Structural Variant Calling in Non-Human Populations
AG2PI Workshop December 2, 2021
Layer Lab, University of Colorado, Boulder

Ryan Layer ryan.layer@colorado.edu
Murad Chowdhury murad.chowdhury@colorado.edu
Devin Burke devin.burke@colorado.edu

Table of Contents

```
### download these first ###
$ iinit # this will prompt you for your username & password
$ iget -PT /iplant/home/shared/ag2pi_workshop/2021-Dec/data/XH82_ssa01.bam .
$ iget -PT /iplant/home/shared/ag2pi_workshop/2021-Dec/data/XH96_ssa01.bam .
$ iget -PT
/iplant/home/shared/ag2pi_workshop/2021-Dec/data/GCF_000233375.1_ICSASG_v2_genomic.fa.gz .

$ iget -PT
/iplant/home/shared/ag2pi_workshop/2021-Dec/data/GCF_000233375.1_ICSASG_v2_genomic.fa.gz.fai .

### download this second - will take a few minutes ###
$ iget -PTr /iplant/home/shared/ag2pi_workshop/2021-Dec/data/ .
```

BioFrontiers Institute

UNIVERSITY OF COLORADO

Introduction

There has been a drastic increase in the interest surrounding the role of structural variants in population wide genomics due to the amassing evidence of their regulatory and structural impacts. Despite this there remains a significant challenge in reliable structural variant detection. Here we lay out a step-by-step tutorial on how to quickly extract high quality structural variants for non-human populations from short-read sequencing data. The process of which will include aligning samples to a reference genome, identifying problematic regions in both the samples and reference, and applying these regions as a mask to a Lumpy-based tool for structural variant calling and genotyping.

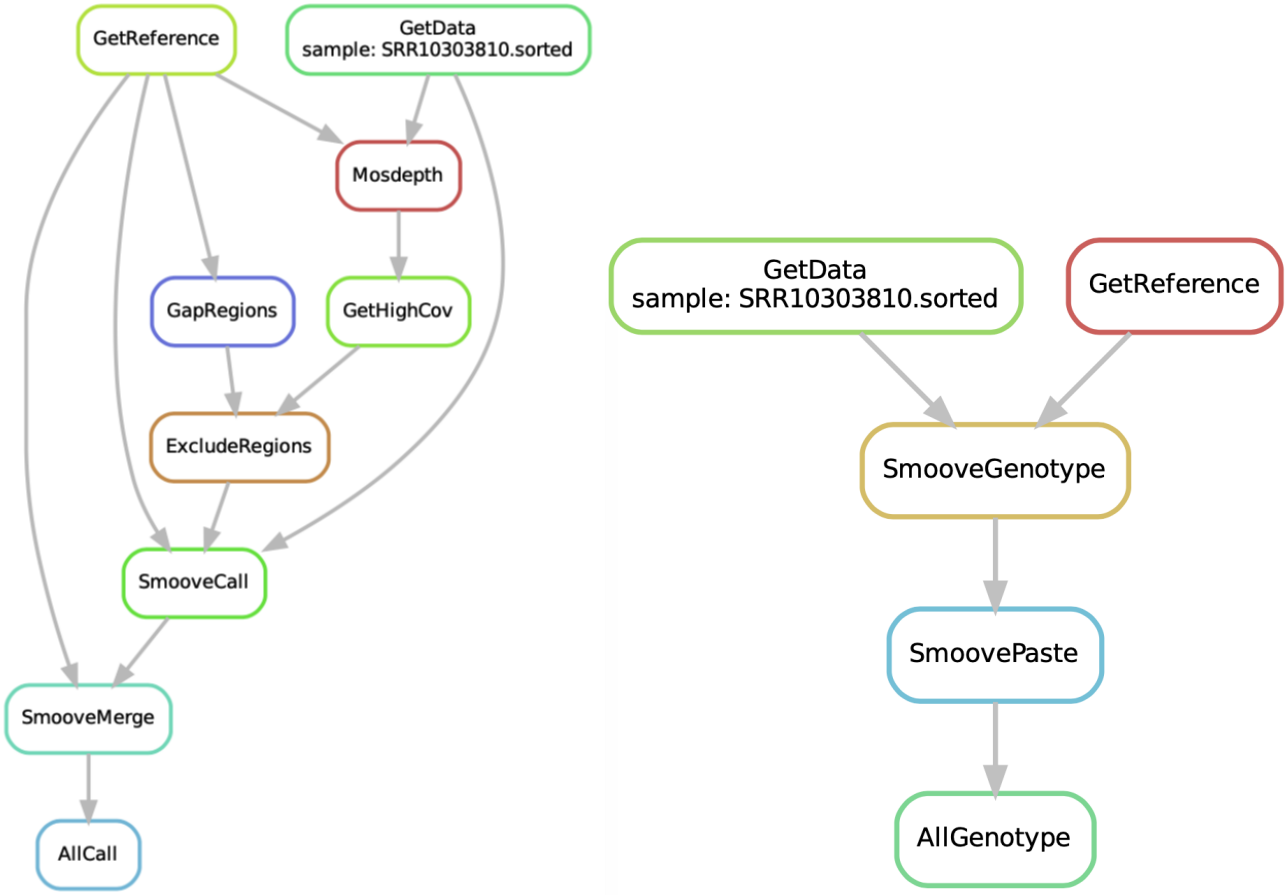

Figure 1: Overview of the sequence of steps involved in structural variant calling and genotyping. (Left) This process begins with the collection of reference genome and set of samples, which are aligned to the reference as well as sorted and indexed properly. Following this the problematic regions are identified and used to construct a sample-specific mask that is applied to the SV caller. Each of the sample-specific SV results are merged and used as input to the final step (right). The merged SV results are used to get population aware genotyping completing the SV calling process.

