Repeat resolving using paired data.

Abstract

In this paper we give a general definition for validness of repeat resolving methods based on paired information usage. We prove that rectangle graph technique is a honest repeat resolving method in case of error-free data and provide some ideas how to improve rectangle graph approach, and how can inexact mate-read gaps be overcame.

1 Motivation

Actually, most of sequencing machines provides not single reads, but mate-reads. It seems that mate-pair information can be used in more efficient way than actual assemblers do. It seems that right place for mate-pair information usage is repeat resolving step. But to discuss different methods of repeat resolving, based on paired information usage, we need to define what the correct repeat resolving method is and understand how we can compare quality different repeat resolving methods.

2 About paired info

Assume that we have a graph $G = (V, E)$ with lengths of all edges. G can be any abstract graph, but in case of genome assembling you may think that G is compressed be Bruijn graph. For given edge a we'll denote it's length as l_a.

Remark 1. *In case of compressed de Bruijn graph length of edge is the number of nucleotides in this edge minus* k. *So, by construction of compressed de Bruijn graph the shortest possible edge marked by* $k+1$ *nucleotides and have length equal one.*

Given a directed path in graph, we define distance between two edges as directed distance from start vertex of first edge to start vertex of second edge.

Definition 1. *Directed distance between two vertices* A *and* B *in directed path is*

- *sum of edge length's, for edges situated between these vertices in path, if* B *is after* A *in this directed path*
- *-distance(*B*,*A*) in other case.*

In this approach we assume that vertices have not length.

For each edge e, we'll define $start(e)$ and $end(e)$ as start and end vertex of edge e respectively. Length of path P is sum of length of it's edges (multiple occurrences allowed). We'll denote it as $|P|$. For such graph we define *edge pair* as ordered pair (a, b) where $a \in E$ and $b \in E$. *Edge pair info* is pair $((a, b), x)$ where (a, b) is edge pair, and $x \in \mathbb{Z}$. For given constant d and directed path P (from graph G), pair info $((a,b),x)$ with positive x is called d-*consistent edge pair info with respect to* P if there exists directed subpath P' of P, such that P' starts from a, ends in $start(b)$ and $|P'| - l_a < d$, $|P'| + l_b > d$, $x = |P'|$.

d-*set of edge pair info* is set S edge pair info such that there exists a Chinese postman cycle C (possibly not unique), satisfying following condition: $((a,b),x) \in S \Leftrightarrow ((a,b),x)$ is d-consistent with respect to cycle C.

2.1 Receiving paired info from reads

It seems to be a subject of another paper...

3 Repeat resolving

We want to define what is "correct" graph transformation due to paired information and provide methods to compare these graph transformations in terms of resulting graph complexity.

3.1 Correct and honest graph transformations

Definition 2. *For given graph* $G = (V, E)$*, we say that graph* $G' = (V', E')$ *is a* ungluing *of* G *if there exists surjective mappings* $f : V' \to V$ *and* $g : E' \to E$*,* f *and* g *satisfy following conditions:*

1. *Edge length in* G' *must be consistent with* g *i.e.* $\forall a \in E', l_a = l_{g(a)}$.
2. $\forall e' \in E' \quad f(start(e') = start(g(e'))$
3. $\forall e' \in E' \quad f(end(e') = end(g(e'))$

Remark 2. *For every* ungluing G' *of* G *we can define gluing operation in terms of [Pevzner 2004] such that* G *is glued* G'.

Definition 3. Compress operation *for graph* G *consist from:*

1. *Remove vertex* A *wit indegree and outdegree both equal to one.*
2. *Remove edges* a *and* b*, where* $end(a) = A \quad start(b) = A$.
3. *Add edge* $(start(a), end(b))$ *labeled by* ab.

Definition 4. *For given graph* $G = (V, E)$*, we say that graph* $G'' = (V'', E'')$ *is a* compressing *of* G *if* G'' *obtained by any number of compress operation from* G.

Definition 5. *For given graph* $G = (V, E)$*, we say that graph* $G'' = (V'', E'')$ *is a* compressed ungluing *of* G *if there exists ungluing* G' *of* G*, and* G'' *is compressing of* G'.

