Using IGV to explore read mapping data

In this document, we will go over a few of the ways that you can use the mapping data visualization tool, Interactive Genome Viewer (IGV), to become comfortable with what SAM files tell you about read mapping. We're going to start by downloading some raw data (some sequencing reads and a "reference" sequence to compare to), performing read mapping, and then exploring the results. After performing an initial run through of the mapping process, we will then manually manipulate our reference sequence to see what our changes do to the mapping results. By making known modifications to our reference and seeing the impact of those changes on the mapping data, I hope that the meaning of the information in SAM files will become clear.

A note before we start: sequencing data files are large, so I recommend you make sure you have at least 5GB of free hard drive space if you want to follow along with the commands in this document.

Installing Integrative Genome Viewer (IGV)

You can install IGV from the website. Windows users should download the version that has Java included unless they have a specific reason to use their own Java. Mac users with Apple Silicon (M1, M2, etc) Macs should consult the message on the download page and ask a TA for help if you have issues.

After downloading the installer, proceed with installation as you would for other software for your operating system.

Setting up our data

Before we can map reads, we need the two basic components of any read mapping workflow: some reads, and something to map them to.

Let's start with the thing we are going to map to. The sequence to which reads are mapped is often called the "reference" sequence. Quick tangent: the name "reference" can be a little misleading as it implies that the sequence is of high quality or otherwise trustworthy. In some cases you do map to high quality sequences. However, if you were to generate random sequence and then map reads to it, the random sequence would still be called the reference in that context. The word reference is just a name to distinguishing the reads from the thing we are mapping them to.

