ISSN: 0975-833X

Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 13, Issue, 02, pp.16407-16419, February, 2021

DOI: https://doi.org/10.24941/ijcr.40519.02.2021

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

RESEARCH ARTICLE

GENOME RECONSTRUCTION USING GRAPH THEORY

***Aakanksha Mahajan**

United States

ARTICLE INFO

Article History:
Received 07th November, 2020
Received in revised form
15th December, 2020
Accepted 18th January, 2021
Published online 28th February, 2021

Key Words:

Genome Reconstruction,
Graph Theory, Bioinformatics.

ABSTRACT

With a given list of k-mers, we are attempting to reconstruct a genome whose k-mers are the same as the input list of k-mers. This will require the development of an algorithm. We will be exploring multiple methods, such as inspection, but we will ultimately be using graph theory. This will be used by, first, creating a graph of the given k-mers. We will then discuss the process of finding a Eulerian path from this constructed graph, leading to the final goal, a reconstructed genome.

Citation: *Aakanksha Mahajan.* "Genome Reconstruction using Graph Theory", *International Journal of Current Research*, 13, (02), 16407-16419.

1.INTRODUCTION

Reading entire genomes of DNA is a monumental task, and current technology cannot read sequences of that length. As this technology can read sequences of shorter length, genomes are broken into smaller sections using restriction enzymes (5). These smaller sections can be read. Using the reads of these smaller sections, our goal is to reconstruct the full genome. To solve this problem, we will be describing an algorithm using graph theory. Using the basics of graph theory, we will be creating a graph of the list of k-mers which will be used in finding the final reconstructed genome (2).

We refer the reader to (4) for more details on a similar topic. This article is organized as follows. We explain DNA in Section 2. The reason the problem we are trying to solve exists is explained in Section 3. One solution to the problem is shown in Section 4. In Section 5, we describe the computation algorithm for genome reconstruction. We explain the problem at hand in more detail in Section 6. We describe the basics of graph theory in Section 7. In Section 8, we describe the methodology used to construct the graph of a list of k-mers. We describe paths in a graph of a list of k-mers in Section 9. In Section 10, we explain Eulerian paths and the conditions for their existence. We describe the algorithm to finding a Eulerian cycle in Section 11, and in Section 12, we describe the algorithm used to find a Eulerian path. We summarize our findings and conclusions in Section 13.

2.DNA

Deoxyribonucleic Acid (DNA) is the blueprint of life. It is a double helix with two antiparallel strands and has all of the information to code for an organism. Each chain in the DNA is called a DNA strand. The DNA is made up of nucleotides. Each nucleotide has a phosphate group, pentose sugar, and a nitrogenous base (3). There are only four types of nitrogenous bases in DNA: adenine (A), cytosine(C), thymine (T), and guanine (G). These nucleotides can be further sorted into two groups: purines and pyrimidines. Purines are double-ringed while pyrimidines are single-ringed. A purine can only pair with a pyrimidine. The two purines are adenine and guanine, and the pyrimidines are cytosine and

Corresponding author:** ***Aakanksha Mahajan,
United States.

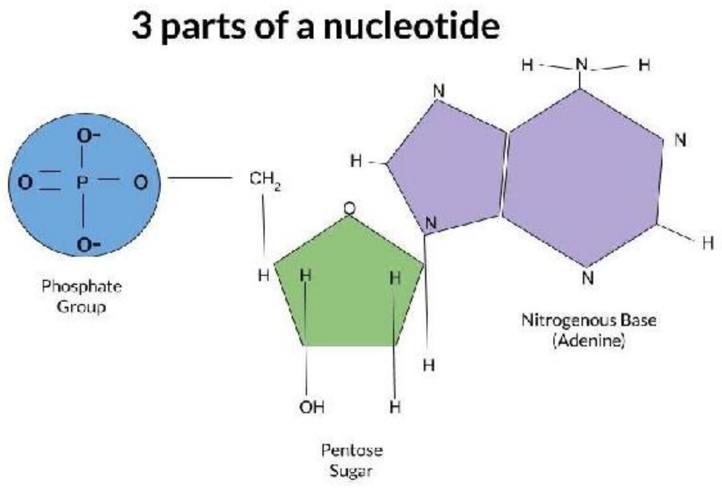

Figure 1. Diagram of a nucleotide

thymine. This makes adenine and thymine complementary base pairs and cytosine and thymine complementary base pairs. For this paper, we are not interested in the DNA structure, only sequences of the bases in each strand. Because of the complementary base pairing, we do not need to know each strand's sequence as the other strand's sequence can be derived from the original strand's sequence. The sequence of bases in DNA can provide us information about mutations, diseases, evolutionary relationships, and much more. Our ultimate goal is to be able to read full.

