IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received December 28, 2021, accepted January 11, 2022, date of publication January 18, 2022, date of current version January 27, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3144113

Graph Theoretical Strategies in De Novo Assembly

KIMIA BEHIZADI, NAFISEH JAFARZADEH, AND ALI IRANMANESH
Department of Mathematics, Faculty of Mathematical Sciences, Tarbiat Modares University, Tehran 14115-111, Iran

Corresponding author: Ali Iranmanesh (iranmanesh@modares.ac.ir)

This work was supported by the Research Core Bio-Mathematics with Computational Approach of Tarbiat Modares University under Grant IG-39706.

ABSTRACT De novo genome assemblers assume the reference genome is unavailable, incomplete, highly fragmented, or significantly altered as in cancer tissues. Algorithms for de novo assembly have been developed to deal with and assemble a large number of short sequence reads from genome sequencing. In this review paper, we have provided an overview of the graph-theoretical side of de novo genome assembly algorithms. We have investigated the construction of fourteen graph data structures related to OLC-based and DBG-based algorithms in order to compare and discuss their application in different assemblers. In addition, the most significant and recent genome de novo assemblers are classified according to the extensive variety of original, generalized, and specialized versions of graph data structures.

INDEX TERMS Combinatorial data structures, de-Bruijn graph, de novo assembly algorithms, high throughput sequencing, overlap graph.

I. INTRODUCTION

Since the completion of the human genome project at the turn of the century, there has been an unprecedented expansion of genomic sequence data. The de novo genome assembly is one of the big data challenges in bioinformatics to reconstruct a genome from a collection of short sequencing reads without the aid of a reference genome. To date, there are three generations of genome sequencing technologies. The first technology, so-called Sanger sequencing, was developed in 1977 [1], [2]. Although this technology is a very expensive cost and low throughput technique but it was used to obtain the first human genome sequence.

Second-generation sequencing, so-called next-generation sequencing (NGS), is the start of high throughput sequencing (HTS) and genome sequencing is being revolutionized by the development and commercialization of HTS. Second-generation sequencing developed a few decades after the Sanger sequencing method as a deep, high-throughput, and reduced-cost sequencing technology. The NGS technology can generate millions of short reads in parallel with a low cost of sequencing and speeding up the process compared with the Sanger method, also the output is detected directly without the need for electrophoresis.

The third-generation sequencing (TGS) started in the 2010s to produce reads longer than NGS without amplification. Mathematically, de novo genome assembly is an NP-hard problem which does not admit an efficient computational solution. Compared with the comparative assembly, the de novo assembly is more demanding and in practice can be a daunting task, especially when there are many reads to assemble, which is generally the case. A fundamental tool used for de novo assembly is a graph representation of the relationships between the reads sharing common prefixes and suffixes. Graph data structures are important and efficient frameworks for algorithms of computational biology which are used for sequence alignment, genome assembly, and analysis of genome rearrangements [3]–[6].

Two main computational approaches for representing overlaps between reads in de novo genome assembly on HTS data are Overlap-Layout-Consensus (OLC) algorithm and the de-Bruijn graph (DBG) algorithm. The OLC algorithm is based on constructing an overlap graph by overlapping similar sequences. This approach initially introduced in 1980 [7] and afterward extended and developed by many scientists. The first OLC assembler was introduced in 2000 [8] for Sanger data and later was updated for NGS data too. The DBG

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

algorithm is based on k-mers approach, which splits the short reads into smaller k-mers and then builds a de-Bruijn graph. Most state-of-the-art de novo assemblers have used the de-Bruijn graph as a data structure of their assembly algorithms. The de-Bruijn graph was first brought to bioinformatics in 1989 as a method for sequencing by hybridization [9]. The de-Bruijn graph algorithm for de novo assembly was originally proposed in 1995 [10] and the first DBG assembler was proposed in 2001 [11]. This work will investigate all specialized versions of the basic graph frameworks used for de novo assembly of HTS data and classify important assemblers in both OLC and DBG approaches based on their graph data structures.

II. DATA STRUCTURES OF OLC-BASED ASSEMBLERS

The OLC approach is composed of three steps, first computing the overlaps between the reads, then laying out the overlap information on a graph data structure, and finally inferring the consensus sequence. The main graph data structure of assemblers based on the OLC method is called overlap graph, also there is a simplified version of overlap graph called string graph which is obtained after removing all redundant edges. In this section, we study the construction of these two graphs and their application in important assemblers on HTS data. Fig. 1 shows a brief overview of the graph data structures in the OLC method.

A. OVERLAP GRAPH

The overlap graph proposed by Kececioglu and Myers [12] is a bi-directed graph whose vertices are the input reads and each edge $e = (u, v)$ represents a connection between two reads u and v if a suffix of u matches a prefix of v. Each edge in the overlap graph has two arrowheads at its endpoints and the orientations of the arrowheads are used to denote the different ways in which the two reads at the ends of an edge can overlap [13].

1) OVERLAP GRAPH-BASED ASSEMBLERS

Assembler *Celera* [8] is the first overlap graph-based assembler which was developed at the time of Sanger sequencing and then modified to support NGS data. Assembler *CABOG* [14] is the revised pipeline of Celera which constructs an overlap graph from the reads and reports the best overlaps which are used to build initial un-gapped multiple sequence alignments and then assemble contigs. Assembler *Newbler* [15] is another assembler was designed for NGS data which adapted the overlap graph. Assembler *Miniasm* [16] builds an overlap graph by mapping all pairs of reads with Minimap aligner [16] and uses the MinHash sketch [17] to compare two sets of k-mers. Assembler *Canu* [18] is derived from the Celera which is specialized in the assembly of TGS long reads. Assembler *HINGE* [19] builds an overlap graph to obtain pairwise alignments between all reads by using DALIGNER [20]. Assembler *Marvel* [21] creates the best overlap graph and obtains contig paths for long-read data. Assembler *Peregrine* [22] scans all the reads to construct a hash map to record the read locations and uses sparse hierarchical minimizers to index reads. Assembler *Raven* [23] is the upgrade version of *Ra* [24] which builds an overlap graph using pairwise overlaps generated by Minimap2 [25]. Assembler *HiCanu* [26] is a modification of the Canu designed for PacBio highly accurate long reads. Assembler *SMARTdenovo* [27] is a single-molecule sequencing assembler which applies the best overlap graph to generate the layout of the reads and the PBDAG-Con algorithm [28] to generate a consensus.

B. STRING GRAPH

The string graph proposed by Myers *et al.* [29] is built by constructing a graph of the pairwise overlaps between sequence reads and transforming it into a string graph by removing transitive edges [30]. A String graph can be derived from the overlap graph by removing duplicate reads and contained reads and then removing transitive edges from the graph. Each edge in a string graph is bidirectional to model the double-stranded nature of DNA and labeled with the unmatched substrings of the sequence reads [31].

1) STRING GRAPH-BASED ASSEMBLERS

Assembler *Edena* [13] is the first string graph-based assembler designed for early short-read sequencing data. This assembler computes overlaps between reads using suffix array and performs transitive edge removal then indexes all sequence reads in a prefix tree. Assembler *SGA* [32] is based on the directed string graph where uses the Burrows-Wheeler Transform [33] and FM-index [30] to find overlaps between reads. Assembler *Fermi* [34] is inspired by SGA and uses FMD-index [33] to represent both DNA strands inside a unique structure. Assembler *RJ* [33] or Read Joiner is based on efficient computation of a subset of exact suffix-prefix matches and by subsequent rounds of suffix sorting, scanning, and filtering obtain the non-redundant edges of the graph. Assembler *FALCON* [35] is a haplotype-aware assembler for large genome assembly which builds a string graph by using DALIGNER [20]. Assembler *FALCON-Unzip* [35] takes the contigs from FALCON and phases the reads based on heterozygous single nucleotide polymorphisms identified in the initial assembly. Assembler *Hifiasm* [36] builds a string graph where a vertex is an oriented read and an edge is a consistent overlap. After transitive reduction, a pair of heterozygous alleles will be represented by a bubble in the string graph. Assembler *NECAT* [37] follows an approach similar to Canu, it constructs a directed string graph and removes transitive edges using Myer's algorithm [29].

