BMC Bioinformatics

METHODOLOGY ARTICLE

Open Access

A tri-tuple coordinate system derived for fast and accurate analysis of the colored de Bruijn graph-based pangenomes

Jindan Guo, Erli Pang, Hongtao Song and Kui Lin*

*Correspondence: linkui@bnu.edu.cn
State Key Laboratory of Earth Surface Processes and Resource Ecology, Ministry of Education Key Laboratory for Biodiversity Science and Ecological Engineering, College of Life Sciences, Beijing Normal University, Beijing, China

Abstract

Background: With the rapid development of accurate sequencing and assembly technologies, an increasing number of high-quality chromosome-level and haplotype-resolved assemblies of genomic sequences have been derived, from which there will be great opportunities for computational pangenomics. Although genome graphs are among the most useful models for pangenome representation, their structural complexity makes it difficult to present genome information intuitively, such as the linear reference genome. Thus, efficiently and accurately analyzing the genome graph spatial structure and coordinating the information remains a substantial challenge.

Results: We developed a new method, a colored superbubble (*cSupB*), that can overcome the complexity of graphs and organize a set of species- or population-specific haplotype sequences of interest. Based on this model, we propose a tri-tuple coordinate system that combines an offset value, topological structure and sample information. Additionally, *cSupB* provides a novel method that utilizes complete topological information and efficiently detects small indels (< 50 bp) for highly similar samples, which can be validated by simulated datasets. Moreover, we demonstrated that *cSupB* can adapt to the complex cycle structure.

Conclusions: Although the solution is made suitable for increasingly complex genome graphs by relaxing the constraint, the directed acyclic graph, the motif cSupB and the *cSupB* method can be extended to any colored directed acyclic graph. We anticipate that our method will facilitate the analysis of individual haplotype variants and population genomic diversity. We have developed a C++ program for implementing our method that is available at https://github.com/eggleader/cSupB.

Keywords: Genome graph, Coordinate system, Variant detection

Introduction

Background

Within a certain species, individual genomes vary in both the gene content and genomic portions of DNA sequences. How to accurately and comprehensively identify the genome-wide diversity of the species remains challenging. Recently, because of the rapid development of accurate long-read sequencing and assembly technologies [1, 2], for many species, abundant high-quality chromosome- and haplotype-resolved assemblies of species- or population-specific genomes have been derived, thereby accelerating the coming of the population genome era [3]. Together with various existing reference genomes, this type of genomic data has spawned a new research field called computational pangenomics [4]. The most accepted definition of pangenome is any collection of genomic sequences to be analyzed jointly or used as a reference [5]. A pangenome provides a complete picture of genomes and complex genomic variants within a species of interest and provides an opportunity for the development of efficient computational methods and various promising applications in medical biology [6], ecology [7], and evolutionary biology [8].

The graph-based model is among the most important representative models for the pangenome [5, 9]. Due to its good mathematical properties and theoretical foundation, graphs have achieved essential long-term status in biological sequence analysis [10–12]. The graph achieves compression and removal of redundancy and retains the continuity between sequences such that each sequence is encoded as a walk on the graph [13]. To better analyze the various visible or potential information provided by the graph, many basic and interesting topics, including graph selection, construction, storage, searching, mapping, and comparison, have been studied [14–18], but many challenges remain. One is to design appropriate coordinate systems for accurate graph spatial structure analysis. Defining a genome graph coordinate system that determines the correspondence between graph topology and sequence information is highly important, such as maintaining the translation between linear reference assemblies and genome graphs and improving annotations between existing assemblies and genome graphs [19].

The current coordinate systems can be roughly divided into two categories. The first type is one-tuple representation, which is based on the idea of linearity. The basic research ideas include blocking, sorting, and constructing a graph according to genome alignments [20, 21]. However, this approach is an NP-hard problem and changes the topological structure of the graph while adding linearity, which inevitably leads to a loss of information. The second type is the multiple-tuple representation. For example, Rand et al. defined an offset-based coordinate system consisting of a region identifier and an offset [19]. One shortcoming of this approach is that it is suitable mainly for small sample sizes and simple variants. Most recently, Li H et al. designed a new system that combines a segment coordinate system with a binary group (segId; segOffset) on the sequence graph [22]; however, the topological information of the graph is not adequately described by these two methods.

Our contribution

In this work, we designed a tri-tuple coordinate system for a class of colored de Bruijn graphs constructed from a set of genomes within a species by selecting an adequate k-mer length. We developed an efficient algorithm to identify each specific graph spatial structure, called a colored superbubble (cSupB), and organized these cSupBs into a tree that accurately reflects their inclusion relationships depicted in the colored de Bruijn graph we constructed. Compared to superbubbles [23] and ultrabubbles [24], each cSupB inherits similar advantages, such as globally organizing the sites without the existing reference genome. More importantly, the tri-tuple coordinate system is derived mainly from the inferred cSupB tree. To demonstrate the feasibility and efficiency of the approach, we elucidated the algorithm's practical performance through a comprehensive set of experiments: (1) a real dataset of 12 human mitogenomes and (2) 310 simulated human mitogenome datasets. The results show that, on the one hand, the tri-tuple coordinate system can accommodate the existing linear reference coordinates and annotation data; on the other hand, the variants detected from the cSupBs are more comprehensive and diverse. Furthermore, we demonstrated that our analysis method is capable of managing tremendous amounts of genomes (5 k/20 k samples tested) with less memory and time consumption.

Notation and related work

Representation

As discussed above, the colored de Bruijn graph, an extension of the classic de Bruijn graph, is essential for our algorithm. Given a set of sequences and a k value, we can transform the sequence into k-mers (unique strings of length k). Two different k-mers are adjacent if the k-1 base suffix of the "from" k-mer and the k-1 base prefix of the "to" k-mer are the same. If we regard each k-mer as a node and the adjacent relationship between k-mers as an edge, we can obtain a de Bruijn graph. By coloring the nodes and edges in the graph by the samples, we can extend classical de Bruijn graphs to the colored de Bruijn graph, as shown in Fig. 1. We provide the following definitions in the directed graph for the convenience of research.

Path: A path from v_0 to v_k refers to a sequence $v_0, e_1, v_1, e_2, \ldots, e_k, v_k$. e_i is the edge connecting v_{i-1} with v_i, and the length of the path is k. If there is a path in the graph with the same start and end node, the path is "closed", indicating that the graph has a **cycle**.

Incoming node/edge: For any two distinct nodes u, v, u is called the **incoming node** of v if there is a path from u to v. The edges on this path are called the **incoming edges** of v, and the number of adjacent incoming edges is called the **indegree** of u.

Outgoing node/edge: Based on the above definition, conversely, v is called the **outgoing node,** and all the edges on the path are called the **outgoing edges** of u. Similarly, the number of adjacent outgoing edges is called the **outdegree** of u.

Degree: The sum of the indegree and outdegree of a node is called the node's **degree**.

Supernode: If the outdegree or indegree of a node is greater than one, then the node is called a **supernode**.

Branch: In this paper, a maximal path is called a branch if the outdegree and indegree of all nodes on the path are equal to 1.

Bridge: *In this paper, a branch is called a bridge; if this branch is deleted, the number of connected components of the graph will increase.*

Bubble: the structure formed by two paths redundant if they start and end at the same nodes is called a bubble [25]; for a general definition, see Definition 1 of superbubble in the cSupB subsection.

