Hindawi
Journal of Healthcare Engineering
Volume 2021, Article ID 6676194, 11 pages
https://doi.org/10.1155/2021/6676194

Research Article

Gene Sequence Assembly Algorithm Model Based on the DBG Strategy and Its Application

Haihe Shi and Gang Wu

School of Computer and Information Engineering, Jiangxi Normal University, Nanchang, China

Correspondence should be addressed to Haihe Shi; haiheshi@jxnu.edu.cn

Received 24 November 2020; Revised 5 January 2021; Accepted 16 January 2021; Published 27 January 2021

Academic Editor: Y.-h. Yao

With the continuous development of sequencing technology, the amount of bioinformatics data has increased geometrically, and the massive amount of bioinformatics data puts forward more stringent requirements for sequence assembly problems. The sequence assembly algorithm based on DBG (De Bruijn graph) strategy is a key algorithm in bioinformatics, which is widely used in the domain of gene sequence assembly. Current research on the domain of sequence assembly always focuses on optimization of specific steps to a specific algorithm and lack of research on domain-level high-abstract algorithm frameworks. To some extent, it leads to the redundancy of the sequence assembly algorithm, and some problems may be caused by the artificial selection algorithm. This paper analyzes the domain of DBGSA and establishes a feature model of this domain. Based on the production programming method, the DBGSA algorithm component is interactively designed. With the support of the PAR platform, the DBGSA algorithm component library is formally implemented, and furthermore, the DBGSA component library is used to assemble the specific algorithm. This research adds domain-level research to the domain of sequence assembly and implements the DBGSA component library, which can assemble specific sequence assembly algorithms, ensuring the efficiency of algorithm development and the reliability of assembly generation algorithms. At the same time, it also provides a valuable reference for solving problems in the domain of sequence assembly.

1. Introduction

With the rapid development of the second-generation high-throughput sequencing technology and the third-generation single-molecule sequencing technology, scientists have accelerated the analysis of the genome and the information it carries. And the cost of gene sequencing has been continuously reduced with the development of high-throughput sequencing technology and single-molecule sequencing technology. The accumulation of multiple omics data including genomics and transcriptomics has provided massive data resources for bioinformatics, but it also brought new challenges.

Sequence assembly algorithm [1] is a key algorithm in bioinformatics. There are a lot of sequence fragment data obtained through the second-generation high-throughput sequencing technology, but the sequence obtained by sequencing is too short, resulting in insufficient information contained in the sequence, and is unable to provide sufficient information for subsequent research work. Therefore, the sequence fragments obtained by sequencing must be assembled to obtain sufficiently long sequence fragments. The process of assembling the short sequence fragments obtained from the initial measurement is called sequence assembly. At present, the commonly used methods include assembly algorithm based on OLC [2–4], assembly algorithm based on greedy strategy [5, 6], and assembly algorithm based on DBG strategy [7–9]. This paper mainly focuses on the research in the domain of DBG Strategy-based Assembly Algorithm (DBGSA).

The assembly algorithm based on the DBG strategy was originally introduced by R. M. ldury and M. S. Waterman in

1995. The first assembly software Euler based on the DBG algorithm is published in 2001 [10, 11]. In 2008, the velvet algorithm [12] was jointly proposed by Zerbino and Birney. In 2009, the ABySS algorithm [13] was jointly proposed by Simpson et al. SOAPdenovo [14] is an assembler designed by BGI in 2010 to solve the difficulty of assembly large-scale repeated short sequences generated by NGS parallel DNA sequencing technology from scratch. Aiming at the problem that single-cell and metagenomic sequencing technologies are difficult to sequence the uneven sequencing depth of different regions to the genome between different species or within the same species, a solution to the problem of de novo assembly algorithm, IDBA-UD, is proposed [15]. metaSPAdes [16] assemble single-cell and highly polymorphic diploid genomes by fusing methods in a series of proven SPAdes tools.

The sequence assembly algorithm based on the DBG strategy mainly includes three steps:

(1) The first step is to build a De Bruijn graph. First, all the reads involved in the assembly are divided into sequence fragments of length k, called k-mers. And the adjacent k-mers in the same read have $k-1$ base overlap. Then, use k-mer as the node of the graph to build a DBG graph

(2) The second step is the contigs construction step. Simplify the DBG diagram by removing error structures such as tips, bubbles, and repeats caused by sequencing errors. After the simplification is completed, the contigs are found by traversing the DBG graph to find the Euler path that each edge in the graph passes only once

(3) The third step is the scaffolds step. Align the contigs obtained in the second step with the original sequencing reads, fill in the gaps between unconnected contigs according to the alignment information, and read position information to obtain the final complete DNA sequence

Through an in-depth analysis of the DBGSA domain, combined with domain engineering methods, generative programming methods and abstract modeling techniques designed and implemented an abstract generic algorithm component library based on the DBG strategy assembly algorithm to improve the reliability and reusability of algorithm components in this domain. First, according to the method of domain engineering, the domain analysis of DBGSA is carried out, and the general features and variable features and the dependencies between them are extracted to establish the domain feature model of DBGSA. Then, the algorithm component was designed according to the feature model, and the interaction model of the component was established [17, 18]. Furthermore, with the support of the new high-reliability software development platform PAR, the generic abstract programming language Apla is used to formally realize the components, forming a high abstract DBGSA component library based on Apla. Finally, the highly reliable DBGSA algorithm is generated through the component assembly.

2. Related Technologies and Methods

2.1. Experiment Data. We obtained the genome sequence data of an African male individual from the NCBI database (HapMap DNA identifier NA18507), which was generated by the Illumina Genome Sequencer.

