ConTExt User Guide

Michael Peter McGurk

April 3, 2018

Contents

1 Overview

Repeat-derived NGS reads are often ignored in sequencing analyses because an essential step in most NGS pipelines is identifying the genomic locus from which a given read originates. For reads originating in repetitive sequence, this is often impossible. Consequently, asking questions about repeats using NGS data requires phrasing the questions differently than one would for unique sequence. ConTExt is a set of tools designed to do just that. Instead of trying to identify the locus from which a repeat-derived read originates, it seeks to determine the repeat-family from which the read originates and the location within the repeat-family (e.g. for a TE-derived read, does it sit near the 5' end of the element or the nearer to the 3' end?). Once reads are organized in this manner, ConTExt makes use of read pair information, coverage, and sequence polymorphism to identify structures involving repetitive sequence (tandem junctions, deletions within a repeat, insertions into unique or repeated sequence), estimate their copy number, and identify the proportion repeats harboring variant alleles. These structures can be examined for population variation.

A guiding tenet of ConTExt is that visualizing data is good practice. Tables are useful for crunching numbers, but visually exploring data is often essential to understanding and trusting it. Consequently, we organize repeat-derived reads in a manner conducive to understandable visualizations. To this end, ConTExt ultimately aligns repeat-derived reads to the consensus sequences of known repeat families. It is much easier to make inferences about

the structures formed by Hobo transposon when all of the Hobo-derived reads are gathered in one place and organized with respect to a single Hobo consensus sequence. However, aligning directly to consensus sequences is likely to miss reads originating from old, divergent copies of a repeat. To get around this, we use a two-step procedure wherein reads are first aligned to the set of all individual instances of repeats identified in the reference genome, and then these alignments are collapsed onto the RepBase consensus sequences. The first step provides power to detect divergent elements, and the second ensures ease of visualization and analysis. In D. melanogaster, we found that this two-step procedure identified about 25% more repeat-derived reads than did aligning directly to the consensus sequences.

Once the data are aligned, ConTExt utilizes two-dimensional representations of the alignments. Because each read pair contains two reads, each of which aligns to a different position, each read pair can be thought of as a point in a 2D scatter plot. A variety of summaries are computed for each data set, including the distributions of mate pair distances and the read depth distributions conditioned on %GC. Structures involving repeats are identified by fitting a generative model to the alignments using Expectation Maximization.

Most of the major scripts that actually organize the data are designed to be run from command-prompt and all of the necessary steps are called automatically. The individual functions are not intended to be called directly by the user. Extensive documentation for using functions is restricted to those intended to be called manually.

I automate as much of the pipeline as is possible, but cookie cutter solutions do always not exist for every step of analysis. Where user decision are necessary, this manual provides guidance and I have included some tools that made my life easier–hopefully they will do the same for yours.

1.1 Major Concepts and Considerations

1.2 Installing ConTExt

Download the repository from GitHub. In the ConTExt directory call:

```
python setup.py install
```

1.2.1 Running the pipeline

ConTExt is comprised of Python scripts which are intended to be called from command-line. The main scripts you will be using are located in the ConTExt directory, while additional scripts containing many of the under-the-hood functions are located in ./ConTExt/tools. Unless otherwise stated, the scripts in tools scripts are not intend to be called manually and I provided no user documentation for them.

I aimed to make these tools simple to use. Consequently most of the functions require only three arguments: an input directory, an output directory, and a specification file. The specification file contains about a number of parameters and file paths required for the analysis and must be modified to suit a given analysis. This file is sufficiently important that a section of this manual is dedicated to it. However, usage

Usage Note: When entering directories as arguments, do not include a trailing slash. For example, enter "/home/me/mydata" rather than "/home/me/mydata/". When the scripts need to access a directory's contents, they will append slashes when necessary.

To align the data:

python AlignmentPipeline.py -i <input directory> -o <alignment directory> -spec <specification file>

To cluster the aligned data:

OMP_NUM_THREADS=1 nohup python ExpMax.py -i <alignment directory> -o <clustered directory> -spec <specification file>

None of the functions in ExpMax are currently designed to run in parallel, but Scikit-Learn's Gaussian Process functions will use every processer available. To avoid sporadic spikes in processor usage in a Linux environment, OMP_NUM_THREADS=1 is needed to limit the process to a single CPU.