III. DATA STRUCTURES OF DBG-BASED ASSEMBLERS

The de-Bruijn graph was developed to represent strings from a finite alphabet motivated by the superstring problem [38]. The vertices of DBG represent all possible k-length strings, so-called k-mers, where k is an arbitrarily fixed integer and the edges represent suffix to prefix perfect overlaps. The DBG approach for genome assembly is performed in two steps, first

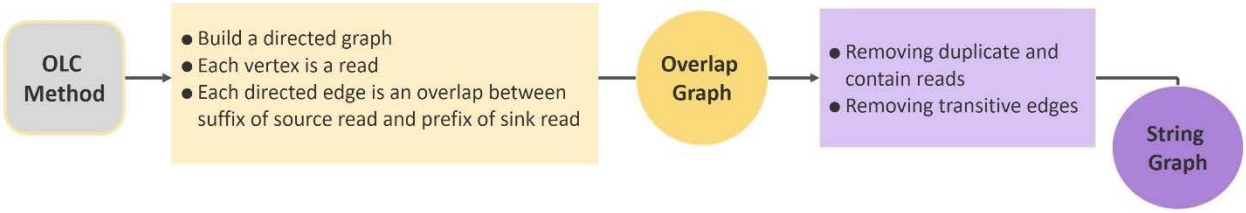

FIGURE 1. Diagram of graph data structures in OLC method. Overview of how to construct an overlap graph in OLC-based algorithm of de novo genome assemblers and how to obtain a string graph from overlap graph.

constructing the de-Bruijn graph from the set of all k-mers and then finding the shortest superstring that contains all possible k-mers. Fig. 2 shows a brief description of all graph data structures in the DBG method.

A. DE-BRUIJN GRAPH

The de-Bruijn graph is a common data structure used for de novo genome assembly which stores all k-mers contained in a given set of sequences as vertices and edges. For a set of reads, there are two types of DBG data structures: Hamiltonian DBG and Eulerian DBG. In the Hamiltonian approach [10] vertices are all the distinct k-mers and the pair (u, v) of k-mers is an arc if the length $(k-1)$ suffix of u is equal to the length $(k-1)$ prefix of v. In this approach, the sub-sequences are assembled by finding Hamiltonian paths that traverse all nodes, each of which is visited only once. This approach is known as the NP-complete problem when the number of nodes is not trivial [9]. In the Eulerian approach [11] the vertices are the set of all $(k-1)$-mers in the reads and there is an arc from u to v if and only if there is a k-mer in the reads with prefix u and suffix v. In this approach, the sub-sequences are assembled by finding Eulerian paths that traverse all edges, each of which is visited only once. The most commonly used approach to construct a de-Bruijn graph for genome assembly is the Eulerian DBG which has polynomial time complexity.

1) DE-BRUIJN GRAPH-BASED ASSEMBLERS

Assembler *EULER* [11] divides the reads into k-mers and represents each read as a walk on a de-Bruijn graph, then searches for a super-walk that contains all the reads. Assembler *Velvet* [39] is an assembler for short-read data which k-mers are first hashed and then velvet finds exact local alignments and builds a de-Bruijn graph from them. Assembler *ABySS* [40] implements a distributed representation of a de-Bruijn graph to parallel computation. In ABySS, the value associated to all indexed k-mers is just eight bits coding the presence or absence of its eight possible neighbors. Assembler *SOAPdenovo* [41] uses de-Bruijn graph data structure and simplifies the graph by merging unambiguously connected vertices into one. Assembler *Gossamer* [42] is based on the succinct representation of de-Bruijn graphs as a bitmap or set of integers [43] and provides multiple operations for removing spurious edges from the graph. Assembler *Platanus* [44] using multiple k-mer sizes, constructs the de-Bruijn graphs from reads, then modifies the graphs and displays the output sequences of contigs. In *EPGA* [45] if the occurrence of one k-mer is over one, the k-mer will be considered in constructing the de-Bruijn graph otherwise, the k-mer will be thought of including erroneous bases and will be removed. Assembler *EPGA2* [46] resolves the memory efficiency problem in EPGA and updates some modules in EPGA. It employs DSK [47] to count k-mers and $(k+1)$-mers which only requires a fixed user-defined amount of memory. Assembler *ScalaDBG* [48] is a scalable genome assembler through parallel de-Bruijn graph construction for multiple k-mers. This assembler first performs graph construction in parallel for each k-value, then for each pair of graphs, the higher k-valued graph is patched using the lower k-valued graph to generate a single graph.

B. A-BRUIJN GRAPH

An important generalized version of the de-Bruijn graph for genome assembly is the A-Bruijn graph [49] which gets its name from being a combination of a de-Bruijn graph and an adjacency matrix or A-matrix. Vertices of the graph represent consecutive columns in multiple sequence alignments and all vertices that are similar to one another are collapsed into one vertex. Let S be a genomic sequence of length n and similarity matrix $A = a_{ij}$ be a binary $n \times n$ representing the set Γ 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. Note that gaps are not considered in A. The A-Bruijn graph $G(V, E)$ is defined as the multigraph on the vertex set V with $(k-1)$ directed edges (v_i, v_{i+1}) for $1 \leq i \leq n$ [49]. For an arbitrary collection of alignments Γ, the A-Bruijn graph is defined to work with imperfect repeats and is equivalent to the de-Bruijn graph in the special case that Γ is the collection of all perfect similarities of k-mers. Also as shown in [50], constructing this A-Bruijn graph is equivalent to constructing the breakpoint graph from multiple genomes to be used for genome rearrangement.

1) A-BRUIJN GRAPH-BASED ASSEMBLERS

Assembler *EULER+* [49] is the first assembler based on the notion of A-Bruijn graphs. It deals with errors in reads by inducing vertices with un-gapped alignments that

FIGURE 2. Diagram of graph data structures in DBG method. Overview of how to construct a de-Bruijn graph in DBG-based algorithm of de novo genome assemblers and how to obtain each of eleven generalized versions of de-Bruijn graph.

allow mismatches, rather than the exact k-mers in de-Bruijn assembly. Assembler *EULER-SR* [51] is a modified version of EULER+ assembler which presents a memory-efficient DBG-based approach. Assembler *ABruijn* [52] is a DBG-based de novo assembler for long and noisy reads which uses an A-Bruijn graph to find the overlaps between reads and does not require them to be error-corrected. Assembler *Dnaasm* [53] is another A-Bruijn graph based assembler which utilizes the frequency of reads to reconstruct tandem repetitive sequences. This assembler makes A-Bruijn graph, then approximates the number of occurrences of a given DNA fragment, restores the tandem repeats by the correction of the edge weights, and finally generates a DNA sequence from the A-Bruijn graph.

C. UniPath GRAPH

A maximal unbranched sequence of edges is called a uni-path [54] and each given k-mer lies in exactly one uni-path. UniPath graph is a simplified representation of DBG whose edges are the uni-paths.

1) UNIPATH GRAPH-BASED ASSEMBLERS

Assembler *ALLPATHS* [55] uses the UniPath graph as a representation of an assembly consisting of edges representing contiguous and unambiguous sequences of bases and vertices representing junction points between edges. Assembler *ALLPATHS2* [56] is a modified version of ALLPATHS where generates assemblies with long, accurate contigs and scaffolds. Assembler *ALLPATH-LG* [57] is an improvement version of ALLPATHS and ALLPATHS2 which is more resilient to repeats and in the UniPath graph, collapses repeats of length more than K, where K is chosen to be short enough that overlaps of length between reads are abundant.

D. SPARSE DE-BRUIJN GRAPH

The Sparse de-Bruijn graph is a model of DBG which skips some k-mers and uses only a subset of them to reduce time and memory use [56]. In the standard DBG structure, every k-mer in the graph has only one neighboring nucleotide base on each side for the linear part and each k-mer is considered in DBG. But in the Sparse de-Bruijn graph, only one out of every g ($g \leq k$) k-mers is stored attempting to subsample as evenly across the original graph as possible. In the sparse de-Bruijn graph, the nodes in the graph represent a $1/g$ subsample of the k-mer variety in the entire genome and skip some other k-mers to save more neighboring bases. An example of Sparse DBG is shown in Fig. 3.

1) SPARSE DE-BRUIJN GRAPH-BASED ASSEMBLERS

Assembler *SparseAssembler* [58] is based on the construction of the sparse DBG where the graph stores only a small fraction of the observed k-mers as vertices and the edges between these vertices allow the de novo assembly of even moderately-sized genomes on a typical laptop computer. Assembler *SOAPdenovo2* [59] is an improvement of SOAPdenovo. It implements sparse DBG approach where reads are cut into k-mers and a large number of the linear unique k-mers are combined as a group instead of being stored independently.

