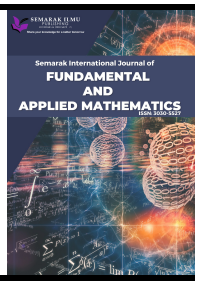
SEMARAK ILMU PUBLISHING

Semarak International Journal of Fundamental and Applied Mathematics

Journal homepage:
https://semarakilmu.my/index.php/sijfam
ISSN: 3030-5527



Graph Reduction Algorithm on Graph of 2-mers for DNA Sequences

Kee Yeong Chua[1], Wan Heng Fong[1,*], Sherzod Turaev[2]

[1] Department of Mathematical Sciences, Faculty of Science, Universiti Teknologi Malaysia, 81310, Johor Bharu, Johor, Malaysia
[2] Department of Computer Science and Software Engineering, College of Information Technology, United Arab Emirates University, P.O. Box 15551. Al Ain, United Arab Emirates

ARTICLE INFO

Article history:
Received 30 December 2025
Received in revised form 25 February 2026
Accepted 10 May 2026
Available online 8 June 2026

Keywords:
DNA sequences; graph theory; graph reduction; algorithm

ABSTRACT

In bioinformatics, graph-theoretical techniques are used to mathematically model DNA sequences. The term graph of k-mers refers to a directed and weighted graph which uses DNA subsequences or fragments of length $k \in \mathbb{N}$ as the vertices. The edges of this graph correspond to the bases between the vertices. In this research, a graph reduction algorithm on the graph of 2-mers is designed in C++ programming where duplicate vertices are combined and unnecessary edges are removed. The output of the algorithm presents a reduced graph which preserves essential information of the corresponding DNA sequence. This algorithm provides an automated and scalable framework for DNA sequence analysis by applying graph theory to bioinformatics.

1. Introduction

Deoxyribonucleic acid (DNA) is a macromolecule formed by smaller units called nucleotides which consists of a sugar, a phosphate group and one of the four nitrogenous bases, namely, Adenine (A), Cytosine (C), Guanine (G), and Thymine (T) [1]. The structure of DNA is a right-handed double helix which is formed by chemically chaining together two long strands of nucleotides where the sugar and phosphate on the outside are the backbone of the structure and the bases are located on the inside of the structure [2]. In the chemical bond of DNA, the stability arises from the existence of the hydrogen bonds of complementary base pairing where A is exclusively paired with T by two hydrogen bonds; while C is paired exclusively with G by three hydrogen bonds [3]. Here, a DNA sequence is the specific order of the bases which gives genetic instructions for the development and function of a living organism. Since the length of a DNA sequence can vary from a few bases to billions of bases, short reads of DNA are frequently used in research [4]. These short reads are also known as the k-mers of the sequence which enable sequence analysis through graph theory [5].

In graph theory, graphs which consist of vertices and edges can be used to model DNA sequences. One example is the de Bruijn graph for DNA sequences where the k-mers of the DNA bases are the vertices of the graph while the edges represent the overlaps between the k-mers [6]. Although de Bruijn graphs are the most used graph in bioinformatics for genome assembly and gene editing, it

* *Corresponding author.*
E-mail address: fwh@utm.my

fails to present the distance information between the k-mers of the DNA sequence [7]. To address these limitations, a distance-based graph known as the graph of k-mers for DNA sequences is introduced. This graph uses the consecutive k-mers in a DNA sequence as the vertices of the graph; while the non-overlapping bases between the vertices serve as the edges of the graph.

As previously stated, there can be up to billions of bases in a DNA sequence. Therefore, the graph formed by a DNA sequence is potentially large and complex. A graph reduction method, namely graph coarsening, can be used to reduce the size of the graph [8]. The technique of graph coarsening combines vertices by grouping them according to various properties instead of removing the vertices as in other graph reduction techniques such as graph sparcification and graph condensation [9]. This prevents the loss of genetic information during the graph coarsening for the graph of k-mers. Furthermore, when storing DNA sequences as computer data, the reduced graph produced by the coarsening is more compact as compared to directly storing large amounts of bases or large graph.

