruction Problem. *Reconstruct a string from its k -mer composition.*

Input: A collection *Patterns* of k -mers.

Output: A string *Text* whose k -mer composition is equal to *Patterns*.

Overlap Graph

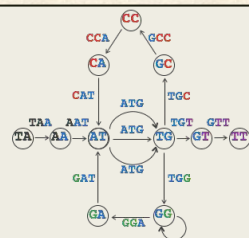


Hamiltonian Cycle Problem

Input: a network with n nodes.

Output: “Yes” if there is a cycle visiting every **node** in the network; “No” otherwise.

De Bruijn Graph



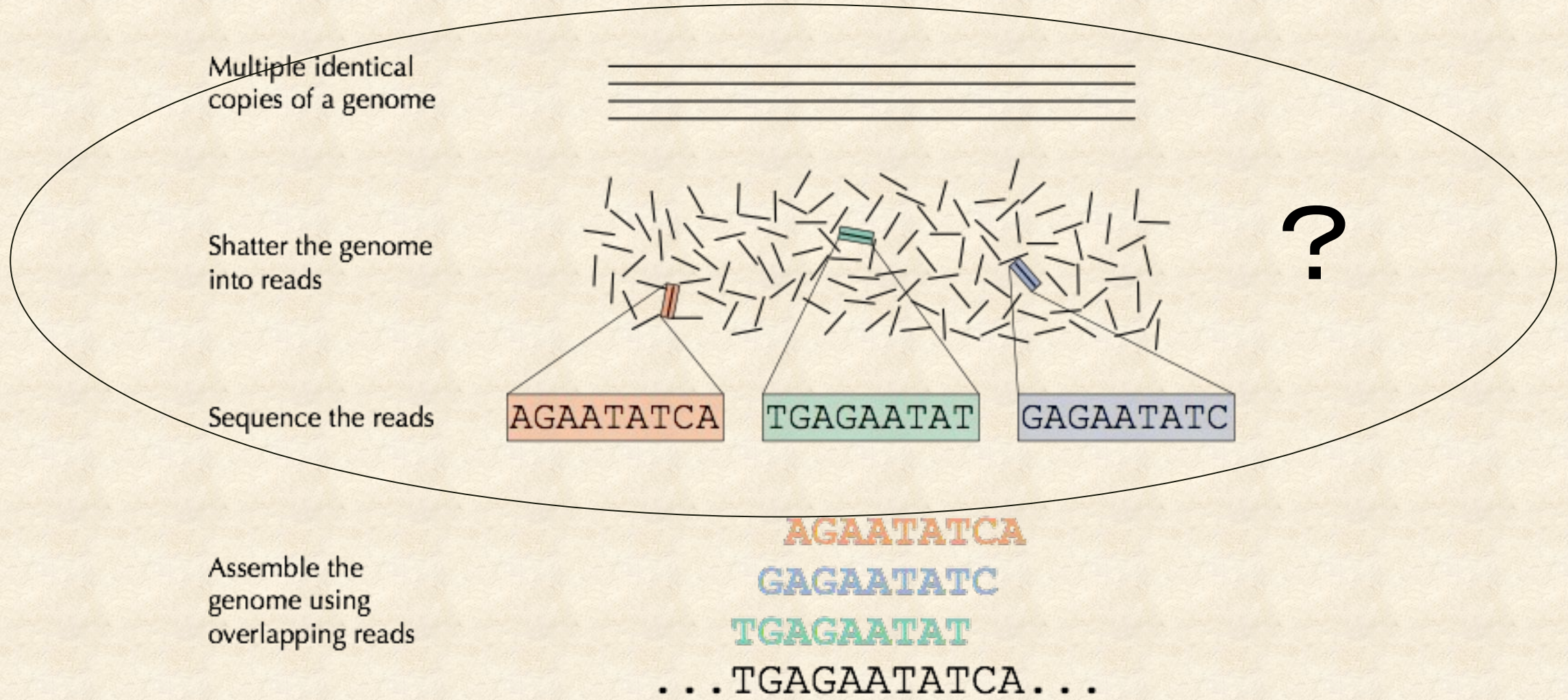
Eulerian Cycle Problem

Input: a network with n nodes.

Output: “Yes” if there is a cycle visiting every **edge** in the network; “No” otherwise.



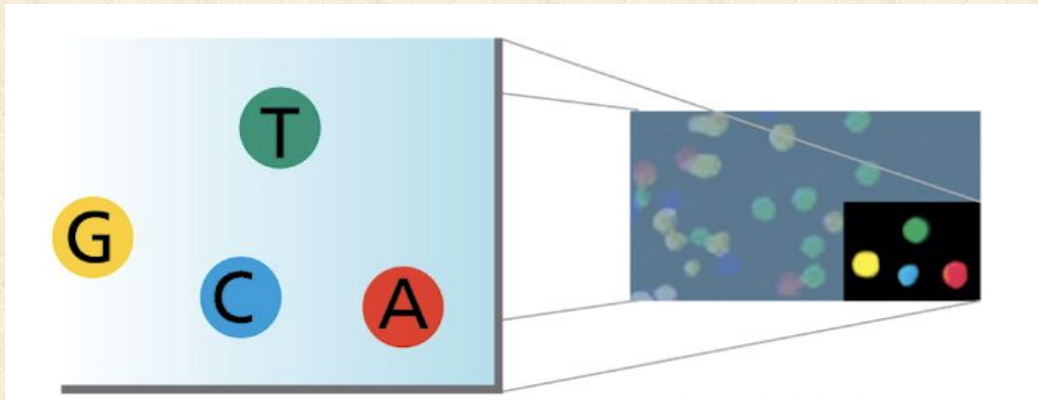
Review of lecture material



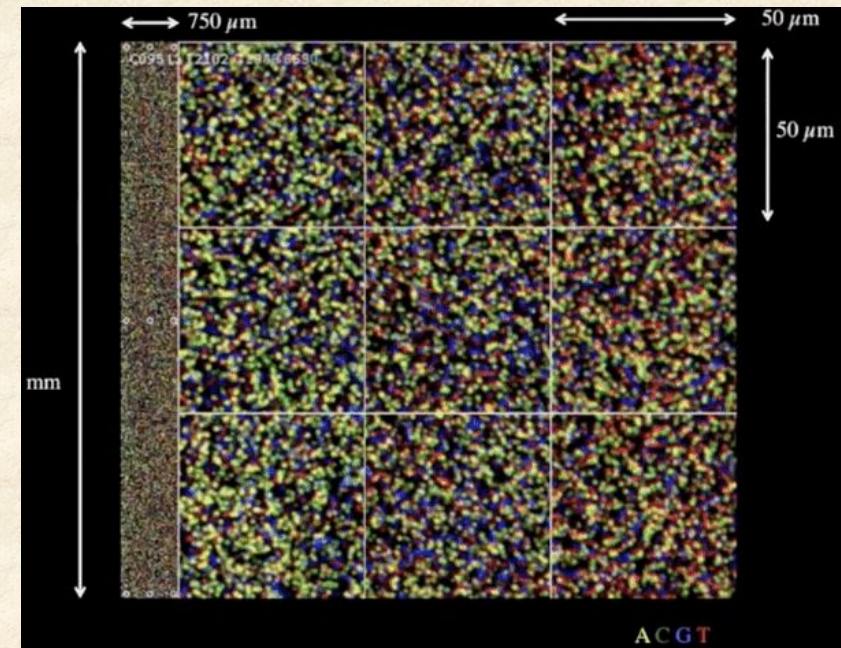
Biological side of genome assembly

How do we get the reads?