E. ITERATIVE DE-BRUIJN GRAPH

The Iterative de-Bruijn graph [60] is built from multi k values. This approach iterates the construction and analysis of the de-Bruijn graph on a range of k values from $k_1 = k_{\min} < k_2 < \ldots < k_{\max} = k_n$. Let DBG$(R, k)$ be the de-Bruijn graph of k-mer size form a set of reads R and consider $G(R, k_1) =$ DBG(R, k_1) and $C(k_i)$ is the set of contigs from $G(R, k_i)$, define $G(R, k_{i+1}) =$ DBG$(R \cup C(k_i), k_{i+1})$ and finally $G(R, k_n)$ is an Iterative DBG. A schematic process of this graph is shown in Fig. 4.

1) ITERATIVE DE-BRUIJN GRAPH-BASED ASSEMBLERS

Assembler *IDBA* [60] maintains an accumulated de-Bruijn graph at each iteration to carry useful information forward as it moves on to higher k-values. Assembler *IDBA-UD* [61] is an extension of IDBA which is designed to utilize paired-end reads to assemble low-depth regions. Assembler *SKESA* [62] is a de-novo assembler based on an Iterative de-Bruijn graph for microbial genomes using Illumina sequencing data.

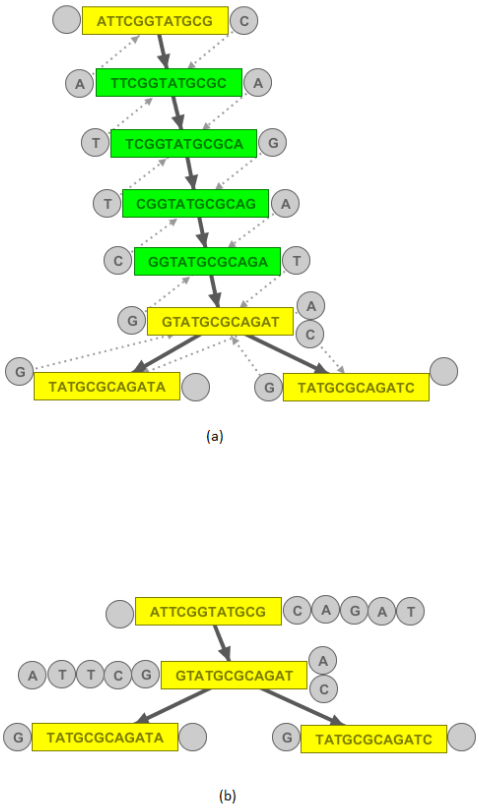

FIGURE 3. Construction of the Sparse de-Bruijn graph. (a) A de-Bruijn graph with branches of each vertex. (b) The Sparse de-Bruijn graph after skipping green vertices.

F. PAIRED DE-BRUIJN GRAPH

Paired de-Bruijn graph (PDBG) is a generalization of DBG that incorporates mate pair information into the graph structure itself instead of analyzing mate-pairs at a post-processing step [63]. A mate pair is a pair of reads with a distance of d between their start positions and a k-bimer $(a|b)$ is a pair of k-mers, a and b where prefix $(a|b) =$ (Prefix (a)|Prefix (b)) and suffix $(a|b) =$ (suffix (a)|suffix (b)). Also a (k, d)-bimer, is a pair of k-mers with a distance of d between their start positions. To construct a PDBG, for each k-bimer $(a|b)$, consider two new vertices $u =$ prefix $(a|b)$, $v =$ suffix $(a|b)$ and label the edge by $(a|b)$, then glue vertices of this graph together when they have the same label, the obtained graph is PDBG. Fig. 5 shows the construction of this graph for a mate pair.

1) PAIRED DE-BRUIJN GRAPH-BASED ASSEMBLER

Assembler *SPAdes* [64] implements iterative DBG and PDBG in the same framework. At first, it constructs an assembly graph using the iterative DBG and derives accurate distance estimates between k-mers in the genome using joint analysis

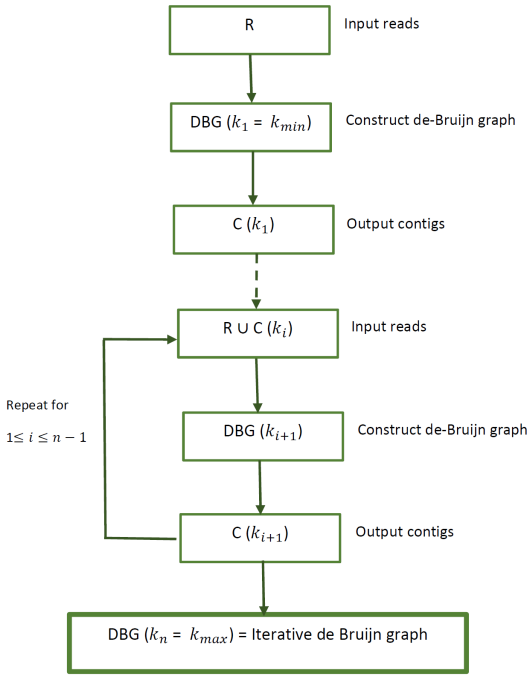

FIGURE 4. **The workflow of construction the Iterative de-Bruijn graph from a set of reads.**

of distance histograms and paths in the graph, then constructs the paired assembly graph inspired by the PDBG approach.

G. COLORED DE-BRUIJN GRAPH

Iqbal *et al.* [65] presented the colored de-Bruijn graph (CDBG) where the vertices and edges structure of CDBG is the same as the classic structure of DBG, but to each vertex ($(k-1)$-mer) and edge (k-mer) is associated a list of colors corresponding to the samples in which the vertex or edge label exists [66]. Colored de-Bruijn graph generalizes the original formulation to multiple samples embedded in a union graph, where the identity of each sample is retained by coloring those nodes present in a sample. The samples may reflect HTS data from multiple samples, experiments, reference sequences, known variant sequences, or any combination of these [65]. Fig. 6 illustrates an example of CDBG with three colors.

1) COLORED DE-BRUIJN GRAPH-BASED ASSEMBLER

Assembler *Cortex* [65] is the first de novo assembly-based algorithm for direct variant calling from short reads. It builds CDBG and performs variant calling and genotyping from HTS data.

H. PROBABILISTIC DE-BRUIJN GRAPH

Pell *et al.* [67] introduced the probabilistic de-Bruijn graph which is a memory-efficient representation of DBG based on Bloom filters [68]. A Bloom filter is a probabilistic data structure used to test set membership and tells if an element may be in a set, or definitely is not. The probabilistic DBG is obtained by inserting all k-mers of a DBG in a Bloom filter.

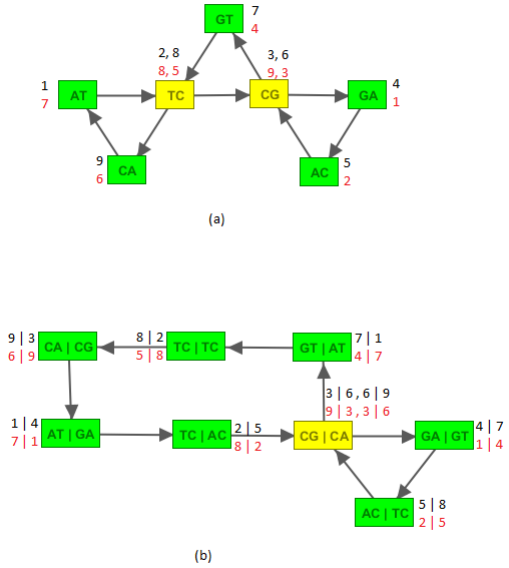

FIGURE 5. **Construction of a paired de-Bruijn of a mate pair ATCGACGTC with $d = 3$. (a) Constructing de-Bruijn graph for a mate pair with $k = 2$. Black and red numbers belong to each sequence of mate pair. There are two repeated vertices in this de-Bruijn graph shown in yellow. (b) Constructing the paired de-Bruijn graph. There is only one repeated vertex in the PDBG.**

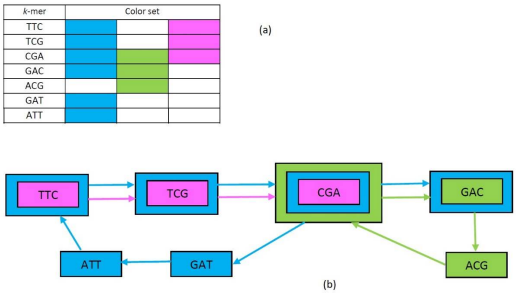

FIGURE 6. **Construction of Colored de-Bruijn graph of three sequences; TTCGA, CGATTCGAC and CGACGA with pink, blue and green colors for k-mers respectively.**

The Bloom filter data structure consists of a bit vector and one or more hash functions, where the hash functions map each k-mer to a corresponding set of positions within the bit vector [69]. Fig. 7 shows an example of Probabilistic DBG.

