MATCH
Communications in Mathematical and in Computer Chemistry

MATCH Commun. Math. Comput. Chem. **83** (2020) 33-62

ISSN 0340 - 6253

On Graph–Based Data Structures to Multiple Genome Alignment

Nafiseh Jafarzadeh, Ali Iranmanesh*

Department of Mathematics, Faculty of Mathematical Sciences,
Tarbiat Modares University, P.O. Box: 14115-137,
Tehran, Iran

(Received August 1, 2019)

Abstract

Multiple genome alignment (MGA) has become widely used in many different areas in bioinformatics from finding sequences family, detecting structural homologies of genomes, and predicting functions of genomes to predict patient's diseases by comparing DNAs of patients in disease discovery. The potential of graphs to intuitively represent all aspects of genome alignments led to the development of graph-based approaches for genome alignment. These approaches construct a graph from a set of local alignments, and derive a genome alignment through identification and removal of graph substructures that indicate errors in the alignment. This review summarizes the presented graphical representation of multiple alignment in large scales and their abilities to represent alignment information and also the role of graph data structures to assist in the development of future genome alignment tools.

1 Introduction

The rapid increment of biological sequences in next generation sequencing (NGS) techniques has highlighted the key role of multiple genome alignment in comparative structure and function analysis of biological sequences. Sequence alignment is usually the first step

*Corresponding author.

Email address: iranmanesh@modares.ac.ir

performed in bioinformatics to understand the molecular phylogeny of an unknown sequence. This is done by aligning the unknown sequence with one or more known database sequences to predict the common portions as the residues perform functional and structural role tend to be preserved by natural selection in the course of evolution [1]. The sequences may be nucleotide sequences (DNAs or RNAs) or amino acid sequences (Proteins). The rearrangement process may introduce one or more spaces or gaps in the alignment. A gap indicates a possible loss or gain of a residue; thus, evolutionary insertion or deletion, translocations and inversion events can be observed in sequence alignment.

There are two types of sequence alignment; pairwise sequence alignment and multiple sequence alignment. The first type, considers two sequences at a time where as the second one aligns multiple (more than two) related sequences. Multiple sequence alignment is more advantageous than pairwise sequence alignment as it considers multiple members of a sequence family and thus provides more biological information. Multiple alignment is also a prerequisite to comparative genomic analyses for identification and quantification of conserved regions or functional motifs in a whole sequence family, estimation of evolutionary divergence between sequences and even for ancestral sequence profiling [2]. In this paper, we investigate about multiple genome alignment. The approach to multiple alignment can be global alignment [3] or local alignment [4]. Global alignment is done when the similarity is counted over the entire length of the sequences. Several multiple alignment techniques perform global alignment but difficulties arise when sequences are only homologous over local regions where clear block of ungapped alignment common to all of the sequences or if there is presence of shuffled domains among the related sequences [5]. In such cases, local alignment is performed to know the local similar regions among the sequences. When there is a large difference in the lengths of the sequences to be compared, local alignment is generally performed [6].

As an effective modeling, analysis and computational tool, graph theory is widely used in biological mathematics to deal with various biology problems like sequence comparison. There are several biological problems that can be treated with graph theoretical methods. In [7-22] you can see some mathematical methods as graphical and numerical representations for comparison of genomes. The graphs have proven to be powerful tools for coping with the complexity of genome-scale sequence alignments. For the step of selecting subsets of local alignments and for inducing a segmentation, graph data structures serve as a convenient tool. The idea is that graphs show substructures indicating errors in the alignment, thus they can assist in improving genome alignments. In addition, graphs provide an intuitive representation of similarities and changes between genomes, and so visualize alignment structures. Also by

having a graph data structure for a multiple genome alignment, it is possible to model colinear and non-colinear changes without the need of choosing a reference genome. In this investigation, we first, briefly introduce two categories of approaches which are frequently used in multiple genome alignment algorithms; exact and heuristic approaches. Then we describe ten data structures of commonly used graphs in terms of their abilities to represent alignment information.

2 Multiple alignment Algorithms

A wide range of computational algorithms have been applied to the multiple alignment problem. The purpose of a multiple alignment algorithm is to assemble alignments reflecting the biological relationship between several sequences. These algorithms can be broadly classified into exact algorithms and heuristic algorithms. In this section, we review the most important exact and heuristic algorithms including dynamic programming, combinatorial algorithms, progressive alignment and Anchor-based alignment. The exact algorithms solve the multiple alignment score maximization problem optimally using either natural extensions of the dynamic programming algorithm [23-26] or a graph theoretic formulation which facilitates the use of combinatorial algorithms [27-29]. In practice, however, optimal methods are only feasible for a few, relatively short sequences. Hence, many fast and accurate heuristics to solve the multiple alignment problem have been proposed. The rapid accumulation of sequence data demanded the development of heuristic methods that are able to align more sequences of greater length than the optimal methods with exponential runtime. For an ordinary protein alignment, the predominant heuristic strategy is called progressive alignment. For genomic DNA sequences, most tools use a heuristic called anchor-based alignment.

2.1 Dynamic programming

Dynamic programming (DP) is a mathematical and computational method which refers to simplifying a complicated problem by subdividing it into smaller and simpler components in a repeated manner. The dynamic programming technique can be applied to global alignments by using methods such as the Needleman-Wunsch algorithm [19] and local alignments by using the Smith-Waterman algorithm [20]. Up to 1980, the traditional multiple sequence alignment algorithms were only best suited for two sequences, so when it came to producing multiple sequence alignment with more than two sequences, it was found that completing the alignment manually was faster than using traditional dynamic programming algorithms [30]. Dynamic

programming algorithms are used for calculating pairwise, this method could be extended to more than two sequences however in practice it is too complex, because the time and space complexity becomes very large [31].

Dynamic programming is originally a method used in the area of optimization in mathematics developed by Richard Bellman in the 1940's [30]. Dynamic programing is solving complex problems by breaking them into a simpler sub-problems. Problem can be divided into many smaller parts. Needleman and Wunsch [19] were the first to propose this method and describes general algorithm for sequence alignment.

In computer science dynamic programming is a programming method based on dividing a problem into sub-problems. Each of the sub-problems are divided further into sub-problems and so on until some base case is reached. Thus a given problem can be defined by some sub problems. An example of dynamic programming could be presented as a table of size $n \times m$ (for two dimensions) where the entry in $[|m|, |n|]$ is the solution to the problem. In order to get this solution the entries$[m-1, n]$, $[m-1, n-1]$ and $[m, n-1]$ need to be calculated and so fourth, until the base case [0,0] is reached. Instead of calculating this recursively the table can just be filled from top to bottom which is called dynamic programming since each entry depends on the previous entries. In the following, a recursive formula according to a dynamic programming table for two sequences, S_1 and S_2 is presented.

$$\mathrm{D(i,j)} = \begin{cases} 0 & \text{if } \mathrm{i} = 0 \text{ and } \mathrm{j} = 0 \\ \mathrm{Max}\big(\mathrm{D(i-1,j)} + \mathrm{C}, \mathrm{D(i-1,j-1)} + \mathrm{S}\big(\mathrm{S}_{1i}, \mathrm{S}_{2j}\big), \mathrm{D(i,j-1)} + \mathrm{C}\big); & \text{else} \end{cases}$$

In the above, $D(x, y)$ is an entry in the dynamic programming table, $S(Sx, Sy)$ is the score for aligning two bases (or proteins) and C is the cost for making a gap. A gap in sequence alignment is denoted with a '-'. Gaps are used in alignments when the sequences do not align well with each other. Then one sequence can be pushed one index, by inserting a gap in it. This way the sequences might align better. The biological interpretation of a gap is that either a deletion or an insertion has happened during the evolution of one of the sequences. There are some boundary cases which have to be taken into account when programming the recursion. It is assumed that the best score is the maximum. The dynamic programming method takes time $O\ (n \times m)$, like the dot matrix method where n and m are the lengths of the sequences.

2.2 Combinatorial algorithms

In applied mathematics and theoretical computer science, combinatorial optimization is a topic that consists of finding an optimal object from a finite set of objects. In many such problems,

exhaustive search is not tractable. It operates on the domain of those optimization problems, in which the set of feasible solutions is discrete or can be reduced to discrete, and in which the goal is to find the best solution. Some common problems involving combinatorial optimization are the travelling salesman problem, the minimum spanning tree problem, and the knapsack problem. Combinatorial optimization is a subset of mathematical optimization that is related to operations research, algorithm theory, and computational complexity theory.

The combinatorial algorithm for multiple alignment which presented by [33], proceeds in four phases consisting of the following combinatorial problems:

1. Constructing a graph of approximate overlaps between pairs of fragments.
2. Assigning an orientation to the fragments, i.e., choosing the forward or reverse complement sequence for each fragment.
3. Selecting a set of overlaps that induce a consistent layout of the oriented fragments.
4. Merging the selected overlaps into a multiple sequence alignment, and voting on a consensus.

