Detecting Repeat Families in Incompletely Sequenced Genomes

José Augusto Amgarten Quitzau[1,2] and Jens Stoye[1]

[1] AG Genominformatik, Technische Fakultät
[2] International NRW Graduate School in Bioinformatics and Genome Research
Bielefeld University, Germany

Abstract. Repeats form a major class of sequence in genomes with implications for functional genomics and practical problems. Their detection and analysis pose a number of challenges in genomic sequence analysis, especially if the genome is not completely sequenced. The most abundant and evolutionary active forms of repeats are found in the form of *families* of long similar sequences. We present a novel method for repeat family detection and characterization in cases where the target genome sequence is not completely known. Therefore we first establish the sequence graph, a compacted version of sparse de Bruijn graphs. Using appropriate analysis of the structure of this graph and its connected components after local modifications, we are able to devise two algorithms for repeat family detection. The applicability of the methods is shown for both simulated and real genomic data sets.

1 Introduction

In the later 1980's, scientists had the first contact with genome sequences of higher-order organisms. At that time, they were amazed by the amount of "junk" in these sequences. Examining this junk in the following decades, they discovered that these portions of the genome were less useless than they first suspected. In fact, there is a myriad of active elements between coding sequences, some of them being able to replicate themselves, acting like virus DNA, termed insertion sequences in bacterial genomes, and mobile elements in eukaryotes. They are in fact believed to be the vestiges of virus infections in ancestral species.

Although repetitive elements may not be active parts of the genome, since they encode only proteins which are related to their own replication, they are able to change the genome in many ways. It is known that pairs of insertion sequences act sometimes together and duplicate not only themselves, but the whole sequence between them [11]. Also when mobile elements work alone, the position where the new copy is inserted may belong to important regions in the genome, like active genes. In fact, insertions of mobile elements are observed in several genetic disorders, like Duchenne muscular dystrophy, type 2 retinitis pigmentosa, β-thalassemia, or chronic granulomatous disease [15].

A maybe less noble, but really important motivation to study repetitive elements is the waste of time and money they cause in genomic research. Finishing

a whole eukaryotic genome sequencing project is neither cheap nor fast, and the study of specific regions in these huge genomes still depends on specific primer design. But even when using very specific primers, PCR experiments may result in garbage, if the sequence to which the primer was designed appears thousands of times in the whole genome. In many cases, however, there may be enough sequenced information available to give an overview of the repetitive elements in the genome. Finding these elements in an incompletely sequenced, unfinished genome is the aim of this work.

Strategies for *de novo* repeat identification usually assume that two similar sequences in a given collection cannot be different copies of the same locus of the genome. Therefore they may assume that alignments with quality above a certain threshold provide evidence of a repeat family. We overcome this limitation by accepting any kind of sequence sets as input, including sets with several copies of the same locus. For doing this, we do not align the sequences, like the traditional approaches [1], but partially assemble them using a de Bruijn graph.

The first use of de Bruijn graphs in Bioinformatics was probably the Eulerian path approach to sequence assembly proposed by Idury and Waterman [9] and extended by Pevzner, Tang and Waterman [14]. Despite the success achieved by Pevzner and colleagues' Euler assembler in assembling bacterial genomes, the use of de Bruijn graphs for other biological applications does not seem to be further explored. We find many extensions of the de Bruijn graph based assembly approach in the recent literature [2–5, 19], but they usually focus either on improvements in error correction methods or in adapting the original method to new sequencing data. Other works present graphs that slightly remind of de Bruijn graphs, but miss the main feature of them, namely, the unique representation of tuples of a given size [13, 17].

The main problem with de Bruijn graphs becomes clear as soon as one starts working with them. As Myers [12] points out, de Bruijn graphs are simply space inefficient. And we believe Myers is right when he says that in the context of sequence assembly the whole process of cutting reads into small pieces to finally build the de Bruijn graph may not be necessary. To circumvent this, we propose an efficient implementation of sparse de Bruijn subgraphs, detailed in [16], and two strategies for using them as repeat family detection tools in incompletely sequenced genomes.

2 Sparse de Bruijn Graphs

A d-dimensional *de Bruijn graph* $G = (V, A)$ on an alphabet Σ is the graph defined as follows:

$$V = \Sigma^d$$
$$A = \{(u, v) \mid u, v \in V \text{ and } u_{i+1} = v_i, \text{ for all } i, 1 \leq i < d\}$$

where u_i denotes the ith character of string u. Strings of length at least d over the same alphabet describe walks on the d-dimensional de Bruijn graph.

