contiBAIT: Improving Genome Assemblies Using Strand-seq Data

Kieran O'Neill, Mark Hills and Mike Gottlieb

May 2, 2019

`koneill@bcgsc.ca`

Contents

1

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 GRanges chromosome table instance can be made, representing the contig name and length. This is generated with makeChrTable, where the resulting object is similar to the header portion of a BAM file. Note a meta column with a name formed of the contig and start and end locations is generated for downstream workflows.

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

ChrTable object with 24 ranges and 1 metadata column:
       seqnames        ranges strand |               name
          <Rle>     <IRanges>  <Rle> |        <character>
   [1]     chr1 1-249250621        * | chr1:1-249250621
   [2]     chr2 1-243199373        * | chr2:1-243199373
   [3]     chr3 1-198022430        * | chr3:1-198022430
   [4]     chr4 1-191154276        * | chr4:1-191154276
   [5]     chr5 1-180915260        * | chr5:1-180915260
   ...      ...           ...    ... .                ...
  [20]    chr20  1-63025520        * | chr20:1-63025520
  [21]    chr21  1-48129895        * | chr21:1-48129895
  [22]    chr22  1-51304566        * | chr22:1-51304566
  [23]     chrX 1-155270560        * | chrX:1-155270560
  [24]     chrY  1-59373566        * |  chrY:1-59373566
  -------
  seqinfo: 24 sequences from an unspecified genome; no seqlengths
```

3.2 Splitting a chromosome table instance

We can also 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 bins 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

3

used with the filter argument in strandSeqFreqTable to generate a sub-divided table.

```
> exampleDividedChr <- makeChrTable(bamFileList[1], splitBy=1000000)
> exampleDividedChr

ChrTable object with 3089 ranges and 1 metadata column:
         seqnames              ranges strand |                      name
            <Rle>           <IRanges>  <Rle> |               <character>
    [1]      chr1           1-1001007      * |           chr1:1-1001007
    [2]      chr1     1001008-2002013      * |     chr1:1001008-2002013
    [3]      chr1     2002014-3003020      * |     chr1:2002014-3003020
    [4]      chr1     3003021-4004026      * |     chr1:3003021-4004026
    [5]      chr1     4004027-5005033      * |     chr1:4004027-5005033
    ...       ...                 ...    ... .                      ...
 [3085]      chrY 54341909-55348239      * | chrY:54341909-55348239
 [3086]      chrY 55348240-56354571      * | chrY:55348240-56354571
 [3087]      chrY 56354572-57360903      * | chrY:56354572-57360903
 [3088]      chrY 57360904-58367234      * | chrY:57360904-58367234
 [3089]      chrY 58367235-59373566      * | chrY:58367235-59373566
    -------
    seqinfo: 24 sequences from an unspecified genome; no seqlengths
```

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 with four regions that overlap a 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)

ChrTable object with 26 ranges and 1 metadata column:
       seqnames           ranges strand |                   name
          <Rle>        <IRanges>  <Rle> |            <character>
   [1]      chr1    1-249250621      * |       chr1:1-249250621
   [2]      chr2    1-243199373      * |       chr2:1-243199373
   [3]      chr3    1-198022430      * |       chr3:1-198022430
   [4]      chr4      1-1429358      * |         chr4:1-1429358
   [5]      chr4 1429359-1434206      * | chr4:1429359-1434206
   ...       ...              ...    ... .                    ...
  [22]     chr20    1-63025520      * |       chr20:1-63025520
  [23]     chr21    1-48129895      * |       chr21:1-48129895
  [24]     chr22    1-51304566      * |       chr22:1-51304566
  [25]      chrX    1-155270560      * |       chrX:1-155270560
  [26]      chrY     1-59373566      * |        chrY:1-59373566
   -------
   seqinfo: 67 sequences from an unspecified genome; no seqlengths
```

What is returned is a GRanges 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 GRanges 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=exampleDividedChr,
+ qual=10,
+ pairedEnd=FALSE,
+ BAITtables=TRUE)
```

This returns a list of two data.frames if the BAITtables argument is set to FALSE, or four if it is set to TRUE. 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 145 contigs over 35 libraries.

$countTable
A matrix of read counts for 145 contigs over 35 libraries.

$WatsonReads
A matrix of read counts for 3089 contigs over 35 libraries.

$CrickReads
A matrix of read counts for 3089 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)
```