Definition 6. *For given directed path* $P' = (e'_0, \dots, e`_n)$ *in* G' *we define* decoding *of this path as* $(g(e'_0), \dots, g(e'_n))$ *and denote it as* $\mathrm{dec}(P')$.

Definition 7. *For given* d*-set* S *of edge pair info and cycle* C*,* C *is* valid *cycle iff* $((a,b),x) \in S \leftrightarrow ((a,b),x)$ *is* d*-consistent with respect to cycle* C.

We say that cycle $C' \in G'$ *is* valid *if* $\mathrm{dec}(C') = C$*,* C *is valid cycle in* G.

Definition 8. *If for given* d*-set* S *of edge pair info* $\mathrm{dec}(\cdot)$ *is a bijection between valid cycles in* G *and* G'*, we'll say that ungluing* G' *is* S-consistent ungluing.

Definition 9. *If for given* d*-set* S *of edge pair info* G'' *is a compressing of any* S*-consistent ungluing of* G*, we'll say that* G'' *is* S-consistent compressed ungluing.

Remark 3. *Note that "ungluing" is a graph notion that knows nothing about pair info. On the contrary, some ungluing* G' *can be* S*-consistent with respect to one* d*-set* S *of paired info and not* S*-consistent with respect to other.*

Remark 4. *Above definitions can be simply reformulated for non-cyclic genome.*

3.2 Informal reformulation

Every valid cycle in G corresponds to consistent with graph and paired information variant of genome. So, our condition means that after our transformation, every possibility for genome must be preserved, and no possible genomic cycle will transform to two different ones.

3.3 Graph complexity estimation - still under discussion!

Problem 1. *Given two* S*-consistent ungluings of* G*, we want to define which one is better.*

For every vertex $v' \in V'$, we'll define *vertex incoming-outgoing set* (and denote it as $O_{v'}$) as $\{(a,b) | a \in E, b \in E\}$, (a,b) satisfies following conditions:

1. $\exists a' \in E' : end(a') = v', g(a') = a$
2. $\exists b' \in E' : start(b') = v', g(b') = b$

We call vertex *simple* if indegree and outdegree of this vertex both equal one. Let's denote the set of simple vertices of graph G' by $U_{G'}$. *Graph incoming-outgoing set* is $\bigcup\limits_{v' \in E' \setminus U_{G'}} O_{v'}$. *Measure of complexity* for G' is size of G' graph incoming-outgoing set. The above complexity for graph G is the same as the complexity of any compressing of G.

There are many other possible measure, for example, minus sum of all edges length.

4 Connection with rectangle graph

Definition 10. Vertex pair info *is a triple* (A, B, x)*, where* A *and* B *are vertices and* $x \in \mathbb{Z}$

Definition 11. Vertex-edge pair info *is a triple* (A, b, x)*, where* A *is vertex,* b *is edge and* $x \in \mathbb{Z}$

Note: labels in rectangle graph can be considered as vertex pair info.

Let's consider rectangle graph processing. There are three steps in it (without less of generality, we'll contract blue edges):

1. Building all possible rectangles.
2. Gluing vertices with same labels.
3. Contracting blue edges.

Let's denote the resulting graph after all three steps as $G_r = (V_r, E_r)$ and result of first two steps as $G_{rb} = (V_{rb}, E_{rb})$. Each vertex v_{rb} from V_{rb} has a label that are vertex pair info $(A(v_{rb}), B(v_{rb}), x(v_{rb}))$. First element of such triple $(A(v_{rb}))$ we call *first mark* of vertex v_{rb} and denote it $fm(v_{rb})$.

Since contracting of blue edges is gluing operation, there exists a partition of V_{rb} into family of disjoint subsets $\{H_{rb}(v_r)\}_{v_r \in V_r}$ such that all vertex from $H_{rb}(v_r)$ glued into v_r. For given vertex $v_r \in V_r$, let's consider $H_{rb}(v_r)$. Since two vertices from G_{rb} connected by blue edge have same first mark, we can define mapping $f : V_r \to V$ by following equation:

$$f(v_r) = fm(v_{rb}), \text{ where } v_{rb} \text{ any element of } H_{rb}(v_r).$$

Each edge of G_r has label from E by construction. So this labeling provides us mapping $g : E_r \to E$.

The conditions on f and g from definition of ungluing are satisfied by construction of rectangle graph.