In phase 1, overlaps are computed within the error rate that maximize a likelihood function on alignments. Edges in an overlap graph correspond to these alignments, and are weighted by likelihood. Given fragments of total length N and error rate ε our method for computing the graph modeling these overlaps takes $O(\varepsilon\, N^{\,2})$ time.

In phase 2, the fragments are oriented to maximize the weight of all edges in the overlap graph that are consistent with the chosen orientation. This sub-problem is NP-complete. An exact algorithm is presented that computes an optimal orientation in *O(K(EV))* time for an overlap graph of V fragments and E edges, where $K < 2^{v}$ is the size of its branch-and-bound search tree. Also an approximation algorithm is presented that computes an orientation of weight at least one-half the maximum in $O(E + V \log V)$ time.

In phase 3, the fragments are placed in an overlapping layout by selecting a set of edges of maximum total weight that form a branching which satisfy a dovetail-chain property. Finding such a branching is also NP-complete and in this method, an exact algorithm is presented that computes an optimal layout by finding a maximum weight dovetail-chain branching in $O(K(E + V \log V))$ time, which $K < 2^{E}$ is the size of its search tree. A greedy approximation algorithm for this problem is well known, and in contrast finds a branching of weight at least one-third the maximum in $O(E \log V)$ time. We further show that our approach naturally lends itself to producing alternate layouts, if desired.

In phase 4, the set of all overlaps in the graph is taken that agree with the fragment layout, and merge them into a multiple sequence alignment as follows.

The alignments represented by the set of overlaps match pairs of characters from the fragments. Of these character pairs, we seek a subset of maximum total weight that forms a multiple alignment. This problem is also NP-complete, though it can be solved in time exponential in the maximum number of fragments that mutually overlap in the layout. Given overlaps that match M pairs of characters from a layout with at most D mutually overlapping fragments, we construct a multiple sequence alignment of length L, and a consensus sequence, in $O(D^2L + M + N)$ time. The set of matched pairs that forms the alignment has weight at least $2/D$ of the maximum. Since the graph theoretic formulation has a favorable combinatorial structure, it can be solved by methods from combinatorial optimization [34] such as integer linear programming (ILP). In Section 3, we will introduce the alignment graph and extended alignment graph which are great varieties of combinatorial optimization methods that can be applied to solve ILPs. One such approach that can be applied to the above graph theoretic model is branch-and-cut [35].

2.3 Progressive alignment

An important development in multiple sequence alignment has been the introduction of the progressive alignment approach [36]. Progressive is a heuristics approach where complex multiple alignment problem is separated into sub-problems which solves direct multiple genome alignment problem indirectly with pairwise genome alignment. This approach assembles all sequences progressively where best pairwise alignment is first taken into account. In 1984, for the first time this algorithm formulated by Hogeweg and Hesper [37]. The basic idea behind this approach is the construction of an approximate phylogenetic tree as a guide tree [36] to solve multiple alignment problem where each leaf represents a sequence to be aligned. Each visited internal node is associated with an MSA of the sequences in its corresponding subtree. Finally, multiple alignment of all considered sequences is associated with the root node. Progressive alignment method is used in several alignment programs such as MULTAL [38-39], MAP [40], PCMA [41], MULTALIGN [42], CLUSTAL [43-44], T-Coffee [45], KAlign [46], MUMMALS/PROMALS [47-48], and others. Among them, the most widely used method is ClustalW [44]. It first performs the global pairwise alignment [19] of the sequences and develops a distance matrix. It then builds a guide tree based on the matrix values. Finally, it generates a consensus alignment by gradually adding sequences following the guide tree where the closest sequence pairs (smallest branch length in guide tree) are aligned

first and thus, it gradually adds the next sequences. However, the greedy nature of these approaches cannot allow to modify the gaps and hence, the alignment cannot be altered in the later stage. There is a possibility that they can be trapped in local minima for this greediness [40-50].

2.4 Anchor–based algorithms

Anchor-based alignment or synonymously seeded alignment, has three steps: (1) the computation of small segment matches of high similarity shared by multiple sequences, (2) the ordering of these segment matches into a collinear chain of non-overlapping segment matches and (3) closure of gaps in-between the anchors. The main purpose of step 1 and step 2 is to abandon a large chunk of the possible alignment space. Only small indels are allowed within the anchors. Hence, the time-consuming dynamic programming is only required between the fixed anchors. Some programs also try to extend anchors first to the left and right to further reduce the search space. The initial segment matches can be, for example, maximal unique or exact matches [51], maximal multiple exact matches [52] and exact or hashed k-mers [53]. Segment matches are optionally extended and finally, the quality of a segment match is assessed using some weight function. Chaining algorithms [54] can be applied to compute the heaviest collinear chain of these segment matches. The resulting list of anchors is refined by applying the above procedure iteratively by using a smaller k-mer or by filling the gaps in-between the anchors using more sensitive approaches such as pairwise dynamic programming. Since genomic rearrangements such as transposition, duplication or inversion are rather likely, novel methods try to cover at least some of these operations, for example, by computing only local chains [55].

3 Alignment graph data structures

Data structures are one of the most important concepts in computer programming. A data structure is a way of storing and managing data. There are two types of data structures as linear and nonlinear data structures. The difference between the linear data structure and the non-linear data structure is on the basis of the relationship between elements of data. Linear data structures allow traversing through the items sequentially. On the other hand, in a nonlinear data structure, each element is attached to one or more elements creating a relationship among the items. In linear data structure data is to arrange no specific order and data is arranged adjacently whereas in non-linear data structure data is arranged in a specific order and there is a relation between data. The main difference between linear and non-linear data structures is

that linear data structures arrange data in a sequential manner while nonlinear data structures arrange data in a hierarchical manner, creating a relationship among the data elements. Linear data structure forms a linear list where is a specific order in which elements are attached to each other in the data structure. Elements in linear data structure consume linear memory space and data elements are store in sequential manner. Also, in linear data structures memory of the data elements should be define at the start of the code. Array, stack, queue, linked list are examples of linear data structure. Non-linear data structure arrange data in a sorted order, where is a hierarchical relationship in this data structure. There are roots, child, and nodes in non-linear data structure, and there are levels that are not available in linear data structure.

A graph is a non-linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph. There are many types of graphs such as directed, non-directed, connected, non-connected, simple and multi-graph. Also the edges in graph can be weighted. Adjacent vertices, path, cycle, degree, connected graph, weighted graph are some important terms in the graph.

When we talk about the application of graphs, there are some well-known examples of graph data structure such as computer networks, transportation system, social network graphs, electrical circuits and project planning. In this section, we review the structures of commonly used graphs to investigate their abilities to represent multiple genome alignment information.

3.1 Alignment graph

Each multiple alignment can be represented as an alignment graph of sequence segments as shown in Figure 1. Let $G = (V, E)$ be an alignment graph data structure [40], for a set $S = \{S_1, S_2, \ldots, S_n\}$ of n sequences, $G = (V, E)$ is an n-partite graph for n sequences where the vertices represent non overlapping sequence segments, then $V = V_1 \cup V_2 \cup \ldots \cup V$n and each vertex $v_{ij} \in Vi$ represents a sequence segment in Si of arbitrary length which covers all positions of the segment. Every position in $S_i = S_{i1}\, S_{i2} \ldots S_{i|Si|}$ is covered by one and only one vertex $v_{ij} \in V_i$, where $1 < i < n$, $1 < j < |S_i|$. An edge $e = (v_{ip}, v_{jq}) \in E$ with $i \neq j$ indicates that vertex v_{ip} can be aligned with vertex v_{jq}. In other words, the sequence substring in S_i covered by v_{ip} can be aligned without gaps to the substring in S_j covered by v_{jq}. This graph structure can be a weighted graph by edge weights that capture some kind of quantitative measure of alignment quality. The score of aligning v_{ip} with v_{jq} is given by edge-weight of e. A possible measure is the pairwise BLOSUM score for each two aligned segments.

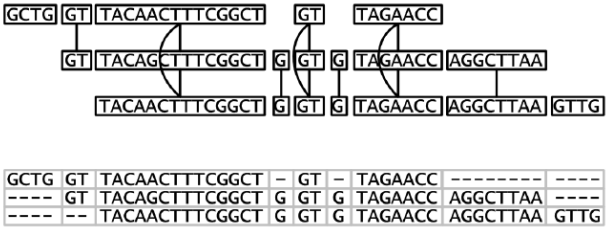

Figure 1. An alignment graph of 3 short sequences.