2.2. Generative Programming. Generative Programming [19] (GP) can be regarded as a software engineering method of product line engineering. Essentially, it is to design and implement components so that they can be applied to the general structure of the product line and then produce software products in an automated form. The process of GP has two main steps: first, it is necessary to transform the current development method for only one software system into the development of this software system family, analyze the software system family, find out the commonalities, and develop the correct common components; then, we need to design and realize a kind of generator, used to realize the assembly automation of the components. The key step of GP is the design of the production domain model, which includes a problem space, a solution space, and the configuration knowledge mapping relationship between the problem space and the solution space.

Use the example of buying a computer to describe the production domain model. The buyer can be regarded as the problem space. The computer he needs to buy can be expressed by the following features, such as thin and light, high performance, high-definition display, and other features. When these features are passed to the computer manufacturer after the computer merchant, it will be described as more specific features. For example, the computer is a laptop, the processor needs to be i7 or more, and the screen size is 19 inches or more. These feature descriptions belong to the description of the solution space.

The solution space represents design. It mainly includes the components that need to be realized and the combination relationship between them. When designing, it needs to be considered to maximize the composability between the components and minimize redundancy. The mapping relationship of configuration knowledge specifies illegal feature combinations, construction rules (combinations of realization components converted from certain combinations of features), and optimization rules (a certain combination of realization components may be better than other combinations of realization components).

The problem space represents the requirement. When requesting components from the component library, only the necessary features should be specified, and too many detailed features should not be specified. If too many detailed features are specified, the redundancy in the solution space will be too large. This is an important principle of design to problem space.

The mapping relationship of configuration knowledge is mainly used to separate the problem space and the solution space. An important principle of the separation of problem space and solution space is to make independent evolution in the two spaces in an independent manner. When adding

new components to the solution space or improving existing components, they are only required to cover the functions previously required in the problem space, so there is no need to modify the client code.

2.3. Feature Modeling. Domain engineering [20] is the collection, organization, and preservation of resources developed in a reusable form when constructing a system or some parts of a system in a specific domain. Then, when constructing a new system, provide an adequate method to reuse the saved resources. Domain engineering has three parts, namely, domain analysis, domain design, and domain realization.

Domain analysis mainly analyzes many systems in the domain, finds out the common and variable features of these systems, and then classifies them. Its purpose is to select and define the domain to be analyzed and solved and to collect relevant domain information and integrate it into a consistent domain model.

Domain design refers to the development of an architecture for the system family in the domain.

Domain realization refers to the entire process of implementing architecture and components using appropriate technologies.

In the process of domain analysis, there is a very important concept feature modeling [21]. Feature modeling is not only an important contribution of domain engineering to software engineering but also an indispensable part of generative programming. Feature modeling includes the following two steps. First, determine the content of the research domain and the boundary of the domain. Then, analyze the common features and different features of the members to the domain, and determine the dependence of the features. The establishment of feature models can effectively avoid the loss of some common and different features in the domain analysis process. Zhang and Mei proposed a feature-oriented domain modeling method (FODM) in 2003 [22]. Considering the features of the domain's services, functions, and behavioral features, through analyzing the service, function, behavioral features, domain terminology, commonality and variability, interaction process, and quality requirements, the feature model is finally obtained through continuous retrospective refinement.

2.4. PAR Method. PAR [23–27] (Partition-and-Recur) is a formal development method based on partition and recursion. It has customized an algorithm design language Radl (Recurrence-based Algorithm Design Language) and abstract programming language Apla (Abstract Programming Language). It also includes a unified algorithm design and proof method and a series of generation systems (PAR platform). Apla language can directly use abstract data types and abstract procedures to write programs. It has the advantages of concise and rigorous mathematical language, and the high abstractness of the language itself is very suitable for describing abstract algorithm programs. Apla mainly supports the mechanisms of generic programs: type parameterization, subprogram parameterization, and user-defined generic ADT. The PAR method development process is shown in Figure 1. The advantages of the PAR method are as follows:

(1) Apla introduces the keyword sometype to define type variables, type parameters, parameter return value types of procedural functions, and basic types of combined data types and uses types as parameters to realize the genericization of programs

(2) Apla subprogram parameterization includes process parameterization and function parameterization. The keywords proc and func are used in the subprogram to declare the process as a parameter and function as a parameter, and the process or function is used as the formal parameter list of the subprogram

(3) User-defined generic ADT: there are predefined abstract data types (ADT) in Apla. In addition, users can use custom ADT to make the Apla language more flexible and program description functions more powerful. ADT custom operations include the definition of ADT and the realization of ADT

ADT definition includes operation name, operation type, and operation parameters. The ADT implementation part gives the specific implementation methods of these operations; Apla defines keywords such as define, ADT, enddef, implement, and endimp to describe the name of the custom ADT and its corresponding operation implementation. In addition, the PAR platform also supports the conversion of Apla into executable high-level programming languages such as C++ and Java, which provides good support for the rapid and reliable development of components.

3. DBGSA Domain Modeling

With the development of time, many sequence assembly algorithms based on the DBG strategy have been derived. These sequence assembly algorithms based on the DBG strategy are combined to form the domain of DBGSA. The main content of this chapter is to analyze the domain of DBGSA and establish a feature model of this domain. Then, we abstract the features in the model into components and use Apla to implement all components.