To classify the junctions and estimate local GC-composition:

python SummarizeOutput.py" -i <cluster directory> -spec <specification file>

Deprecation Warning: In the near future, SummarizeOutput.py will be incorporated into ExpMax.py

Finally, to match junctions across samples:

python MatchJunctions.py -i <clustered directory> -o <final directory> -spec <specification file>

Each of these steps is explained in

2 Preparing the repeat index and reference genome

2.1 Preparing the repeat consensus index

Lists of repeat consensus sequences can be obtained from a number of sources, the most comprehensive of which is RepBase. I find it useful to supplement this

with sequences for interesting multigene families, like the ribosomal RNA genes and the tandemly arrayed (in Drosophila) histone genes. Some exploration of the literature is useful when constructing a good repeat index.

However, as useful as RepBase libraries are, they often include redundant sequences and multiple sub-families of individual repeat families. These can greatly complicate downstream analyses with ConTExt, and need to be removed. Manually curating a *.fasta containing hundreds of entries is not a trivial task. To simplify it, I provide a submodule called FindHomology which carries out all pairwise alignments within your index of repeat consensus sequences and then uses the Louvain community finding algorithm to identify groups of potentially redundant entries. This greatly reduces the effort required to obtain a suitable repeat index.

FindHomology first uses BLASTn to carry out all pairwise alignments. It then represents these alignments as a graph, which is a structure useful for representing relationships between objects. In this case, the objects are sequences and the relationships are alignments. Therefore, we represent each sequence in the repeat index with a node and we connect a pair of sequences with an edge if BLASTn identified an alignment between them with greater than 80% identity. Groups of sequences that tend to align to each other create communities in this graph, which are readily identified by the Louvain algorithm.

We do not implement the Louvain algorithm ourselves, but rather make use of Thomas Aynaud's easy to use implementation. This dependency is not available through PIP but can be found at https://github.com/taynaud/python-louvain/. Installation is as easy as running the module's setup.py.

Additional dependencies for FindHomology are Biopython and Networkx, both of which can be installed with PIP.

To run the script from command-line:

python ./tools/FindHomology.py -C [<0-100> percent identity cutoff] -B [Path to Blast directory] -i [Path to repeat index] -o [Path to output directory]

For example:

python FindHomology.py -B "C:/Program Files/NCBI/blast-2.6.0+" -i c:/barbashlab/Repbase_19.06_DM_Curated_5-1-15_sim.fa -o c:/barbashlab/find_homology_test

Note: This script creates a temporary folder called /temp in the parent of the output directory. Make sure no such folder already exists.

The output folder will contain a few different files:

- _err.txt
- alignment_output.tsv
- alignments_<##>_<##>.tsv

- output.csv
- communities.txt
- community_<##>.fa

The interesting files are communities.txt and the community_<#>.fa. Communities.txt lists all of the communities identified and the names of repeat entries comprising each community. Removing redundant entries is simply a matter of going through each community and removing any entry that contains significant homology with another entry. In the extreme case, this means paring each community down to a single entry. In practice, this isn't always necessary and some personal judgement is required here. To aid in this, each community is assigned a number and has a corresponding community_<#>.fa file which contains the sequences of all repeats that comprise the community. Aligning the community_<#>.fa against itself provides insight into the nature of the homologies that create the cluster. In some cases, the community may by driven by a short tract of sequence with low (<85%) homology. In this case, it may be fine to leave both entries in the index; though, be sure to remember that this homology exists if you happen to see some interesting signal involving one of these elements in downstream analyses. Often, communities contain a full-length element along with internally deleted, fragmented, or variant elements. In this case it makes sense to retain only one full-length element and it can be useful to align the community_<#>.fa against the reference genome to get a sense of which entry appears most representative. Remember, though, non-autonomous DNA transposons are often internally deleted and abundant, so the most abundant variant isn't necessarily the best. Looking for ORFs within repeat entries may help resolve ambiguities here. In others cases, a community may be comprised of many variants of a very diverse repeat family. In this case, we favor examining all pairwise alignments in the web BLAST utility and retaining a few entries based on the distance tree. The most irritating instances are composite repeats, like the Suppressor of Stellate sequence in D. melanogaster, which contains a subsequence of a tandemly repeated gene (Stellate) flanked by the ends of a DNA transposon (Protop). To resolve this, we discarded the Su(Ste) entry, and simply retained the Protop and Stellate entries. This seems unsatisfying, but it is the best that can be done: Given our alignment pipeline, there is no

Remember: The index should contain one good representative of each repeat family you are interested in. It doesn't have to perfect though, because it will be fed through RepeatMasker to identify many instances of these repeat families in the reference genome.