Remark 5. *By construction, every vertex in one blue-connected component of* G_{rb} *has the same vertex* $V \in G$ *as first mark.*

Remark 6. *The third step (contracting blue edges) is equivalent to constructing a graph with blue-connected components of* G_{rb} *as vertex set. Two vertices in new graph a connected with edge labeled* e *iff in corresponding connected components of* G_{rb} *contained two vertices, connected with edge labeled* e*.*

For given compressed de Bruijn graph $G = (V, E)$ and d-set of edge pair info P we say that vertex pair info (A, B, x) is *obtained* from P if there exists edge pair info $((a, b), y)$ such that:

either a incoming edge for A, b incoming edge for B and $x = y - l_a + l_b$,

either a outgoing edge for A, b incoming edge for B and $x = y + l_b$,

either a incoming edge for A, b outgoing edge for B and $x = y - l_a$,

either a outgoing edge for A, b outgoing edge for B and $x = y$.

For given compressed de Bruijn graph $G = (V, E)$ and d-set of edge pair info P we say that vertex-edge pair info (A, b, x) is *obtained* from P if there exists edge pair info $((a, b), y)$ such that:

either a incoming edge for A and $x = y - l_a$,

either a outgoing edge for A and $x = y$.

We'll denote such edge pair info $((a, b), y)$ (possibly not unique) as *precursor* of vertex-edge pair info (A, b, x).

Two vertex-edge pairs (V, a, x) and (V, b, y) are called *adjacent* if a and b have at least one common vertex in graph G and distances from V to common vertices of a and b, obtained from (V, a, x) and from (V, b, y), are same.

We denote set of all vertex-edge pair info obtained from P as P_v. For vertex Z, $P_v(Z) = \{(A, b, x) | (A, b, x) \in P_v; A = Z\}$.

Now we can define *rectangle split operation*:

Definition 12. *For given vertex $Z \in V$ rectangle split operation consisting following steps:*

1. *Part $P_v(Z)$ into equivalence classes by transitive closure of adjacency relationship: $P_v(Z) = \bigcup C_i$.*
2. *For each such class, create vertex Z_i.*
3. *For each precursor $((a, b), y)$ of each vertex-edge pair info's $(Z, b, x) \in C_i$, create a copy a_i of edge a, replacing $\mathrm{start}(a)$ (and resp. $\mathrm{end}(a)$) to Z_i iff $\mathrm{start}(a)$ (resp. $\mathrm{end}(a)$) $= Z$ and vertex-edge pair $(\mathrm{start}(a), b, y)$ (resp. $(\mathrm{end}(a)$) obtained from $((a, b), y)$ belongs to C_i.*
4. *For each precursor $((a, b), y)$ of each vertex-edge pair info's $(Z, b, x) \in C_i$, replace $((a, b), y)$ in P with $((a_i, b), y)$*
5. *Remove Z with all it's incoming and outgoing edges.*

Remark 7. *Step's 3 is so complicated in definition because of loops. We need to replace Z as start/end vertex of outgoing/incoming edge a, and also choose correctly only one of copies of Z in case a is a loop .*

Theorem 1. *After applying rectangle split operation to each vertex of initial graph* G *in any order we'll receive a graph* $G' = (V', E')$*, such that there exists one-to-one mapping from* (V', E') *to* (V_r, E_r) *and this mapping respects labeling.*

Remark 8. *Initial graph* G *is also correct ungluing of itself, with identical* f *and* g*. We'll iteratively modify* f *and* g *in each rectangle split operation:* $f(Z_i) := f(Z)$, $g(a_i) := g(a)$ *It's evident, that after each rectangle split ungluing conditions (from definition of correct ungluing) on* f *and* g *are still satisfied.*

Proof. Let's fix some order of rectangle split operations. While applying rectangle split operations to each vertex except Z, we can somehow modify P, but the set $P_v(Z)$ stays constant (because vertex-edge paired info, obtained from removed edge e can be still obtained from an edge e_i for some i). By construction of rectangle graph, there is a one-one mapping from C_i(and so from Z_i) to set of all blue-connected components, having Z as first vertex in label. Let's consider union of this mappings for all vertices $Z \, in V$.