1) PROBABILISTIC DE-BRUIJN GRAPH-BASED ASSEMBLERS

Assembler *Minia* [70] is a short-read assembler based on probabilistic DBG which is implicitly encoded as a Bloom filter. Assembler *ABySS2* [69] is an improvement of ABySS where follows the approach of Minia to encode the DBG to a probabilistic DBG.

I. REPEAT GRAPH

Repeat graph [49] is a simplified version of the A-Bruijn graph where similar k-mers are collapsed into a single vertex and this vertex labeled by the consensus sequence of all collapsed k-mers. Two positions in the genome are defined

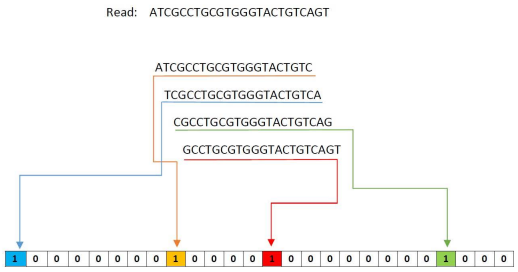

FIGURE 7. An example of storing the *k*-mers of a read into a Bloom filter by computing the hash values of each *k*-mer and setting the corresponding bit in the Bloom filter.

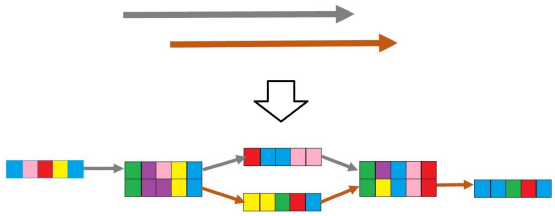

FIGURE 8. Construction of Fuzzy Bruijn graph. Consider pairwise alignment of two binning sequences and construct the Fuzzy Bruijn graph. Bins with same colors denote shared *k*-mers of two reads.

as equivalent if they are aligned against each other in one of these alignments. The repeat graph compactly represents all repeats in a genome and reveals their mosaic structure.

1) REPEAT GRAPH-BASED ASSEMBLERS

Assembler *Flye* [71] is a de novo assembler for single molecule sequencing reads which constructs the repeat graph of long reads with the goal to approximate the DBG in the case of a large k. Flye assembler utilizes the constructed repeat graph for the resolution of unbridged repeats which are not bridged by any reads. Assembler *MosaicFlye* [72] is an algorithm for resolving complex unbridged repeats where uses variations between various copies of a mosaic repeat for resolving these copies and thus untangling the repeat graph of reads constructed by Flye assembler. Also, *MetaFlye* [73] is a special mode of Flye assembler for metagenome assembly and *CentroFlye* [74] is an assembler for centromere assembly using long error-prone reads.

J. MARKER GRAPH

A recently published assembly tool [75] uses Run-Length-Encoding (RLE) [25] as a representation of sequences. The RLE is a data compression method for text which contains a large repetition of the same character. In this form, identical consecutive bases are collapsed, and the base and repeat count are stored. For instance, the sequence *GATTTACCA* would be represented as (*GATACA*, 113121). In this representation each k-mer is called a marker and a marker graph is similar to DBG, where a k-mer is a marker and an edge is built between two markers if a read contains this succession of markers.

1) MARKER GRAPH-BASED ASSEMBLER

Assembler *Shasta* [75] uses a compact representation of the marker graph where an edge is built between two markers if a read contains this succession of markers and that is weighted by the number of reads that contains this succession.

K. FUZZY BRUIJN GRAPH

A new data structure for sequence assembly which is related to sparse DBG and A-Bruijn graphs is the Fuzzy Bruijn graph (FBG) [76]. The FBG extends basic ideas behind the DBG to work with long noisy reads. In FBG, each base is considered as a 256 bp bin and a vertex is a k-bin which is a sequence of k consecutive bins, different k-bins may be represented by a single vertex if they are aligned together in a sequence alignment routine. An edge between two vertices in FBG indicates their adjacency on a read. Fig. 8 shows a schematic example of FBG construction from two sequences.

1) FUZZY BRUIJN GRAPH-BASED ASSEMBLER

Assembler *Wtdbg2* [76] reads all input sequences into memory and encodes each base with 2 bits and builds a hash table for the k-mers occurring at least twice and at most thousand times. It takes each bin as a base pair and applies Smith–Waterman dynamic programming [77] between binned sequences, penalizing gaps and mismatching bins that do not share k-mers.

L. MINIMIZER-SPACE DE-BRUIJN GRAPH

Minimizer-space de-Bruijn graph (mdBG) [78] is a novel data structure which instead of building an assembly over sequence bases, performs assembly over short sequences of bases called minimizers and later converts it back to bases assemblies. For an integer $k > 2$ and an integer $l > 1$, a mdBG of order k is a de-Bruijn graph of order k over the Σ^l alphabet. The nodes are k-min-mers (an ordered list of k minimizers), and edges correspond of identical suffix-prefix overlaps of length $(k-1)$ between k-min-mers.

1) MINIMIZER-SPACE DE-BRUIJN GRAPH-BASED ASSEMBLER

Assembler *Rust-mdbg* [78] is a modular assembler which uses mdBG structure for assembling long and accurate reads. It runs in minimizer-space where the reads, assembly graph, and the final assembly are all represented as ordered lists of minimizers instead of strings of bases.

IV. HYBRID ASSEMBLY

De novo assemblers are classified into short-read, long-read, and hybrid assemblers. Short-read assemblers are considered for the second-generation sequencing data with lengths ranging less than 200-400 bp. For the third-generation sequencing data where the size of reads is more than 400 bp, long-read assemblers are used. And hybrid assemblers are applied when a combination of the short and long reads is considered. Combine the concept of DBG and OLC method to make an

TABLE 1. Short-read de novo assemblers.

Assembler	Algorithm	Graph data structure	Sequencing platforms	Reference
Celera	OLC-based	Overlap graph	454, Illumina, SOLiD, Ion Torrent	[8]
Newbler	OLC-based	Overlap graph	454	[15]
Edena	OLC-based	String graph	Illumina	[13]
CABOG	OLC-based	Overlap graph	454, Illumina, SOLiD, Ion Torrent	[14]
Euler-SR	DBG-based	de-Bruijn graph	454, Illumina	[51]
Velvet	DBG-based	de-Bruijn graph	454, Illumina, SOLiD	[39]
ALLPATHS	DBG-based	UniPath graph	Illumina	[55]
ALLPATHS 2	DBG-based	UniPath graph	Illumina	[56]
ABySS	DBG-based	de-Bruijn graph	454, Illumina, SOLiD	[40]
SOAPdenovo	DBG-based	de-Bruijn graph	Illumina	[41]
IDBA	DBG-based	Iterative de-Bruijn graph	Illumina	[60]
ALLPATHS-LG	DBG-based	UniPath graph	Illumina	[57]
SGA	OLC-based	String graph	Illumina	[32]
Fermi	OLC-based	String graph	Illumina	[34]
Readjoiner	OLC-based	String graph	Illumina	[33]
SparseAssembler	DBG-based	Sparse de-Bruijn graph	Illumina	[58]
SOAPdenovo 2	DBG-based	Sparse de-Bruijn graph	Illumina	[59]
IDBA-UD	DBG-based	Iterative de-Bruijn graph	Illumina	[61]
Gossamer	DBG-based	de-Bruijn graph	Illumina	[42]
SPAdes	DBG-based	Paired de-Bruijn graph	Illumina, IonTorrent	[64]
Cortex	DBG-based	Colored de-Bruijn graph	Illumina	[65]
Minia	DBG-based	Probabilistic de-Bruijn graph	Illumina	[70]
Platanus	DBG-based	de-Bruijn graph	Illumina	[44]
EPGA/ EPGA 2	DBG-based	de-Bruijn graph	Illumina	[45]/ [46]
ABySS2	DBG-based	Probabilistic de-Bruijn graph	Illumina	[69]
SKESA	DBG-based	de-Bruijn graph	Illumina	[62]
dnaasm	DBG-based	A-Bruijn graph	Illumina	[53]
SCALADBG	DBG-based	de-Bruijn graph	Illumina	[48]

efficient algorithm for hybrid reads and in general, there are four hybrid assembly strategies [79]:

1) Long reads could be mapped directly onto the DBG, which is built from the short reads. Then, dedicated algorithms allow us to resolve some ambiguity in the DBG to improve the consistency of the resulting sequences.
2) Long reads could be de novo assembled with dedicated assemblers and the created contigs are improved by mapping short reads and correcting assembly errors.
3) Short reads could be used to correct long reads and then long and corrected reads could be assembled with assemblers for third-generation sequencing data.
4) Short reads could be de novo assembled using assemblers dedicated to second-generation sequencing data and then long reads link the resulting contigs.