The remaining files are _err.txt, which is an error log created by BLAST and should be empty (we hope...). The Blast outputs are stored in alignment_output.tsv, alignments_<#>_<#>.tsv, output.csv.

Once you're satistfied with your curation, run the curated repeat index through FindHomology again to make sure there aren't any communities that were missed the first time. If not, you're ready to mask the genome.

2.2 Preparing the reference genome

Before moving forward, it may be necessary to modify the reference genome a bit. Because ConTExt produces a separate file for each entry in the reference genome and repeat consensus index, things get messy if the reference has a separate entry for each unmapped contig or scaffold. Even well curated assemblies, like D. melanogaster's, have thousands of short, unmapped contigs. These are mostly repetitive, and actually quite useful for identifying individual insertions to which reads can be aligned. But we recommend concatenating all unmapped or alternative contigs to reduce the number of output files.

Once the repeat consensus index is constructed, it is used to mask the reference genome to which the reads will be aligned. RepeatMasker can be somewhat tricky to install. It requires a Linux operating system (though you might be able to manage in a virtual environment, like VirtualBox). It may also be necessary to modify the permissions of the executables to allow them to execute. Once RepeatMasker is installed, you can mask the reference as follows:

RepeatMasker -s -lib <path to consensus sequences> -dir <output directory> -pa 3 <reference genome>

2.3 Obtaining the index of individual repeats

The RepeatMasker output includes a few files, including the masked genome and a *.ori.out file. The *.out file contains the coordinates of each masked repeat, which we will use to finish our repeat index and construct the table needed to collapse alignments onto the repeat consensus sequences. First, however, we need to convert the *.out to a *.gff3 using one of RepeatMasker's utilities:

perl utils/rmOutToGFF3.pl *.out > *.gff

You can tell RepeatMasker to generate a *.gff when it masks the genome, but the format of this differs from that expected by the next few steps, so running rmOutToGFF3 is neccesary.

The next step requires calling AlignmentConverter.py from command prompt. Here, the *.gff, the repeat consensus index, and the unmasked reference genome are used to generate both the final repeat index and the conversion table as follows:

python AlignmentConverter.py -fxn build -cons <path to FASTA of repeat consensus sequences> -ref <path to the unmasked reference genome> -gff <path to a *.gff3 generated from RepeatMasker> -out <path to your output file> -BLAST <path to BLASTn executable>

This command will both create a tab-delimited file that stores the conversion table and a *.fasta containing both all of the individual repeats extracted from the genome and consensus sequences.

2.4 Generating the Bowtie2 indexes

The *.fasta created by AlignmentConverter will be the repeat index to which reads are initially aligned and the masked reference genome will be the assembly to which reads are aligned. These need to be converted to BowTie2 indices. Refer to the Bowtie2 manual for details on how to do this.

3 Preparing the specification file

The ConTExt pipeline involves a number of steps and requires a number of additional files be provided, external programs called, and parameters specified. Requiring the user to enter these when running the script in command-line would make for a confusing and error-prone experience. To avoid this, the script takes as input a text file which contains the paths and values of each user-defined parameter. A sample file named specification.txt is provided in the GitHub repository, which the user must modify to fit their analysis. The file provides instructions what each line should contain. Rather than repeat that here, this section outlines how the various fields in the file impact the analysis.

The parameters in the specification file follow a few conventions. The first character of each line determines how it will be interpreted:

Begins with	Value Type
#	Comment
$	String
@	Comma-separated list (no spaces)
%	Number
!	Number or String
?	Boolean

Comments are ignored by the ConTExt scripts, so you are free to add your own to the specification file. In fact, this is encouraged, as the file serves to store many of the choices made in the analysis. Lines beginning with the other symbols are interpreted as specifying parameters in the following manner:

$$\langle ParameterName \rangle = \langle ParameterValue \rangle$$

Summaries of each expected parameter are provide in the following table. We recommend

1. %MaxReadLen: Reads will be trimmed to this length from their 3'-ends
2. !Phred: The PHRED encoding of the data (33 or 64)
3. ?RenameReads: Rename the reads to the format Sample.Number.End (default: True)
4. %threads: For operations that can be parallelized, how many threads to use