Consider any edge $a'_{ij} = (Z_i, Y_j)$ with label a. Without losing of generality we can assume that vertex Z of original graph G split before Y. In splitting of Y we have precursor $((a_i, b), t)$ (that remapped into (a_{ij}, b, t)) of vertex-edge pair info's $(Y_j, b, t - l_{a_i}) \in D_j$, such that $end(a_i) = Y$ where $P_v(Y) = \bigcup D_j$. In splitting of Z we have precursor $((a, b), t)$ of vertex-edge pair info's $(Z, b, t) \in C_i$, such that $start(a) = Z$. The edge pair info $((a, b), t)$ belongs to C_i and D_j, this means that edge pair info $((a, b), t)$ produce rectangle such that red edges of this rectangle connect two blue edges components that mapped into C_i and D_j, and this edges labeled by a. We shown that for each edge in our graph G' there exist edge in G_{rb}.

If we take some edge from G_{rb} than we can produce split operation like above. But edge b will be obtained from rectangle. And we produce sequence of precursor $(a, b, t) \to (a_i, b, t)) \to (a_{ij}, b, t)$. This sequence show as that in G' exist edge connecting Z_i and Y_j, and labeled with a. □

Theorem 2. *In terms of Son's paper, result of his gluing procedure and contracting blue edges is a honest ungluing of initial graph* G*.*

sketch. Each rectangle split operation for honest ungluing of G produce honest ungluing of G, so if we apply them consequently, we'll receive a honest ungluing of G. □

Theorem 3. *In terms of Son's paper, result of his gluing procedure and contracting blue edges has measure of complexity, which is less or equal to measure of complexity of initial graph* G*.*

Each rectangle split operation doesn't increase incoming-outgoing set(due to function g) of G, so their combination's incoming-outgoing set is also a subset of incoming-outgoing set of G.

5 Our ideas

5.1 Bad cases for rectangle graph

Rectangle graph is a powerful tool, but it's simple to construct graph that can be resolved manually, but stays unchanged after applying of rectangle graph techniques: Lets consider graph G with vertex set $V = A, B, C, E, F, G, H$ and edges

$$\{(A, B), (B, C), (C, E), (E, F), (F, G), (G, A), (C, H), (H, F)\}$$

. Pair information is set of all edge pairs with distance(on some subpath of G) 2 i.e. (A, B) and (C, E). Both red and blue rectangle graph are isomorphic to G but after using of slightly modified split operation for this graph we can receive a correct ungluing, consisting of two cycles: $(A \to B_1 \to C_1 \to E \to F_1 \to G_1 \to A)$ and $(A \to B_2 \to C_2 \to H \to F_2 \to G_2 \to A)$.

5.2 Rectangle split modifications

First approach is to modify adjacency definition for vertex-edge pairs.

Definition 13. *Two vertex-edge pairs* (Z, a, x) *and* (Z, b, y) *are called* adjacent *if* $start(a) = end(b)$ *or* $start(b) = end(a)$ *and distances from* Z *to common vertices of* a *and* b*, obtained from* (Z, a, x) *and from* (Z, b, y)*, are same.*

This definition is slightly different from previous adjacency definition. Using this definition we can split vertex V by vertex-edge information even if there two edges paired with V has common start or end vertex.

In case of inexact insert length we have to somehow weaken the condition on distances. Also, for highly nonuniform coverage, the request of sharing vertex for adjacent edges should be weakened:

Definition 14. *Two vertex-edge pairs* (Z, a, x) *and* (Z, b, y) *are called* $l - \varepsilon$ adjacent *if there exists path* P *from* a *to* b*:* $|P| < l$ *and* $||P| - |y - x|| < \varepsilon$*.*

5.3 Usage of split operations

After performing rectangle split for all vertices of initial graph we can recalculate pair information and do split for new vertices. It can be useful if after splitting of all vertices from initial graph, some connected components of pair infos break into unconnected parts. Also, for some vertices it is possible to split them by using paired information in other direction - consider *edge-vertex info* (a, V, x) and look on adjacency(defined in same way as for vertex-edge infos) for such objects. This splitting is equivalent (in terms of rectangle graph) to contracting red edge.