Align each sample to the reference genome

The process of calling structural variants (SVs) begins with the alignment of *paired end* short read sequencing data to a reference genome. We want to use paired end sequencing data because of its intrinsic ability to project distinctive patterns when mapped to the reference genome. This data can be found using NCBI's Sequence Read Archive (SRA) portal (https://www.ncbi.nlm.nih.gov/sra) and downloaded using NCBI's SRA Toolkit or Cloud Data Delivery Service for larger population sizes. Once the desired fastq sample files and respective reference file are accessible, we can align each of the samples. To do this we need to download BWA (https://github.com/lh3/bwa). Once downloaded we can uncompress our reference genome and index it.

```
$ gzip -d reference.fa.gz

$ bwa index -p ref reference.fa
```

This will produce a number of different files each with the prefix *ref*. Now that our reference genome is indexed properly, we can proceed with the alignment of our samples. The green text in the command line below is known as a read group. Read groups contain information to help later steps trace results back to specific samples. The most simple read group is annotated in green below and is simply the name or identifier of the sample, other information can be added at the discretion of the user.

```
$ bwa mem
    -M -t $threads # BWA scales proportionally to thread count
    -R "@RG\tID:1\tSM:""$sample_name" \
    ref \
    sample_1.fastq sample_2.fastq  \ # two files for paired-end data
    2> bwa_sample.err \ # standard error log
    > sample.bam        # output
```

This step will produce a BAM file for each of our desired samples. This step can take quite some time depending on the number of threads given and the size of the input files (5+ hours with 4 threads). We now just have one more step to prepare the BAM files. Each of the BAM files must be sorted and indexed for easy and quick navigation in the following steps. In order to do this we will want to install Samtools (http://www.htslib.org/download/).

```
$ samtools sort sample.bam -o sample.sorted.bam -@$threads
$ samtools index sample.sorted.bam -@$threads
```

These steps will produce a coordinate-sorted BAM file and a BAM index file (*.bam.bai*) which ultimately allows for fast random access.

Summary
In the aforementioned steps, we successfully indexed a reference file and aligned each of our sample fastqs to produce BAM files for each of our samples. We then sorted and indexed each of our BAM files.

Tools:	BWA | Samtools	
Input:	reference.fa | sample_1.fastq, sample_2.fastq	(for each sample in population)
Output:	sample.sorted.bam, sample.sorted.bam.bai	(for each sample in population)

Checkpoint 1:

With the initial preparations of our data complete, we can proceed to performing some intermediate extractions on our samples which will be required as input into our final SV calling and genotyping step. Before proceeding ensure the correctness and completeness of the above steps.

Prep: Mask problematic regions using sample-specific genome exclude files

An artifact of the sequencing data that has the potential to produce low confidence or misleading SV calls are areas of abnormally large coverage. To reduce this risk we will find regions within our samples with depth much larger than the norm across the entire genome.

In addition to high depth regions, we want to find gap regions within our reference genome, which allows us to discard SV calls overlapping into areas void of mappings. Combined we create a sample-specific mask of problematic regions to boost the quality and confidence of our SV calls and genotypes.

Calculating BAM depth

We begin by calculating the depth of each of our BAM files. We do this with the Mosdepth toolset (https://github.com/brentp/mosdepth), which allows for fast BAM/CRAM depth calculations for whole genome sequencing. Once we have Mosdepth installed we can proceed.

```
$ mosdepth \
      --by 100 \ #window size of 100
      --fast-mode \
      --fasta reference.fa \
      --no-per-base  \
      sample.sorted.bam
```

Once finished executing, this step will produce a number of files specific to the input sample. Keep in mind we want to run this Mosdepth on each of our samples such that we can create one mask per sample. These files are described below in Table 1.

Table 1: Mosdepth output files

File Name	Contents
sample.mosdepth.global.dist.txt	Distribution of the proportion of bases above a genomic-wide threshold
sample.mosdepth.summary.txt	Summary statistics
sample.mosdepth.region.dist.txt	Distribution of the proportion of bases above a region-specific threshold
sample.regions.bed.gz	Mean per-window depth

Extracting high depth regions

Now that we have the depth calculations for each of our samples, we can extract the high depth regions. To do this we want to find regions with coverage greater than twice the standard deviation above the mean coverage of the sample. This is the baseline for what we will classify as *abnormal* coverage and will thus warrant exclusion in the SV calling step. We will accomplish this using the *regions.bed.gz* file produced by Mosdepth as well as Pandas (https://pandas.pydata.org/docs/getting_started/install.html) and Numpy (https://numpy.org/install/).

```
$ python
>>> import pandas as pd
>>> import numpy as np
>>> mosdepth_bed = pd.read_csv(          #read Mosdepth regions into df
              'sample.regions.bed.gz', #input file
              compression = 'gzip',    #compression type
              sep = '\t',              #file delimiter
              names = ['chrom','start', 'end', 'depth']) # column names

>>> mean_depth = mosdepth_bed.depth.mean() #mean depth
>>> std_depth  = mosdepth_bed.depth.std()  #std depth
>>> high_cov_bed = mosdepth_bed.loc[       #get regions >2 std above mean
        mosdepth_bed.depth > (mean_depth + 2*std_depth)]

>>> high_cov_bed.to_csv(                   #df into bed file
              'sample.high_cov.bed',      #output file
              sep = '\t',                 #file delimiter
```

```
                header = False,
                index = False)
```

The output of these commands will yield a sample specific BED file annotated with the regions that surpassed our depth threshold.

