The aim of the algorithm is to produce alternate sequences, from an input segment of reference sequence (active region) and a set of reads. First, the active region and the set of reads are transformed into a graph very similar to the de Bruin one. The structure of the graph is then optimized (e.g. less populated areas are removed). Finally, the alternate sequences are extracted from the the graph by finding all paths from its starting vertex to the ending one.

Graph Representation

Each vertex of the graph represents one k-mer present either in the reference sequence, or in some of the input reads. A *k-mer* is a sequence of k consecutive bases of fixed length . In the scope of this document, we add a special number to each k-mer, called a *k-mer context number.* The number allows distinguishing k-mers covering the same sequence of bases which is useful for when detecting and avoiding repeats within the reference sequence. The k-mer context number is not used when working with k-mers coming from the input read set – such k-mers have their context always set to zero.

Vertices

Each graph vertex stores the following information:

- the k-mer it represents (together with its context number),
- references to incoming and outgoing edges.
- Vertex type (whether it was created when processing the reference sequence or one of input reads).

References to all graph vertices are stored in a vertex table, a data structure that allows performing the following operations effectively:

- Map a given k-mer and its context number to corresponding vertex.
- Given a k-mer, return all vertices that represent k-mers covering the same base sequence (the context number is ignored). The list of vertices is sorted by the context number of the represented k-mers.
- Add a new k-mer-to-vertex mapping.
- Remove an existing mapping.

The graph contains two special vertices. The *starting vertex* represents a k-mer covering first k-1 bases of the reference sequence prefixed with **B** base. Similarly, the k-mer of the *ending vertex* covers the last k-1 bases of the reference, suffixed with **E** base. For example, when k-mer size is set to 5 (number of consecutive bases covered by one k-mer) and the reference starts with **ACGTC** and ends with **TGCGA**, the k-mer of the starting vertex covers sequence **BACGT** and the sequence for the ending vertex is **GCGAE**. The B and E bases never occur within a DNA sequence which is exactly the reason of their usage as start and end markers.

Edges

Each edge contains the following information:

- References to its source and destination vertices, representing its source and destination k-mers.
- Weight. The value usually corresponds to a number of reads covering the edge.
- Length. Denotes number of bases one must go forward within a read or reference sequence to obtain the sequence covered by the destination k-mer from the one represented by the source one.
- Type. Three types of edges are defined: reference edges, created when transforming the reference sequence to the graph, read edges, constructed when adding reads to the graph, and variant edges that cover information about two alternate sequences going between the source and destination k-mers. Read edges are also created when optimizing graph structure. Variant edges are usual results of resolving bubbles.
- Primary sequence. Covers bases starting after the end of the source k-mer and ending by the last base of the destination k-mer (non-inclusive). This information allows to reconstruct part of the reference sequence or read(s) covered by the edge. Edges of length 1 do not contain any sequence since they connect adjacent k-mers.
- Alternate sequence. Present only in variant edges. Covers alternate sequence coming through the edge. The meaning is de facto the same as in the primary sequence case.

Similarly to the vertex table, references to all edges are stored inside an edge table, a data structure that maps pairs of source and destination k-mers to corresponding edges and allows time-effective insertions and deletions of such mappings. Unlike the vertex one, the edge table does support any sort of queries ignoring k-mer context numbers.

Graph Construction

This section provides a step-by-step description of how the de Bruin-like graph is constructed from a given segment of reference sequence and a set of input reads. Certain aspects of the construction algorithm will be demonstrated by various figures, usually displaying the graph during certain construction phase. Such figures follow this color scheme:

- graph starting vertex is always yellow,
- blue color is used to draw graph ending vertex and edges of variant types,
- green is used for vertices and edges added to the graph during reference sequence parsing,
- edges and vertices added as a result of read processing are red.

The section also takes advantage of unified notation for certain global parameters:

- the graph being constructed is marked G_R,
- k denotes size of a k-mer (number of bases covered by the k-mer),
- l defines the length of the active region (length of the reference sequence segment),
- t stands for threshold value that is used to determine which parts of the graph are not supported enougth to be treated as important. Such parts may be removed completely.

- p denotes the length of a consecutive region covered by a part of a read. A read might not cover one consecutive region due to soft and hard clipping. Such a read may be divided into parts, also called subreads, each covering smaller but consecutive region.