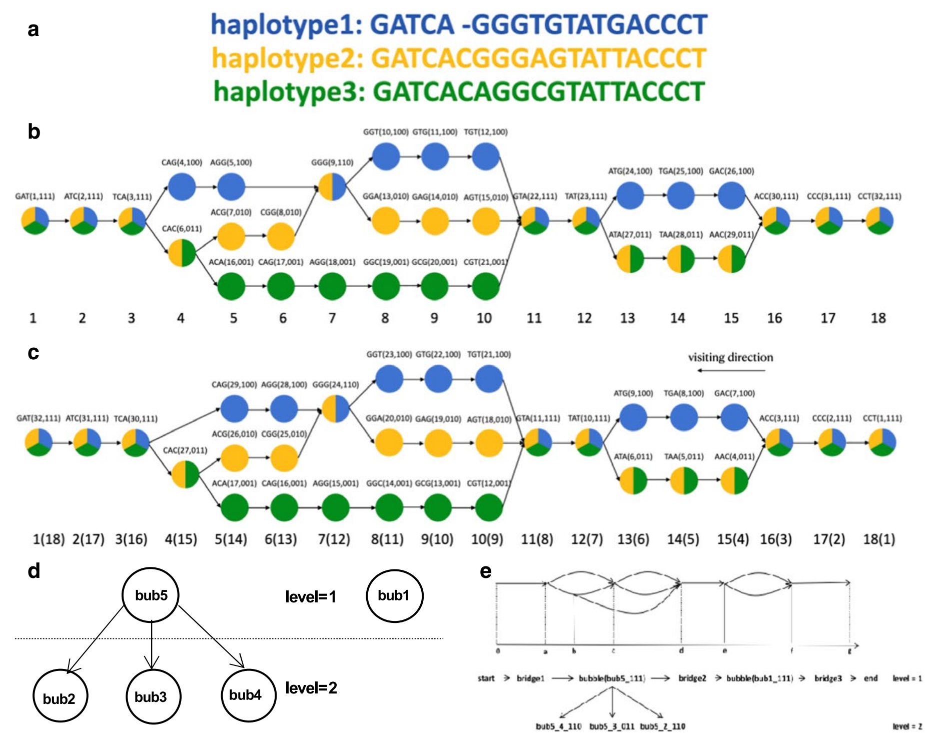
a
haplotype1: GATCA -GGGTGTATGACCCT
haplotype2: GATCACGGGAGTATTACCCT
haplotype3: GATCACAGGCGTATTACCCT
b
GAT(1,111) ATC(2,111) TCA(3,111) CAG(4,100) AGG(5,100) GGG(9,110) GGT(10,100) GTG(11,100) TGT(12,100)
CAC(6,011) ACG(7,010) CGG(8,010) GGA(13,010) GAG(14,010) AGT(15,010) GTA(22,111) TAT(23,111)
ACA(16,001) CAG(17,001) AGG(18,001) GGC(19,001) GCG(20,001) CGT(21,001)
ATG(24,100) TGA(25,100) GAC(26,100) ATA(27,011) TAA(28,011) AAC(29,011) ACC(30,111) CCC(31,111) CCT(32,111)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
c
visiting direction
1(18) 2(17) 3(16) 4(15) 5(14) 6(13) 7(12) 8(11) 9(10) 10(9) 11(8) 12(7) 13(6) 14(5) 15(4) 16(3) 17(2) 18(1)
d
bub5
level=1
bub1
bub2 bub3 bub4
level=2
e
start → bridge1 → bubble(bub5_111) → bridge2 → bubble(bub1_111) → bridge3 → end level = 1
bub5_4_110 bub5_3_011 bub5_2_110 level = 2

Fig. 1 Schematic diagram for the construction, decomposition and reorganization of the colored de Bruijn graph. **a** The three original sequences. **b** The colored de Bruijn graph with k equal to 3. Each colored circle represents a node, and the black arrows represent edges. The characters above the node represent each node's label, visit order and color. The number below node represents the theoretical offset value of each node. **c** The result of visiting against the direction of the edge. The numbers below the node represent the final offset value and preoffset value. **d** The cSupB tree structure. Here, we find five cSupBs in the following order: bub1.< TAT, ACC, 111 >; bub2.< GGG, GTA, 110 >; bub3.< CAC, GTA, 011 >; bub4.< TCA, GGG, 110 >; and bub5.< TCA, GTA, 111 >. **e** The final decomposition and representation of the colored de Bruijn graph

Construction

Iqbal et al. first introduced the colored de Bruijn graph aimed at "detecting and genotyping simple and complex genetic variants in an individual or population" [26, 27]. As the space-efficient representations of de Bruijn graphs have also been heavily researched [28–31], many succinct data structures for the colored de Bruijn graph were developed and related software mainly includes the bloom filter trie (BFT) [32],VARI [33], rainbowfish [34], pufferfish [35], mantis [36], and bifrost [37]. In this paper, we use the VARI-merge [38], the new VARI version, to construct the colored de Bruijn graph. Given the large-scale samples and large size of the sequence data involved in whole genomes, VARI-merge can store and traverse genomes in a space- and time-efficient manner. In fact, the choice of the software is not particularly important because our study focuses on the analysis of graph structure instead of construction.

The colored de Bruijn graph construction includes four steps: k-mer extraction and counting, k-mer sorting, construction of the de Bruijn graph, and construction of the color matrix. The detailed installation process and usage method can be found at https://github.com/cosmo-team/cosmo/tree/VARI.

Roadmap

In the next section, we describe the method from graph construction to decomposition and reorganization to applications including variant detection and coordinate system construction. Section 3 then elucidates the practical performance of the tri-tuple coordinate system and variant detection method. Section 4 offers some concluding remarks.

Method

Our data structure for colored de Bruijn graphs is the most important research object in this paper, so we start by describing its construction. We then give a detailed explanation of how we traverse, decompose and reorganize the graph because graph spatial structure analysis is the basis of further research. Using analytical results, on the one hand, we propose a variant detection method for small indels; on the other hand, we design a tri-tuple coordinate system with details of our implementation.

Graph construction

Let $\{s_1, s_2, \ldots, s_n\}$ be a set of haplotype genomic sequences. Because degeneracy is intractable for VARI, we replaced degenerate bases with the most frequent base of other sequences and recorded the modification. In addition, to ensure that each sequence does not have repeat k-mers, we find the longest repeat segment length k_i of each sequence s_i and take $K = max\{k_1, k_2, \ldots, k_i\}$. When the parameter k value is greater than K, the colored de Bruijn graph is almost guaranteed to be acyclic. Additionally, we randomly generate two segments consisting of ACGT with a length of k and add them to the front and end of each sequence to guarantee that the graph has only one start node and end node. The segment's function is only to anchor the sequences, which does not affect the spatial framework, so it cannot appear in any original sequences. The segments are not fixed and change with the samples.

Finally, the obtained colored de Bruijn graph has the following characteristics: directed, acyclic, nondegeneracy, and unique start and end nodes. Moreover, each sequence corresponds to a unique path from the start node to the end node in the graph (Additional file 1: property 1; all the proofs of the properties, theorems, and corollaries can be found in Additional file 1).

Decomposition and reorganization

This section includes four parts. The first part is the theoretical preparation where we define two important concepts: cSupB and offset value. The second part decomposes the graph and obtains all the cSupBs. The third part determines the relations between cSupBs, and the last part addresses cycles.

Colored superbubble (cSupB)

Before defining the cSupB, we need to know the definition of superbubble.

Definition 1 An assembly graph $G = (V, E)$ is a directed graph. Denote $V(G)$ and $E(G)$ as the set of nodes and edges, respectively, of graph G. For any two distinct nodes

s and t in G, $< s, t >$ is called a **superbubble** if it satisfies the following four criteria [23, 39]:

(1) reachability: there is a path from s to t;
(2) matching: the set of nodes reachable from s without passing through t is equal to the set of nodes from which t is reachable without passing through s;
(3) acyclicity: the subgraph induced by U is acyclic, where U is the set of nodes satisfying the matching criterion;
(4) minimality: no node in U other than t forms a pair with s that satisfies the conditions above;

Here, s and t are called the source node and sink node, respectively.

Based on Definition 1, there are two superbubbles in Fig. 1b:$< TAT, ACC >$; $< TCA, GTA >$. Here, we only discuss the superbubbles that contain at least two supernodes.

Many superbubble search methods have been developed [23, 39–41]. These methods can be applied to different graphs, and all are based on the topological structure. However, a superbubble contains structural information but lacks sample information; therefore, based on the colored de Bruijn graph, we propose the cSupB, a generalization of the superbubble, and its relevant algorithm.