WC distributions from 35 libraries
with an average of 72 fragments
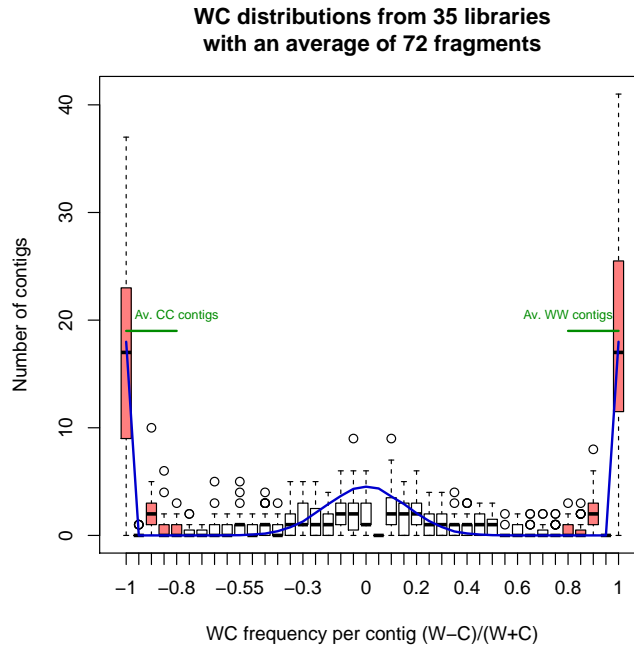
4   Creating a strand state matrix

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

7

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.

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

A strand state matrix for 76 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. It is important to note however 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, by default the function performs clustering between homozygous calls (WW or CC treated the same) and heterozygous calls (WC) to identify contigs that belong together despite their misorientation status with respect to each other. Using the clusterBy='homo' option will perform the clustering just between WW and CC calls (ignoring WC calls) and misoriented fragments from the same chromosome will cluster into different linkage groups. Once clustered, the misorientated fragments are identified in each linkage group using reorientAndMergeLGs. Since chromosome orientation cannot be determined by sequence alone, the largest sub-group is arbitrarily considered "+", while the smaller group is considered "-". The product of this function is a list where the first element is a StrandStateMatrix instance that has been correctly oriented, the second element is a data.frame of contigs and orientations, and the third element is a recomputed LinkageGroupList that merges groups that were previously discordant based on misorientation status. This merger occurs by computing a consensus strand state across libraries within each linkage group and comparing them. Note for the purposes of this example, we are using the