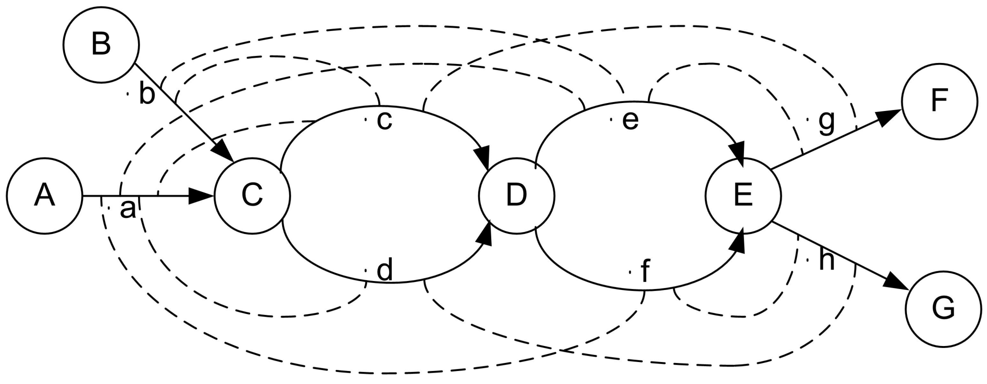

Figure 1: Graph example

6 Details about real implementation

We have to ignore low confirmed information about distance between edges to cope with chimeric pairs. Also, it may turn out that after out resolving we'll need to do tip clipping again because of erroneous connections.

7 Connected ideas

7.1 Insert-length distribution specifying

We can use mate-pairs aligning the same edge to specify insert-length distribution.

7.2 Complicated repeat resolving

Next step is to resolve repeats using distances distributions on neighbouring edges (to divide sum of two close normal distributions with one). Also, in case of big coverage we can try to correct distance information. For example, we can obtain information about distances to vertex from distances to it's incoming and outgoing edges. This two series of distances should be same. If they aren't, we can find minimal correction to get same distances and apply it to both of series.

7.3 Extending the model

In this paper we used only paired information. It seems that coverage can also be introduced to our model.

8 Example

Let's consider the graph on fig. 1. Dotted lines connect edges on distance d. Let's consider vertex E. The vertex-edge paired information obtained from neighboring edges is $\{(E, g, 0), (E, h, 0)\}$. In terms of rectangle split operation this is one connected component, but in terms of our slightly modified split operation there are two components. We split E into E_g and E_h, and edges remaps into this vertices as follow: $e \to (D, E_g)$, $g \to (E_g, F)$, $f \to (D, E_h)$, $h \to (E_h, G)$.

Now let's consider vertex C. We obtain next set of vertex-edge pair infos:

$$\{(C, c, 0), (C, e, l_c), (C, g, l_c + l_e), (C, d, 0), (C, f, l_d), (C, h, l_d + l_f)\}.$$

If lengths l_c and l_d are equal then all pair infos above belong to one component. In this case we can not split anything. Furthermore, there are no reasons to do this split if we know only original graph and only this infos.

Let's see the vertex D. Infos are $\{(D, g, l_e), (D, h, l_f)\}$. By our rule this set can be splitted into two components. For first component $\{(D, g, l_e)\}$ we create vertex D_g, and have two precursors: $(c, g, *)$, and $(e, g, *)$. So we remap edge c into (C, D_g), and e into (D_g, E_g). For second component $\{(D, h, l_f)\}$ we have vertex D_h and remap d into (C, D_h) and f into (D_h, E_h).

Now we can return to vertex C. Set of vertex-edge pair infos is the same

$$\{(C, c, 0), (C, e, l_c), (C, g, l_c + l_e), (C, d, 0), (C, f, l_d), (C, h, l_d + l_f)\}.$$

But now edge c and edge f no more adjacent, because c remapped into vertex D_g, but f remapped into vertex D_h. The same happens with edges d and e. Now we have TWO component $\{(C, c, 0), (C, e, l_c), (C, g, l_c+l_e)\}$, and $\{(C, d, 0), (C, f, l_d), (C, h, l_d+l_f)\}$. For first component we create vertex C_g. We obtain next remapping of edges: $a \to (A, C_g)$, $b \to (B, C_g)$, $c \to (C_g, D_g)$. For second component we create vertex C_h and obtain remapping $a \to (A, C_f)$, $d \to (C_f, D_f)$. For both this component we remap edge a. And now we have two edge marked by a.

So we split all possible vertices.