Initialization and Preprocessing

Unlike algorithms performing global reassembly (some of them were presented in the beginning of this document), local reassembly needs some extra preprocessing to be done, especially for the input read set. It is necessary to identify reads that fit into the active region with at least some parts. That can be done by looking at their mapping position (POS) and length. It would be theoretically possible to use also reads that are either not mapped at all, or their mapping is a low-quality one, if they proved to be similar enough to other reads known to overlap with the active region. However, the algorithm presented in this document does not follow this approach. It selects only reads with sufficient mapping quality and that overlap with the active region. By default, mapping quality above 20 is considered as good enough.

Because one read might cover more than one consecutive region of the desired alternate sequence (due to clipping, both soft and hard), it is necessary to split it into parts for which the statement about one consecutive region holds. Such parts are called *subreads*. For each subread, a test is made to decide whether it overlaps with the active region. A read is filtered out if none of its parts overlaps. Additionally, only overlapping subreads are used in the further steps of the algorithm.

Reads can be divided into multiple subreads using information from their CIGAR strings. If the CIGAR string is not available, or does not contain any indication of clipping, the corresponding read is treated as covering one consecutive region; it already forms one subread. When the clipping information is available, the read is divided. For example, if the CIGAR string is equal to **8S45M9H89M**, the corresponding read mapped at position **X** is divided into these subreads:

- r_1, mapped at **X+8**, 45 bases long,
- r_2, mapped at **X+62**, 89 bases long.

When it is clear which of the input reads to use for the given active region, a real work begins. The reference sequence covering the active region is transformed into de Bruin-like graph G_R.

Reference Sequence Transformation

The process of transforming the active region into the graph G_R is quite straightforward. The reference sequence is divided into l-k+1 k-mers K_1, …, K_{l-k+1}. Two extra k-mers, K_0 and K_{l-k+2}, are added to represent start and end of the region. Figure 1 illustrates the situation on a sample active region starting with **ACGTTTGC** and ending with **CCGTAGCT** and with k-mer size set to 5 (k = 5). It may happen that two or more k-mers cover the same sequence. In such a case, their k-mer context number is used to distinguish them. Let's denote such k-mers as K_{s0}, …, K_{sn} where $s_0 < … < s_n$. The context number for K_{s0} is set to zero, K_{s1} obtains 1 and so forth until K_{sn} gets n - 1. K-mers that cover unique sequences (sequences not covered by any other k-mer) have their context number always set to zero.