2) HYBRID ASSEMBLERS

Assembler *Meraculous* [80] is a hybrid assembler which follows the first hybrid assembly strategy. Meraculous first constructs and traverses a simplified de-Bruijn graph to assemble unique regions of the genome into uncontested "UU" contigs. In the next step, the contigs are aligned to paired-end read data, and gaps are filled using localized assemblies of relevant reads. Super-Read Celera Assembler (*MaSuRCA*) [81] is a hybrid assembler based on the second hybrid assembly strategy. The assembler uses a modified version of the CABOG assembler that turns large numbers of reads into much smaller numbers of longer super-reads. Super-reads can be easily computed using a de-Bruijn graph. Once the super-reads are created, they, along with the mate pairs that connect them, collectively replace the de-Bruijn graph. Incorporating pair-mate information is performed using the OLC assembly. Assembler *DBG2OLC* [82] is a hybrid assembler which follows the third hybrid assembly strategy. The algorithm starts with linear unambiguous regions of a de-Bruijn graph and ends up with linear unambiguous regions in an overlap graph. Assembler *HASLR* [83] is also a hybrid assembler which uses the third hybrid assembly strategy. The input is a set of long-reads and a set of short-reads from the same sample, together with an estimation of the genome size. It builds short-read contigs using Minia assembler [70], then it uses long-reads to put contigs in the order of their expected appearance in the genome. Assembler *HybridSPAdes* [84] is a hybrid assembler which uses the fourth hybrid assembly strategy. The tool first constructs the assembly graph from short reads using SPAdes assembler [64], then maps long reads to the assembly graph and generates read-paths, then closes gaps in the assembly graph using the consensus of long reads that span the gaps. Another hybrid assembler following the fourth hybrid assembly strategy is *WENGAN* [85]. This assembler integrates short reads in the early phases of the assembly process. Assembler WENGAN starts by building short-read contigs using a de-Bruijn graph assembler. Then, the pair-end reads are pseudo-aligned back to detect and error-correct chimeric contigs as well as to classify them as repeats or unique sequences. Assembler *Unicycler* [86] is also a hybrid assembler which uses the fourth hybrid assembly

TABLE 2. Long-read de novo assemblers.

Assembler	Algorithm	Graph data structure	Sequencing platforms	Reference
Miniasm	OLC-based	Overlap graph	PacBio, ONT	[16]
ABruijn	DBG-based	A-Bruijn graph	PacBio, ONT	[52]
FALCON	OLC-based	String graph	PacBio	[35]
FALCON-Unzip	OLC-based	String graph	PacBio	[35]
Canu	OLC-based	Overlap graph	PacBio, ONT	[18]
HINGE	OLC-based	Overlap graph	PacBio	[19]
Marvel	OLC-based	Overlap graph	PacBio	[21]
Flye	DBG-based	Repeat graph	PacBio, ONT	[71]
Peregrine	OLC-based	Overlap graph	PacBio	[22]
HiCanu	OLC-based	Overlap graph	PacBio, ONT	[26]
Shasta	DBG-based	Marker graph	ONT	[75]
Wtdbg2	DBG-based	Fuzzy Bruijn graph	PacBio, ONT	[76]
MetaFlye	DBG-based	Repeat graph	PacBio, ONT	[73]
CentroFlye	DBG-based	Repeat graph	PacBio, ONT	[74]
MosaicFlye	DBG-based	Repeat graph	PacBio, ONT	[72]
Hifiasm	OLC-based	String graph	PacBio	[36]
NECAT	OLC-based	String graph	ONT	[37]
Raven	OLC-based	Overlap graph	PacBio, ONT	[23]
SMARTdenovo	OLC-based	Overlap graph	PacBio, ONT	[27]
Rust-mdbg	DBG-based	Minimizer-space de-Bruijn graph	PacBio	[78]

strategy. This tool builds an initial assembly graph from short reads using the de novo assembler SPAdes [64] and then simplifies the graph using information from short and long reads. Unicycler uses a semi-global aligner to align long reads to the assembly graph.

V. DISCUSSION

Tables 1 and 2 describe the most commonly used and recent de novo genome assemblers on second-and third-generation sequencing data and classify them based on algorithm types and graph data structures. Approximately 74% of the short-read and 40% of long-read de novo assemblers are based on the DBG approach, also 60% of long-read and 26% of short-read de novo assemblers are based on the OLC approach. Generally, it can be estimated that 43% of de novo assemblers on HTS are based on the OLC approach and 57% are based on the DBG approach. As will be discussed in this section, these approaches have different advantages and disadvantages. In general, OLC-based assemblers are the most popular for long-read data. Overlap graph and string graph data structures lead to finding a Hamiltonian path which is known as an NP-complete problem, but they are more suitable than the de-Bruijn graphs for long sequences and single-molecule sequencing reads of high error rate. Vertices in an overlap graph are the input reads and an edge between two vertices is assigned when they overlap larger than a cutoff length. Also, a string graph is the simplified version of an overlap graph after removing duplicates and contains reads and also removing transitive edges. The string graph formulation is similar to the concept of the de-Bruijn graph with the advantage of not requiring the reads to be split into k-mers and also a string graph always maintains read coherence. The OLC approaches have major disadvantage of requiring alignments between every possible combination of reads which are extremely time-consuming for large sequencing datasets. The DBG-based data structures lead to resolving the Eulerian path problem to derive contig sequences and it is easier to find an Eulerian path for short-reads data than a Hamiltonian path. Another key advantage of de-Bruijn graphs is their ability to exploit the redundancy of high coverage HTS data. Most of the short-read assemblers are based on the standard representation of DBG, UniPath graph, Iterative graph, or Sparse DBG. Also, DBG-based assemblers for long reads are mainly based on construction A-Bruijn graph or its simplified version repeat graph. Marker graph, Fuzzy-Bruijn graph, and Minimizer space de-Bruijn graph are other graphs based on DBG-method used for long-read assembly. These graphs use some models to compress k-mers without losing data which can be efficient for long reads.

VI. CONCLUSION

The overlap-layout-consensus and the de-Bruijn graph algorithms are the main computational strategies for the de novo genome assembly problem. Overlap graph and de-Bruijn graph are two basic graph frameworks of genome assembly algorithms and there are some generalized and specialized versions of these graphs which can make assembly more efficient and easier. The purpose of this review is to provide an overview of the combinatorial side of de novo genome assembly algorithms on high-throughput sequencing data. This review described the construction and application of overlap graph and string graph, also investigated the de-Bruijn graph construction and all specialized representation of that. In addition, the important and recent genome de novo assemblers are classified according to the extensive variety of original, generalized, and specialized versions of graph data structures which were reviewed in detail.

ACKNOWLEDGMENT

(Kimia Behizadi and Nafiseh Jafarzadeh are co-first authors.) The authors would like to thank the referees for their valuable comments.

REFERENCES

[1] F. Sanger, S. Nicklen, and A. R. Coulson, "DNA sequencing with chain-terminating inhibitors," *Proc. Nat. Acad. Sci. USA*, vol. 74, no. 12, pp. 5463–5467, Dec. 1977, doi: 10.1073/pnas.74.12.5463.

[2] A. M. Maxam and W. Gilbert, "A new method for sequencing DNA," *Proc. Nat. Acad. Sci. USA*, vol. 74, no. 2, pp. 560–564, Feb. 1977, doi: 10.1073/pnas.74.2.560.

[3] N. Jafarzadeh and A. Iranmanesh, "On graph–based data structures to multiple genome alignment," *MATCH Commun. Math. Comput. Chem.*, vol. 83, no. 1, pp. 33–62, Feb. 2020.

[4] S. El-Metwally, T. Hamza, M. Zakaria, and M. Helmy, "Next-generation sequence assembly: Four stages of data processing and computational challenges," *PLoS Comput. Biol.*, vol. 9, no. 12, Dec. 2013, Art. no. e1003345, doi: 10.1371/journal.pcbi.1003345.

[5] J. Ghurye and M. Pop, "Modern technologies and algorithms for scaffolding assembled genomes," *PLOS Comput. Biol.*, vol. 15, no. 6, Jun. 2019, Art. no. e1006994, doi: 10.1371/journal.pcbi.1006994.

[6] J. Ghurye and M. Pop, "Modern technologies and algorithms for scaffolding assembled genomes," *PLOS Comput. Biol.*, vol. 15, no. 6, Jun. 2019, Art. no. e1006994, doi: 10.1371/journal.pcbi.1006994.