Basic idea: during DNA replication, we can add nucleotides with different fluorescent labels, detect the light emitted, and determine which nucleotide was attached



Mardis ER, 2008



Youtube: [Illumina four-color sequencing by synthesis](#)

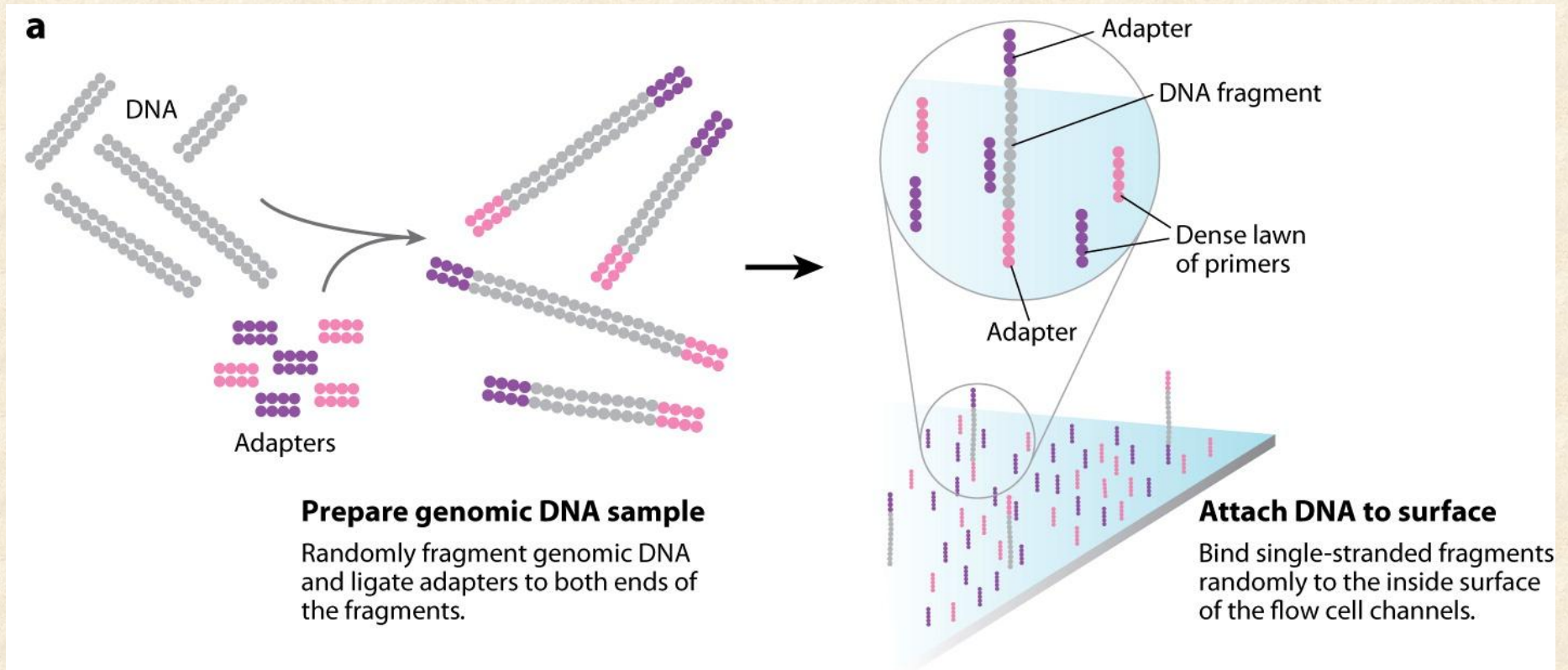
Sequencing by synthesis