Extracting gap regions

With any reference genome assembly, we are presented with areas known as gaps where highly complex or repetitive regions present extremely challenging mapping problems and are often simply annotated with 'N'. These gaps present additionally problematic regions and need to be accounted for in our mask. We present an easy to use and parallelized python script to extract these regions from our reference genome. (https://github.com/mchowdh200/animal_svs/blob/multisample/src/scripts/gap_regions.py).

```
$ python gap_regions.py reference.fa $threads > gap_regions.bed
```

This step will produce one BED file annotated with the identified gap regions in our reference genome.

Summary

In the aforementioned steps, we have collected the necessary prerequisites to create a mask of the problematic regions in each of the samples of our population, mainly the abnormally high coverage regions in our input samples and the gap regions in our reference genome.

Tools: Mosdepth | Python, Numpy, Pandas | gap_regions.py

Input:

BAM coverage :	reference.fa | sample.sorted.bam	(for each sample in population)
High coverage regions :	sample.regions.bed.gz	(for each sample in population)
Gap regions :	reference.fa	

Output:

BAM coverage :	see Table 1	
High coverage regions :	sample.high_cov.bed	(for each sample in population)
Gap regions :	gap_regions.bed	

Checkpoint 2:

At this point we have prepared the necessary prerequisites to create a mask to exclude the described problematic regions from our SV calling, helping to ensure that our produced calls and genotypes are of high confidence and quality. In the next step we will combine these prerequisites to create the sample-specific masks. Before continuing make sure to review the previous steps for correctness and completeness.

Creating sample-specific genome exclude file with established prerequisites

Now that we have successfully established the necessary components of the mask we can go ahead and execute the steps to officially create the mask which will be a BED file annotated with the regions to be excluded per specific sample. We will use bedtools to sort and merge proximal regions of some parameterized distance (https://bedtools.readthedocs.io/en/latest/content/installation.html).

```
$ cat gap_regions.bed <(cut -f1-3 sample.high_cov.bed) | #concat gap regions
      bedtools sort -i stdin |                           #sort
      bedtools merge -d 10 -i stdin > sample.exclude.bed #Merge near regions
```

From executing this we will produce a BED file (sample.exclude.bed) for each sample annotating the regions to be excluded.

Summary
In the previous step we successfully applied our gap regions and sample specific high depth regions to create a BED file for each of our samples annotated with problematic regions specific to the sample.

Tools: Bedtools
Input: gap_regions.bed | sample.high_cov.bed (for each sample in population)
Output: sample.exclude.bed (for each sample in population)

Checkpoint 3:

At this point we now have successfully aligned each of our samples to the reference genome and created a set of sample specific masks of problematic regions we want to exclude. We are now finally ready to start SV calling on our population. Again, review all of the above steps for completeness before proceeding.

Sample specific structural variant calling

We are now ready to start SV calling on each of our samples. Here we will first get sample specific SV calls and then find the union of all produced sites across all the samples and merge them into a single VCF file. For this we will use a Lumpy based tool called Smoove (https://github.com/brentp/smoove). Smoove allows for simplified and parallelized population-level calling. We will start this process by running the smoove call feature on each of our samples.

```
$   smoove call --genotype \
                --duphold \
                --processes 1 \
                --fasta ref.fa \
                --exclude sample.exclude.bed
                --name sample \
                --outdir outdir/sample/ \
                sample.bam
```

This will yield genotyped SV calls for each of our samples extracted to a sample.smoove.genotyped.vcf.gz file. Once we have SV calls for each of the samples in the population we can proceed with merging them.

Merging individual SV calls to produce population-specific SV landscape

We want to initially do SV calling on sample specific data as it allows us to gather the union of all returned sites across all samples via a parallelized method. Again we will use smoove, but now execute smoove merge on the set of all *.smoove.genotyped.vcf.gz files produced from the previous step. Notice the input to this step is the list of all sample VCF files across the population.

```
$  smoove merge --name merged \
                --fasta ref.fa \
                --outdir outdir/ \
                [sample.genotyped.vcf.gz]
                ### list of all vcfs from smoove call ###
```

This will produce a singular file, merged.sites.vcf.gz, containing the union of all SV sites across all samples in the given population. These merged sites will also include sample-specific genotypes, but we would like to go one step further and extract population-wide and population-aware genotypes.

Summary

In the last two steps we have successfully extracted sample-specific SV calling using our problematic region mask on each of our samples and then merged across all the sites to produce a unioned file with sample specific genotypes.

Tools: Smoove

Input:

Call :	reference.fa | sample.sorted.bam	(for each sample in population)
Merge :	sample.smoove.genotyped.vcf.gz	(for each sample in population)

Output:

Call :	sample.smoove.genotyped.vcf.gz	(for each sample in population)
Merge :	merged.sites.vcf.gz	

Checkpoint 4:

At this point we have a file containing all of the SV calls merged across all samples. This file contains sample-specific genotyping information and now we want to finalize by getting more robust genotyping via smoove's genotype and paste features.

