Graph theoretical Strategies in De Novo Assembling

KIMIA BEHIZADI, NAFISEH JAFARZADEH, AND ALI IRANMANESH
Department of Mathematics, Faculty of Mathematical Sciences, Tarbiat Modares University, Tehran, Iran
Corresponding author: Ali Iranmanesh (e-mail: Iranmanesh@modares.ac.ir).

This work was supported by 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 manuscript, 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. 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 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]. 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 [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].

OVERLAP GRAPH-BASED ASSEMBLERS

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. 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. Newbler [15] is another assembler designed for NGS data which adapted the overlap graph. 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. Canu [18] is derived from the Celera which is specialized in the assembly of TGS long reads. HINGE [19] builds an overlap graph to obtain pairwise alignments between all reads by using DALIGNER [20]. Marvel [21] creates the best overlap graph and obtains contig paths for long-read data. Peregrine [22] scans all the reads to construct a hash map to record the read locations and uses sparse hierarchical minimizers to index reads. Raven [23] is the upgrade version of Ra [24] which builds an overlap graph using pairwise overlaps generated by Minimap2 [25]. HiCanu [26] is a modification of the Canu designed for PacBio highly accurate long reads. SMARTde-novo [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].

STRING GRAPH-BASED ASSEMBLERS

Edena [13] is the first string graph-based assembler designed for early short-read sequencing data. Edena computes overlaps between reads using suffix array and performs transitive edge removal then indexes all sequence reads in a prefix tree. 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. Fermi [34] is inspired by SGA and uses FMD-index [33] to represent both DNA strands inside a unique structure. RU [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. FALCON [35] is a haplotype-aware assembler for large genome assembly which builds a string graph by using DALIGNER [20]. FALCON-Unzip [35] takes the contigs from FALCON and phases the reads based on heterozygous single nucleotide polymorphisms identified in the initial assembly. 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. 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 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.

DE-BRUIJN GRAPH-BASED ASSEMBLERS

**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. **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. **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. **SOAPdenovo** [41] uses de-Bruijn graph data structure and simplifies the graph by merging unambiguously connected vertices into one. **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. **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. **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. **ScalaDBG** [48] is a scalable genome assembler through parallel de-Bruijn graph construction for multiple k-mers. **ScalaDBG** 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 \(\Gamma\) 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 \(\Gamma\), the A-Bruijn graph is defined to work with imperfect repeats and is equivalent to the de-Bruijn graph in the special case that \(\Gamma\) 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.

A-BRUIJN GRAPH-BASED ASSEMBLERS

**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 allow mismatches, rather than the exact k-mers in de-Bruijn assembly. **EULER-SR** [51] is a modified version of EULER+ assembler which presents a memory-efficient DBG-based approach. **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. **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.

UNIPATH GRAPH-BASED ASSEMBLERS
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. ALLPATHS2 [56] is a modified version of ALLPATHS where generates assemblies with long, accurate contigs and scaffolds. 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.

SPARSE DE-BRUIJN GRAPH-BASED ASSEMBLERS
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. 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 $k_1 = k_{\text{min}} < k_2 < \ldots < k_{\text{max}} = k_n$. Let DBG$(R, k)$ be the de-Bruijn graph of $k$-mer size form a set of reads $R$ and consider $G(R, k_1) = \text{DBG}(R, k_1)$ and $C(k_1)$ is the set of contigs from $G(R, k_1)$, define $G(R, k_{i+1}) = \text{DBG}(R \cup C(k_i), k_{i+1})$ and

![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.](image)

![FIGURE 4. The workflow of construction the Iterative de-Bruijn graph from a set of reads.](image)
finally \( G(R, k_n) \) is an Iterative DBG. A schematic process of this graph is shown in Fig. 4.

**Iterative De-Bruijn Graph-Based Assemblers**

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

**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, is a pair of \( (k \times 1) \)-mers with a distance of \( d \) between their start positions and a \( k \)-bimer. 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 \( u = \text{prefix}(a|b) \), \( v = \text{suffix}(a|b) \) 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.

**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 \(( (k - 1) \text{-mer}) \) and edge \((k \text{-mer}) \) is associated a list of colors corresponding to the samples in which the vertex or edge label exists [66]. CDBG 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.

**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. 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.
PROBABILISTIC DE-BRUIJN GRAPH-BASED ASSEMBLERS

Minia [70] is a short-read assembler based on probabilistic DBG which is implicitly encoded as a Bloom filter. 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 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.

REPEAT GRAPH-BASED ASSEMBLERS

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. 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.

MARKER GRAPH-BASED 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.

FUZZY BRUIJN GRAPH-BASED 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 $\Sigma^l$ 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.
MINIMIZER-SPACE DE-BRUIJN GRAPH-BASED 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 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.

HYBRID ASSEMBLERS

Meraculous [80] is a hybrid assembler which follows the first hybrid assembly strategy. Meraculous first constructs and traverses a simplified de-Brujin 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. MaSuRCA [81] (Super-Read Celera Assembler) 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-Brujin graph. Once the super-reads are created, they, along with the mate pairs that connect them, collectively replace the de-Brujin graph. Incorporating pairwise information is performed using the OLC assembly. DBG2OLC [82] is a hybrid assembler which follows the third hybrid assembly strategy. The algorithm starts with linear unambiguous regions of a de-Brujin graph and ends up with linear unambiguous regions in an overlap graph.