5. $Bowtie2: Path to the Bowtie2-align executable
6. $Trimmomatic: Path to the Trimmomatic executable
7. $Samtools: Path to the Samtools executable
8. $AssemblyIndex: Path to the indexed, masked reference genome without file extension
9. $RepeatIndex: Path to the indexed set of individual repeats and consensus sequences without file extension
10. $ConversionTable: Path to the *.tsv describing how alignments are converted from individual insertions to the corresponding consensus (see 2.3). If blank, skips this step.
11. $Ref: The *.fasta for the unmasked reference
12. $Masked: The *.fasta for the masked reference; this is used mostly when calling TE insertions to ensure reference insertions are not missed.
13. $Cons: The *.fasta containing ONLY the repeat consensus sequences
14. @CV: The chromosomes to be checked for TE insertions when constructing the training set used in parameter choice. We recommend including all well-assembled scaffolds, but excluding small (probably) heterochromatic or unmapped contigs.
15. @Train: The TE families to include in the training set. As the training set is used to set clustering parameters, this can affect clustering performance. We recommend including several active TE families such that the training set is likely to include at least a hundred entries.
16. @Cvg: The chromosomes from which the coverage distribution will be estimated. For large genomes, it may be advisable to choose a subset of chromosomes.
17. %Ins: Not currently used
18. !Cov1: The maximum value to be considered when determining the first eigenvector of the covariance matrix. If set to AUTO this will be chosen based on the mean gap size.
19. !Cov2:The maximum value to be considered when determining the second eigenvector of the covariance matrix. If set to AUTO this will be chosen based on the standard deviation of the gap size distribution.
20. %Cutoff1: The minimum per-read clustering error rate to accepted when choosing the first eigenvector.
21. %Cutoff2: The minimum per-read clustering error rate to accepted when choosing the second eigenvector. Should be lower than Cutoff1.

4 Aligning the data

The first major step of ConTExt is aligning repeat-derived reads to repeat consensus sequences. Because ConTExt is intended to be used with large numbers of samples, the alignment pipeline is automated in ConTExt.py, which is intended to be run from command line—the internal functions are not general enough to be useful without modification. The general

4.1 Running the alignment

The alignment pipeline is called through command-line using the following command:

python ./tools/AlignmentPipeline.py -i <input directory> -o <output directory> -spec <specification file>

where

<...> Any string (can include underscores)
(...) indicates something which is optional
{.../...} indicates allowed values

This script does the following:

1. Identifies all *.fastq files in the input directory corresponding to paired end data
2. Iterates over each paired end sample and...
   (a) Carries out pre-alignment processing with Trimmomatic (adapter removal, quality trimming, removal of very short reads)
   (b) Trims reads from their 3'-ends to increase the mate-pair distance (gap size)
   (c) Aligns reads to both the masked reference genome and a set of repeat sequences
   (d) Collapses repeat alignments onto the corresponding consensus sequence (using a separate script: tools.AlignmentConverter.py)
   (e) Matches paired reads
   (f) Creates a new directory containing all of the output files for the sample

4.2 Setting up the input directory

ConTExt was designed to to work with population genomic data. To simplify working with many datasets, ConTExt isn't run on a single paired-end dataset at a time, but rather on a directory which contains at least one (and possibly hundreds) paired-end datasets. So to begin, place all of the samples you wish to work with in a single directory. It doesn't matter if there are extraneous files in the directory—they will be ignored.

Two important points:

1. For each sample, the forward and reverse reads must be split into separate files, flagged with "_1" for forward reads and "_2" for reverse reads. If a sample does not have two such files, it will be ignored by ConTExt. If you are working with data downloaded from the Short Read Archive (SRA), splitting forward and reverse reads into separate files can be accomplished with SraToolkit's FastqDump by specifying –split-3 at command-line.
2. The function that identifies samples expects input files to be named as follows:

   <sample_name>_{1/2}.{fastq/fq}(.gz)

4.3 The Specification File

As described above, the specification file

4.3.1 Conversion Table

This is the tab-delimited table necessary for collapsing alignments onto the corresponding consensus sequences, as described in 2.3. It is passed to the -conv parameter.