Let s be a string over Σ. The d-dimensional *spectrum* of s, spectrum(s, d), is the set of all substrings of s with length d. The spectrum of a set of sequences is the union of the individual spectra. Given a set $\mathcal{S}$ of strings, the associated d-dimensional *de Bruijn subgraph* is the graph $G_{\mathcal{S}} = \{V_{\mathcal{S}}, A_{\mathcal{S}}\}$, where:

$$V_{\mathcal{S}} = \text{spectrum}(\mathcal{S}, d)$$
$$A_{\mathcal{S}} = \{(u, v) \mid u, v \in V_{\mathcal{S}} \text{ and } u_1 \dots u_d v_d \in \text{spectrum}(\mathcal{S}, d+1)\}.$$

A vertex in an associated de Bruijn subgraph is called a *junction* when it has in-degree greater than 1. A vertex with out-degree greater than 1 is called *bifurcation*.

Sequence associated de Bruijn subgraphs have a very nice asymptotic behavior. Their maximum number of nodes increases linearly with the size of the input, and even decreases with the dimension of the graph. The main problem is that, although these graphs scale very well with the sequence set size, the graphs corresponding to genomes as small as bacterial genomes are already huge.

2.1 Sequence Graph

De Bruijn graphs are by definition sparse [6, Chapter 7]. Even in applications where smaller dimensions are required [8], their number of edges in the DNA world is not greater than four times the number of nodes. Their subgraphs are surely sparser. In a typical sequence analysis application [14, 20], the probability of having a node with maximum in- or outdegree is very low. Therefore the graph construction in such applications is usually followed by a step where long branch-free paths are collapsed to single nodes. In [16], we present a way to directly construct the compact representation of a sparse de Bruijn graph, called a *d-dimensional sequence graph*, or simply *sequence graph*.

An example of a sequence graph is shown in Figure 1. Like a d-dimensional de Bruijn subgraph, every d-tuple over the given alphabet may be represented by at most one vertex in the sequence graph. Furthermore, a sequence graph may contain an arc (u, v) only if the suffix of length $d-1$ of u matches perfectly the prefix of v. The main difference between a sequence graph and a de Bruijn graph is that vertices in a sequence graph are not limited to the size d, but may have any size between d and $|\Sigma|^d + d - 1$. This allows the representation of non-branching paths in a single node. The compression, however, depends on the way the structure is built.

There is an index mapping every d-tuple represented by the sequence graph to the node in which it is found. The tuple position in the node is also stored, so that it may be directly accessed, after the index search. We also extend the natural concept of neighborhood from nodes to tuples. In a sequence graph, two d-tuples a and b are called *neighbors* if either there is an arc (u, v) such that the suffix of length d of u is a, and the prefix of length d of v is b; or both a and b are in the same node and the occurrence of a precedes the occurrence of b by one position.

Apart from the inclusion of nodes, there are two operations that can be applied on the set of nodes.

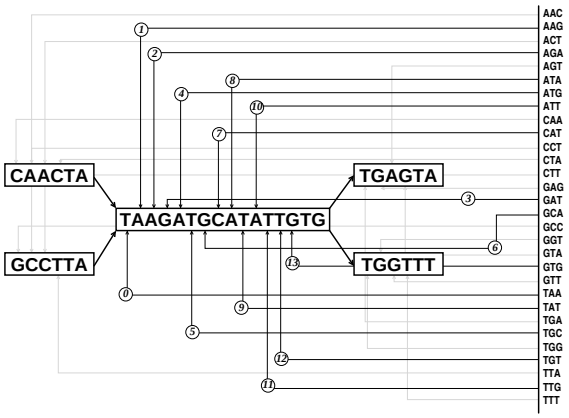

Fig. 1. Sequence graph corresponding to a 3-dimensional de Bruijn subgraph on the alphabet $\Sigma = \{A, C, G, T\}$. Links connecting the index to the node `TAAGATGCATATTGTG` are shown as black arrows with offsets, all other such connectors are shown in gray.

Cut: Transforms a single node in two neighbor nodes. A cut does not change the set of sequences represented by the graph, since no new tuple of size d or greater is created, and the new edge binds two tuples that were neighbors before.