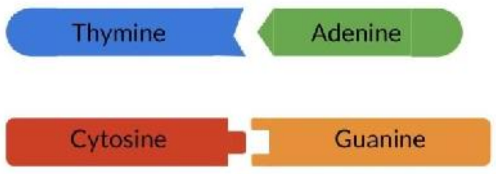

Figure 2. Diagram of complementary base pairing

genomes. More specifically, we want to find all of the nucleotides in an entire genome and the order in which they appear. We only need to find the sequence of one strand of the genome because of the reasons given above. This means, in this paper, a genome is only a long string of letters with each of the letters is A, G, C, or T (1).

3. READING GENOMES

Every cell in our body contains a copy of our DNA in its nucleus. This means even a small droplet of a human's bodily fluid can contain billions of copies of the genome. Human genomes are about 3 billion nucleotides long, and current experiments cannot read sequences of that length, but they can read sequences of shorter lengths. Because of this, genomes are broken into smaller sections using chemical reactions. These smaller sections are read and used to reconstruct the full genome. The simple example below illustrates this process.

The original genome:

AGGTCAGCTATCAGTACGTA

Genomes in the sample:

AGGTCAGCTATCAGTACGTA
AGGTCAGCTATCAGTACGTA
AGGTCAGCTATCAGTACGTA
AGGTCAGCTATCAGTACGTA

These genomes are broken down into smaller pieces, all of which are four letters, except the first and/ or last pieces of the genome

Genome sections:

```
  AG GTCA     GCTA     TCAG     TACG     TA
 AGGT     CAGC     TATC     AGTA     CGTA
A     GGTC     AGCT     ATCA     GTAC     GTA
 AGG     TCAG     CTAT     CAGT     ACGT     A
```

We only keep the sections that are four letters long.

GTCA	*GCTA*	*TCAG*	*TACG*	
AGGT	*CAGC*	*TATC*	*AGTA*	*CGTA*
GGTC	*AGCT*	*ATCA*	*GTAC*	
TCAG	*CTAT*	*CAGT*	*ACGT*	

These sections are now readable by experiments. These sections are called reads. These sequences will not be in the order they are in the genome. So, they are listed in the lexicographic order below.

ACGT AGCT AGGT AGTA ATCA CAGC CAGT CGTA CTAT GCTA GGTC GTAC GTCA TACG TATC TCAG

The goal of this paper is to reconstruct the original genome using the reads.

4. GENOME RECONSTRUCTION WITH INSPECTION

Inspection is one method to reconstruct a genome. While it is useful, it is time-consuming and repetitive and therefore works best with short genomes. The examples in this section are a continuation of the previous section.

A prefix is a sequence of letters excluding the last letter of the sequence. A suffix is a sequence of letters excluding the first letter of a sequence. For example, the prefix of *ACGT* is *ACG*, and the suffix is *CGT*. A kmer is a sub-sequence of text with *k* letters in each element. For example, *TTC* is a 3-mer.

To do reconstruction with inspection, you must have a list of k-mers. In this example, they are 4-mers. The list we will be using is this:

AAGC AGCC CGTA GCCA GTAA TAAG

You start by taking the first 4-mer and placing it with space above and below it. Like this:

AAGC

The next step is to find a 4-mer with a prefix that matches the suffix of the first 4-mer. *AGCC* meets these criteria. You will then place this 4-mer on top of the first 4-mer, leaving a space for the first character. This makes it so all of the matching letters overlap.

```
 AGCC
AAGC
```

You continue this process by finding a 4-mer that matches one of the following criteria: the suffix of it matches the prefix of the bottom 4-mer (*AAGC*), or the prefix matches the suffix of the top kmer (*AGCC*). The selected 4-mer must also not have been selected yet. A 4-mer that matches these criteria is *GCCA*.

```
  GCCA
 AGCC
AAGC
```

This process continues until all of the 4-mers have been placed.

```
     GCCA
    AGCC
   AAGC
  TAAG
 GTAA
CGTA
```

Now that all of the 4-mers have been placed, we can reconstruct the genome. The letters in the columns are all the same. To reconstruct the genome, bring down the letter in each column down.

```
 GCCA
AGCC
```

AAGC
TAAG
GTAA
CGTA

CGTAAGCCA

This means the final genome is CGTAAGCCA. A genome cannot always be reconstructed using inspection. Sometimes, a point is reached where you cannot continue stacking k-mers. This also works best for short genomes because as the number of k-mers increases, the more time consuming this task becomes. The goal of this paper is to complete this same task using a computational algorithm.