8

argument randomise=FALSE to ensure conformity of the vignette when running sweave. Randomisation is recommended for most applications and is set to TRUE by default.

```
> exampleWCMatrix <- exampleStrandStateMatrix[[1]]
> clusteredContigs <- clusterContigs(exampleWCMatrix, randomise=FALSE)

Initializing contig chr2:3002462-4003282 [1/76] as LG1
Clustering contig chr2:170139480-171140300 [2/76]
  -> Adding chr2:170139480-171140300 to LG0 for a cluster of 2
Clustering contig chr2:20016410-21017230 [3/76]
  -> Adding chr2:20016410-21017230 to LG0 for a cluster of 3
Clustering contig chr4:20016156-21016962 [4/76]
Clustering contig chr4:100080774-101081581 [5/76]
  -> Adding chr4:100080774-101081581 to LG1 for a cluster of 2
Clustering contig chr2:6004924-7005743 [6/76]
  -> Adding chr2:6004924-7005743 to LG0 for a cluster of 4
Clustering contig chr2:190155889-191156709 [7/76]
  -> Adding chr2:190155889-191156709 to LG0 for a cluster of 5
Clustering contig chr3:3000341-4000453 [8/76]
Clustering contig chr1:20020131-21021137 [9/76]
Clustering contig chr4:140113083-141113890 [10/76]
  -> Adding chr4:140113083-141113890 to LG1 for a cluster of 3
Clustering contig chr3:30003399-31003512 [11/76]
  -> Adding chr3:30003399-31003512 to LG2 for a cluster of 2
Clustering contig chr3:6000681-7000793 [12/76]
  -> Adding chr3:6000681-7000793 to LG2 for a cluster of 3
Clustering contig chr2:60049229-61050048 [13/76]
  -> Adding chr2:60049229-61050048 to LG0 for a cluster of 6
Clustering contig chr4:10008078-11008885 [14/76]
  -> Adding chr4:10008078-11008885 to LG1 for a cluster of 4
Clustering contig chr3:190021525-191021637 [15/76]
  -> Adding chr3:190021525-191021637 to LG2 for a cluster of 4
Clustering contig chr3:160018126-161018239 [16/76]
  -> Adding chr3:160018126-161018239 to LG2 for a cluster of 5
Clustering contig chr2:80065638-81066458 [17/76]
  -> Adding chr2:80065638-81066458 to LG0 for a cluster of 7
Clustering contig chr2:240196913-241197732 [18/76]
  -> Adding chr2:240196913-241197732 to LG0 for a cluster of 8
Clustering contig chr1:60060392-61061397 [19/76]
  -> Adding chr1:60060392-61061397 to LG3 for a cluster of 2
Clustering contig chr2:40032820-41033639 [20/76]
  -> Adding chr2:40032820-41033639 to LG0 for a cluster of 9
Clustering contig chr4:80064619-81065426 [21/76]
  -> Adding chr4:80064619-81065426 to LG1 for a cluster of 5
Clustering contig chr2:50041024-51041844 [22/76]
```

```
  -> Adding chr2:50041024-51041844 to LG0 for a cluster of 10
Clustering contig chr4:6004847-7005654 [23/76]
  -> Adding chr4:6004847-7005654 to LG1 for a cluster of 6
Clustering contig chr2:200164094-201164913 [24/76]
  -> Adding chr2:200164094-201164913 to LG0 for a cluster of 11
Clustering contig chr2:230188708-231189527 [25/76]
  -> Adding chr2:230188708-231189527 to LG0 for a cluster of 12
Clustering contig chr1:10010066-11011072 [26/76]
  -> Adding chr1:10010066-11011072 to LG3 for a cluster of 3
Clustering contig chr3:130014728-131014840 [27/76]
  -> Adding chr3:130014728-131014840 to LG2 for a cluster of 6
Clustering contig chr2:10008206-11009025 [28/76]
  -> Adding chr2:10008206-11009025 to LG0 for a cluster of 13
Clustering contig chr2:220180503-221181323 [29/76]
  -> Adding chr2:220180503-221181323 to LG0 for a cluster of 14
Clustering contig chr3:150016993-151017106 [30/76]
  -> Adding chr3:150016993-151017106 to LG2 for a cluster of 7
Clustering contig chr1:40040261-41041267 [31/76]
  -> Adding chr1:40040261-41041267 to LG3 for a cluster of 4
Clustering contig chr2:30024615-31025434 [32/76]
  -> Adding chr2:30024615-31025434 to LG0 for a cluster of 15
Clustering contig chr4:90072696-91073503 [33/76]
  -> Adding chr4:90072696-91073503 to LG1 for a cluster of 7
Clustering contig chr4:110088851-111089658 [34/76]
  -> Adding chr4:110088851-111089658 to LG1 for a cluster of 8
Clustering contig chr1:30030196-31031202 [35/76]
  -> Adding chr1:30030196-31031202 to LG3 for a cluster of 5
Clustering contig chr3:50005665-51005777 [36/76]
  -> Adding chr3:50005665-51005777 to LG2 for a cluster of 8
Clustering contig chr1:180181173-181182178 [37/76]
  -> Adding chr1:180181173-181182178 to LG3 for a cluster of 6
Clustering contig chr2:1000821-2001641 [38/76]
  -> Adding chr2:1000821-2001641 to LG0 for a cluster of 16
Clustering contig chr1:50050327-51051332 [39/76]
  -> Adding chr1:50050327-51051332 to LG3 for a cluster of 7
Clustering contig chr1:220221433-221222439 [40/76]
  -> Adding chr1:220221433-221222439 to LG3 for a cluster of 8
Clustering contig chr3:1000114-2000227 [41/76]
  -> Adding chr3:1000114-2000227 to LG2 for a cluster of 9
Clustering contig chr4:150121160-151121967 [42/76]
  -> Adding chr4:150121160-151121967 to LG1 for a cluster of 9
Clustering contig chr4:170137315-171138121 [43/76]
  -> Adding chr4:170137315-171138121 to LG1 for a cluster of 10
Clustering contig chr4:160129237-161130044 [44/76]
  -> Adding chr4:160129237-161130044 to LG1 for a cluster of 11
Clustering contig chr2:150123071-151123890 [45/76]
```

```
  -> Adding chr2:150123071-151123890 to LG0 for a cluster of 17
Clustering contig chr1:70070457-71071462 [46/76]
  -> Adding chr1:70070457-71071462 to LG3 for a cluster of 9
Clustering contig chr4:130105006-131105812 [47/76]
  -> Adding chr4:130105006-131105812 to LG1 for a cluster of 12
Clustering contig chr3:10001134-11001246 [48/76]
  -> Adding chr3:10001134-11001246 to LG2 for a cluster of 10
Clustering contig chr3:110012462-111012574 [49/76]
  -> Adding chr3:110012462-111012574 to LG2 for a cluster of 11
Clustering contig chr4:30024233-31025040 [50/76]
  -> Adding chr4:30024233-31025040 to LG1 for a cluster of 13
Clustering contig chr3:40004532-41004645 [51/76]
  -> Adding chr3:40004532-41004645 to LG2 for a cluster of 12
Clustering contig chr3:20002267-21002379 [52/76]
  -> Adding chr3:20002267-21002379 to LG2 for a cluster of 13
Clustering contig chr3:100011329-101011442 [53/76]
  -> Adding chr3:100011329-101011442 to LG2 for a cluster of 14
Clustering contig chr3:170019259-171019371 [54/76]
  -> Adding chr3:170019259-171019371 to LG2 for a cluster of 15
Clustering contig chr4:60048465-61049271 [55/76]
  -> Adding chr4:60048465-61049271 to LG1 for a cluster of 14
Clustering contig chr1:1001008-2002013 [56/76]
  -> Adding chr1:1001008-2002013 to LG3 for a cluster of 10
Clustering contig chr3:70007931-71008043 [57/76]
  -> Adding chr3:70007931-71008043 to LG2 for a cluster of 16
Clustering contig chr1:90090587-91091592 [58/76]
  -> Adding chr1:90090587-91091592 to LG3 for a cluster of 11
Clustering contig chr2:70057434-71058253 [59/76]
  -> Adding chr2:70057434-71058253 to LG0 for a cluster of 18
Clustering contig chr2:130106661-131107481 [60/76]
  -> Adding chr2:130106661-131107481 to LG0 for a cluster of 19
Clustering contig chr1:240241563-241242569 [61/76]
  -> Adding chr1:240241563-241242569 to LG3 for a cluster of 12
Clustering contig chr1:110110717-111111723 [62/76]
  -> Adding chr1:110110717-111111723 to LG3 for a cluster of 13
Clustering contig chr1:160161043-161162048 [63/76]
  -> Adding chr1:160161043-161162048 to LG3 for a cluster of 14
Clustering contig chr1:100100652-101101658 [64/76]
  -> Adding chr1:100100652-101101658 to LG3 for a cluster of 15
Clustering contig chr3:120013595-121013707 [65/76]
  -> Adding chr3:120013595-121013707 to LG2 for a cluster of 17
Clustering contig chr1:150150978-151151983 [66/76]
  -> Adding chr1:150150978-151151983 to LG3 for a cluster of 16
Clustering contig chr4:120096928-121097735 [67/76]
  -> Adding chr4:120096928-121097735 to LG1 for a cluster of 15
Clustering contig chr3:60006798-61006910 [68/76]
```

```
   -> Adding chr3:60006798-61006910 to LG2 for a cluster of 18
Clustering contig chr4:70056542-71057349 [69/76]
  -> Adding chr4:70056542-71057349 to LG1 for a cluster of 16
Clustering contig chr3:80009064-81009176 [70/76]
  -> Adding chr3:80009064-81009176 to LG2 for a cluster of 19
Clustering contig chr1:200201303-201202309 [71/76]
  -> Adding chr1:200201303-201202309 to LG3 for a cluster of 17
Clustering contig chr1:80080522-81081527 [72/76]
  -> Adding chr1:80080522-81081527 to LG3 for a cluster of 18
Clustering contig chr4:40032310-41033117 [73/76]
  -> Adding chr4:40032310-41033117 to LG1 for a cluster of 17
Clustering contig chr4:3002424-4003231 [74/76]
  -> Adding chr4:3002424-4003231 to LG1 for a cluster of 18
Clustering contig chr4:1000809-2001615 [75/76]
  -> Adding chr4:1000809-2001615 to LG1 for a cluster of 19
Clustering contig chr1:120120782-121121788 [76/76]

> reorientedMatrix <- reorientAndMergeLGs(clusteredContigs,
+  exampleWCMatrix)
> exampleLGList <- reorientedMatrix[[3]]
> exampleLGList

A linkage group list containing 5 linkage groups.

  NumberOfContigs
1              19
2              19
3              19
4              18
5               1

> exampleLGList[[1]]

 [1] "chr4:20016156-21016962"   "chr4:100080774-101081581"
 [3] "chr4:140113083-141113890" "chr4:10008078-11008885"
 [5] "chr4:80064619-81065426"   "chr4:6004847-7005654"
 [7] "chr4:90072696-91073503"   "chr4:110088851-111089658"
 [9] "chr4:150121160-151121967" "chr4:170137315-171138121"
[11] "chr4:160129237-161130044" "chr4:130105006-131105812"
[13] "chr4:30024233-31025040"   "chr4:60048465-61049271"
[15] "chr4:120096928-121097735" "chr4:70056542-71057349"
[17] "chr4:40032310-41033117"   "chr4:3002424-4003231"
[19] "chr4:1000809-2001615"
```

