contiBAIT: Improving Genome Assemblies Using Strand-seq Data

Kieran O'Neill, Mark Hills and Mike Gottlieb

December 29, 2015

koneill@bcgsc.ca

Contents

1 Licensing

Under the Two-Clause BSD License, you are free to use and redistribute this software.

2 Introduction

Strand-seq is a method for determining template strand inheritance in single cells. When strand-seq data are collected for many cells from the same organism, spatially close genomic regions show similar patterns of template strand inheritance. ContiBAIT allows users to leverage this property to carry out three tasks to improve draft genomes. Firstly, in assemblies made up entirely of contigs or scaffolds not yet assigned to chromosomes, these contigs can be clustered into chromosomes. Secondly, in assemblies wherein scaffolds have been assigned to chromosomes, but not yet placed on those chromosomes, those scaffolds can be placed in order relative to each other. Thirdly, for assemblies at the chromosome stage, where scaffolds are ordered and separated by many unbridged sequence gaps, the orientation of these sequence gaps can be found.

All three of these tasks can be run in parallel, taking contig-stage assemblies and ordering all fragments first to chromosomes, then within chromosomes while simultaneously determining the relative orientation of each fragment. This vignette will outline some specific functions of contiBAIT, and is comparible to the contiBAIT() master function included in this package that will perform the same sequence of function calls outlined below.

3 Input

ContiBAIT requires input in BAM format. Multiple BAM files are required for analysis, so ContiBAIT specifically calls for users to identify a BAM directory in which to analyse. Sorted BAM files will speed up analysis.

```
> # Read in BAM files. Path denotes location of the BAM files.
> # Returns a vector of file locations
>
> library(contiBAIT)
> bamFileList <- list.files(
+ path=file.path(system.file(package='contiBAIT'), 'extdata'),
+ pattern=".bam$",
+ full.names=TRUE)
```

The example data provided by contiBAIT is from a human blood sample and has been aligned to GRCh38/hg38. Since this assembly is already complete, we must first split this genome into chunks to simulate a contig-stage assembly. To do this we need to extract information on the assembly the bam file is aligned to by creating a chromosome table instance, then splitting this table.

3.1 *Creating a chromosome table instance*

The example data provided with the contiBAIT package is derived from GRCh38/hg38 data (only aligned to all autosomes and allosomes; no alternative locations or contigs were included). To subset these data, and for further downstream analysis, a chromosome table instance can be made, representing the contig name and length. This is generated with makeChrTable, where the resulting data.frame is similar to the header portion of a BAM file. Using the addBED option will output a bed-format version of the same data.frame, with columns for chromosome name, start and end locations.

```
> # build chr table from BAM file in bamFileList
>
> exampleChrTable <- makeChrTable(bamFileList[1])
> exampleChrTable

A data.frame of 24 fragments from a 3095.677 Mb genome.
      chr    length
chr1 chr1 249250621
chr2 chr2 243199373
chr3 chr3 198022430
chr4 chr4 191154276
chr5 chr5 180915260
chr6 chr6 171115067
...
        chr    length
chr19 chr19  59128983
chr20 chr20  63025520
chr21 chr21  48129895
chr22 chr22  51304566
chrX   chrX 155270560
chrY   chrY  59373566
```

3.2 *Splitting a chromosome table instance*

Now we can split the above chromosome table instance into 1 Mb fragments. This subdivision isn't just for testing purposes. For chromosome- and contig-stage assemblies with very large fragments, subdividing the data into smaller fragments can help identify chimeric fragments and misorientations. Some assemblies have a large degree of misorientations or chimerism in the data, and subdividing them aids in clustering these fragments. For example, if a region is misoriented within a contig, the strand state will change in this region, skewing this contig toward a WC call in every library. However, while fragmenting can improve the overall number of contigs included in analysis and improve clustering, as the fragments get further subdivided, the number of reads used to make strand state calls decreases, and the probability of there being insufficient reads