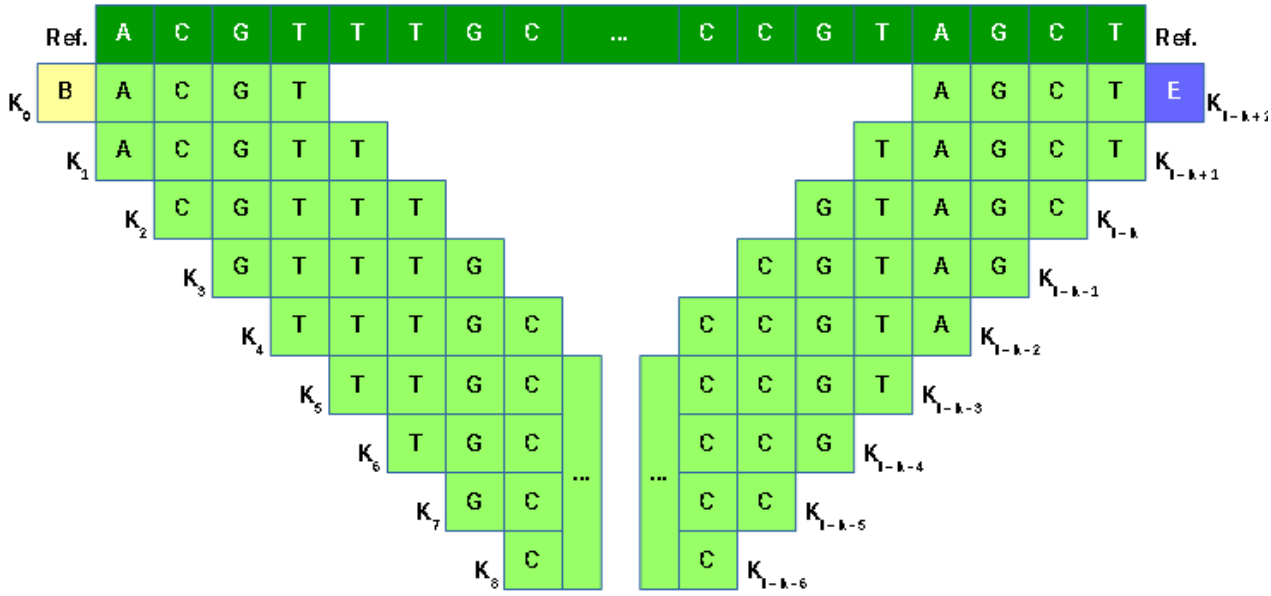

Figure 1: Dividing the active region reference into k-mers

For each of l-k+3 k-mers, a graph vertex is created to represent it. Then, adjacent k-mers are connected with oriented edges going along increasing k-mer indices. Formally speaking, the resulting state of the graph G_R looks like this:

$G_R = (V_R, ER)$

$V_R = \{v_i \mid v_i \text{ represents k-mer } K_i, i \backslash elem \{0, \ldots, l - k + 2\}\}$

$E_R = \{(v_i, v_{i+1}) \mid i \backslash in \{0, \ldots, l - k + 1\}\}$

Length of all the edges is 1 since they connect adjacent k-mers. The graph is also a linear one; input and output degree of all vertices, except the starting and ending one, is 1. All vertices are also marked as coming from the reference sequence. Figure 2 shows how such a graph would look like for an active region of **ACGTAAAA** (l = 8) and k = 3.

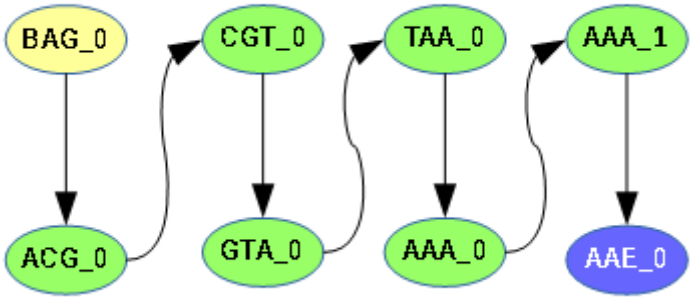

Figure 2: A sample graph obtained by transforming a reference sequence ACGTAAAA with k-mer size of 3

Adding Reads

The algorithm works rather with individual subreads than whole reads. Its individual steps are quite similar to the process of reference sequence transformation. The subread of length p is divided into p-k+1 k-mers (K_0, …, K_{p-k}) using exactly the same approach, except that no extra starting and ending k-mers are added. Further processing has to be different since the graph may contain more vertices the k-mers of which cover the same sequence.

For each k-mer K_i, a set of vertices V_i is created. The set contains all vertices of the graph covering the same sequence as K_i does. K-mer context numbers are ignored. If the k-mer does not appear within the active region or already processed reads, the corresponding V_i set would be empty. In that case, a new vertex, covering the k-mer, is added to the graph G_R and also inserted into V_i. The vertex is marked as created by a read. Some interesting properties holds for each V_i:

- all its vertices come from the same source (either reference sequence, or reads). The source is also referred as a type of the set.
- Reference V_i sets may contain one or multiple vertices, read V_i sets contains exactly one since k-mer context numbers are not used to differentiate k-mers present only in reads.
- If V_i and V_j contain vertices k-mers of which cover the same sequence, V_i is equal to V_j.

To make things more obvious, let's again have a reference sequence of **ACGTAAAA** and let's have a read of **ACGGAAAA** that maps on its beginning (its POS value is 1 and its MAPQ >= 20). Figure 3 shows contents of V_i sets constructed for the read when k = 3. In this example, the length of the read is the same as the size of the reference sequence. This is just a coincidence (in reality, reads are usually much shorter than the size of the active region).

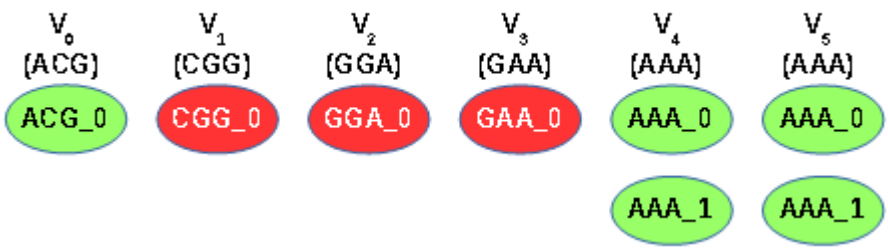

Figure 3: V_i sets for the read **ACGGAAAA** and reference sequence **ACGTAAAA** (k = 3)

To complete the addition of the read (or subread) in an ideal way, one needs to select exactly one vertex v_i from each V_i and connect these representants by oriented edges to form a path from V_0 to V_{p-k}. This is trivial for read V_i sets since they always contain only one vertex, however, it may be hard for those of reference type. There may exist quite many paths connecting a vertex in V_0 with one in V_{p-k} and from the information available, it is not clear which is the right one (or the most probable one). To filter out improbable paths, two heuristics may be applied:

- reduce sizes of the V_i sets by looking at reference edges connecting them together,
- choose best path based on the number of edges that need to be added to the graph G_R.