An edge in E is called an alignment edge, the alignment edges represent ungapped matches of sequence segments and also gaps are implicitly represented by the topology of the graph, it means a vertex without any outgoing edge is aligned to gaps in all other sequences [56]. For instance, GCTG vertex in Figure1, has no outgoing edges, so it is aligned to gaps in all other sequences. The alignment graph is a very compact and versatile description of an alignment. Large-scale alignments can be efficiently stored since long segments are represented by only a single vertex. Furthermore, the extension and direction of an alignment is completely defined by the alignment edges. That is, the graph formulation is equally suitable to align globally related sequences or thousands of reads where only subsets are related by mutual overlaps.

An alignment edge is realized by an alignment if the endpoints of the edge are placed in the same column of the alignment array. The subset of E which realized by an alignment is called the trace [57] of this alignment. Our goal is to select the subset of edges of maximum cardinality. A more practical approach is to augment the alignment graph by edge weights that capture some kind of quantitative measure of alignment quality. The objective is then to find the maximum weight trace, that is, the subset of edges with maximum weight [56].

Given sequences S and a corresponding alignment graph $G = (V, E)$ with edge weights. The maximum weight trace problem is to find a trace $T \subseteq E$ of maximum weight.

The notion of a trace of two strings is a basic concept in sequence comparison which Kececioglu [56] generalized to multiple sequence alignment with the notion of a trace of an alignment graph. The relationship between multiple alignment and multi-partite graphs was also examined by Vingron and Pevzner [58] in the context of filtering pairwise dot-plots of a set of sequences.

Maximum Trace Problem (MT) is introduced to model the final multiple alignment phase of DNA sequence assembly. In MT, every edge in the alignment graph has a positive weight representing the benefit of aligning the endpoints of the edge and the goal is to compute an

alignment whose trace has maximum weight. Kececioglu [56] showed that MT is NP-complete and developed a branch-and-bound algorithm for the problem based on dynamic programming, with worst-case time complexity $O(k^3 2^k N)$ and space complexity $O(kN)$, where $N = \prod_i |S_i|$, which is able to solve to optimality relatively small problem instances. The maximum trace problem can be generalized to accommodate different scoring schemes.

Theorem 3.1 [56] Maximum weight trace is NP-complete.

The Generalized Maximum Trace Problem (GMT) [40] is allowed multiple edges between two vertices in the alignment graph G and we can partition the edge set E into a set D of so called blocks.

3.2 Extended alignment graph

To analyze non-colinear changes among the input genomes, it is convenient to extend the alignment graph $\hat{G}$ to a mixed graph G by adding a set of directed edges (arcs). This model is defined with both directed and undirected edges, so G is a mixed graph and the set of edges decomposes into a set of directed adjacency edges and a set of undirected edges. Because of these edges, extended alignment graph is not n-partite as in its original definition.

A mixed graph G is a tuple $(V; E \cup A)$, where V is a set of vertices, E is a set of edges and A is a set of arcs. A path in a mixed graph is an alternating sequence of vertices and arcs or edges. A path is called a mixed path if it contains at least one arc in A and one edge in E. A mixed path is called a mixed cycle if and only if the first and the last vertex on the path is the same. Since a mixed path P (or a mixed cycle C) is determined by the set of arcs and edges in P (respectively in C), we often identify paths and cycles by their set of edges and arcs. The length of a mixed path P (cycle C) is the number of edges and arcs it contains. The size of a mixed path P (cycle C) is the number of edges in E it contains.

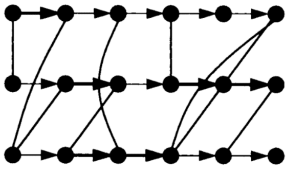

Figure 2. An extended alignment graph.

Let $\hat{G} = (V; E)$ be an alignment graph and $G = (V; E \cup A)$ be the extended alignment graph according to $\hat{G}$ by adding a set of arcs which are horizontal directed edges between each vertex in each part of n-partite alignment graph.

We call a mixed cycle R in G critical if all vertices related to each sequence which is in R, occur consecutively in R. The extended alignment graph gives us a simple way of testing whether its edge set represents a trace or not by simply looking for a critical mixed cycle in it which is proved in the following theorem.

Theorem 3.2 [26] Let $G = (V; E \cup A)$ be an extended alignment graph, let $T \subseteq$ E and $G_x = (V; T \cup A)$ be the extended alignment graph induced by T. Then T is a trace if and only if there is no critical mixed cycle in G_x.

To find a characterization of the Generalized Maximum Trace Problem (GMT) according to the extended alignment graph structure, if we are given a graph and a partition D of blocks, using Theorem 3.2, we can formulate the GMT problem for extended alignment graph $G = (V; E; H)$ as follows:

Given an extended alignment graph $G = (V; E; H)$ and a partition D into blocks with weights w_d $(\forall d \in D)$. Find a set $M \subseteq D$ of maximum weight such that $\bigcup_{d \in M} d$ does not induce a critical mixed cycle on G.

3.3 De-Bruijn graph

The De-Bruijn graph is a data structure, which first brought to bioinformatics as a method to assemble genomes from the experimental data generated by sequencing by hybridization [59]. It later became the base of algorithm in genome assembly that resulted in dozens of software tools. In addition, the De-Bruijn graphs have been used for repeat classification, de novo protein sequencing [60], synteny block construction, multiple sequence alignment, and other applications in genomics and proteomics [61]. The classical De-Bruijn graph is defined on a single genome string. Given a string $S = s_1 s_2 \ldots s_n$, the De-Bruijn graph $DB(s_1 s_2 \ldots s_n, k)$ is defined as follows:

Each vertex in the graph is labeled by a $(k-1)$-mer. The De-Bruijn graph DB(S, k) results from gluing identically labeled vertices in the graph whose $(n - k + 1)$ edges are labeled by k-mers $s_1 s_2 \ldots s_k$, $s_2 s_3 \ldots s_{k+1}$, … , and $s_{n-k+1} s_{n-k+2} \ldots s_n$. Implicitly, [28]. An example of a De-Bruijn graph for a single sequence is shown in Figure 3.

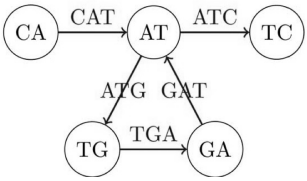

Figure 3. A De-Bruijn graph.

Given genomes GA and GB (represented as sets of strings), we classify their k-mers into 3 classes: A (occur only in GA), B (occur only in GB) and AB (occur in both GA and GB).

De-Bruijn graphs for representing an alignment are constructed by subdividing each sequence into overlapping k-mers, where a k-mer is a contiguous substring of length k. Two adjacent k-mers overlap by $k - 1$ letters. Vertices are connected by an edge if there is at least one pair of adjacent k-mers present in one of the sequences. In other words, the $(k + 1)$-mer defined by two adjacent vertices must be present in one of the sequences. The edge weight usually indicates how often this $(k + 1)$-mer was observed.

In a De-Bruijn graph of multiple sequences, each original input sequence is mapped to a path traversing the graph. Conserved subsequences are highlighted by heavy weight edges, that is, edges that are part of almost all sequence paths.

3.4 A-Bruijn graphs

Let S be a genomic sequence of length n and $A = (a_{ij})$ be a binary $n \times n$ similarity matrix representing the set of all significant local pairwise alignments between regions from S. The matrix A is defined as $a_{ij} = 1$ if and only if the positions i and j are aligned in at least one of the pairwise alignments and $a_{ij} = 0$ otherwise.

Matrix A represents an adjacency matrix of a graph which called the A-graph on n vertices$1, \dots, n$, where vertices i and j are connected if and only if $a_{ij} = 1$. Let V be the set of connected components of this graph and let $vi \in V$ be the connected component containing vertex$i$ $(1 \leq i \leq n)$.

The A-Bruijn graph $G(V, E)$ is defined as the multigraph on the vertex set V with $(n - 1)$ directed edges (v_i, v_{i+1}) for $1 \leq i \leq n$. an A-graph and related A-Bruijn graph are shown in Figure 4. The A-Bruijn graph can be considered as the Eulerian path obtained from the path $(1, \dots, n)$ after contracting each connected component into a single vertex [62].When the matrix A corresponds to all starting positions of perfect alignments of length l within a genome, the

A-Bruijn graph corresponds to the classical De-Bruijn graph with minor technical modifications. However, the A-Bruijn graph is well defined for any collection of alignments. The A-Bruijn graphs can be viewed as weighted graphs with the weight of an edge between two vertices equal to the number of edges connecting these vertices. A cycle in a graph is called short if it has less than girth edges, where girth is a parameter. There are two types of short cycles in the A-Bruijn graphs: whirls and bulges. Whirls are short, oriented cycles, where all edges are oriented the same way. Bulges are short cycles that contain both forward and reverse edges. A gap of length δ in a pairwise alignment typically creates a bulge of length $\delta + 2$ in the A-Bruijn graph. Whirls are caused by inconsistencies in pairwise alignments. Bulges and whirls may further aggregate into networks of bulges/ whirls that complicate the analysis of the A-Bruijn graph (as compared with the De-Bruijn graph) and hide the underlying repeat structure. The De-Bruijn graphs often can be simplified by collapsing every simple path (a maximal directed path in the graph satisfying the condition that all its internal vertices have one incoming and one outgoing edge) into a single edge. Such collapsing does not help much in the case of A-Bruijn graphs with numerous bulges and whirls. To produce a sensible repeat classification, one has to remove whirls and bulges. Such removal may sacrifice the fine details of some repeats in favor of revealing the mosaic structure shared by different repeat copies [62].

