GENOME ASSEMBLY

* The largest obstacle to sequencing a genome is the fact that biologists lack the technology to read the nucleotides of a genome from beginning to end in the same way one would read a book. The best they can do is sequence much shorter DNA fragments called READS

Traditional Method for Sequencing Genome:

Researchers take a small tissue or blood sample containing millions of cells with identical DNA, use biochemical methods to break the DNA into fragments, and then sequence these fragments to produce READS

Reasons Genome Assembly is difficult

(1) DNA is double-stranded, and there is no way of knowing a priori which strand a given read derives from, meaning that one can't know whether to use ~~it~~ a read or its reverse complement when assembling a particular strand of a genome

(2) Modern sequencing machines are not perfect, and the reads that they generate often contain errors. These errors complicate genome assembly, as they prevent identification of all overlapping reads

(3) Some regions of the genome may not be covered by any reads, making it impossible to reconstruct the entire genome

Reconstruct a String with this Composition:

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA

TGC TGG TGT

TAA

AAT

ATG

TGC (without looking ahead)

GCC

CCA

CAT

ATG

TGG ← GGG can be here

GGA

GAT

ATG

TGT

GTT

⇒ TAATGCCATGGATGTT

Repeats in the Human Genome

* Repeats complicate genome assembly
* In the construction I solved in the previous page, it is obvious I used 14 reads instead of the whole 15 reads. This is due to the fact that ATG appears 3 times, which caused me to have 3 choices: TGG, TGC and TGT
* With millions of reads, repeats make it much more difficult to 'look ahead' and construct the genome assembly
* Approximately 50% of the human genome is made up of REPEATS e.g. the approximately 300 nucleotide-long Alu Sequence is repeated over a million times, with only a few nucleotides inserted | deleted | substituted each time

* A TRANSPOSON is a DNA fragment that can change its position within the genome often resulting in a duplication (repeat). A transposon that inserts itself into a gene will most likely disable that gene. Diseases that are caused by transposons include HEMOPHILIA, PORPHYRIA, DUCHENNE MUSCULAR DYSTROPHY, and many others
* TRANSPOSONS make up a large fraction of the human genome and are divided into 2 classes according to their MECHANISM OF TRANSPOSITION, which can be described as either RETRO-TRANSPOSONS or DNA TRANSPOSONS

⇒ RETROTRANSPOSONS

* They are copied in 2 Stages:

(i) first they are transcribed from DNA to RNA, and the RNA produced is then reverse transcribed to DNA by a REVERSE TRANSCRIPTASE

(ii) This copied DNA fragment is then inserted at a new position into the genome

⇒ DNA TRANSPOSONS

* DNA transposons do not involve an RNA intermediate but instead are catalyzed by TRANSPOSASES
* The transposase cuts out the DNA transposon, which is inserted into a new site in the genome, resulting in a repeat

* Transposons make up a large fraction of the genome and are responsible for much of the mass of DNA in a eukaryotic cell

String Spelled by a Genome Path Problem

* Reconstruct a String from its genome path

Input: A Sequence path of K-mers $Pattern_1, \ldots, Pattern_n$ such that the last $(K-1)$ symbols of $Pattern_i$ are equal to the first $(K-1)$ symbols of $Pattern_{i+1}$ for $1 \leq i \leq n-1$

Output: A String Text of length $K+n-1$ such that the i-th K-mer in Text is equal to $Pattern_i$ (for $1 \leq i \leq n$)

Let Prefix and Suffix denote the first (k-1) nucleotides and last (k-1) nucleotides of a k-mer respectively.

For example, Prefix (TAA) = TA and Suffix (TAA) = AA

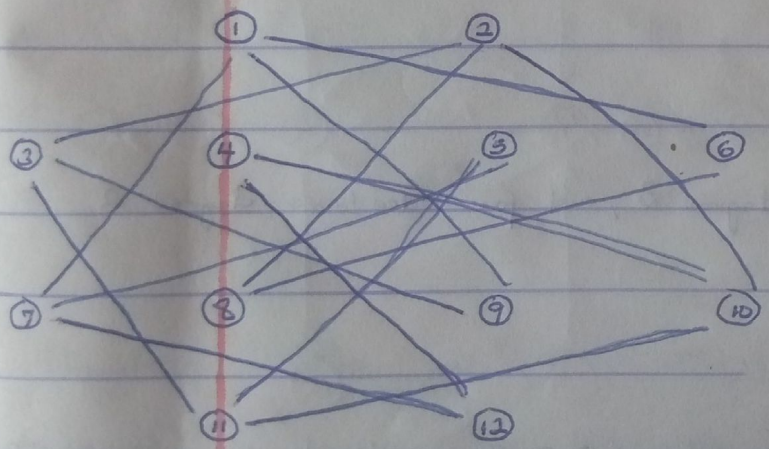
It is important to note that the Suffix of a 3-mer in the genome path is equal to the prefix of the following 3-mer in the path. For example,

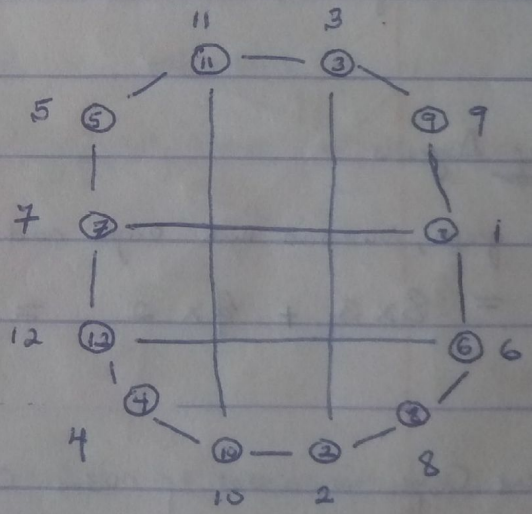
Suffix (TAA) = Prefix (AAT) = AA in the genome path for TAATGCCATGGGATGTT

⇒

TAA → AAT → ATG → TGC → GCC → CCA → CAT → ATG → TGG → GGG → GGA → GAT → ATG → TGT → GTT

GRAPHS





Node Set: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12

Edge Set:

1-6 1-7 1-9 2-3 2-8 2-10 3-9 3-11

4-10 4-12 5-7 5-11 6-8 6-12 7-12 10-11

Path

A path in a graph is a sequence of edges of edges, where each successive edge begins at the node where the previous edge ends. For example,

The path $8 \rightarrow 6 \rightarrow 1 \rightarrow 9$ starts at node 8 and ends at node 9.

It has 4 nodes and 3 edges

Cycle

Paths that start and end at the same node are referred to as CYCLES

e.g Cycle $3 \rightarrow 2 \rightarrow 10 \rightarrow 11 \rightarrow 3$ starts and ends at node 3 and consists of 4 edges

Degree

The number of edges incident to a given node K is called the degree of K. For example, node 1 has degree 3, node 5 has degree 2

Sum of Degrees

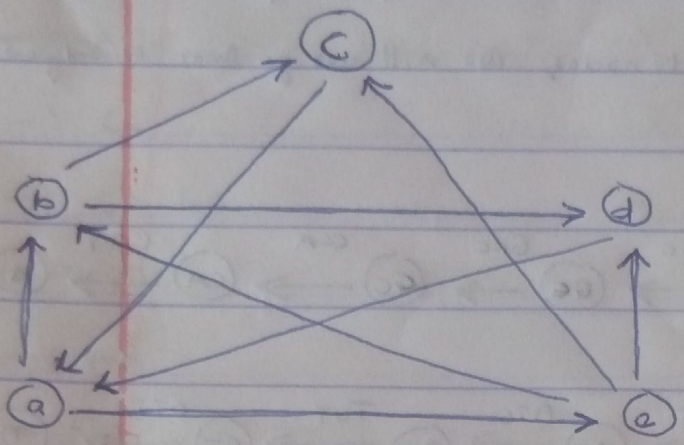
From the graph, we see that eight nodes have degree 3, and four nodes have degree 2

$= 8 \times 3 + 4 \times 2 = 32$ //

⇒ Directed Graph is a set of nodes connected by edges, with each node having a direction associated with it. Edges are usually represented by arrows pointing in the direction the graph can be traversed

⇒ Undirected Graphs can be traversed in any direction, as their edges have no direction associated with them

Graph Representations



a → b, e
b → c, d
c → a
d → a
e → b, c, d

Adjacency Matrix

	a	b	c	d	e
a	0	1	0	0	1
b	0	0	1	1	0
c	1	0	0	0	0
d	1	0	0	0	0
e	0	1	1	1	0

Adjacency List

a is adjacent to b and e
b is adjacent to c and d
c is adjacent to a
d is adjacent to a
e is adjacent to b, c, d

Hamiltonian Paths and Universal Strings

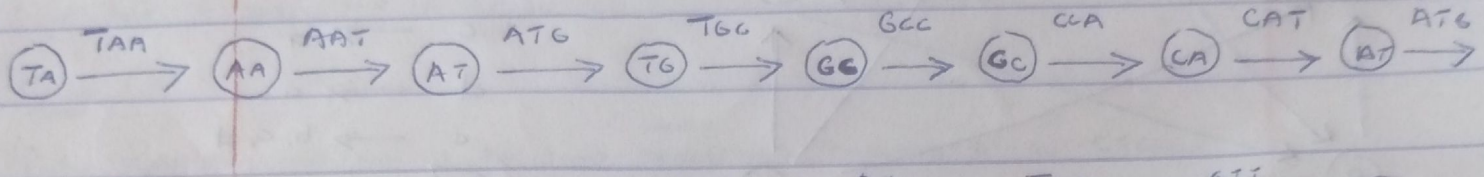
A path in a graph visiting every node once is called a Hamiltonian Path