4.3.2 Consensus File

Collapsing alignments on to the consensus sequences requires the original index of repeat consensus sequences. So the path to the index of repeat consensus sequences, as described in 2.1, must be passed to the -cons parameter

4.3.3 Trimming reads to increase the mate-pair distance

Ultimately the aligned data will be used to identify structures formed by repetitive sequence, which can be used as structures. These structures are identified by looking for read pairs spanning junctions, sequence coordinates that are not

adjacent in the reference sequences, but which are adjacent in the sequenced genome (e.g. deletion breakpoints, the boundary between two tandemly arrayed elements). Such junctions are only identified when they fall between the two reads in a read pair, and are missed when they are contained in a sequencing read (such a read will not align). To increase our power to identify junctions, we employ a trick—we trim sequence from the 3' ends of reads, which shortens the reads but increases the mate pair distance.

The maximum read length allowed is specified by the parameter MaxReadLen in the specification file, and any reads longer than this value are trimmed from their 3' ends. For the Global Diversity Lines, where the average read length is 100 nt, we set MaxReadLen to 70.

4.4 Usage Note

All of the external programs called can make use of multiple processors, so if you have access to multiple CPUs, setting %threads >1 will speed up this step.

4.5 Output

The following output files will be generated for each sample <sample>:

1. ./<sample>_log.txt
2. ./<sample>_{1/2}_{as/te}_bowtie_log.txt (4 files in total)
3. ./<sample>_errlog.txt
4. ./<sample>/
    - Repeat1.dat
    - Repeat2.dat
    - ...
    - Chr1.dat
    - Chr2.dat
    - ...

_log.txt is the Trimmomatic output. The four _bowtie_log.txt files contain the Bowtie2 alignment statistics. _errlog.txt is mostly useless and is description how Samtools manipulated the aligned data. In addition, once the entire input directory is processed, a runlog.txt file is created. This contains the contents of sample-specific log files and some additional information. Importantly, at the end of runlog.txt is a list of the samples that were successfully processed, as well as any that samples where ConTExt failed.

You will note that no *.sam or *.bam files are included in the output. These are generated during the pipeline, but are subsequently deleted to save space. The alignments are instead organized in custom tables that simplifies the downstream clustering. These tables are created in a subdirectory named after the sample (./<sample>/). Any read pair where at least one end is anchored in Repeat1 is written as a row in ./<sample>/Repeat1.dat. Any read pair where at least one end is anchored in Chr1 (but no end is anchored in a repeat) is written as a row in ./<sample>/Chr1.dat. Read pairs where both ends are unmapped are discarded.

The columns of these files are as follows:

1. Reference sequence to which Read 1 aligns
2. Strand to which Read 1 aligns
3. Start coordinate of Read 1 alignment
4. End coordinate of Read 1 alignment
5. Sequence of Read 1
6. Quality string of Read 1
7. Reference sequence to which Read 2 aligns
8. Strand to which Read 2 aligns
9. Start coordinate of Read 2 alignment
10. End coordinate of Read 2 alignment
11. Sequence of Read 2
12. Quality string of Read 2
13. CIGAR String (Read 1)
14. CIGAR String (Read 2)
15. MAPQ (Read 1)
16. MAPQ (Read 2)
17. SNP string (Read 1)
18. SNP string (Read 2)

IMPORTANT NOTE: The SNP strings are derived from the MD strings of the aligned data, but are not equivalent. Where an MD string indicates the reference allele at each mismatch, the SNP strings indicate of the nucleotide in the read at each mismatch. Thus, they provide a compressed summary of each read's sequence, eliminating the need to continue storing the read's sequence in subsequent analyses.

4.6 Alternatives

The above alignment pipeline is geared toward finding structures within repeats. If you are simply interested in assigning repeat-derived reads to the corresponding consensus sequences, you can take an alternative approach. First using BowTie2, align your sequencing reads to the set of all individual insertions (*.fasta generated in section 2.3), rather than the set of all consensus sequences. The output of the alignment step needs to be a *.sam rather than a *.bam. This is the *.sam used as input in AlignmentConverter.py.

Then in command prompt use AlignmentConverter to collapse the alignments onto the consensus:

python [path to AlignmentConverter.py] -fxn convert -i [path to *.sam file] -conv [path to conversion table] -cons [path to consensus sequence] -p [number of threads] -replace True/False

The output file generated has the same name as the input, but with "_consensus" appended before the extension. If -replace is set to True, the input is deleted and replaced with the output. Otherwise, both files are retained.

IMPORTANT NOTE: This function currently converts the MD strings to SNP strings. In the near function I will include an option report the MD strings.