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)
```
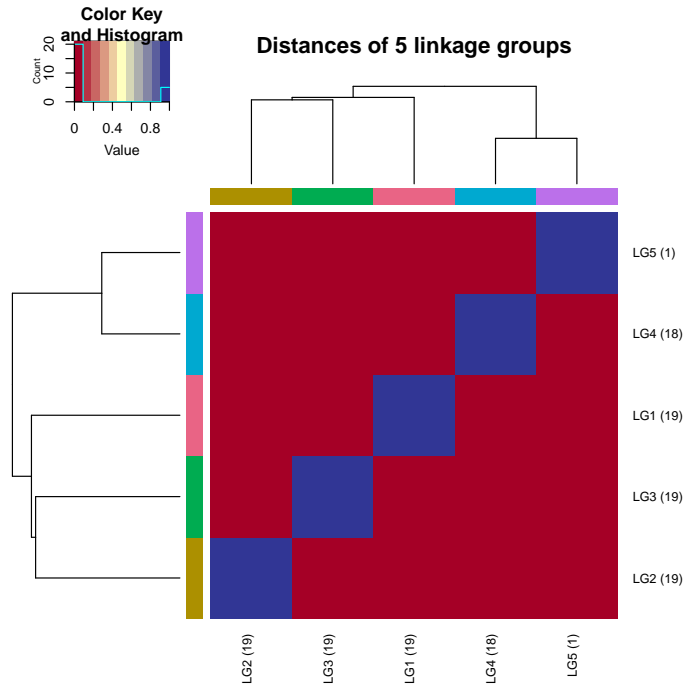

While the similarity within linkage groups can be visualized using plotLinkageGroup (here, the first linkage group is used for creating this heatmap). Note, side by side comparisons of linkage group members can be performs with multiple lg options (e.g. lg=1:2 will plot the first two linkage groups, lg=c(1,4) will plot the first and forth etc.).