Merge: Is the inverse operation of cut. Notice that this operation can only be applied on neighbor nodes u and v such that u has outdegree 1 and v has indegree 1. The operation removes the edge (u, v) by merging its nodes into a single node.

2.2 Repeat Families in Sequence Graphs

The length of repetitive elements may vary from the few bases of short tandem repeats to the thousands of bases of long transposons. We know that only exact repeats with length greater than the underlying graph dimension can be identified in such a graph, since they are represented by single nodes both in the sequence graphs, and in the original form of de Bruijn graphs.

Although they may be much smaller than the graph dimension, the exhaustive, uninterrupted succession of almost perfect copies in tandem repeats is able to create tangled patterns in the graph. In these cases, the large number and perfection of copies is responsible for the rising of larger perfect matches.

In the case of interspersed repeats, their replication mechanism allows the appearance of copies which are physically far away from each other in the DNA molecule. On the other hand, the content of a single copy is usually unique. Apart from the usual reverse short repeats in their extremities, the sequence inside mobile elements often lacks exact repeats. Therefore the portion of a sequence graph corresponding to a repeat family is much better organized than the tangled tandem repeats regions. Often the sequence graphs of repeat families are directed acyclic graphs. This can be used as a starting point for repeat identification.

3 Repeat Family Detection

Although the real challenge we want to address with this work is the detection of repeat families in eukaryotic genomes, our first subject of study are the simpler, easier to understand bacterial genomes. A typical bacterial genome is not bigger than six or seven million base pairs, and roughly the same number of l-tuples, when l is relatively small. The number of possible strings of length l for an alphabet of size 4, by contrast, is already huge for very small values of l. For typical sequence analysis applications, like approximate string matching, the value of l is chosen large enough to allow the assumption that very few tuples appear twice in the genome just by chance.

The number of repeat families in a single genome is quite small. The average number of different families in a single genome found in [11] is 2.79. The copy number of a family in a single genome is also not big. Although the number of copies can be as big as 14, like the number of copies belonging to the family IS1 in Mycoplasma, the average number of distinct elements of the same family is 2.27. Therefore, assuming that copies are uniformly spread along the genome sequence, we may expect repeats to be separated by quite long non-repetitive sequences. This may be also true for some eukaryotes, like *Arabidopsis thaliana*, which has 10% of its genome composed by mobile elements [7], while other eukaryotes have a much more complicated genome structure.

3.1 Connected Components

Nodes corresponding to repetitive sequences may be discovered and marked during the sequence graph construction. Nodes corresponding to unique sequences either represent larger sequences from the unique parts of the genome, or are the result of small dissimilarities between elements of the same repeat family. In the second case, unique nodes are not larger than a repeat family element, since entities of the same repeat family are similar enough to share perfect matches. On the other hand, unique sequences between repeat copies can be much longer.

The sequence graph for a genome must therefore be composed of clusters of small repetitive nodes, interconnected by longer single ones. As a result, the deletion of long unique nodes may decompose the graph into a few connected components, containing one or more repeat families. Based on this simple principle, we devised a method for separating repeat families in a genome. The procedure is described in Algorithm 1. The input is a set $\mathcal{S}$ of reads of some genome and a length threshold value l. We start building the sequence graph for this set of sequences. Originally, nodes with different sequence sets cannot be merged. As a result, every read end coincides with a node end, which leads in many cases to branch free paths in the sequence graph. Therefore we ignore this restriction and merge nodes in branch free paths, as long as they are either both marked as repeats, or both unmarked, even if their sequence sets are not identical. The resulting graph may contain long single nodes, exceeding the length threshold l. We delete them, and merge the repeated nodes that were not

merged only because of the now deleted long nodes. Notice that, at this point, no other node can be merged or deleted.

The resulting graph is already a collection of separated connected components. However, some of them may be the result of unrelated small perfect matches. These repeats created by chance are easy to identify. They are in components with few nodes (not more than 5), with a single, short repeated node in the center. We call these components *small components*. The small components are removed from the graph as well, leaving only components corresponding to larger families.

We implemented this approach in the Java programming language and tested it with artificially created chromosomes, simulating situations from simple bacterial chromosomes to chromosomes which are more than 50% composed of repetitive elements. Details about simulation and results are given in Section 4.1.