5 Identifying structures

In the last step we aligned reads, now we need to think about the patterns in those alignments to identify structures in repetitive sequence. The approach ConTExt takes is to find a generative model that well-describes the alignment patterns in a NGS data set and which relates these to the underlying structures. Each structure in a sequenced genome generates a distinct distribution of discordant read pairs. The distribution of all discordant read pairs in a data set can then be thought of as a mixture of the individual distributions produced by different structures. So, ConTExt approaches this problem using the well-founded toolbox of Mixture Modeling. The actual generative model is difficult to fit, so we employ an easy to solve approximation and model the discordant read pairs as arising from a Gaussian Mixture Model (GMM). We fit the GMM to the data using an accelerated Expectation Maximization algorithm and use the fitted model to identify structures.

5.1 Running ExpMax.py

In command line, call:

OMP_NUM_THREADS=1 nohup python ExpMax.py -i <input di-

rectory> -o <output directory> -spec <specification file>

1. Estimate the gap size distribution (the distance between the 3'-ends of read pairs) from the chromosomes indicated by @Cvg in the specification file.

2. Estimates the read depth distribution conditioned on local GC-composition from the chromosomes indicated by @Cvg in the specification file.

3. Employs distance-based clustering to build a training set of TEs insertion in the unique sequence of the reference genome. The distance cutoff is determined from Monte Carlo simulations of the gap size distribution. The training set is limited to those TE families specified by the @Train field and the reference chromosomes specified by the @CV field.

4. Identifies a good covariance matrix for the mixture model by evaluating the performance of the EM algorithm on the training set. Chooses the smallest eigenvalue along eigenvector 1 such that the clustering performance is at least %Cutoff1, and then chooses the eigenvalue along eigenvector 2 such that the clustering performance is at least %Cutoff2. Note that %Cutoff2 must be less than %Cutoff1. See the Supplemental Methods of the ConTExt paper for details

5. Evaluates the performance of the model on data simulated from the gap size distribution

6. Clusters the data by fitting a Gaussian Mixture Model to all repeat-anchored discordant read pairs in the data

7. Creates a new directory containing all of the output files for the sample

5.1.1 Additional Arguments

If you wish to process only a subset of samples in the input directory, provide the optional arguments -head <char> and -find <string>. -head tells ExpMax to only process those samples that begin with the specified character. -find tells ExpMax to only process those samples that contain the specified string. Neither of these parameters are case-sensitive. As ExpMax does not run in parallel, these functions are particularly useful for processing multiple subsets of the data at the same time, cutting down the amount of time it takes to cluster a population dataset.

We recommend explicitly limiting the process to a single CPU, such as with OMP_NUM_THREADS=1 in a Linux environment. While the vast majority of the script runs on a single-processor, some of the numerical operations will use as much processing power as they can, with little gain in speed. Instead, it simply leads to sporadic spikes in CPU usage.

5.2 Output

ExpMax.py generates a number of output files.

1. <sample>_out.tsv: This the output file containing the clustered data.
2. <sample>_len.tsv: The read length distribution of the processed data.
3. <sample>_kde.tsv: The mate pair distance (gap) distribution. There are four rows. The first is the observed frequency art which a given gap is observed, the second is the kernel density estimate of the underlying distribution, and the third is the expected distribution conditioned on the read pair spanning a junction. The fourth row is the corresponding gap size; negative values indicate overlapping reads.
4. <sample>_cvg_exp_{chromosome}.tsv: The average depth on a given chromosome as a function %GC.
5. <sample>_cvg_hist.npy:A Numpy array that stores the joint distribution of read depth and %GC for each chromosome. This is used downstream to estimate the copy numbers of structures.
6. <sample>_cvg_GCcounts.npy: The frequency of %GC across the reference genome.
7. <sample>_opt.tsv: Describes the covariance optimization step. The first four rows describe the optimization of the first eigenvalue. The first row is the clustering performance, the second row contains the corresponding scale parameter, the third row is the expected performance at that scale, and the fourth row is the chosen scale parameter. The next four rows describe the same thing, but for the second eigenvalue.
8. <sample>_opt_{x/y}.jpeg: A visualization of clustering performance (dots) and expected performance (line) during the the optimization steps.
9. <sample>_eigenvector{1/2}_precision.jpeg: If two junctions are separated by some distance along the specified eigenvector, this depicts the precision with reads from different junctions are assigned to different clusters, the recall with which reads from the same junction are assigned to the same cluster, and the Falkes-Mallow score combining the two summaries, all as the distance between clusters varies. Computed from simulated data.
10. <sample>_recall.jpeg: Summarizes how recall varies as a function of the number of reads spanning a junction. Computed from simulated data.
11. sample_summary_.tsv: This table summarizes the parameters and performance of all samples analyzed. If ExpMax is run with the -head argument the specified value will be appended to the end of the file name (eg sample_summary_B.tsv). As described in the ConText manuscript: "Summaries of expected clustering performance and library parameters