5. COMPUTATIONAL ALGORITHM FOR GENOME RECONSTRUCTION

Our final goal is to recover genomes from the information given. To do this, we must define a few terms. A genome is a finite sequence of letters. Each of the letters must be either *A*, *C*, *G*, or *T*. For example, *AGTCTGATCGATCGTA* is a genome. A k-mer is a sequence of *k* letters. Again, the letters must be *A*, *G*, *C*, or *T*. A k-merof a genome is a sequence of *k* letters of the genome. For example, *AGTC* is a 4-mer of *AGTCTGATCGATCGTA* because it appears in the genome. When referring to the k-mers of a genome, we refer to the list of the k-mers in a genome. Going back to the previous example, the 3-mers of *AGTCTGATCGATCGTA* are *AGT*, *GTC*, *TCT*, *CTG*, *TGA*, *GAT*, *ATC*, *TCG*, *CGA*, *GAT*, *ATC*, *TCG*, *CGT*, and *GTA*. The 3-mer *TCG* is repeated in the genome and, therefore, repeated in our list of 3-mers.

6. GENOME RECONSTRUCTION PROBLEM

The problem we are attempting to solve is given an input of a set of k-mers. Our goal is to provide an output of a genome whose k-mers are the same as the input list. The order of the k-mers is irrelevant. If we are successful in doing so, we have reconstructed the genome. For example, if the input is *TAC*, *ACG*, *GAT*, *ATT*, *TTG*, a possible output is the genome *TACGATTG*. In a list of k-mers, we can have no output, one output, or multiple outputs.

7. GRAPH THEORY

Solving the problem described above required the development of an algorithm that, if given a set of k-mers as an input, it will output a genome whose k-mers are equivalent to the list inputted. The algorithm we used will be based on elements of Graph Theory. Graph Theory is a branch of mathematics that studies graphs. We will be explaining the relevant concepts in the theory.

A directed graph is a set of nodes and a set of edges. An edge connects two nodes and has direction. The two nodes an edge points to are called the tail and the head. The edge begins at the tail and ends at the head (2).

Graphs are represented by drawings, where each node is a small circle, and each edge is an arrow. The arrow is drawn from the edge's tail to the head. An example is shown in Fig. 3. The table below illustrates each edge with its respective head and tail.

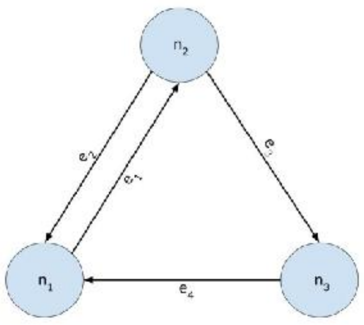

Figure 3. Example of a graph

Edge	Head	Tail
e_1	n_2	n_1
e_2	n_1	n_2
e_3	n_3	n_2
e_4	n_1	n_3

A path is a sequence of edges such that the head of e_i equals the tail of e_{i+1}. An example of a path is highlighted in red in Fig. 4.

A sequence is not a path if an edge is visited twice or if the head of one edge is not the next edge's tail in the sequence. An example of an edge being visited twice is shown in Fig. 5 and an example of a head of one edge not being the next edge's tail is shown in Fig. 6.

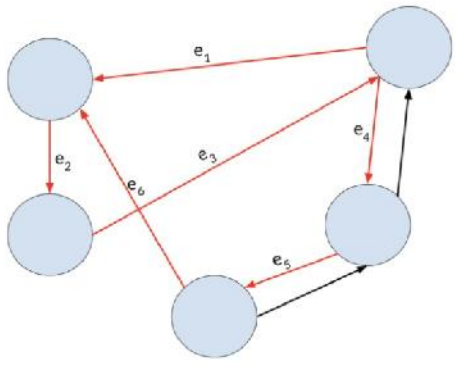

Figure 4: Example of a path

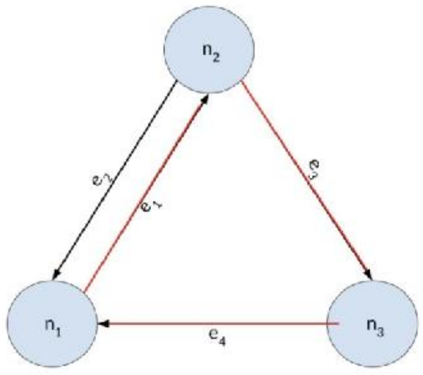

Figure 5: Example of an edge being visited twice

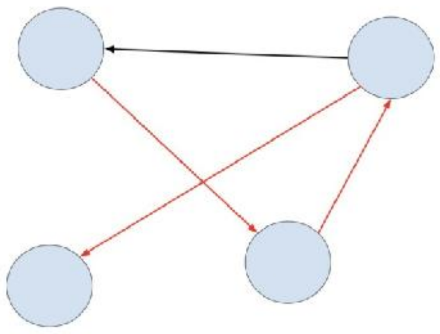

Figure 6: Example of one edge not being the next edge's tail

8. GRAPH OF A LIST OF K-MERS

We will be using a list of k-mers to construct a graph. To do so, we must define the following. An element may appear more than once in a list but not in a set. To distinguish between the two, from now on, we will be using square brackets ( ) to display lists and curly brackets { } to display sets. In a list of k-mers, a k-mer may appear more than once in a list. Given that K is a list of k-mers, the set $N(K)$ is the set of prefixes and suffixes of the k-mers in

K. For example, consider this list of 4-mers, $K = (AAGT,GTAG,AGTC,TCGA,GATA)$. From each 4-mer in the list, we can extract the prefix and the suffix. This is illustrated below.