[7] R. Staden, "A mew computer method for the storage and manipulation of DNA gel reading data," *Nucleic Acids Res.*, vol. 8, no. 16, pp. 3673–3694, 1980, doi: 10.1093/nar/8.16.3673.

[8] E. W. Myers, G. G. Sutton, A. L. Delcher, I. M. Dew, D. P. Fasulo, M. J. Flanigan, and S. A. Kravitz, "A whole-genome assembly of drosophila," *Science*, vol. 287, no. 5461, pp. 2196–2204, Mar. 2000, doi: 10.1126/science.287.5461.2196.

[9] P. A. Pevzner, "L-tuple DNA sequencing: Computer analysis," *J. Biomol. Struct. Dyn.*, vol. 7, no. 1, pp. 63–73, Aug. 1989, doi: 10.1080/07391102.1989.10507752.

[10] R. M. Idury and M. S. Waterman, "A new algorithm for DNA sequence assembly," *J. Comput. Biol.*, vol. 2, no. 2, pp. 291–306, Jan. 1995, doi: 10.1089/cmb.1995.2.291.

[11] P. A. Pevzner, H. Tang, and M. S. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proc. Nat. Acad. Sci. USA*, vol. 98, no. 17, pp. 9748–9753, Aug. 2001, doi: 10.1073/pnas.171285098.

[12] J. D. Kececioglu and E. W. Myers, "Combinatorial algorithms for DNA sequence assembly," *Algorithmica*, vol. 13, nos. 1–2, pp. 7–51, Feb. 1995, doi: 10.1007/BF01188580.

[13] D. Hernandez, P. François, L. Farinelli, M. Østerås, and J. Schrenzel, "*De novo* bacterial genome sequencing: Millions of very short reads assembled on a desktop computer," *Genome Res.*, vol. 18, no. 5, pp. 802–809, May 2008, doi: 10.1101/gr.072033.107.

[14] J. R. Miller, A. L. Delcher, S. Koren, E. Venter, B. P. Walenz, A. Brownley, J. Johnson, K. Li, C. Mobarry, and G. Sutton, "Aggressive assembly of pyrosequencing reads with mates," *Bioinformatics*, vol. 24, no. 24, pp. 2818–2824, Dec. 2008, doi: 10.1093/bioinformatics/btn548.

[15] M. Margulies, M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, and J. Berka, "Genome sequencing in microfabricated high-density picolitre reactors," *Nature*, vol. 437, no. 7057, pp. 376–380, Sep. 2005, doi: 10.1038/nature03959.

[16] H. Li, "Minimap and miniasm: Fast mapping and *de novo* assembly for noisy long sequences," *Bioinformatics*, vol. 32, no. 14, pp. 2103–2110, Jul. 2016, doi: 10.1093/bioinformatics/btw152.

[17] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. Compress. Complex. SEQUENCES*, Salerno, Italy, 1997, pp. 21–29.

[18] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and M. A. Phillippy, "Canu: Scalable and accurate long-read assembly via adaptive *k*-mer weighting and repeat separation," *Genome Res.*, vol. 27, no. 5, pp. 722–736, 2017.

[19] G. M. Kamath, I. Shomorony, F. Xia, T. A. Courtade, and D. N. Tse, "HINGE: Long-read assembly achieves optimal repeat resolution," *Genome Res.*, vol. 27, no. 5, pp. 747–756, May 2017, doi: 10.1101/gr.216465.116.

[20] G. Myers, "Efficient local alignment discovery amongst noisy long reads," in *Proc. Int. Workshop Algorithms Bioinf.*, Sep. 2014, pp. 52–67.

[21] S. Nowoshilow, S. Schloissnig, J.-F. Fei, A. Dahl, A. W. C. Pang, M. Pippel, S. Winkler, A. R. Hastie, G. Young, J. G. Roscito, F. Falcon, D. Knapp, S. Powell, A. Cruz, H. Cao, B. Habermann, M. Hiller, E. M. Tanaka, and E. W. Myers, "The axolotl genome and the evolution of key tissue formation regulators," *Nature*, vol. 554, no. 7690, pp. 50–55, Feb. 2018, doi: 10.1038/nature25458.

[22] J. Chin and A. Khalak, "Speeding up genome assembly with sparse and hierarchical minimizer indices," in *Proc. Plant Animal Genome XXVIII Conf. (PAG)*, San Diego, CA, USA, 2020, pp. 11–15.

[23] R. Vaser and M. Šikić, "Time- and memory-efficient genome assembly with raven," *Nature Comput. Sci.*, vol. 1, no. 5, pp. 332–336, May 2021, doi: 10.1038/s43588-021-00073-4.

[24] R. Vaser and M. Sikic, "Yet another *de novo* genome assembler," in *Proc. 11th Int. Symp. Image Signal Process. Anal. (ISPA)*, Dubrovnik, Croatia, Sep. 2019, pp. 147–151.

[25] H. Li, "Minimap2: Pairwise alignment for nucleotide sequences," *Bioinformatics*, vol. 34, no. 18, pp. 3094–3100, 2018, doi: 10.1093/bioinformatics/bty191.

[26] S. Nurk, B. P. Walenz, A. Rhie, M. R. Vollger, G. A. Logsdon, R. Grothe, K. H. Miga, E. E. Eichler, A. M. Phillippy, and S. Koren, "HiCanu: Accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads," *Genome Res.*, vol. 30, no. 9, pp. 1291–1305, Sep. 2020, doi: 10.1101/gr.263566.120.

[27] H. Liu, S. Wu, A. Li, and J. Ruan, "SMARTdenovo: A *de novo* assembler using long noisy reads," *Gigabyte*, vol. 2021, pp. 1–9, Mar. 2021, doi: 10.46471/gigabyte.15.

[28] C.-S. Chin, D. H. Alexander, P. Marks, A. A. Klammer, J. Drake, C. Heiner, A. Clum, A. Copeland, J. Huddleston, E. E. Eichler, S. W. Turner, and J. Korlach, "Nonhybrid, finished microbial genome assemblies from long-read SMRT sequencing data," *Nature Methods*, vol. 10, no. 6, pp. 563–569, Jun. 2013, doi: 10.1038/nmeth.2474.

[29] E. W. Myers, "The fragment assembly string graph," *Bioinformatics*, vol. 21, no. 2, pp. ii79–ii85, Sep. 2005, doi: 10.1093/bioinformatics/bti1114.

[30] J. T. Simpson and R. Durbin, "Efficient construction of an assembly string graph using the FM-index," *Bioinformatics*, vol. 26, no. 12, pp. i367–i373, Jun. 2010, doi: 10.1093/bioinformatics/btq217.

[31] J. T. Simpson and R. Durbin, "Efficient *de novo* assembly of large genomes using compressed data structures," *Genome Res.*, vol. 22, no. 3, pp. 549–556, 2012, doi: 10.1101/gr.126953.111.

[32] M. Burrows and D. Wheeler, "A block-sorting lossless data compression algorithm," Digit. SRC Res. Rep., Tech. Rep., 1994.

[33] G. Gonnella and S. Kurtz, "Readjoiner: A fast and memory efficient string graph-based sequence assembler," *BMC Bioinf.*, vol. 13, no. 1, pp. 1–19, Dec. 2012, doi: 10.1186/1471-2105-13-82.

[34] H. Li, "Exploring single-sample SNP and INDEL calling with whole-genome *de novo* assembly," *Bioinformatics*, vol. 28, no. 14, pp. 1838–1844, Jul. 2012, doi: 10.1093/bioinformatics/bts280.

[35] C.-S. Chin, P. Peluso, F. J. Sedlazeck, M. Nattestad, G. T. Concepcion, A. Clum, C. Dunn, R. O'Malley, R. Figueroa-Balderas, A. Morales-Cruz, G. R. Cramer, M. Delledonne, C. Luo, J. R. Ecker, D. Cantu, D. R. Rank, and M. C. Schatz, "Phased diploid genome assembly with single-molecule real-time sequencing," *Nature Methods*, vol. 13, no. 12, pp. 1050–1054, Dec. 2016, doi: 10.1038/nmeth.4035.

[36] H. Cheng, G. T. Concepcion, X. Feng, H. Zhang, and H. Li, "Haplotype-resolved *de novo* assembly using phased assembly graphs with hifiasm," *Nature Methods*, vol. 18, no. 2, pp. 170–175, Feb. 2021, doi: 10.1038/s41592-020-01056-5.