Algorithm 1 Connected Components

```
1: function IsolateComponents(S, l)
2:     Build the sequence graph for S
3:     Merge all possible pairs of nodes
4:     Remove all single nodes of length ≥ l
5:     Merge all possible pairs of nodes
6:     Remove all small components
7:     return the resulting connected components
8: end function
```

3.2 Combining Nodes

In cases when elements of the same repeat family differ in many close bases, the connected component based detection method may miss some less represented families. This happens because the few sequences do not share any l-tuple in a certain region.

As a result, although two (or more) long closely related single nodes can be found in the graph, they are discarded as long unique nodes, and the remaining part is either detected as a partial family, or is discarded as a small component. On the other hand, repeats of the same family often share at least one l-tuple somewhere along their sequence. And in the cases where sequences share only a few tuples, the nodes representing the common ones can be extended by combining the closely related unique nodes around them.

By *combining* two nodes we mean replacing both nodes by a single node whose sequence is the consensus between them. The procedure is shown in Algorithm 2 and represented in Figure 2. In the most general case, the two nodes to be combined, n_1 and n_2, are of different length. We assume w.l.o.g. that n_1 is longer than n_2.

In the first step, the node prefixes are aligned. We use a semi-global alignment algorithm [18, Section 3.2.3] for that. The score matrix considers matches

Algorithm 2 Combine

```
1: procedure COMBINE(n_1, n_2, t)
2:     Let n_1 be the longer of the two nodes
3:     Align the sequences of n_1 and n_2, creating a semi-global alignment of length l
4:     if the alignment score is smaller than t × ⌈(l−d+1)/d⌉ then
5:         Cut the node n_1 at the end of the aligned prefix
6:         Let n̄_1 be the left portion of the cut node n_1
7:         Create a new node n with the consensus of n̄_1 and n_2
8:         Bind the nodes in the neighborhood of n̄_1 and n_2 to n
9:         Remove n̄_1 and n_2
10:    end if
11: end procedure
```

between any two possible symbols found in the IUPAC standard code for nucleotides. For symbols that represent a single nucleotide (`A`, `C`, `G`, `T`), the score is simply 0 for a match and 1 for a mismatch. For matches involving at least one symbol representing a set of nucleotides, like `W` = {`A`, `T`}, the score is 0 if one set contains the other; otherwise it is the minimum number of replacements and deletions needed to transform one set into the other. For instance, the score for aligning `W` with `G` is 2, since we need to replace one of the elements of `W` by `G`, and delete the remaining one; on the other hand, the score for aligning `W` with `T` is 0, since `W` contains `T`.

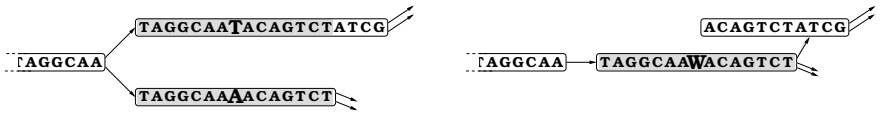

Fig. 2. The *combine* operation. The two shaded nodes on the left are combined, and result in the shaded node on the right. Two nodes are only combined when the edit distance between their prefixes is below a certain threshold. The new node label contains the consensus sequence.

Alignments with a score below a certain threshold allow the combination. The threshold is defined by the minimum number of mismatches needed to separate the nodes, m, rescaled by a user defined scale factor t. The minimum number of mismatches is given by

$$m = \left\lceil \frac{l - d + 1}{d} \right\rceil,$$

where l is the alignment length and d is the underlying de Bruijn graph dimension.

When the alignment score allows a combination, the longer of the two nodes, n_1, is cut at the point where the aligned prefix ends. The prefix and the second

node n_2 are then replaced by a node n representing the alignment consensus. This new node is finally connected to the neighborhood of the replaced nodes.

In the general case, combining two nodes does not reduce the graph size, but its complexity. In practice, a series of combinations may reduce tangled subgraphs to simple paths, which may be finally merged into a single longer node. Especially for more complex datasets sequenced at low genome coverage, this procedure gives a considerable advantage over the simple connected component approach, as shown by the results in Section 4.2.

4 Results

We applied our methods to both real and artificially created data. Comparing our methods to already published *de novo* repeat identification methods is not possible because they differ both in the input and in the output. We have as input not continous portions of an incompletely sequenced genome, but a low coverage set of reads. And we output connected components, which may be interpreted as collections of reads which belong to the same repeat family. Therefore, when measuring the success of our method in separating repeat families in a genome, the most successful scenario is clearly the situation where, we get each family in a different connected component, and components containing only members of a single family.