In this research, an algorithm is designed in C++ programming for the graph reduction of the graph of 2-mers using graph coarsening. This algorithm aims to decrease both the time taken and the error rate during manual calculation for graph reduction. In addition, this algorithm bridges the gap of biology and mathematics by providing a platform where biologist can use the algorithm to gain insights regarding structure of the DNA sequence.

2. Literature Review

Graph-based modelling of biological sequences, particularly DNA, has become an effective tool in modern bioinformatics [10]. This approach transforms complex sequential data into mathematical structures known as graphs, enabling analysis through established graph algorithms such as Hamiltonian and Eulerian paths [11]. This literature review section discusses the key concepts used in this research, focusing on k-mers, de Bruijn graphs, graph coarsening, and computational data types and structures.

The fundamental unit for graph-based DNA sequence analysis are the k-mers, a substring of length $k \in \mathbb{N}$ extracted from a longer sequence [12]. The concept of k-mers has been used in numerous applications, including genome assembly, sequence alignment, and metagenomic classification [13]. Analyzing the frequency, distribution, and adjacency of these k-mers can also provide insights on the composition, repeats, and sequencing errors in a DNA sequence. The value of k is also important as smaller values increase sensitivity to repeats and errors, while larger values offer more uniqueness but demand higher computational resources [14].

The most common graph model in genomics which uses the k-mers is the de Bruijn graph [15]. In this model, each unique k-mer is represented as a vertex, and a directed edge connects two vertices if the suffix of length $k-1$ of a vertex equals the prefix of another vertex [16]. This structure represents sequence overlaps and is used in *de novo* genome assemblers [15]. However, a limitation of the de Bruijn graph is its loss of positional and distance information between each k-mer [7]. Therefore, to address this limitation, the graph of k-mers is introduced in this research to preserve distance information between k-mers in DNA sequences, making it suitable for studies of sequence structure, repeat spacing, and mapping.

Meanwhile, graph coarsening is a method in graph reduction where vertices are grouped according to their properties. This produces a coarsened graph of reduced size while preserving important graph properties such as the spectral properties of a graph [17]. This method is especially useful in machine learning in which graph coarsening is used for Graph Neural Network (GNN) acceleration, where large graphs are coarsened to reduce training and inference time while attempting to maintain model performance [18].

Next, the literature review proceeds with a discussion on the data types and structures used in C++ programming. Here, data types such as “int” and “string” are used to classify different types of data, signalling the complier on how the data should be used and stored in memory [19]. Data structures, however, is an organized collection of items for efficient access, modification, and processing [20]. In this research, the two main data structures that are used are “queue” and “map”. A queue is a linear data structure that follows the First In, First Out (FIFO) principle [21]. This means the first element added to the queue is the first one to be removed. The data structure of map stores elements as a collection of unique key-value pairs, where a unique key is used to access its associated value [21].

3. Methodology

In this section, the methodology of this research is discussed, starting with the construction of the graph of 2-mers. For this graph, the vertices are the consecutive 2-mers of a DNA sequence. For example, given the DNA sequence of TGTGAGT, the vertices in the graph of 2-mers are $v_1 =$ TG, $v_2 =$ GT, $v_3 =$ TG, $v_4 =$ GA, $v_5 =$ AG, and $v_6 =$ GT. A visual example for the graph of 2-mers for TGTGAGT is shown in **Figure 1**. Note that the brackets indicate the DNA bases for the edge weight.