to make an accurate call increases. Note the following divided chromosome table can be used with the filter argument in strandSeqFreqTable to generate a sub-divided table.

```
> dividedChrTable <- divideMyChr(exampleChrTable, splitBy=1000000)
> dividedChrTable

A data.frame of 3113 fragments from a 3095.677 Mb genome.
                       chr   start     end
chr1:0-1000000       chr1       0 1000000
chr1:1000000-2000000 chr1 1000000 2000000
chr1:2000000-3000000 chr1 2000000 3000000
chr1:3000000-4000000 chr1 3000000 4000000
chr1:4000000-5000000 chr1 4000000 5000000
chr1:5000000-6000000 chr1 5000000 6000000
...
                         chr    start      end
chrY:54000000-55000000 chrY 54000000 55000000
chrY:55000000-56000000 chrY 55000000 56000000
chrY:56000000-57000000 chrY 56000000 57000000
chrY:57000000-58000000 chrY 57000000 58000000
chrY:58000000-59000000 chrY 58000000 59000000
chrY:59000000-59373566 chrY 59000000 59373566
```

3.3 *Splitting a chromosome table based on strand state changes*

A change in strand state within a contig can represent a number of things. At it's simplest, it could represent a sister chromatid exchange switching the templates in that particular cell. In cases where the same location is a site of recurrent strand state changes, the more likely explanation is that the fragment is chimeric or has a misorientation within it. contiBAIT allows users to cut contigs at these locations to allow for better clustering. The most likely site of incorrectly oriented or placed fragments is at unbridged or bridged gap regions. A function is included that allows us to look for overlaps between recurrent strand state changes and gap regions.

```
> library(rtracklayer)
> # Download GRCh38/hg38 gap track from UCSC
> gapFile <- import.bed("http://genome.ucsc.edu/cgi-bin/hgTables?hgsid=465319523_SLOtFPExny4
> # Create fake SCE file containing four regions that overlap one such gap
> sceFile <- GRanges(rep('chr4',4),
+ IRanges(c(1410000, 1415000, 1420000, 1425000),
+ c(1430000, 1435000, 1430000, 1435000)))
> overlappingFragments <- mapGapFromOverlap(sceFile,
+  gapFile,
```

```
+  exampleChrTable,
+  overlapNum=4)
> show(overlappingFragments)

A data.frame of 25 fragments from a 3095.682 Mb genome.
                         chr   start       end
chr4:0-1434206         chr4       0   1434206
chr4:1429359-191154276 chr4 1429359 191154276
chr1:0-249250621       chr1       0 249250621
chr2:0-243199373       chr2       0 243199373
chr3:0-198022430       chr3       0 198022430
chr5:0-180915260       chr5       0 180915260
...
                    chr start       end
chr19:0-59128983 chr19     0  59128983
chr20:0-63025520 chr20     0  63025520
chr21:0-48129895 chr21     0  48129895
chr22:0-51304566 chr22     0  51304566
chrX:0-155270560  chrX     0 155270560
chrY:0-59373566   chrY     0  59373566
```

What is returned is a chromosome table instance where the gap that is coincident with the recurrent strand state change has split that contig into two smaller fragments. Note the example table now has 25 fragments as chr4 has been split.

3.4 *Creating a strandFreqMatrix instance*

To read in BAM files into ContiBAIT, create a strandFreMatrix instance by calling strandSeqFreqTable(). This program will read each BAM file, calculate the ratio of W and C reads, and return this value along with the total number of reads used to make the call. Note, we will use the divided chromosome table to simultaneously cut the assembly into 1 Mb fragments. By default duplicate reads are removed, a minimal mapping quality of 0 is used and the function expects to see paired end data. Because of the way BAM files store strand information, it is important to ensure that the pairedEnd parameter is correctly set. A warning will be issued if single-end data is run as if it is paired-end.

```
> # Create a strandFreqTable instance
>
> strandFrequencyList <- strandSeqFreqTable(bamFileList,
+ filter=dividedChrTable,
+ qual=10,
+ pairedEnd=FALSE)
```