```
> plotLGDistances(exampleLGList, exampleWCMatrix, lg=1)
```

Distances of 19 linkage groups
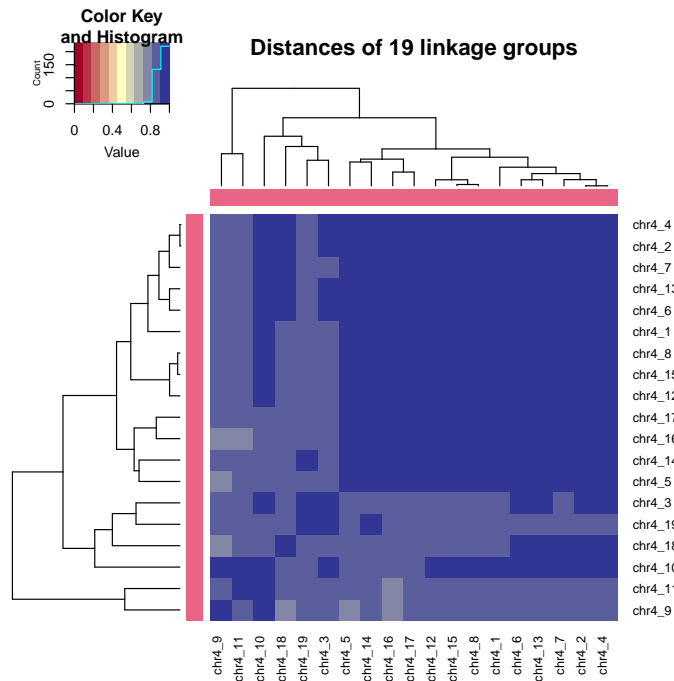
6 Ordering contigs within chromosomes

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. The output will be an S4 object of type contigOrdering.

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

14

```
A matrix of 1 LGs split into 12 sub-groups from 19 ordered fragments.
```
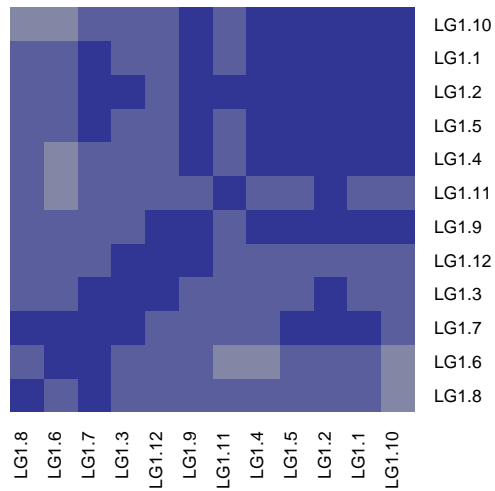
```
 LG1.1 LG1.10 LG1.11 LG1.12  LG1.2  LG1.3
     3      2      1      1      3      1
...
LG1.4 LG1.5 LG1.6 LG1.7 LG1.8 LG1.9
    1     3     1     1     1     1
```
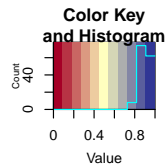

If the assembly is mostly complete and you wish to compare the actual location of the fragments in the assembly you're working with against the output of orderAllLinkageGroups, contiBAIT has the built in plotContigOrder function. This assumes that the contig name from the contigOrdering object is in a format of chr:start-end.