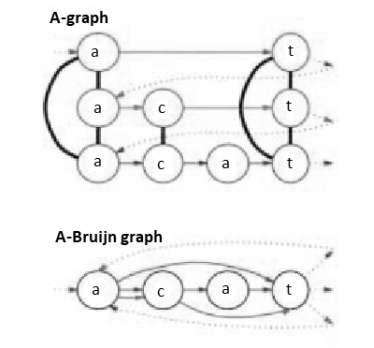

Figure 4. A-graph and A-Bruijn graph.

To represent a multiple genome alignment as an A-Bruijn graph, given t sequences $S_1, \ldots, S_t$ of total length n and $t(t-1)/2$ pairwise alignments between these sequences, one can concatenate $S1, \ldots, St$ into a single sequence S of length n and compose the $n \times n$ similarity matrix A from $t(t-1)/2$ pairwise alignments. The only difference between the A-Bruijn

graph of multiple sequences $S_1, \ldots, S_t$ and the A-Bruijn graph of a single sequence S is that edge (v_i, v_{i+1}) is removed from the A-Bruijn graph of multiple sequences if positions i and $(i + 1)$ in S correspond to the last and the first positions in the consecutive sequences S_k and S_{k+1}. Therefore, the A-Bruijn graph may have up to t sources and t sinks.

3.5 Partial order graphs

An algorithm for aligning some multiple sequences encoded as a partial order alignment graph was given in [47]. The partial order alignment-based algorithm is an extension of the traditional dynamic programming algorithms for sequences [31] to handle partial orders. Partial order alignment (POA) graphs are directed acyclic graphs (DAGs) where the sequences are encoded as paths. Under such a formulation, gapped alignment of a m length sequence to a POA graph on E edges takes $O(mE)$ time [63].

The POA graphs share resemblance to genome variation graphs in the sense that the variations are encoded as alternative paths using additional vertices and edges. In [64], POA-based technique was used for aligning sequences to genome variation graphs. POA-based techniques are restricted to acyclic graphs whereas genome variation graphs can have cycles. To handle cycles, they first construct acyclic extensions of the input graphs through expensive loop unrolling (DAGification) steps and the alignment is then performed on the acyclic extensions. A k-length DAGification of a graph G aims to compute a DAG G' such that all paths (not necessarily simple) of length k or less in G are present in G' and vice versa. For aligning an m length sequence, the value of k has to be m or more. Acyclic graph extensions can have considerable blowup in their size. The edge and vertex blow-up factor in the worst case is proportional to the input sequence length. Prohibitively large size of the DAGified graph results in increased preprocessing and alignment time and thereby affects the overall alignment performance [63]. For large reference graphs, the sequence alignment follows a seed and extend strategy where candidate subgraphs of the reference graph with potentially large alignment scores are first identified [64]. The final alignment is then performed on these filtered subgraphs. In this case, the DAGification is restricted to these candidate subgraphs.

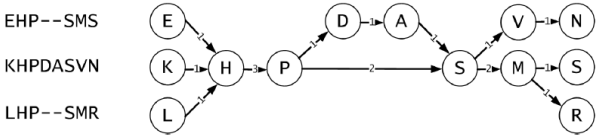

Figure 5. A partial order alignment graph.

An individual sequence can be represented by a trivial partial order graph. Each character is converted to a vertex and all vertices have exactly one outgoing edge to the vertex for the subsequent character, except for the vertex of the last sequence character. In a MSA these single-sequence graphs are merged. Similar to the De-Bruijn graph, each sequence is mapped to a path traversing the graph but since it is a partial order graph no cycles are allowed. That is, the partial order graph is a directed acyclic graph. Consequently, the partial order graph enforces the collinearity condition and does not allow two crossing aligned regions where the order of characters in the sequences is not preserved. The key characteristic of the partial order graph is that matching sequence characters are merged to a single node whereas mismatches cause the graph to bifurcate. This is shown in Figure 5.

3.6 Enredo graphs

Let G = (V, E) be an Enredo graph structure model. This model takes two inputs, the first input is a set of complete genomes which comprised of a set of chromosomes. Each chromosome is a non-zero length linear or circular string of double-stranded DNA. And the second input is some short segment-groups, which each group containing two or more homologous segments. A segment-group is defined as a set of directed segments where the global alignment of directed segments is significant. Segment-groups overlap if their segments share any common positions. Let the term genome point anchor (GPA) be synonymous for a short segment-group containing two or more homologous segments, each of which is between 50 and a few hundred positions in length. More precisely, the second input to Enredo is a set of non-overlapping GPAs. In this method, generally prefer GPAs to be short because this makes it easier to avoid overlap between the ends of GPAs, though GPAs must necessarily be long enough for us to be confident of the homology relationships they contain.

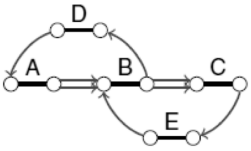

Figure 6. An Enredo graph structure.

In an Enredo graph [65], there is two sets of edges; a set of directed adjacency edges E_1 (between the GPAs) and a set of undirected edges E_2 (inside the GPA). Then the set of edges would be $E = E_1 \cup E_2$. Let $*A$ and $A*$ represent the two ends of a GPA A. For each A in the

set of non-overlapping GPAs an undirected edge connects the vertices representing $*A$ and $A*$. For each segment x in a GPA A, let $e(x,* A)$ denote the position in x closest in the order of the segment to the GPA end $*A$. Let $e'(* A) = \cup xe(x,* A)$. An adjacency-edge exists between the vertices representing two GPA ends a and b if and only if there exists a segment that defines a contiguous, maximal range between a member of $e'(a)$ and $e'(b)$ and this segment does not contain any position that is a member of a GPA in the no overlapping GPA set, also this segment be <200 kb in length.

3.7 Cactus graph

In graph theory, a cactus graph is a connected graph which any two simple cycles have at most one vertex in common. To represent the common structure between substrings of homologous bases in a cactus structure model, we consider two types of aforementioned homology structure: blocks and ends. A block is formally a maximal set of maximal-length homologous oriented strings. Each maximal length oriented string in a block is a segment. The blocks are boxes containing gapless two-dimensional alignments and the ends are maximal sets of homologous caps. Paten et al.,[66] have defined two types of end for this graphs, block ends, which are the ends of blocks and stub ends, which include the previously mentioned telomeres, and which will also include block ends from higher level problems which we introduce multi-level cactus graphs in the next section.

The adjacency graph in Figure7.A is a graph with a node for every end and an adjacency edge between two ends if there is an adjacency, potentially containing a nonempty substring from the input sequence, between a cap in one of the ends and a cap in the other end, i.e., if the caps would abut except for a possibly non-empty intervening adjacency substring in the input chromosome in which they appear. Self-edges are allowed in the adjacency graph and occur when two homologous caps in opposite orientation share an adjacency. Multi edges are not included in the adjacency graph; i.e., there is at most one adjacency edge between any two nodes, even if there are several adjacencies between them. In this case, the adjacency edge is labeled with the set of adjacencies and their substrings, which uniquely pair caps between the ends they link. Unlike blocks, the substrings within the adjacencies of an adjacency edge are not assumed to be homologous and are therefore not aligned. The two ends of each block are also connected by an edge in the adjacency graph; these edges are called block edges and are labeled with the oriented set of aligned segments of the block they represent. In addition to the block edges, the adjacency graph also includes end edges; for each stub end, the adjacency graph includes one end edge that connects the node representing the stub end to a special dead-

end node. All dead end nodes are in turn connected in a clique by unlabeled backdoor adjacency edges [67].

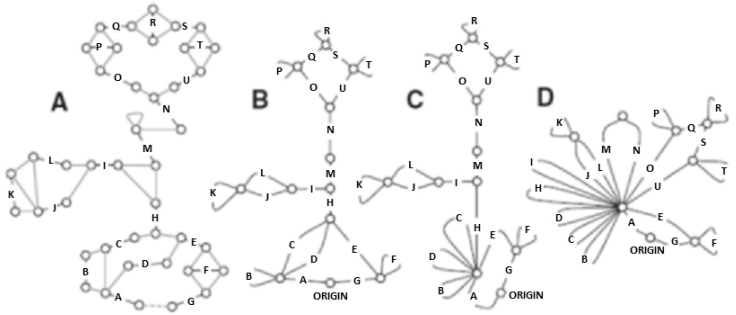

Figure 7. Construction of the final cactus graph from an initial adjacency graph G_0 (A), the same graph after the collapse of the adjacency components G_1 (B) after the collapse of the three edge connected components G_2(C) after modifications to bridge edge components to make the graph Eulerian G_3(D).

