sciendo

DOI: 10.2478/awutm-2024-0001

Annals of West University of Timisoara
Mathematics and Computer Science
60 (2024), 1–24

# A Study of Computational Genome Assembly by Graph Theory

Bijan Sarkar

**Abstract** The assembly of billions of short sequencing reads into a contiguous genome is a daunting task. The foundation knowledge of current DNA assembly models is concentrated among a select group, where the solution to the genome assembly challenge lies in proper ordering the genomic data. This contribution's objective is to provide an overview of the original graph models used in DNA sequencing by hybridization. With the updated analytical approach based on the bidirectional bipartite graph class, the theoretical basic structure of the DNA assembly model has been described in new perspective by incorporating few short hypothetical DNA sequences. On the Galaxy platform, by using Spades assembler and Velvet assembler, the comparative outcomes of an experiment are presented, and we also identify their working schemes. Here, the working principle of de Bruijn graph has been discussed in broader point of view.

**AMS Subject Classification (2020)** 92C40; 92D20; 05Cxx; 05C15; 68N30; 68Q07
**Keywords** Genome assembly; De novo assembly; De Bruijn graphs; Bidirectional bipartite graphs; Spades assembler; Velvet assembler

## 1 Introduction

Assembling a genome is similar to doing a jigsaw puzzle. The overall image guides us in assembling pieces to form a picture in a puzzle. As everyone knows, this particular jigsaw puzzle is extremely difficult and time-consuming to solve since, in the absence of the whole picture, we must estimate the prior information that explains the connections between each and every individual components. Similarly, in genome assembly, we have DNA pieces (reads) and must fit them together without knowing the final sequence. It's like a challenging puzzle where we lack the complete picture, requiring us to infer

connections between genetic components. Although there are similarities between jigsaw solution and genome assembly, genome assembly is more challenging due to the large number of individual pieces that form its collection [5, 23].

Among the various theoretical computation approaches to propel modern biology, we frequently use mathematical based algorithms to tackle intricate questions and overcome biological hurdles. In a scenario like this, utilizing the graph theory field to solve a specific problem brought a fresh look and stimulated new findings. One such problem may be genome assembly or DNA sequencing, which is a fundamental problem in molecular biology. Genome assembly advances show the ongoing advancement of science across time, integrating biology and technology [3]. In this paper, we attempt to give an outline of genome assembly to which working mechanism of assembly has been explained through graph theory in order to understand the technical arrangement based on the pure theoretical perspective.

The four nitrogenous bases adenine (`A`), guanine (`G`), cytosine (`C`), and thymine (`T`) combine to generate each monomer that makes up the DNA polymer. The term DNA stands for deoxyribonucleic acid and it is composed of two twisted strands of four nucleotides forming a double helix, which encrypts the genetic data of living things. The four different nucleotide types in DNA are joined together by hydrogen bonds in such a way that adenine and thymine are always paired, whereas guanine and cytosine are always paired. For this reason, from given a fragment of DNA of one strand we can always figure out how the matching segment in the opposing strand is organized. Complementarity is the name given to such a characteristic. The pentose carbon atoms $5'$ and $3'$ dictate the starting and terminating positions of a DNA strand, respectively. A DNA strand is read $5'$ to $3'$ direction and running in the opposite direction is the complementary strand. The number of nucleotides of single stranded chains is usually expressed in base pairs (bp) [17].

A specific method for figuring out a DNA's nucleotide sequence is called DNA sequencing. In 1977, Frederik Sanger with his colleagues [19] proposed a DNA sequencing method that was able to read DNA fragments (i.e., reads) upto $700 - 900$ bp in length. The method which was a pure wet laboratory approach was slow and costly. Sanger et al. [13, 20] suggested breaking the DNA sequences at random points to shorten processing time. Consequently, the order of the fragments is unknown that generates an NP-complete (non-deterministic polynomial time) computational problem. Shotgun technique is the term used to describe this approach. Later, in 1988 Edwin Southern introduce sequencing by hybridization (SBH) [22] approach in which very short DNA fragments are reads and in order to assemble the original genome (typically a few hundred of nucleotides) a computational algorithmic method is used. In application of algorithmic approach to SBH, we must cite Iu. P. Lysov with his colleagues [10] and P. A. Pevzner [15] who formulate the genome assembly problem as finding a Hamiltonian path and an Eulerian path, respectively. Second-generation sequencing or next-generation sequencing (NGS) developed a few decades after the Sanger sequencing method. Massively parallel sequencing is a feature of the NGS technology, which provides extremely high throughput, scalability, and speed. At relatively low cost NGS can produce a large number of short DNA reads typically between 35 and 500 bp. The third-generation sequencing (TGS) started in the 2010s to produce reads longer than NGS without amplification where the graph theory

layer of the algorithm solution is more or less the same.

One of the biggest data issues in bioinformatics is the de novo genome assembly, which involves reconstructing a fresh genome from a collection of short reads where there is no reference sequence available for alignment. A graph representation of the links between the reads sharing similar prefixes and suffixes is a crucial tactic employed in de novo assembly. Initially, greedy algorithms which aim for local optima were used to find the order of the fragments; later the graph method algorithms which aim for global optima were introduced with different values for $k$-mer (section of $k$ consecutive bases). In the graph method algorithms, the overlap-layout-consensus algorithm and the de Bruijn graph algorithm are two main computational approaches for representing overlaps between reads in de novo assembly. The overlap-layout-consensus algorithm is an approach where each read is graphed as a node and edges are created between two nodes if $k-1$ size prefix of a left read node is equal to $k-1$ size suffix of a right read node. Here, the algorithm determines the Hamiltonian path through the graph. On the other hand, the de Bruijn graph algorithm is based on $k$-mers methodology which breaks the short reads into $k$-mers those represent as edges where nodes correspond to either a $k-1$ size prefix or a $k-1$ size suffix of one of the $k$-mers, and then builds a de Bruijn graph. Here, the algorithm determines Eulerian path through the graph. The first stable overlap-layout-consensus assembler is introduced in 2000 [12] whereas the first stable de Bruijn graph assembler is proposed in 2001 [16]. The de Bruijn graph assembler typically performs better on large read sets than other assembler. In this connection, it is important to mention that Idury and Waterman first developed the Eulerian approach to genome assembly in [7] and the approach based on de Bruijn graph for paired end reads is introduced by Medvedev et al. [11]. In our study, we include two de Bruijn graph assemblers, Spades and Velvet, to measure their performance to genome assembly.

It is no doubt that the assembly of billions of short sequencing reads into a contiguous genome is a formidable challenge. In spite of all difficulties de Bruijn graph structures are the foundation of most well organized modern assembly software tools, they can handle huge datasets with millions of reads. The Velvet algorithm [24] plays a major role in this field. It is not just a powerful algorithm, but also based on this algorithm many modern assemblers have been developed, for examples: IDBA, Spades, Soapdenovo2, Megahit etc. [1, 8, 9, 14]. In comparative studies, the de Bruijn graph-based assemblers show efficient performance during handling extremely large datasets with relatively longer read lengths (75bps), where the system requires relatively short runtime as well as short memory space [2–4, 21, 25].

The remaining portions of the paper are arranged as follows. Section 2.1 describes the simplest forms of a group of terminologies which are basic requirements to understand as well as to run the Spades algorithm. The initial arrangement of computational genome assembly on the Galaxy platform has been provided as well. Section 2.2 focuses on the graphical concept of computational genome assembly. It gives a theoretical description of paired-end reads and concludes with a detailed discussion. Section 2.3 reveals the basic understanding of the Velvet algorithm and describes the initial preparation of computational genome assembly on the Galaxy platform. Different kinds of errors occur during genome assembly, as explained in Section 2.4. Section 3 presents a computational comparison of the considered experiment. Finally, in section 4, we draw our conclusion.

# 2 The subject of genome assembly

## 2.1 Spades algorithm: Basic concept and preparation for computational genome assembly

To understand the preparation for computational genome assembly at the initial situation, we have to keep in mind the simplest forms of the following terminologies.

Shotgun sequencing: A method in which the genome is randomly divided into small DNA fragments, each of which is then separated sequenced, as part of a laboratory operation. A computer program searches for overlaps in the DNA sequences to reassemble the fragments in the proper order to reconstruct the genome.

Illumina DNA sequencing instrument: Following the specific steps – steps of shotgun sequencing – Illumina DNA sequencing instrument generates many millions of highly accurate reads, the whole process is much faster and cheaper than other available sequencing methods.

Coverage: The term coverage in NGS always describes a relation between sequence reads and a whole genome (reference) unlike sequence depth which describes a total read number. A technique for calculating genome coverage is the Lander/Waterman equation. The equation is $C = \frac{LN}{G}$, where $C$ stands for coverage, $G$ is the haploid genome length, $L$ is the read length, $N$ is the number of reads.

FASTS format: A FASTQ file format normally uses four lines per sequence (reads). The @ character at the start of the first line is followed by the sequence ID (identifier). The index sequences indicated by read1 and read2 are contained in it. The second line is the raw/base sequence letters. The third line begins with a + character where the sequence ID is often omitted. The fourth line encodes the quality values for the base sequence in the second line which are determined by the ASCII values of characters.

Scaffold: A section of the reference genome sequence known as a scaffold which is built up using end-sequenced whole-genome shotgun clones. Contigs and gaps make up scaffolds where a contig is a continuous length of known genomic sequence. Here, we have to remember that one genomic sequence (chromosome) may be presented by one scaffold or many scaffolds. Known sequence (read) and unknown sequence compose fragment and during the alignment a gap occurs in between two contigs at the position of a fragment of the not known sequence.