```
> plotContigOrder(contigOrder[[1]])
```

chr4 plot of 19 fragments (19 sub–linkage groups)
Spearman correlation = 0.48
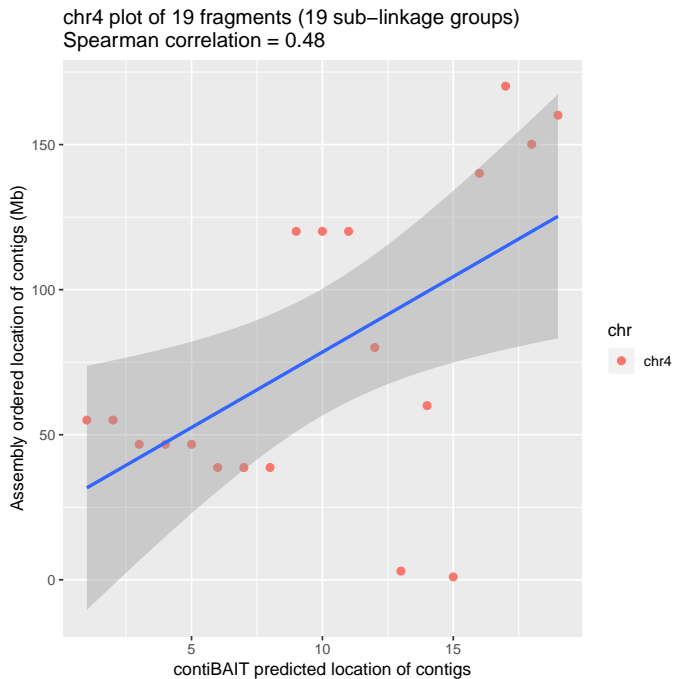
Alternatively, all contigs can be ordered simultaneously by omitting the whichLG argument. If saveOrdered is set to TRUE, plots will be generated for every linkage group. By ordering all of the linkage groups we can proceed to create BED files of these data for the new assemblies.

```
> contigOrderAll <- orderAllLinkageGroups(exampleLGList,
+ exampleWCMatrix,
+ exampleStrandFreq,
+ exampleReadCounts)
> contigOrderAll[[1]]

A matrix of 5 LGs split into 14 sub-groups from 76 ordered fragments.

 LG1.1 LG1.10 LG1.11 LG1.12  LG1.2  LG1.3
     3      2      1      1      3      1
...
LG4.5 LG4.6 LG4.7 LG4.8 LG4.9 LG5.1
    1     1     3     1     2     1
```

7   Checking order using BAIT ideograms

It is possible to visually validate the ordering by creating ideogram plots of the data. The supplied test data comprises of reads from the first four chromosomes. Below is example code that will plot the second library from the

output of strandSeqFreqTable. Note the third and forth elements of the strand-FrequencyList are only generated if BAITtables is set to TRUE when running strandSeqFreqTable. The plot below shows the location of the reads and highlights an SCE on chromosome 3. Note to only display the first library, we need to subset the strandReadMatrix and retain these as strandReadMatrix objects.