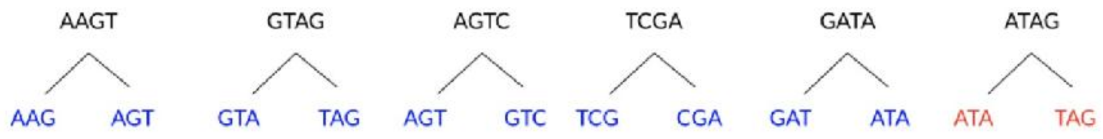

Figure 7: Illustration of the extraction of the prefix and suffix of a kmer

In blue, we have the prefix or suffix when it appears for the first time, and in red, we have the prefix or suffix that has already appeared. The set *N*(*K*) is all of the blue 3-mers. *N*(*K*) = {*AAG*,*AGT*,*GTA*,*TAG*,*AGT*,*GTC*,*TCG*,*CGA*,*GAT*,*ATA*}. The notation for the prefix and suffix for the k-mer w is prefix(w) and suffix(w), respectively. For example, prefix(*GTGA*) = *GTG* and suffix(*GTGA*) = *TGA*. The graph of N(K) is defined below:

- For each k-mer in *K*, an edge is defined that is labeled as that k-mer. The label is denoted as label (*e*).
- For each (*k*−1)-mer in *N*(*K*), a node is defined that is labeled as that (*k*−1)-mer. The label is denoted as label (*n*).
- The tail of edge *e* is the node *t* such that prefix(label(*e*)) = label(*t*)
- The head of edge *e* is the node *h* such that prefix(label(*e*)) = label(*h*)

G(*K*) is the graph of the list of k-mers*k*. For example, consider this list of 3-mers:
K = (*AAG*,*AGC*,*GCT*,*CTA*,*TAG*,*AGC*,*GCC*,*CCA*,*CAT*)
The set *N*(*K*) is:
N(*K*) = *AA*,*AG*,*GC*,*CT*,*TA*,*AT*,*CC*,*CA*

The table below contains all of the information on the graph:

Label of Edge	Label of Head	Label of Tail
AAG	*AA*	*AG*
AGC	*AG*	*GC*
GCT	*GC*	*CT*
CTA	*CT*	*TA*
TAG	*TA*	*AG*
AGC	*AG*	*GC*
GCC	*GC*	*CC*
CCA	*CC*	*CA*
CAT	*CA*	*AT*

The graph that is created from this information is Fig. 8. This graph labeled the edges, but it is not necessary as the label can be found from the tail and head labels.

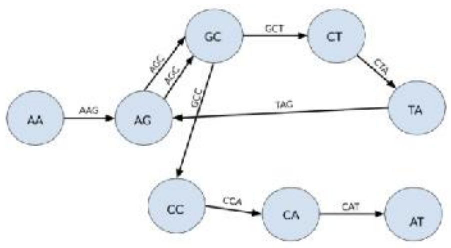

Figure 8: Graph of *N*(*K*)

9. GENOME OF A PATH IN A GRAPH OF A LIST OF K-MERS

Below are some definitions essential to the understanding of this section: Given a k-mer *k*, we denote the last letter of *k* by last (*k*). For example, last(*AGTCTC*) = *C*. Given two sequences of letters, *u* and *v*, we show their concatenation by *u* + *v*. For example, *TGCTA* + *TAG* = *TGCTATAG*. Let *K* be a list of k-mers. Let *G*(*K*) be the graph of *K*. Let e_1, e_2, e_3....$e_n$be a path in *G*(*K*). We define the genome of this path as label(e_1) + last(e_2) + ... + last(e_n).