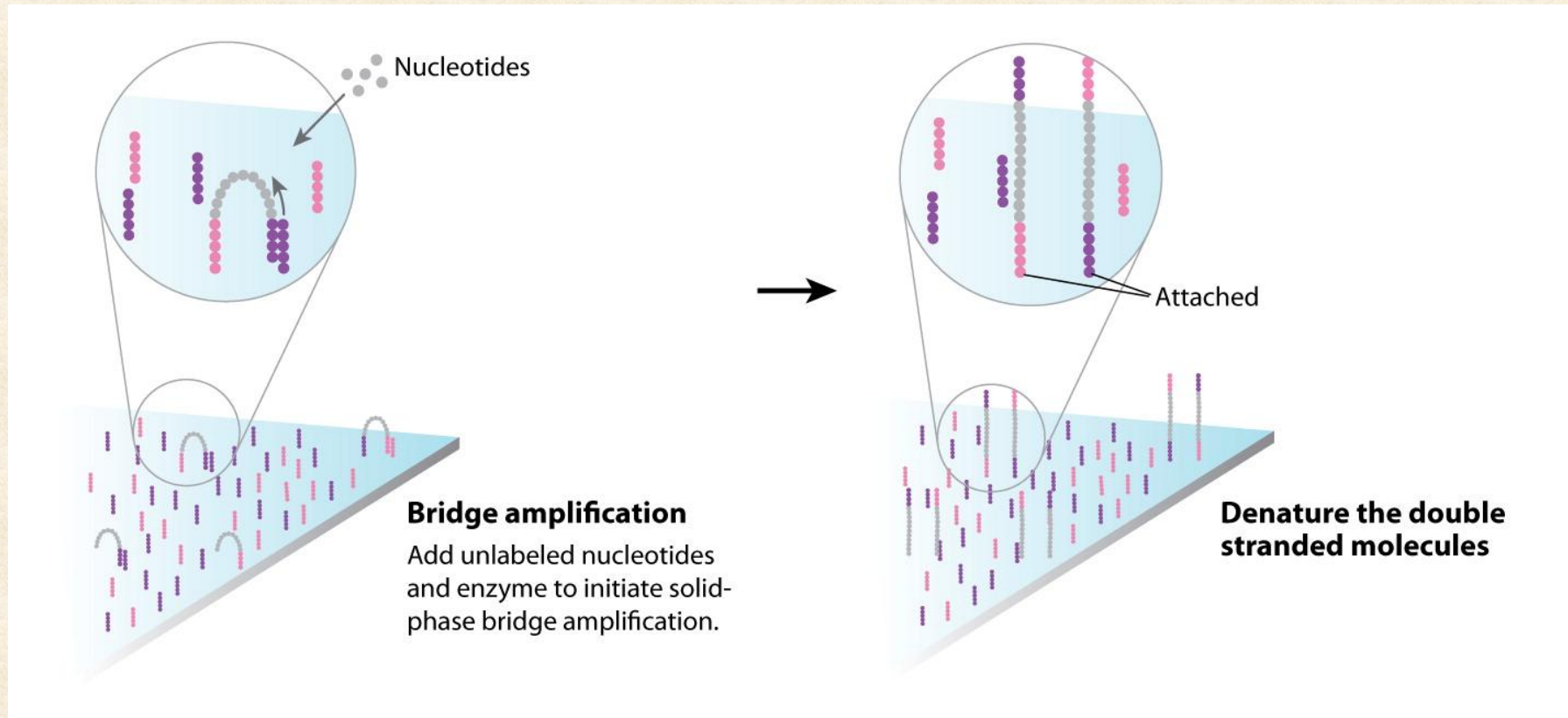
Sequencing by synthesis



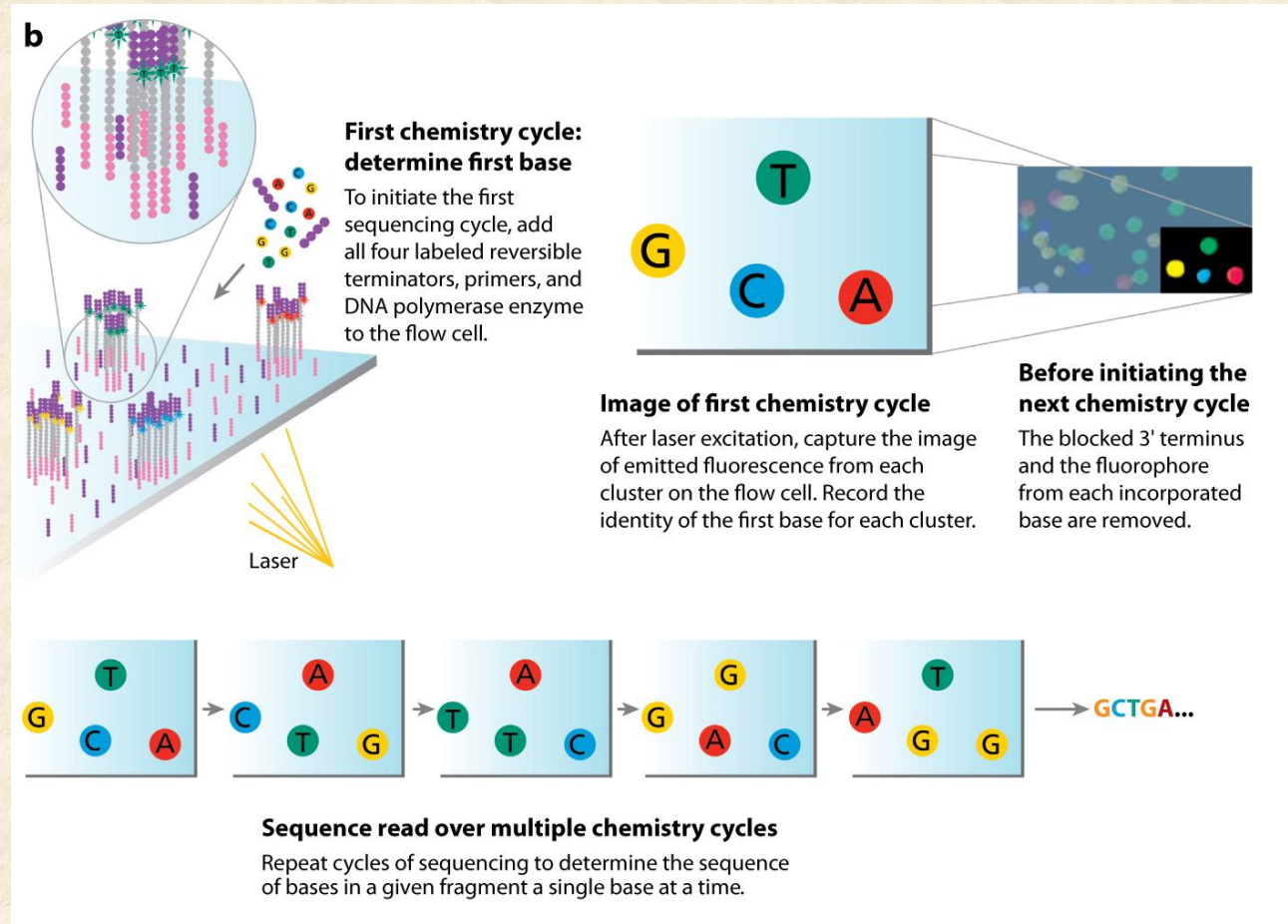
Sequencing by synthesis - Modification



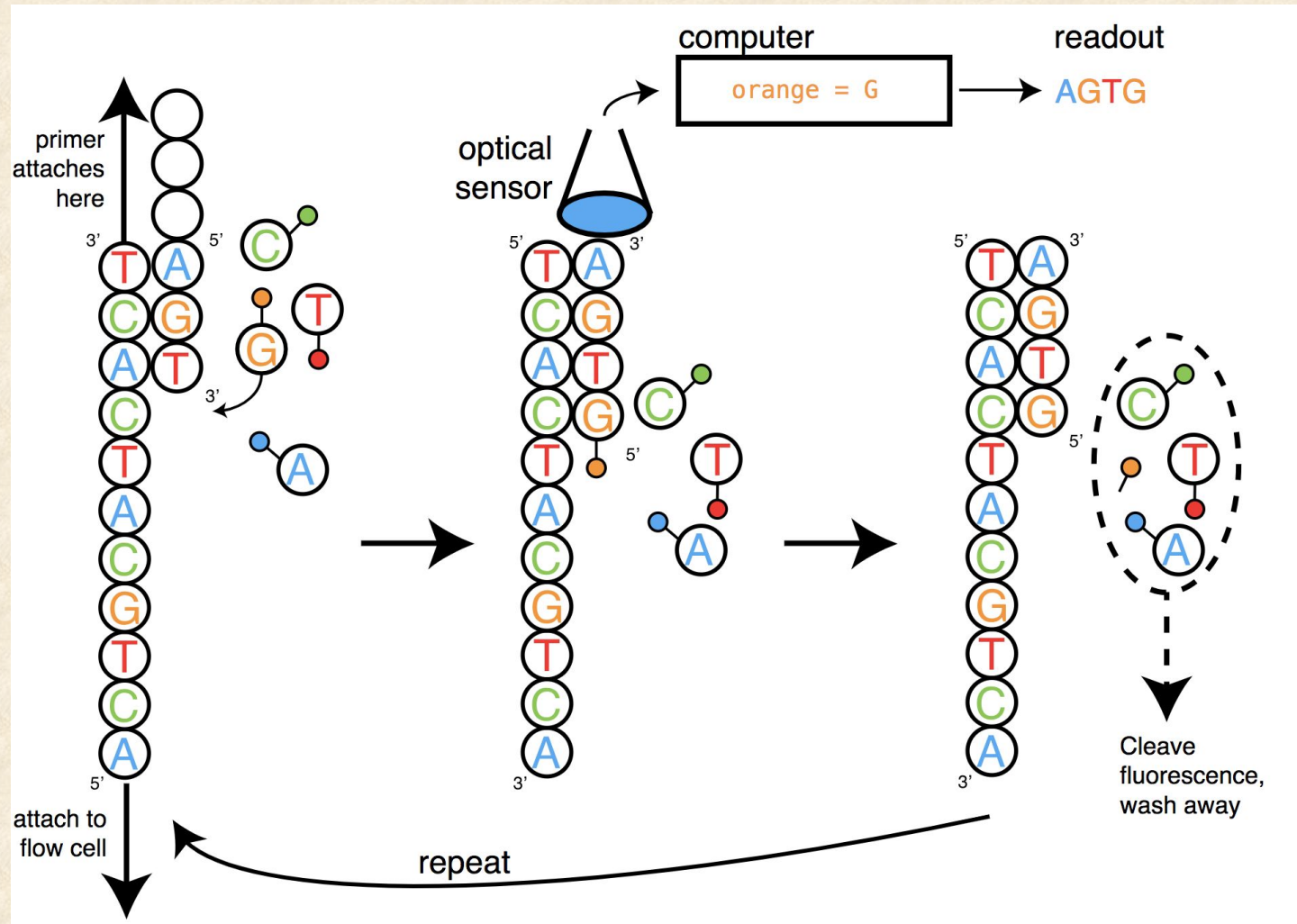
Sequencing by synthesis - Amplification



Sequencing by synthesis - Sequencing



Sequencing by synthesis - Sequencing



Chemistry cycle during sequencing:

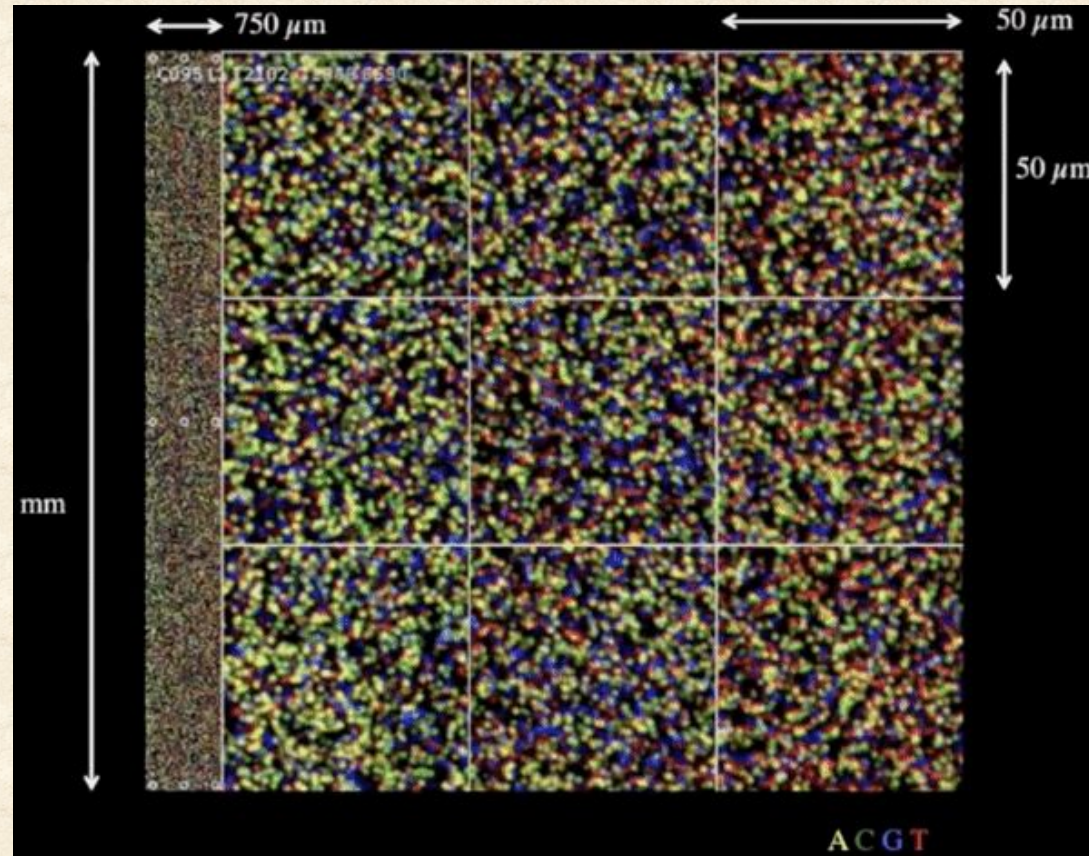
Reversible terminator binding to DNA template

Fluorescent light emitted and captured by the sensor

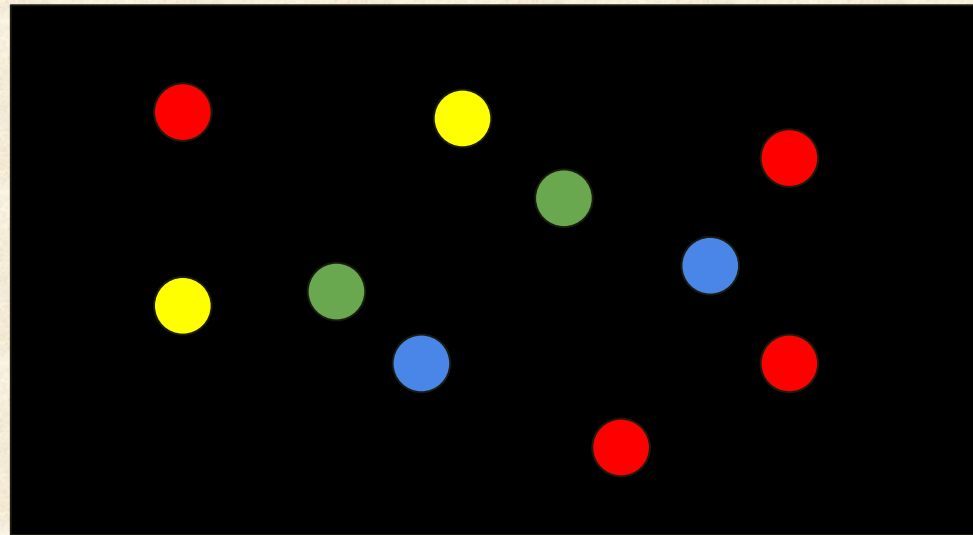
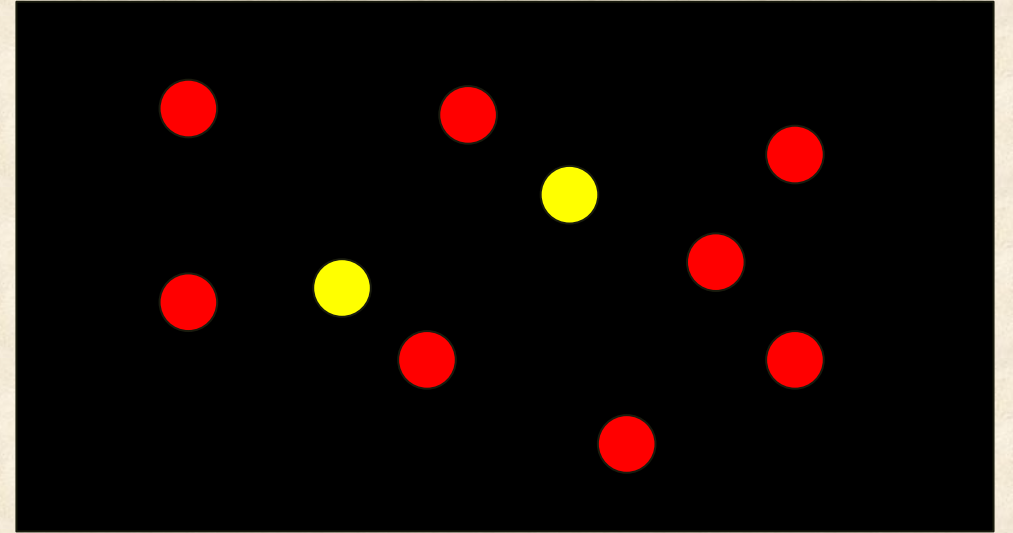
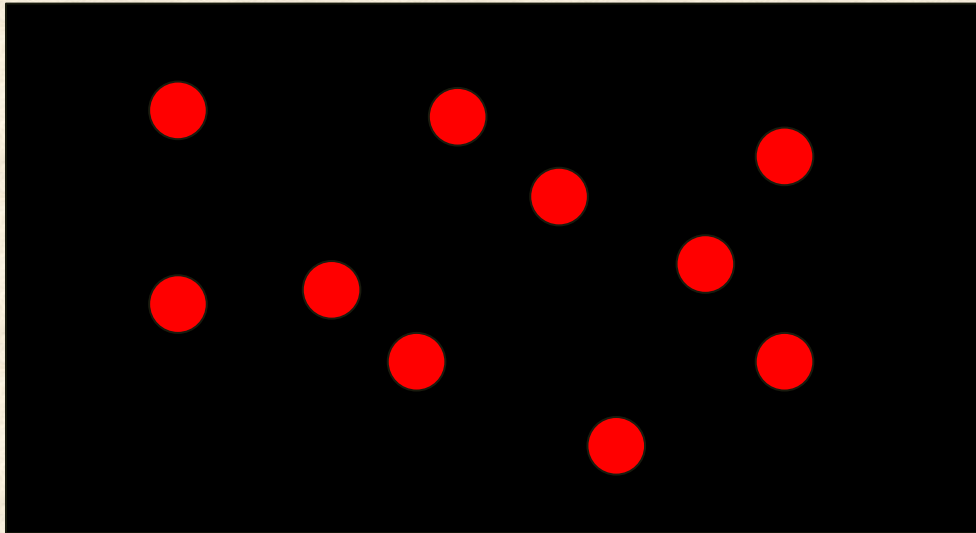
Blocking removed, allowing a new reversible terminator to bind