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. HASLR 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. 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. WENGAN starts by building short-read contigs using a de-Brujin 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. Unicycler [86] is also a hybrid assembler which uses the fourth hybrid assembly 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-Brujin 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-Brujin 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
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        | [39]      |
| ALLPATHS  | DBG-based | UniPath graph        | Illumina             | [55]      |
| ALLPATHS 2 | DBG-based | UniPath graph        | Illumina             | [56]      |
| ABYSs     | DBG-based | de-Bruijn graph      | Illumina, SOLiD      | [40]      |
| SOAPdenovo| DBG-based | de-Bruijn graph      | Illumina, SOLiD      | [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]      |
| EPGAD/EPGA 2 | DBG-based | de-Bruijn graph      | Illumina             | [45, 46] |
| ABysns2  | DBG-based | Probabilistic de-Bruijn graph | Illumina     | [69]      |
| SKEA      | DBG-based | de-Bruijn graph      | Illumina             | [62]      |
| dnaasm   | DBG-based | A-Bruijn graph       | Illumina             | [53]      |
| SCALADBG  | DBG-based | de-Bruijn graph      | Illumina             | [48]      |

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]      |
| Wdgb2     | 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]      |
| Hiftasm   | OLC-based | String graph         | PacBio               | [36]      |
| NECN      | 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]      |

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-Brujin 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 Behzadi and Natieh Jafarzadeh contributed equally to this work and share the first authorship. The authors would like to thank the referees for their valuable comments.

REFERENCES
[1] F. Sanger, S. Nicklen, A. R. Coulson, “DNA sequencing with chain terminating inhibitors,” Proc. Natl. Acad. Sci., vol. 74, no. 12, pp. 5463-5467, Dec. 1977. DOI: 10.1073/pnas.74.12.5463.
[2] A. M. Maxam, W. A. Gilbert, “A new method for sequencing DNA,” Proc. Natl. Acad. Sci., vol. 74, no. 2, pp. 560-564, Feb. 1977. DOI: 10.1073/pnas.74.2.560.
[3] N. Jafarzadeh, A. Irannamesh, “On Graph-Based Data Structures to Multiple Genome Alignment,” MATCH Commun. Math. Comput. Chem., vol. 83, no. 1, pp. 33-69, Feb. 2020.
[4] S. El-Metwally, T. Hamza, M. Zakaria, 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, M. Pop, “Modern technologies and algorithms for scaffolding assembled genomes,” PLoS computational biology, vol. 5, no. 6, Jun. 2019. Art.no. e1006994 DOI: 10.1371/journal.pcbi.1006994.
[6] J. Ghurye, A. Rhee, B. P. Walenz et al., “Integrating Hi-C links with assembly graphs for chromosome-scale assembly,” PLoS computational biology, vol. 15, no. 8, Aug. 2019. Art.no. 1007273 DOI: 10.1371/journal.pcbi.1007273.
[7] R. Staden, “A new computer method for the storage and manipulation of DNA gel reading data,” Nucleic Acids Research, vol. 8, no. 16, pp. 1349-1364, 1980. DOI: 10.1093/nar/8.16.1349.
[8] E. W. Myers, G. G. Sutton, A. L. Delcher et al., “A whole-genome assembly of Drosophila,” Science, vol. 287, pp. 5461, 2196-2204, Mar. 2000. DOI: 10.1126/science.287.5461.2196.
[9] J. Venter, D. Adams et al., “The sequence of the human genome,” Science, vol. 287, no. 5461, pp. 1757-1784, 2000. DOI: 10.1126/science.287.5461.1757.
[10] L. Sanger, S. Nicklen, A. R. Coulson, “DNA sequencing with chain termination,” Nature, vol. 274, no. 5677, pp. 798-805, Dec. 1978. DOI: 10.1038/274798a0.
[11] W. Li, H. Durbin, “Fast and accurate short reads alignment,” Genome research, vol. 18, no. 3, pp. 351-357, Mar. 2008. DOI: 10.1101/gr.070917.107.
[12] P. A. Pevzner, A. L. Delcher, S. Koren et al., “Aggressive assembly of pyrosequencing reads with mates,” Bioinformatics, vol. 24, no. 24, pp. 2917-2923, Dec. 2008. DOI: 10.1093/bioinformatics/btn586.
[13] J. Miller, A. L. Delcher, S. Koren et al., “Aggressive assembly of pyrosequencing reads with mates,” Bioinformatics, vol. 24, no. 24, pp. 2917-2923, Dec. 2008. DOI: 10.1093/bioinformatics/btn586.
[14] D. R. Zerbino, E. Birney, “Velvet: algorithms for de novo short read assembly using de-Brujin graphs,” Genome research, vol. 18, no. 5, pp. 821-829, May 2008. DOI: 10.1101/gr.074492.107.
[84] D. Antipov, A. Korobeynikov, J. S. McLean et al., “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 et al., “Efficient hybrid de novo assembly of human genomes with WENGAN,” *Nature biotechnology*, 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 et al., “Unicycler: resolving bacterial genome assemblies from short and long sequencing reads,” *PLoS computational biology*, 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, and now she is a Ph.D. student at Tarbiat Modares University. Her current research interests include graph data structures and algorithms, comparative genomic, phylogenetic trees and genome assembly.

NAFISEH JAFARZADEH completed the B.Sc. in Mathematics at Alzahra University, Tehran, Iran in 2010, and the M.Sc. in 2012 and Ph.D. in Biomathematics at Tarbiat Modares University, Tehran, Iran in 2018. From 2016 to 2017, she was a Visiting Scholar at the Department of Computational Biology, University of Southern California, Los Angeles, USA. From 2018 to 2019, she worked as a Postdoctoral Research Fellow for 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, Tehran, Iran. Her main research interests are graph data structures and algorithms, comparative genomic and genome assembly.

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