As an example, the same graph as above is shown in Fig. 9, but we label the edges that form the path e_1,e_2,e_3,e_4, in red. In this example, the path is e_1,e_2,e_3,e_4. Label(e_1) is *AGC*, last(e_2) = *C*, last(e_3) = *A*, last(e_4) = *T*. The genome of this graph is *AGCCAT*. In this example, the

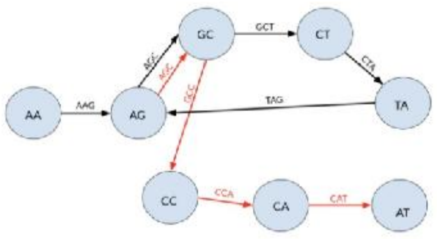

Figure 9. Graph of $N(K)$ with path e_1,e_2,e_3,e_4 .

path is e_1,e_2,e_3,e_4. Label(e_1) is *AGC*, last(e_2) = *C*, last(e_3) = *A*, last(e_4) = *T*. The genome of this graph is *AGCCAT*.
Let $G(K)$ be the graph of K. Let $e_1,e_2,e_3....e_n$ be a path in $G(K)$. The k-mers of this path's genome is the list of the labels of the edges of the path. This is an important observation. In the case of the graph above, the genome of the path is *AGCCAT*. The list of 3-mers of this genome is (*AGC*,*GCC*,*CCA*,*CAT*). The list of the edges' labels in this path is (*AGC*,*GCC*,*CCA*,*CAT*), which is the same as the list of 3-mers. Let K be a list of k-mers. Let $G(K)$ be the graph of K. Let $e_1,e_2,e_3....e_n$ be a path in $G(K)$. Assume this path contains all the edges of $G(K)$. Then the list of k-mers of this path's genome is K. This genome is the solution to our genome reconstruction problem. This means the answer to our genome reconstruction problem can be found by finding a path in $G(K)$ that contains all of the edges.

For example, consider the graph and path in the figure below. For graph $G(K)$,K= (*AAG*,*AGC*, *GCT*,*CTA*,*TAG*,*GCC*,*CCA*,*CAT*). In this example, the path $e_1,e_2,e_3....e_8$ contains all of the graph's edges. Its genome is *AAGCTAGCAT*. The list of 3-mers of this genome is (*AAG*,*AGC*,*GCT*, *CTA*,*TAG*,*AGC*,*GCA*,*CAT*), which is the same list that we used to create the graph. The graph is illustrated in Fig. 10

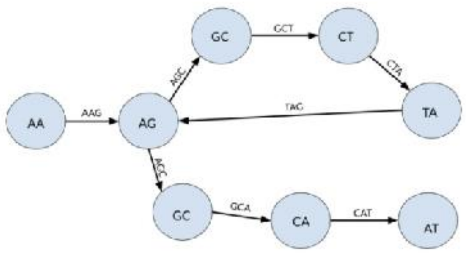

Figure 10: Graph of *K*.

More than one path contains all the edges, and these different paths can lead to different genomes. For example, as illustrated in Fig. 11, this graph has multiple genomes. One path results

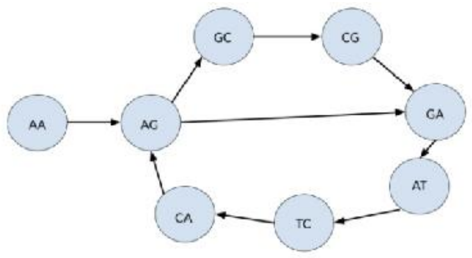

Figure 11: Graph with multiple paths

in a genome of *AGCGATCAGA*, and another path results in a genome of *AAGATCAGCGA*.

10. Eulerian paths and conditions for their existence

In the last section, we reduced the genome reconstruction problem from the list K to finding a path in graph $G(K)$ that contains all its edges. We will continue to study that problem in this section. An Eulerian path is a path that contains all the edges in the graph. So, if K is a list of k-mers, and our goal is to reconstruct the genome with K, our goal is to find an Eulerian path of $G(K)$. We will not be worried about the fact that we can find more than one genome. Our goal is to find just one genome.

Let n be a node in a directed graph. The indegree of n, indeg(n), is the number of edges that have n as a head. The outdegree of n, outdeg(n), is the number of edges that have n as a tail. An example of a graph with the nodes listed with their indegrees and outdegrees listed is shown in Fig. 12.

Node	Indegree	Outdegree
n_1	1	1

n_2	1	2
n_3	1	0
n_4	1	1

The sum of all of the indegrees equals the sum of all of the outdegrees. This is always true because every edge contributes to 1 indegree of its head and also contributes 1 to the outdegree of its tail. A connected graph is one that for every pair of different nodes n_1 and n_2 there is a path that starts with an edge that has n_1 as tail and ends with an edge with n_2 as its head. An example of a

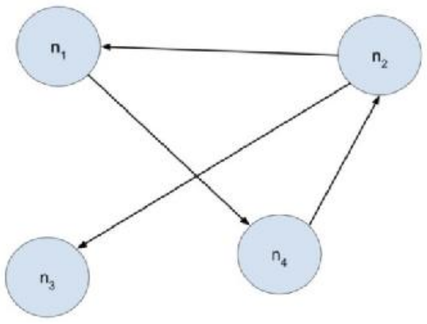

Figure 12: Graph of K.

connected graph is shown in Fig. 13 and examples of graphs that are not connected are shown in Fig. 14.