Sequencing by synthesis - Sequencing

Tracking the light is also a computational problem!!!



Sequencing by synthesis - Sequencing



More Background in Assemblies:

- The Old Guards
- Metric of Quality
- The Assemblathon

Basic Algorithms Stuff

- Finding a Eulerian Circuit
- Non-Branching Paths
- Finding All Eulerian Circuits

The OLC Family

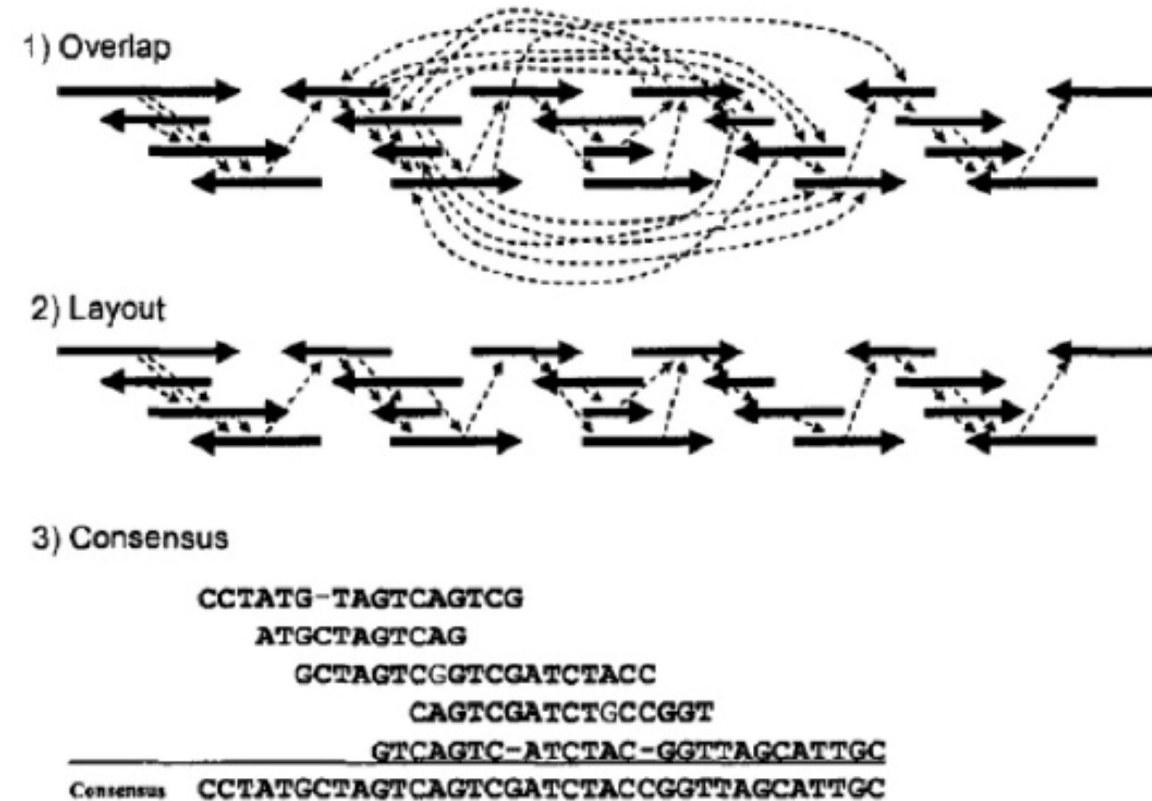
a.k.a Hamiltonian-Circuit Family

Overlap: Align sequences against each other and find overlaps

Layout: Find an arrangement of reads that respect some rules

Consensus: Merge all seen sequences with overlaps

- Mainstream algorithm in early times
- Gets pretty messy with the rules, got scaling issues, and is now superseded by de Burgin graph based algorithms



*Credit: Masahiro Kasahara, Large-Scale Genome Sequence Processing,
Imperial College Press*

Reality of Assemblies

You don't get a single sequence for one chromosome every time(or practically, never).

Now say I and Wendy have two assemblies. Each of several contigs/scaffolds.

Think about this: How do you know which one is better?

Quality Assessment

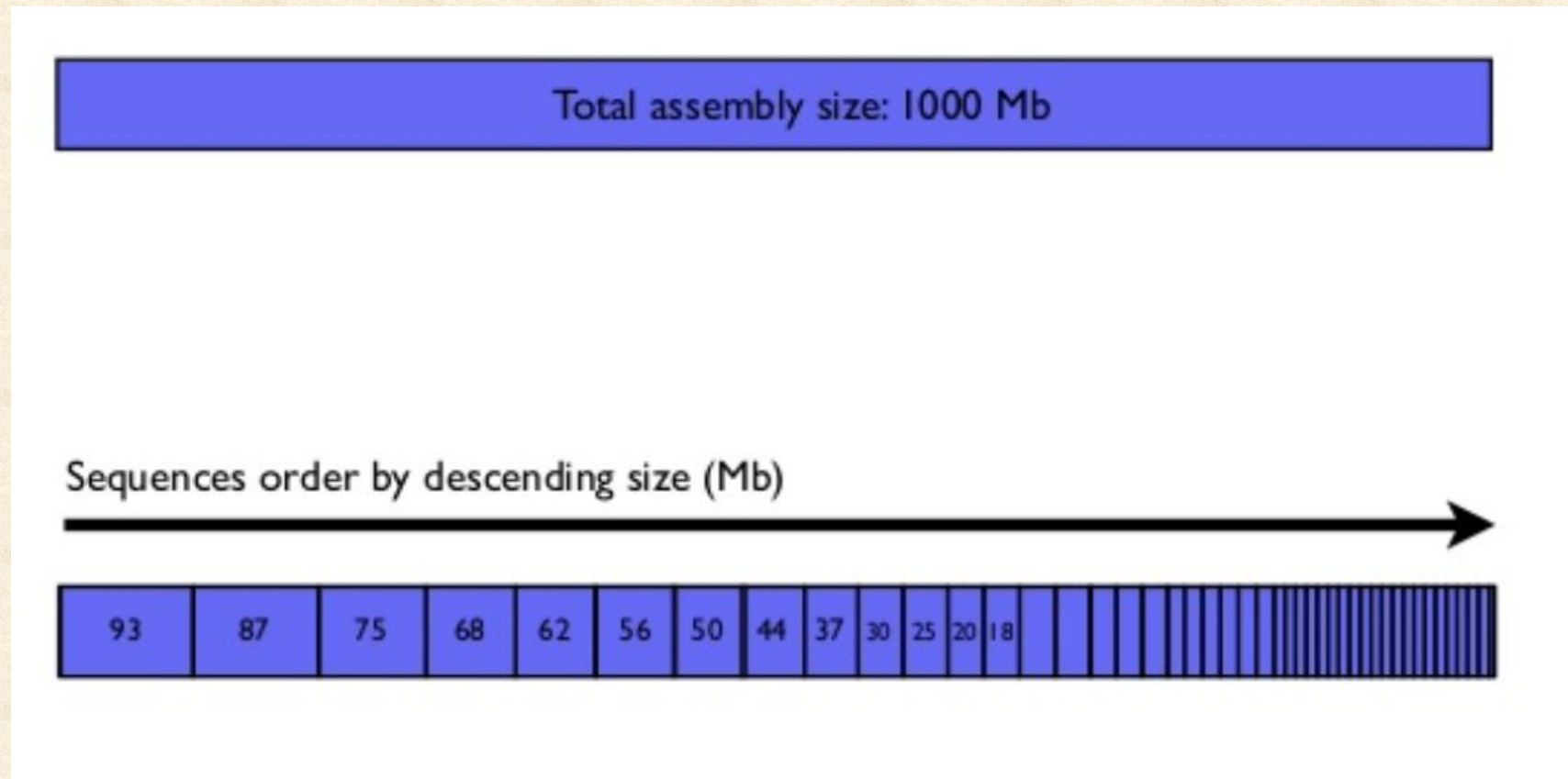
Some simple case.