Definition 2 An assembly graph $G = (V, E, C)$ is a colored de Bruijn graph. Denote $V(G)$, $E(G)$ and $C(G)$ as the sets of nodes, edges, and colors, respectively, of graph G. For a color set C_1 where $C_1 \subset C$, $G_1 = (V_1, E_1, C_1)$ is the subgraph induced by the nodes $u \in V$ whose color $color(u)$ satisfies $color(u_i) \cap C_1 \neq \emptyset$. For any two distinct nodes s and t in G,$\langle s, t, C_1 \rangle$ is called a cSupB if it satisfies the four criteria of a superbubble in graph G_1.

We can also define **separation, intersection, and inclusion** of cSupBs.

For any two different *cSupBs* named $cSupB1$ and $cSupB2$, $G_1 = (V_1, E_1, C_1)$ and $G_2 = (V_2, E_2, C_2)$ are the subgraphs induced by the nodes in $cSupB1$ and $cSupB2$ respectively. Then we set $V_0 = V_1 \cap V_2, C_0 = C_1 \cap C_2$ and we can determine the relations of *cSupBs* according to five conditions as follows:

(1) Separation: a. $V_0 = \varnothing$; b. $V_0 \neq \varnothing$ and $C_0 = \varnothing$;
(2) Intersection: c. $V_0 \neq \varnothing$ and $C_0 \neq \varnothing$ and $V_0 \neq V_1$ or V_2;
(3) Inclusion: d. $V_0 = V_1$ and $C_0 = C_1$; e. $V_0 = V_2$ and $C_0 = C_2$.

In particular, if two cSupBs are inclusive relations, for example: $V_0 = V_2$ and $C_0 = C_2$, we name $cSupB1$ the parent bubble of $cSupB2$andcSupB2 is the child bubble of $cSupB1$.

Based on Definition 2, there are five cSupBs in Fig. 1b: bub1.$< TAT, ACC, 111 >$; bub2.$< GGG, GTA, 110 >$; bub3.$< CAC, GTA, 011 >$; bub4.$< TCA, GGG, 110 >$; and bub5.$< TCA, GTA, 111 >$. Bub5 is the parent of bub2, bub3 and bub4. Here, we can prove that a superbubble is only a special group of cSupBs whose source and sink node colors are the same. Each cSupB can be thoroughly decomposed to simple cSupBs that have only two paths from the source node to the sink node. Simple cSupB is the minimum bubble structure that can be used to study relationships and variants for any two samples, but superbubbles cannot.

Definition 3 In a directed acyclic graph (DAG) $G=(V,E)$ with unique start node $u^{'}$, for each node $v_i \in V$, let S_i denote all the adjacent incoming nodes of v_i. If $pos(v_i)=max\{pos(u_j)\}+1$, where $u_j \in S_i$, and $pos(u^{'})$ is known, then $pos(v_i)$ is called the **offset value** of v_i.

Indeed, in a DAG $G=(V,E)$, for any two distinct nodes $u,v \in V$ and $pos(u)=a, pos(v)=b$, if there exists at least one path between u and v, $|a-b|$ must be the length of the longest path between u and v (Additional file 1: property 3).

Graph traversal and cSupB Matching

Before traversing and decomposing the graph, we need to label four visiting states for eachnode as unvisited, half-visited, to-be-visited and fully visited. See Additional file 1 for the details of the description.

We define a **postorder-like traversal** strategy: in a DAG, **a node can be visited if all its parent nodes have been visited.** In other words, for each node in the graph, we could visit it and its outgoing nodes if and only if all its incoming nodes have been visited.

Matching principle: At least two adjacent outgoing edges' colors of the source node intersect with the color of the sink node. Similarly, at least two adjacent incoming edges' colors of the sink node intersect with the color of the source node. Then, the source node and sink node are matched, and the cSupB color is the color intersection of the source node and the sink node.

If we encounter a node s whose outdegree is greater than one, we place it in a to-be-visited source node queue Q; if we encounter a node t whose outdegree is greater than one, we start to find its matched source node s from Q reversely based on the matching principle. Specifically, if $color(s) \subseteq color(t)$, we remove s from Q and continue; if $color(s) \subseteq color(t)$, we stop.

We can prove that for each node t whose indegree is greater than one, at least one node s whose outdegree is greater than one can be used to construct a cSupB with t (Theorem 2, Corollary 5). Moreover, s is visited before t, and s is the parent node of t (Corollaries 3 and 4).

cSupB subordination

After postorder-like traversal, we can obtain all the cSupBs and their information, including the source/sink node, cSupB color, and ordering. Ordering denotes the order of obtaining the cSupB, and we use the function *order* denoting it in the following study. Here, we reorganized the cSupBs by determining the relationship of cSupBs.

To organize the cSupBs, we adopt a hierarchical tree structure by exploiting the relations of cSupBs. Furthermore, after confirming the root cSupB (containing all samples), we assigned each cSupB a hierarchical level. Here, the root cSupB's level is 1.

Method to determine the inclusion of cSupB

For $cSupB1$, $cSupB2$ and $cSupB3$, $G_1 = (V_1, E_1, C_1)$, $G_2 = (V_2, E_2, C_2)$ and $G_3 = (V_3, E_3, C_3)$ are the subgraphs induced by the nodes in $cSupB1$, $cSupB2$ and $cSupB3$ respectively. If the following three conditions are satisfied simultaneously:

a. a.order$(\mathrm{G}_1) <$ order(G_2);
b. b.$\mathrm{C}_1 \subseteq C_2$ and $C_1 \neq C_0$(C_0 is a color set containing all samples);
c. there is no $cSupB3$, s.t. $order(G_1) < order(G_3) < order(G_2)$ and $C_1 \subset C_3$;

we call $cSupB1$ a child cSupB of $cSupB2$ and call $cSupB2$ the nearest parent cSupB of $cSupB1$ (Additional file 1: Theorem 3).

An essential conclusion is that a child cSupB cannot be obtained before its parent cSupB (Additional file 1: Corollary 6), and the cSupB structure must be a tree (Additional file 1: Corollary 7). Figure 1 D, E is an example of the final decomposition and representation of the colored de Bruijn graph.

To store various information about the colored de Bruijn graph, we define several file formats, such as cSupB topology (CST), cSupB detailed information (CSDI), node nearby information (NNI) and offset value information (OVI). See Additional file 1 for the details of the file formats.

Cycle

A cycle is the representation of repeat sequences on the graph and is sometimes common but inevitable. Cycles can be divided into two types: type I, formed by a single sample; and type II, formed by different samples. Our k value selection method can only avoid type I. Here, we propose a method to address cycles including whether cycles exist, where the cycles are and how to cut the sequences.

Cycle identification There exists a depth-first-search strategy that can identify all the cycles simultaneously; however, the iterative recursion and high computational cost limit its application. In this paper, we propose a cycle identification method based on the offset value.

According to Theorem 2, if we traverse graph G from the start node using the postorder-like traversal strategy, we can visit all the nodes. In particular, if the traversal is finished but nodes still exist that are unvisited, we call the traversal finished in advance and there must be cycles in graph G (Theorem 2). Information about half-visited nodes is collected to prepare for cycle interval determination.

We divide half-visited nodes into two types: type I: on the cycle; type II: not on the cycle. We can prove that if the traversal is finished in advance, we must obtain at least one type I half-visited node (property 5) (Fig. 2).

Cycle interval location At least one half-visited node must exist on the cycle when the traversal ends in advance (property 5). Therefore, we need to identify this node, mark it as fully visited, and restart the graph traversal. Due to the existence of the cycle, this node is revisited, and we record the offset value information again to help us determine the interval of the cycle.