```
> # extract elements from strandSeqFreqTable list
> WatsonFreqList <- strandFrequencyList[[3]]
> CrickFreqList <- strandFrequencyList[[4]]
> # subset elements to only analyze one library
> singleWatsonLibrary <- StrandReadMatrix(WatsonFreqList[,2, drop=FALSE])
> singleCrickLibrary <- StrandReadMatrix(CrickFreqList[,2, drop=FALSE])
> # Run ideogram plotter
> ideogramPlot(singleWatsonLibrary,
+ singleCrickLibrary,
+ exampleDividedChr)
```
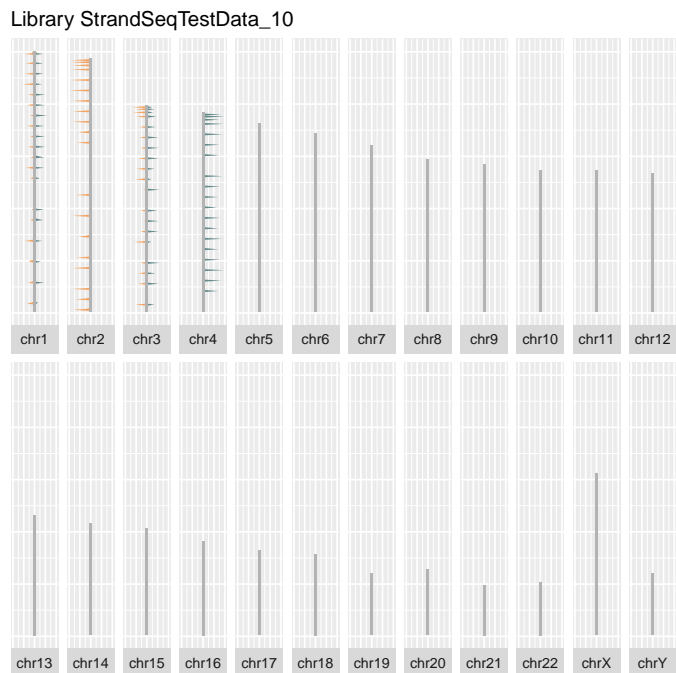

If chromosome builds are not complete (and so each contig has not been assigned a chromosome in the chrTable instance), these ideograms can be plotted using only the represented fragments in the order given to the function from the orderedContig object generated from orderAllLinkageGroups. Here we will use the orderedContig object representing all 4 linkage groups.

```
> ideogramPlot(singleWatsonLibrary,
+ singleCrickLibrary,
```

17

```
+ exampleDividedChr,
+ orderFrame=contigOrderAll[[1]])
```

Library StrandSeqTestData_10
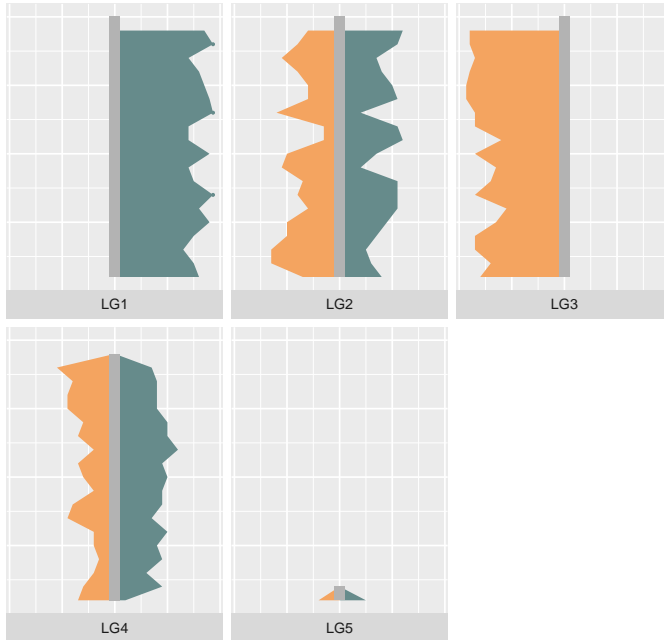

Alternatively, all libraries can be compared side by side for a single chromosome. Because this will print to multiple pages, the showPage option can be used to limit the output to a user=specified page. Here we will use the orderedContig object representing just one linkage group.

```
> ideogramPlot(WatsonFreqList,
+ CrickFreqList,
+ exampleDividedChr,
+ orderFrame=contigOrder[[1]],
+ plotBy='chr',
+ showPage=1)

[1] "LG1"
```

Chromosome LG1 (Page1)
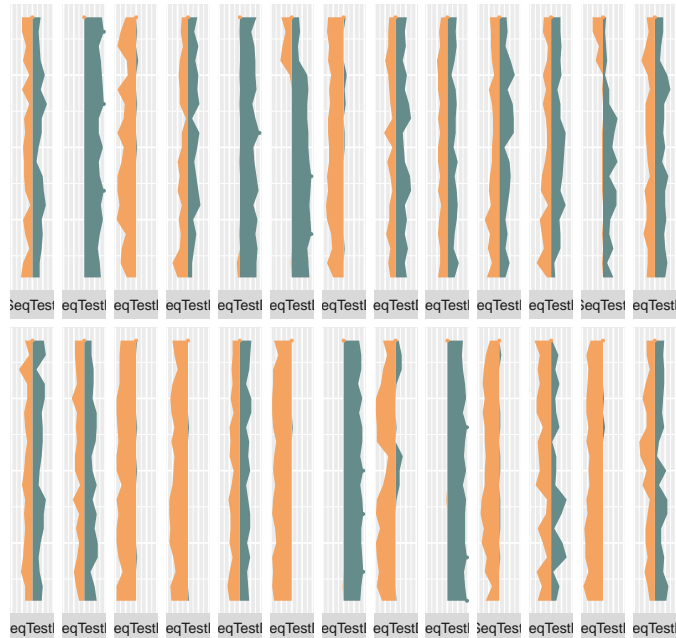
8 Writing out to a BED file

This file can be passed to bedtools along with the original (draft) reference genome to create a new FASTA file containing the assembled genome. the writeBed function requires a chrTable of class ChrTable with which to extract the contig names and locations, the orientation information derived from reorientLinkageGroups to populate the strand column, the library weight to populate the score column, and an object of type ContigOrdering to invoke the actual order of fragments. A fileName can be supplied, or the default is used. BED files will be written to the working directory