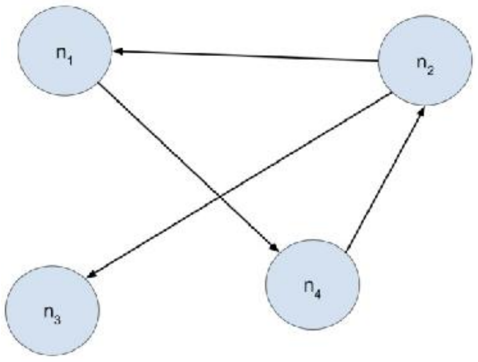

Figure 13: Connected Graph

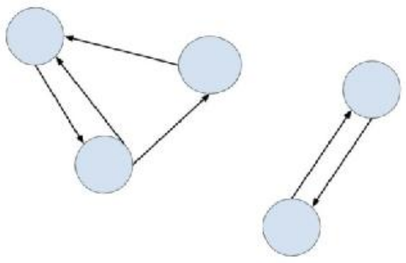

Figure 14: Not connected graphs

A cycle is a path if the tail of e_1 is equal to the head of e_n. An example of a graph with a cycle is shown in Fig. 15 and examples of a graph without a cycle is shown in Fig. 16. Let *G* be a directed graph with nodes *V* and edges *E*. Assume *G* is connected. *G* as an Eulerian cycle if and only if indeg(n) = outdeg(n) for every node *n*. Let *G* be a directed graph with nodes *V* and edges *E*. Assume *G* is connected. *G* has an Eulerian path that is not a cycle if and if there exists a node a such that outdeg(a) - indeg(a) =

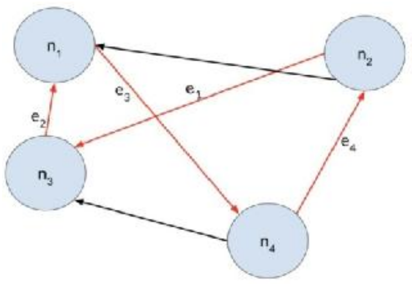

Figure 15: Graph with a cycle

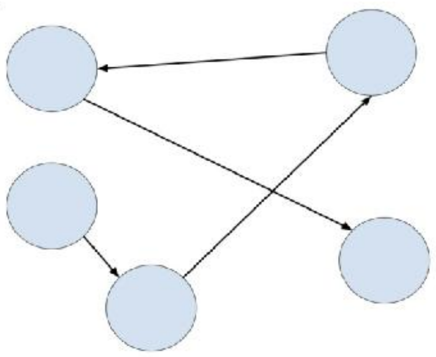
Figure 16: Graph without a cycle

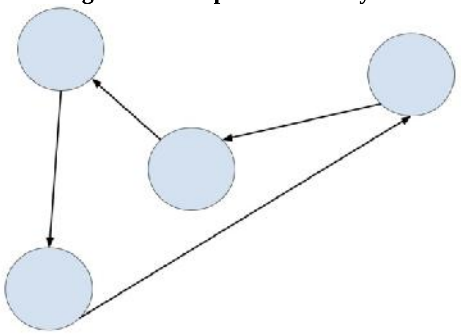
Figure 17: Graph with a Eulerian cycle

1, a node b such that indeg(a) - outdeg(a) = 1 and for every other node n, indeg(n) = outdeg(n). Also, any Eulerian path starts with an edge having a as its tail and ends with an edge having b as its head.

11. ALGORITHM TO FIND AN EULERIAN CYCLE

In this part, we will be describing an algorithm to find an Eulerian cycle in a graph. We assume that such a cycle exists. So, we assume that for node n, indeg (n) = outdeg (n). We describe the algorithm on a simple example, using the graph in Fig. 20.

Pick an edge. This can be any edge. Call it e_1. This is shown in Fig. 21
.

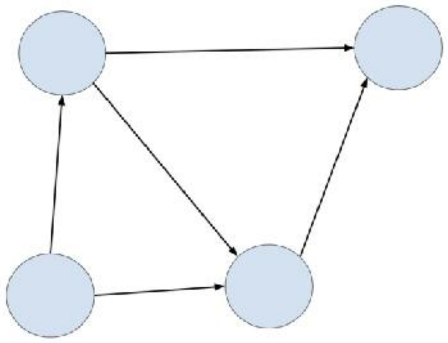
Figure 18: Graph without a Eulerian cycle.

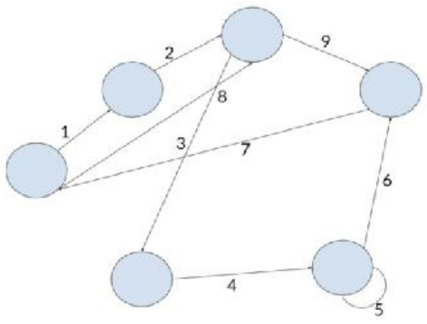

Figure 19: Graph G.