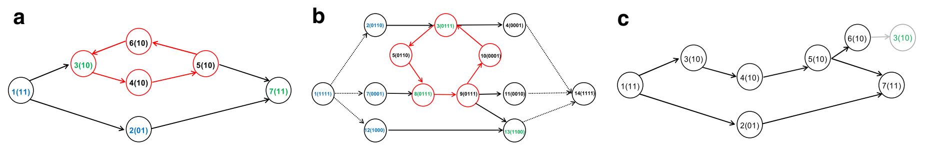

Fig. 2 Half-visited nodes in two types of cycles. Each circle represents a node, and the red dots and edges form a cycle. In the circle, the number before the bracket indicates the node ID, and the number in the bracket represents the node color. The color of the number indicates the visiting state of the node: blue indicates fully visited, green indicates half-visited, and black indicates unvisited. **a** Graph composed of two samples, where the cycle belongs to type I. When the traversal stops, two half-visited nodes, 3 and 7, are generated. Among them, 3 is of the first type and 7 is of the second type. **b** Graph composed of four samples, where the cycle is a type II cycle. When the traversal stops, three half-visited nodes (3, 8, and 13) are generated. Among them, 3 and 8 are of the first type, and 13 is of the second type. **c** The revisiting result of **a**. When the traversal is finished in advance, two half-visited nodes (3 and 7) are obtained, and the intersection of the colors of the two nodes is not empty. At this time, $\text{pos}(1) = 1, \text{pos}(3) = \text{pos}(2) = 2, \text{pos}(7) = 3$, and node 3 is selected as cycle_start_node, $\text{cycle\_start\_pos} = 2$. Then, by continuing to visit, we can obtain $\text{pos}(4) = 3, \text{pos}(5) = 4, \text{pos}(6) = \text{pos}(7) = 5$ and know that cycle_end_node is 6, $\text{cycle\_end\_pos} = 5$. Finally, the interval of the cycle is [2, 5]

Steps of interval determination

1. Let V_0 represent the collection of all half-visited state nodes.
2. Choose node u from V_0 whose offset value is minimal as the cycle start node.
3. Mark u as fully visited and restart the graph traversal from u. If traversal is again finished in advance, we encounter a new cycle and return to step 1. If we meet a node v, v is the adjacent parent node of u. Then, we set $cycle\_start\_pos = pos(u)$ $cycle\_end\_pos = pos(v), cycle\_start\_node = u, cycle\_end\_node = v$.
4. Then, $[cycle\_start\_pos, cycle\_end\_pos]$ is the cycle interval.

We choose the node with the minimum offset value for simplicity. If the selected node is on the cycle, the traversal can proceed; otherwise, the traversal will terminate in advance, and the previous steps are repeated.

Sequence segmentation and graph reconstruction After the interval $[cycle\_start\_pos, cycle\_end\_pos]$ of the cycle is determined, we select a cutting point in this interval to divide the sequence into two parts and construct the graph.

Here, we divide the cutting points into three categories: type I: the bridge. This type is our ideal cutting point; type II: Not the bridge but involving all samples; type III: only involving a portion of the samples.

Based on the above ideas, we propose a cutting point selection model as follows.

1. Position transformation. We establish a connection between the offset value and bases. Here, we define the offset value of the node according to its last base in the k-mer.
2. Information obtainment. We take each position as the key and the color, bases, thickness, and other information as the values to establish hash tables in the interval $[cycle\_start\_pos, cycle\_end\_pos]$.
3. Evaluation strategy. We set an index $R = L/t$, in which L denotes the number of continuous loci with the same position information and t denotes the thickness. In par-

ticular, if there are multiple choices with the same *R*-value, we prefer bridges; if the *L* value is 0, only the position with the smallest thickness can be selected as the cutting position. In other cases, we make the decision according to the size of the *R*-value.

4. Cutting position determination. We select the optimal area as the reference cutting position. If there are multiple cycles, comprehensively considering the information of all cycles and their intersection should be the priority. The output format is cycle cut position (CCP) format (see Additional file 1 for details).

Sequence cutting and graph reconstruction Although we know the cutting position of the graph, we must obtain the specific cutting position of each sample. We searched the cutting area sequence in each relevant sample. If there is only one search result, the middle position is the cutting position; if there is more than one search result, the nearest one is selected as the reference cutting position. Then, we cut each sequence, add the head and tail sequence again, divide them into several files, put them in multiple directories, and build graphs.

Variant detection

Variant information of an individual or population can be deduced from the structure of the colored de Bruijn graph. Here, we propose a node-based variant detection algorithm.

In practice, the traversal starts from the end node of the graph and against the direction of the edge. The final offset value can be obtained using simple subtraction from the preoffset value shown in Fig. 1c.

We choose reverse traversal because the appearance of each source node means that a variant occurs. If it is a base substitution, the offset value is not affected; however, if it is an indel, using traversal along the edge direction, the offset break node is the sink node (node *GGG* in Fig. 1b) instead of the source node (node *TCA*) we want. In this paper, we only discuss small indels (< 50 bp).

In the colored de Bruijn graph, we assume that the variant occurrence corresponds to the source node one-to-one, so we discuss variants near the source node. When traversing the graph, all the nodes' variant information is stored in the offset value; thus, the variant, source node, and offset value are closely linked.

Here, we propose a **node-based** method:

(1) Save the maximum gap length and color near each source node to obtain gap information;
(2) Analyze the two types of source nodes (with reference and without reference) in turn, determine the variant type of each source node (nested variant detection algorithm (NVD)), and modify the gap at each source node to determine the node variant;
(3) Determine the relations between the nodes' position on the reference genome and graph (reference position determination (RPD)); then, transform the node variant to a locus variant (Fig. 3).

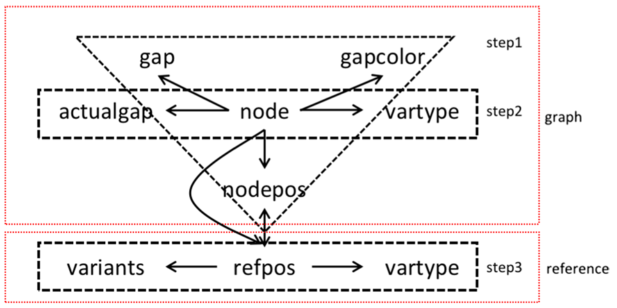

Fig. 3 Variant identification and variable transformation

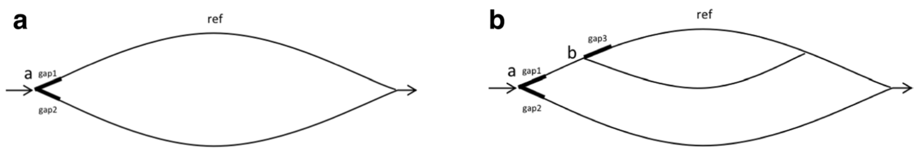

Fig. 4 Different gap distributions near the source node

Nested variant detection (NVD) algorithm

Step 1: Detect variant types of source nodes involving the reference

We traversed the cSupB id from small to large and only discuss the cSupB containing the reference genome. A simple situation is shown in Fig. 4a. If gap1 > 0, $var(a) = INS$; if gap2 > 0, $var(a) = DEL$; and if gap1 = gap2 = 0, $var(a) = SNP$. Here, function *var* denotes the variant type of the node. For the more general case, for example, in Fig. 4b, if the variant types at nodes *a* and *b* are the single nucleotide polymorphism (SNP) and insertion (INS), respectively, the actual gap conditions are gap2 > 0 and gap3 > 0 based on the traversal against the direction of the edge. Therefore, since the correct results cannot be obtained by considering the gap of a node such as *a* independently, we need to determine the node variant type more comprehensively. For detailed algorithm flows, see the NVD-1 flow chart in Additional file 1.

Step2: Detect variant types of other source nodes

This step traverses the cSupB id from small to large again and performs variant analysis on cSupB whose variant type has not been determined, that is, cSupB that does not contain the reference genome. The basic idea is to search the nearest parent source node containing the reference according to the cSupB tree structure when encountering a source node whose variant type has not yet been determined and record all the source nodes involved in turn. Then, by ordering from the parents to children, we can determine the variant types of all source nodes (see Additional file 1: father-children variant detection (FCVD)). For detailed algorithm flows, see the NVD-2 flow chart in Additional file 1.

Reference position determination (RPD) All the source nodes can be divided into two types: source nodes contained in the reference genome or those that are not. For the former, we used a variable *refpos* to record its position in the reference genome. Therefore, the focus here is to infer the reference genome position of the second type of source nodes and finally obtain the approximate correspondence between the graph position *nodepos* and the *refpos*.