```
> writeBed(exampleDividedChr,
+ reorientedMatrix[[2]],
+ contigOrder[[1]])
```

9   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(exampleDividedChr, exampleLGList)
```


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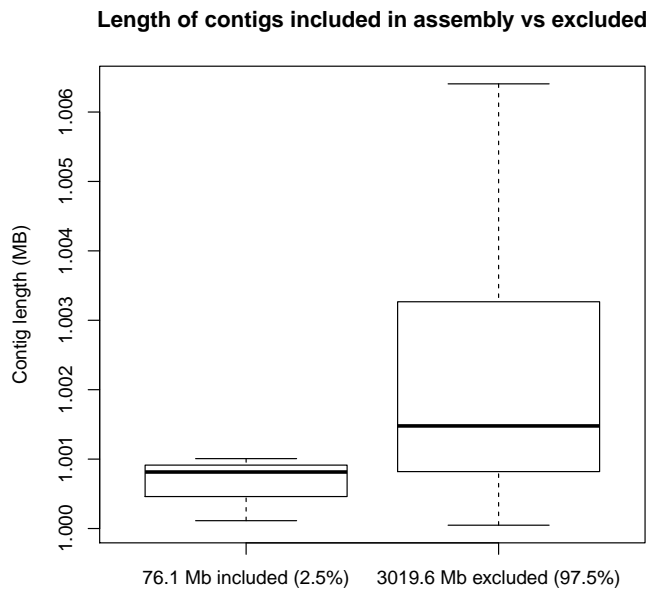
Furthermore we can determine the proportion of assembly fragments in each linkage group in a barplot. If data are in the format chr:start-end, then each unique chromosome name will have a unique color. If data are not in this format, then each fragment will have a unique color. Here, all fragments from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

```
> barplotLinkageGroupCalls(exampleLGList, exampleDividedChr)
```

Barplot of 4 contigs clustered into 5 linkage groups
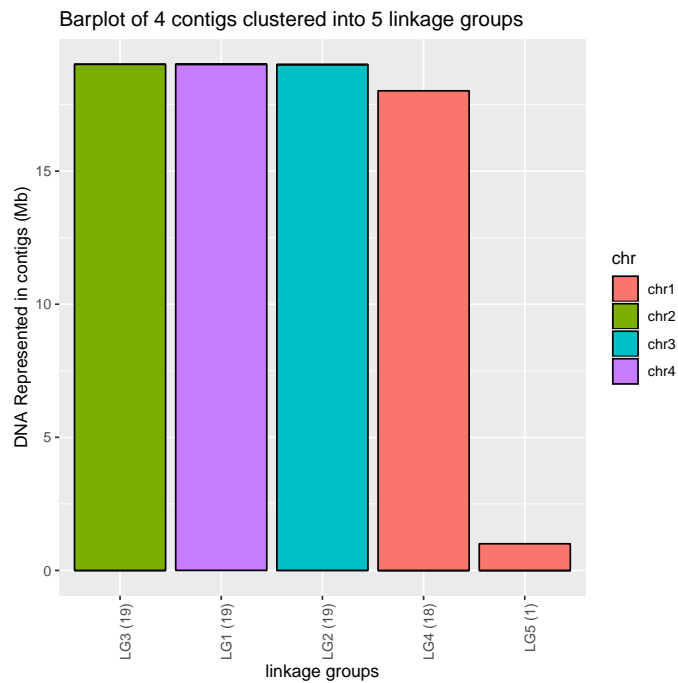
Note that if clustering did not occur correctly, some bars would be a mixture of colors. While the above displays the proportion of fragments from one chromosome that has clustered into each linkage group, but omitting the by='chr' parameter, the plot changes to the proportion of linkage groups within each chromosome.

10 Flow diagram

Here is the basic functionality of the contiBAIT package. The input BAM file(s) and output BED file are displayed in the grey ovals. All plotting functions are shown in blue hexagons and all analysis functions are in white boxes.