Sample specific structural variant genotyping

From the previous steps we now have a file containing merged sites across the population. We can now genotype each sample at each one of those sites to extract population-aware genotyping calls.

```
$  smoove genotype --processes $threads \
                   --duphold \
                   --fasta ref.fa \
                   --name sample \
                   --outdir outdir/sample/ \
                   --vcf merged.sites.vcf.gz \
                   sample.bam
```

After execution we will now have genotyping calls for every SV site for every sample in our population. Additionally, we now have depth annotations to get better insight into the confidence of these genotypes in perspective of the population.

Creating population-level genotyped SV cohort

We are finally ready to produce a population-level SV landscape. In the previous step we produced one file for each sample containing genotyping SV calls in perspective of the population's merged sites. So now all that we have to do is merge all the single sample genotyped VCF files that contain the same number of variants to produce a singular and squared population-level VCF file.

```
$    smoove paste --name sites \
                  --outdir outdir/ \
                  [sample.smoove.genotyped.vcf.gz]
                  ### list of all vcfs from smoove genotype ###
```

Having executed this, we have successfully produced a VCF file containing the SV landscape across our population.

Summary

Tools: Smoove

Input:

Genotype :	reference.fa | merged.sites.vcf.gz | sample.sorted.bam	(for each sample in population)
Paste :	list(sample.smoove.genotyped.vcf.gz)	(for all samples in population)

Output:

Genotype :	sample.smoove.genotyped.vcf.gz	(for each sample in population)
Paste :	sites.smoove.square.vcf.gz	

Analysis

Extracting SVs of particular length and performing phylogenetic analysis

Now that we have successfully extracted the structural variants for our population we can discuss a few avenues for analytic exploration. Over the next few steps we will look at the distribution of SVs across our samples, extract sequences of interest, execute a multiple alignment, and then explore a phylogenetic analysis. Additionally, we will be using the full dataset of 492 Atlantic Salmon Genomes.

In our VCF file we have multiple different types of structural variants ranging from deletions, translocations, and duplications. Here we want to simply extract only the deletions as they are the simplest to map back to the reference genome for sequence extraction. Compared to single-nucleotide variants (SNVs) the challenge of inferring the true sequence of SVs is exponentially more difficult due to the size and their overlapping nature. To extract just the deletions we will use BCFtools, a subset of the Samtools package that is specifically designed for use on VCF files (https://samtools.github.io/bcftools/bcftools.html). Here we will extract using the SVTYPE column where an entry's SVTYPE equals a deletion.

```
$  bcftools view -i 'SVTYPE="DEL"' salmon.vcf.gz -o deletions_salmon.vcf
```

This command will output a separate VCF file containing only the deletions. Just extracting all of the variants is not incredibly useful by itself. As depicted in the histogram below, the diversity in SV deletion sizes in this dataset forces us to further narrow down the subset of SVs.