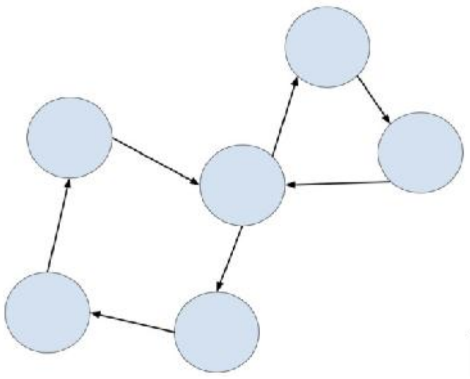
Figure 20: Graph of *K*.

Create a path until it is no longer possible to continue. This is shown in Fig. 22. If we had visited all the edges by the time we had to stop because we could not continue, we would be done.

However, most of the time, as shown in the above example, this is not the case. The black edges are not in this cycle. As the graph is connected, at least one node in the cycle is the tail of an edge that is not in the cycle. The next step is to go around the cycle and stop at one of these nodes. This is shown in Fig. 23. The red node is the first node in the cycle e_1, e_2, e_3, e_4 that has a black edge coming out. This black edge was labeled e_5.

Retrace the cycle, but starting at the red node, and change the label of the edges accordingly.

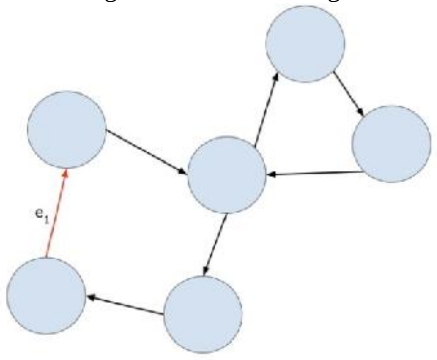

Figure 21:

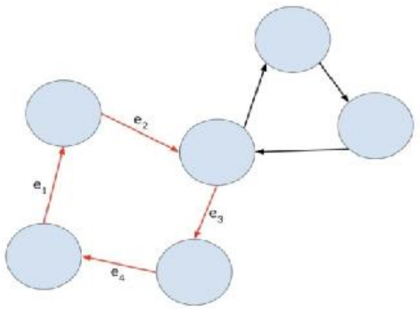

Figure 22:

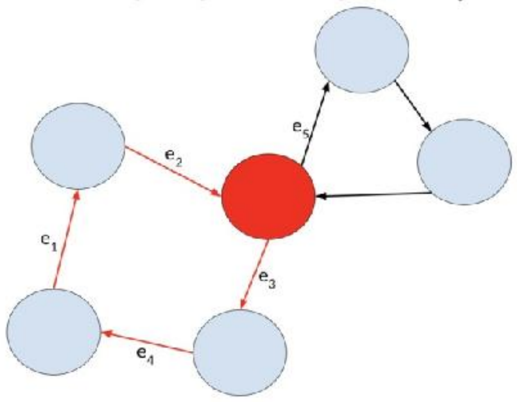

Figure 23:

As shown in Fig. 24. Because we always end at the same we started with when tracing a cycle, we ended at the red node. However, the red node was selected because it has an edge coming out that is not in cycle. This was edge e_5. This means that we do not need to stop. We can continue and obtain a cycle that contains all the edges of the red cycle and some new edges. This is shown below. In the case below, we have an Eulerian cycle, and we are done. It is important to note that this is not always the case, but we can continue repeating this strategy, and we will eventually end up with an Eulerian cycle.

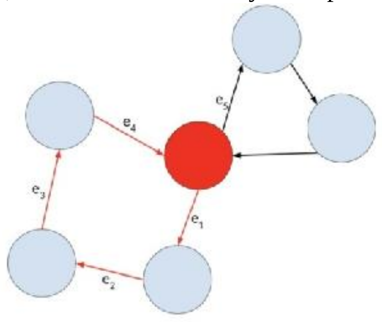

Figure 24. An Eulerian cycle

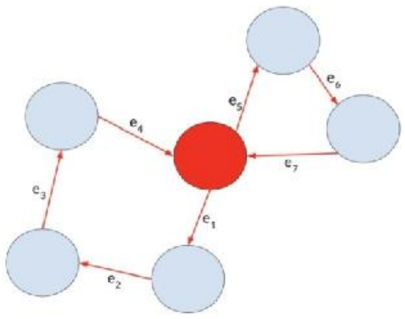

Figure 25

12. ALGORITHM TO FIND AN EULERIAN PATH

Our original goal was to find an Eulerian path, not an Eulerian cycle. We first verify that such a path exists. To do that, we must compute indeg(n) - outdeg(n) for every node n, and verify that this quantity is 1 for only one node, that we call b, it is −1 for only one node, that we call 1, and is 0 for every other node. We then add an edge with b as tail and a as head. We now have that indeg(n)-outdeg(n) for every node, and thus, there is an Eulerian cycle. Next, we find the Eulerian cycle in this new graph (with the edge from b to a added).