As we are planning to add the code written for this assignment into our overall magnumopus, which we are using to extract 16S ribosomal RNA genes, let's use a 16S gene as our reference. An easy source for 16S genes is the Greengenes database. You can select any reference you like from that database and experiment with mapping reads to it. However, as the 16S gene is quite variable, you will have an easier time if you stick with reads and reference sequence from somewhat closely related organisms. For this demo we will stick with gram-negative bacteria from the phylum *Pseudomonadota* (previously called *Proteobacteria*) such as *Escherichia coli*, *Pseudomonas aeruginosa*, and *Vibrio cholerae*, which we have been working with already. For this demo I will use a *P. aeruginosa* reference sequence. To get the DNA sequence corresponding to the gene, just scroll to the bottom of the window and click "Copy as DNA", which will copy the sequence to your clipboard. You can then paste the sequence into a file, add a header line, and now we have a reference sequence to work with. Mine looks like this:

```
>Pa_16S
TGAACTGAAGAGTTTGATCATGGCTCAGATTGAACGCTGGCGGCAGGCCTAACACATGCAAGTCGAGCGGATGAAGGGAGCTT
GCTCCTGGATTCAGCGGCGGACGGGTGAGTAATGCCTAGGAATCTGCCTGGTAGTGGGGGATAACGTCCGGAAACGGGCGCTA
ATACCGCATACGTCCTGAGGGAGAAAGTGGGGGATCTTCGGACCTCACGCTATCAGATGAGCCTAGGTCGGATTAGCTAGTTG
GTGGGGTAAAGGCCTACCAAGGCGACGATCCGTAACTGGTCTGAGAGGATGATCAGTCACACTGGAACTGAGACACGGTCCAG
ACTCCTACGGGAGGCAGCAGTGGGGAATATTGGACAATGGGCGAAAGCCTGATCCAGCCATGCCGCGTGTGTGAAGAAGGTCT
TCGGATTGTAAAGCACTTTAAGTTGGGAGGAAGGGCAGTAAGTTAATACCTTGCTGTTTTGACGTTACCAACAGAATAAGCAC
CGGCTAACTTCGTGCCAGCAGCCGCGGTAATACGAAGGGTGCAAGCGTTAATCGGAATTACTGGGCGTAAAGCGCGCGTAGGT
GGTTCAGCAAGTTGGATGTGAAATCCCCGGGCTCAACCTGGGAACTGCATCCAAAACTACTGAGCTAGAGTACGGTAGAGGGT
GGTGGAATTTCCTGTGTAGCGGTGAAATGCGTAGATATAGGAAGGAACACCAGTGGCGAAGGCGACCACCTGGACTGATACTG
ACACTGAGGTGCGAAAGCGTGGGGAGCAAACAGGATTAGATACCCTGGTAGTCCACGCCGTAAACGATGTCGACTAGCCGTTG
GGATCCTTGAGATCTTAGTGGCGCAGCTAACGCGATAAGTCGACCGCCTGGGGAGTACGGCCGCAAGGTTAAAACTCAAATGA
ATTGACGGGGGCCCGCACAAGCGGTGGAGCATGTGGTTTAATTCGAAGCAACGCGAAGAACCTTACCTGGCCTTGACATGCTG
AGAACTTTCCAGAGATGGATTGGTGCCTTCGGGAACTCAGACACAGGTGCTGCATGGCTGTCGTCAGCTCGTGTCGTGAGATG
TTGGGTTAAGTCCCGTAACGAGCGCAACCCTTGTCCTTAGTTACCAGCACCTCGGGTGGGCACTCTAAGGAGACTGCCGGTGA
CAAACCGGAGGAAGGTGGGGATGACGTCAAGTCATCATGGCCCTTACGGCCAGGGCTACACACGTGCTACAATGGTCGGTACA
AAGGGTTGCCAAGCCGCGAGGTGGAGCTAATCCCATAAAACCGATCGTAGTCCGGATCGCAGTCTGCAACTCGACTGCGTGAA
GTCGGAATCGCTAGTAATCGTGAATCAGAATGTCACGGTGAATACGTTCCCGGGCCTTGTACACACCGCCCGTCACACCATGG
GAGTGGGTTGCTCCAGAAGTAGCTAGTCTAACCGCAAGGGGGACGGTTACCACGGAGTGATTCATGACTGGGGTGAAGTCGTA
ACAAGGTAGCCGTAGGGGAACCTGCGGCTGGATCACCTCCTTA
```