Let G_0 be the adjacency graph. The cactus graph is built from G_0 in a series of steps in thioe following. Also is shown in Figure 7.

(1) Ignoring the block and end edges, we compute the connected components of G_0 formed by the adjacency edges only. These are called adjacency-connected components. All dead ends will be in a single component, which we call the origin component. The graph G_1 represents this decomposition of G_0 into the resulting adjacency-connected components (Fig. 7B). There is a node in G_1 for every adjacency-connected component in G_0. The graph G_1 has only block and end edges, no adjacency edges. Two nodes X and Y in G_1 representing adjacency-connected components in G_0 which are connected by an edge in G_1 for every block or end edge in G_0 from some $x \in X$ to some $y \in Y$. Thus, the graph G_1 is formed by merging adjacency-connected nodes in G_0 and retaining only the block and end edges in the merged graph. We call the node in G_1 and subsequent graphs containing the origin component of G_0 the origin node.

(2) Computing the decomposition of G1 into three-edge connected components using the linear time algorithm in [68]. To define this decomposition, consider that two nodes x and y in G_1 are equivalent if there is no set of up to two edges in G_1 which, upon removal, disconnect G_1 in such a way that there is no path from x to y. Thus, two nodes are equivalent if it takes the removal of three or more edges to disconnect them. The equivalence classes of nodes are called Three-edge connected components. The graph G_2 represents this decomposition (Fig. 5C). It has one node for each three edge connected component. Two nodes X and Y in G_2 are connected

by an edge for every edge in G_1 between some node $x \in X$ and some node $y \in Y$. Thus, the graph G_2 is formed by merging equivalent nodes in G_1. The theory of graph decomposition into three-edge connected components shows that G_2 is in fact a cactus graph in the combinatorial sense. However, it is not yet the cactus graph.

(3) Finally, to construct the cactus graph, the tree-like structures in G_2 are folded in to obtain an Eulerian cactus graph $G = G_3$ (Fig. 5D). Formally, an edge in G_2, or indeed in any graph, is called a bridge if its removal disconnects the connected component in which it is contained. Consider the subgraph formed by only the bridge edges. It is easy to see that this subgraph is a forest, i.e., a collection of disjoint trees. In the fold-in process, for each such tree, we merge all leaf nodes and branching nodes into a single tree loop node. Only the non-branching internal nodes in the tree are left out of this merge, and appear on simple cycles emanating from the tree loop node, along with other cycles that were already present before this merge step. It is easy to see that the resulting graph G is also a cactus graph with one origin node. In fact, every node is either in a unique simple cycle or is the unique intersection of two or more simple cycles. Thus, all the nodes in G have an even number of edges incident upon them, i.e., are of even degree [66].

3.8 Multilevel cactus graph

In this section, an extension to the basic cactus graph is described which formed by nesting cactus graphs within cactus graphs. Paten et al., [67] called this graph structure, a multi-level cactus graph and described how it can represent progressively more detailed levels of alignment. This approach, allows us to define sparse cactus graphs in which only a portion of the genomes are aligned; for example, one might initially define a high level sparse cactus graph in which the blocks were composed of homologous sets of exons. All bases outside the exons are contained in the adjacencies. Each node in the cactus graph is a net that is built from some set of these adjacencies. Now, suppose we extend our notion of homology by aligning some of the bases that occur inside the adjacencies in the nets. It is easier to define a high-level cactus graph using the segments and adjacencies at the lower level.

Essentially, an adjacency at the higher level is a thread at the lower level. Formally, let us say that two threads are similar if they have homologous caps in the at least one end. A group is a minimal set of disjoint threads that is closed under similarity, i.e., a pairwise non-overlapping set of threads such that there are no threads that are similar to any of those in the set that are not already in the set, and there is no proper subset of these threads that has this closure property. A group is self-contained if there is no homology between any segment in a thread in

the group with any segment outside of the threads in the group. Each net in a high-level cactus graph is a union of self-contained groups from the lower level segments and adjacencies. There are two kinds of groups. A link is a group in which all the caps are part of two homology classes and a tangle is a group in which the caps form more than two homology classes. We call a net whose adjacencies contain non-empty substrings of S non-terminal and conversely a net whose adjacencies contain only empty substrings terminal [68]. In a multi-level cactus graph, for each self-contained group in a net, termed a net contained group (either link or tangle), a child cactus graph is constructed in which: (1) Threads connecting caps from the group's ends are treated as linear chromosomes. (2) The group's ends become stub ends in the child cactus graph, and thus map the boundaries between the parent net and the child cactus graph. (3) Homologies between segments within the threads form the blocks. Then the adjacencies in the parent net are divided up into lower level segments and further adjacencies in the child cactus graph, repeating the process recursively that was used to construct the cactus graph containing the parent net.

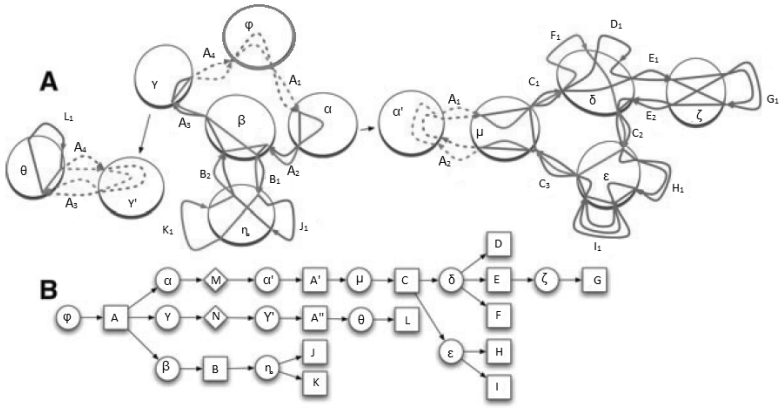

Figure 8. A multi-level cactus graph (A) and a multi-level cactus tree (B).

The recursion creating child cactus graphs can be continued until all non-terminal nets have defined child chains and child cactus graphs, and thus all bases in S become part of a block in one cactus graph of the set of cactus graphs that comprise a multi-level cactus graph (Fig. 8 A). Just as the parent-child organization of chains and nets in a cactus graph can be represented by a bi-layered cactus tree, a multilevel cactus graph's chains, nets, and net-contained groups can

be represented in a bi-layered multi-level cactus tree (Fig. 8 B). The chain layer of a cactus tree in the multi-level cactus tree contains both chain- and net-contained group nodes.

3.9 Outerplanar graph

Outerplanar graphs occur for the first time in the literature in Harary's classical book [69]. In graph theory, a graph is outerplanar if it can be embedded in the plane such that all vertices lie on the outerface boundary. A graph G is called biconnected if $|G| > 2$ and $G - \{u\}$ be connected for every vertex $u \in G$. The outerplanar graphs are a subset of the planar graphs and the circle graphs. The maximal outerplanar graphs, those to which no more edges can be added while preserving outerplanarity, are also chordal graphs and visibility graphs. There are some useful theorems related to outerplanar graph structure.

Theorem 3.3. [69] A graph G is outerplanar if and only if it contains no induced subgraph isomorphic to K_4 or $K_{2,3}$.

Theorem 3.4. [69] A biconnected outerplanar graph contains a unique Hamiltonian cycle.

Theorem 3.5. [69] Every maximal outerplanar graph of order at least 3 is biconnected.

In [70], a construction of outerplanar graph model structure for genomic data is described which introduce in the following.

Let S be the set of input whole genome sequences, we can assume that the input sequences be either linear or circular sequences. Mathematically, a sequence is just a string (likely circular) of symbols taken from an alphabet set.

The whole genome base pairs alphabet set is $\{A/T, T/A, C/G, G/C\}$ and using sings according to oriented reverse complement, we have: $A/T = -T/A$ and $C/G = -G/C$.

To represent the common structure between homologous segments in a set of whole genome sequences, we define the concept of "Alignment-set" as follows:

"Alignment-set" is a set of maximal homologous segments with maximal length and denoted by A-set. The size of "A-set" is the number of aligned segments.

We denote all of the A-sets of genome S by Σ_S which is the input for building a genome alignment graph. Two A-sets $\sigma_1, \sigma_2 \in \Sigma_S$ are adjacent if there exist two segments $s_1 \in \sigma_1$, and $s_2 \in \sigma_2$ which are adjacent. The adjacency is defined by the set of positions.