We now reach to an informational infrastructure where we can begin the computational preparation. For computational purpose, we choose Galaxy [26] – a web-based scientific analysis platform. Our intention is that the overall computational genome assembly step-result, performed on the Galaxy platform, is to be explained and examined by the concept of the graph theory – basically by the de Bruijn graph.

To reach the objectives, we first consider a very fundamental genome assembly computation using Spades; the latter we will use Velvet assembler. Using short DNA reads from high-throughput sequencing, de novo assembly creates a whole genome sequence without a reference. De novo assemblers include members like Spades that employ input data from small read sets in which the de Bruijn graphs serve as the sole foundation for the assembly process. For demonstration the genome assembly, we use a set of reads from an imaginary staphylococcus aureus bacterium with a miniature genome (197394 bp). Using an Illumina DNA sequencing instrument, the whole genome shotgun tech-

nique was used to sequence our mutant strain read set. Here, we want to rebuild our reference genome (imaginary staphylococcus aureus bacterium) via a de novo assembly of a short read set using the Spades assembler [6, 28]. We collect the files, mutant_R1.fastq and mutant_ R2.fastq, from Zenodo [27] for assembly. The reads are paired-end and each read is 150 bases long. The genome coverage value is equal to 19, i.e., it describes that in average 19× the genome sequence of the wildtype strain (reference sequence) is covered by bases from the reads. We are aware that the term *coverage* refers to the ratio of the total number of bases in the fragments to the total number of bases in the entire genome.

Now again, we have to keep in mind the simplest forms of the few more following terminologies.

MDA (single-cell) data: Spades, a genome assembly algorithm, is intended for bacterial data sets with one or more cells. Compared to sequencing DNA isolated from several cells, single cell sequencing (SCS) has an advantage. To maximize the accuracy and quality of SCS, a consistent DNA amplification is needed where multiple displacement amplification (MDA) is a method of DNA amplification. With this technique, it is possible to quickly amplify incredibly small DNA sample amounts to a sufficient number for genome analysis.

Read error correction: We start with a lengthy DNA sequence but are unable to directly read it. The reason of that, we do sequencing in order to obtain numerous random small pieces of DNA (reads). For widely used sequencing techniques, reads are usually $L \approx$ 35-400 bp long. Reconstructing the original sequence from the set of reads is the assembly problem. Generally, we read length of $L$ nucleotides from both ends of a long DNA fragment, and this leads to paired-end reads (read pairs) where the distance between them is known. Projects involving single-cell sequencing use MDA technology, which creates non-uniform coverage. This means that a coverage threshold would throw a lot of the genome away. Also not nearly identical is the insert size distribution. There are just more errors as a result, along with many chimeric connections. To correct these errors, we split reads into $k$-mers and examine the group of $k$-mers. It is simple to correct reads if we know what $k$-mers are correct and how to correct others. Regular (multi-cell) error correction allows us to determine the number of copies of a $k$-mer and presume that rare $k$-mers are errors. However, in single-cell datasets this idea does not work due to non-uniform coverage. Actually, we examine the collection of identical $k$-mers and choose the nucleotides of the central $k$-mer which will be higher covered than others, but there may be several centers in such a cluster. This idea, called BayesHammer correction is used to correct single-cell datasets [1].

$k$-mers: Substrings taken into consideration in the read are $k$-mers.

Coverage cutoff: A subpart of the coverage that has a user-assigned value is called the coverage cutoff.

Paired-end/Single reads: Self explanatory terminologies where paired-end reads have a specific distance value between them.

We assemble the forward reads: mutant_R1.fastq and the reverse reads: mutant_ R2.fastq on the Galaxy platform by running Spades. Spades chooses at least 3 $k$-mer sizes, one low, one medium and one high. We use auto option of *33*, *55* and *91*. All other parameters either are in custom setting or select an auto option. As an examination output, we get two fastq files of the resulting contigs and scaffolds, two files for statistics

about these, and Spades logfile. In the output, the short reads are assembled into much longer contigs. The *stats* file provides information about the length of each of the contigs. Now, let us look at the mechanical framework for assembling genomes from a graphical perspective before delving into further discussions about the computational results. We mainly confine ourselves to the de Bruijn graph presentation of the $k$-mer set of reads using the overlap-layout-consensus approach. In order to integrate science, technology, and mathematics, we use different terminologies to consider a DNA sequence at different positions.

### 2.2 Graphical approach of computational genome assembly

Multiple copies of a genome are broken in different places during DNA sequencing in order to produce short reads. After that, utilizing overlapping, the reads are sequenced and assembled into the genome's nucleotide sequence. As the short reads, $k$-mers, often have the same length, we can assume that reads are all $k$-mers for specific value of $k$. Additionally, if the same strand is used for all reads, have no errors, and show flawless coverage, then every $k$-mer substring of the genome can perfectly act as read. In considering a string text `TATCCCCTCG` as an example, we get the following list of 3-mers in lexicographic order:

$$\{\texttt{ATC}, \texttt{CCC}, \texttt{CCC}, \texttt{CCT}, \texttt{CTC}, \texttt{TAT}, \texttt{TCC}, \texttt{TCG}\}$$

We arrange reads in lexicographic order because, at the time of generating reads, the correct order of them is unknown.

In the computational problem of modeling genome assembly, the reference string is reconstructed from $k$-mers. To solve the string reconstruction problem, we connect a pair of $k$-mers if they overlap in $k-1$ symbols. In the graphical presentation of genome assembly in Fig.1, in each row two symbols are overlapped with the previous row and

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| **T** | **A** | **T** | | | | | | | |
| | A | T | **C** | | | | | | |
| | | T | C | **C** | | | | | |
| | | | C | C | **C** | | | | |
| | | | | C | C | **C** | | | |
| | | | | | C | C | **T** | | |
| | | | | | | C | T | **C** | |
| | | | | | | | T | C | **G** |
| T | A | T | C | C | C | C | T | C | G |

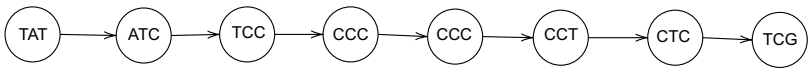



Figure 1: Top: Solution to the string reconstruction problem. Bottom: Genome path solution to the string reconstruction problem.

all bold symbols are attached one by one to construct the reference genome. In this method, arranging the order of $k$-mers is not easy. Without following a specific trick, it is very difficult to solve the string reconstruction problem because repeated $k$-mers complicate genome assembly. With millions of reads, repeated substrings in the genome are a severe issue where repeats make it challenging to plan ahead and put the right assembly together. To overcome this difficulty, we find out the genome path in the graph presentation of $k$-mers (see the bottom presentation in Fig.1).

If the suffix of a pattern matches the prefix of another pattern, we can use an arrow to connect any $k$-mer pattern to a $k$-mer pattern in the string's genomic path. From this graphical presentation our task to find out the Hamiltonian path which is the path representing the correct assembly. It is not possible to know the reference genome in advance, and when we connect all consecutive 3-mers according to the rule, its representation becomes complicated. It becomes more complicated in the arrangement of 3-mers lexicographically which results in the overlap graph displayed in Fig.2. In this presentation, the correct assembly is harder to see. In order to resolve the string reconstruction issue, we must locate a path in the overlap graph that visits each node precisely once; it is the character of Hamiltonian path. However, as the Hamiltonian path optimization problem is NP-hard, we often face a problem to find out a Hamiltonian path in a huge graph or even it is absolutely unclear whether such path exists! Consequently, instead of searching for Hamiltonian paths in huge graph we are generally interested in a de Bruijn development – a graph representation of a $k$-mer composition in an entirely different approach.

In the de Bruijn graph presentation, instead of assigning $k$-mers to the nodes we assign them to edges as shown in Fig.3 for 3-mers. It is obvious that by taking this route from left to right and adding a new nucleotide at each step, we can recreate the genome. Nodes are $(k-1)$-mers, where an arrow connects any $(k-1)$-mer prefix pattern to a

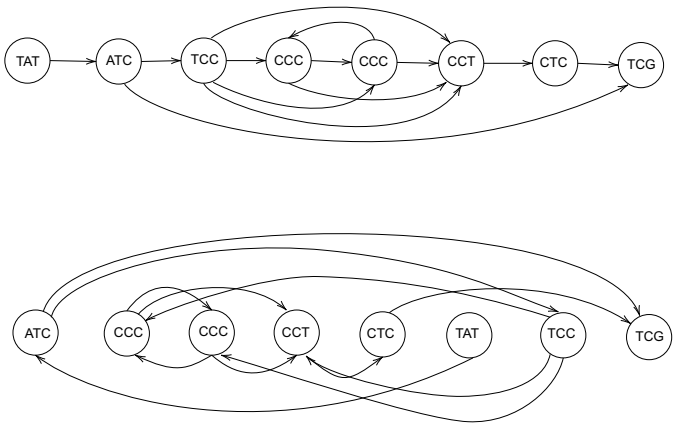



Figure 2: Top: Actual graphical presentation of 3-mers. The genome be identified by walking along the horizontal edges from `TAT` to `TCG`, which is nothing but the Hamiltonian path of this graphical presentation. Bottom: Graphical presentation of 3-mers ordered lexicographically. Our task is to find out the Hamiltonian path from this graphical presentation.