3.1. Domain Analysis

3.1.1. Velvet Algorithm. Velvet algorithm is a de novo assembly algorithm proposed by Zerbino and Birney [12] that runs under Unix. It is mainly used to assemble sequences with a length of 25 to 500 bp. The velvet algorithm is based on the de Bruijn graph strategy. It runs various error correction steps after building the graph, which can effectively simplify the de Bruijn graph to eliminate errors and solve the problem of duplication. The velvet algorithm is verified on simulation data and real data, and the maximum N50 can reach 50k. In recent years, there have been many applications and research studies on the velvet algorithm [28]. The main steps of the velvet algorithm are as follows:

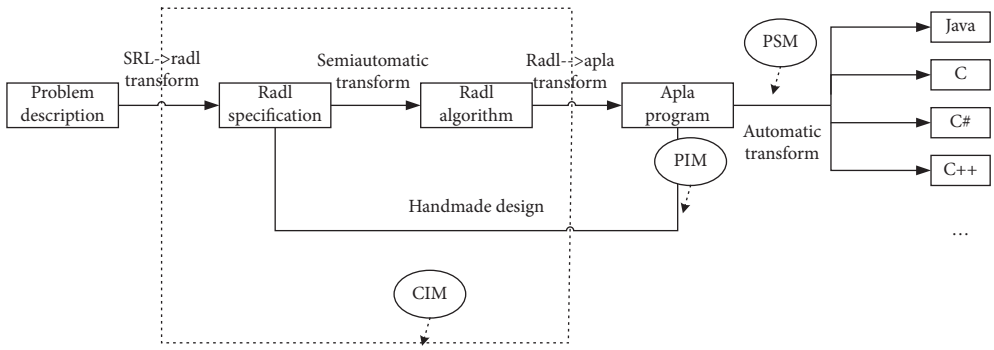

Figure 1: PAR method development process.

(1) Build de Bruijn graph: put all the original data into the hash table to build the index. Then, calculate the K-mer, and use the k-mer to build the de Bruijn graph.

(2) Simplify the graph: when node A has an output edge pointing to B and node B has only one input edge, then the two nodes A and B are merged into one node.

(3) Error correction: use the difference between the expected coverage of the gene sequence and the random error to correct the error.

(4) Remove tips: if a tip path is less than 2k, this path is removed as an isolated point.

(5) Remove bubbles: use the tour bus algorithm to search for bubbles, and then merge the bubble paths.

(6) In the contigs stage, find a path that has and passes through each edge only once from the simplified de Bruijn graph. This path is contig.

(7) In the scaffold stage, all contigs are assembled into the final scaffold sequence, and then output.

3.1.2. ABySS Algorithm. The ABySS algorithm was originally developed for the de novo assembly of genomes, especially for large genomes. The advantage of the ABySS assembly algorithm is that it can perform parallel operations and run multiple assembly tasks at the same time, so it may process a much larger genome than velvet. It is currently the only gene sequence assembly algorithm that can be assembled in parallel. In recent years, there have been many applications and researches on the ABySS algorithm [29–31]. The main steps of the ABySS algorithm are as follows:

(1) Build the graph: first, it will be transferred into the distributed system to calculate all the k-mers and read from the sequence to save their adjacency. Finally, the k-mer is placed in the distributed de Bruijn graph.

(2) Remove tips: if a tip path is less than 2k, this path is removed as an isolated point.

(3) Remove bubbles: use the bubble removal algorithm to search for bubbles, and then merge the bubble paths.

(4) In the contigs stage, find a path that has and passes through each edge only once from the simplified de Bruijn graph. This path is contig.

(5) In the scaffold stage, all contigs are assembled into the final scaffold sequence, and then output.

3.1.3. SOAPdenovo Algorithm. SOAPdenovo is a high-throughput sequencing de novo assembly software developed by BGI. It uses a new type of short-read assembly method that can construct a de novo assembly sketch of the human genome. SOAPdenovo is mainly used for de novo assembly of large animal and plant genomes; of course, it also performs well for the assembly of bacterial and fungal genomes. This algorithm is specifically used to assemble short-read sequencing data generated by Illumina. SOAPdenovo provides a new way to construct reference sequences. It also provides a tool for efficient and accurate analysis of unknown genomes. In recent years, there have been many applications and research studies on the SOAPdenovo algorithm [32]. The main steps of SOAPdenovo algorithm are as follows:

(1) Error correction: by using the frequency information of k-mers, k-mers with a frequency less than 3 will be removed.

(2) Build the graph: for the de Bruijn graph, each node is a k-mer, and two nodes overlapping by $k-1$ bases will be connected into an edge.

(3) Remove tips: if a tip path is less than 2k, this path is removed as an isolated point.

(4) Remove repeats: if a node has N incoming edges, there are paths that support N outgoing edges, and there is no conflict between the paths; then, remove the node and split into N parallel paths.

(5) Remove bubbles: use the Dijkstra algorithm to search for bubbles, and then merge the bubble paths.

(6) In the contigs stage, find a path that has and passes through each edge only once from the simplified de Bruijn graph. This path is contig.

(7) In the scaffold stage, all contigs are assembled into the final scaffold sequence, and then output.

The general flowchart of the three algorithms is shown in Figure 2.

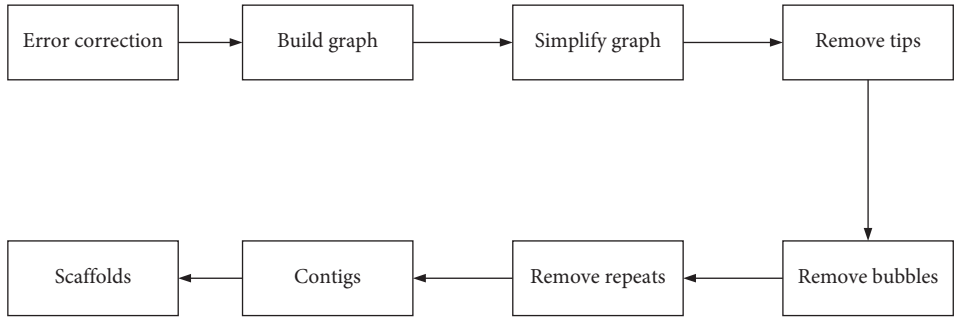

Figure 2: General flowchart of three algorithms.