- Size - If you know the expected sequence is 400Mbp(bp = base pair) but I gave you something of 20Mbp...
- Coverage - If I don't even most of the inputs in your assemblies...

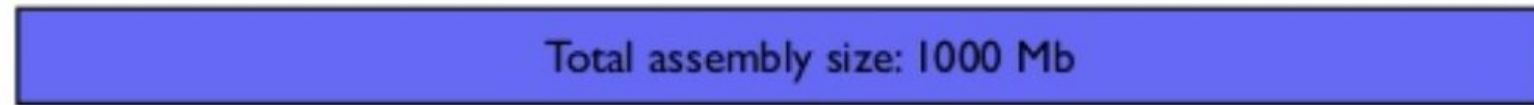
Most people don't make this type of mistakes.

- Mean Fragment Size - Trick the judge by throwing out short fragments
- The most accepted metric for nowadays is **N50/L50**

Setting Up



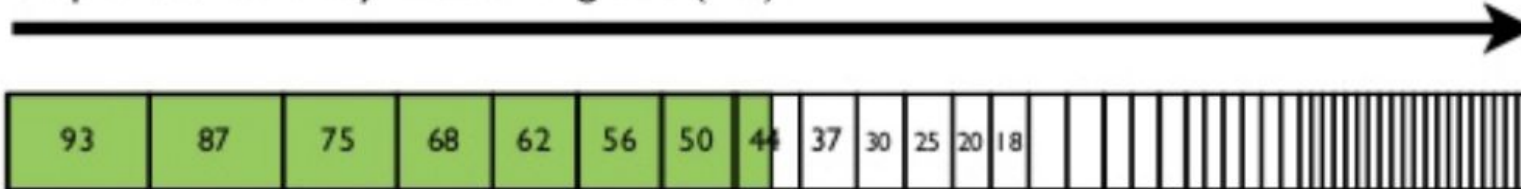
N50/L50



N50



Sequences order by descending size (Mb)



N50 = 7 sequences

L50 = 50 Mb

The Assemblathon

Question: What is the best assembler in the world?

If everyone claims their assembly method is the best, then let's hold a contest. They got a cool logo. At least by 2007 standards, I guess.

The logo for 'The Assemblathon' features the word 'THE' in a smaller, bold, blue font with a yellow outline, positioned above the word 'ASSEMBLATHON'. 'ASSEMBLATHON' is in a much larger, bold, blue font with a yellow outline. Both words have a slight 3D effect with a grey shadow underneath. The entire logo is set against a white rectangular background.

THE ASSEMBLATHON

The Assemblathon (The Sequel)

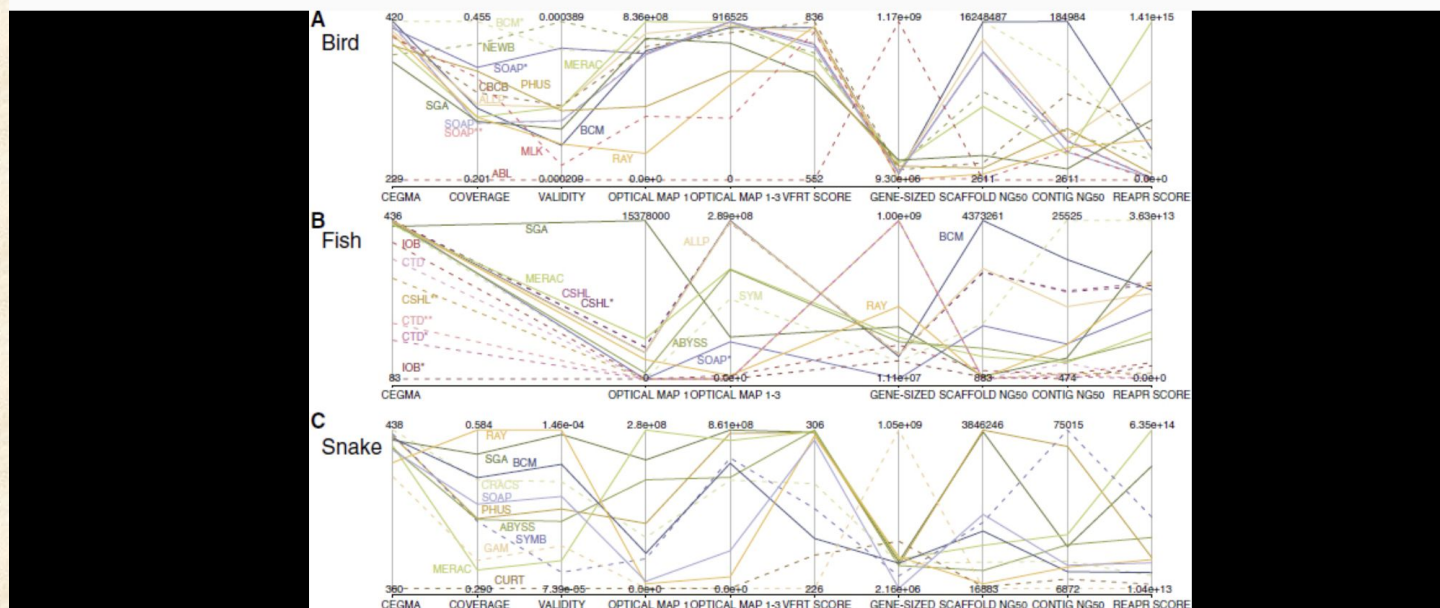
Table 2 Overview of sequencing data provided for Assemblathon 2 participants

Species	Estimated genome size	Illumina	Roche 454	Pacific biosciences
Bird (<i>Melopsittacus undulatus</i>)	1.2 Gbp	285x coverage from 14 libraries (mate pair and paired-end)	16x coverage from 3 library types (single end and paired-end)	10x coverage from 2 libraries
Fish (<i>Maylandia zebra</i>)*	1.0 Gbp	192x coverage from 8 libraries (mate pair and paired-end)	NA	NA
Snake (<i>Boa constrictor constrictor</i>)	1.6 Gbp	125x coverage from 4 libraries (mate pair and paired-end)	NA	NA

*Also described as *Metriaclima zebra* and *Pseudotropheus zebra*.
See Additional file 1 for a full description of all sequence data.