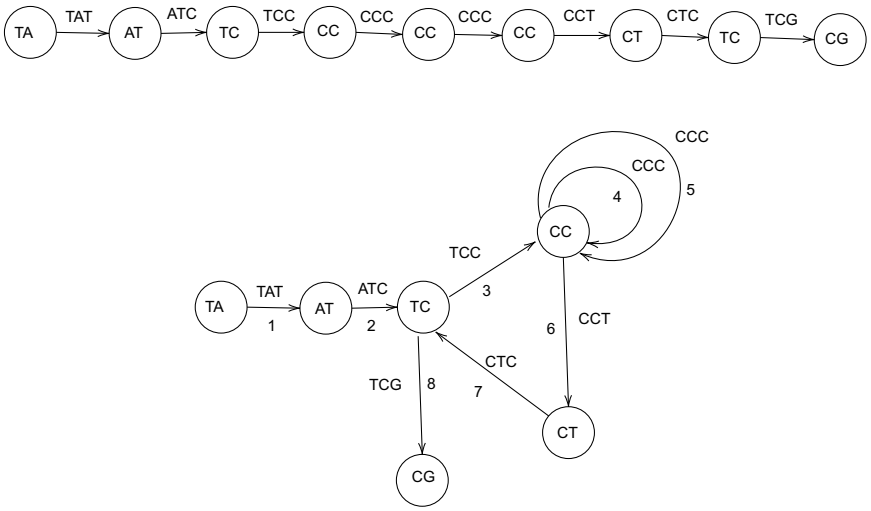



Figure 3: Top: Genome path representation. Edges are 3-mers, connecting nodes of 2-mers with exact suffix-prefix overlap. Bottom: The de Bruijn graph of string text `TATCCCCTCG`.

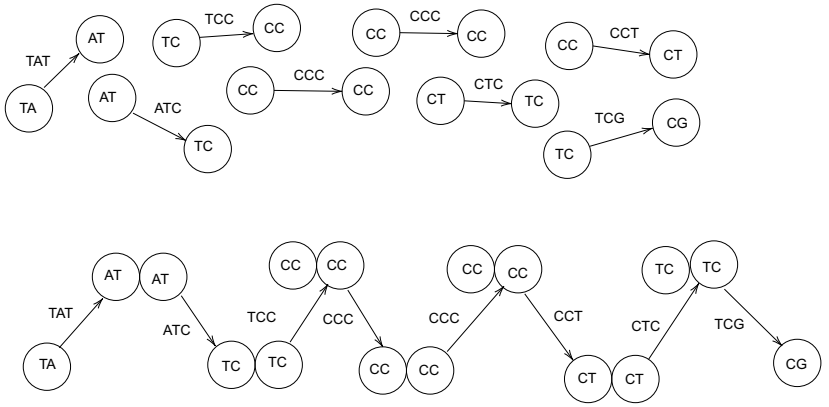



Figure 4: Top: Isolated 3-mers presentation. Bottom: Ordering the sequence of 3-mers by gluing the identically labelled nodes of 2-mers.

$(k-1)$-mer suffix pattern. The common node is labelled by the overlapping nucleotides of the suffix pattern of left edge and prefix pattern of the right edge.

Next, for size reduction of graphical presentation we glue the three `CC` nodes and the two `TC` nodes into a single node and doing so we get two loops connecting `CC` to itself. The obtained graph is called the de Bruijn graph of the considering string text. After gluing, the straight path has now become tangled, and to solve the string reconstruction problem our task is to find out a path in de Bruijn graph that visits every edge exactly once and we know that which is characteristic of Eulerian path, indicated in the bottom representation in Fig.3.

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| T | A | T | | C | C | C | | | | | | | | | | |
| | A | T | C | | C | C | T | | | | | | | | | |
| | | T | C | C | | C | T | C | | | | | | | | |
| | | | C | C | C | | T | C | G | | | | | | | |
| | | | | C | C | C | | C | G | A | | | | | | |
| | | | | | C | C | T | | G | A | T | | | | | |
| | | | | | | C | T | C | | A | T | T | | | | |
| | | | | | | | T | C | G | | T | T | C | | | |
| | | | | | | | | C | G | A | | T | C | C | | |
| | | | | | | | | | G | A | T | | C | C | C | |
| | | | | | | | | | | A | T | T | | C | C | T |
| T | A | T | C | C | C | C | T | C | G | A | T | T | C | C | C | T |

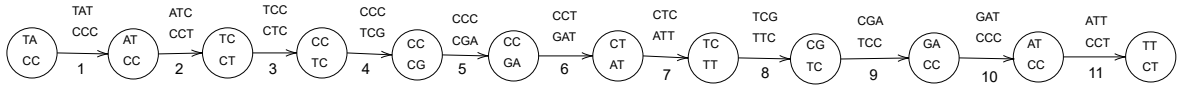
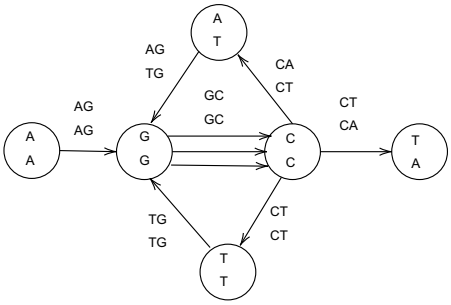



Figure 5: Top: Solution to the string reconstruction problem for (3,1)-mers presentation. Bottom: Genome path solution to the string reconstruction problem for (3,1)-mers presentation.

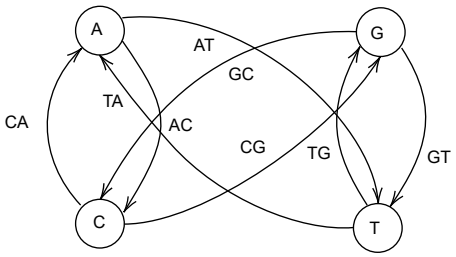
Here, a question likely arises: Can we determine the genome order of $k$-mers in graphical presentation? We do not know the order of $k$-mers in advance. However, gluing all identically labelled nodes undoubtedly gives us a clue about the genome order of $k$-mers, as shown in Fig.4. Thus, in de Bruijn presentation, it is possible to avoid the complicated lexicographical presentation of list of $k$-mers. But, the de Bruijn graph is not hard to draw on the arrangement of $k$-mers lexicographically.

Now, the discussion of genome assembly is changed from reads to read-pairs that help us to appreciate as well as to understand the sophisticated techniques used by modern assemblers. Practically, we face the long repeats, but random strings are not expected to have long repeats and hence it is a simple task to assemble the reads that have been sampled from a randomly produced text. It is not difficult to show that the de Bruijn graph becomes less and less tangled with an increase of read length and when the read length exceeds the length of all repeats in a genome, the de Bruijn graph naturally turns into a path. Undoubtedly, the increasing read length helps us to identify the correct assembly, and according to biologist suggestion by using the read-pairs methodology, read length is indirectly increased, in which read-pairs are pairs of reads separated by a fixed distance $d$ in the genome. A read-pair can be thought of as a lengthy gapped read with a length of $k + d + k$ in which we know the first and last $k$-mers whereas unknown is the middle segment of length $d$.

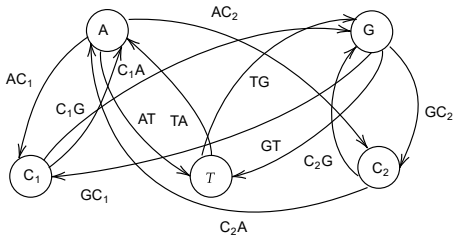
A $(k, d)$ in a paired composition is a pair of $k$-mers in text (genome) that are separated by $d$. We use the symbol $(\text{pattern}_1|\text{pattern}_2)$ to refer to a $(k, d)$-mer where $k$-mers are $\text{pattern}_1$ and $\text{pattern}_2$. The Fig.5 represents the genome for the $(3, 1)$-mers in the




| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | G | | A | G | | | | | | | | | | | A | G | | A | G | | | | | | | | |
| | G | C | | G | C | | | | | | | | | | | G | C | | G | C | | | | | | | |
| | | C | A | | C | T | | | | | | | | | | | C | T | | C | T | | | | | | |
| | | | A | G | | T | G | | | | | | | | | | | T | G | | T | G | | | | | | |
| | | | | G | C | | G | C | | | | | | | | | | | G | C | | G | C | | | | | |
| | | | | | C | T | | C | T | | | | | | | | | | | C | A | | C | T | | | |
| | | | | | | T | G | | T | G | | | | | | | | | | | A | G | | T | G | | | |
| | | | | | | | G | C | | G | C | | | | | | | | | | | G | C | | G | C | |
| | | | | | | | | C | T | | C | A | | | | | | | | | | | C | T | | C | A |
| A | G | C | A | G | C | T | G | C | T | G | C | A | | | A | G | C | ? | G | C | ? | G | C | T | G | C | A |




Graph of $\tilde{K}_{2,2}$




Graph of $\tilde{K}_{2,3}$

Figure 6: Top: A presentation of paired de Bruijn graph for a collection of nine $(2, 1)$-mers. Middle: The inadequacy of the right-side arrangement in solving the string reconstruction problem. Bottom: The forms of bidirectional bipartite graph within the de Bruijn graph representation of the 2-mer set of reads.

paired composition of our another example of reference genome `TATCCCCTCGATTCCCT`. Each identical column represents the each symbol of the genome. Here, we get two identical nodes `AT|CC`, and thus paired de Bruijn graph is formed by gluing those identically labelled nodes in the genome path in which arrows are drawn according to the previous suffix-prefix rule. In this de Bruijn network, the genome can be reassembled by taking an Eulerian path.