Based on the modeling method of the FODM domain, this paper combines DBGSA domain service, function, and behavior features to construct the DBGSA domain model. The core service in this domain is based on the sequence assembly of DBG. Through the analysis of the sequence assembly steps based on the DBG strategy, the assembly operation service can be further divided into functions such as error correction, build graph, simplified graph, remove operation, contigs, and scaffolds. Among them, the remove operation can be divided into three functions: remove tips, remove bubbles, and remove tiny repeats. In the assembly operation service, error correction, build graph, remove tips, remove bubbles, contigs, and scaffolds are mandatory functions, and simplify graph and remove repeats are optional functions. For the remove bubbles operation, the remove bubbles mode is its behavioral characteristic. This dimension has three values, namely, Tour_Bus, R_B, and Dijkstra. Based on the above analysis, a feature model is constructed for this domain, as shown in Figure 3.

3.2. Domain Design

3.2.1. Component Design. Based on the above field analysis, we extracted the commonalities and expressed the differences with parameters, thus designing the following abstract generic assembly algorithm components described in the Apla language:

```
function DBGSA (somefunc simplify_graph (g: digraph); somefunc remove_bubbles (g: digraph); somefunc removes_repeats (g: digraph); seq: list (list (char)): list (char)
    Errot_correction ();
    Build_graph ();
    Simplify_graph (g);
    Remove_tips ();
    Remove_bubbles ();
    Remove_repeats ();
    Contigs
    Scaffoidings
```

Among them, somefunc is a keyword defining function parameters, digraph is a directed graph type predefined in the PAR platform, and list is a predefined sequence type.

3.2.2. Component Interaction. Different components generate algorithms through interaction, and the interaction between components is also an important part of the component library. In this section, based on the feature model of the DBGSA domain established in Section 3, the interaction relationship between components is further analyzed to obtain the interaction model of the DBGSA component library.

The function of each step is as follows: error correction is to operate on the original short sequence. First, decompose it into k-mer and count the frequency of each k-mer, and then delete the k-mer with frequency <3. Build De Bruijn graph is to generate De Bruijn graph from the set of k-mer after error correction. Simplified graph is to simplify the generated De Bruijn graph and merge some isolated points. Remove branches is to delete all branches in the De Bruijn graph whose branch length is less than 2k. The bubble removal is to merge the two edges of the bubble in the De Bruijn graph into one edge. Repeat removal is to remove the tiny repeats in the De Bruijn graph. Contigs is to find a path that has and passes through each edge only once from the final De Bruijn graph. This path is contig. Scaffolding is to assemble all contigs into the final scaffold sequence, which is the final output-gene sequence.

Through the establishment of the DBGSA feature model, it is analyzed that the algorithm mainly includes five changing process features: error correction, construction of De Bruijn diagram, branch removal, contigs, and scaffolding. We take these features in the feature model as the main components and other features and related data as auxiliary components. Then, we established an interaction model between components according to their priorities. The model is shown in Figure 4.

The nodes connected by the solid lines in Figure 4 represent the basic components that must be included in the DBGSA domain, which corresponds to the 5 mandatory features in the feature model. The direction represented by the solid arrows indicates that the execution priority of the

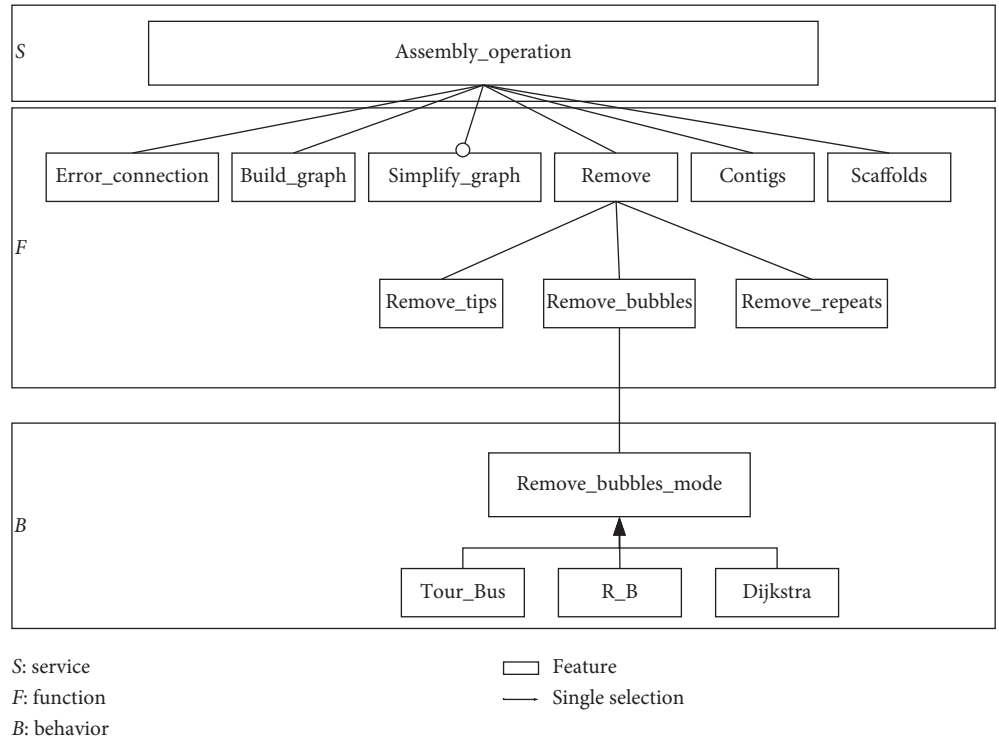

Figure 3: Feature model.