This returns a list of two data.frames. The first data.frame consists of a strand state frequency, calculated by taking the number of Watson (- strand)

reads, subtracting the number of Crick (+ strand) reads, and dividing by the total number of reads. These values range from -1 (entirely Watson reads) through to 1 (entirely Crick reads). The second data.frame consists of the absolute number of reads covering the contig. This is used in thresholding the data, and in weighting the accuracy of calls in subsequent orderings. Note that the fewer reads used to make a strand call, the less accurate that call will be. In the absence of background reads, WC regions will follow a binomial distribution. If we assume any contigs with <-0.8 are WW, and >0.8 are CC (this is the default strandTableThreshold parameter used in preprocessStrandTable), then there is a probability of 0.044 that the strand state call is incorrect. As such we exclude calls that are made with fewer than 10 reads. Increasing this number will make calls more accurate, but will reduce the number of contigs included in analysis. Since the contigs are weighted based on read density during clustering, a minimum of 10 reads to support a strand call provides a good balance between accuracy and inclusion.

```
> # Returned list consisting of two data.frames
> strandFrequencyList

$strandTable
A matrix of strand frequencies for  76  contigs over  35  libraries.

$countTable
A matrix of read counts for  76  contigs over  35  libraries.

> # Exclude frequencies calculated from
> # contigs with less than 10 reads
>
> exampleStrandFreq <- strandFrequencyList[[1]]
> exampleReadCounts <- strandFrequencyList[[2]]
> exampleStrandFreq[which(exampleReadCounts < 10)] <- NA
```

Additional information can be found on the help page for strandSeqFreqTable including all parameters.

The quality of the libraries, specifically whether the files being analysed appear to show the expected distributions of directional reads, can be assessed with plotWCdistributions. In a diploid organism, there is an expectation that chromosomes will be derived from either two Watson homologues, one Watson and one Crick homologue, or two Crick homologues in a Mendelian 1:2:1 ratio. In Strand-seq data, this will mean about 1/4 of the contigs will only have Watson reads mapping to them, and have a strand state frequency of -1, 1/4 of the contigs will only have Crick reads mapping to them, and have a strand state frequency of +1, and 1/2 of the contigs will have an approximately even mix of Watson and Crick reads (based on a binomial distribution of sampling). plotWCDistribution generates boxplots for different strand state frequencies and models the expected distribution (blue line). The average called WW or CC

contigs are shown in green, and should match closely with the expected distribution line.

```
> # Assess the quality of the libraries being analysed
> plotWCdistribution(exampleStrandFreq)
```

4 Creating a strand state matrix

The returned list of strandSeqFreqTable can be converted to a strand state matrix that makes a contig-wide call on the overall strand state based on the frequencies of Watson and Crick reads. The function removes BAM files that either contain too few reads to make accurate strand calls or are not strand-seq libraries (i.e. every contig contains approximately equal numbers of + and - reads). Conversely the function removes contigs that either contain too few reads, or always contain roughly equal numbers of + and - reads. More details on the parameters can be found in the function documentation. The function returns a similar data.frame to strandSeqFreqTable, but with the frequencies converted to strand calls: 1 is a homozygous Watson call (by default, a frequency less than -0.8, but this can be changed with the filterThreshold argument), 2 is a heterozygous call (a frequency between -0.8 and 0.8 by default) and 3 is a homozygous Crick call (by default, a frequency above 0.8). These factors can then be used to cluster similar contigs together.

It is important to note that the relative directionality of any two fragments within an assembly is unknown. Contigs which belong on the same chromosome but are in different orientations will display as complete opposites; every library where one contig is homozygous Watson will have the other contig as homozygous Crick. However, heterozygous contigs (where chromosomes inherited one Watson template and one Crick template), will not be mirrored, with one contig being "WC", while the other will be "CW". As such, a data.frame without heterozygous calls is also generated to identify orientation issues.

```
> # Convert strand frequencies to strand calls.
>
> exampleStrandStateMatrix <- preprocessStrandTable(
+ exampleStrandFreq,
+ lowQualThreshold=0.8)
> exampleStrandStateMatrix[[1]]

A strand state matrix for  75  contigs over  35  libraries.
```