The basic idea here is similar to the NVD algorithm, and we still traverse the cSupB id from small to large. When encountering a source node in which the *refpos* has not been determined, we need to obtain its nearest parent source node that contains the reference according to the cSupB tree structure and record all the source nodes involved in turn. Then, following the ordering from parents to children, source nodes' *refpos* can be determined, and the relationship between *refpos* and *nodepos* can also be obtained. Finally, we can simply obtain the variant at the reference genome site combined with the node variant information. For algorithm details, see the RPD flow chart in Additional file 1.

Construction of the coordinate system

In the linear coordinate system, a simple positive integer a and binary (a, b) can be represented by the base and sequence information. The biological relationship between sequences can also be discussed. However, this representation is not suitable for genome graphs. Here, based on the cSupB tree model, we constructed a haplotype pangenome coordinate system.

We define a triple *(position, bubid, basecolor)* to represent each base location (BL) in a graph. Since every sample contained in any cSupB has only one path, this representation provides a one-to-one correspondence to nodes in the graph. The *position* represents the offset value of the bases, and *bubid* represents the smallest cSupB where the base is located. *basecolor* represents the samples that include this base at this position, which has two representations. One is to use a string consisting of '0' and '1', and its length is equal to the number of samples. This representation is same as the node or edge color. The other is to randomly select a sample id containing the base. The former is more comprehensive, and the latter is simpler: the selection can be made according to the application. In this triplet, *position*, *bubid* and *basecolor* provide numerical, topology and sample information, respectively.

Similar to the linear coordinate system, we discuss the sequence information. The sequence mapped to the graph is a path, which is represented by a six-tuple (path location (PL)), as follows:

$$(startpos, startbub, endpos, endbub, pathbub, pathcolor)$$

The *startpos* and *endpos* represent the offset values of the start and end nodes of the path, and *startbub* and *endbub* represent the smallest cSupB where the start and end nodes are located, respectively. *pathbub* represents the smallest cSupB that contains all the paths. *startbub* and *endbub* are both child bubbles of *pathbub*. If the path spans n root cSupBs, *pathbub* is recorded as $-n$. The *pathcolor* represents the color of the path, which is represented as a 0–1 string. When the length of a path is equal to 1, the path is a base, $startpos = endpos$, $startbub = endbub = pathbub$, and $pathcolor = basecolor$. Then, the six-tuple of the path becomes a three-tuple of bases.

In the linear coordinate system, if two intervals are given, there are three kinds of relations: separation, intersection, and inclusion. Similarly, on the genome graph, we can provide the correlation between two paths.

Given two random paths, path1: *(a1, bub1, b1, bub2, bub3, color1)* and path2: *(a2, bub4, b2, bub5, bub6, color2).*

1. If there is no intersection between [*a1, b1*] and [*a2, b2*], path1 and path2 are separated;
2. If the relationship between [*a1, b1*] and [*a2, b2*] is inclusion and *color1* and *color2* also have an inclusion relationship, then path1 and path2 have an inclusion relationship;
3. In other cases, path1 and path2 intersect.

Specifically, if [*a1, b1*], [*a2, b2*] have an intersection and *color1* and *color2* do not intersect, then path1 and path2 intersect. At this time, sequence similarity analysis can be performed. According to the cSupB tree, we can find the nearest parent bubble *bub7* of *bub3* and *bub6* and its color *color3*. If *bub7* does not exist, the path spans the root cSupB, and *color3* includes all samples; if *bub7* exists, similarity analysis can be performed in *bub7*, and the union of *color1* and *color2* is a subset of *color3*.

Taking Fig. 1c as an example, we know that there are five cSupBs bub1.$< TAT, ACC, 111 >$; bub2.$< GGG, GTA, 110 >$; bub3.$< CAC, GTA, 011 >$; bub4.$< TCA, GGG, 110 >$; and bub5.$< TCA, GTA, 111 >$. We randomly select a node such as TCA, and its base position is (3, 4, 111) in which 3, 4, and 111 denote TCA's offset value, smallest cSupB id and base color, respectively. Similarly, we can also obtain other nodes' base positions such as ATG: (13, 1, 100), CCC: (17, -1, 111) and GGG: (7, 2, 110). Then, we randomly select three paths: *a*. CAGGGTGTA- > (5, 4, 11, 2, 5, 100); *b*. GGGAGTA—> (7, 2, 11, 2, 2, 010); and *c*. ATAACCC- > (13, 1, 17, 1, 1, 011). Taking path *a* as an example, we give a brief introduction to its components. The first node of path *a* is CAG, whose base position is (5, 4, 100), and the last node GTA's base position is (11, 2, 111). All the nodes on path *a* are contained in cSupB5, and their intersection of node colors is 100. Here, path *a* and path *b* intersect because they have common nodes such as GGG, and path *a*(*b*) and path *c* are separated because there is no intersection between [5, 11] ([7, 11]) and [13, 17].

Based on the genome graph coordinate system and variant detection algorithm, we can achieve more functions. For example, given a genome annotation file (e.g., gene transfer format (gtf), general feature format (gff)), we can obtain the relations between the annotation information and the topological structure and predict all the variants; if given a variant file (e.g., variant call format (vcf)), we can not only find the topological information based on the variants but also predict the variant information and compare it to the variant file to explore further discoveries.

Results

We evaluated the method performance on four different datasets described below. For this evaluation, we discussed the performance of graph decomposition, the mapping between graph and variant (annotation) files, variant detection, and cycle processing with dataset 1. To further evaluate the variant detection method, we compared precision and recall rate with other tools with dataset 2. In addition, we also assessed the computational ability for large-scale genomes with simulated dataset 3 and dataset 4.

Datasets

The human mitochondrial genome has the following advantages: (1) the high variant rate ensures that a large number of variants can be used; (2) the rich genome data contains almost all common and abnormal situations, and we can select the required data according to the concrete purpose; (3) the quality of the mitochondrial genome is high, which is consistent with our assumption that the genome represents the true genetic information of the individual. In addition, other research results, such as annotations, may help verify the accuracy of the model.

Four datasets were chosen to test and evaluate the performance of *cSupB*. Dataset 1: 12 human mitochondrial genomes, including a reference genome [42]. Sample names are AP008459, AP010675, EF153784, EF153791, EF153794, EF153814, EF397559, EU007868, FJ493500, GQ895144, GU377085, and NC_012920. Dataset 2: A series of random simulations with 10 samples for each time. Dataset 3: 5000 randomly simulated human mitochondrial genomes. Dataset 4: 20,000 randomly simulated human mitochondrial genomes. The simulation method is provided in Additional file 1.

Implementation of the tri-tuple coordinate system

Given 12 human mitochondrial genomes (see Additional file 2), we calculated that the longest repeat length of the samples was 16, except that of AP010675, which was 20, and its repeat region was TATAGCACCCCCTCTACCCCCTCTACCCCCTCTA, in which two repeats intersected. When the k value is greater than 20, the constructed colored de Bruijn graph is a DAG; otherwise, the constructed graph is a directed cyclic graph. Here, we set k = 22 for VARI-merge to construct an acyclic graph. In addition, there was a degenerate base N at the 3107th position in the reference genome, and we replaced N with C to avoid affecting the annotation information.

Using the postorder-like traversal strategy, we traversed the colored de Bruijn graph only once to find all cSupBs and their affiliation. In total, 118 cSupBs were identified, including 74 root cSupBs (65 simple cSupBs) and the largest cSupB tree, whose level was 5. These results are shown in Table 1, and more information can be obtained from the CST file.

After obtaining the necessary information about cSupBs, we can collect more detailed information about each cSupB stored in the CSDI file. This additional information includes the node/edge/branch/supernode count, sample length, and branch information.