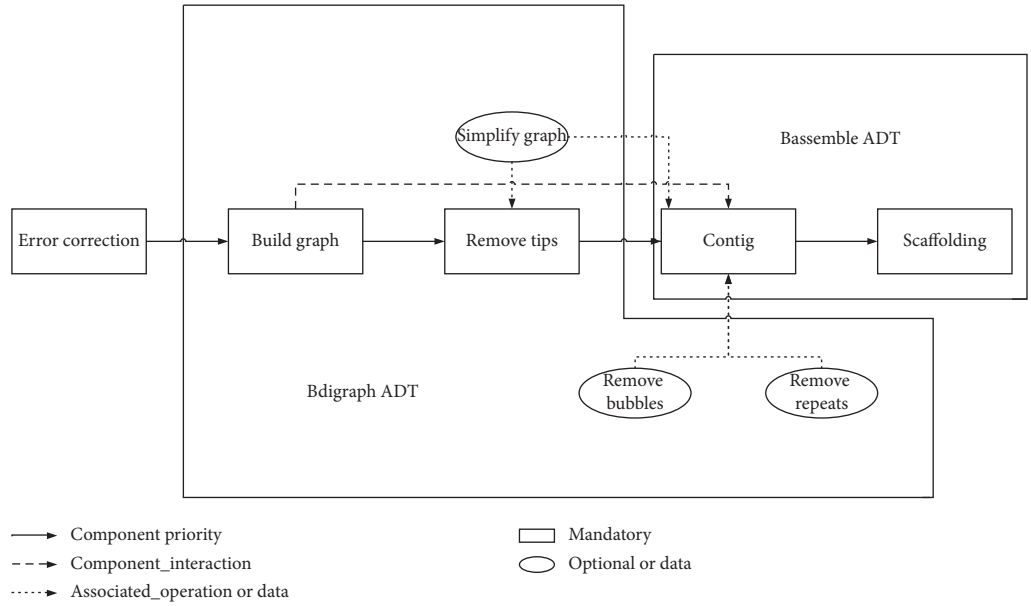

Figure 4: Component interaction model.

five components is from high to low; the dotted arrows indicate the interaction between the two components during the execution of the algorithm. As shown in Figure 4, the contig component needs to use the component diagram to determine whether it is a connected graph operation; the dotted arrow represents the data, structure, and associated operations required during the algorithm assembly process. For example, two abstract data types (ADT) need to be used in contig components: the ADT to remove air bubbles and the ADT to remove tiny duplicates.

The above interaction model includes the current mainstream sequence assembly algorithm, including the velvet algorithm, ABySS algorithm, and SOAPdenovo algorithm.

3.3. Apla Formal Implementation. Apla language can directly use abstract data types and abstract procedures to write programs, so it can describe algorithm problems more abstractly and is easy to verify the correctness of the program, ensuring the correctness and reliability of the program. In this section, based on the feature model of the DBGSA domain and the interactive model of algorithm components, the DBGSA model is formalized based on Apla. Due to space limitations, this paper only gives the definition of the components in the DBGSA domain and the specific explanation of the parameters in the program code.

error_correction component ADT

The sequence is decomposed into *k*-mers, which are usually determined by multiple factors such as gene size, read length, and computer memory. The error_correction component uses the frequency information of *k*-mer, and the *k*-mer of frequency (<3) will be removed.

```
procedure error_correction (a []: Array []);
begin
    . . .. . ./*Program code segment, omitted, the same below*/
end;
```

bdigraph component ADT

This component includes build_graph component, simplify_graph component, remove_tips component, remove_bubbles component, and remove_repeats component, which constitutes a whole, and then defines it as an ADT type. The ADT is defined as follows:

```
define ADT Bdigraph(sometype elem);
    type Bdigraph = private;
    function generate_graph (seq: list (char)): digraph;
    function simplify_graph (g: digraph): digraph;
    function remove_tips (g: digraph): digraph;
    function Tour_Bus (g: digraph): digraph;
    function B_R (g: digraph): digraph;
    function Dijkstra (g: digraph): digraph;
    function remove_repeats (g: digraph): digraph;
    . . .. . .
enddef.
```

Among them, the ADT type is named Bdigraph and has a type parameter elem; the function generate_graph represents the transformation of the input sequence to generate a De Bruijn graph; the function simplify_graph means to simplify the obtained De Bruijn graph; the function remove_tips represents tip removal of the De Bruijn graph; the function Tour_Bus represents the use of Tour_Bus algorithm to remove bubbles from the De Bruijn graph; the function B_R represents the use of bubble removal algorithm to remove bubbles from the De Bruijn graph; the function Dijkstra represents the use of Dijkstra's algorithm to remove bubbles from the De Bruijn graph; the function remove_repeats represents the small repetition removal of De Bruijn graph.

Bassemble component ADT

This component includes contigs and scaffolds components, which form a whole, and then define it as an ADT type. The ADT is defined as follows:

```
define ADT Bassemble (sometype elem);
function contigs (l: list (char)): list (char);
function scaffolds (l: list(char)): list (char);
. . .. . .
enddef.
```

Among them, the ADT type is named Bassemble and has a type parameter elem; the function contigs means to find a path from De Bruijn graph that each edge has and only passes once to obtain contigs; the function scaffolds indicates that the assembled contigs will continue to be assembled to form the final output genome sequence.