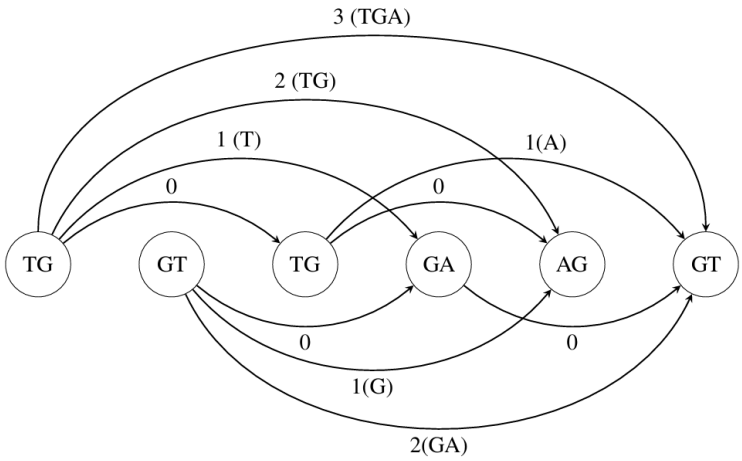


Fig. 1. Graph of 2-mers for TGTGAGT

Here, the number of vertices is always the length of the DNA sequence minus one. Next, the edges of the graph of 2-mers are directed and are only formed between vertices with non-overlapping DNA bases. For instance, v_1 and v_2 have an overlapping base of G, hence no edge can be formed. However, v_1 and v_3 does not have overlapping bases, hence an edge can be formed. In addition, the edge weight for the graph of 2-mers is obtained from the number of DNA bases between the vertices. Hence, the edge connecting v_1 to v_3 will have an edge weight of zero as there are no bases between these two vertices; while the edge connecting v_1 to v_5 has an edge weight of two as the bases of T and G are between the vertices.

The graph reduction for the graph of 2-mers is then done by method of graph coarsening where duplicate vertices are merged. In **Figure 1** there are two instances of the vertices TG and GT, therefore, these vertices are merged, leaving only four vertices in the reduced graph, namely $v_1 =$ TG, $v_2 =$ GT, $v_3 =$ GA, and $v_4 =$ AG. The graph after merging the vertices is shown in **Figure 2**.

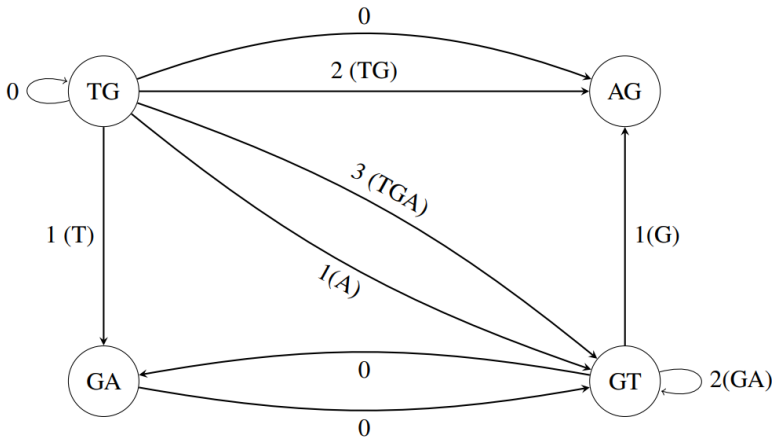


Fig. 2. Graph of 2-mers for TGTGAGT after vertex merge

In **Figure 2** it can be observed that the graph has loops and multiple edges connecting the vertices. Thus, only the edge with lowest edge weight is kept to form the reduced graph. Hence, the reduced graph of 2-mers for TGTGAGT is shown in **Figure 3**.