[37] Y. Chen, F. Nie, S.-Q. Xie, Y.-F. Zheng, Q. Dai, T. Bray, Y.-X. Wang, J.-F. Xing, Z.-J. Huang, D.-P. Wang, L.-J. He, F. Luo, J.-X. Wang, Y.-Z. Liu, and C.-L. Xiao, "Efficient assembly of nanopore reads via highly accurate and intact error correction," *Nature Commun.*, vol. 12, no. 1, pp. 1–10, Jan. 2021, doi: 10.1038/s41467-020-20236-7.

[38] A. Golovnev, A. S. Kulikov, and I. Mihajlin, "Approximating shortest superstring problem using de bruijn graphs," in *Proc. Annu. Symp. Combinat. Pattern Matching*, Jun. 2013, pp. 120–129.

[39] D. R. Zerbino and E. Birney, "Velvet: Algorithms for *de novo* short read assembly using de Bruijn graphs," *Genome Res.*, vol. 18, no. 5, pp. 821–829, May 2008, doi: 10.1101/gr.074492.107.

[40] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, J. M. Jones, and I. Birol, "ABySS: A parallel assembler for short read sequence data," *Genome Res.*, vol. 19, no. 6, pp. 1117–1123, 2009, doi: 10.1101/gr.089532.108.

[41] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, S. Li, H. Yang, J. Wang, and J. Wang, "*De novo* assembly of human genomes with massively parallel short read sequencing," *Genome Res.*, vol. 20, no. 2, pp. 265–272, 2010, doi: 10.1101/gr.097261.109.

[42] T. Conway, J. Wazny, A. Bromage, J. Zobel, and B. Beresford-Smith, "Gossamer—A resource-efficient *de novo* assembler," *Bioinformatics*, vol. 28, no. 14, pp. 1937–1938, Jul. 2012, doi: 10.1093/bioinformatics/bts297.
[43] T. C. Conway and A. J. Bromage, "Succinct data structures for assembling large genomes," *Bioinformatics*, vol. 27, no. 4, pp. 479–486, Feb. 2011, doi: 10.1093/bioinformatics/btq697.
[44] R. Kajitani, K. Toshimoto, H. Noguchi, A. Toyoda, Y. Ogura, M. Okuno, M. Yabana, M. Harada, E. Nagayasu, H. Maruyama, Y. Kohara, A. Fujiyama, T. Hayashi, and T. Itoh, "Efficient *de novo* assembly of highly heterozygous genomes from whole-genome shotgun short reads," *Genome Res.*, vol. 24, no. 8, pp. 1384–1395, Aug. 2014, doi: 10.1101/gr.170720.113.
[45] J. Luo, J. Wang, Z. Zhang, F.-X. Wu, M. Li, and Y. Pan, "EPGA: *De novo* assembly using the distributions of reads and insert size," *Bioinformatics*, vol. 31, no. 6, pp. 825–833, Mar. 2015, doi: 10.1093/bioinformatics/btu762.
[46] J. Luo, J. Wang, W. Li, Z. Zhang, F.-X. Wu, M. Li, and Y. Pan, "EPGA2: Memory-efficient *de novo* assembler," *Bioinformatics*, vol. 31, no. 24, pp. 3988–3990, Dec. 2015, doi: 10.1093/bioinformatics/btv487.
[47] G. Rizk, D. Lavenier, and R. Chikhi, "DSK: K-mer counting with very low memory usage," *Bioinformatics*, vol. 29, no. 5, pp. 652–653, Mar. 2013, doi: 10.1093/bioinformatics/btt020.
[48] K. Mahadik, C. Wright, M. Kulkarni, S. Bagchi, and S. Chaterji, "Scalable genome assembly through parallel de Bruijn graph construction for multiple k-mers," *Sci. Rep.*, vol. 9, no. 1, Oct. 2019, Art. no. 14882, doi: 10.1038/s41598-019-51284-9.
[49] P. A. Pevzner, H. Tang, and G. Tesler, "*De novo* repeat classification and fragment assembly," *Genome Res.*, vol. 14, no. 9, pp. 1786–1796, Sep. 2004, doi: 10.1101/gr.2395204.
[50] Y. Lin, S. Nurk, and P. A. Pevzner, "What is the difference between the breakpoint graph and the de Bruijn graph?" *BMC Genomics*, vol. 15, no. S6, pp. 1–11, Oct. 2014, doi: 10.1186/1471-2164-15-S6-S6.
[51] M. J. Chaisson and P. A. Pevzner, "Short read fragment assembly of bacterial genomes," *Genome Res.*, vol. 18, no. 2, pp. 324–330, Feb. 2008, doi: 10.1101/gr.7088808.
[52] Y. Lin, J. Yuan, M. Kolmogorov, M. W. Shen, M. Chaisson, and P. A. Pevzner, "Assembly of long error-prone reads using de Bruijn graphs," *Proc. Nat. Acad. Sci. USA*, vol. 113, no. 52, pp. E8396–E8405, Dec. 2016, doi: 10.1073/pnas.1604560113.
[53] W. Kuśmirek and R. Nowak, "*De novo* assembly of bacterial genomes with repetitive DNA regions by dnaasm application," *BMC Bioinf.*, vol. 19, no. 1, pp. 1–10, Dec. 2018, doi: 10.1186/s12859-018-2281-4.
[54] T. Onodera, K. Sadakane, and T. Shibuya, "Detecting super bubbles in assembly graphs," in *Proc. Int. Workshop Algorithms Bioinf.*, Sep. 2013, pp. 338–348.
[55] J. Butler, I. MacCallum, M. Kleber, I. A. Shlyakhter, M. K. Belmonte, E. S. Lander, C. Nusbaum, and D. B. Jaffe, "ALLPATHS: *De novo* assembly of whole-genome shotgun microreads," *Genome Res.*, vol. 18, no. 5, pp. 810–820, May 2008, doi: 10.1101/gr.7337908.
[56] I. Maccallum, D. Przybylski, and S. Gnerre, "ALLPATHS 2: Small genomes assembled accurately and with high continuity from short paired reads," *Genome Biol.*, vol. 10, no. 10, pp. 1–10, Oct. 2009, doi: 10.1186/gb-2009-10-10-r103.
[57] S. Gnerre, I. MacCallum, D. Przybylski, F. J. Ribeiro, J. N. Burton, B. J. Walker, T. Sharpe, G. Hall, T. P. Shea, S. Sykes, A. M. Berlin, D. Aird, M. Costello, R. Daza, L. Williams, R. Nicol, A. Gnirke, C. Nusbaum, E. S. Lander, and D. B. Jaffe, "High-quality draft assemblies of mammalian genomes from massively parallel sequence data," *Proc. Nat. Acad. Sci. USA*, vol. 108, no. 4, pp. 1513–1518, Jan. 2011, doi: 10.1073/pnas.1017351108.
[58] C. Ye, Z. S. Ma, C. H. Cannon, M. Pop, and D. W. Yu, "Exploiting sparseness in *de novo* genome assembly," *BMC Bioinf.*, vol. 13, no. S6, pp. 1–8, Dec. 2012, doi: 10.1186/1471-2105-13-S6-S1.
[59] R. Luo, B. Liu, Y. Xie, Z. Li, W. Huang, J. Yuan, and G. He, "SOAPdenovo2: An empirically improved memory-efficient short-read *de novo* assembler," *GigaScience*, vol. 1, no. 1, pp. 1–6, Dec. 2012, doi: 10.1186/2047-217X-1-18.
[60] Y. Peng, H. C. Leung, S. M. Yiu, and F. Y. L. Chin, "IDBA—A practical iterative de Bruijn graph *de novo* assembler," in *Proc. Annu. Int. Conf. Res. Comput. Mol. Biol.*, Apr. 2010, pp. 426–440.
[61] Y. Peng, H. C. M. Leung, S. M. Yiu, and F. Y. L. Chin, "IDBA-UD: A *de novo* assembler for single-cell and metagenomic sequencing data with highly uneven depth," *Bioinformatics*, vol. 28, no. 11, pp. 1420–1428, Jun. 2012, doi: 10.1093/bioinformatics/bts174.
[62] A. Souvorov, R. Agarwala, and D. J. Lipman, "SKESA: Strategic k-mer extension for scrupulous assemblies," *Genome Biol.*, vol. 19, no. 1, pp. 1–13, Dec. 2018, doi: 10.1186/s13059-018-1540-z.
[63] P. Medvedev, S. Pham, M. Chaisson, G. Tesler, and P. Pevzner, "Paired de Bruijn graphs: A novel approach for incorporating mate pair information into genome assemblers," *J. Comput. Biol.*, vol. 18, no. 11, pp. 1625–1634, Nov. 2011, doi: 10.1089/cmb.2011.0151.
[64] A. Bankevich, S. Nurk, D. Antipov, A. A. Gurevich, M. Dvorkin, A. S. Kulikov, V. M. Lesin, S. I. Nikolenko, S. Pham, A. D. Prjibelski, A. V. Pyshkin, A. V. Sirotkin, N. Vyahhi, G. Tesler, M. A. Alekseyev, and P. A. Pevzner, "SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing," *J. Comput. Biol.*, vol. 19, no. 5, pp. 455–477, May 2012, doi: 10.1089/cmb.2012.0021.
[65] Z. Iqbal, M. Caccamo, I. Turner, P. Flicek, and G. McVean, "*De novo* assembly and genotyping of variants using colored de Bruijn graphs," *Nature Genet.*, vol. 44, no. 2, pp. 226–232, Feb. 2012, doi: 10.1038/ng.1028.
[66] M. D. Muggli, A. Bowe, N. R. Noyes, P. S. Morley, K. E. Belk, R. Raymond, T. Gagie, S. J. Puglisi, and C. Boucher, "Succinct colored de Bruijn graphs," *Bioinformatics*, vol. 33, no. 20, pp. 3181–3187, Oct. 2017, doi: 10.1093/bioinformatics/btx067.
[67] J. Pell, A. Hintze, R. Canino-Koning, A. Howe, J. M. Tiedje, and C. T. Brown, "Scaling metagenome sequence assembly with probabilistic de Bruijn graphs," *Proc. Nat. Acad. Sci. USA*, vol. 109, no. 33, pp. 13272–13277, Aug. 2012, doi: 10.1073/pnas.1121464109.
[68] A. Kirsch and M. Mitzenmacher, "Less hashing, same performance: Building a better Bloom filter," in *Proc. Eur. Symp. Algorithms*, Sep. 2006, pp. 456–467.
[69] S. D. Jackman, B. P. Vandervalk, and H. Mohamadi, "ABySS 2.0: Resource-efficient assembly of large genomes using a Bloom filter," *Genome Res.*, vol. 27, no. 5, pp. 768–777, 2017, doi: 10.1101/gr.214346.116.
[70] R. Chikhi and G. Rizk, "Space-efficient and exact de Bruijn graph representation based on a Bloom filter," *Algorithms Mol. Biol.*, vol. 8, no. 1, pp. 1–9, Jan. 2013, doi: 10.1186/1748-7188-8-22.
[71] M. Kolmogorov, J. Yuan, Y. Lin, and P. A. Pevzner, "Assembly of long, error-prone reads using repeat graphs," *Nature Biotechnol.*, vol. 37, no. 5, pp. 540–546, May 2019, doi: 10.1038/s41587-019-0072-8.
[72] A. Bankevich and P. Pevzner, "MosaicFlye: Resolving long mosaic repeats using long reads," in *Proc. Int. Conf. Res. Comput. Mol. Biol.*, May 2020, pp. 226–228.
[73] M. Kolmogorov, D. M. Bickhart, B. Behsaz, A. Gurevich, M. Rayko, S. B. Shin, K. Kuhn, J. Yuan, E. Polevikov, T. P. L. Smith, and P. A. Pevzner, "MetaFlye: Scalable long-read metagenome assembly using repeat graphs," *Nature Methods*, vol. 17, no. 11, pp. 1103–1110, Nov. 2020, doi: 10.1038/s41592-020-00971-x.
[74] A. V. Bzikadze and P. A. Pevzner, "Automated assembly of centromeres from ultra-long error-prone reads," *Nature Biotechnol.*, vol. 38, no. 11, pp. 1309–1316, Nov. 2020, doi: 10.1038/s41587-020-0582-4.
[75] K. Shafin, T. Pesout, R. Lorig-Roach, and M. Haukness, "Nanopore sequencing and the Shasta toolkit enable efficient *de novo* assembly of eleven human genomes," *Nature Biotechnol.*, vol. 38, pp. 1044–1053, May 2020, doi: 10.1038/s41587-020-0503-6.
[76] J. Ruan and H. Li, "Fast and accurate long-read assembly with wtdbg2," *Nature Methods*, vol. 17, no. 2, pp. 155–158, Feb. 2020, doi: 10.1038/s41592-019-0669-3.
[77] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, no. 1, pp. 195–197, Mar. 1981, doi: 10.1016/0022-2836(81)90087-5.
[78] B. Ekim, B. Berger, and R. Chikhi, "Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer," *Cell Syst.*, vol. 12, no. 10, pp. 958–968, Oct. 2021, doi: 10.1016/j.cels.2021.08.009.
[79] W. Kuśmirek, W. Franus, and R. Nowak, "Linking *de novo* assembly results with long DNA reads using the dnaasm-link application," *BioMed Res. Int.*, vol. 2019, pp. 1–10, Apr. 2019, doi: 10.1155/2019/7847064.
[80] J. A. Chapman, I. Ho, S. Sunkara, S. Luo, G. P. Schroth, and D. S. Rokhsar, "Meraculous: *De novo* genome assembly with short paired-end reads," *PLoS ONE*, vol. 6, no. 8, Aug. 2011, Art. no. e23501, doi: 10.1371/journal.pone.0023501.
[81] A. V. Zimin, G. Marçais, D. Puiu, M. Roberts, S. L. Salzberg, and J. A. Yorke, "The MaSuRCA genome assembler," *Bioinformatics*, vol. 29, no. 21, pp. 2669–2677, Nov. 2013, doi: 10.1093/bioinformatics/btt476.