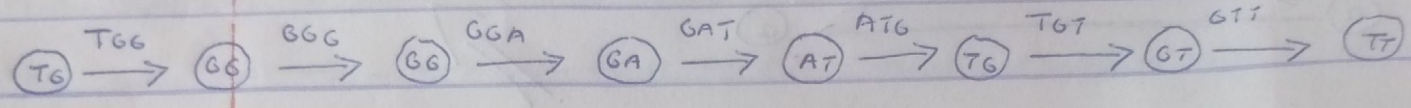
A binary string is a string composed of 0's and 1's; a binary string is k-universal if it contains every binary k-mer exactly once. For example, 0001110100 is a 3-universal string, as it contains each of the eight binary 3-mers (000, 001, 011, 111, 110, 101, 010, 100) exactly once

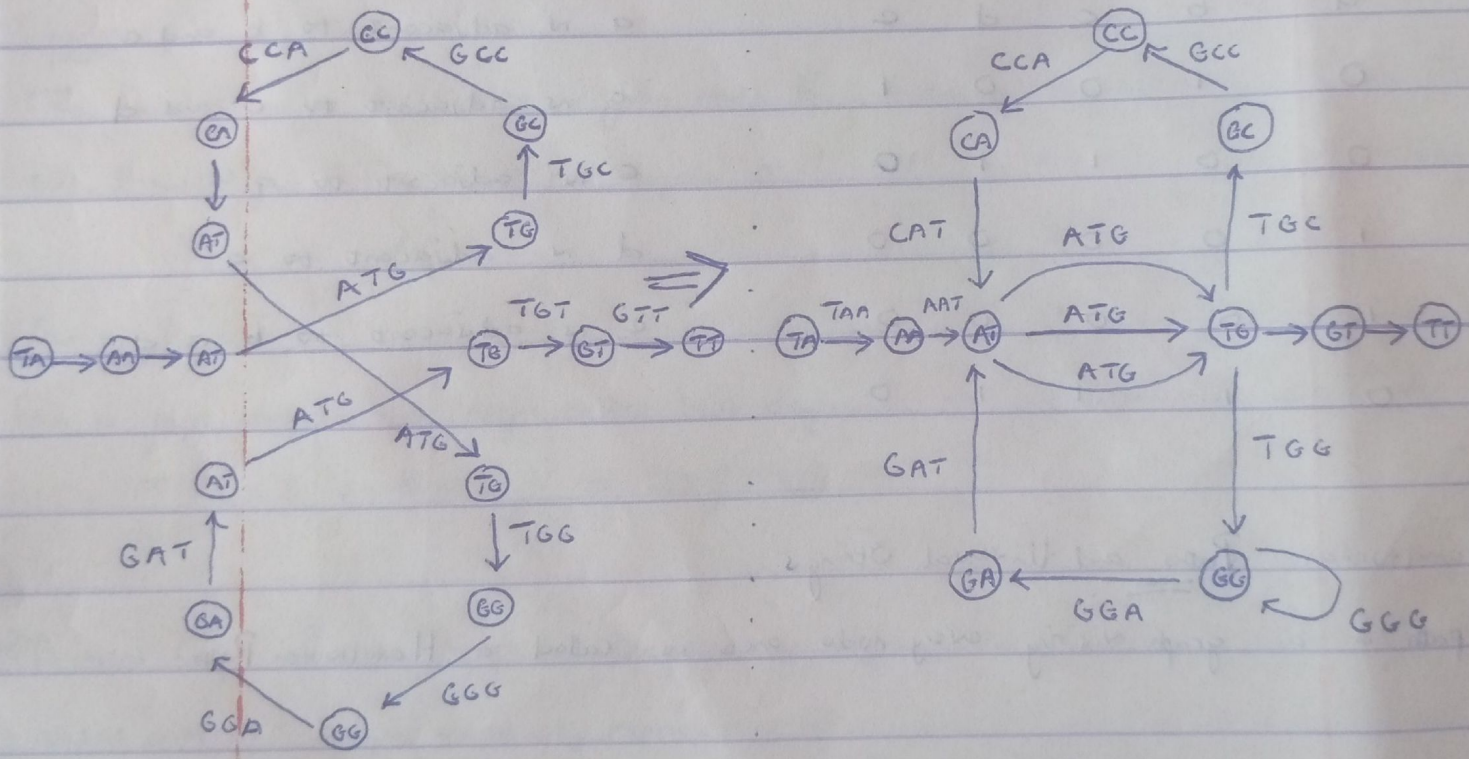
Gluing nodes and de Bruijn Graphs

Represent the genome TAATGCCATGGGATGTT as a sequence of its 3-mers.

This time, instead of assigning these 3-mers to nodes, we will assign them to edges:







16 nodes

11 nodes

Number of edges is the same in both graphs

TAATGCCATGGGATGTT

DeBruijn (TAATGCCATGGGATGTT)

This is the DeBruijn graph of TAATGCCATGGG ATGTT

From DeBruijn$_4$(TAATGCCATGGGATGTT), we have:

AA ⟶ AT
AT ⟶ TG, TG, TG
CA ⟶ AT
CC ⟶ CA
GA ⟶ AT
GC ⟶ CC
GG ⟶ GA, GG
GT ⟶ TT
TA ⟶ AA
TG ⟶ GC, GG, GT

if node == Prefix(edge):
    node ⟶ Suffix(edge)

De Bruijn Graph from a String Problem:

Construct the de Bruijn graph of a string

Input: An integer k and a string Text

Output: DeBruijn$_k$(Text)

Sample Input:

4
AAGATTCTCTAAGA

Sample Output:

AAG ⟶ AGA, AGA
AGA ⟶ GAT
ATT ⟶ TTC
CTA ⟶ TAA
CTC ⟶ TCT
GAT ⟶ ATT
TAA ⟶ AAG
TCT ⟶ CTA, CTC
TTC ⟶ TCT

EULER and HAMILTONIAN PATHS and CIRCUITS

Euler

* An Euler Path visits every edge once
* An Euler Circuit visits every edge once AND begins and ends at the same vertex

An Euler Circuit is a special type of Euler Path

* An Euler circuit is possible if every vertex has even degree
* An Euler Path is possible, but an Euler Circuit is not possible, if exactly two vertices have odd degree
* No Euler Path is possible if more than vertices have odd degree

Hamiltonian

* A Hamiltonian Path visits every vertex once
* An Hamiltonian Circuit visits every vertex once AND begins and ends at a same vertex

String Reconstruction Problem

Genome Sequencing Problem: Reconstruct a genome from reads

Input: A collection of strings Reads

Output: A string Genome reconstructed from Reads

The K-mer Composition of a string is the set of K-mers in that particular string

e.g Composition$_3$ (TAATGCCATGGGATGTT)

= TAA AAT ATG TGC GCC CCA CAT ATG TGG GGG GGA GAT ATG TGT GTT

The K-mer Composition should be arranged in LEXICOGRAPHIC ORDER.

The Genome Sequencing problem is intuitive and very easy to understand. However, it is not a computational problem. We need to redefine the problem in clear terms.

Modified Genome Sequencing Problem: Reconstruct a genome from its K-mer Composition

Input: A collection of K-mers

Output: A string Genome such that Composition$_K$ (Genome) is equal to the collection of K-mers

The Modified Genome Sequencing Problem is now a defined problem for Computational approach

A naive way to approach the Modified Genome Sequencing Problem is to study ~~the Composition~~ Composition$_k$ (Genome), arrange them to form the genome i.e pick each k-mer in the Composition and arrange, each time sorting for the base eg

Composition$_k$ (TAATGCCATGGGATGTT) =

~~TAA~~ AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT

TAA TGC TGG TGT

Arranging:

TAA

AAT

ATG

TGC

GCC ⇒ TAATGCCATGGGATGTT

CCA

CAT

ATG

TGG

GGG

GGA

GAT

ATG

ATG

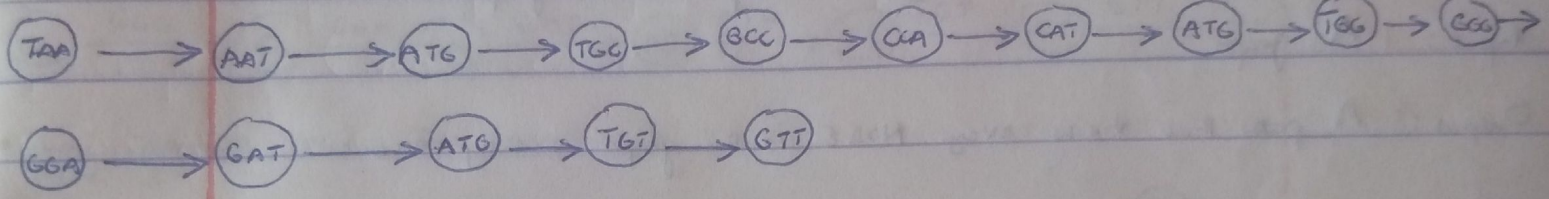
TGT

GTT

Genome Assembly as a Hamiltonian Path Problem

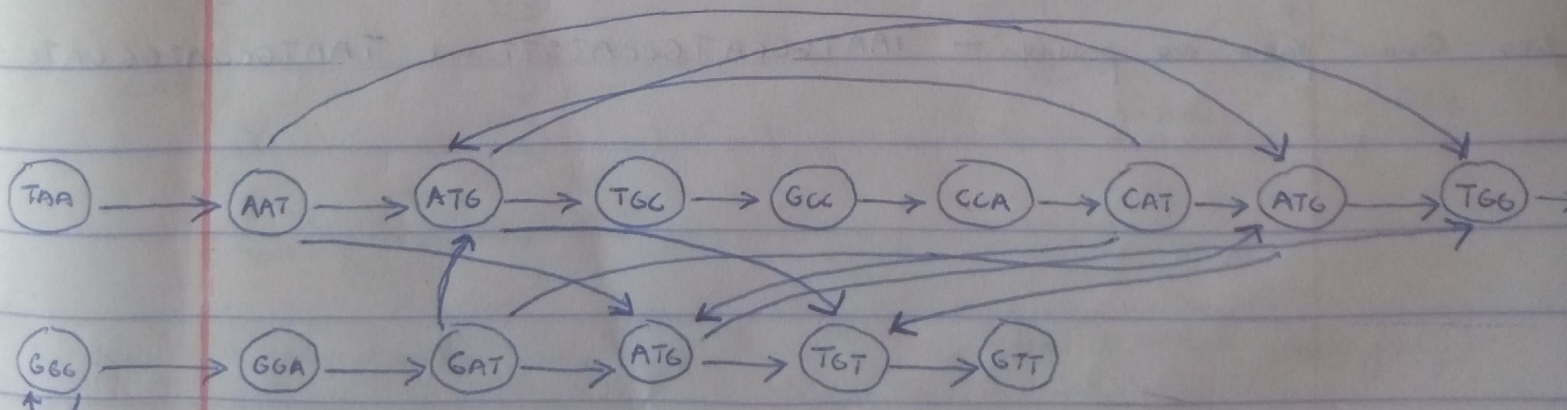
If we have a Genome with its 3-mer Composition, which is a set of all 3-mer generated from the string, if we represent each k-mer as a node, then the Genome is a path through all the nodes i.e

Composition$_3$ (TAATGCCATGGGATGTT) =



Here arises the MILLION DOLLAR QUESTION: Can we Construct this genome path without knowing the genome TAATGCCATGGGATGTT, only from its Composition?

The answer is YES. We simply need to Connects k-mer$_1$ with k-mer$_2$ if Suffix(k-mer$_1$) = prefix(k-mer$_2$) e.g TAA ⟶ AAT



The path turns into a Graph

Now what if we had only the Composition of the genome, without knowing the genome itself?

In this case, we arrange the nodes from left to right ~~not arrow~~ in LEXICOGRAPHIC ORDER

In the end, we end up with a big graph and try to find a PATH that visits every NODE in the graph just once. This path is the HAMILTONIAN PATH

Hamiltonian Path Problem: Find a Hamiltonian path in a graph

Input: A graph

Output: A path that visits every NODE in the graph exactly once

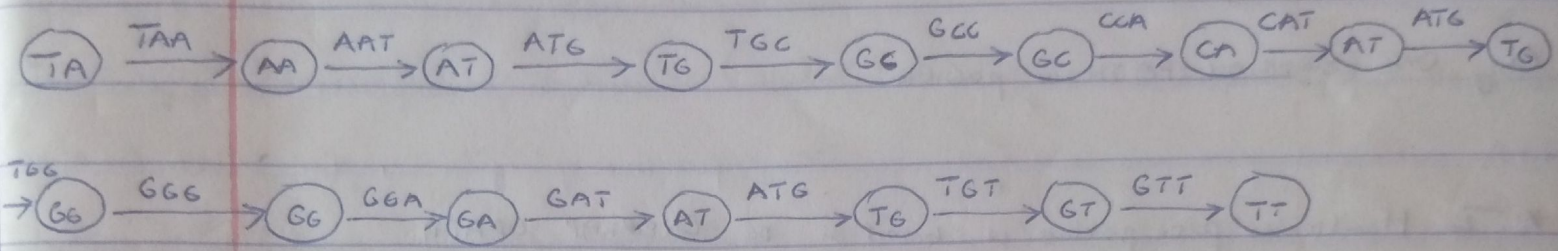
It is important to note that it is a very possible to get more than one Hamiltonian path in a genome graph. For instance in our Composition =

AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

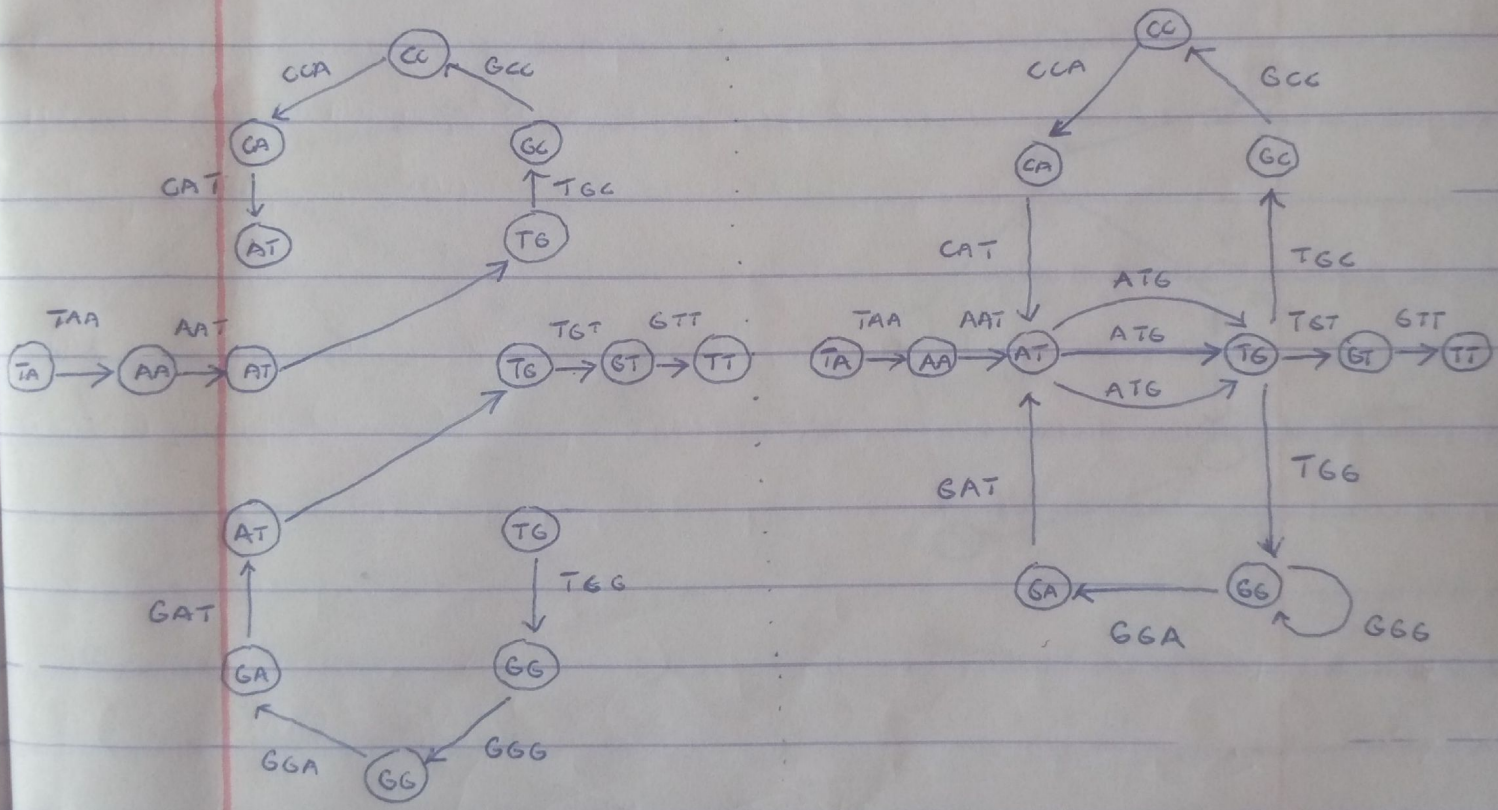
two Genome paths are possible — TAATGGGATGCCATGTT and TAATGCCATGGGATGTT

Genome Assembly as a Eulerian Path Problem

Here, we label every k-mer in the composition of a genome as the edges of a graph. For instance, TAATGCCATGGATGTT will be:



Gluing identical nodes we have:



Has 16 nodes (before gluing)

Has 11 nodes (after gluing)

When is the graph in the Genome hiding?

Eulerian Path Problem : Find an Eulerian path in a graph

Input : A graph

Output: A path visiting every edge in the graph exactly once

Only ONE EULERIAN PATH is possible

* The Hamiltonian path problem is based on the OVERLAP GRAPH
* The EULER PATH PROBLEM is based on the DEBRUIJN GRAPH

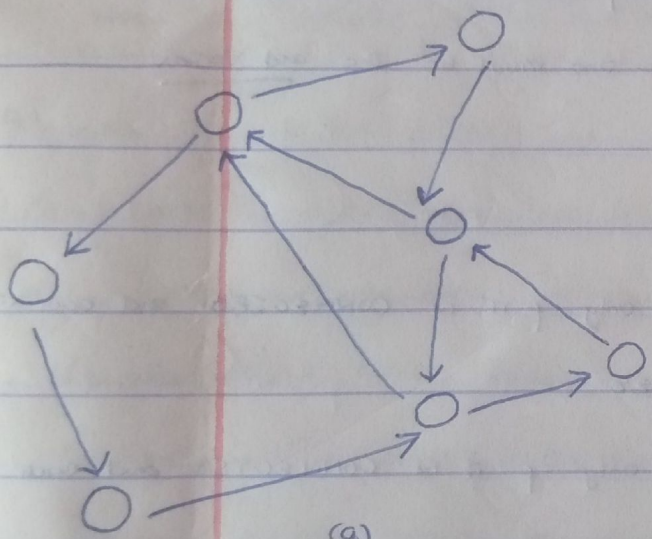
Euler Theorem

Consider an ant walking along the edges of an Eulerian cycle. Every time the ant enters a node of this graph by an edge, he is able to leave this node by another, unused edge.