We retrace the cycle, starting with the edge that comes after the added edge and stops just before the turn of this new edge. This path is an Eulerian path.

These steps are illustrated below.

Add the red edge from b to a. This is illustrated in Fig. 27.

Find the Eulerian cycle. This is shown in Fig. 28.

Go around the cycle, but relabel the edges. Start at e_1 is the first edge after the red edge. The red edge would then be the last edge of the cycle, but do not include it to end up with the original graph’s Eulerian path. This is shown in Fig. 29.

An Eulerian path in the example we have been considering is shown in Fig. 30.

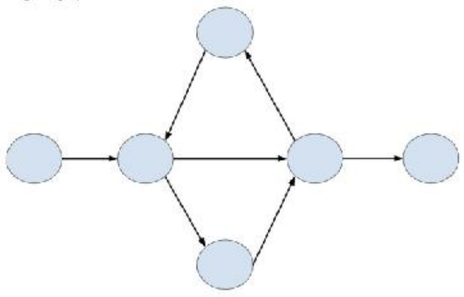

Figure 26. Original graph.

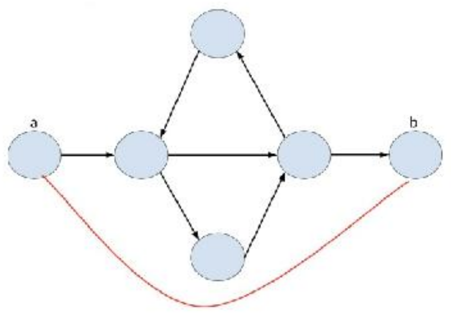

Figure 27:

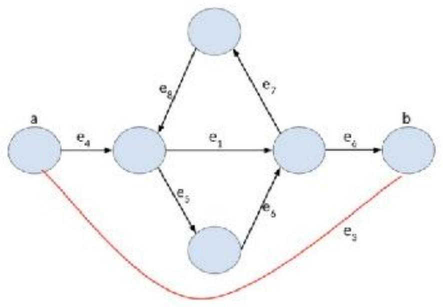

Figure 28

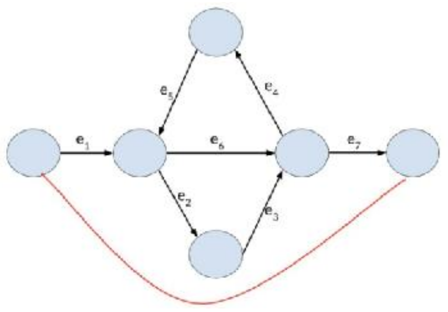

Figure 29:

This leads to the genome *TAATGGGATGCCATGTT*.

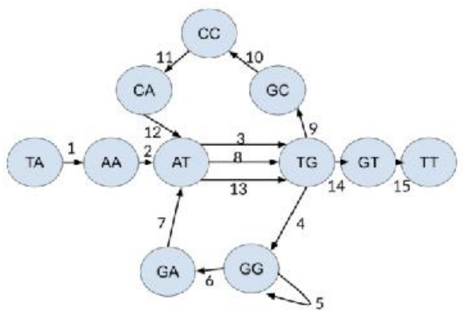

Figure 30

DISCUSSION

In this short article, we discuss the method in which we solve the genome reconstruction problem. We discuss why this problem exists, and how we can employ graph theory to solve it. We first go about the construction of a graph from a list of k-mers. We then use this graph to first make an algorithm to find an Eulerian path, and then make an algorithm to find an Eulerian cycle. This final Eulerian cycle will lead to a genome that as the same k-mers as the original list.

REFERENCES

(1) Deoxyribonucleic acid (dna).
(2) Douglas B. 2008. West. *Introduction to graph theory*. Prentice Hall.

(3) Michael D. Topal and Jacques R. Fresco. Complementary base pairing and the origin of substitution mutations. *Nature*, 263(5575):285–289, Sep 1976.
(4) Phillip E. C. 2010. Compeau and Pavel A. Pevzner. Genome reconstruction: a puzzle with a billion pieces. *Bioinformatics for Biologists*, page 1–30,
(5) Wil A. M. Loenen, David T. F. Dryden, Elisabeth A. Raleigh, Geoffrey G. Wilson, and Noreen E. Murray. 2013. Highlights of the dna cutters: a short history of the restriction enzymes. *Nucleic Acids Research*, 42(1):3–19.

****