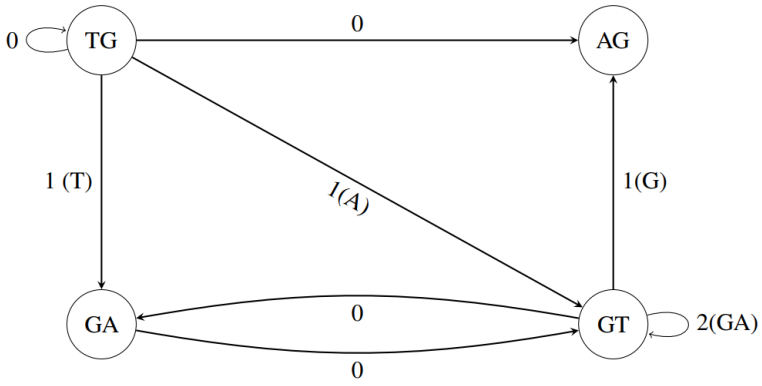


Fig. 3. Reduced graph of 2-mers for TGTGAGT

From this process of reducing the graph of 2-mers, the flowchart of the algorithm is formed, as presented in **Figure 4**.

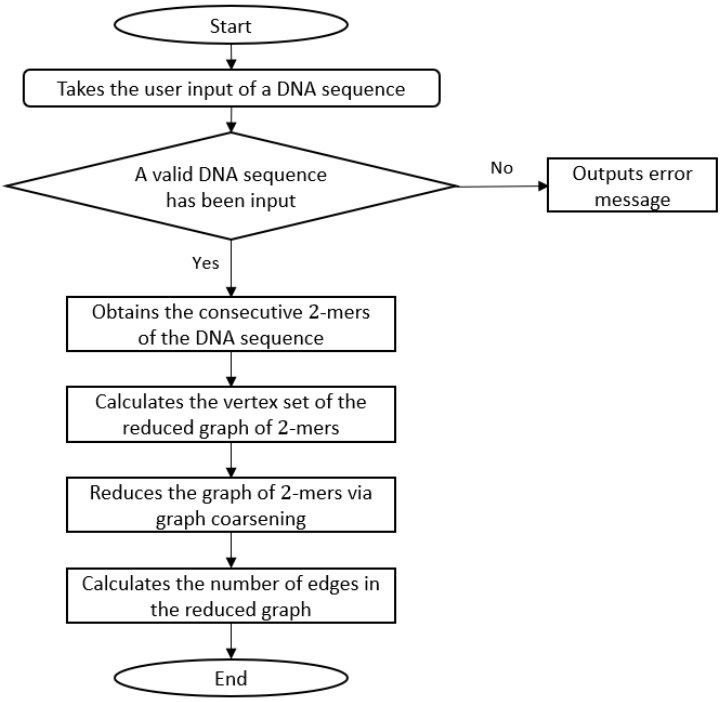


Fig. 4. Flowchart of the algorithm

From **Figure 4**, a validation algorithm is first used to check whether a user has input a valid DNA sequence. This is followed by the algorithm to obtain the consecutive 2-mers of the DNA sequence, which corresponds to the vertices of the graph of 2-mers. Next, an algorithm for obtaining the vertex set of the reduced graph of 2-mers is designed. This algorithm corresponds with the vertex merging in reducing the graph. In addition, an algorithm for reducing the graph in order to keep the lowest edge weight is also designed.

4. Results and Discussion

In this research, an algorithm is designed in order to obtain the reduced graph of 2-mers. For the clarity of discussion, the algorithm is split into five. The first algorithm is for checking the validity of a user input, shown in **Algorithm 1**.

Algorithm 1 Checking input validity

1.) **START**
2.) **DECLARE** the string of *dna_seq*
3.) the integers of *length* and *sum*
4.) **OBTAIN** *dna_seq* from user
5.) **SET** *length* to the number of characters in *dna_seq*
6.) **SET** $sum = 0$
7.) **COUNT** the number of "A" in *dna_seq* and add the number to *sum*
8.) **REPEAT** step 6 with "C", "G", and "T"
9.) **IF** $sum = length$ **THEN**
10.) **OUTPUT** "Valid input"
11.) **ELSE**
12.) **OUTPUT** "Invalid input"

Algorithm 1 (cont.)