Thus, in order for a graph to be Eulerian, the number of incoming edges at any node must be equal to the number of outgoing edges at that node.

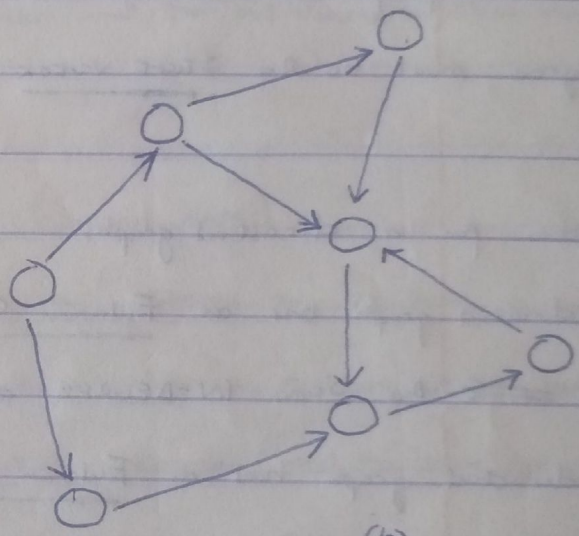
The indegree and outdegree of a node v (denoted $in(v)$ and $out(v)$, respectively) is defined as the number of edges leading into and out of v.

A node v is balanced if $in(v) = out(v)$, and a graph is balanced if all its nodes are balanced. Any Eulerian graph must be balanced!



(a)

Balanced Graph



(b)

Unbalanced Graph

A Strongly Connected graph is a graph in which from each node, a path exists to get to every other node, no matter the length of that path.

A disconnected graph is a graph in which some nodes cannot be reached from other nodes

Euler's Theorem States "Every balanced, strongly connected directed graph is Eulerian"

Task: Given an undirected or a directed graph, find a path or circuit that passes through each edge exactly once

Solution

Conditions for an undirected graph:

(1) An undirected graph has an Eulerian Circuit if and only if it is CONNECTED and each vertex has an EVEN DEGREE

(2) An undirected graph has an Eulerian Path if and only if it is CONNECTED and all vertices except 2 have an even degree. One of those 2 vertices that have an odd degree must be the Start vertex, and the other one must be the end vertex

Condition for a directed graph:

(1) A directed graph has an Eulerian Circuit if and only if it is CONNECTED and each vertex has the same IN-DEGREE as OUT-DEGREE

(2) A directed graph has an Eulerian Path if and only if it is CONNECTED and each vertex except 2 have the same in-degree as out-degree, and one of those 2 vertices has out-degree with one greater than in-degree (this is the Start vertex), and the other vertex has in-degree with one greater than out-degree (this is the end vertex)

Algorithm for undirected graphs:

1) Start with an empty stack and an empty circuit (eulerian path)
   * if all vertices have even degree, choose any of them
   * if there are exactly 2 vertices having an odd degree, choose one of them
   * Otherwise, no Euler circuit or path exists
2) If the current vertex has no neighbors — add it to circuit, remove the last vertex from the stack and set it as the current one. Otherwise (in case it has neighbors) — add the vertex to the stack, take any of its neighbors, remove the edge between selected neighbor and vertex, and set that neighbor as the current vertex
3) Repeat step 2 until the current vertex has no more neighbors and the stack is empty
   Note that the obtained circuit will be in reverse order — from end vertex to start vertex

Algorithm for directed graphs:

(1) Start with an empty stack and an empty circuit (eulerian path)
   * if all vertices have same out-degrees as in-degrees, choose any of them
   * if all but 2 vertices have same out-degree as in-degree, and any one of those 2 vertices has out-degree with one greater than its in-degree, and the other has in-degree with one greater than its out-degree — then choose the vertex that has its out-degree with one greater than its in-degree
   * Otherwise, no Euler circuit or path exists
2) If the current vertex has no outgoing edges (i.e neighbors) — add it to circuit, remove the last vertex from the stack and set it as the current one. Otherwise (in case it has out-going edges, i.e neighbors) — add the vertex to the stack, take any of its neighbors, remove the edge between that vertex and selected neighbor, and set that neighbor as the

Current Vertex

3) Repeat step 2 until the current vertex has no more out-going edges (neighbors) and the stack is empty)

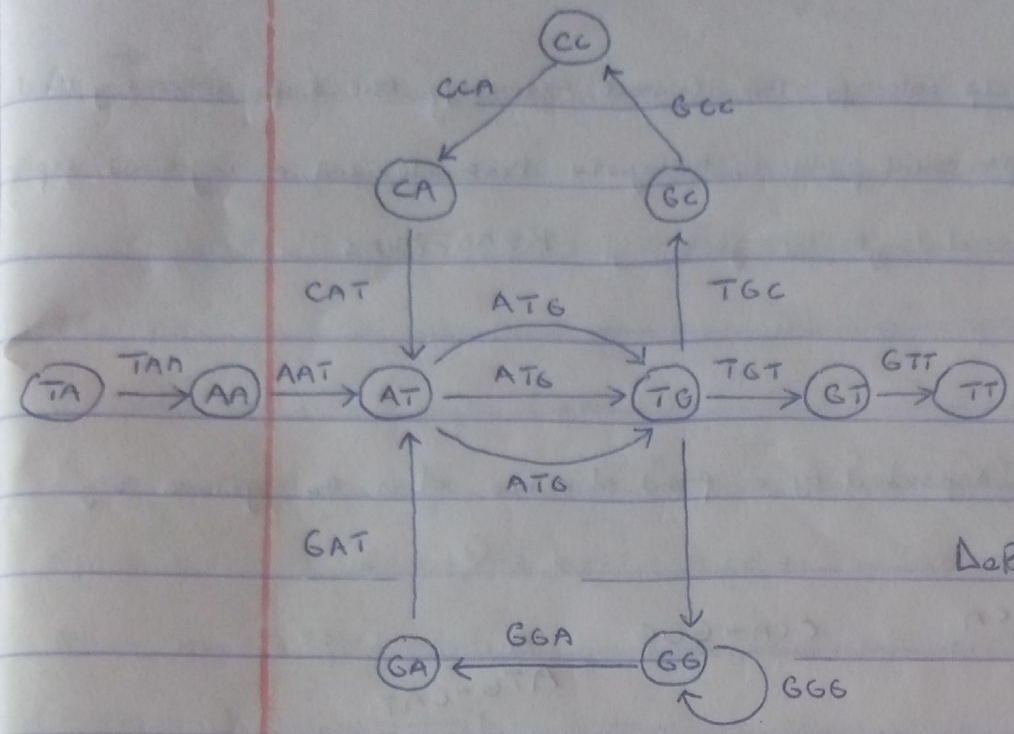
Note that the obtained circuit will be in reverse order — from end vertex to start vertex

The Complexity of both algorithms is $O(N+M)$, where N is the number of vertices and M is the number of edges

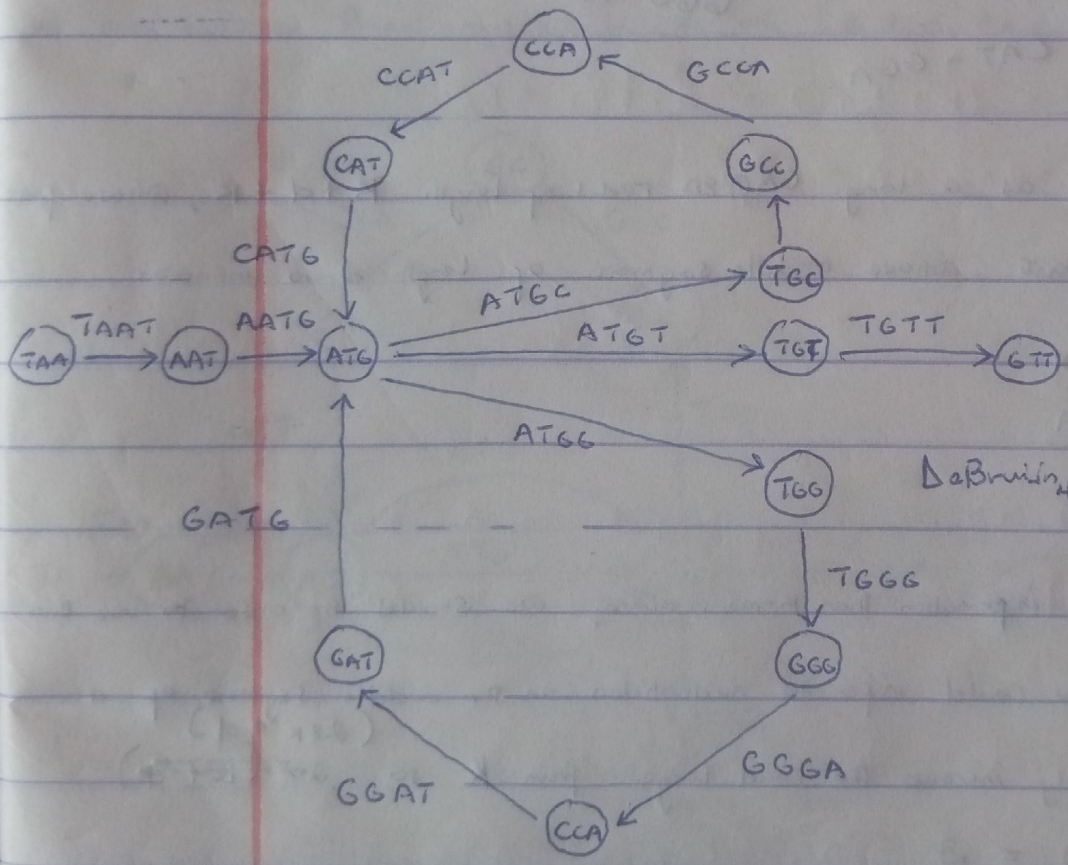
From Reads to Read-Pairs

* Assembling reads sampled from a randomly generated text is trivial, since random strings are not expected to have long repeats
* de Bruijn graphs become less and less tangled when read length (i.e. the k-mer size) increases
* As soon as read length exceeds the length of all repeats in a genome (provided the reads have no errors), the de Bruijn graph turns into a path
* Technically, k-mer length is able to increase if read length is able to increase — but it is technically the fact that the k-mers are getting longer that directly make the graph less tangled
* Biologists have not yet figured out how to generate long and accurate reads. The most accurate sequencing technologies available today generate reads that are only about 300 nucleotides long