5 Clustering contigs into chromosomes

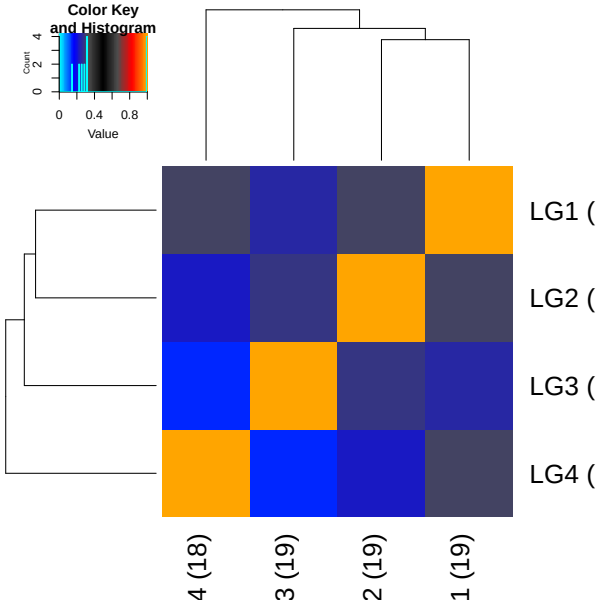
clusterContigs utilizes a custom algorithm to cluster all fragments together that share a similar strand state across multiple cells. For example, if two contigs

are adjacent on the same chromosome, then they will inherit the same strand state in every cell that is analyzed. The function performs clustering only on homozygous calls (WW or CC) to identify dissimilarity arising from misoriented fragments that would otherwise be lost when heterozygous calls (WC) are included.

```
> exampleWCMatrix <- exampleStrandStateMatrix[[1]]
> exampleWWCCMatrix <- exampleStrandStateMatrix[[2]]
> clusteredContigs <- clusterContigs(exampleWWCCMatrix)
> LGOrientations <- reorientLinkageGroups(clusteredContigs,
+  exampleWWCCMatrix)
> reorientedMatrix <- reorientStrandTable(exampleWCMatrix,
+ linkageGroups=clusteredContigs,
+ orientation=LGOrientations)
> exampleLGList <- mergeLinkageGroups(clusteredContigs,
+ reorientedMatrix)
> exampleLGList

A linkage group list containing  4  linkage groups.

  NumberOfContigs
1              19
2              19
3              19
4              18

> exampleLGList[[1]]

 [1] "chr4:3000000-4000000"     "chr4:30000000-31000000"
 [3] "chr4:10000000-11000000"   "chr4:1000000-2000000"
 [5] "chr4:100000000-101000000" "chr4:40000000-41000000"
 [7] "chr4:170000000-171000000" "chr4:150000000-151000000"
 [9] "chr4:160000000-161000000" "chr4:130000000-131000000"
[11] "chr4:6000000-7000000"     "chr4:120000000-121000000"
[13] "chr4:140000000-141000000" "chr4:90000000-91000000"
[15] "chr4:70000000-71000000"   "chr4:20000000-21000000"
[17] "chr4:60000000-61000000"   "chr4:80000000-81000000"
[19] "chr4:110000000-111000000"
```