4.1 Connected Components

For a proof of concept, we applied the connected component strategy to artificially created chromosomes with different numbers of repeat families. Each simulated chromosome has a total length of 1 million base pairs and is composed by two kinds of sequences:

Background Sequence: The background sequence corresponds to the non-repetitive genome sequence. In our tests we used 19-dimensional de Bruijn subsequences as background, which means that the background sequences do not contain any duplicated substring of length 19.

Repeat Families: The repeat families are collections of similar sequences, called the *family members*. They originate from a 19-dimensional de Bruijn subsequence, called the family's *base sequence*, which is then used to create the other family members. Families are created in an incremental tree-like fashion: for creating a new member, we randomly take a pre-existing one and imperfectly duplicate it by simulating insertions, deletions and replacements. Each newly created sequence differs from its original in 6% of the nucleotides on average. This agrees with real cases, like the Alu family in the human genome, where the sequences diverge by up to 12% from other elements in the family [10]. The number of members in a family is called the *family size*.

The inserted repeat families were of size 2, 4, 16, and 256. In our tests, an artificial chromosome can have either 0 or 2 families of each size. All possible

combinations were used, giving a total of 15 chromosome configurations. For each configuration we created 15 different chromosomes and read sets with 0.25, 0.5, 0.75, and 1 time coverage, simulating partially finished sequencing projects. The artificially created reads have average length of 250 base pairs.

The sets of reads were given as input to the connected component based repeat family detector, resulting in a collection of connected components. Because we know which sequences correspond to family members, we were able to associate each resulting graph component to the families contained in it. The result of this association is shown in Table 1.

In the ideal case, we would find each family contained in a single connected component. Table 1 shows a different reality. In the left column ("Components per Family"), we see that families are usually split into more than three components. However, each component usually contains sequences of a single family, which is shown by the column "Families per Component". This shows that although the families are split, they are at least not so mixed up that their separation is impossible.

In the rightmost column ("Discovered Families (%)") we see how much of the inserted families could be detected by the method. The fact that we were never able to identify all the families in the odd rows is expected. These are cases where the chromosomes have families of size two. In such cases, depending on the underlying sequence graph dimension used, it can happen that the two family sequences do not share any tuple, or the number of shared tuples is so small that they end up being discarded as small components. In these cases, the combine operation plays an important role, as the next section shows.

Components per Family				**Families per Component**				**Discovered Families (%)**			
25	**50**	**75**	**100**	**25**	**50**	**75**	**100**	**25**	**50**	**75**	**100**
6.33	5.57	4.41	5.68	1.00	1.00	1.00	1.00	20	40	70	73
4.50	3.50	4.33	4.70	1.00	1.00	1.00	1.00	63	97	97	100
5.26	4.78	4.82	4.32	1.00	1.00	1.00	1.00	43	70	85	92
4.27	4.37	4.18	4.42	1.00	1.00	1.00	1.00	100	100	100	100
3.93	4.48	4.31	4.12	1.00	1.00	1.00	1.02	58	78	87	93
4.41	4.01	4.31	4.47	1.00	1.01	1.00	1.02	83	95	100	100
4.31	4.83	4.25	4.20	1.01	1.01	1.02	1.05	64	82	91	94
4.08	3.80	3.89	4.44	1.93	1.93	1.93	1.93	100	100	100	100
4.68	4.56	3.98	4.51	1.78	1.93	1.61	1.51	65	70	80	90
4.38	4.93	4.45	4.72	1.78	1.50	1.58	1.91	85	100	100	100
4.88	4.36	4.28	4.14	1.64	1.55	1.40	1.48	60	77	89	96
4.83	4.69	5.08	4.23	2.18	2.89	2.98	3.31	100	100	100	100
4.48	4.89	4.81	4.50	2.29	2.19	2.36	2.28	72	89	90	98
4.28	4.62	4.63	4.24	1.66	1.82	2.35	2.50	89	98	99	100
4.26	4.51	4.91	4.98	1.60	1.66	1.78	1.87	68	93	97	98

Table 1. Summary of the experiments with artificial data described in Section 4.1. Each row corresponds to one of our 15 data sets. On the left we see the average number of different components containing sequences of the same family. In the middle are the average numbers of different families found in a single component. On the right we see the percentage of inserted families which could be found in the graph after eliminating long nodes.