Next we need some sequencing reads to map to our reference. You can search for sequencing reads from millions of experiments on the NCBI SRA or linked databases such as the European Nucleotide Archive. In this demonstration we'll be mapping *P. aeruginosa* reads to the *P. aeruginosa* reference so we have a simple mapping output. I'll use these reads. You can download reads from the NCBI SRA website directly, but it is sufficiently laborious that I would recommend using the command line tool sra-tools even for single samples.

Downloading reads with sra-tools

sra-tools can be installed easily using conda. I recommend you download the latest version as some of the previous versions have bugs. You may need to create a new conda environment in which to install sra-tools or downgrade your Python. In my testing, python3.13 is not compatible with the latest version of sra-tools because of dependency conflicts, and the version of sra-tools that is compatible with python3.13 does not work for me.

After installing sra-tools you can use the included command line utilities to download sequencing reads using their accessions. We will be using two of these tools: `prefetch` and `fasterq-dump`. `prefetch` downloads the raw SRA archive data from the NCBI servers, while `fasterq-dump` unpacks those data into raw reads file. `fasterq-dump` can actually perform both steps, but it is faster to use to two tools separately (more than twice as fast in my tests) and `prefetch` can also resume a download if it was interrupted.

The download step can be done by simply running `prefetch` and providing the accession of the sample you want to retrieve. `prefetch` will default to downloading to your current directory, but we will want to specify the download directory with `-O`. sra-tools allows users to configure this default download location so specifying it will allow us to override any user configuration and make sure that we know where the downloaded files are. This isn't so important here, but if you were to use sra-tools in your magnumopus eventually, then this might be something you would care about.

```
$ prefetch -O temp_files SRR21376282
2024-10-27T18:39:56 prefetch.3.1.1: 1) Resolving 'SRR21376282'...
2024-10-27T18:39:57 prefetch.3.1.1: Current preference is set to retrieve SRA
Normalized Format files with full base quality scores
2024-10-27T18:39:57 prefetch.3.1.1: 1) Downloading 'SRR21376282'...
2024-10-27T18:39:57 prefetch.3.1.1:  SRA Normalized Format file is being retrieved
2024-10-27T18:39:57 prefetch.3.1.1:  Downloading via HTTPS...
2024-10-27T18:40:04 prefetch.3.1.1:  HTTPS download succeed
2024-10-27T18:40:07 prefetch.3.1.1:  'SRR21376282' is valid: 489923126 bytes were
streamed from 489907485
2024-10-27T18:40:07 prefetch.3.1.1: 1) 'SRR21376282' was downloaded successfully
2024-10-27T18:40:07 prefetch.3.1.1: 'SRR21376282' has 0 unresolved dependencies
```

You should now have a directory in your working dir called temp_files with the following contents:

```
$ tree temp_files/
temp_files/
└── SRR21376282
    └── SRR21376282.sra

1 directory, 1 file
```

That .sra file contains all the data associated with the sample we downloaded. Next we want to extract the sequencing reads from that file. We do that with `fasterq-dump`, which takes an optional output directory with `-O` and the directory containing our .sra file (sometimes there is more than one file, so you should always provide the directory path not the sra file). I'm going to write my reads to a directory called "pa_reads" as the accession doesn't tell us the organism name so I'll use the directory to tell us more about the files.

This step is going to read and decompress a lot of data so it will take a while before any output is printed to your terminal (about 30 seconds to a minute).

```
$ fasterq-dump -O pa_reads temp_files/SRR21376282/
spots read      : 5,469,234
reads read      : 10,938,468
reads written   : 10,938,468
```

The printout from `fasterq-dump` is just telling us how much data there is. Each spot corresponds to one of the patches of reads described in the Illumina video in the main document. Each spot produces a forward and reverse read of a single fragment of DNA.

`fasterq-dump` has automatically split the .sra file into two fastq files full of reads because these are paired Illumina reads. If we look at just the first line of each file to see the first header, we will see that they actually have the same headers

```
$ head -1 pa_reads/*
==> pa_reads/SRR21376282_1.fastq <==
@SRR21376282.1 1 length=151

==> pa_reads/SRR21376282_2.fastq <==
@SRR21376282.1 1 length=151
```

The headers are the same in each file in order to correspond the two read mates that make up a pair. The read SRR21376282.1 in each file is the name for the two mates. Each file similarly has a SRR21376282.2, SRR21376282.3, etc. which are the mates for the second, third, etc. pairs. FASTQ headers vary between sources (e.g., databases or whether it's straight out of a sequencing machine). SRA headers have the format `@<SRA accession>.<read number> <read number again> <length of read>`.

If we now look at the first four lines of each file to view the entire first entry in each, we will see that while their headers are the same, the sequence data in each is not. Instead, these are partial sequences of a fragment of DNA read from either end (and therefore come from opposite strands of the DNA fragment).

```
$ head -4 pa_reads/*
==> pa_reads/SRR21376282_1.fastq <==
@SRR21376282.1 1 length=151
TNGCCCCGGGTCGCCCTGCTCAACGTGGGGACCGAGGAGATCAAGGGCAACCAGCAGGTCAAGCTGGCGGCGAGCCTGTTGCA
GAAGGCGCAGGGGTTGAATTTCAGCGGATATATAGAGGGCAGATCGGAAGAGCACACGTCTGAACTCC
+SRR21376282.1 1 length=151
F#FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

==> pa_reads/SRR21376282_2.fastq <==
@SRR21376282.1 1 length=151
GCCCTCTATATATCCGCTGAAATTCAACCCCTGCGCCTTCTGCAACAGGCTCGCCGCCAGCTTGACCTGCTGGTTGCCCTTGA
TCTCCTCGGTCCCCACGTTGAGCAGGGCGACCCGGGGCGAAGATCGGAAGAGCGTCGTGTAGGGAAAG
+SRR21376282.1 1 length=151
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFF:FFF:FFFFFFFF
```