The biconnected outerplanar graph G_1 is built in fourth steps:

1. At first we construct a graph G which every A-set is a vertex of G and there is an edge between two A-sets if there adjacency between two segments of them. In fact, the graph G is an adjacency graph. As an example see Figure 9. Since one A-set may contain more

than one segment from the same genome, each A-set can be adjacent to itself and also there may be multiple edges between two vertices.

2. Using pDFS $Algorithm$ [71], we compute biconnected components of G. Since in [69] has shown that a graph is outerplanar if and only if every one of its biconnected components is outerplanar, we restrict the outerplanarity to biconnected subgraphs. In [72] a conceptually simple algorithm is presented to determine if a graph is a maximal outerplanar or outerplanar graph. The algorithm is linear in the number of vertices. It relies on the fact that a maximal outerplanar graph has a unique Hamiltonian cycle which forms the outer face, the remainder of the graph is a triangulation of this cycle. So we apply MOP-TEST $Algorithm$ [72] to recognize outerplanar and non-outerplanar subgraphs of . If all of the connected components of G are outerplanar graphs, then we do not need to third step and we can skip that. But if there is one or more non-outerplanar components, they contain some minors isomorphic to K_4 or $K_{2,3}$.
3. By merging two adjacency-connected vertices of K_4 minors and two non-adjacency vertices of $K_{2,3}$ minors, we form graph G to an outerplanar graph G_1
4. Finally, to make our graph biconnected, we need to make it bridgeless. In the second step, if there is any bridge as a biconnected component, we easily merge vertices of that bridge.

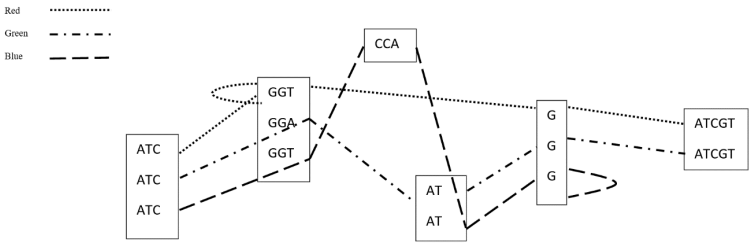

Figure 9. The graph G according to 3 sequences ATCGGTTGGGATCGT (Red), ATCAGGATGATCGT (Green) and ATCTGGCCATAGG (Blue).

To show a circular visualization of this model, we can use $Algorithm$ 1 in [73]. We close this section by another interesting property of this method which show that there is a unique permutation of vertices which give us a cross-free circular visualization of our outerplanar graph data structure model.

Theorem 3.6. [73] There exists only one clockwise ordering of the nodes in a biconnected outerplanar graph G such that the drawing of G with the nodes in that order around the embedding circle is plane.

3.10 Line outerplanar graph

The Line graph $L(G)$ of undirected graph G is another graph that represents the adjacencies between the edges of G. The Line graph is defined as follows:

The Line graph of G denoted by $L(G)$ is the intersection graph of the edges of G, representing each edge by the set of its two end vertices. In other words, $L(G)$ is a graph such that each vertex of $L(G)$ represents an edge of G andTwo vertices of $L(G)$ are adjacent if their corresponding edges share a common end point in G.

In this method [70], we start with graph G_1 which we have mentioned it is a biconnected outerplanar graph with a Hamiltonian cycle which passes every A-set just once. So we are going to construct a new graph G_2 which the Line graph of this graph, is isomorphism to G_1. This approach is inspired by Pevsner's approach [45] for fragment assembly using De-bruijn graphs. We need the following theorems to prove that G_2 is Eulerian.

Theorem 3.7. [74] Let G be a non-outerplanar graph, then $L(G)$ is also non-outerplanar.

Theorem 3.8. [75] Let H be the line graph of G, then there is an Eulerian path/circuit in G if and only if there is a Hamiltonian path/circuit in H.

Our new graph G_2 is built in the following steps:

1. Let C be the set of all adjacencies of segments. The vertices V of G_2 will be a pairwisely disjoint subset of C and the edges E of G_2 represent A-sets. It's like block edges in Enredo graph [6]. We consider each A-set, as an undirected edge $e = u, v$ which the endpoint $u \in V$ of e represents a subset of adjacencies in C that contains all adjacencies at one end of e, and the other endpoint $v \in V$ contains all adjacencies at the other end of e. It is possible that $u = v$. Easily one can see that each maximal component in G which includes vertices connected only by adjacency edges is considered as a vertex in this new graph by ignoring all the A-sets which were defined as edges in G_2.
2. According to the collapsed vertices of G_1, we do similar collapses for some edges in graph G_2 corresponded to those vertices.

In 2015, Liu [76] presented a new and efficient algorithm, "ILIGRA", for inverse line graph construction. Given a line graph H, ILIGRA constructs its root graph G with the time

complexity being linear in the number of nodes in H. using ILIGRA $Algorithm$ and considering G_1 as input, easily we can compute the graph G_2 which $L(G_2) = G_1$.

Theorem 3.9. [70] The graph G_2 is an Eulerian Outerplanar graph.

4 Discussion

In this section, we discuss about the usage of mentioned graph data structures to multiple genome alignment. Also we investigate and compare them to how each method is different from others. The investigations of genome rearrangements often employ graph data structures used for multiple genome alignment. Graphs can assist in improving genome comparison through multiple alignments and also analysis of rearrangements. In addition, graph data structures provide an intuitive representation of similarities and changes between genomes, and so visualize alignment structures. In comparison to tabular alignments, genome alignment graphs are more versatile insofar that it is possible to model collinear and non-collinear changes without the need of choosing a reference genome [77]. The earliest graph is the alignment graph which has been proposed by Kececioglu [56]. The alignment graph defined for collinear multiple alignment and this graph contains a vertex for each sequence character and edges for aligned characters. The alignment graph has been used in various versions which a collinear alignment can be obtained from the alignment graph by solving the maximum weight trace problem [33]. In alignment graph data structure, the graph contains a vertex for each sequence character and edges for aligned characters. The alignment graph has since been used in various versions, e. g., with additional sequence edges, with weighted edges and with genes or segments instead of single characters. The extended of alignment graph structure can model non-colinear changes among the input genomes, in particular translocations and duplications. Translocations appear in extended alignment graph as mixed cycles. Duplications appear as block edges within the set of vertices of one genome. Because of these edges an extended alignment graph is not an n-partite as in its original graph alignment.

The de Bruijn graph are a key data structures in the studies of genome rearrangements and genome assembly. Comparing graph alignment structure, the classical de Bruijn graphs are defined on a single genome (represented as DNA strings). The use of the de Bruijn graphs in computational molecular biology to fragment assembly, resequencing with DNA arrays, EST analysis and computational mass spectrometry. The de Bruijn graph represents every *k*-mer in a genomic sequence as a vertex and connects two vertices by a directed edge if they correspond to a pair of overlapping *k*-mers in the genome. The genomic sequence corresponds to an Eulerian path in the resulting multigraph. A-Bruijn graphs are introduced as a generalization

of de Bruijn graphs which often use for genome sequencing and fragment assembly. The structure of A-Bruijn graphs revisits an idea briefly mentioned by Kececioglu [33], the idea of merging aligned vertices. A-Bruijn graphs have one vertex for sets of aligned positions, and edges represent sequence adjacencies. A-Bruijn graph is defined for an arbitrary collection of alignments. The A-Bruijn graphs are equivalent to the de Bruijn graphs in the special case that is the collection of all perfect similarities of length k (k-mers). The applications of A-Bruijn graphs in bioinformatics extend well beyond repeat classification and include multiple alignments and fragment assembly. Repeat classification is a multifaceted problem that covers many biological tasks. The A-Bruijn graphs can be viewed as weighted graphs with the weight (multiplicity) of an edge between two vertices equal to the number of edges connecting these vertices. An individual sequence can be represented by a trivial partial order graph. Each character is converted to a vertex and all vertices have exactly one outgoing edge to the vertex for the subsequent character, except for the vertex of the last sequence character. In a multiple alignment these single-sequence graphs are merged. Similar to the De-Bruijn graph, each sequence is mapped to a path traversing the graph but since it is a partial order graph no cycles are allowed. That is, the partial order graph is a directed acyclic graph. Consequently, the partial order graph enforces the collinearity condition and does not allow two crossing aligned regions where the order of characters in the sequences is not preserved. The key characteristic of the partial order graph is that matching sequence characters are merged to a single node whereas mismatches cause the graph to bifurcate. In contrast to partial order graphs, a De Bruijn graph allows cycles and hence, it can be used to detect repeated, in other words, the De Bruijn graph does not enforce a collinear alignment where one alignment column precedes the next one. Enredo graphs which applied for collinear alignments of segments, have two vertices per set of aligned segments, a head and a tail vertex, resembling breakpoint graphs from rearrangement studies. The Enredo method iteratively eliminates various substructures from the Enredo graph before deriving a final genome segmentation. The construction of the Enredo graph has some similarities to the use of de Bruijn-like graphs in sequence analysis where consistent sequences between two nodes in extant species are collapsed on to the same edge. Clearly the Enredo graph and the de Bruijn graph are radically different in terms of elements. The nodes in the Enredo graph are sets of genomic anchor points, and the edges are the large intervening sequences, whereas the nodes in a de Bruijn are strings of length k (k-mers) and the edges represent the adjacencies between these k-mers in an observed sequence. A cactus graph model structure is introduced as a dissimilar graph which has vertices for adjacencies and edges for genome segments. Their structure has two valuable properties. The cactus property subdivides