4.2 Combine

In order to evaluate our more advanced algorithm, we created a dataset with real bacterial genome sequences and their known insertion sequences. The bacterial chromosomes were obtained from the NCBI Website[1], while the corresponding insertion sequences were obtained from the insertion sequence database *IS Finder*[2]. We created 15 read sets covering 25, 75, 50, and 100 percent of the genome on average. Each of the read sets was used twice as input for the *combine* method: once with combine scale factor $t = 1.0$, and a second time with scale factor $t = 3.0$. Again we associated the resulting connected components to the repeat families found in each of them.

In Table 2 we see the percentages of the known insertion sequence families which could be detected by the method at different coverages. The biggest percentage is shown in bold. We see that, by allowing more divergent nodes to combine, we are not only able to identify more families, but also to identify them at lower coverage.

Combine Factor (t)	**1.0**				**3.0**			
Sequencing Coverage (%)	**25**	**50**	**75**	**100**	**25**	**50**	**75**	**100**
*Bacillus anthracis (plasmid PX01)**	0	7	0	40	**17**	**58**	**92**	**100**
Bifidobacterium longum	14	54	63	64	**39**	**68**	**81**	**83**
Burkholderia xenovorans	44	**67**	73	78	**50**	**67**	**75**	**81**
*Colwellia psychrerythraea**	40	**100**	**100**	**100**	**83**	**100**	**100**	**100**
*Desulfitobacterium hafniense**	67	80	97	**100**	**71**	**96**	**100**	**100**
*Desulfovibrio desulfuricans**	**33**	47	**93**	**100**	**33**	**75**	92	**100**
Escherichia coli	17	50	62	70	**44**	**65**	**85**	**92**
Geobacter uraniumreducens	32	62	67	70	**46**	**68**	**75**	**80**
Gloeobacter violaceus	30	70	60	83	**54**	**75**	**88**	**100**
*Granulibacter bethesdensis**	7	7	40	53	**25**	**33**	**42**	**75**
Haloarcula marismortui	3	12	22	28	**13**	**25**	**50**	**63**
Halobacterium sp-plasmid pNRC100	**37**	**57**	**56**	**61**	**37**	42	53	57
Legionella pneumophila-Paris	0	**13**	**20**	7	**8**	0	0	**17**
Legionella pneumophila-Philadelphia 1	27	**63**	**93**	**93**	**54**	**63**	63	92
Methanosarcina acetivorans	88	98	98	**100**	**93**	**100**	**99**	**100**
Methylococcus capsulatus	22	65	77	83	**44**	**71**	**90**	**96**
*Nitrosospira multiformis**	53	93	**100**	**100**	**92**	**100**	**100**	**100**
Photobacterium profundum	87	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Pseudomonas syringae	92	99	**100**	**100**	**97**	**100**	**100**	**100**
Pyrococcus furiosus	47	58	71	73	**50**	**72**	**81**	**89**
Ralstonia solanacearum	38	60	75	89	**53**	**78**	**93**	**95**
Rhodopirellula baltica	82	98	**100**	**100**	**97**	**100**	**100**	**100**
*Roseobacter denitrificans**	40	80	87	**100**	**42**	**92**	**100**	92
*Salinibacter ruber**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**
Shewanella oneidensis	10	26	**18**	23	**18**	**38**	15	**41**
Sulfolobus solfataricus	**94**	99	**100**	99	**94**	**100**	**100**	**100**

Table 2. Percent of known insertion sequence families found in incompletely sequenced bacterial genomes at different coverages. Bacteria marked with a '$*$' symbol do *not* have any family with a single member. Numbers marked in **bold** indicate for which of the two combine factors more families were identified, on average.

[1] NCBI: `http://www.ncbi.nlm.nih.gov`
[2] IS Finder: `http://www-is.biotoul.fr/is.html`

5 Conclusion

We presented two methods for detecting repeat families in incompletely sequenced genomes. The methods are based on operations on the set of nodes and edges of a sequence graph, a compacted variant of a sparse de Bruijn graph.

The first method was based on the deletion of long nodes corresponding to non-repetitive genome portions. Based on experiments involving artificially created data, we showed that the use of this method for family detection is possible, although the families may be split in a few components in the resulting graph. We also noticed that families appearing in small copies are in many cases not detected by the method.