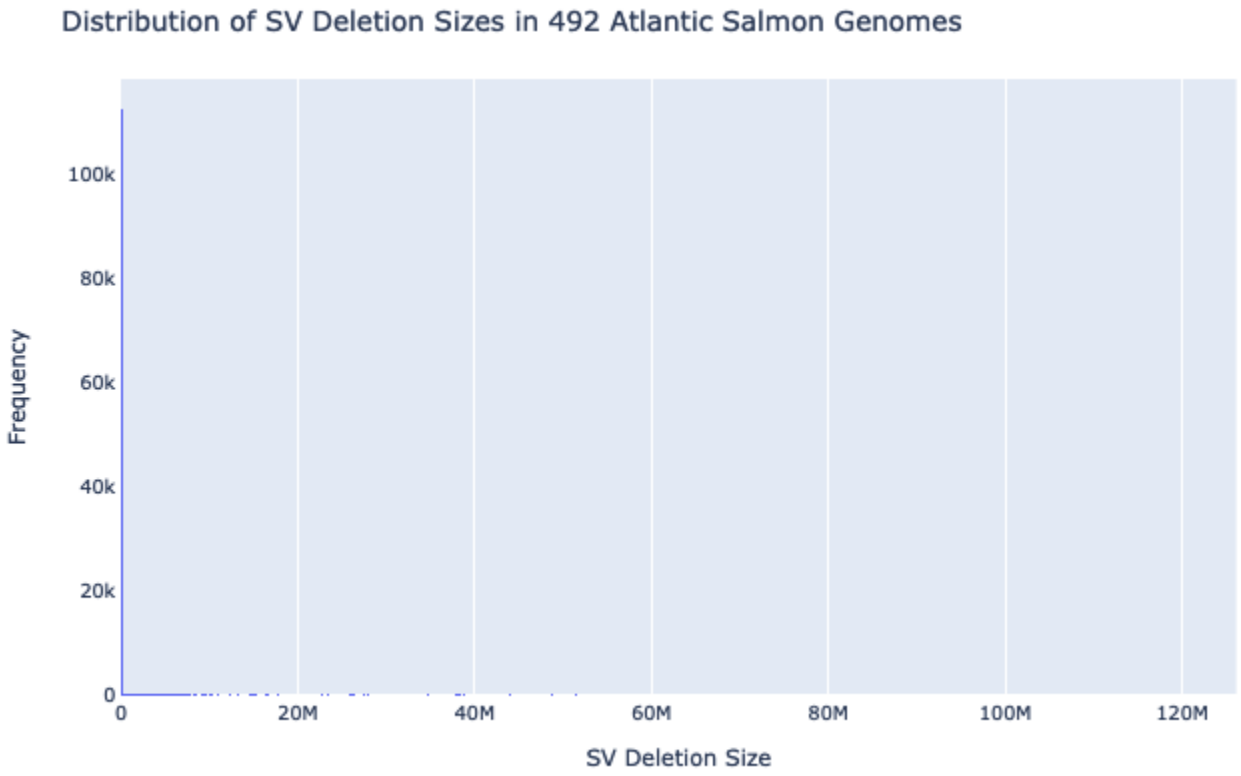

Figure 2: Complete distribution of SV deletion sizes in the full dataset containing 492 Atlantic Salmon genomes. The extremely large deletion events are not accurate enough to be included in downstream analysis and must be filtered.

We want to narrow filter out the extremely large SVs because they do not offer useful information in analysis, mainly due to the inaccuracies they would yield. So we will try filtering out SVs less than 3500 bps in length. Keep in mind that deletions will be negative in value. Again we will use BCFtools to accomplish this and instead of using the SVTYPE column we will use SVLEN.

```
$  bcftools view -i 'SVLEN>-3500' deletions_salmon.vcf.gz
        > small_deletions_salmon.vcf
```

Now we have another VCF file that contains deletions of length 3500 bps or less. We can again plot the contents of these SVs. The code below uses scikit's allel package (https://scikit-allel.readthedocs.io/en/latest/) and a plotting library called plotly (https://plotly.com/graphing-libraries/). Allel is not as robust as other VCF libraries but it allows for quick integration into Pandas for quick and easy data exploration. Here is a useful blog on extracting data from VCF files (http://alimanfoo.github.io/2017/06/14/read-vcf.html). Similarly, ploty does not compare in feature set to other libraries, but it's interactive capabilities allow for easy exploration, which avoids recalculating and replotting to shift the frame of the plot.

```
$ python
    >>> import allel
    >>> import plotly.express as px

    >>> callset = allel.read_vcf('small_deletions_salmon.vcf', \
                                 fields=['INFO'])

    >>> sv_len = list(callset['variants/SVLEN'])
    >>> fig = px.histogram(np.abs(sv_len))
    >>> fig.show()
```

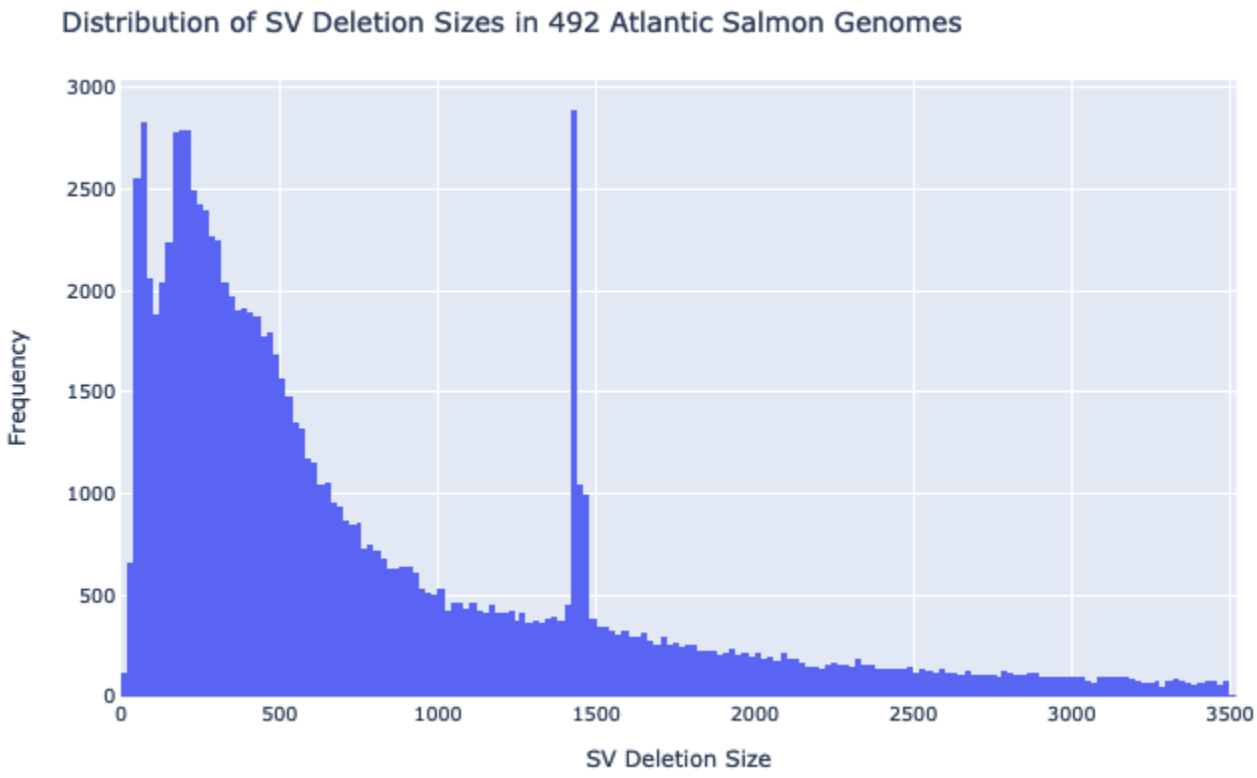

Figure 3: Filtered distribution of SV deletion sizes in the complete dataset. Here we can see a highly frequent SV of about 1500 bps in length.