Table 1 Comparison of the graph decomposition results of two k values

k	Graph	All cSupBs	Root CSupB	Max level	Simple cSupB
22	Graph	118 (74 + 20 + 14 + 9 + 1)	74	5	91 (65 + 26)
18	Graph1	55 (37 + 8 + 6 + 4)	37	4	44 (33 + 11)
	Graph2	64 (37 + 15 + 7 + 4 + 1)	37	5	47 (32 + 15)

Table 2 The statistics of cSupBs involved in gene regions

cSupB included	0	1	2	3	4	5	6	8	
Count	21	5	1	3	1	3	2	1	37

Table 3 Node variant statistics

Vartype	1	2	3	4	
Count	82	3	6	1	92

Table 4 SNV statistics

Vartype	1	2	3	4	
Count	82	9	17	2	110

Table 5 SNVs that cannot be found in the annotation files but that actually exist

Position	Ref	Alt	Vartype	Involved sample
97 ~ 102	GCTGGA	-	Deletion	EF153794
449	T	C	SNP	EF397559
632	C	T	SNP	GQ895144
3213	A	G	SNP	EF153794
4658	A	G	SNP	EF153784
6101	C	T	SNP	EF153794
8290–8299	G	CCCCCTCTAG	Insertion	AP010675
13,086	C	T	SNP	EF153784
15,142	C	T	SNP	FJ493500
15,814	A	G	SNP	EF153784

We can also map the existing annotation file to the graph: 578 of 3892 variants in the mtDNA vcf file are located in cSupB regions, and the others are located on bridges. Additionally, 37 genes in the gtf file are located on the bridges or span at least one cSupB, as shown in Table 2. All the variants and gene coordinates can be seen in vcf2graph.txt (see Additional file 3) and gtf2graph.txt (see Additional file 4).

While obtaining the cSupB relations, we detected the variants using the node-based variant detection algorithm. The statistical results of the variants are shown in Tables 3 and 4. Here, vartype indicates different variant types (1: substitution, 2: deletion, 3: insertion, 4: indel (unsure 2 or 3)).

Compared to the mtDNA annotation file downloaded from 1000 Genomes (ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/release/20130502/ALL.chrMT.phase3_callmom-v0_4.20130502.genotypes.vcf.gz), 89 of 110 single nucleotide variants (SNVs) can be location mapped. Moreover, 20 of 21 unmapped images can be verified for accuracy, as shown in Table 5.

The overall flow includes graph construction, decomposition, reorganization, and variant determination, and the total calculation time is less than 1 min. To further assess the

Table 6 Calculation time for two large-scale datasets

Samples	5000	20,000
cSupB	7.5 h + 3 min	30.5 h + 8 min

Table 7 Parameters of the simulation

Part	Experimental parameter	Para1:SNP	Para2: deletion	Para3: Insertion	Para4: k value	Set (groups*repeats)
1	SNP	100–2000 (0.6–12.0%)	10(~ 0.06%)	10(~ 0.06%)	28	20*5
2	Deletion	100(~ 0.60%)	5–100 (0.03–0.60%)	0	24	20*5
3	Insertion	100 (~ 0.60%)	0	5–100 (0.03–0.60%)	24	20*5
4	k value	500(~ 3.0%)	50(~ 0.3%)	50(~ 0.3%)	20–60	22*10

computational ability for large-scale genome data, we simulated 5000 and 20,000 mitochondrial genomes. The calculation time shown in Table 6 is divided into two components: graph construction and analysis. Our algorithm is minimally constrained by the sample size; allowing for constructing the graph model we need, we can quickly obtain the decomposition result.

Assessment of the variant detection accuracy

Two factors are related to the node-based variant detection algorithm: the number of variants, such as SNP, DEL, and INS, and the k value. Therefore, we used simulated data based on the different parameters mentioned above to evaluate the model. To assess the variant detection results, we used multiple sequence alignment tools, including multiple sequence comparison by log-expectation (MUSCLE) [43], multiple sequence alignment based on fast Fourier transform – GINSI alignment (MAFFT-GINSI) [44], Clustal Omega [45], and Mugsy [46], to identify SNVs, calculate the precision and recall rate, and compare the results. All these tools use a single CPU thread.

Our simulation can be divided into four parts based on the experimental parameters, and the detailed parameters are listed in Table 7. For each set, we simulated 10 human mitochondrial genomes, including a reference genome. Here, precision and recall are calculated by means of location-mapped variants (Fig. 5), and the results used by type-mapped variants are shown in Additional file 1.

The node-based variant detection algorithm has high accuracy. For each simulation and each parameter, there is little difference between the various methods. In some sets, our method has the best precision and recall rate. In addition, our method has the best stability.

Indel has a greater impact than SNP. Additionally, Fig. 5a, b show that as the number of SNPs increases, the precision and recall increase gradually. However, simulations of deletion and insertion show the opposite trend, and the indel variable has a significant impact on the results (Fig. 5c, d, e, f). Theoretically, if there are only SNP variants, the precision of our method is 100%.

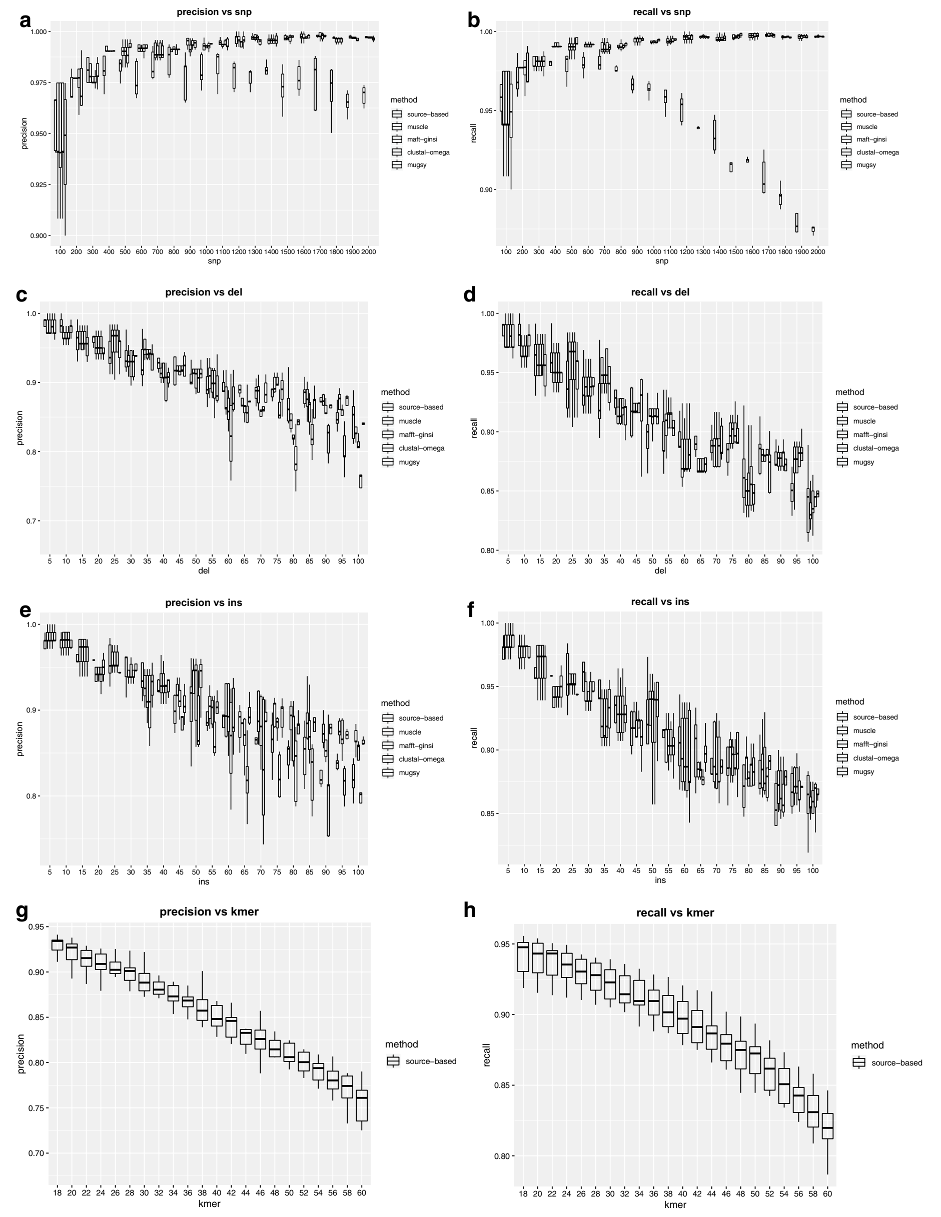

Fig. 5 Precision and recall for different simulation parameters

The k value is closely related to the structure of the graph: as the k value increases, the precision and recall decrease continuously (Fig. 5g, h). The reason is that when the k value increases, the k-mer becomes longer, the opportunity for node merging increases, and the structure of the graph becomes loose, which leads to a reduction in the source node number. Therefore, regardless of other factors, when we select the k value, the smaller the k value is, the better.