De Bruijn graphs become less tangled as the length of the k-mers increases

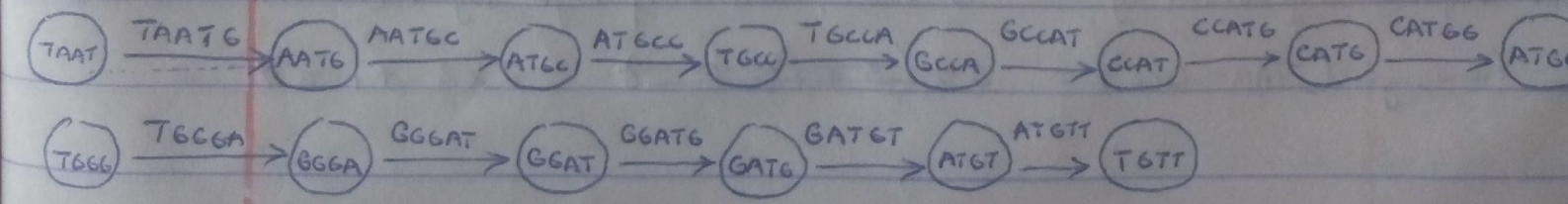


$DeBruijn_3$ (TAATGCCATGGGATGTT)



$DeBruijn_4$ (TAATGCCATGGGATGTT)

$DeBruijn_5$ (TAATGCCATGGGATGTT)



* Increasing read length would help identify the correct assembly, but since increasing read length presents a difficult experimental problem, biologists have devised an ingenious experimental approach to increase read length by generating READ-PAIRS

READ-PAIRS

Read-pairs are pairs of reads separated by a fixed distance d in the genome e.g

TAA - GCC AAT - CCA CCA - GGG ATG - CAT

ATG - GAT GCC - TGG GGG - TGT

GGA - GTT CAT - GGA

* Read pairs can be thought of as a long GAPPED read of length K + d + K, whose first and last K-mers are known but whose middle segment of length d is unknown

K K

AAT — CCA

d

* Since read-pairs contain more information than K-mers alone, we should be able to use them to improve assemblies. If you could infer the nucleotides in the middle segment of a read-pair, you would increase the read length from K to (2*K + d) ~~2 * (K + d)~~

⇒ ReadLength (TAA — GCC) = 3

if the sequence represented by — is known, and is of length, say 1 (i.e just a nucleotide),

ReadLength (TAA - GCC) = ~~2 x (3 + 1) = 8~~ 2 x 3 + 1 = 7

* Increasing read length would help identify the correct assembly, but since increasing read length presents a difficult experimental problem, biologists have devised an ingenious experimental approach to increase read length by generating READ-PAIRS

READ-PAIRS

Read-pairs are pairs of reads separated by a fixed distance d in the genome e.g

TAA - GCC AAT - CCA CCA - GGG ATG - CAT

ATG - GAT GCC - TGG GGG - TGT

GGA - GTT CAT - GGA

* Read pairs can be thought of as a long GAPPED read of length K + d + K, whose first and last K-mers are known but whose middle segment of length d is unknown

K K

AAT — CCA

d

* Since read-pairs contain more information than K-mers alone, we should be able to use them to improve assemblies. If you could infer the nucleotides in the middle segment of a read-pair, you would increase the read length from K to (2*K + d) ~~2 * (K + d)~~

⇒ ReadLength (TAA — GCC) = 3

if the sequence represented by — is known, and is of length, say 1 (i.e just a nucleotide),

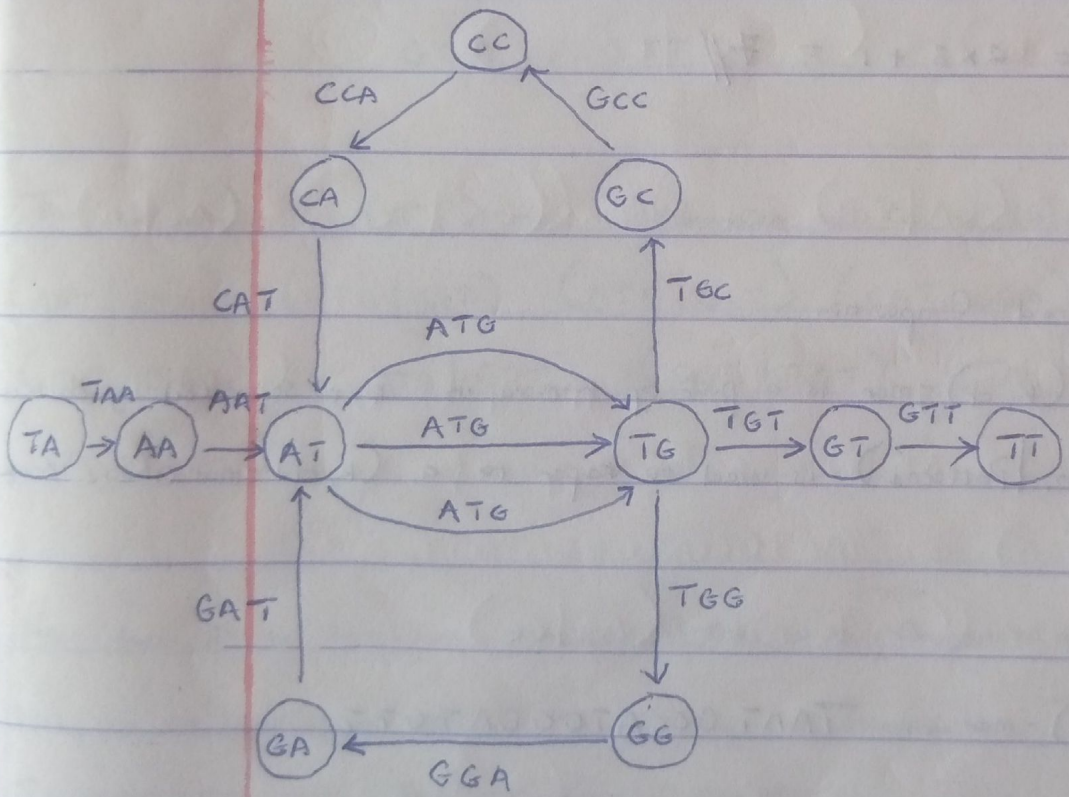
ReadLength (TAA - GCC) = ~~2 x (3 + 1) = 8~~ 2 x 3 + 1 = 7

Transforming read-pairs into long virtual reads

* A read-pair formed by K-mer reads $Read_1$ and $Read_2$ corresponds to two edges in the de Bruijn graph $DeBruijn_K(Reads)$

* The total length of a read-pair is $(2K + d)$ — This means the read-pair is overall a whole read of length $(2K+d)$.

Since a read of length n has $(n-K+1)$ number of K-mers obtainable from it, we can say that $((2K+d)-K+1) = K+d+1$ number of K-mers are obtainable from the whole read. In the de Bruijn graph, this means that between $Read_1$ and $Read_2$, there are $(K+d+1)$ edges — (Since K-mers are edges in the de Bruijn graph), making the length of path between $Read_1$ and $Read_2$ to be $(K+d+1)$



In the de Bruijn graph in the previous page, all reads are generated from the previous step. Let's Consider AAT-CCA

Since AAT and CCA are a read-pair, each read can be represented by the edge of the de-Bruijn graph. Moving from AAT to CCA, we have:

AAT

ATG ⇒ AATGCCA

TGC This shows that the nucleotide G is the nucleotide present

GCC in the distance d between AAT and CCA

CCA

AATGCCA

(1) Length of path between reads = (k+d+1) = 3+1+1 = 5//

(2) ReadLength (AATGCCA) = 2×3+1 = 7//

From Composition to Paired Composition

Given a string Text, a (k, d)-mer is a pair of k-mers in Text separated by distance d. The notation (Pattern$_1$|Pattern$_2$) is used to refer to a (k, d)-mer whose k-mers are Pattern$_1$ and Pattern$_2$.

For example,

(AAT|TGG) is a (3, 4)-mer in TAATGCCATGGGATGTT

The (k, d)-mer Composition of Text, denoted PairedComposition$_{k,d}$(Text), is the collection of all (k, d)-mers in Text (including (k, d)-mers).

For example, below is Paired Composition$_{3,1}$ (TAATGCCATGGGATGTT):

TAA GCC

AAT CCA

ATG CAT

TGC ATG

GCC TGG

CCA GGG

CAT GGA

ATG GAT

TGG ATG

GGG TGT

GGA GTT

Length of path between readv = 3+1+1 = 5

Upper part: TAATGCCATGGGA

Lower part: GCCATGGGATGTT

~~AA/TAATGCCATGGCATGTT~~

We join the upper and lower part at the 5th nucleotide

TAATGCCATGGGA

GCCATGGGATGTT

⇒ TAATGCCATGGGATGTT

The order of the (3,1)-mers is unknown, and can be listed in lexicographic order:

(AAT|CCA) (ATG|CAT) (ATG|GAT) (CAT|GGA) (CCA|GGG)
(GCC|TGG) (GGA|GTT) (GGG|TGT) (TAA|GCC) (TGC|ATG)
(TGG|ATG)

Note that while there are repeated 3-mers in the 3-mer Composition of TAATGCCATGGGATGTT, there are no repeated (3,1)-mers in its paired composition. Furthermore, although TAATGCCATGGGATGTT and TAATGGGATGCCATGTT have the same 3-mer composition, they have different (3,1)-mer compositions.