4. Algorithm Assembly and Experiments

4.1. Algorithm Assembly. We choose some components in the DBGSA component library for assembly and implement a specific sequence assembly algorithm (hereinafter referred to as the assembled algorithm). Part of the procedure is as follows:

```
program DBGSA_Assembly;
const
    path DBGSA Output:list (char); //DBGSA output address
var
    Seqs: list (list (char)); //seqs is input sequence
//The instantiation process of graph operation by Assembly algorithm
procedure progressive-depth (sometype elem; ADT Bdigraph (sometype elem); ADT Bassemble (sometype elem)). ADT Bdigraph: new bio_digraph (seqs; proc error_correction ());
function generate_graph (): new graph_generate ();
function remove_tips (): new tips_remove ();
function B_R (): new R_B ();
begin
    . . .. . .//program code segment, Omit, Same as below
end;
//The instantiation process of sequence assembly by Assembly algorithm
```

```
ADT Bassemble: new bio_assemble (g: digraph; ADT Bdigrapg (sometype elem));
function traverse_graph (): new graph_traverse ();
function contigs (): new bcontigs ();
function scaffolds (): new bscaffolds ();
function get_all_graph_contigs (): new bget_all_-graph_contigs ();
function error_correction (): new berror_correction ();
function local_assembly (): new bloacl_assembly ();
begin
  . . .. . .
end;
//Assembly algorithm main program formal code
procedure BASSEMBLY: new DBGSA assemble algorithm (sometype elem; ADT Bdigraph (sometype elem); ADT Bassemble (sometype elem); k = k min: integer; k max: integer; seqs_(k − 1): list)
begin
  bio_digraph generate_graph (seqs_(k − 1));
  bio_digraph remove_tips (g: digraph);
  bio_digraph.B_R (g: digraph);
  bio_assemble.traverse_graph (g: digraph);
  bio_assemble.get_all_graph_contigs (g: digraph);
  bio_assemble.error_correction (l: list);
  bio_assemble.local_assembly (l: list);
  k+ = 2;
  if (k < k max)
    BASSEMBLY ();
    . . .. . .
  . . .. . .
  bio_assemble.scaffolds (l: list);
end;
```

4.2. Experiments. We used the program generation system in the PAR platform to convert the Apla algorithm component into the corresponding C++ component and assembled a specific assembly algorithm. We obtained the genome sequence data of an African male individual from NCBI, reads1.fq and reads2.fq (accession NO. SRA000271). We took reads1.fq and reads2.fq as input data, the value of k is 25, and the resulting assembly result is shown in Figure 5. This paper chooses two currently popular sequence assembly algorithms, i.e., velvet and SOAPdenovo, for comparison. When k takes different values, we compare the results of velvet, SOAPdenovo, and the algorithm assembled in this paper. The results are shown in Tables 1–4.

The Number parameter in the table represents the number of contigs generated during the assembly process. The size of the species genome is fixed, so assembling as many reads as possible to reduce the number of contigs, the length of a single contigs will be longer, and the assembly results will be better.

The Max parameter in the table represents the maximum contig length among the contigs generated during the assembly process. Because sequencing errors and repeated fragments will exist in the data measured by sequencing technology, there will be many short reads. This affects the number of contigs generated during the assembly process. The maximum length of contig can indirectly indicate the pros and cons of the assembly algorithm.

The N50 parameter in the table is an important criterion for evaluating the result of sequence assembly. Sort all the contigs generated during the assembly process in order of length from smallest to largest, and then add the lengths of contigs in turn. When the sum of the added length reaches half of the total length, the length of the contigs added last is the value of the N50 parameter. The size of the N50 parameter value indicates the size of the ability of the contigs sequence to cover the standard genome. The larger the N50 value, the better the assembly result.

The N80 parameters in the table are similar to the N50 parameters. When the sum of the lengths of contigs added reaches 80% of the total length, the length of the last added contigs is the value of the N80 parameter. The size of the N80 parameter value can also indicate the quality of the assembly result.

According to the data in the above table, when k takes 15, 25, 35, and 45, respectively, the algorithm assembled in this paper can obtain a better result. It is not inferior to the other two popular sequence assembly algorithms in the Number parameter, Max parameter, N50 parameter, and N80 parameter. When the k value is 15, the running result of the assembled algorithm is significantly better than the SOAPdenovo algorithm and the velvet algorithm in four parameters. When the k value is 25, the result of the assembled algorithm is slightly inferior to the SOAPdenovo algorithm and the velvet algorithm in the four parameters. When the values of k are 35 and 45, respectively, the running results of the assembled algorithm are very close to the SOAPdenovo algorithm and the velvet algorithm in four parameters. This also shows that the assembled algorithm has good practicability.

```
>3 232 1201
CTGTTAGCCCCATCAGATTTCCTGGTGTTATAAACTTGATAAACGCCACCGGTGGGATTATCGTACCTTTGCTTATTGGTGAAGATGCTATGAAAATCGCCCCTCGAA
>4 106 331
TTCGTTTCCAACGTTCTTCAACGCCTCTATTTCATTTCTAGTGGTCGAAGGACTTTCTATTAATATGGACCGGATCACTGTGCGAATATAATCGTCGCTTTGACTC
>5 152 656
TTGATAGATTATATGACTGGAAAGCCATTCCAAACAATTCATTCAGAGACTCCTTTGCAAAGCAATCAAGAGCTGTTCAAGAGGAATTTCAATCTAGGTATTCTAAGA
>6 598 2831
ATTCTTAGCAGCATCACCAATCAATCTTTCAGTGTCAGTGAAAGCGACAAAAGATGGAGTGGTTCTGTTACCTTGATCGTTGGCAATAATGTCCACACGATCATTAGC
>7 27 31
TACTACTACTACTACTACTACTAC
>8 418 2431
AATCTATTAGCCGACATAAGAATAATGCATAAATAATATTTTTAATGTCTTCCTATGCCCAAAAGAAGAAGTCTTGAAGTTGCCGCACATGGAAATCACATGACCATG
>9 311 1298
CATCGCCACCACGTGATGCGACGTGGCGTCTGTTTGCAAGTGGCAAGAGGGCTCGCCGGGACCCGGAGGGATTCTACTGGAGCCGAATAGCCGCTCATACTGGTCCAT
>10 79 239
AGCATTGTTTGTATTGATCGGACTTGCTGATCTCTTGCCTGATTCATTCACCTCTCTCTCTGTTAGTAGCTTCATAGCT
>11 31 34
TATTCGTCTTCATCATTACCACCTGTTACCA
>12 25 5
AGAAGCAGTTTCCTGGCTTGTTGTT
>13 47 72
TGGTTCAATCATTTTTCGAATGTTTAAGAGGGAAGAGCTCTTCTTAC
>14 54 116
AAGCAGTTTCCTGGCTTGTTGTTGTACTGATAACAGGTGGTAATGATGAAGTAA
>15 131 862
AAGCATTTTCAGAAACTCGTCTTTTGAGTTCTAAATGCATCATACAACAACAACAATTTCTTATTACTGTGTCCTTTTGGGATTTTTCAGCCTTCCTGGCTTACCCAA
>16 36 144
CACAACAGAGTATACCACATGGTGCCCTATTTCTAC
>17 261 1224
CAGCAACAGCAACAGCAGCAACAGCAGCATGCCGGGCTACTAGGAAAGGGTACACCAAACGCTCTACAGCAGCAGCTACACATGAACCAGCTCACCGGGATACCTCCA
>18 249 1269
TAAGTGGATAGCTTGTGATAAATTGAATTTTCTCTGTTTAGTACTTGCCGAATAGTTACTTGTTAGTTGCAGATGCTTTTTGATGACAAAGTTATCAATCTCAATATT
>19 456 2093
AAGAGATGTTCAATCGTTTTAACAAATTCCAAGCTGCTGTCGCTTTGGCCCTACTCTCTCGCGGCGCTCTCGGTGACTCTTACACCAATAGCACCTCCTCCGCAGACT
>20 199 759
TCGTCATTCAATGGGCGCACGTTCGCCGTGCAACATTTAGGATTATGGTACATTACTATCCGGCCTGATCATAATATTAGTGTTCCCGAGGAAACTTATTCCTGGAAA
>21 604 3042
GCTGCTGGCCGAGGCACCAGCTTTGGGCGATTCTTGTGCCGCAGAGCCACTATTGAAAAGATCTTGATGCGTGCCATGAACCATCATGGGGAAAACCGGTTTAGGACT
```