[82] C. Ye, C. M. Hill, S. Wu, J. Ruan, and Z. Ma, "DBG2OLC: Efficient assembly of large genomes using long erroneous reads of the third generation sequencing technologies," *Sci. Rep.*, vol. 6, no. 1, pp. 1–9, Aug. 2016, doi: 10.1038/srep31900.

[83] E. Haghshenas, H. Asghari, J. Stoye, C. Chauve, and F. Hach, "HASLR: Fast hybrid assembly of long reads," *iScience*, vol. 23, no. 8, Aug. 2020, Art. no. 101389, doi: 10.1016/j.isci.2020.101389.

[84] D. Antipov, A. Korobeynikov, J. S. McLean, and P. A. Pevzner, "HybridSPAdes: An algorithm for hybrid assembly of short and long reads," *Bioinformatics*, vol. 32, no. 7, pp. 1009–1015, Apr. 2016, doi: 10.1093/bioinformatics/btv688.

[85] A. Di Genova, E. Buena-Atienza, S. Ossowski, and M.-F. Sagot, "Efficient hybrid *de novo* assembly of human genomes with WENGAN," *Nature Biotechnol.*, vol. 39, no. 4, pp. 422–430, Apr. 2021, doi: 10.1038/s41587-020-00747-w.

[86] R. R. Wick, L. M. Judd, C. L. Gorrie, and K. E. Holt, "Unicycler: Resolving bacterial genome assemblies from short and long sequencing reads," *PLOS Comput. Biol.*, vol. 13, no. 6, Jun. 2017, Art. no. e1005595, doi: 10.1371/journal.pcbi.1005595.

KIMIA BEHIZADI received the B.Sc. and M.Sc. degrees in mathematics from Alzahra University, in 2012. She is currently pursuing the Ph.D. degree with Tarbiat Modares University. Her current research interests include graph data structures and algorithms, comparative genomic, phylogenetic trees, and genome assembly.

NAFISEH JAFARZADEH received the B.Sc. degree in mathematics from Alzahra University, Tehran, Iran, in 2010, and the M.Sc. and Ph.D. degrees in biomathematics from Tarbiat Modares University, Tehran, in 2012 and 2018, respectively. From 2016 to 2017, she was a Visiting Scholar at the Department of Computational Biology, University of Southern California, Los Angeles, CA, USA. From 2018 to 2019, she worked as a Postdoctoral Research Fellow with the Iran National Science Foundation. In 2019, she was a Visiting Researcher at the Department of Bioinformatics, University of Göttingen, Germany. She is currently a Postdoctoral Research Fellow at Tarbiat Modares University. Her main research interests include graph data structures and algorithms, comparative genomic, and genome assembly.

ALI IRANMANESH received the B.Sc. degree from Shiraz University and the Ph.D. degree from Tarbiat Modares University, Tehran, Iran, in 1995. Since 1995, he has been employed with Tarbiat Modares University, where he is currently a Full Professor of mathematics. From 2014 to 2020, he worked as the President of the Iranian Nanotechnology Society. In 2018, he was a Visiting Scholar at the University of California at Berkeley, Berkeley, CA, USA. He supervised 35 Ph.D. students and 90 M.Sc. students in mathematics and its applications. His research interests include algebraic graph theory, combinatorics, computational graph theory, and biomathematics.