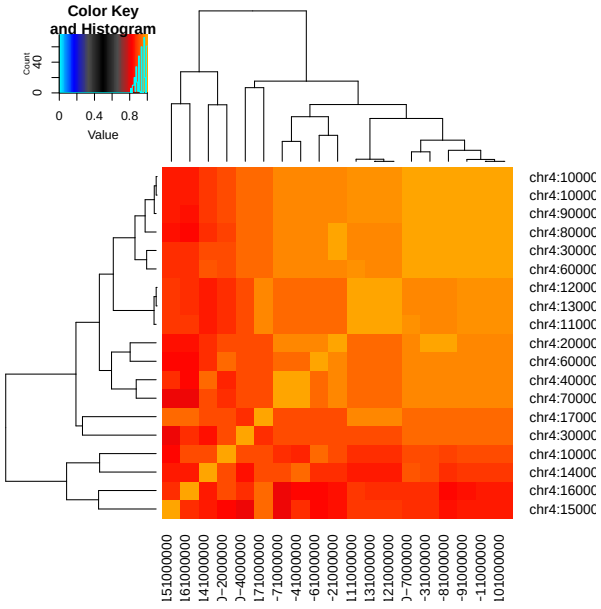
The clusterContigs function generates a list of linkage groups consisting of all the clustered contigs. After reorientation and merging, all contigs within the linkage groups are highly similar, while the contigs between linkage groups are highly dissimilar. The similarity between linkage groups can be visualized using plotLGDistances.

```
> plotLGDistances(exampleLGList, exampleWCMatrix)
```



While the similarity within linkage groups can be visualized using plotLinkageGroup (here, the first linkage group is used for creating this heatmap).

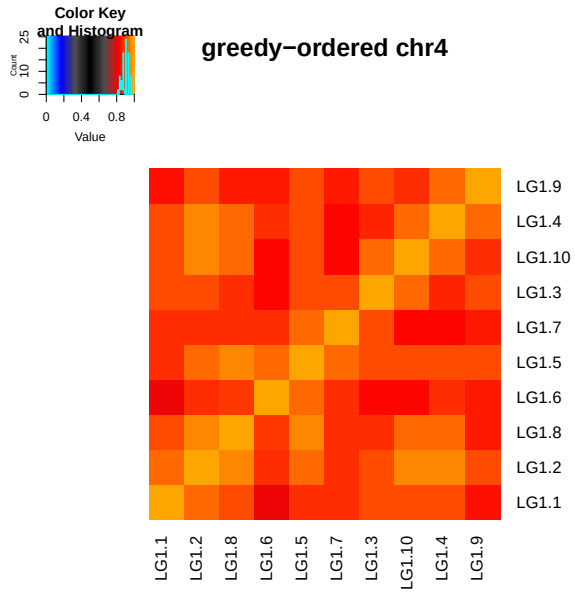
```
> plotLinkageGroup(exampleLGList[[1]], exampleWCMatrix)
```



6 Ordering contigs within chromosomes

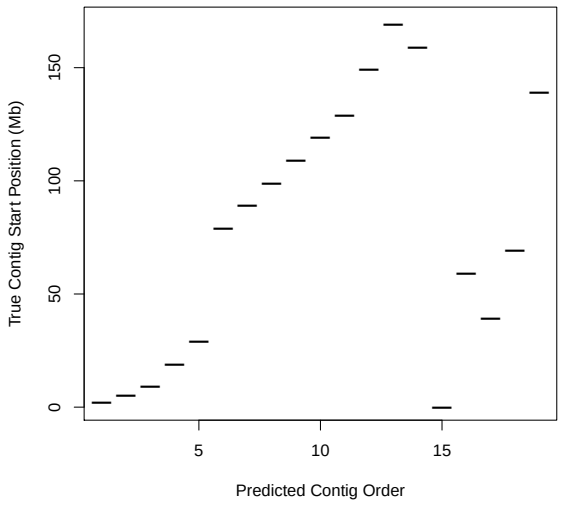
With contigs clustered to chromosomes, we can then order them within chromosomes. Just as meiotic recombination shuffles loci and allows genetic distances between them to be determined, sister chromatid exchanges (SCE) events reshuffle templates, and similarly allow us to infer a linkage distance. We have employed a greedy algorithm to do this, but have an argument allowing a TSP solution as an alternative. Contigs are ordered by similarity across libraries, then by contig name. Contigs that are zero distance apart (ie have no SCE events between them and are therefore unordered) are returned in contig name order. The output is split into sub-linkage groups, so Linkage group 1 will be split into a number of groups depending on the number of SCE events that occur within the chromosome.

```
> contigOrder <- orderAllLinkageGroups(exampleLGList,
+ exampleWCMatrix,
+ exampleStrandFreq,
+ exampleReadCounts,
+ whichLG=1,
+ saveOrdered=TRUE)
> contigOrder

A data.frame of 1 LGs split into 10 sub-groups from 19 ordered fragments.

 LG1.1 LG1.10  LG1.2  LG1.3  LG1.4  LG1.5  LG1.6  LG1.7  LG1.8  LG1.9
     1      1      7      1      2      1      1      1      3      1
```