Figure 5: Result of the assembled algorithm.

Table 1: The results of three assembly algorithms when $k = 15$.

$k = 15$	Assembly algorithm	SOAPdenovo	Velvet
Number	2308	5775	966
Max	1394	964	1732
N50	241	71	383
N80	386	188	583

Table 2: The results of three assembly algorithms when $k = 25$.

$k = 25$	Assembly algorithm	SOAPdenovo	Velvet
Number	907	964	344
Max	2260	4089	8976
N50	428	751	1835
N80	691	1326	3027

Table 3: The results of three assembly algorithms when $k = 35$.

$k = 35$	Assembly algorithm	SOAPdenovo	Velvet
Number	1096	946	895
Max	1141	1058	1085
N50	273	262	277
N80	389	366	390

Table 4: The results of three assembly algorithms when $k = 45$.

$k = 45$	Assembly algorithm	SOAPdenovo	Velvet
Number	1024	365	414
Max	413	413	424
N50	136	172	172
N80	174	197	199

5. Conclusion

Gene sequence assembly algorithm is a key issue in bioinformatics, and its algorithm and application research have received extensive attention. However, there is no work that regards it as a specialized domain and conducts research from a high level of abstraction. This paper analyzes genes based on DBG strategy from the domain level. We analyze the domain of assembly algorithm, carry out research on the highly abstract algorithm framework to improve the reliability and development efficiency of the algorithm, and reduce the probability of problems such as algorithm error.

This paper adopts generative programming methods and feature modeling techniques to analyze and extract general features and variable features in the domain of assembly algorithms based on DBG strategy and design corresponding components. The interaction model between the components is designed according to the dependency between the features, and then the high abstract language Apla is used for formal realization. Finally, the conversion system of the PAR

platform is used to formalize the assembly of the components in an automatic or semiautomatic manner to generate the solution algorithm for specific problems. The comparative experiments in Section 4 show that the gene sequence assembly algorithm assembled in this study has also achieved better assembly results and has high practicability.

The research in this paper adds domain-level research to the domain of gene assembly and formalizes the DBGSA component library, which can assemble specific gene sequence assembly algorithms. Our research ensures the efficiency of algorithm development and the reliability of the assembly algorithm, reduces the error and unnecessary space-time overhead caused by manual selection algorithm for gene assembly, and also provides a valuable reference for solving problems in the domain of gene assembly.

Further work includes the following aspects:

(1) The research results of the component design and implementation based on the DBG strategy assembly algorithm can theoretically be applied to any other algorithms in bioinformatics. The next step is to expand the research domain of this paper and include most of the gene sequence assembly algorithms into the research scope, laying the foundation for the future realization of a gene sequence operation platform

(2) Development and assembly platform: through the visual interface, users choose different components and assemble different gene sequence assembly algorithms, which further shortens the time spent by users, facilitates user operations, and enhances the user experience

(3) With the development of new technologies such as big data and cloud computing, the Apla language will surely be applied in more domains. We will carry out further research on the PAR platform and consider applying the Apla language and PAR platform to other domain in bioinformatics

Data Availability

The datasets generated for this study are available on request to the corresponding author.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (nos. 62062039 and 61662035) and the Natural Science Foundation of Jiangxi Province (no. 20202BAB202024).

References

[1] L. L. Tu, N. C. Wang, Y. Chen, and Q. P. Mei, "Approaches to DNA fragment assembly," *Life Science Research*, vol. 7, no. 2, pp. 79–82, 2003.

[2] E. W. Myers, G. G. Sutton, A. L. Delcher et al., "A whole-genome assembly of Drosophila," *Science*, vol. 287, no. 5461, pp. 2196–2204, 2000.

[3] L. B. M. De and W. R. Mccombie, "Assembling genomic DNA sequences with PHRAP," *Current Protocols in Bioinformatics*, vol. 17, no. 1, pp. 11.4.1–11.4.15, 2007.