Thus, if we can generate the (3,1)-mer composition of these two strings, we will be able to

distinguish between them. This brings us to the String Reconstruction from Read-Pairs Problem

String Reconstruction from Read-Pairs Problem: Reconstruct a string from its paired composition

Input: A collection of paired k-mers PairedReads and an integer d

Output: A string Text with (k, d)-mer composition equal to PairedReads (if such a string exists)

Paired de Bruijn Graphs

Given a (k, d)-mer $(a_1 \dots a_k | b_1 \dots b_k)$, we define Prefix and Suffix as the following (k-1, d+1)-mers:

$\text{PREFIX}((a_1 \dots a_k | b_1 \dots b_k)) = (a_1 \dots a_{k-1} | b_1 \dots b_{k-1})$

$\text{SUFFIX}((a_1 \dots a_k | b_1 \dots b_k)) = (a_2 \dots a_k | b_2 \dots b_k)$

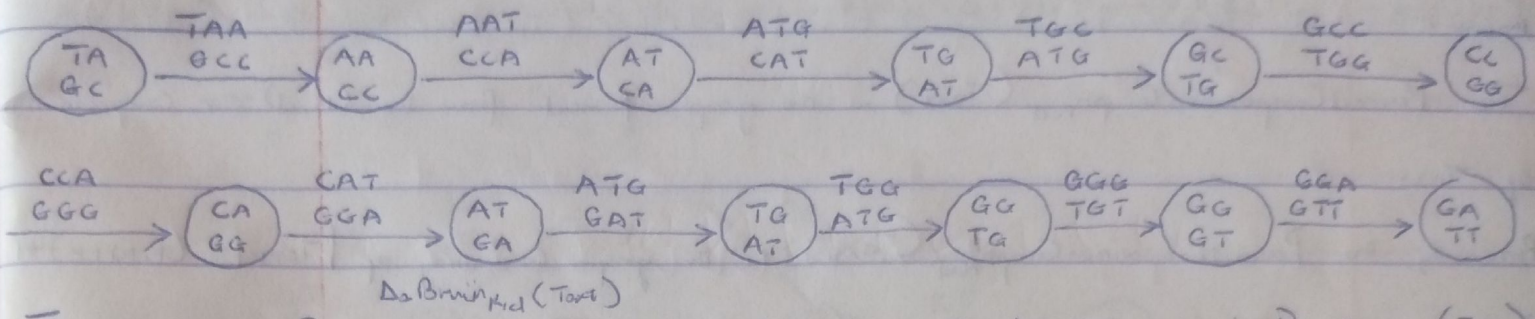
e.g. $\text{PREFIX}((GAC|TCA)) = (GA|TC)$ and $\text{SUFFIX}((GAC|TCA)) = (AC|CA)$

For consecutive (k, d)-mers appearing in Text, the Suffix of the first (k, d)-mer is equal to the prefix of the second (k, d)-mer. For example, for the consecutive (k, d)-mers (TAA|GCC) and (AAT|CCA) in TAATGCCATGGGATGTT,

$\text{SUFFIX}((TAA|GCC)) = \text{PREFIX}((AAT|CCA)) = (AA|CC)$

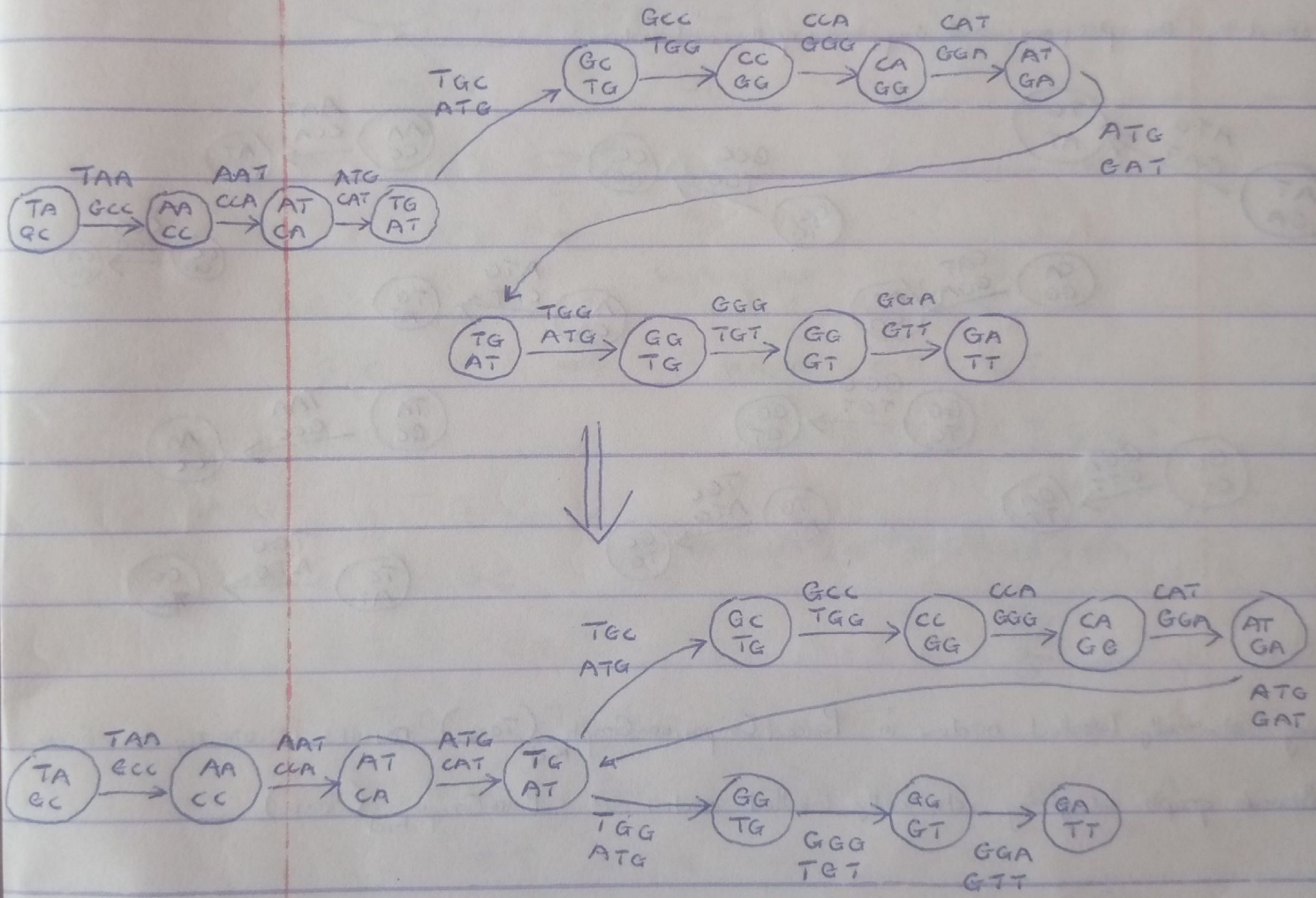
Given a string Text, we construct a graph $\text{PathGraph}_{k,d}(\text{Text})$ that represents a path formed by $|\text{Text}| - (k+d+k)+1$ edges corresponding to all (k, d)-mers in Text. We label edges in this path by (k, d)-mers and label the starting and ending nodes of an edge

by its prefix and suffix, respectively. For example, $PathGraph_{3,1}(TAATGCCATGGGATGTT)$:



$DeBruijn_{k=3}(Text)$

The paired de Bruijn graph is formed by gluing identically labelled nodes in $PathGraph_{k,d}(Text)$:

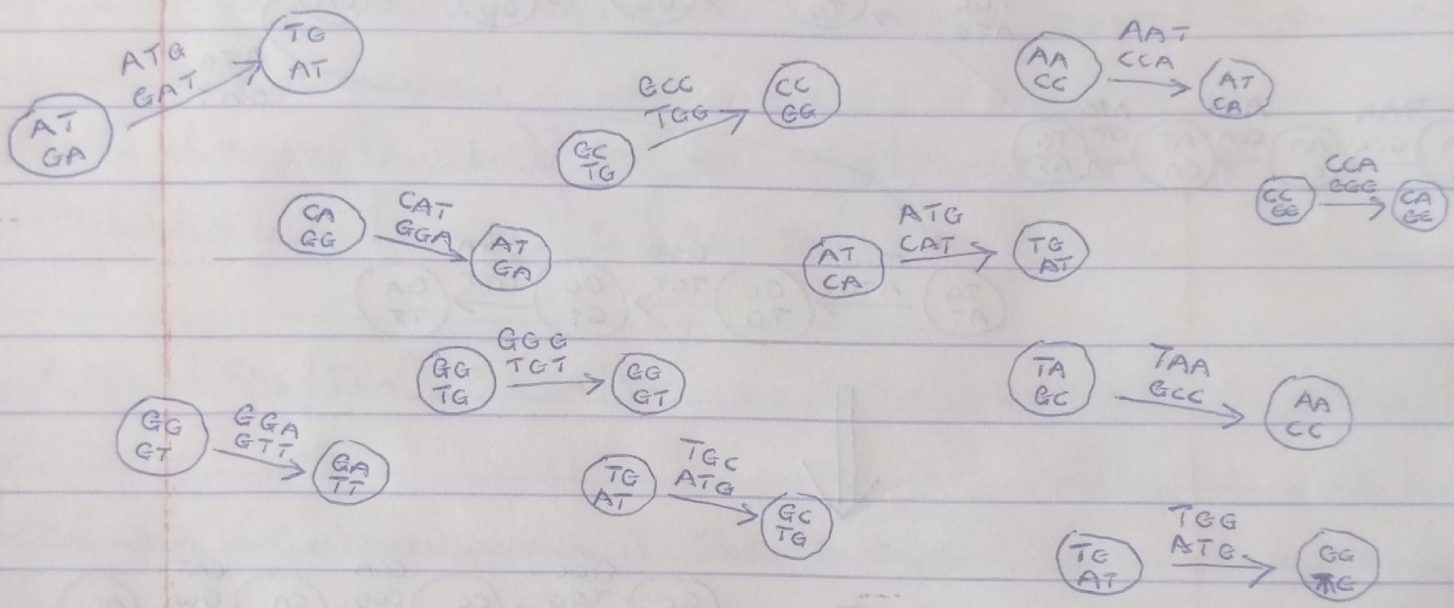


Paired Composition Graph

In practice, Text is not known. However, the (k, d)-mer composition of Text is always known.