13.) **END IF**
14.) **END**

Algorithm 1 validates a user input of a DNA sequence by ensuring that it exclusively consists the four DNA bases which are A (adenine), C (cytosine), G (guanine), and T (thymine). This algorithm works by first calculating the total length of the input string and then computing a sum by counting the occurrences of each of the four valid characters. If the sum of all A, C, G, and T equals the total length of the string, then every character in the sequence is a valid nucleotide, and the program outputs "Valid input". If the sum is less than the length, it means the string contains at least one invalid character such as “B”, “5”, or “t”, and the program outputs "Invalid input".

The next algorithm continues with obtaining the 2-mers of the input DNA sequence which is presented in **Algorithm 2**.

Algorithm 2 Obtaining the 2-mers of the input DNA sequence

```
START
DECLARE the string of dna_seq and pair
        the integers length and i
        the queue two_mers
SET length to the number of characters in dna_seq
FOR i = 1 TO i = length
    SET pair to be the concatenation of the ith and i + 1th character of dna_seq
    ENQUEUE pair into two_mers
END FOR
RETURN two_mers
PRINT two_mers
END
```

This algorithm works by iterating through the sequence one character at a time, starting from the first position, and concatenating each character with the following character to form a pair. Each pair is then added into a queue named as “two_mers”. For example, the sequence "ACTG" would produce the 2-mers of AC, CT, and TG. After processing the entire sequence, the algorithm returns the queue containing all consecutive 2-mers of the DNA sequence so that it can be used in the subsequent algorithms.

Following that, **Algorithm 3** which is used to obtain the vertex set of the reduced graph of 2-mers is presented.

Algorithm 3 Obtaining the vertex set of the reduced graph of 2-mers

```
START
DECLARE the string of dna_seq
        the array of strings all_two_mers
        the queue two_mers and reduced_vertices
SET all_two_mers to to have all possible 2-mers in a DNA sequence (i.e. [AA, AC, …])
FOR each 2-mer in all_two_mers
    COUNT the number of occurrences of the 2-mer in two_mers
    IF the number of occurrences is more than 0 THEN
    ENQUEUE the current 2-mer into reduced_vertices
```

Algorithm 3 (cont.)

```
    END IF
END FOR
RETURN reduced_vertices
PRINT reduced_vertices
END
```

This algorithm generates the vertex set for the reduced graph of 2-mers by creating an array named *all_two_mers* which lists all 16 possible 2-mers for the four DNA bases, namely, AA, AC, AG,

AT, CA, CC, CG, CT, GA, GC, GG, GT, TA, TC, TG, and TT. The algorithm then checks this list with the queue *two_mers* from **Algorithm 2**. Here, only the 2-mers that appear at least once in *two_mers* are added to another queue called *reduced_vertices* which represent the unique 2-mers that are in the input DNA sequence.

The next algorithm which is shown in **Algorithm 4** reduces the graph of 2-mers via graph coarsening method.

Algorithm 4 Reducing the graph of 2-mers via graph coarsening method

```
1.)  START
2.)  DECLARE the queue of two_mers and reduced_vertices
3.)            the integers of distance and min
4.)            the map of red_graph
5.)  SET red_graph to have all possible pairings of 2-mers from reduced_vertices
6.)       note that these pairings are denoted as (start, end)
7.)  SET all the pairings of 2-mers in red_graph to infinity
8.)  FOR each pair, (start, end)
9.)       OBTAIN all positions of start and end from two_mers
10.)      SET min= infinity
11.)      FOR each position of start and end
12.)           SET distance = start position − end position−2
13.)           IF distance < min and distance > 0
14.)                SET min = distance
15.)           END IF
16.)      END FOR
17.)      MAP (start, end) to min in red_graph
18.) END FOR
19.) RETURN red_graph
20.) PRINT red_graph
21.) END
```

Algorithm 4 constructs the reduced graph of 2-mers or *red_graph*, where vertices are the 2-mers obtained from *reduced_vertices*. This algorithm first creates all possible directed pairs of vertices, which represents edges in the reduced graph, and initializes each pair of vertices to infinity. Then, for each pair of vertices, the position of the 2-mers represented by the start and end vertices are found in the *two_mers* queue. Next, for all positions obtained, the formula of "start position − end position−2" is used to calculate the number of bases between the vertices where only the minimum value is kept. This is as shown in line 12 and 13 of **Algorithm 4**. This minimum value is then mapped to the corresponding vertex pairs in *red_graph*.