We have seen that an Eulerian path is always a solution to the string reconstruction problem in the de Bruijn graph building from $k$-mer composition. Therefore, it would be expected that in the paired de Bruijn graph created from a $(k, d)$-mer composition, each solution to the string reconstruction from read-paired problem corresponds to an Eulerian path. However, this is not true in all possible situations. We consider an example

of de Bruijn representation [5], shown in Fig.6, in which two Eulerian paths of $(2,1)$-mers are present : $(\texttt{AG|AG}) \xrightarrow{1} (\texttt{GC|GC}) \xrightarrow{2} (\texttt{CA|CT}) \xrightarrow{3} (\texttt{AG|TG}) \xrightarrow{4} (\texttt{GC|GC}) \xrightarrow{5} (\texttt{CT|CT}) \xrightarrow{6} (\texttt{TG|TG}) \xrightarrow{7} (\texttt{GC|GC}) \xrightarrow{8} (\texttt{CT|CA})$, and another Eulerian path is $(\texttt{AG|AG}) \xrightarrow{1} (\texttt{GC|GC}) \xrightarrow{2} (\texttt{CA|CT}) \xrightarrow{3} (\texttt{TG|TG}) \xrightarrow{4} (\texttt{GC|GC}) \xrightarrow{5} (\texttt{CA|CT}) \xrightarrow{6} (\texttt{AG|TG}) \xrightarrow{7} (\texttt{GC|GC}) \xrightarrow{8} (\texttt{CT|CA})$. Here, we notice that the second Eulerian path in the paired de Bruijn graph constructed from a $(2,1)$-mer composition does not spell out a solution to the string reconstruction from the read-pairs problem (see the middle presentation in Fig.6). In the figure presentation, the dissimilar nucleotides in a specific column are shown in different colors.

Thus, there is no doubt that the investigation of the specific structure of the de Bruijn graph is crucial in the string reconstruction problem. Here, the class of the bidirectional bipartite graph $\tilde{K}_{m,n}$ in the de Bruijn representation of the $k$-mer set of reads can play a vital role in understanding how it extends an Eulerian path that spells out a solution to the string reconstruction from the read-pairs problem. The bidirectional bipartite graph is a special type of bipartite graph; in such a graph, edges have directionality, but there are edges connecting vertices in both directions, allowing for movement or traversal in both directions between connected vertices. In mathematics, the term *vertex* is used instead of *node*. We start with the pure de Bruijn representation of the graph $\tilde{K}_{2,2}$ for the 2-mer set of reads (see the left side bottom presentation in Fig.6). For each vertex as the starting nucleotide, we get 4 Eulerian paths, i.e., a total 16 Eulerian paths out of 8 edges. For every distinct Eulerian path in the graph, there is a unique 9 bp read associated with it. As for example, the Eulerian path: $(\texttt{A}) \xrightarrow{\texttt{AC}} (\texttt{C}) \xrightarrow{\texttt{CG}} (\texttt{G}) \xrightarrow{\texttt{GC}} (\texttt{C}) \xrightarrow{\texttt{CA}} (\texttt{A}) \xrightarrow{\texttt{AT}} (\texttt{T}) \xrightarrow{\texttt{TG}} (\texttt{G}) \xrightarrow{\texttt{GT}} (\texttt{T}) \xrightarrow{\texttt{TA}} (\texttt{A})$ does spell out the read ACGCATGTA. Prominently, each Eulerian path for a set of single reads can spell out a specific long contig if using the de Bruijn representation approach we are able to construct such graph architecture by the read set.

Now, a question can arise: On the same graph structure $\tilde{K}_{2,2}$, can we reconstruct 9 bp reads using these Eulerian paths, considering eight $(2,1)$-mers? The answer is that none of these 16 Eulerian paths spells out any 9 bp read. However, parts of these Eulerian paths where the Hamiltonian condition is also satisfied can spell out shorter reads than the previous ones. As for example, following path: $(\texttt{A|C}) \xrightarrow{\texttt{AC|CA}} (\texttt{C|A}) \xrightarrow{\texttt{CG|AT}} (\texttt{G|T}) \xrightarrow{\texttt{GC|TG}} (\texttt{C|G})$ does spell out maximum 7 bp read ACGCATG which is a subread of the read of above example, where the nodes of the path are the 4 different vertices of eight $(2,1)$-mers presentation of the graph $\tilde{K}_{2,2}$. Ultimately, a query may come up in order to establish an observation rule: On what patterned nodes of an Eulerian path in the $(k,d)$-mers graphic configuration, can spell out a read? Here we set a firm rule in such a way that an Eulerian path can spell out a read, if the upper $(k-1)$-strings of the nodes can spell out the forward subread of length, (the reconstructed read length)-$(k+d)$, starting from the $1^{st}$ position of the reconstructed read, while the lower $(k-1)$-strings of the nodes can simultaneously spell out the reverse subread of length, (the reconstructed read length)-$(k+d)$, starting from the $(k+d+1)^{th}$ position of the reconstructed read.

It is clearly visible that the graph presentation of $\tilde{K}_{2,3}$ is not pure de Bruijn approach and to get the de Bruijn representation, we must glue $C_1$ and $C_2$ vertices as they represent same cytosine nucleotide at different positions (see the right side bottom presentation in Fig.6). However, as the glue vertex decreases the reconstruction length of a DNA sequence

in a pair-read structure, we can conclude that in such situation in genome assembly the graph architecture of $\tilde{K}_{m,n}$ class is more effective than the pure de Bruijn approach. Obviously, the presentations are identical with different vertices. We notice that in $\tilde{K}_{2,3}$, for ever different 48 Eulerian path can spell out maximum 48 different DNA sequences or exactly its 48 subsequences. To get the answer to another question: How many Eulerian paths can we get in the bidirectional bipartite graph $\tilde{K}_{m,n}$?, here we mention the following new formula which helps us to determine the number of Eulerian paths: $mn({}^{n}C_1 \times {}^{n-1}C_1 \times ..... \times {}^{1}C_1) + mn({}^{m}C_1 \times {}^{m-1}C_1 \times ..... \times {}^{1}C_1)$. The formula can easily be verified through the mathematical induction procedure.

In reality, prominent repeats make it impossible for biologists to deduce a specific Eulerian path and due to gaps in $k$-mer coverage, the search for an Eulerian path often fails. These two reasons lead us to the position where we are forced to break the genome into contigs. However, into a final assembly the cost of ordering the contigs and closing the gaps by utilizing a more expensive experimental method is frequently disallowed. From the de Bruijn graph, we can derive contigs in the sense that contig corresponds to strings spelled by maximal non-branching paths in the de Bruijn graph. And, if $indegree(v) = outdegree(v) = 1$ for each intermediate node $v$ of a path in a graph, the path is said to be non-branching. The term maximal is used if it is not possible to extend the path into a more longer non-branching path. As example, the de Bruijn graph in Fig.3, constructed for the 3-mer composition of `TATCCCCTCG`, has seven maximal non-branching paths that spell out the contigs: `TATC(:TA` $\rightarrow$ `AT` $\rightarrow$ `TC)`, `TCC(:TC` $\rightarrow$ `CC)`, `CCC(:CC` $\rightarrow$ `CC)`, `CCC(:CC` $\rightarrow$ `CC)`, `CCT(:CC` $\rightarrow$ `CT)`, `CTC(:CT` $\rightarrow$ `TC)`, `TCG(:TC` $\rightarrow$ `CG)`. This procedure is mentioned as the mapping of reads to contigs by backtracking graph simplifications.

Next, we define some new computational terminologies those are applied to calculate accurate distances between $k$-mers in the genome. The distance measures by the number of edges in the assembly graph. We see that a vertex $v$ in a graph $G$ is referred to as a hub if either $indegree(v) \neq 1$ or $outdegree(v) \neq 1$. A directed path in $G$ is said to be a $h$-path (hub-path) if it has hubs at the beginning and finish and non-hub vertices in the middle. Each $h$-path's first edge, known as the $h$-edge, is specific to that $h$-path. If the $h$-path starts at $\alpha$, the length of the $h$-path is shown by the expression $|\text{PATH}(\alpha)|$. Here, the normal convention is that if $a$ is the $i$-th edge in a $h$-path, H-EDGE($a$) and OFFSET($a$) are set to $\alpha$ and $i$, respectively. In the presentation of de Bruijn graph as DB(READS, $k$), $k$-mers occur in strings in READS as substrings (a string of length $k$: $(a_1, a_2, .....a_k)$), where an $h$-path passing through $n$ vertices $(a_1, a_2, .....a_{k-1}) \rightarrow (a_2, a_3, .....a_k) \rightarrow ..... \rightarrow (a_n, a_{n+1}, .....a_{n+k-2})$ defines an $(n + k - 2)$-mer $(a_1, a_2, .....a_{n+k-2})$ called a $h$-read (hub-read). The graph structure leads to a condensed de Bruijn graph substituting every $h$-path in DB(READS, $k$) by a single edge labelled by its $h$-read.