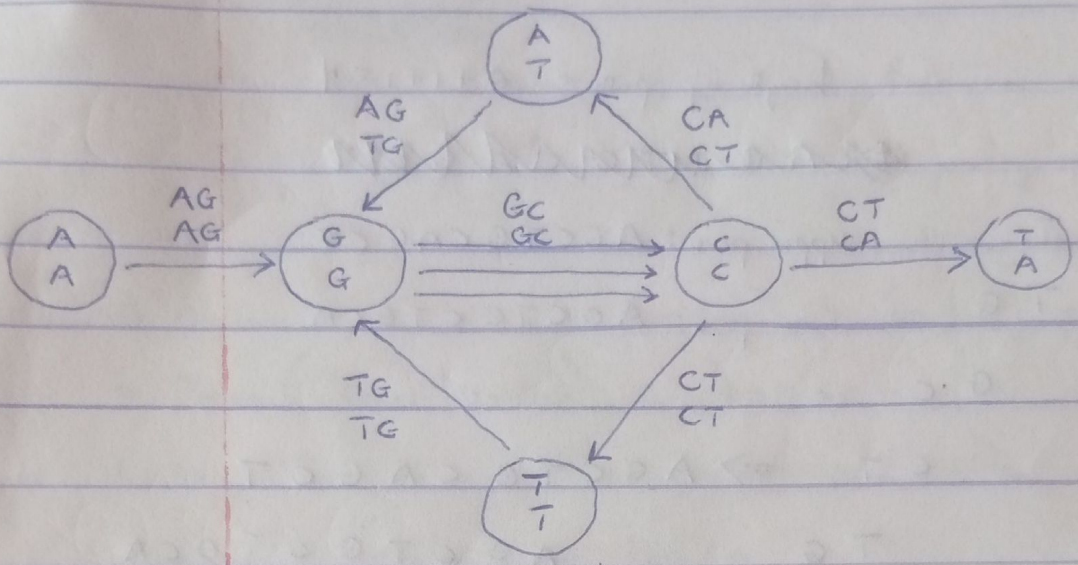
The $\text{PairedCompositionGraph}_{k,d}(\text{Text})$ is the graph formed from the (k, d)-mer composition of Text.

Basically, the $\text{PairedCompositionGraph}_{k,d}(\text{Text})$ is the graph consisting of $|\text{Text}| - (k+d+k)$ isolated edges that are labelled by the (k, d)-mers in Text, and whose nodes are labelled by the prefixes and suffixes of those labels.



Gluing identically labelled nodes in $\text{PairedCompositionGraph}_{k,d}(\text{Text})$ results in exactly the same de Bruijn graph as gluing identically labelled nodes in $\text{PathGraph}_{k,d}(\text{Text})$

Exercise: In the paired de Bruijn graph below, reconstruct the genome spelled by the following Eulerian path of (2,1)-mers: (AG|AG) → (GC|GC) → (CA|CT) → (AG|TG) → (GC|GC) → (CT|CT) → (TG|TG) → (GC|GC) → (CT|CA)



length of path = 2+1+1 = 4

AG AG

GC GC ⇒ AGCAGCTGCT

CA CT AGCTGCTGCA

AG TG

GC GC

CT CT

TG TG

GC GC ⇒ AGCAGCTGCTGCA

CT CA

This is a Valid Eulerian path that gives a valid Genome

In the de Bruijn graph shown in the previous page, reconstruct the genome spelled by the Eulerian path: (AG|AG) → (GC|GC) → (CT|CT) → (TG|TG) → (GC|GC) → (CT|CT) → (AG|TG) → (GC|GC) → (CT|CA)

of (2,1)-mers

A G A G
 G C G C
  C T C T
   T G T G
    G C G C
     C A C T
      A G T G
       G C G C
        C T C A

length of path = 2+1+1 = 4

upper part: AGCTGCAGCT

Lower part: AGCTGCTGCA

⇒ AGCTGCAGCT

AGCTGCTGCA

Prefix Suffix

In the above exercise, the upper part and lower part do not overlap at position 4 and position 7

AGC ? GC ? GCT GCA

This is a Valid Eulerian path that gives an Invalid genome

IN A NUTSHELL, NOT EVERY EULERIAN PATH IN THE PAIRED DEBRUIJN GRAPH CONSTRUCTED FROM (k, d)-mer COMPOSITION SPELLS OUT A SOLUTION OF THE STRING RECONSTRUCTION FROM READ-PAIRS PROBLEM

String Spelled by a Gapped Genome Path Problem: Reconstruct a sequence of (k, d)-mers corresponding to a path in a paired de Bruijn graph

Input: A sequence of (k, d)-mers $(a_1|b_1), \ldots, (a_n|b_n)$ such that $\text{Suffix}((a_i|b_i)) = \text{Prefix}((a_{i+1}|b_{i+1}))$ for $1 \leq i \leq n-1$

Output: A string Text of length $k+d+k+n-1$ such that the i-th (k, d)-mer of Text is equal to $(a_i|b_i)$ for $1 \leq i \leq n$

GENOME ASSEMBLY FACES REAL SEQUENCING DATA

In reality, De Bruijn Graphs face harsh realities in assembly

Some Ridiculously Unrealistic Assumptions:

(1) Perfect Coverage of genome by reads

(2) Reads are error-free

(3) Multiplicities of K-mers are known

(4) Distances between reads with read-pairs are exact

In reality:

(1) Imperfect Coverage of genome by reads

(2) Reads are error-prone

(3) Multiplicities of K-mers are unknown

(4) Distances between reads within read-pairs are inexact

ANTIBIOTICS

SEQUENCING

ANTIBIOTICS SEQUENCING

HOW DO BACTERIA MAKE ANTIBIOTICS?

How peptides are encoded by the genome

Tyrocidine B1 is one of the many antibiotics produced by Bacillus brevis. Tyrocidine is defined by the 10 amino acid-long sequence:

Val - Lys - Leu - Phe - Pro - Trp - Phe - Asn - Gln - Tyr

V K L F P W F N Q Y

The central Dogma of Molecular Biology states that DNA makes RNA makes protein. A gene from a genome is first transcribed into a strand of RNA composed of Adenine, Cytosine, Guanine and Uracil. The RNA transcript is then translated into an amino acid sequence of a protein.

Transcription simply transforms a DNA string into a RNA string by replacing all occurrences of T with U. The resulting RNA strand is translated into an amino acid sequence as follows:

During translation, the RNA strand is partitioned into non-overlapping 3-mers called CODONS. Then each codon is converted into one of 20 amino acids via the GENETIC CODE — the resulting sequence can be represented as an amino acid string formed over a 20-letter alphabet

For example, the DNA string TATACGAAA transcribes into RNA string UAUACGAAA, which in turn translates into the amino acid Y-T-K

Exercise: How many DNA strings of length 30 transcribe and translate into Tyrocidine B1?

Tyrocidine B1 ⟹ Val → Lys → Leu → Phe → Pro → Trp → Phe → Asn → Gln → Tyr

From the genetic code figure:

Val → V: GUA, GUC, GUG, GUU = 4

Lys → K: AAA, AAG = 2

Leu → L: CUA, CUC, CUG, CUU, UUA, UUG = 6

Phe → F: UUC, UUU = 2

Pro → P: CCA, CCC, CCG, CCU = 4

Trp → W: UGG = 1

Phe → F: UUC, UUU = 2

Asn → N: AAC, AAU = 2

Gln → Q: CAA, CAG = 2

Tyr → Y: UAC, UAU = 2

$= 4 \times 2 \times 6 \times 2 \times 4 \times 1 \times 2 \times 2 \times 2 \times 2 = 6144$ //

A DNA string PATTERN encodes an amino acid string PEPTIDE if the RNA string string transcribed from PATTERN or its reverse Complement $\overline{\text{PATTERN}}$ translated into PEPTIDE

For example,

DNA string GAAACT is transcribed into GAAACU and translated into ET

The reverse Complement AGTTTC is transcribed into AGUUUC and translated into SF

Thus, GAAACT encodes both ET and SF

PEPTIDE ENCODING PROBLEM

(1) Form all possible K-mers from the DNA Text. The K-mers should be of size 3* length of Peptide

(2) For each possible K-mer from Text, develop its ReverseComplement, and translate the K-mer and its reverse Complement into their respective protein translation Protein_1 and Protein_2