While the first heuristics does not guarantee existence of only one path, it may be sufficient in many active regions where repetitions are not counted as everyday events. The second approach always produces one path but may be quite challenging to tune it not to miss the right one.

Reducing Sizes of the V_i Sets

The heuristic is based on an assumption that when adjacent V_i sets are of reference type and are connected with reference edges, only these edges are candidates for covering the right path. The size of such sets may be reduced to vertices that serve as either inputs, or outputs to that edges. The size reduction continues until the whole round through all V_i sets has any effect.

Figure 4 shows how the heuristic may decrease number of possible paths between the starting and ending vertex of the graph (each such path denotes at least one alternate sequence). The upper part of the Figure shows the V_i sets from Figure 3 without applying the size reduction. In such a case, four different alternate sequences may be deduced from the resulting graph G_R as shown in Table 1. However, only one reference edge connects vertices **AAA_0** and **AAA_1**, so paths that do not use it are considered improbable. After the reduction is applied (bottom of Figure 4), only one path remains. Deleted edges were removed from the figure, deleted vertices are displayed in black.

Table 1: Paths deducible from the graph when no heuristics are applied

Path (edge numbers)	Sequence	Note
1, 2, 3, 4, 6, 7*	ACGGAAAAA	The path ends in **AAAA_0** but, so the corresponding alternate sequence must lead to the ending vertex). Also, goes twice across **AAA_0**.
1, 2, 3, 4, 7	ACGGAAAA	Is the right path
1, 2, 3, 5, 9, 7*	ACGGAAAAA	The path ends in **AAAA_0** but, so the corresponding alternate sequence must lead to the ending vertex). Goes twice accross **AAA_0**.
1, 2, 3, 5, 8	ACGGAAAA	Forms the right sequence but goes twice across **AAA_1**