From the above figure we clearly have found something interesting, a very frequent grouping of SV deletions of about 1500 bps long. The frequency of a structural variant will become more and more unique as its size increases so to see such a high occurrence of the same length deletion in such a large population is certainly anomalistic. Now that we found something specific we can proceed to further explore this isolated area of the dataset. Specifically, we want to extract the sequences within this group of deletions and perform a phylogenetic analysis in order to gain a deeper understanding of their similarities and differences.

To prepare the data for sequence extraction we want to first create an additional VCF file with only the variants of interest defined in Figure 3. This is an optional step but it will greatly increase the computation time if the VCF file only contains the SVs of interest. In this dataset we will reduce the number of records from 116969 to 5448 and reduce the size of the search space by 16 fold. In the following commands, we will extract SVs that are between 1400 and 1500 bps into a new VCF file and then index it.

```
$  bcftools view -i 'SVLEN<-1432 && SVLEN>-1446'
       deletions_salmon.vcf.gz > final_deletions_salmon.vcf
$  bgzip final_deletions_salmon.vcf

$  tabix -p vcf final_deletions_salmon.vcf.gz
```

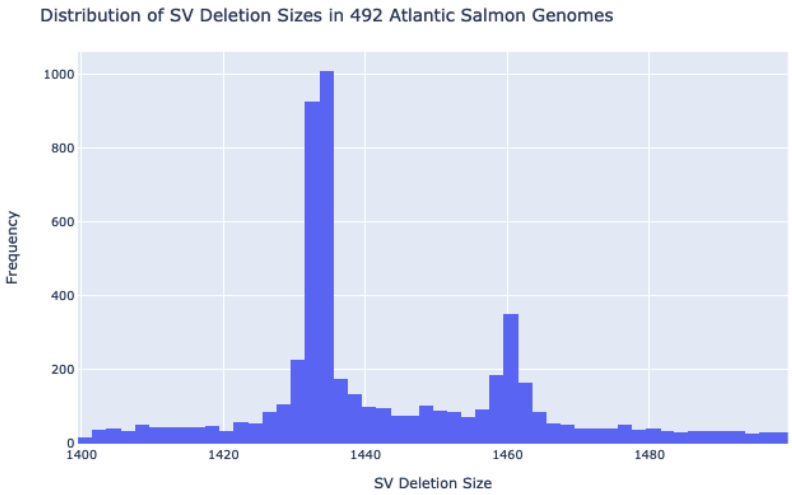

Figure 4: Distribution of SV deletion sizes restricted to the 1400-1500bp length.

The outcome of these commands will be a new VCF file containing only our interesting SVs as well as an index file associated with it. Figure 4 shows the resulting distribution of this new VCF file. We will also need our reference genome and its associated index from earlier in this tutorial. The below commands will regenerate the necessary components if required.

```
$  gzip -d salmon.fa.gz
$  bwa index -p salmon salmon.fa
```

The last item we need is a list of the genomic coordinates for each of these variants in this reduced VCF file. These coordinates will allow us to map back to the reference genome and extract the sequences that were found to be deleted in the sample. To accomplish this we will use bcftool's query feature to get the chromosome, start, and end coordinate for each variant and write them to a new file.

```
$  bcftools query -f '%CHROM\t%POS\t%INFO/END\n'
       final_deletions_salmon.vcf.gz > salmon_extract.txt
```

Once we have this file containing the location of each variant we can now extract the sequences from the reference genome. To do this we will simply use samtool's faidx which takes in genomic coordinates and returns the sequence in the reference genome within the coordinates. We will write those sequences along with the respective coordinates to a file in the fasta format. This step can also be performed using a BED file instead.

```
$ while read X Y Z; do chroms[++i]=$X; starts[i]=$Y; ends[i]=$Z; done
        <salmon_extract.txt

$ for i in "${!chroms[@]}"; do

   samtools faidx salmon.fa
        "${chroms[i]}":"${starts[i]}"-"${ends[i]}"
        >> salmon_sequences.txt

   echo "" >> salmon_sequences.txt

  done
```

Now we have successfully extracted the sequences for all of the deletion events of interest and we can proceed with some phylogenetic analysis on them. For these next steps we will use a program called SplitsTree which will perform many calculations for us and simplify the entire process. SplitsTree will take in the fasta file of our sequences, perform a multiple-sequence alignment, calculate genomic distance, perform splits, and then create networks, trees or even perform PCA. Additionally, it allows us to vary between different distance metrics or even evaluate character and quartet based divergences.

In figure 5, we constructed a rooted equal angle neighbor joining network based on the uncorrected_p genomic distance metric. Here each black dot is one of the sequences we extracted. It is apparent from the distance scale that even the largest difference across the network is quite close in similarity globally. Furthermore, the significant clustering of nodes shows a high amount of similarity in the sequences. In phylogenetic recombinant networks edges between nodes model some form of recombination or horizontal transfer in attempts to infer some evolutionary relationship between taxa. Therefore it is apparent that this shared deletion is quite possibly a divergent evolutionary artifact for the species. We could then take these sequences and perform a BLAST search across similar species to show evolutionary distinctiveness in perspective of only these deletions.