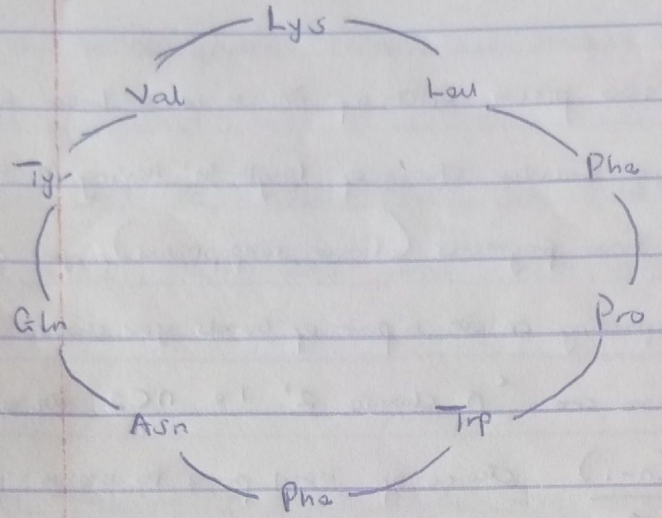
3) Check whether Protein_1 or Protein_2 is same as Peptide

PLASMID

A plasmid is a small, extrachromosomal DNA molecule within a cell that is physically separated from chromosomal DNA and can replicate independently. They are most commonly found as small circular, double-stranded DNA molecules in bacteria; however, plasmids are sometimes present in archea and eukaryotic organisms

From Linear to Cyclic Peptides

Tyrocidines and Gramicidins are actually CYCLIC PEPTIDES. The cyclic representation for Tyrocidine B1 is:



Thus Tyrocidine has ten different linear combinations — and the Peptide Encoding Problem should be run on every one of these sequences to find potential 30-mer coding for Tyrocidine B1. Yet if the Peptide Encoding Problem is solved for each of the ten strings, no 30-mer in the Bacillus brevis genome encodes Tyrocidine B1

The CYCLOPEPTIDE SEQUENCING PROBLEM

Let's assume for simplicity that a mass spectrometer breaks the copies of a cyclic peptide at every possible two bonds, so that the resulting experimental spectrum contains the masses of all possible linear fragments of the peptide, which are called SUBPEPTIDES.

For example, the cyclic peptide NQEL has 12 subpeptides: N, Q, E, L, NQ, QE, EL, LN, NQE, QEL, ELN, and LNQ. We will also assume subpeptides occur more than once if an amino acid occurs multiple times in the peptide ——

e.g ELEL also has 12 subpeptides: E, L, E, L, EL, LE, EL, LE, ELE, LEL, ELE and LEL

Exercise: How many subpeptides does a cyclic peptide of length n have?

A cyclic peptide is fragmented into two pieces, each of which is a linear sub-peptide. A cyclic peptide of length 'n' is simply a circular chain of length 'n' having 'n' bonds or links. To break the circular chain into two linear fragments (linear subpeptides), you need to break any two of the 'n' bonds — and depending on which pair of bonds you choose to break, you will get two fragments. This means there are 'n choose 2' i.e nC_2 which is

$$\frac{n!}{(n-2)!2!} = \frac{n(n-1)}{2}$$ choice of bond pairs to break

Since each choice will result in two different fragments, the total number of different linear sub-peptides are $2 * \frac{n(n-1)}{2} = n(n-1)$ //

Thus a cyclic peptide of length 4 has $4(4-1) = 12$ subpeptides

" " " " length 31315 has $31315(31315-1) = 980,597,910$ subpeptides //

The Theoretical Spectrum

The theoretical spectrum of a cyclic peptide PEPTIDE, denoted Cyclospectrum(Peptide), is the collection of all of the masses of its subpeptides, in addition to the mass 0 and the mass of the entire peptide, with masses ordered from smallest to largest. For NQEL,

	L	N	Q	E	LN	NQ	EL	QE	LNQ	ELN	QEL	NQE	NQEL
0	113	114	128	129	227	242	242	257	355	356	370	371	484

Brute Force Algorithm for Cyclopeptide Sequencing

We denote the total mass of Peptide as Mass(Peptide). In mass spectrometry experiments, where the peptide that generated the Spectrum is unknown, Mass(Peptide) can be inferred from Spectrum and is denoted ParentMass(Spectrum). For simplicity, we assume ParentMass(Spectrum) is equal to the largest mass in Spectrum.

The brute force cyclopeptide sequencing algorithm BFCyclopeptideSequencing generates all possible peptides whose mass is equal to ParentMass(Spectrum) and then checks which of these peptides has theoretical spectra matching Spectrum.

Given an experimental spectrum Spectrum of a cyclic peptide, a linear peptide is Consistent with Spectrum if every mass in its theoretical spectrum is contained in Spectrum. If a mass appears more than once in the theoretical spectrum of the linear peptide, then it must appear at least that many times in Spectrum in order for the linear peptide to be Consistent with Spectrum. For example, a linear peptide can still be Consistent with the theoretical spectrum of NQEL if the peptide's spectrum contains 242 twice. But it cannot be Consistent with the theoretical spectrum of NQEL if its spectrum contains 113 twice.

From theoretical to real Spectra

* A false mass is present in the experimental spectrum but absent from the theoretical spectrum
* A missing mass is present in the theoretical spectrum but absent from the experimental spectrum

Adapting cyclopeptide Sequencing for Spectra with errors

To generalize the cyclopeptide Sequencing problem to handle noisy Spectra, we need to relax the requirement that a candidate's peptide's theoretical spectrum must match the experimental spectrum exactly, and instead incorporate a scoring function that will select the peptide whose theoretical spectrum matches the given experimental spectrum the most closely.

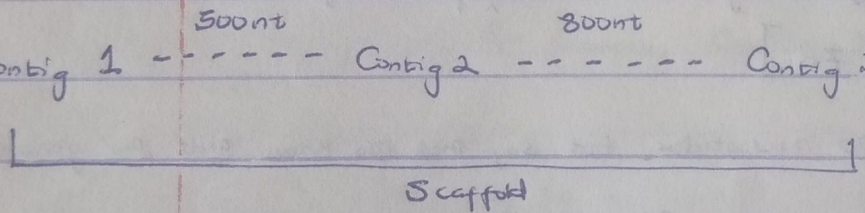
Since no assembly tool is perfect, Biologists need to evaluate the quality of various assemblers by comparing their results.

CONTIG

A Contig is the Contiguous segment of the genome that has been reconstructed by an assembly algorithm

SCAFFOLD

A Scaffold is an ordered sequence of contigs (possibly separated by gaps between them) that are reconstructed by an assembly algorithm. The order of Contigs in a Correctly assembled scaffold Corresponds their order in the genome. Existing assemblers specify the approximate lengths of gaps between Contigs in a scaffold



N50 Statistic

* N50 is a statistic that is used to measure the quality of an assembly
* N50 is defined as the MAXIMAL CONTIG LENGTH FOR WHICH ALL CONTIGS GREATER THAN OR EQUAL TO THAT LENGTH COMPRISE AT LEAST HALF OF THE SUM OF THE LENGTHS OF ALL THE CONTIGS

For example, Consider the 5 toy contigs with the following lengths:

[10, 20, 30, 60, 70]

Here, the total length of the Contigs is 190, and Contigs of length 60 and 70 account for at least 50% of the total length of Contigs (60 + 70 = 130), but the Contig of length 70 does not account for 50% of the total length of Contigs. Thus, N50 is equal to 60.

NG50 Statistic

* the NG50 length is a modified version of N50 that is defined when the length of the genome is known (or can be estimated).
* it is defined as the MAXIMAL CONTIG LENGTH FOR WHICH ALL CONTIGS OF AT LEAST THAT LENGTH COMPRISE AT LEAST HALF OF THE LENGTH OF THE GENOME
* NG50 allows for meaningful comparision between different assemblies for the same genome

For example, Consider the 5 toy Contigs which were previously Considered:

[10, 20, 30, 60, 70]

These Contigs only add to 190 nucleotides, but say that we know that the genome from which they have been generated has length 300.

In this example, the Contigs of length 30, 60, and 70 account for at least 50% of the genome length (30 + 60 + 70 = 160); but the Contigs of length 60 and 70 no longer account for at least 50% of the genome length (60 + 70 = 130). Thus NG50 is equal to 30.

NGA50 Statistic

* If the reference genome for a specie is known, then we can test the accuracy of a newly assembled genome against this reference
* The NGA50 statistic is a modified version of NG50 accounting for assembly errors (called misassemblies)
* To compute NGA50, errors in the contigs are accounted for by comparing contigs to a reference genome. All of the misassembled contigs are broken at misassembly breakpoints, resulting in a larger number of contigs with the same total length. For example, if there is a misassembly breakpoint at position 10 in a contig of length 30, this contig will be broken into contigs of length 10 and 20

NGA50 is calculated as the NG50 statistic for the set of contigs resulting after breaking at misassembly breakpoints. For example, consider the previous example, for which the length is 300.

If the largest contig in [10, 20, 30, 60, 70] is broken into two contigs of length 20 and 50 (resulting in the set of contigs [10, 20, 20, 30, 50, 60]), then contigs of length 20, 30, 50, and 60 account for at least 50% of the genome length (20+30+50+60 = 160). But contigs of length 30, 50, and 60 do not account for at least 50% of the genome length (30 + 50 + 60 = 140). Thus, NGA50 is equal to 20

* Biologists are mainly interested in long contigs (i.e contigs with length $\geq$ 1000 nucleotides) and often discard short contigs —— This is because short contigs often harbor only fragments of genes rather than complete genes

* The 'total length of long contigs' is a statistic that can be combined with N50 and the number of long contigs.

$\Rightarrow$ A good assembly is one that has relatively few long contigs, but the total length of long contigs is high, as is N50.