Furthermore, **Algorithm 5** shows how the number of edges is obtained from the reduced graph obtained in **Algorithm 4**.

Algorithm 5 Calculating the number of edges in the graph of 2-mers

```
1.) START
2.) DECLARE the map of red_graph
3.)            the integer of edge_num
4.) COUNT the number of pairs in red_graph that is not mapped to infinity
5.) SET edge_num to the number obtained
6.) RETURN edge_num
7.) END
```

Since the graph was initialized with all possible edges set to infinity, only vertex pairs with a minimum value calculated during the coarsening process have a finite value. Hence, **Algorithm 5** counts the number of finite value edges and returns that total as *edge_num*.

Lastly, the algorithms have been developed into a C++ programming code. The visual output after running the C++ program is shown in **Figure 5** where a prompt to input a DNA sequence is given to the user.

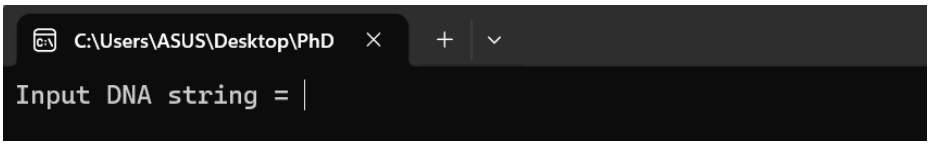


Fig. 5. Prompt given after running the C++ program

If an invalid DNA sequence is input such as “CCAgTT1”, an error notification is output to the user as shown in **Figure 6**.

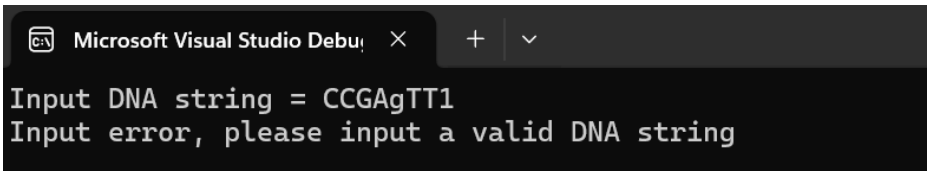


Fig. 6. Error given after an invalid DNA sequence of “CCAgTT1” is input

The algorithm only gives the reduced graph after a valid DNA sequence is input, which is shown in **Figure 7** where the previous DNA sequence of TGTGAGT is input.