the graph (and genomes) into independent units by ensuring that any edge is part of at most one simple cycle. The second property is the existence of an Eulerian circuit. This circuit traverses all genome segments exactly once, even duplicated segments, conveniently providing a consensus genome. For substantial regions, it is possible to construct large multi-level cactus graphs that are reasonably balanced and highly branching so that their median depth from root to leaves is short. The generality of the cactus graph representation makes them widely applicable to almost any genome comparison problem. An outerplanar graph data structure is presented for multiple genome alignment and analyzing genome rearrangement which comparing with traditional alignment matrix or partial order alignment graph, in common with A-Bruijn and Cactus graphs, this model is flexible by classification non-collinear structural changes like inversion, translocations and duplications as well as collinear changes like insertion and deletion. But in addition, outerplanar graph data structure provides a unique circular visualization to simplify the study of evolutionary relationships between aligned genomes. Also in the line graph of this structure, we can get a unique Eulerian path in the representation.

5 Conclusion

Multiple genome alignment is an indispensable tool for comparing genomes and finding their shared histories. In bioinformatics, a sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Genome sequence alignment problem try to uncover homologies by assigning sequence positions. This review provided an overview on the different approaches for multiple genome alignment methods with focused on application of graph data structures. Graphs have proven to be a powerful tool for coping with the complexity of genome-scale sequence alignments. Also graphs provide an intuitive representation of similarities and changes between genomes, and so visualize alignment structures. In addition by having a graph data structure for a multiple genome alignment, it is possible to model colinear and non-colinear changes without the need of choosing a reference genome. In this investigation, we first, briefly introduced four type of algorithms which are used for computational part of genome alignment and then we described ten data structures of commonly used graphs in terms of their abilities to represent alignment information.

Acknowledgment: This research was supported by Research Core: "Bio-Mathematics with computational approach" of Tarbiat Modares University, with grant number "IG-39706". Also, the first author was supported in part by Iran National Science Foundation (INSF) (Grant No. 96012405).

References

[1] J. D. Thompson, B. Linard, O. Lecompte, O. Poch, A comprehensive benchmark study of multiple sequence alignment methods: current challenges and future perspectives, *PloS One* **6** (2011) #e18093.

[2] S. Kumar, A. Filipski, Multiple sequence alignment: in pursuit of homologous DNA positions, *Genome Res*. **17** (2007) 127–135.

[3] S. B. Needleman, C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J. Mol. Biol*. **48** (1970) 443–453.

[4] T. F. Smith, M. S. Waterman, Identification of common molecular subsequences, *J. Mol. Biol*. **147** (1981) 195–197.

[5] J. Heringa, W. R. Taylor, Three–dimensional domain duplication, swapping and stealing, *Curr. Opinion Struct. Biol.* **7** (1997) 416–421.

[6] N. Lakshmi, P. Gavarraju, J. Jeevana, P. Karteeka, A literature survey on multiple sequence alignment algorithms, *Int. J. Adv. Res. Comput. Sci. Softw. Engin*. **6** (2016) 280–288.

[7] W. Chen, B. Liao, Y. Liu, W. Zhu, Z. Su, A numerical representation of DNA sequences and its applications, *MATCH Commun. Math. Comput. Chem*. **60** (2008) 291–300.

[8] B. Liao, C. Zeng, F. Q. Li, Y. Tang, Analysis of similarity/dissimilarity of DNA sequences based on dual nucleotides, *MATCH Commun. Math. Comput. Chem*. **56** (2006) 209–216.

[9] Z. Mu, G. Li, H. Wu, X. Qi, 3D-PAF curve: A novel graphical representation of protein sequences for similarity analysis, *MATCH Commun. Math. Comput. Chem.* **75** (2016) 447–462.

[10] D. Panas, P. Waz, D. Bielinska–Waz, A. Nandy, S. C. Basak, 2D–dynamic representation of DNA/RNA sequences as a characterization tool of the zika virus genome, *MATCH Commun. Math. Comput. Chem.* **77** (2017) 321–332.

[11] J. Pesek, A. Žerovnik, Numerical characterization of modified Hamori curve representation of DNA sequences, *MATCH Commun. Math. Comput. Chem*. **60** (2008) 301–312.

[12] Z. H. Qi, M. Z. Jin, An intuitive graphical method for visualizing protein sequences based on linear regression and physicochemical properties, *MATCH Commun. Math. Comput. Chem.* **75** (2016) 463–480.

[13] D. Sun, C. Xu, Y. Zhang: A novel method of 2D graphical representation for proteins and its application, *MATCH Commun. Math. Comput. Chem*. **75** (2016) 431–446.

[14] S. Wang, J. Yuan, DNA computing of directed line–graphs, *MATCH Commun. Math. Comput. Chem*. **56** (2006) 479–484.

[15] R. Wu, Q. Hu, R. Li, G. Yue, A novel composition coding method of DNA sequence and its application, *MATCH Commun. Math. Comput. Chem.* **67** (2012) 269–276.

[16] R. Wu, R. Li, B. Liao, G. Yue, A novel method for visualizing and analyzing DNA sequences, *MATCH Commun. Math. Comput. Chem.* **63** (2010) 679–690.

[17] J. F. Yu, J. H. Wang, X. Sun, Analysis of similarities/dissimilarities of DNA sequences based on a novel graphical representation, *MATCH Commun. Math. Comput. Chem.* **63** (2010) 493–512.

[18] Q. Zhang, B. Wang, On the bounds of DNA coding with H-distance, *MATCH Commun. Math. Comput. Chem.* **66** (2011) 371–380.

[19] Y. Zhang, W. Chen, A new measure for similarity searching in DNA sequences, *MATCH Commun. Math. Comput. Chem.* **65** (2011) 477–488.

[20] Y. Zhang, W. Chen, New invariant of DNA sequences, *MATCH Commun. Math. Comput. Chem.* **58** (2007) 197–208.

[21] X. Zhou, K. Li, M. Goodman, A. Sallam, A novel approach for the classical Ramsey number problem on DNA-based supercomputing, *MATCH Commun. Math. Comput. Chem.* **66** (2011) 347–370.

[22] N. Jafarzadeh, A. Iranmanesh, A new graph theoretical method for analyzing DNA sequences based on genetic codes, *MATCH Commun. Math. Comput. Chem*. **75** (2016) 731–742.

[23] S. K. Gupta, J. D. Kececioglu, A. A. Schäffer, Improving the practical space and time efficiency of the shortest–paths approach to sum-of-pairs multiple sequence alignment, *J. Comput. Biol*. **2** (1995) 459–472.

[24] M. Lermen, K. Reinert, The practical use of the A* algorithm for exact multiple sequence alignment, *J. Comput. Biol.* **7** (2000) 655–671.

[25] D. J. Lipman, S. F. Altschul, J. D. Kececioglu, A tool for multiple sequence alignment, *Proceed. National Acad. Sci*. **86** (1989) 4412–4415.

[26] H. P. Lenhof, B. Morgenstern, K. Reinert, An exact solution for the segment-to-segment multiple sequence alignment problem, *Bioinf.* **15** (1999) 203–210.

[27] E. Althaus, S. Canzar, LASA: A tool for non-heuristic alignment of multiple sequences, in: M. Elloumi, J. Küng, M. Linial, R. F. Murphy, K. Schneider, C. Toma (Eds.), *Bioinformatics Research and Development*, Springer, Berlin, 2008, pp. 489–498.

[28] E. Althaus, A. Caprara, H. P. Lenhof, K. Reinert, Multiple sequence alignment with arbitrary gap costs: Computing an optimal solution using polyhedral combinatorics, *Bioinf.* **18** (2002) S4–S16.

[29] E. Althaus, A. Caprara, H. P. Lenhof, K. Reinert, A branch–and–cut algorithm for multiple sequence alignment, *Math. Program*. **105** (2006) 387–425.

[30] I. M. Wallace, G. Blackshields, D. G. Higgins, Multiple sequence alignments, *Curr. Opin. Struct. Biol*. **15** (2005) 261–266.