relevant to clustering. For each library, columns present from left to right: the mean, the standard deviation, the 0.5% and 99.5% quantiles of the gap size distribution; the distance cutoff used to identify the euchromatic TE insertions from which clustering parameters are learned; the covariance parameters of the GMM used to cluster the reads; the parameters of the generalized logistic regression that summarize how precision decays along the first and second eigenvectors of the covariance matrix; and the average recall for read counts between 2 and 19."

5.3 Clustering Output

The clustering output is a tab-delimited file with the following structure:

Column 1: Anchor sequence

Column 2: Target sequence

Column 3: Anchor strand

Column 4: Target strand

Column 5: Feature type (Junction or Consensus)

Column 6: Anchor coordinate

Column 7: Target coordinate

Column 8: Expected error along first eigenvector

Column 9: Expected error along second eigenvector

Column 10: Read count

Column 11: Comma-separated list of read 3'-coordinates on Anchor

Column 12: Comma-separated list of read 3'-coordinates on Target

Column 13: Comma-separated list of read 5'-coordinates on Anchor

Column 14: Comma-separated list of read 5'-coordinates on Target

Column 15: Comma-separated list of Mapping Quality to the Anchor

Column 16: Comma-separated list of Mapping Quality to the Target

Column 15: Comma-separated list of Mapping Quality to the Anchor

Column 16: Comma-separated list of Mapping Quality to the Target

Column 17: The cluster number for this sample

Column 18: Comma-separated list of CIGAR strings for Anchor mapped reads

Column 19: Comma-separated list of CIGAR strings for Target mapped reads

Column 20: Comma-separated list of SNP strings for Anchor mapped reads

Column 21: Comma-separated list of SNP strings for Target mapped reads

Column 22: Comma-separated list of QD strings for Anchor mapped reads

Column 23: Comma-separated list of QD strings for Target mapped reads

Note: QD strings indicate which positions in each read have a PHRED quality scores less than 30.

5.4 Evaluating the output

5.5 Details of parameter choice

There are several parameters that determine the behavior of the clustering algorithm: Cutoff1, Cutoff2, Cov1, and Cov2.

Cutoff1 and Cutoff2 define the minimum clustering performance accepted. When choosing the covariance matrix for the GMM, the smallest covariance matrix such that clustering performance exceeds Cutoff2 is chosen. This is done iteratively, so Cutoff1 is used to choose the first eigenvector of the covariance matrix, and Cutoff2 is used to choose the second eigenvector of the covariance matrix.

Clustering performance here is defined relative to the training set as the average proportion of reads assigned to the largest cluster in each training run. Each training cluster should represent a single true cluster. If a training cluster is comprised of ten reads, and the EM algorithm identifies assigns all to a single cluster then it has recovered the original cluster. In this case, its performance is 1. On the other hand, if it recovers two clusters sized 7 and 3, then the performance is $\frac{7}{3+7} = .7$.

If we think about the correctness of clustering in terms of the following statement "These two reads reflect the same structure", we can define False Negatives and False Positives. A False Negative occurs when two reads are incorrectly assigned to different clusters. False Positives occur when reads from two distinct junctions are clustered together. Thus, the clustering performance used to choose the covariance matrix explicitly aims to minimize False Negatives. This is because False Negatives are a risk for every junction in the genome, but False Positives can only occur when two junctions are near each other. False Positives are minimized implicitly by choosing the smallest covariance where performance exceeds the cutoff. Intuitively, the larger the covariance matrix is, the fewer the number of False Negatives. Conversely, the smaller the covariance matrix, the greater the number of False Positives. While we care about both types of errors, so we explicitly try to limit this to a particular threshold, and then control for False Positives

Cov1 and Cov2 define the largest covariance matrix tested when trying to determine the optimal clustering behavior. These can be set manually or automatically. The automatic choice of these parameters sets Cov1 equal to 150% of the mean gapsize and Cov2 equal to twice the standard deviation of the gap size distribution.

6 Summarizing clustered output

Deprecation Warning: The script discussed in this section will be merged into ExpMax in the near future, but for now it is necessary.