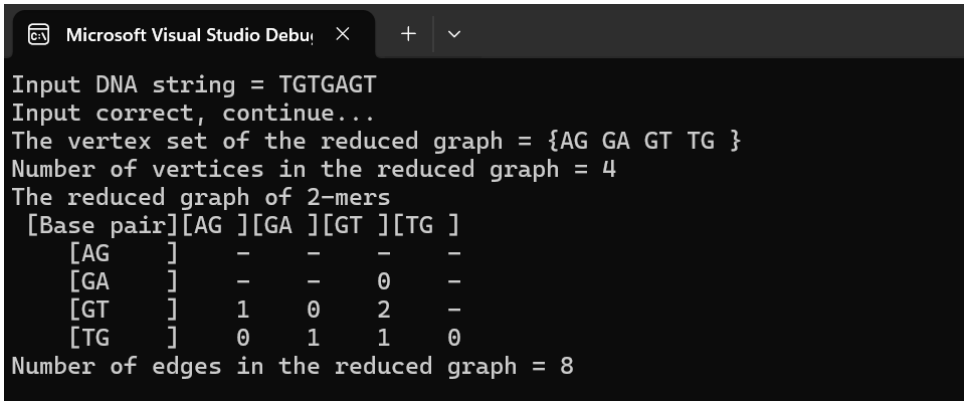


Fig. 7. The reduced graph after a valid DNA sequence is input

Next, the output of the reduced graph is presented in table form due to the constraint of C++ programming which needs an external library to visualize the graphs. In the table, the row labels indicate the start vertices; while the column labels indicate the end vertices. Furthermore, each entry in the table specifies the weight of the directed edge from the start vertex to the end vertex. A numerical entry indicates the edge weight, and a dash indicates that there are no edges. For example, the value of “1” in row three and column one in the table shows that the edge directing from GT to AG has an edge weight of one. It should also be stated that the graph obtained from the table in **Figure 7** is the same as the reduced graph of 2-mers obtained in **Figure 3**, where both graphs have the same number of vertices and edges, which are four and eight respectively. The two graphs also

have four edges with edge weight of zero, three edges with edge weight of one, and one edge with edge weight of two.

5. Conclusion

In conclusion, this research designed a C++-based graph reduction algorithm for DNA sequences using the graph of 2-mers model. By applying graph reduction techniques to merge duplicate vertices and to retain edges with lowest edge weight, the algorithm transforms complex DNA sequences into graphical representations that preserve distance information. This algorithm thus provides a systematic and automated tool for DNA sequence analysis through computational graph theory. Future works for this research include extending the algorithm to larger k-mers and parallelization for more efficient analysis of large-scale genomic sequences.

Acknowledgements

The first author would like to acknowledge the Ministry of Higer Education of Malaysia for the financial support through the scholarship programme of MyBrainSc.

References

[1] Alberts, B., A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. "DNA, chromosomes, and genomes." *Molecular biology of the cell* (2014): 173-236. https://doi.org/10.1201/9781315815015-5

[2] Watson, James D., and Francis HC Crick. "The structure of DNA." In *Cold Spring Harbor symposia on quantitative biology*, vol. 18, pp. 123-131. Cold Spring Harbor Laboratory Press, 1953. https://doi.org/10.1101/SQB.1953.018.01.020

[3] Yakovchuk, Peter, Ekaterina Protozanova, and Maxim D. Frank-Kamenetskii. "Base-stacking and base-pairing contributions into thermal stability of the DNA double helix." *Nucleic acids research* 34, no. 2 (2006): 564-574. https://doi.org/10.1093/nar/gkj454

[4] Jin, Xin, Qian Jiang, Yanyan Chen, Shin-Jye Lee, Rencan Nie, Shaowen Yao, Dongming Zhou, and Kangjian He. "Similarity/dissimilarity calculation methods of DNA sequences: A survey." *Journal of Molecular Graphics and Modelling* 76 (2017): 342-355. https://doi.org/10.1016/j.jmgm.2017.07.019

[5] Rahman, Atif, Ingileif Hallgrímsdóttir, Michael Eisen, and Lior Pachter. "Association mapping from sequencing reads using k-mers." *Elife* 7 (2018): e32920. https://doi.org/10.7554/eLife.32920

[6] Etzion, Tuvi. *Sequences and the de Bruijn Graph: Properties, Constructions, and Applications*. Elsevier, 2024. https://doi.org/10.1016/B978-0-44-313517-0.00008-1

[7] Turner, Isaac, Kiran V. Garimella, Zamin Iqbal, and Gil McVean. "Integrating long-range connectivity information into de Bruijn graphs." *Bioinformatics* 34, no. 15 (2018): 2556-2565. https://doi.org/10.1093/bioinformatics/bty157