Another node operation was presented in order to combine similar, but not identical, nodes of different members in a family. This operation was applied before the deletion of long nodes, in order to avoid the splitting of family components. Although the combination of nodes leads to a reduction in the number of components per families (data not shown), the main advantage of this operation is better observed in experiments involving real bacterial genomes, where the node combination leads to the detection of only weakly represented families.

The main obstacle for using these methods in practical applications is the splitting of repeat families in separated components. This is for us the main problem to be tackled before applying the method in genomes of higher complexity, like eukaryotes.

References

1. Zhirong Bao and Sean R. Eddy. Automated de novo identification of repeat sequence families in sequenced genomes. *Genome Res.*, 12:1269–1276, 2002.
2. Shahid H. Bokhari and Jon R. Sauer. A parallel graph decomposition algorithm for DNA sequencing with nanopores. *Bioinformatics*, 21(7):889–896, 2005.
3. Jonathan Butler, Iain MacCallum, Michael Kleber, Ilya A. Shlyakhter, Matthew K. Belmonte, Eric S. Lander, Chad Nusbaum, and David B. Jaffe. ALLPATHS: De novo assembly of whole-genome shotgun microreads. *Genome Res.*, 18:810–820, March 2008.
4. Mark Chaisson, Pavel Pevzner, and Haixu Haixu Tang. Fragment assembly with short reads. *Bioinformatics*, 20(13):2067–2074, 2004.
5. Mark J. Chaisson and Pavel A. Pevzner. Short read fragment assembly of bacterial genomes. *Genome Res.*, 18:324–330, March 2008.
6. Reinhard Diestel. *Graph Theory*. Number 173 in Graduate Texts in Mathematics. Springer-Verlag, third edition, 2005.
7. Anne E. Hall, Aretha Fiebig, and Daphne Preuss. Beyond the arabidopsis genome: Opportunities for comparative genomics. *Plant Physiol.*, 129:1439–1447, August 2002.
8. Lenwood S. Heath and Amrita Pati. Genomic signatures in de Bruijn chains. In Raffaele Giancarlo and Sridhar Hannenhalli, editors, *Proceedings of WABI 2007*, volume 4645 of *LNCS*, pages 216–227. Springer, 2007.
9. Ramana M. Idury and Michael S. Waterman. A new algorithm for DNA sequence assembly. *J. Comput. Biol.*, 2(2):291–306, 1995.

10. Warren R. Jelinek, Thomas P. Toomey, Leslie Leinwald, Craig H. Duncan, Paul A. Biro, Prabhakara V. Choudary, Sherman M. Weissman, Carol M. Rubin, Catherine M. Houck, Prescott L. Deininger, and Carl W. Schmid. Ubiquitous, interspersed repeated sequences in mammalian genomes. *Proc. Natl. Acad. Sci. USA*, 77(3):1398–1402, March 1980.
11. Jacques Mahillon and Michael Chandler. Insertion sequences. *Microbiol. Mol. Biol. Rev.*, 62(3):725–774, September 1998.
12. Eugene W. Myers. The fragment assembly string graphs. *Bioinformatics*, 21:ii79–ii85, 2005.
13. Pavel A. Pevzner, Haixu Tang, and Glenn Tesler. *De novo* repeat classification and fragment assembly. In *Proceedings of RECOMB 2004*, pages 213–222, March 2004.
14. Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An Eulerian path approach to DNA fragment assembly. *Proc. Natl. Acad. Sci. USA*, 98(17):9748–9753, August 2001.
15. Eline T. Luning Prak and Haig H. Kazazian Jr. Mobile elements and the human genome. *Nature Rev.*, 1:134–144, November 2000.
16. José A. Amgarten Quitzau and Jens Stoye. A space efficient representation for sparse de Bruijn subgraphs. Report, Technische Fakultät der Universität Bielefeld, Abteilung Informationstechnik, 2008. `http://bieson.ub.uni-bielefeld.de`.
17. Benjamin Raphael, Degui Zhi, Haixu Tang, and Pavel Pevzner. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Res.*, 14:2336–2346, 2004.
18. João C. Setubal and João Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing, 1997.
19. Daniel R. Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res.*, 18:821–829, March 2008.
20. Yu Zhang and Michael S. Waterman. An Eulerian path approach to local multiple alignment for dna sequences. *Proc. Natl. Acad. Sci. USA*, 102(5):1285–1290, February 2005.