The Assemblathon (The Sequel)

Figure 21



Parallel coordinate mosaic plot showing performance of all assemblies in each key metric. Performance of bird, fish, and snake assemblies (panels A–C) as assessed across ten key metrics (vertical lines). Scales are indicated by values at the top and bottom of each axis. Each assembly is a colored, labeled line. Dashed lines indicate teams that submitted assemblies for a single species whereas solid lines indicate teams that submitted assemblies for multiple species. Key metrics are CEGMA (number of 458 core eukaryotic genes present); COVERAGE and VALIDITY (of validated Fosmid regions, calculated using COMPASS); OPTICAL MAP 1 and OPTICAL MAP 1–3 (coverage of optical maps at level 1 or at all levels); VFRT SCORE (summary score of validated Fosmid region tag analysis), GENE-SIZED (the amount of an assembly's scaffolds that are 25 Kbp or longer); SCAFFOLD NG50 and CONTIG NG50 (the lengths of the scaffold or contig that takes the sum length of all scaffolds/contigs past 50% of the estimated genome size); REAPR SCORE (summary score of scaffolds from REAPR tool).

The Assemblathon (The Sequel)



Can't get enough authors!

Results?

- However, the high degree of variability between the entries suggests that there is still much room for improvement in the field of genome assembly and that approaches which work well in assembling the genome of one species may not necessarily work well for another.

Basically there are no overall winner

Assemblathon 2: evaluating *de novo* methods of genome assembly in three vertebrate species

Keith R Bradnam , Joseph N Fass, Anton Alexandrov, Paul Baranay, Michael Bechner, Inanç Birol, Sébastien Boisvert, Jarrod A Chapman, Guillaume Chapuis, Rayan Chikhi, Hamidreza Chitsaz, Wen-Chi Chou, Jacques Corbeil, Cristian Del Fabbro, T Roderick Docking, Richard Durbin, Dent Earl, Scott Emrich, Pavel Fedotov, Nuno A Fonseca, Ganeshkumar Ganapathy, Richard A Gibbs, Sante Gnerre, Élénie Godzaridis, Steve Goldstein, Matthias Haimel, Giles Hall, David Haussler, Joseph B Hiatt, Isaac Y Ho, Jason Howard, Martin Hunt, Shaun D Jackman, David B Jaffe, Erich D Jarvis, Huaiyang Jiang, Sergey Kazakov, Paul J Kersey, Jacob O Kitzman, James R Knight, Sergey Koren, Tak-Wah Lam, Dominique Lavenier, François Laviolette, Yingrui Li, Zhenyu Li, Binghang Liu, Yue Liu, Ruibang Luo, Iain MacCallum, Matthew D MacManes, Nicolas Maillet, Sergey Melnikov, Delphine Naquin, Zemin Ning, Thomas D Otto, Benedict Paten, Octávio S Paulo, Adam M Phillippy, Francisco Pina-Martins, Michael Place, Dariusz Przybylski, Xiang Qin, Carson Qu, Filipe J Ribeiro, Stephen Richards, Daniel S Rokhsar, J Graham Ruby, Simone Scalabrin, Michael C Schatz, David C Schwartz, Alexey Sergushichev, Ted Sharpe, Timothy I Shaw, Jay Shendure, Yujian Shi, Jared T Simpson, Henry Song, Fedor Tsarev, Francesco Vezzi, Riccardo Vicedomini, Bruno M Vieira, Jun Wang, Kim C Worley, Shuangye Yin, Siu-Ming Yiu, Jianying Yuan, Guojie Zhang, Hao Zhang, Shiguo Zhou, Ian F Korf 

Author Notes

GigaScience, Volume 2, Issue 1, 1 December 2013, 2047-217X-2-10,
<https://doi.org/10.1186/2047-217X-2-10>

Published: 22 July 2013 **Article history** ▼

Theory Homework #1

PSA: For some of these homeworks you are graded on your effort even if your solution is not correct.

Answer #1: Use nodes as k-mers, directed edges to denote (k-1)-mer overlap.

Answer #2: Find a Hamiltonian path over the nodes (assuming no duplicate k-mers).

- What if in #1, I add an edge as long as nodes have some overlap? (Say, if two 5-mers have 3-bp overlap, I add an edge as well.)

Theory Homework #1

You will get suboptimal assemblies. Or blowups. (Boom.)

Example:

Sequence is ATCAT. 3-mers are ATC, TCA, CAT.

Correct/optimal Hamiltonian path is ATC->TCA->CAT.

a valid Hamiltonian path: CAT->TCA->ATC, get the sequence CATCATC instead.

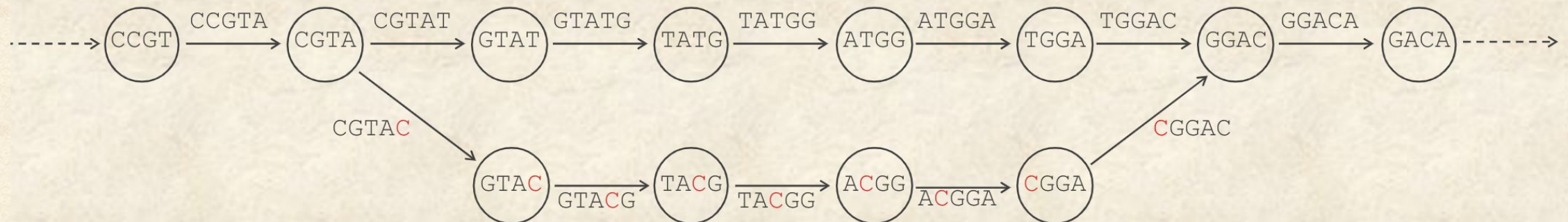
Non-Branching Paths

Input:

A Directed Graph $G = (V, E)$.

Output:

Set of Maximal Non-Branching Paths - Paths that all internal vertices have in degree of 1 and out degree of 1, and is not subpath of any other Non-Branching Paths



Non-Branching Paths

Key observation:

- No two MNBP share starting edge.
- Starting edge of MNBP decides the maximal path itself.

Just iterate over all possible starting edges and be done.

- For each edge (u, v) : If u is not a 1-in-1-out node, (u, v) is eligible to start an MNBP.
- Walk from v until current vertex is no longer 1-in-1-out.
- Add visited nodes to results.

Non-Branching Paths

Pitfall: 1-in-1-out edges can start NMBP. It happens in isolated circles.

The **MaximalNonBranchingPaths** pseudocode below generates all non-branching paths in a graph. It iterates through all nodes of the graph that are not 1-in-1-out nodes and generates all non-branching paths starting at each such node. In a final step, **MaximalNonBranchingPaths** finds all isolated cycles in the graph.

```
MaximalNonBranchingPaths(Graph)
  Paths  $\leftarrow$  empty list
  for each node v in Graph
    if v is not a 1-in-1-out node
      if out(v) > 0
        for each outgoing edge (v, w) from v
          Path  $\leftarrow$  the path consisting of single edge (v, w)
          while w is a 1-in-1-out node
            extend NonBranchingPath by the edge (w, u)
            w  $\leftarrow$  u
          add NonBranchingPath to the set Paths
  for each isolated cycle Cycle in Graph
    add Cycle to Paths
  return Paths
```

Building an Eulerian Circuit

Input:

A Graph $G = (V, E)$. Ensures G have a Eulerian Circuit (strongly connected, each vertex have same in-degree and out-degree).