Clearly, breaking reads into $k$-mers affects the construction of the de Bruijn graphs. The chance that the $k$-mer coverage is flawless increases as the value of $k$ decreases, but more repeats are produced by smaller values of $k$, making the graph more tangled de Bruijn graph and consequently it is difficult to infer the genome from this graph. On the other hand, in the low coverage regions the larger value of $k$ may fail to detect overlaps between reads, making the graph more fragmented. Since low coverage regions are typically for SCS data, small values of $k$ are used there generally to reduce fragmentation;

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| C | A | T | C | A | G | A | T | A | G | G | A |
| C | A | T | C | | | | | | | | |
| | A | T | C | A | | | | | | | |
| | | T | C | A | G | | | | | | |
| | | | C | A | G | A | | | | | |
| | | | | A | G | A | T | | | | |
| | | | | | G | A | T | A | | | |
| | | | | | | A | T | A | G | | |
| | | | | | | | T | A | G | G | |
| | | | | | | | | A | G | G | A |
| C | | | | | | | | | G | G | A |
| C | A | | | | | | | | | G | A |
| C | A | T | | | | | | | | | A |

Figure 7: The perfect coverage of a circular genome by 4-mers in DB(READS, 3, 4). The low coverage region is defined in the sense that the three 4-mers are missing in the graph DB(READS, 4), as shown in color. The three $h$-paths of length 2 in graph DB(READS, 3) correspond to $h$-reads of the three missing 4-mers. Collectively, in multisized de Bruijn graph, all 4-mers from the genome are present.

and to reduced tangled, large values of $k$ are used in high coverage regions. According to our requirement, we are able to vary values of $k$ in the construction of the multisized de Bruijn graph DB(READS, $k-\delta, k$), provided a positive integer $\delta < k$. In particular, in low coverage regions $h$-reads of de Bruijn graph of $k-\delta$-mer composition participate to constructed of de Bruijn graph of $k$-mer composition. Bankevich et al. [1] point out an example in which in low coverage region a circular genome `CATCAGATAGGA` constructs a tangled de Bruijn graph DB(READS, 3), a fragmented de Bruijn graph DB(READS, 4) whereas on the same graphical construction the multisized de Bruijn graph BD(READS, 3, 4) is neither tangled nor fragmented (see the Fig.7).

A bimer is a triple $(a|b,d)$ in the paired de Bruijn graphs that consists of the $k$-mers $a$ and $b$ along with an integer $d$ that calculates the separation between specific instances of $a$ and $b$ in a genome. The $k$-bimer adjusted approach transfers the inexact distance estimates of the $k$-bimer set to the exact or nearly exact distance estimates of an adjusted $k$-bimer set. We get an adjusted $k$-bimer set $E(A(H(B(\text{BIREADS}))))$ by applying the four transformations one by one to the initial set of bireads BIREADS [1]. Bireads are split into $k$-bimers by the $B$-transformation in the Spades algorithm. $H$-transformation shifts a biedge $(a|b,d)$ to an $h$-biedge $H(a|b,d) = (\text{H-EDGE}(a)|\text{H-EDGE}(b), D)$ where $D = d + \text{OFFSET}(a) - \text{OFFSET}(b)$. An $h$-biedge histogram $(\alpha|\beta,*)$ is generated by applying $H$-transformation to various biedges $(a|b,d)$ with $a \in \text{PATH}(\alpha)$ and $b \in \text{PATH}(\beta)$. The operation $A(\alpha|\beta,*)$ transforms the histogram into a single adjusted $h$-biedge $(\alpha|\beta,D)$ (if the histogram only has one peak ) or a few adjusted $h$-biedge (if several peaks are present). $E$-transformation transforms an $h$-biedge $(\alpha|\beta,D)$ to $E(\alpha|\beta,D)$ as the set of all biedges $(a|b,d)$ such that $a \in \text{PATH}(\alpha)$, $b \in \text{PATH}(\beta)$ and $d + \text{OFFSET}(a) - \text{OFFSET}(b) = D$, and in such approach revealing $k$-mers with an exact distance (or nearly exact distance

estimate) make the construction of paired de Bruijn graphs practical.

Following Bankevich et al. [1], we consider a circular 24 bp genome ACGTCAAGTTCT GACGTGGGTTCT for construction of paired assembly graph of bireads sampled. The tabulated solution to the string reconstruction problem through path representation of the edge patterns of the de Bruijn graph DB(READS, 4) has been shown in Fig.8. The six $h$-paths in the order of $P_1, P_6, P_2, P_4, P_1, P_5, P_2, P_3$ ($P_1$ and $P_2$ represent repeats) passes through the graph structure in the way that 4-mer composition spells out the genome, where $P_1$ : ACGT, $P_2$ : GTTCT, $P_3$ : TCTACG, $P_4$ : TCTGACG, $P_5$ : CGTGGGTT, $P_6$ : CGTCAAGTT. The number of rows occupied by a specific path is equal to the length of that path in the representation of the tabulated form. The graph has four hubs: ACG,CGT,GTT,TCT. First four nucleotides of each path $P_i$ represents each $h$-edge $\alpha_i$. In bireads sampled, reads are paired with separation $d = 5$ (distance between the first nucleotides of the bireads), and thus we can define 13 $h$-biedges $(\alpha_i|\alpha_j, D)$ those are listed as: $R_1 = (\alpha_1|\alpha_6, 1)$, $R_2 = (\alpha_6|\alpha_6, 0)$, $R_3 = (\alpha_6|\alpha_2, 6)$, $R_4 = (\alpha_6|\alpha_4, 8)$, $R_5 = (\alpha_2|\alpha_4, 2)$, $R_6 = (\alpha_2|\alpha_1, 6)$, $R_7 = (\alpha_4|\alpha_5, 5)$, $R_8 = (\alpha_1|\alpha_5, 1)$, $R_9 = (\alpha_5|\alpha_2, 5)$, $R_{10} = (\alpha_5|\alpha_3, 7)$, $R_{11} = (\alpha_5|\alpha_1, 5)$, $R_{12} = (\alpha_2|\alpha_6, 6)$, $R_{13} = (\alpha_3|\alpha_6, 4)$. Each of $h$-biedge corresponds to a set of biedges, as for examples: $E(\alpha_6|\alpha_2, 6) = \{(\texttt{GTCA}|\texttt{GTTC}, 5), (\texttt{TCAA}|\texttt{TTCT}, 5)\}$, $E(\alpha_6|\alpha_4, 8) = \{(\texttt{CAAG}|\texttt{TCTG}, 5), (\texttt{AAGT}|\texttt{CTGA}, 5), (\texttt{AGTT}|\texttt{TGAC}, 5)\}$ in which START(FIRST$(\alpha_6|\alpha_4, 8)$) = START$(\texttt{CAAG}|\texttt{TCTG}, 5)$ = (CAA|TCT) and END(LAST$(\alpha_6|\alpha_4, 8)$) = END$(\texttt{AGTT}|\texttt{TGAC}, 5)$ = (GTT|GAC). It is obvious that there are less $h$-biedges than there are biedges. At that stage of assembling genomes using a set of exact $h$-biedges HBE, the $h$-biedges graph for a graph $G$ is constructed directly as follows:

- Create a baseline graph $G_0$ on 2|HBE|. For each $h$-biedge $(\alpha|\beta, D) \in$ HBE, introduce two new vertices $u =$ START(FIRST$(\alpha|\beta, D)), v =$ END(LAST$(\alpha|\beta, D))$ and form an edge $u \to v$. Label the resulting edge by $(\alpha|\beta, D))$.

- If two $G_0$ vertices have the same label, they should be joined.

According to these rules, here we get following $h$-biedges graph: (ACG|AAG) $\xrightarrow{R_1}$ (CGT|AGT) $\xrightarrow{R_2}$ (GTC|GTT) $\xrightarrow{R_3}$ (CAA|TCT) $\xrightarrow{R_4}$ (GTT|GAC) $\xrightarrow{R_5}$ (TTC|ACG) $\xrightarrow{R_6}$ (TCT|CGT) $\xrightarrow{R_7}$ (ACG|GGT) $\xrightarrow{R_8}$ (CGT|GTT) $\xrightarrow{R_9}$ (TGG|TCT) $\xrightarrow{R_{10}}$ (GTT|ACG) $\xrightarrow{R_{11}}$ (GTT|CGT) $\xrightarrow{R_{12}}$ (TCT|GTC) $\xrightarrow{R_{13}}$ (ACG|AAG). Each edge with vertices can be graphically presented. As, for example, the rectangle diagram of $h$-biedge $(\alpha_6|\alpha_4, 8)$ is a rectangle $(R_4)$ with sides $P_6$ and $P_4$ and $45^0$ line segment $y = x + (d - D) = x - 3$ from $(3, 0)$ to $(6, 3)$ (see the bottom figure in Fig.8). On the other hand, in case of assembling genomes using inexact $h$-biedges in the sense of inexact distance $D$, the constructed $45^0$ line shifts as compared to the correct $45^0$ line to $y = x + d - D + \delta$ (where $\delta$ is an error in $D$). Consequently, both $u$ and $v$ vertices are labelled by two different sets of bivertices where gluing is applied to all bivertices in these two sets. On the concept of gluing vertices together whenever they share at least one label, we modify the above two assembling rules to address the inexact distance complication [1].

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | C | G | T | C | A | A | G | T | T | C | T | G | A | C | G | T | G | G | G | T | T | C | T |
| A | C | G | T | | | | | | | | | | | | | | | | | | | | |
| | C | G | T | C | A | A | G | T | T | | | | | | | | | | | | | | |
| | C | G | T | C | A | A | G | T | T | | | | | | | | | | | | | | |
| | C | G | T | C | A | A | G | T | T | | | | | | | | | | | | | | |
| | C | G | T | C | A | A | G | T | T | | | | | | | | | | | | | | |
| | C | G | T | C | A | A | G | T | T | | | | | | | | | | | | | | |
| | C | G | T | C | A | A | G | T | T | | | | | | | | | | | | | | |
| | | | | | | | G | T | T | C | T | | | | | | | | | | | | |
| | | | | | | | G | T | T | C | T | | | | | | | | | | | | |
| | | | | | | | | | T | C | T | G | A | C | G | | | | | | | | |
| | | | | | | | | | T | C | T | G | A | C | G | | | | | | | | |
| | | | | | | | | | T | C | T | G | A | C | G | | | | | | | | |
| | | | | | | | | | T | C | T | G | A | C | G | | | | | | | | |
| | | | | | | | | | | | | | A | C | G | T | | | | | | | |
| | | | | | | | | | | | | | | C | G | T | G | G | G | T | T | | |
| | | | | | | | | | | | | | | C | G | T | G | G | G | T | T | | |
| | | | | | | | | | | | | | | C | G | T | G | G | G | T | T | | |
| | | | | | | | | | | | | | | C | G | T | G | G | G | T | T | | |
| | | | | | | | | | | | | | | C | G | T | G | G | G | T | T | | |
| | | | | | | | | | | | | | | | | | | | G | T | T | C | T |
| | | | | | | | | | | | | | | | | | | | G | T | T | C | T |
| A | C | G | | | | | | | | | | | | | | | | | | | T | C | T |
| A | C | G | | | | | | | | | | | | | | | | | | | T | C | T |
| A | C | G | | | | | | | | | | | | | | | | | | | T | C | T |