[31] K. Katoh, H. Toh, Recent developments in the MAFFT multiple sequence alignment program, *Briefings Bioinf*. **9** (2008) 286–298.

[32] R. Bellman, *Dynamic Programming*, Princeton Univ. Press, Princeton, 1957.

[33] J. D. Kececioglu, E. W. Myers, Combinatorial algorithms for DNA sequence assembly, *Algorithmica* **13** (1995) 7–18.

[34] J. D. Kececioglu, H. P. Lenhof, K. Mehlhorn, P. Mutzel, K. Reinert, M. Vingron, A polyhedral approach to sequence alignment problems, *Discr. Appl. Math.* **104** (2000) 143–186.

[35] E. Althaus, G. W. Klau, O. Kohlbacher, H. P. Lenhof, K. Reinert, Integer linear programming in computational biology, in: S. Albers, H. Alt, S. Näher (Eds.), *Efficient Algorithms*, Springer, Berlin, 2009, pp. 199–218.

[36] D. F. Feng, R. F. Doolittle, Progressive sequence alignment as a prerequisitetto correct phylogenetic trees, *J. Mol. Evol*. **25** (1987) 351–360.

[37] P. Hogeweg, B. Hesper, The alignment of sets of sequences and the construction of phyletic trees: an integrated method, *J. Mol. Evol*. **20** (1984) 175–186.

[38] W. R. Taylor, Multiple sequence alignment by a pairwise algorithm, *Bioinf*. **3** (1987) 81–87.

[39] W. R. Taylor, A flexible method to align large numbers of biological sequences, *J. Mol. Evol*. **28** (1988) 161–169.

[40] X. Huang, On global sequence alignment, *Bioinf*. **10** (1994) 227–235.

[41] J. Pei, R. Sadreyev, N. V. Grishin, PCMA: fast and accurate multiple sequence alignment based on profile consistency, *Bioinf.* **19** (2003) 427–428.

[42] F. Corpet, Multiple sequence alignment with hierarchical clustering, *Nucleic Acids Res*. **16** (1988) 10881–10890.

[43] D. G. Higgins, P. M. Sharp, CLUSTAL: a package for performing multiple sequence alignment on a microcomputer, *Gene* **73** (1988) 237–244.

[44] J. D. Thompson, D. G. Higgins, T. J. Gibson, CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice, *Nucleic Acids Res.* **22** (1994) 4673–4680.

[45] C. Notredame, D. G. Higgins, J. Heringa, T-Coffee: A novel method for fast and accurate multiple sequence alignment, *J. Mol. Biol*. **302** (2000) 205–217.

[46] T. Lassmann, E. L. Sonnhammer, Kalign–an accurate and fast multiple sequence alignment algorithm, *BMC Bioinf*. **6** (2005) #298.

[47] J. Pei, N. V. Grishin, MUMMALS: multiple sequence alignment improved by using hidden Markov models with local structural information, *Nucleic Acids Res*. **34** (2006) 4364–4374.

[48] J. Pei, N. V. Grishin, Erratum: MUMMALS: Multiple sequence alignment improved by using hidden Markov models with local structural information, *Nucleic Acids Res*. **34** (2006) 6064–6064.

[49] D. F. Feng, R. F. Doolittle, Progressive alignment of amino acid sequences and construction of phylogenetic trees from them, *Methods Enzym*. **266** (1996) 368–382.

[50] J. D. Thompson, P. Koehl, R. Ripp, O. Poch, BAliBASE 3.0: Latest developments of the multiple sequence alignment benchmark, *Proteins Struct Function Bioinf.* **61** (2005) 127–136.

[51] S. Kurtz, A. Phillippy, A. L. Delcher, M. Smoot, M. Shumway, C. Antonescu, S. L. Salzberg, Versatile and open software for comparing large genomes, *Genome Biol*. **5** (2004) #R12.

[52] M. Hohl, S. Kurtz, E. Ohlebusch, Efficient multiple genome alignment, *Bioinf*. **18** (2002) S312–S320.

[53] J. Buhler, Efficient large-scale sequence comparison by locality-sensitive hashing, *Bioinf*. **17** (2001) 419–428.

[54] M. I. Abouelhoda, E. Ohlebusch, Multiple genome alignment: Chaining algorithms revisited, in: R. Baeza–Yates, E. Chávez, M. Crochemore (Eds.), Combinatorial Pattern Matching, Springer, Berlin, 2003, pp. 1–16.

[55] A. C. Darling, B. Mau, F. R. Blattner, N. T. Perna, Mauve: multiple alignment of conserved genomic sequence with rearrangements, *Genome Res*. **14** (2004) 1394–1403.

[56] J. Kececioglu, The maximum weight trace problem in multiple sequence alignment, in: A. Apostolico, M. Crochemore, Z. Galil, U. Manber (Eds.), *Combinatorial Pattern Matching,* Springer, Berlin, 1993, pp. 106–119.

[57] D. Sankoff, J. Kruskal, *Time Warps, String Edits, and Macromolecules – The Theory and Practice of Sequence Comparison*, Addison-Wesley, Reading, 1983.

[58] M. Vingron, P. A. Pevzner, Multiple sequence comparison and consistency on multipartite graphs, *Adv. Appl. Math*. **16** (1995) 1–22.

[59] Y. Lin, S. Nurk, P. A. Pevzner, What is the difference between the breakpoint graph and the de Bruijn graph, *BMC Genomics* **15** (2014) S6–S6.

[60] P. A. Pevzner, 1-Tuple DNA sequencing: computer analysis, *J. Biomol. Struct. Dyn*. **7** (1989) 63–73.

[61] P. A. Pevzner, H. Tang, M. S. Waterman, An Eulerian path approach to DNA fragment assembly, *Proceed. National Acad. Sci*. **98** (2001) 9748–9753.

[62] P. A. Pevzner, H. Tang, G. Tesler, De novo repeat classification and fragment assembly, *Genome Res*. **14** (2004) 1786–1796.

[63] C. Lee, C. Grasso, M. F. Sharlow, Multiple sequence alignment using partial order graphs, *Bioinf*. **18** (2002) 452–464.

[64] A. M. Novak, G. Hickey, E. Garrison, S. Blum, A. Connelly, A. Dilthey, J. Eizenga, M. S. Elmohamed, S. Guthrie, A. Kahles, Genome graphs, *BioRxiv* (2017) #101378.

[65] B. Paten, J. Herrero, K. Beal, S. Fitzgerald, E. Birney, Enredo and Pecan: genome-wide mammalian consistency-based multiple alignment with paralogs, *Genome Res*. **18** (2008) 1814–1828.

[66] B. Paten, D. Earl, N. Nguyen, M. Diekhans, D. Zerbino, D. Haussler, Cactus: Algorithms for genome multiple sequence alignment, *Genome Res*. **21** (2011) 1512–1528.

[67] B. Paten, M. Diekhans, D. Earl, J.S. John, J. Ma, B. Suh, D. Haussler, Cactus graphs for genome comparisons, *J. Comput. Biol*. **18** (2011) 469–481.

[68] Y. H. Tsin, A simple 3-edge-connected component algorithm, *Theory Comput. Syst*. **40** (2007) 125–142.

[69] F. Harary, *Graph Theory*, Addison–Wesley, Reading, 1969.

[70] N. Jafarzadeh, A. Iranmanesh, Outerplanar graph data structure: A new computational analysis model of genome rearrangements, *MATCH Commun. Math. Comput. Chem.* **82** (2019) 581–598.

[71] J. A. Edwards, U. Vishkin, Brief announcement: speedups for parallel graph triconnectivity, *Proceed. Twenty–Fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures,* 2012, pp. 190–192.

[72] S. L. Mitchell, Linear algorithms to recognize outerplanar and maximal outerplanar graphs, *Inf. Process. Lett*. **9** (1979) 229–232.

[73] J. M. Six, I.G. Tollis, Circular drawings of biconnected graphs, in: M. T. Goodrich, C. C. McGeoch, *Algorithm Engineering and Experimentation*, Springer, Berlin, 1999, pp. 57–73.

[74] M. M. Syslo, Characterizations of outerplanar graphs, *Discr. Math*. **26** (1979) 47–53.

[75] F. Harary, C. S. J. Nash–Williams, On eulerian and hamiltonian graphs and line graphs, *Canad. Math. Bull*. **8** (1965) 701–709.

[76] D. Liu, S. Trajanovski, P. Van Mieghem, ILIGRA: an efficient inverse line graph algorithm, *J. Math. Model. Alg. Operations Res*. **14** (2015) 13–33.

[77] B. Kehr, K. Trappe, M. Holtgrewe, K. Reinert, Genome alignment with graph data structures: a comparison, *BMC Bioinf*. **15** (2014) #99.