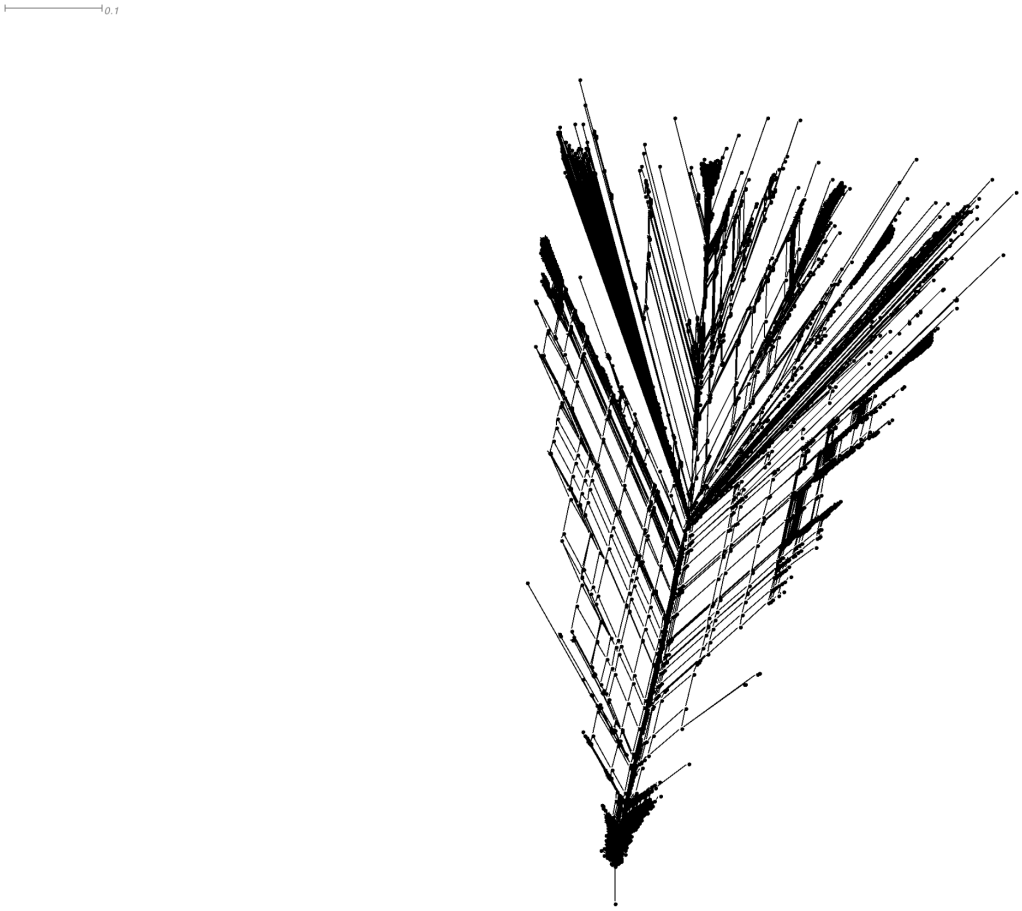

Figure 5: Rooted Equal Angle NeighborJoining Network of the extracted salmon sequences. The high degree of similarity among all the sequences is indicative of some evolutionary relationship among the population.

Extracting SVs for particular sample

We can also extract sequences based on the individual's identified SVs instead of restrictive to a variant size. This type of filtering is useful if we'd like to look at the phylogenetic differences of a specific gene or chromosome that each sample's unique SV's result in. For this tutorial we will look at 100 human samples from the 1000 genomes project as it offers a concise regional representation for exploring the phylogenetic relationships of the immunoglobulin (IGH) gene. IGH is responsible for production of antibodies and undergoes somatic recombination, which explains the diverse antibody landscape across human populations. This should yield a much more diverse phylogenetic network. Again we will look at just the deletion SVs and filter them to a new VCF. The only other items we need are the coordinates of theIGH gene in the reference and the names of all the samples in the population. We can extract these sames with another bcftools query execution.

```
$  bcftools query -l deletions_human.vcf > sample_names.txt
```

We are now ready to extract the sequences for the IGH gene for every sample in our population. To do this we will use the bcftools consensus feature which applies the variants in the VCF file to the reference genome. We also specify which sample name we will be focusing on in the VCF file and write its respective results to a new file in the fasta format.

```
$   while read val1; do
        echo ">${val1}" >> human_igh.txt

        samtools human.fa chr14:105430726-105759156 |
            bcftools consensus -s ${val1} deletions_human.vcf |
            tail -n +2 >>  human_igh.txt

    done <sample_names.txt
```

Now that we have extracted the sequences we can proceed with analysis via SplitTree as before. In the figure below we can see the much more diverse spread of distances amongst the samples compared to that in figure 5.