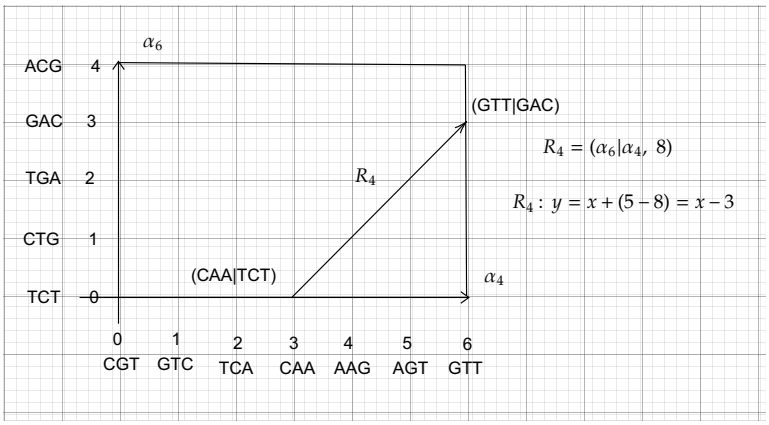



Figure 8: Top: The tabulated solution to the string reconstruction problem through path representation of the edge patterns of the de Bruijn graph DB(READS, 4), where reads are shown in color. Bottom: The rectangle diagram of h-biedge $(\alpha_6|\alpha_4, 8)$.

### 2.3 Velvet algorithm: Basic concept and preparation for computational genome assembly

We are now in the position where we perform the same previous experiment on the Galaxy platform by using Velvet assembler. Our intension is to rebuild the imaginary staphylococcus aureus bacterium via de novo assembly of a short read set. Velvet algorithm has been designed to deal short read sequencing alignments where the task is completed through the manipulation of de Bruijn graph for genomic sequence assembly via the removal of errors and the simplification of repeated regions. We import the files mutant_R1.fastq, mutant_R2.fastq and wildtype.fna from Zenodo data library [6] for assembly, where clicking on the eye icon one can view the content of data sets in the central panel. We already mentioned the key features of fastq file in which fourth line informs the quality of sequencing per nucleotide. Two files are imported here because the bacteria (reference genome) is sequenced using paired-end sequencing. Between the two files, the first file corresponds to forward reads and the second file to reverse reads.

Before doing any assembly, we evaluate the input reads using FastQC tool. Running through a standard series of tests on read set, this tool returns a relatively easy-to-interpret report. The obtained result is used to understand the quality of our fastq files, and with help of MultiQC tool, we get an aggregate outcome of the all results. On the FastQC framework, we use raw data from our current history: mutant_R1.fastq and mutant_R2.fastq, those are selected by clicking on the Multiple datasets icon. Next, MultiQC framework uses all raw data files generated by FastQC. On both datasets, MultiQC creates a webpage combining reports for FastQC. From this combining report, we check the length of the sequence, set maximum $k$-mer size for an assembly, and determine the average coverage of the genome. In our case, $L = 150$ bp is the read length, $G = 197394$ bp is the haploid genome length, $N = 2 \times 12480$ is the number of reads having 150 bp, so the average genome coverage is $\frac{NL}{G} = \frac{2\times12480\times150}{197394}$ or approximately $19\times$ coverage. The combining report also presents sequence quality histogram, where $y$-axis of mean quality score represents the quality score for each base, simply it is an estimate of the error during sequencing. We notice that the quality score is decreasing across the length of the reads because the sequence GC content, per base nitrogen content and recurring $k$-mers in reads may point to a poor quality sequencing run.

After evaluating the input reads, now we have appeared on an informational infrastructure where we want to assemble our reads to find the sequence of the imaginary staphylococcus aureus bacterium. In the first step of a de novo assembly, assembler breaks our reads into $k$-mers and builds a de Bruijn graph. Here, it is essential to assign a value of $k$ ($k$-mer size) for the assembly process. We have already noted small $k$-mers yield greater connectivity, whereas large $k$-mers yield better specificity. In the current situation, we have two files of pair-end reads, but Velvet deals with only one file where each read is next to its mate read. Before doing assembly, we combine them using FASTQ-interlacer tool, in which Types-of-paired-end-datasets parameter takes the character of 2 separate datasets, Left-hand-mates is specified by file mutant_R1.fastq and Right-hand-mates is specified by file mutant_R2.fastq. In the single file if two reads have a /1 and a /2 ending, the /1 read is used as the left-hand read and the /2 read as the right-hand read.

Velveth tool takes outputs of FASTQ-interlacer and break them in $k$-mers. For assem-

bly, we have to set the hash length, also known as $k$-mer length. Take hash length equal to *29*. Here, hash length must be an odd number, below or equal to MAXKMERHASH and strictly inferior to read length. Velvet works mainly with short reads having a fastq format where we select *single ended* for Choose-the-input-type parameter. Two output files are generated, *Sequences* and *Roadmaps*, which are necessary to Velvetg.

Velvetg tool actually does the assembly. The outputs of Velveth are used for Velvet-dataset parameter, and Using-paired-reads parameter takes the option *yes*. Along with other files, two important files are generated: the contigs.fa file and stats.txt file. Sequences of contigs greater than $2k$, where $k$ is the word length used in Velveth, are contained in the *contigs* file; and on the other hand *stats* file is a simple tabbed-delimited description of the nodes.

### 2.4 Types of error corrections

As a whole the system recognizes the following errors:

Bubbles: Error-prone reads create a barrier to real sequencing projects those are appeared in the structure where two short disjoint paths connecting the same pair of nodes in the de Bruijn graph. One path of them spells out the correct read and another representing the erroneous read. This structure is called a bubble. In the structure of a correct path along with an incorrect path, it is difficult to identify which path is correct. A de Bruijn graph may present many bubbles. Bubble removal algorithm, i.e., Tour Bus algorithm in general leaves only the correct path, but occasionally it removes the correct path too, thus introducing error rather than fixing them.

Tips: A node is considered as a tip and should be erased when it has the following characters such that it is disconnected on one of its ends, i.e., one end having total degree one, it stores the length of information shorter than $2k$ and the arc leading to this node has a low multiplicity (i.e., low appearing frequency) and consequently it cannot be compared to other alternative path. We simply say that a tip results from a single error close to the end of a read. We get a simplified graph for removed of these errors.

Erroneous connections: This type of error is created when a connection does not generate recognizable structure within the graph. To remove the error we have to specify a simple coverage cut-off value.

Bulge: A bulge occurs when the error – due to miscalled bases and indels – is in the middle of a read. It also arises from small variation between repeats in the genome.

Chimeric: A chimeric error occurs when a connection joins two unrelated parts of the graphs. Basically, it is part of erroneous connections.

Isolated h-paths: It may be considered as a part of an erroneous connection. As isolated h-paths are topologically distinct from other errors, thus it is removed after all other graph simplification procedures.

In order to deal with error in reads, we generally follow two ways: error correction in reads and bulge/bubble removal in de Bruijn graphs. However, Spades use the bulge corremoval procedure which stands for bulge correction and removal, where Spades assembler records information about removed edges from bulge for later use before discarding them. Following and assigning the specific conditions, Spades gradually remove low coverage $h$-paths occurring from sequencing errors and chimeric reads but not from repeats.

## 3 Comparative study of outcomes

On the Velvet framework, we observe that sequences of contigs longer than $2k$ are contained in a *contigs* file. The header of a *contigs* file provides the information about the $k$-mer length (called length) and $k$-mer coverage (called coverage). If the value $k$ is chosen in assembly, then the length ($k$-mer length) is a measure of how many $k$-mers overlap to give this length of target contig and the coverage ($k$-mer coverage) is a measure of how many $k$-mers overlap each base position. In our setting 222 contigs are built. To compute the information about mean, minimum, and maximum length of the contigs, we use the Datamash tool on the second column of *stats* file, where we choose *yes* option for Input-file-has-a-header-line parameter. According to outcomes, we obtain 505.39 as mean length of contigs, 1 as minimum length of contigs, and 8836 as the maximum length of contigs. In *stats* file, the second column corresponds to the length of the contig in $k$-mers, i.e., in order to obtain the real length of each of them, we need to add $k - 1 = 28$ to each length. Thus, we obtain a mean contig length of 533.39 bp, a minimum contig of 29 bp and a maximum contig of 8864 bp.