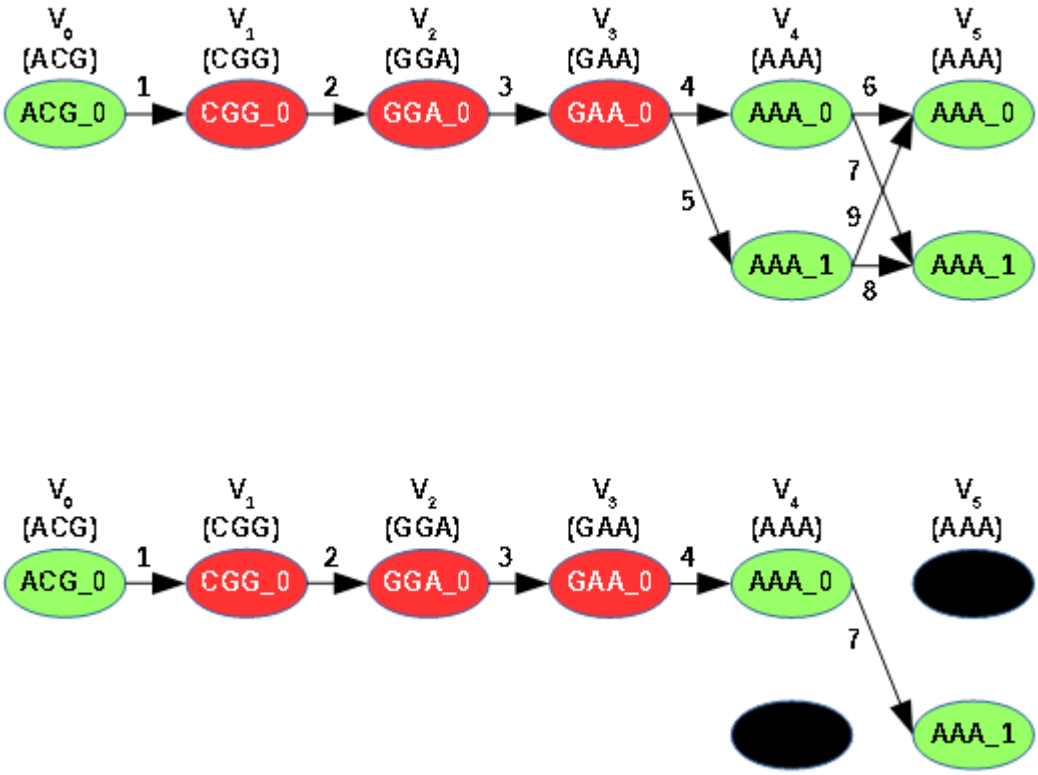

Figure 4: Reducing sizes of the V_i sets by examining the reference edges

Finding the Best Path

After application of this heuristics, all V_i sets contain exactly one vertex. Connecting the vertices with edges leading from V_i to V_{i+1} for i \elem {0,…,p – k - 1} produces the path by which the current subread commits to the graph G_R. The lesser edges needed to add to E_R, the better the path is. So, the task reduces to finding the shortest path connecting a vertex from V_0 with one from V_{p-k}. The graph G' = (V', E', w') on which the search runs, is formally defined as follows:

$$V' = \{v_{ij} \mid i \elem \{0,\ldots,p - k\}, j \elem \{0, \ldots, |V_i| - 1\} \}$$

$$E' = \{(v_{ij};v_{i+1m}) \mid i \elem \{0,\ldots,p - k - 1\}, j \elem \{0, \ldots, |V_i| - 1, m \elem \{0,\ldots,|V_{i+1}| - 1\}\}$$

The graph structure is a layered one. Two adjacent layers always form a complete bipartite subgraph of G'.

The edge evaluation function w' follows these rules (e = (u,v)):

- w'(e \elem E') = 0, if the graph G_R contains an edge leading from u to v,
- w(e \elem E) = 1, is no such edge exists,
- w'(e \in E') = 2, if the shortest path from a vertex in V_0 to the vertex u already contains vertex v. Because the graph G' is layered, layers processed after the one containing the vertex u cannot change such path. The reason for introducing this criteria is to penalize paths

that go through one vertex multiple times which should be avoided completely on reference vertices. Troubles may be caused if more than one shortest path from a vertex in V_0 to the vertex u exist and some of them do not contain v. So, this evaluation criteria would need additional adjustements.

When the shortest path is found, all its edges are inserted into the graph G, together with information about the subread they cover. If some of the edges already exist, only their read information is updated.

Read Connection Candidates

When processing a subread of length p, only two types of V_i sets may be created – either reference ones, or read ones. A sequence of V_i sets may contain a portion that satisfies the following:

- a, b, c, d \in N_0, a <= b < c < d,
- V_a,...V_b contains one read vertex each,
- V_{b+1},...V_c are of reference type,
- V_{c+1},…,V_d store read vertices.

So, the subread, within the (a, …, b) “interval”, covers an alternate sequence different from the reference one. Then, it goes together with the reference ((b+1, …, c)) and finally, differs from it again in (c+1, …, d). It is possible to create a shortcut and connect a vertex from V_b with one in V_{c+1} directly which has the effect displayed on Figure 5. These shortcuts are built during the phase of graph structure optimizations described in the next section. For now, all vertices that may happen to be inputs or outputs of a shortcut, are marked as *read connection candidates*; they can be used to connect two parts of the graph covered by one read directly. Such an optimization may, in some cases, reduce number of alternate sequences deducible from the resulting graph.

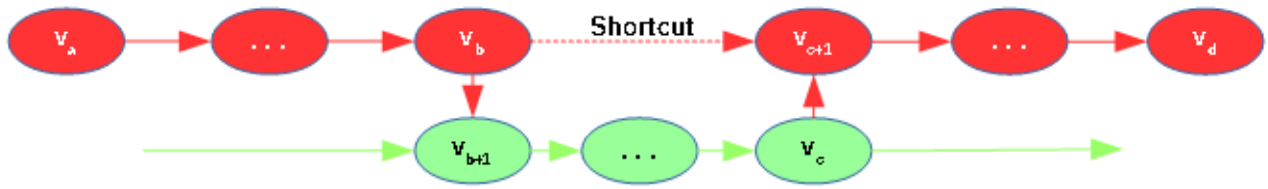

Figure 5: A shortcut connecting two parts of the resulting graph covered with the same read

Graph Structure Optimizations