Furthermore, as the variants continue, the precision and the recall decrease. We know that indel variants have a great negative effect on precision and recall. In a certain range, the precision and recall increase with SNV variants, but the recall decreases sharply when the SNV mutation rate is outside of this range. The main reason is that the variant

Table 8 Basic information about the two cycles

Id	cycle_start_node	cycle_end_node	cycle_start_pos	cycle_end_pos	Half-visited nodes	Type
1	199	71	405	16,288	5	II
2	8532	11,846	8282	8290	1	I

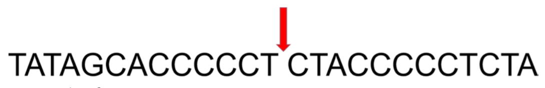

Fig. 6 Final cutting region and reference cut position

results cannot be mapped to the graph due to the decrease in sequence similarity; that is, the source number is far less than the variant number, which is not in line with our primary hypothesis of direct variant and source one-to-one correspondence. Thus, if the sequence similarity is insufficient, the variant detection model is not applicable.

In short, the higher the sample similarity is, the higher the precision and recall of SNVs for the node-based variant detection method.

Cycle processing

To test the reliability of cycle processing, we used dataset 1 as an example and set k = 18. After analyzing the graph, we found two cycles, as shown in Table 8.

Selection of the cutting point.

a. Combine adjacent cutting points of the same type and calculate the *R*-value. Here, cycle 1 has a total of 145 integrated cutting areas, in which the numbers of the three types of points are 73, 66, and 6. The *R*-value ranges from 0.5 to 1236; cycle 2 has only one type I cutting area, and the length is 26.
b. Cutting point selection. Here, cycle 2 is included in cycle 1, and the cutting area of cycle 2 is part of cycle 1. The *R*-value of cycle 1 in this cutting area is 1236, which is the most ideal cutting area. Therefore, we chose only one cutting area of cycle 2 as the final cutting area, output the base information and position information of the cutting area as the reference position, and select the middle position as the reference cutting point, as shown in Fig. 6.

Sequence cutting and graph reconstruction

Next, the segment of the cutting region is searched in the original sequence file. Then, we placed the sequences in the same area into the same directory. For each colored de Bruijn graph we constructed, we ensured that all sequences had the same head and tail. Therefore, before reconstructing the colored de Bruijn graph from the cutting sequences, we need to supply the same head and tail; that is, if the head/tail is missing, we added the appropriate head/tail. After the sequence preparation is complete, the previous construction steps are followed to rebuild the colored de Bruijn graph.

The statistical results are shown in Table 1. Notably, sequence cutting does not substantially impact the topological structure of a graph.

Discussion

We present *cSupB*, which is an implementation of graph decomposition and reorganization, variant detection, and coordinate system design of the colored de Bruijn graph. First, we constructed a set of population haplotype genomes into a DAG with a unique start and end node. On the basis of the graph, we proposed a cSupB data structure that incorporates sample information, such as node sources and path linkages, while inheriting the features of superbubbles. Then, we can quickly obtain cSupB structure trees through traversing, matching and relationship determination. Moreover, to describe the node location information, we introduced the offset value to measure the length of the longest path from the start node to each node. Ultimately, we proposed a node-based algorithm to identify variants in specified samples and construct a genome graph tri-tuple coordinate system.

Despite the relative effectiveness of the reference as a coordinate system, using the reference as a lens to study all other genomes introduces a pervasive reference bias, especially in the population genome era. Our coordinate system, which satisfies monotonicity, readability, and spatiality, provides a means to discuss the genome graph in terms of topologic structure to genetic element location. Indeed, the number of sequences is amenable to a global or local directed acyclic representation, and our algorithm is compatible with any directed acyclic colored graph; thus, it has wide application prospects. Additionally, we can discuss the relationships of any paths or samples quickly without reconstructing and querying once the acyclic graph is constructed. The coordinate system can also handle repetitive events. By selecting appropriate k values or cutting sequences, we can transform genomes into one or more directed acyclic colored de Bruijn graphs. However, the method of sequence cutting has certain limitations; that is, for highly repetitive sequences, especially tandem repeats, the cutting method will make the sequence too fragmented, thereby increasing the computational difficulty and error.

The essential assumption of the node-based variant detection algorithm is that the generation of the variant corresponds to the source node in a one-to-one manner. Only the gap information and the cSupB structure tree near the source node are needed to obtain the final variant information efficiently. We demonstrated that when the similarity of the sample decreases, the variant type becomes complicated, and the precision of the algorithm is reduced. Therefore, this approach is applicable only when the sample has a high similarity and the variant type is relatively simple. In addition, the choice of the k value impacts the accuracy of the algorithm, and we preferred a smaller k value to ensure acyclicity.

With the introduction of more chromosome-level and haplotype-resolved assemblies, our decomposition and reorganization method could also be effectively handled. Moreover, this method is not limited to the VARI/VARI-merge tool, even the colored de Bruijn graph, but can theoretically be generated for any colored DAG. Compared to other motifs, such as ultrabubbles, the cSupB we proposed overcomes the limitation of topology but cannot handle more complex situations. Therefore, truly universal and comprehensive spatial framework research for genomic graphs will become a focus in the future. We anticipate that these studies will allow graph research to become a more practical analysis approach and increase the accessibility of existing possibilities.

Nevertheless, our model has several shortcomings. First, although graph research has steadily improved in recent years, almost all colored de Bruijn graph construction tools, including the VARI-merge used in this paper, do not have functions such as mapping, which significantly limits data selection and model promotion. Second, if the length of the gene element is less than k, there may be multiple corresponding nodes on the graph, which makes the analysis complicated and ambiguous. Third, we cannot handle degeneracy efficiently. Fourth, our variant detection method could only address small indels, but for more complicated conditions, such as large indels, inversions, and reverse tandem duplications, we do not have an effective solution; thus, we expect to introduce more high-quality genomes and comprehensive tools to perform genome graph research in the future.

In short, we anticipate that the genome graph research model presented in this paper will not only be compatible with more abundant information but also introduce additional computational methods to explore potential biological functions. We believe strongly that with the arrival of the population genome era, genome graphs and their spatial structure will be deeply studied and will play an important role in the research on population genomic diversity.

Abbreviations

BL: Base location; cSupB: Colored superbubble; CCP: Cycle cut position; CSDI: CSupB detailed information; CST: CSupB topology; DAG: Directed acyclic graph; NNI: Node nearby information; NVD: Nested variant detection; SNP: Single nucleotide polymorphism; SNV: Single nucleotide variants; OVI: Offset value information; PL: Path location; RPD: Reference position determination.

Supplementary information

The online version contains supplementary material available at https://doi.org/10.1186/s12859-021-04149-w.

Additional file 1 Proofs, file format details, variant detection method, simulation method and additional variant accuracy calculation results.

Additional file 2 Twelve human mitogenomes used in the article.

Additional file 3 The variant position coordinates on the graph in the mitochondrial genomevariant call format (vcf) file

Additional file 4 The gene region coordinates on the graph in the mitochondrial genome genetransfer format (gtf) file.

Acknowledgements

We cordially thank the anonymous reviewers for their invaluable comments. We also thank all members of the Lin lab for their valuable suggestions.

Authors' contributions

JDG developed and implemented the method and drafted the manuscript. ELP, HTS and KL contributed to the design of the study and the writing of the article. All authors read and approved the final manuscript.

Funding

This work was supported by the State Key Basic Research and Development Plan (2017YFA0605104) and a key project of the State Key Laboratory of Earth Surface Processes and Resource Ecology.