To collect more basic statistics on our assembly, we use Quast tool. We execute the operation with the following choices of parameters such that, Contigs/Scaffolds-file: *Contigs*, Types-of-assembly: *Genome*, Use-a-reference-genome?: *Yes*, Reference-genome: *wild-type.fna*, Generate-Circos-plot: *Yes*, Type-of-organism: *Prokaryotes*, Lower-threshold-for-a contig-length: *500*, Comma-separated-list-of-contig-length-thresholds: *0,1000*, and for choices of other parameters keep in their default settings. Among the various output files, we focus on the PDF report, where Lcarus is a cutting-edge genome visualizer for precise evaluation and analysis of genomic draft assemblies. To assess the quality of the assembly here many statistics are computed by Quast tool. The proportion of aligned nucleotides in the reference genome is represented by the metric genome fraction (%). It is effectively a measure of what proportion of our reference genome appeared in the assembly. We see that the genome fraction (%) takes the value 87.313. The ratio of aligned bases in the assembly to aligned bases in the reference genome is known as the duplication ratio. The value of duplication ratio is equal to 1.001 which is expected for the high value of the genome fraction (%). We note that only 53 contigs have a length greater than 500 bp, among them 43 contigs having a length greater than 1000 bp. Here, 1 misassembly event (breaking point) has been found, and it is corresponding to relocation, i.e., the left flanking sequence aligns over 1 kbp away from the right flanking sequence on the reference genome, and both flanking sequences align on the same chromosome. Misassembled contigs are the number of contigs those contain misassembly events. We observe 1 misassembled contig of length of 5823 bp. In the report, 8.11 mismatches per 100 kbp and 4.06 indels per 100 kbp are found where an insertion or deletion of bases in an organism's genome is referred to as an indel in molecular biology. Also, the report confirms that GC (%) equal to 33.64% and reference GC (%) equal to 33.43% are almost similar to each other. By dividing the total number of G and C nucleotides in the assembly by its overall length, we get the value of GC (%). The metric N50 is the length for which all contigs of that length or longer together cover at least half of an assembly, whereas the number of contigs equivalent to or longer than N50 is the measure L50; alternatively stated, L50 is the bare minimum of contigs needed to cover half the assembly.

Here, the total length of the assembly is 172665 bp and the reference length is 197394 bp, and we see that N50 takes the value 5059 bp whereas L50 is equal to 14. Through plot descriptions, the outcome report also provides distribution information about the metrics such as Nx, NAx, NGx, NGAx, the cumulative length of aligned contigs, the GC content etc.

In view of comparative study, we now measure the performance of the much more advanced assembler Spades. Basically, like Velvet, Spades goes through a process in which it uses and simplifies de Bruijn graphs, but it employs several $k$-mer size values and combines the resulting graph. This combination procedure makes Spades an advanced assembler. Here, we use metaSPAdes tool because metagenomics has emerged as a technology of choice for analyzing bacterial populations. On the metaSPAdes framework, in our setting 9 contigs are built. We obtain a mean contig length of 19951.44 bp, a minimum contig of 57 bp and a maximum contig of 132140 bp. As before, under the same setting to collect more basic statistics on our assembly we use Quast tool. Here, a value of genome fraction (%) equal to 91.301 and a value of duplication ration equal to 1.001 are found. We note that 6 contigs have a length greater than 500 bp, among them 4 contigs having a length greater than 1000 bp. On the other hand, 3 misassembly events (break points) have been found and it is corresponding to 1 relocation and 2 inversions. A misassembly event (breakpoint) is inversion where the flanking sequences align on opposite strands of the same chromosome. We observe 1 misassembled contig of length of 132140 bp. In the report, 7.83 mismatches per 100 kbp and 3.91 indels per 100 kbp are found. Also, the report confirms that GC (%) equal to 33.60% and reference GC (%) equal to 33.43%, i.e., the values are almost similar to each other. Here, the total length of the assembly is 178912 bp where the reference length is 197394 bp, and we see that as N50 takes the value 132140 bp, thus L50 is obviously equal to 1. Clearly, for each of the possible cases, we get better results than the Velvet assembler.

A summary has been set up in Fig.9 to compare the results produced by Velvet and Spades, the two assemblers. We see that in *stats* file the lengths of the contigs are given in $k$-mers, and we know how to calculate the real $k$-mer lengths of contigs. However, the $k$-mer lengths of contigs alone cannot be used to properly compute the actual length of contigs in genome assembly. The $k$-mer length is intended to show the size of the overlapping subsequences used in the assembly process, not the actual length of the contigs in base pairs. Typically, we would need further information to determine the actual length of contigs, such as the average coverage depth or the entire size of the assembled genome. The following formula can be used to predict the actual length of contigs if we know the expected size of the genome we are assembling: Actual contig length $= \frac{\text{Contig length}}{k\text{-mer length}} \times$ Genome size, and on the other hand, if we have knowledge of the average coverage depth of our sequencing data, we can apply the following method to determine the actual length of contigs: Actual contig length $= \frac{\text{Contig length}}{k\text{-mer length}} \times$ Coverage depth.

On the same set up, we consider the Circos plot presentation to demonstrate that Spades is a superior assembler over Velvet. In genomics, a circular visualization technique called a Circos plot is used to show and examine complex interactions between genetic components. Knowing the contig and coverage details, including as the lengths, orientations, and orders of individual contigs within the assembled genome, as well as the depth of sequencing coverage for each contig, is necessary to create a Circos plot. Here,

| Column 1 | Column 2 | Column 3 | Column 4 | Column 5 | Column 6 | Column 7 | Column 8 | Column 9 |
|---|---|---|---|---|---|---|---|---|
| ID | lgth | out | in | long_cov | short1_cov | short1_Ocov | short2_cov | short2_Ocov |
| 1 | 160 | 2 | 2 | 0.000000 | 20.800000 | 18.806250 | 0.000000 | 0.000000 |
| 2 | 70 | 1 | 1 | 0.000000 | 16.871429 | 14.742857 | 0.000000 | 0.000000 |
| 3 | 1875 | 2 | 2 | 0.000000 | 15.284800 | 14.182400 | 0.000000 | 0.000000 |
| 4 | 3 | 3 | 1 | 0.000000 | 74.333333 | 72.333333 | 0.000000 | 0.000000 |
| 5 | 1636 | 1 | 1 | 0.000000 | 14.766504 | 13.467604 | 0.000000 | 0.000000 |
| 6 | 9 | 1 | 1 | 0.000000 | 34.777778 | 31.333333 | 0.000000 | 0.000000 |
| 7 | 2248 | 1 | 2 | 0.000000 | 33.223754 | 30.165925 | 0.000000 | 0.000000 |
| 8 | 4610 | 2 | 2 | 0.000000 | 13.872234 | 12.728850 | 0.000000 | 0.000000 |
| 9 | 79 | 1 | 1 | 0.000000 | 21.734177 | 20.075949 | 0.000000 | 0.000000 |
| 10 | 26 | 1 | 2 | 0.000000 | 13.230769 | 13.192308 | 0.000000 | 0.000000 |
| 11 | 14 | 1 | 1 | 0.000000 | 31.357143 | 27.214286 | 0.000000 | 0.000000 |
| 12 | 550 | 2 | 2 | 0.000000 | 29.800000 | 26.960000 | 0.000000 | 0.000000 |
| 13 | 40 | 2 | 1 | 0.000000 | 20.600000 | 20.600000 | 0.000000 | 0.000000 |
| 14 | 20 | 2 | 1 | 0.000000 | 24.400000 | 24.400000 | 0.000000 | 0.000000 |

Velvet: Part of the *stats* file

| name | length | coverage |
|---|---|---|
| #name | length | coverage |
| NODE_1 | 132140 | 12.622785 |
| NODE_2 | 35102 | 12.669358 |
| NODE_3 | 5683 | 12.935856 |
| NODE_4 | 4664 | 26.001302 |
| NODE_5 | 790 | 65.740136 |
| NODE_6 | 533 | 10.723849 |
| NODE_7 | 297 | 24.425620 |
| NODE_8 | 297 | 21.004132 |
| NODE_9 | 57 | 32.500000 |

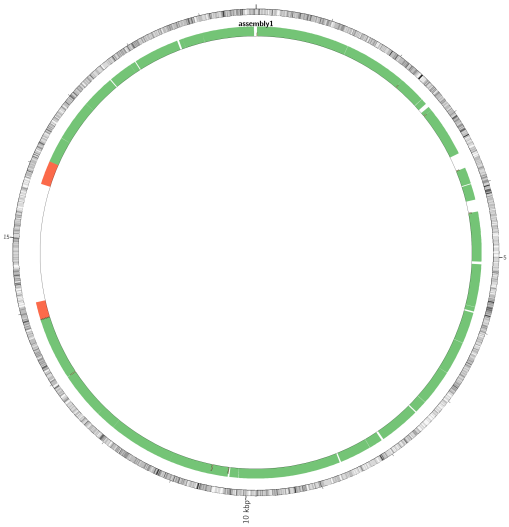
Spades: The *stats* file

| Column 1 | Column 2 |
|---|---|
| Assembly | velvetg_on_data_21__Contigs |
| # contigs (>= 0 bp) | 222 |
| # contigs (>= 1000 bp) | 43 |
| Total length (>= 0 bp) | 188302 |
| Total length (>= 1000 bp) | 165940 |
| # contigs | 53 |
| Largest contig | 8864 |
| Total length | 172665 |
| Reference length | 197394 |
| GC (%) | 33.64 |
| Reference GC (%) | 33.43 |
| N50 | 5059 |
| NG50 | 4683 |
| N90 | 1664 |
| NG90 | - |