[8] Chen, Jie, Yousef Saad, and Zechen Zhang. "Graph coarsening: from scientific computing to machine learning." *SeMA Journal* 79, no. 1 (2022): 187-223. https://doi.org/10.1007/s40324-021-00282-x

[9] Hashemi, Mohammad, Shengbo Gong, Juntong Ni, Wenqi Fan, B. Aditya Prakash, and Wei Jin. "A comprehensive survey on graph reduction: Sparsification, coarsening, and condensation." *arXiv preprint arXiv:2402.03358* (2024). https://doi.org/10.48550/arXiv.2402.03358

[10] Khan, Riaz Hussain, Nadeem Salamat, A. Q. Baig, Zaffar Ahmed Shaikh, and Amr Yousef. "Graph-based analysis of DNA sequence comparison in closed cotton species: a generalized method to unveil genetic connections." *PLoS One* 19, no. 9 (2024): e0306608. https://doi.org/10.1371/journal.pone.0306608

[11] Ashton, Banda. "Graph theory in DNA sequencing: Unveiling genetic patterns." *International Journal of Biology and Life Sciences* 3, no. 1 (2023): 9-13.https://doi.org/10.54097/ijbls.v3i1.9593

[12] Mouratidis, Ioannis, Nikol Chantzi, Umair Khan, Maxwell A. Konnaris, Candace SY Chan, Manvita Mareboina, Camille Moeckel, and Ilias Georgakopoulos-Soares. "Frequentmers-a novel way to look at metagenomic next generation sequencing data and an application in detecting liver cirrhosis." *BMC genomics* 24, no. 1 (2023): 768. https://doi.org/10.1186/s12864-023-09861-w

[13] Moeckel, Camille, Manvita Mareboina, Maxwell A. Konnaris, Candace SY Chan, Ioannis Mouratidis, Austin Montgomery, Nikol Chantzi, Georgios A. Pavlopoulos, and Ilias Georgakopoulos-Soares. "A survey of k-mer

methods and applications in bioinformatics." *Computational and Structural Biotechnology Journal* 23 (2024): 2289-2303. https://doi.org/10.1016/j.csbj.2024.05.025

[14] Chikhi, Rayan, and Paul Medvedev. "Informed and automated k-mer size selection for genome assembly." *Bioinformatics* 30, no. 1 (2014): 31-37. https://doi.org/10.1093/bioinformatics/btt310

[15] Rizzi, Raffaella, Stefano Beretta, Murray Patterson, Yuri Pirola, Marco Previtali, Gianluca Della Vedova, and Paola Bonizzoni. "Overlap graphs and de Bruijn graphs: data structures for de novo genome assembly in the big data era." *Quantitative Biology* 7, no. 4 (2019): 278-292. https://doi.org/10.1007/s40484-019-0181-x

[16] Huang, Shien, Hang Zhang, and Ergude Bao. "A comprehensive review of the de bruijn graph and its interdisciplinary applications in computing." *Engineered Science* 28 (2023): 1061. http://dx.doi.org/10.30919/es1061

[17] Jin, Yu, Andreas Loukas, and Joseph JaJa. "Graph coarsening with preserved spectral properties." In *International Conference on Artificial Intelligence and Statistics*, pp. 4452-4462. PMLR, 2020.

[18] Cai, Chen, Dingkang Wang, and Yusu Wang. "Graph coarsening with neural networks." *arXiv preprint arXiv:2102.01350* (2021). https://doi.org/10.48550/arXiv.2102.01350

[19] Hohmuth, Michael, and Hendrik Tews. "The semantics of C++ data types: Towards verifying low-level system components." In *Theorem Proving in Higher Order Logics*, pp. 127-144. 2003. https://doi.org/10.1007/b11935

[20] Ford, William, Ford William, and William Topp. *Data structures with C++*. Simon & Schuster, Inc., 1995.

[21] Goodrich, Michael T., Roberto Tamassia, and David M. Mount. *Data structures and algorithms in C++*. John Wiley & Sons, 2011.