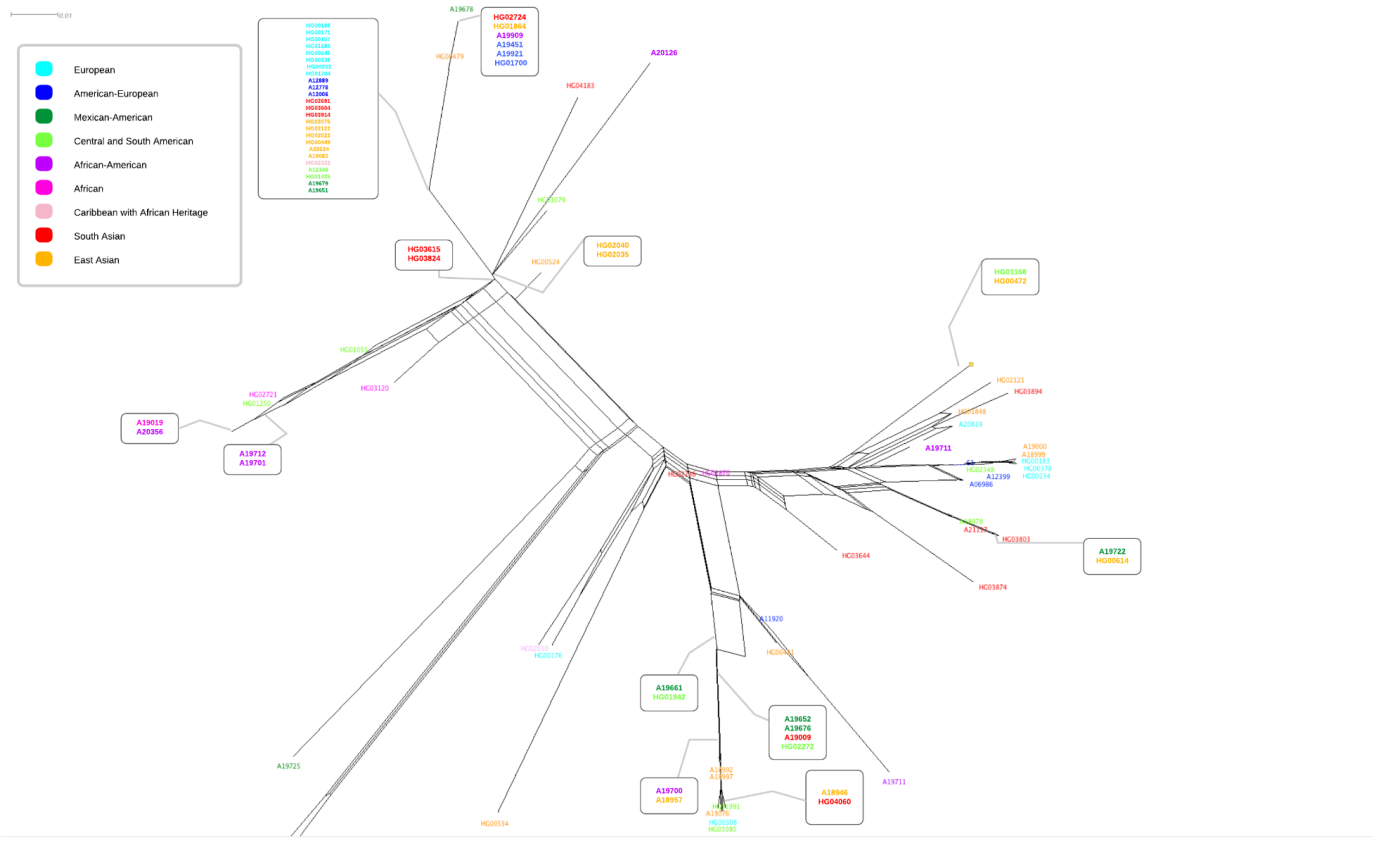

Figure 6: Equal Angle NeighborNet results for human immunoglobulin gene with SV deletions. The diversity in spits among taxa is indicative of the divergence produced by IGH's constant somatic recombination.

Authors: Devin Burke, Murad Chowdhury

Glossary

Tools

BWA

BWA is a tool used to align sequences to a reference genome using the Burrows-Wheeler Alignment algorithm.
http://bio-bwa.sourceforge.net/bwa.shtml

Samtools

Samtools is a software package that allows for the interaction with high-throughput sequencing data. Samtools has sub-packages that allows for specific manipulation of files in the BAM/SAM/CRAM format as well as another for VCF files, called BCFtools.
http://www.htslib.org/

Mosdepth

Mosdepth calculates BAM/CRAM depth calculation for whole genome sequencing data.
https://github.com/brentp/mosdepth

Bedtools

Bedtools is a suite of tools useful for a wide range of genomic analysis tasks.
https://bedtools.readthedocs.io/en/latest/

Smoove

Smoove is a tool for fast calling and genotyping of structural variants on short read sequencing data.
https://github.com/brentp/smoove

SplitsTree

SplitsTree is a program used for the easy computation of split graphs on a sequence alignment input.
https://uni-tuebingen.de/fakultaeten/mathematisch-naturwissenschaftliche-fakultaet/fachbereiche/informatik/lehrstuehle/algorithms-in-bioinformatics/software/splitstree/

File formats

FASTQ

The FASTQ file format is used to store sequencing data as it is produced via next generation sequencing methods.
https://support.illumina.com/bulletins/2016/04/fastq-files-explained.html

BAM, BAI

The Binary Alignment Map (BAM) file format is a compressed binary file that represents aligned sequences. The BAM is the binary form of the Sequence Alignment Map (SAM) file format. The respective Binary Alignment Index (BAI) file is the index file associated with each BAM file.
https://samtools.github.io/hts-specs/SAMv1.pdf

BED

The Browser Extensible Data (BED) file format is specifically used to store genomic regions in the form of coordinates.
https://uswest.ensembl.org/info/website/upload/bed.html

VCF

The Variant Call Format (VCF) file format is used for the storing of genetic sequence variation along with meta-information associated with each variant.
https://samtools.github.io/hts-specs/VCFv4.2.pdf