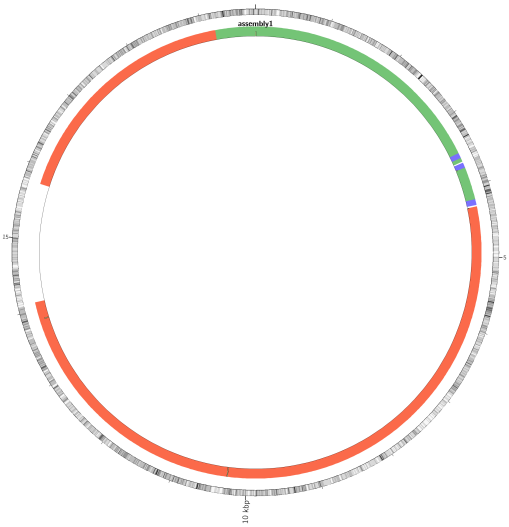
Velvet: Part of the tabular report

| Column 1 | Column 2 |
|---|---|
| Assembly | metaSPAdes_on_data_2_and_data_1__Contigs |
| # contigs (>= 0 bp) | 9 |
| # contigs (>= 1000 bp) | 4 |
| Total length (>= 0 bp) | 179563 |
| Total length (>= 1000 bp) | 177589 |
| # contigs | 6 |
| Largest contig | 132140 |
| Total length | 178912 |
| Reference length | 197394 |
| GC (%) | 33.60 |
| Reference GC (%) | 33.43 |
| N50 | 132140 |
| NG50 | 132140 |
| N90 | 35102 |
| NG90 | 790 |

Spades: Part of the tabular report




Velvet: The Circos plot




Spades: The Circos plot

Figure 9: A comparative presentation of the outcomes

the assembled genome is represented by the inner circle, while the reference genome is represented by the outermost circle. In order to find changes and differences in the assembled genome, the reference genome is used as a baseline for comparison. Gaps or breaks in the inner circle indicate regions where the genome assembler was unable to confidently determine the sequence. Regions with higher or lower sequencing coverage are highlighted by variations in color intensity in the coverage circle, which indicate potential assembly errors or areas of interest. To provide more precise information on particular genomic elements, like genes or known functional areas, annotations and labels are added to the graphic.

## 4 Conclusion

In the present study, we covered a number of topics of interest pertaining to genome assembly in the real world. In fact, there are currently no technologies available to determine the complete genomes. The complexity of this problem can be quite high which can be readily comprehensible when we compare the sizes of reads having few hundred nucleotides with the full genomes which typically contain from a few to thousands of millions of nucleotides. Apart from the size of the genome, the following factors make genome assembly as a very complex problem:

- There are two complimentary strands in DNA molecules and there is no way to tell that the taken fragment is part of the which strand.
- Most of reads are associated with inherent error because all types of genome sequencing equipments have relatively high error rates. Consequently, there are few situations the obtained fragment can be different from the original sequence.
- Some genomic regions make sequencing extremely challenging or even impossible and as a result the regions are not covered by any reads.
- In the repeated regions, it may be a complex task to identify real bubbles. The genome could have a lot of repetitive regions which is a portion of the genome that may make up 50% of the total.
- According to the results of the used algorithms, the order of the contigs is unknown.

In the face of such difficulties, a common belief is that to make this process easier one procedure is to sequence the same genome using different technologies based on particular science and mathematical theories, and in order to get the final single genome try to combine their results [18].

Here, we provide a detailed technical overview, and using the mathematical foundation of our current study, we show that the bidirectional bipartite graph structure is able to spell out relatively long contigs in both repeated and non-repeated pair-read structures. In addition to pointing out a novel formula to ascertain the total number of Eulerian paths in the bidirectional bipartite graph, we have also highlighted the observation rule for the node nucleotide patterns of an Eulerian path. It is our belief that certain graph

configurations within the partially de Bruijn representation are capable of effectively reducing the error in genome reconstruction. This is the reason why our next research endeavors need to include the search for such graph architectures.

## Acknowledgement

I would like to express my gratitude to the insightful reviewer for inspiring me to uncover new findings in the original manuscript of my work.

## References

[1] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, A. V. Pyshkin, A. V. Sirotkin, N. Vyahhi, G. Tesler, M. A. Alekseyev, P. A. Pevzner, *SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing*, Journal of Computational Biology **19 (5)** (2012), 455-477.

[2] K. Behizadi, N. Jafarzadeh, A. Iranmanesh, *Graph theoretical strategies in de novo assembly*, IEEE Access **10** (2022), 9328-9339.

[3] J. Blazewicz, M. Kasprzak, M. Kierzynka, W. Frohmberg, A. Swiercz, P. Wojciechowski, P. Zurkowski, *Graph algorithms for DNA sequencing–origins, current models and the future*, European Journal of Operational Research **264 (3)** (2018), 799-812.

[4] P. E. Compeau, P. A. Pevzner, G. Tesler, *How to apply de Bruijn graphs to genome assembly*, Nature Biotechnology **29 (11)** (2011), 987-991.

[5] P. Compeau, P. A. Pevzner, *Bioinformatics Algorithms: An Active Learning Approach*, Active Learning Publishers, 2015.

[6] S. Gladman, *An introduction to genome assembly (galaxy training materials)*, 18 - 10 (2022), URL: https://training.galaxyproject.org/training-material/topics/assembly/tutorials/general-introduction/tutorial.html, Accessed: 2022-10-21.

[7] R. M. Idury, M. S. Waterman, *A new algorithm for DNA sequence assembly*, Journal of Computational Biology **2 (2)** (1995), 291-306.

[8] D. Li, C.M. Liu, R. Luo, K. Sadakane, T. W. Lam, *MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph*, Bioinformatics **31 (10)** (2015), 1674-1676.

[9] R. Luo, B. Liu, Y. Xie, Z. Li, W. Huang, J. Yuan, G. He, Y. Chen, Q. Pan, Y. Liu and others, *SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler*, Gigascience **1 (1)** (2012), 2047-217X.

[10] Iu. P. Lysov, V. L. Florent'ev, A. A. Khorlin, K. R. Khrapko, V. V. Shik, *Determination of the nucleotide sequence of DNA using hybridization with oligonucleotides. A new method*, Dokl Akad Nauk SSSR **303 (6)** (1988), 1508-1511.

[11] P. Medvedev, S. Pham, M. Chaisson, G. Tesler, P. A. Pevzner, *Paired de bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers*, Journal of Computational Biology **18 (11)** (2011), 1625-1634.

[12] E. W. Myers, G. G. Sutton, A. L. Delcher, I. M. Dew, D. P. Fasulo, M. J. Flanigan, S. A. Kravitz, C. M. Mobarry, K. H. Reinert, K. A. Remington, E. L. Anson, R. A. Bolanos, H. H. Chou, C. M. Jordan, A. L. Halpern, S. Lonardi, E. M. Beasley, R. C. Brandon, L. Chen, P. J. Dunn, Z. Lai, Y. Liang, D. R. Nusskern, M. Zhan, Q. Zhang, X. Zheng, G. M. Rubin, M. D. Adams, J. C. Venter, *A whole-genome assembly of drosophila*, Science **287 (5461)** (2000), 2196-2204.

[13] E. W. Myers Jr, *A history of DNA sequence assembly*, IT - Information Technology **58 (3)** (2016), 126-132, URL: https://doi.org/10.1515/itit-2015-0047, Accessed: 2022-10-17.

[14] Y. Peng, H. Leung, S. M. Yiu, F. Y. Chin, *IDBA–a practical iterative de Bruijn graph de novo assembler*, Annual International Conference on Research in Computational Molecular Biology, Springer, 2010, 426-440.

[15] P. A. Pevzner, *1-Tuple DNA sequencing: computer analysis*, Journal of Biomolecular Structure and Dynamics **7 (1)** (1989), 63-73.

[16] P. A. Pevzner, H. Tang, M. S. Waterman, *An Eulerian path approach to DNA fragment assembly*, Proceedings of the National Academy of Sciences **98 (17)** (2001), 9748-9753.

[17] J. E. Quiroz-Ibarra, G. M. Mallén-Fullerton, G. Fernández-Anaya, *DNA paired fragment assembly using graph theory*, Algorithms **10 (2)** (2017), 36.

[18] M. Rocha, P. G. Ferreira, *Bioinformatics Algorithms: Design and Implementation in Python*, Academic Press, 2018.

[19] F. Sanger, S. Nicklen, A. R. Coulson, *DNA sequencing with chain-terminating inhibitors*, Proceedings of the National Academy of Sciences **74** (1977), 5463-5467.

[20] F. Sanger, A. R. Coulson, G. F. Hong, D. F. Hill, G. B. Petersen, *Nucleotide sequence of bacteriophage λ DNA*, Journal of Molecular Biology **162 (4)** (1982), 729-773.

[21] J. I. Sohn, J. W. Nam, *The present and future of de novo whole-genome assembly*, Briefings in Bioinformatics **19 (1)** (2018), 23-40.

[22] E. Southern, *Analyzing polynucleotide sequences*, International patent application **PCT/GB89/00460** (1988).

[23] B. Wajid, E. Serpedin, *Review of general algorithmic features for genome assemblers for next generation sequencers*, Genomics, Proteomics and Bioinformatics **10 (2)** (2012), 58-73.

[24] D. R. Zerbino, E. Birney, *Velvet: algorithms for de novo short read assembly using de Bruijn graphs*, Genome Research **18 (5)** (2008), 821-829.

[25] W. Zhang, J. Chen, Y. Yang, Y. Tang, J. Shang, B. Shen, *A practical comparison of de novo genome assembly software tools for next-generation sequencing technologies*, PloS One **6 (3)** (2011), e17915.

[26] Galaxy, URL: https://usegalaxy.org/, Accessed: 2022-10-21.

[27] Zenodo, URL: https://zenodo.org/record/582600#.Y1NvKUpBxN5, Accessed: 2022-10-21.

[28] Assembly using Spades, URL: https://www.melbournebioinformatics.org.au/tutorials/tutorials/assembly/spades/, Accessed: 2022-10-21.

Bijan Sarkar
Department of Mathematics
Neotia Institute of Technology, Management and Science
Diamond Harbour Road, 24 Parganas (South)
West Bengal-743368, India
E-mail: bijan0317@yahoo.com

Received: 07.02.2023
Revised: 03.01.2024
Accepted: 13.02.2024