Once the data is clustered, each junction needs to be annotated with a classification (e.g. Tandem versus Probable Artifact) as well as expected GC-content of a read spanning the junction. To do this, run:

python SummarizeOutput.py -i <clustered directory> -spec <specification file>

This will create new files entitled <sample>__outCN.tsv.

6.1 Estimating Repeat Copy Number from Consensus Coverage

Run:

python SummarizeOutput.py -i <clustered directory> -spec <specification file> -count True

This generates a file entitled count_table.tsv

7 Matching junctions between samples

7.1 Automated clustering

Because we estimate junction location using the mean position of read pairs spanning the junction, the error of these junction estimates may be approximated with a bivariate normal distribution (c.f. the Central Limit Theorem). The error of our estimates depends both upon the read count of the structure in that sample and on the gap distribution of the sequencing library. That the error can be readily described means it provides information useful in matching structures across samples and we tackle this with an iterative approximation procedure.

The details of this algorithm can be found in the Supplementary Methods of the ConTExt paper.

7.1.1 Running MatchJunctions.py

After running SummarizeOutput.py, run:

python MatchJunctions.py -i <input directory> -o <output directory> -spec <specification file>

This script does the following:

1. Creates a subdirectory in output directory called ./metatables and fills with tab-delimited files containing every junction from every sample for a given given repeat. See below for the metatable file format. The coverage distributions conditioned on GC-content are also copied to a subdirectory ./metatables/cvg
2. Within each sample, calls TE insertions in unique sequence by matching junctions across opposite quadrants. Outputs a file entitled TE_insertions.tsv
3. Uses the left and right junctions of non-reference LTR retrotransposon insertions to estimate the expected uncertainty of junction estimates in a given sample
4. Clusters junctions across samples using a C-means like algorithm. Outputs files two files each repeat family–<rpt>.tsv and <rpt>_full.tsv–in a subdirectory entitled ./cluster_tables

7.2 Output files

7.2.1 Metatable format

Column 1: Sample Name

Column 2: Junction ID number

Column 3: Anchor Sequence: Always matches the file name

Column 4: Target Sequence

Column 5: Anchor Strand

Column 6: Target Strand

Column 7: Structure type

Column 8: Read count

Column 9: Anchor coordinate

Column 10: Target coordinate

Column 11: Standard deviation along the first eigenvector

Column 12: Standard deviation along the second eigenvector

Column 13: Read count

Column 14: Expected %GC of read pair spanning the junction

Column 15: Estimated Copy Number

Column 16: 98% Credible interval lower bound

Column 17: 98% Credible interval upper bound

7.2.2 Insertion table format

After aggregating junctions from every sample into a single file, junctions within samples are matched to identify TE insertions into the reference genome. These are principally used to obtain an empirical estimate of the uncertainty around each junction estimate. The idea is that each TE insertion has two junctions with the reference, which we call Left and Right. For many TE families the estimated coordinates of the locations of the these two junctions canbe reformulated into two independent estimates of a single sequence coordidate. The amount by which these estimates disagree is informative of the amount of uncertainty. See the supplemental methods of the paper for details.

This table is formatted as follows:

Column 1: Sample Name

Column 2: Repeat Family

Column 3: Reference Chromosome

Column 4: Insertion orientation relative to the reference (+ or -)

Column 5: Junction ID numbers

Column 6: Estimate reference location

Column 7: Reference coordinate of the left junction

Column 8: Repeat coordinate of the left junction

Column 9: Reference coordinate of the right junction

Column 10: Repeat coordinate of the right junction

Column 11: Read count for the left junction

Column 12: Read count for the right junction

7.2.3 Cluster table format

Once the junctions are matched across samples, a single output table is generated for each repeat family containing every distinct junction identified in the data set and the copy number of that junction in each sample.

Column 1: Anchor Sequence

Column 2: Target Sequence

Column 3: Anchor Strand

Column 4: Target Strand

Column 5: Estimated anchor coordinate

Column 6: Estimated target coordinate

Column 7: Population Frequency

Column 8: Average Copy Number

Columns 9-11: Placeholders

Columns 12-: Copy number of the junction in corresponding sample

In the output files ending in _full.tsv, the cells in Columns 12- contain four values separated by semicolons:

Copy Number; Anchor Coordinate; Target Coordinate; Junction ID

If multiple clusters in that sample are believed to reflect the same junction, the values are themselves comma separated lists describing each cluster.