If the assembly is mostly complete and you wish to compare the actual location of the fragments against the output of orderAllLinkageGroups, contiBAIT has the built in plotContigOrder function

```
> plotContigOrder(contigOrder$contig)
```



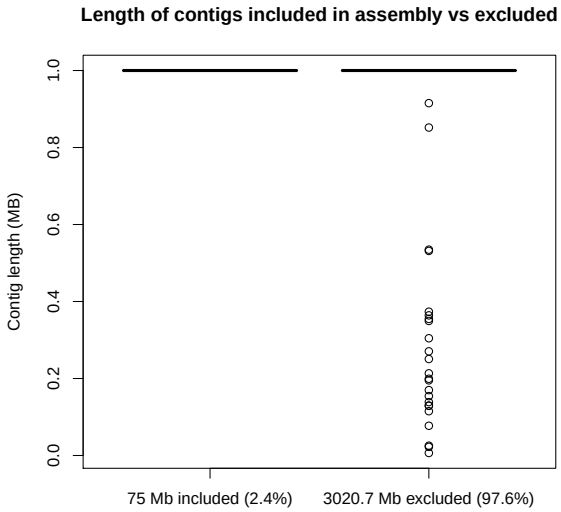
7 Writing out to a BED file

This file can be passed to bedtools along with the original (draft) reference genome to create a new FASTA file containing the assembled genome.

8 Additional plotting functions

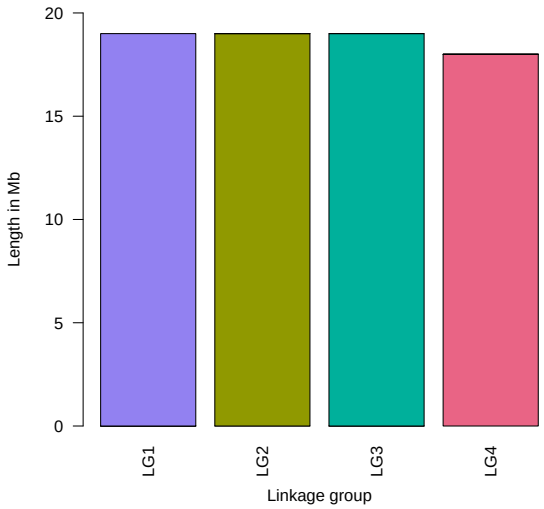
Using a chromosome table instance, comparisons can be made between the portion of contigs that are included in the analysis verses those that are excluded based on either poor coverage or non-Strand-seq patterning. The code below generates a box plot of contig sizes that are included in the analysis. Note, since sample data are uniform 1 Mb framgents, the box plot does not deviate from the median. The example bam files contain reads from 76 separate 1 Mb fragments from chromsomes 1, 2, 3, and 4. Since the assembly is >3 Gb in size, only a few percent of the assembly will be included in our analysis.

```
> makeBoxPlot(dividedChrTable, exampleLGList)
```



Furthermore we can determine the proportion of assembly fragments in each linkage group in a barplot. If data are in the format chr:start-end, then each unique chromosome name will have a unique color. If data are not in this format, then each fragment will have a unique color. Here, all fragments from chr1 will be colored differently to fragments from chr2, etc.

```
> barplotLinkageGroupCalls(exampleLGList, dividedChrTable)
```



Note that if clustering did not occur correctly, some bars would be a mixture of colors. While the above displays the proportion of fragments from one chromosome that has clustered into each linkage group, but omitting the by='chr' parameter, the plot changes to the proportion of linkage groups within each chromosome.