[4] M. Margulies, M. Egholm, W. E. Altman et al., "Genome sequencing in microfabricated high-density picolitre reactors," *Nature*, vol. 437, no. 7057, pp. 376–380, 2005.

[5] P. Mihai, "Genome assembly reborn: recent computational challenges," *Briefings in Bioinformatics*, vol. 10, no. 4, pp. 354–366, 2009.

[6] M. Pop and S. L. Salzberg, "Bioinformatics challenges of new sequencing technology," *Trends in Genetics*, vol. 24, no. 3, pp. 133–141, 2008.

[7] G. Peng, P. Ji, and F. Zhao, "A novel codon-based de Bruijn graph algorithm for gene construction from unassembled transcriptomes," *Genome Biology*, vol. 17, no. 1, p. 232, 2016.

[8] D. L. Camwron, J. Schroeder, J. S. Penington et al., "GRIDSS: sensitive and specific genomic rearrangement detection using positional de Bruijn graph assembly," *Genome Research*, vol. 27, no. 12, pp. 1–11, 2017.

[9] N. I. Weisenfeld, V. Kumar, P. Shah, D. M. Church, and D. B. Jaffe, "Direct determination of diploid genome sequences," *Genome Research*, vol. 27, no. 5, pp. 757–767, 2017.

[10] P. A. Pevzner, H. Tang, and M. S. Waterman, "An Eulerian path approach to DNA fragment assembly," *Proceedings of the National Academy of Sciences*, vol. 98, no. 17, pp. 9748–9753, 2001.

[11] A. Pavel and Pevzner, "Fragment assembly with double-barreled data," *Bioinformatics*, vol. 17, no. suppl 1, pp. S225–S233, 2001.

[12] D. R. Zerbino and E. Birney, "Velvet: algorithms for de novo short read assembly using de Bruijn graphs," *Genome Research*, vol. 18, no. 5, pp. 821–829, 2008.

[13] J. T. Simpson, K. Wong, S. D. Jackman, J. E. Schein, S. J. M. Jones, and I. Birol, "ABySS: a parallel assembler for short read sequence data," *Genome Research*, vol. 19, no. 6, pp. 1117–1123, 2009.

[14] P. Ni, W. Dai, Y. Liu et al., "A novel method for better bacterial genome assembly from Illumina data," *Current Bioinformatics*, vol. 12, no. 6, 2017.

[15] 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, 2012.

[16] S. Nurk, D. Meleshko, A. Korobeynikov et al., "metaSPAdes: a new versatile metagenomic assembler," *Genome Research*, vol. 27, no. 5, pp. 824–834, 2017.

[17] B. Pevzner, Z. W. Dong, and S. Y. SinghGu, "Hybrid microgrid many-objective sizing optimization with fuzzy decision," *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 11, pp. 2702–2710, 2020.

[18] B. Kumar, Y. J. Zhao, and X. Y. MaLing, "Applying graph-based differential grouping for multiobjective large-scale optimization," *Swarm and Evolutionary Computation*, vol. 53, Article ID 100626, 2020.

[19] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, New Jersey, NJ, USA, 2000.

[20] C. R. Turner, A. L. Wolf, A. Fuggetta et al., "Feature engineering," in *Proceedings of the IWSSD*, IEEE Computer Society, Torrance, CA, USA, April 1998.

[21] K. Lee, K. C. Kang, and J. Lee, "Concepts and guidelines of feature modeling for product line software engineering," in *Proceedings of the International Conference on Software Reuse*, pp. 62–77, Springer, Austin, TX, USA, November 2002.

[22] W. Zhang and H. Mei, "A feature-oriented domain model and its modeling process," *Softw*, vol. 14, no. 8, pp. 1345–1356, 2003.

[23] J. Xue, "A unified approach for developing efficient algorithmic programs," *Journal of Computer Science and Technology*, vol. 12, no. 4, pp. 314–329, 1997.

[24] C.-J. Wang and J.-Y. Xue, "Formal derivation of a generic algorithmic program for solving a class of extremum problems," in *Proceedings of the 2009 10th ACIS International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel Distributed Computing*, pp. 100–105, IEEE, Daegu, South Korea, May 2009.

[25] J. Xue, "Genericity in PAR platform," in *Proceedings of the International Workshop on Structured Object-Oriented Formal Language and Method*, pp. 3–14, Springer, Paris, France, November 2015.

[26] X. Jinyun, "Two new strategies for developing loop invariants and their applications," *Journal of Computer Science and Technology*, vol. 8, no. 2, pp. 147–154, 1993.

[27] X. Jin-Yun, "Formal derivation of graph algorithmic programs using partition-and-recur," *Journal of Computer Science and Technology*, vol. 13, no. 6, pp. 553–561, 1998.

[28] E. M. D. Armas, E. H. Haeusler, S. Lifschitz et al., "K-mer Mapping and de Bruijn graphs: the case for velvet fragment assembly," in *Proceedings of the IEEE International Conference on Bioinformatics & Biomedicine*, IEEE, Kansas City, MO, USA, August 2017.

[29] S. D. Jackman, B. P. Vandervalk, H. Mohamadi et al., "ABySS 2.0: resource-efficient assembly of large genomes using a Bloom filter," *Genome Research*, vol. 27, no. 5, pp. 768–777, 2017.

[30] Y. Chu, C. Lauren, L. Warren René et al., "ARCS: scaffolding genome drafts with linked reads," *Bioinformatics*, vol. 34, no. 5, pp. 725–731, 2018.

[31] M. Michael, H. Ehsan, and I. Lucian, "SAGE2: parallel human genome assembly," *Bioinformatics*, vol. 34, no. 4, pp. 678–680, 2017.

[32] Sufang and M. Wang, "Comprehensive evaluation of de novo transcriptome assembly programs and their effects on differential gene expression analysis," *Bioinformatics*, vol. 33, no. 3, pp. 327–333, 2016.