Output:

A sequence of vertices.

Hierholzer's Algorithm (1873)

Pick any vertex as a starting point.

Travel on the graph, removing visited edges on the way. Stop when you get back to the starting vertex.

If there are vertices on the tour that have unvisited edges, start from that vertex and join the resulting path.

Proof of Correctness:

1. You cannot get stuck at other vertices.
2. You don't end up with unvisited cycles.

Hierholzer's Algorithm (1873)

```
function explore(v) {  
    ret := [], cur := v  
    while (cur != v) {  
        find an outgoing edge: (cur -> next)  
        remove edge, ret += [cur], cur := next  
    }  
    return ret  
}
```

Hierholzer's Algorithm (1873)

```
function explore(v)
function circuit {
    path := explore(0)
    while graph is not empty {
        find v in path that have nonzero degree
        idx = path.index(v)
        path = path[:idx] + explore(v) + path[idx:]
    }
    return path
}
```

Hierholzer's Algorithm (1873)

OK, that's not very cool. Concatenating lists take a long time in some cases.

Brainstorming: An $O(M)$ algorithm?

Hierholzer's Algorithm (1873)

```
tour := []
result := []
function explore(v) {
    cur := v
    while (cur != v) {
        find an outgoing edge: (cur -> next)
        remove edge, push cur to tour, cur := next
    }
}
```


Hierholzer's Algorithm (1873)

```
tour := []
result := []
function explore(v) {
    cur := v
    while (cur != v) {
        find an outgoing
edge: (cur -> next)
        remove edge, push
cur to tour, cur := next
    }
}
```

```
function circuit {
    explore(0)
    while tour is not empty {
        pop v from tour, push v to result
        if v has more edges: explore(v)
    }
    return result # read from top
}
```

Counting Eulerian Circuits

Input:

A Directed Graph $G = (V, E)$. Ensures G have a Eulerian Circuit (strongly connected, each vertex have same in-degree and out-degree). We also assume G have no duplicate edges.

Output:

Number of Eulerian Circuits in G . Two ECs are same if list of visited vertices are the same after rotation.

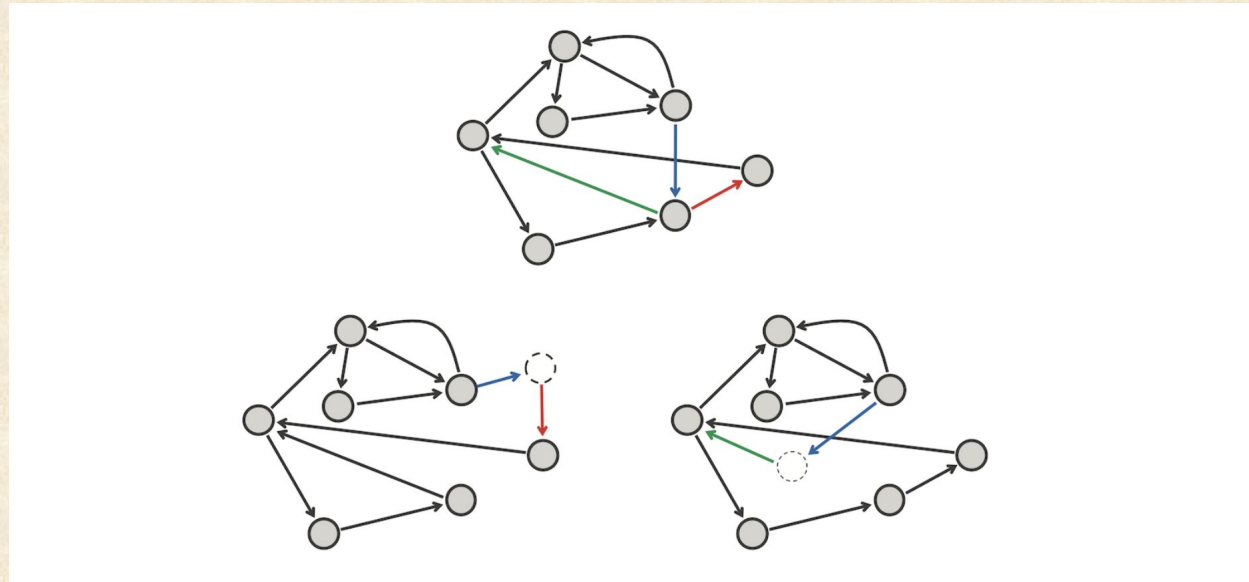
Counting Eulerian Circuits

The core idea: Transform one graph with n possible circuit into n graphs with 1 possible circuit.

Fix edge (u, v) . If v have multiple out edges, for each of them (v, w) , create a new graph where the edges (u, v) and (v, w) are removed, a new vertex x is added alongside with edges (u, x) and (x, w) .

Question 1: Why does this converge?

Question 2: Correctness?



Counting Eulerian Circuits

The core idea: Transform one graph with n possible circuit into n graphs with 1 possible circuit.

Fix edge (u, v) . If v have multiple out edges, for each of them (v, w) , create a new graph where the edges (u, v) and (v, w) are removed, a new vertex x is added alongside with edges (u, x) and (x, w) .

Question 1: Why does this converge?

- Each new graph have one more vertex than the old graph so it can't go on forever.

Question 2: Correctness?

- Fix starting edge, every Eulerian circuit can be mapped to a expanded simple graph and vice versa.

Exercise Break: Show that any Eulerian cycle in *Graph* passing through (u, v) and then (v, w) corresponds to an Eulerian cycle (with the same edge labels) in the (u, v, w) -bypass graph passing through (u, x) and then (x, w) .

In general, given an incoming edge (u, v) into v along with k outgoing edges $(v, w_1), \dots, (v, w_k)$ from v , we can construct k different bypass graphs (the figure in the previous step (bottom)). Note that since no two bypass graphs have the same Eulerian cycle, the total number of Eulerian cycles in all k of these bypass graphs is equal to the number of Eulerian cycles in the original graph.

Our idea, roughly stated, is to iteratively construct every possible bypass graph for *Graph* until we obtain a large family of simple directed graphs; each one of these graphs will correspond to a distinct Eulerian cycle in *Graph*. This idea is implemented by the pseudocode in the next step.

AllEulerianCycles(*Graph*)

AllGraphs \leftarrow the set consisting of a single graph *Graph*

while there is a non-simple graph *G* in *AllGraphs*

v \leftarrow a node with indegree larger than 1 in *G*

for each incoming edge (*u*, *v*) into *v*

for each outgoing edge (*v*, *w*) from *v*

NewGraph \leftarrow (*u*, *v*, *w*)-bypass graph of *G*

if *NewGraph* is connected

 add *NewGraph* to *AllGraphs*

 remove *G* from *AllGraphs*

for each graph *G* in *AllGraphs*

 output the (single) Eulerian cycle in *G*

Extra Question...

How many Eulerian Circuits can a graph of M edges have?

Extra Question...

At least $(M/2-1)!$: the star graph

Group Discussion

- In practice, biologists use multiple libraries of read-pairs with different “insert sizes” (distance between ends of the reads). Why might they do this?
- How might we measure the “quality” of a collection of different assemblies?