Availability of data and materials

The datasets analyzed and the C++ program implemented during the current study are available at https://github.com/eggleader/cSupB.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 31 January 2021 Accepted: 25 April 2021
Published online: 27 May 2021

References

1. Rhoads A, Au KF. PacBio sequencing and its applications. Genomics Proteomics Bioinformatics. 2015;13(5):278–89.
2. van Dijk EL, Jaszczyszyn Y, Naquin D, Thermes C. The third revolution in sequencing technology. Trends Genet. 2018;34(9):666–81.
3. Kucherov G. Evolution of biosequence search algorithms: a brief survey. Bioinformatics. 2019;35(19):3547–52.
4. Zekic T, Holley G, Stoye J. Pan-genome storage and analysis techniques. Methods Mol Biol. 2018;1704:29–53.
5. Computational Pan-Genomics C. Computational pan-genomics: status, promises and challenges. Brief Bioinform. 2018;19(1):118–35.
6. Dreux C, Maquart FX, Bonnefont-Rousselot D, Delpech M, Gueant JL, Le Bouc Y, Massoubre B, Porquet D, Rives N, Vigneron C. Medical biology in the face of the evolution of health care needs. Ann Biol Clin (Paris). 2018;76(5):485–91.
7. Azarian T, Huang IT, Hanage WP: structure and dynamics of bacterial populations: pangenome ecology. In: *The Pangenome: Diversity, Dynamics and Evolution of Genomes.* Edited by Tettelin H, Medini D. Cham (CH); 2020: 115–128.
8. Brockhurst MA, Harrison E, Hall JPJ, Richards T, McNally A, MacLean C. The ecology and evolution of pangenomes. Curr Biol. 2019;29(20):R1094–103.
9. Ameur A. Goodbye reference, hello genome graphs. Nat Biotechnol. 2019;37(8):866–8.
10. Rakocevic G, Semenyuk V, Lee WP, Spencer J, Browning J, Johnson IJ, Arsenijevic V, Nadj J, Ghose K, Suciu MC, et al. Fast and accurate genomic analyses using genome graphs. Nat Genet. 2019;51(2):354–62.
11. Pevzner PA. 1-Tuple DNA sequencing: computer analysis. J Biomol Struct Dyn. 1989;7(1):63–73.
12. Alekseyev MA, Pevzner PA. Breakpoint graphs and ancestral genome reconstructions. Genome Res. 2009;19(5):943–57.
13. Paten B, Novak AM, Eizenga JM, Garrison E. Genome graphs and the evolution of genome inference. Genome Res. 2017;27(5):665–76.
14. Rosen Y, Eizenga J, Paten B. Modelling haplotypes with respect to reference cohort variation graphs. Bioinformatics. 2017;33(14):i118–23.
15. Rautiainen M, Marschall T. GraphAligner: rapid and versatile sequence-to-graph alignment. Genome Biol. 2020;21(1):253.
16. Rasko DA, Rosovitz MJ, Myers GS, Mongodin EF, Fricke WF, Gajer P, Crabtree J, Sebaihia M, Thomson NR, Chaudhuri R *et al*: The pangenome structure of Escherichia coli: comparative genomic analysis of E. coli commensal and pathogenic isolates. *J Bacteriol* 2008, **190**(20):6881–6893.
17. Jandrasits C, Dabrowski PW, Fuchs S, Renard BY. seq-seq-pan: building a computational pan-genome data structure on whole genome alignment. BMC Genom. 2018;19(1):47.
18. Kim D, Paggi JM, Park C, Bennett C, Salzberg SL. Graph-based genome alignment and genotyping with HISAT2 and HISAT-genotype. Nat Biotechnol. 2019;37(8):907–15.
19. Rand KD, Grytten I, Nederbragt AJ, Storvik GO, Glad IK, Sandve GK. Coordinates and intervals in graph-based reference genomes. BMC Bioinformatics. 2017;18(1):263.
20. Nguyen N, Hickey G, Zerbino DR, Raney B, Earl D, Armstrong J, Kent WJ, Haussler D, Paten B. Building a pan-genome reference for a population. J Comput Biol. 2015;22(5):387–401.
21. Gartner F, Honer Z, Siederdissen C, Muller L, Stadler PF. Coordinate systems for supergenomes. Algorithms Mol Biol. 2018;13:15.
22. Heng Li XF, Chong C: The design and construction of reference pangenome graphs. *Method* 2020.
23. Onodera T, Sadakane K, Shibuya T. Detecting superbubbles in assembly graphs; 2013.
24. Paten B, Eizenga JM, Rosen YM, Novak AM, Garrison E, Hickey G. Superbubbles, ultrabubbles, and cacti. J Comput Biol. 2018;25(7):649–63.
25. Zerbino DR, Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome Res. 2008;18(5):821–9.
26. Iqbal Z, Caccamo M, Turner I, Flicek P, McVean G. De novo assembly and genotyping of variants using colored de Bruijn graphs. Nat Genet. 2012;44(2):226–32.
27. Iqbal Z, Turner I, McVean G. High-throughput microbial population genomics using the Cortex variation assembler. Bioinformatics. 2013;29(2):275–6.
28. Conway TC, Bromage AJ. Succinct data structures for assembling large genomes. Bioinformatics. 2011;27(4):479–86.
29. Chikhi R, Rizk G. Space-efficient and exact de Bruijn graph representation based on a Bloom filter. Algorithms Mol Biol. 2013;8(1):22.
30. Chikhi R, Limasset A, Jackman S, Simpson JT, Medvedev P. On the representation of de Bruijn graphs. J Comput Biol. 2015;22(5):336–52.

31. Marcus S, Lee H, Schatz MC. SplitMEM: a graphical algorithm for pan-genome analysis with suffix skips. Bioinformatics. 2014;30(24):3476–83.
32. Holley G, Wittler R, Stoye J. Bloom filter trie: an alignment-free and reference-free data structure for pan-genome storage. Algorithms Mol Biol. 2016;11:3.
33. Muggli MD, Bowe A, Noyes NR, Morley PS, Belk KE, Raymond R, Gagie T, Puglisi SJ, Boucher C. Succinct colored de Bruijn graphs. Bioinformatics. 2017;33(20):3181–7.
34. FatemehAlmodaresi PP, Rob P. Rainbowfish- a succinct colored de bruijn graph representation. Bioinformatics. 2017;18:1811–5.
35. Almodaresi F, Sarkar H, Srivastava A, Patro R. A space and time-efficient index for the compacted colored de Bruijn graph. Bioinformatics. 2018;34(13):i169–77.
36. Almodaresi F, Pandey P, Ferdman M, Johnson R, Patro R. An efficient, scalable, and exact representation of high-dimensional color information enabled using de bruijn graph search. J Comput Biol. 2020;27(4):485–99.
37. Holley G, Melsted P. Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs. Genome Biol. 2020;21(1):249.
38. Muggli MD, Alipanahi B, Boucher C. Building large updatable colored de Bruijn graphs via merging. Bioinformatics. 2019;35(14):i51–60.
39. Brankovic L, Iliopoulos CS, Kundu R, Mohamed M, Pissis SP, Vayani F. Linear-time superbubble identification algorithm for genome assembly. Theoret Comput Sci. 2016;609:374–83.
40. Sung WK, Sadakane K, Shibuya T, Belorkar A, Pyrogova I. An O(m log m)-time algorithm for detecting superbubbles. IEEE/ACM Trans Comput Biol Bioinform. 2015;12(4):770–7.
41. Gartner F, Muller L, Stadler PF. Superbubbles revisited. Algorithms Mol Biol. 2018;13:16.
42. Andrews RM, Kubacka I, Chinnery PF, Lightowlers RN, Turnbull DM, Howell N. Reanalysis and revision of the Cambridge reference sequence for human mitochondrial DNA. Nat Genet. 1999;23(2):147.
43. Edgar RC. MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res. 2004;32(5):1792–7.
44. Katoh K, Misawa K, Kuma K, Miyata T. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. Nucleic Acids Res. 2002;30(14):3059–66.
45. Sievers F, Higgins DG. Clustal Omega for making accurate alignments of many protein sequences. Protein Sci. 2018;27(1):135–45.
46. Angiuoli SV, Salzberg SL. Mugsy: fast multiple alignment of closely related whole genomes. Bioinformatics. 2011;27(3):334–42.

Publisher's Note