Take a moment to look over these reads and identify each of the components described in the main document. If you pay attention to the phred scores, you'll notice that the read from the first file has one absolutely awful quality base (the one with the quality score of #). In that case, the quality score is so low that the read has an N in the sequence which indicates that the sequencer doesn't know what the base is.

Subsetting reads when there are more than you need

10,000,000 reads is a huge amount of data and these files are very large as a result.

```
$ du -h pa_reads/*
2.0G    pa_reads/SRR21376282_1.fastq
2.0G    pa_reads/SRR21376282_2.fastq
```

Each reads file contains half of the 10,938,468 reads that `fasterq-dump` reported having written. That's 5,469,234 reads! We can estimate how much sequencing depth that represents and decide whether that is more data than we need. Sequencing depth is a measure of how many times each base in a genome was sequenced (we'll see it a bit later when we visualize reads mapped to a reference sequence). Because these reads are derived from the same bacterial species as our reference sequence, it's likely that the genomes are quite similar. For our depth estimate we'll assume they are identical. **P. aeruginosa** has a genome size of about 6.5 million bases. As we saw when we looked at the first read in each of our read files, our 5 million reads are each about 150 bases long. That means we have 10,000,000 * 150 bases of sequence which is 1,500,000,000 or 1.5 billion bases of sequence. If the sequencing is pretty even across our genome then that means we should expect a depth equal to the number of bases sequenced divided by the length of our genome so 1,500,000,000 / 6,500,000 = 230. That's way more than we need. We only need depth of 20 or so in order to be able to judge which base is present at each position in our reads. We can therefore subset our data to about 1/10th its current size. Subsetting will save us storage space and speed up processing times without negatively impacting the accuracy of any analysis.

We can subset with `seqtk`. In order to subset 1/10th of our original reads, we will want 500,000 from each of the reads files. `seqtk` offers a subcommand called `sample` to randomly subsample a specified number of reads. Because we are using paired data we need to specify the random seed value so that we get the same read numbers from each file (this is an important step to remember!!! The information in pairs of reads is hugely valuable when working with Illumina data!). This step will take a while again as there are so many reads. This investment of time is worth it moving forward though!

```
seqtk sample -s1 pa_reads/SRR21376282_1.fastq 500000 >
pa_reads/sub_SRR21376282_1.fastq
seqtk sample -s1 pa_reads/SRR21376282_2.fastq 500000 >
pa_reads/sub_SRR21376282_2.fastq
```

Now that we have subsetted our reads we can delete the original reads. If we need them again later we can always download them again and otherwise they are using up a feature length movie file worth of storage space.

```
$ du -h pa_reads/*
2.0G    pa_reads/SRR21376282_1.fastq
2.0G    pa_reads/SRR21376282_2.fastq
165M    pa_reads/sub_SRR21376282_1.fastq
165M    pa_reads/sub_SRR21376282_2.fastq
```

Mapping reads

Now that we have our reference sequence and a sensible number of reads, lets map our reads and actually get to the main point of this document...

I will use `minimap2` here to demonstrate mapping. `minimap2` is quick and works for both long and short reads. You might also like to use another mapping tool such as `bwa-mem` or `bowtie` and are free to do so. When using `minimap2` with short reads, you simply use the option `-ax sr` which activates the short read preset and changes the output format to SAM. Otherwise the usage is the same as for other modes: `minimap2 [options] -x <mode> <ref.fasta> <read1.fastq> [<read2.fastq>]`

I'm going to save the output in a file called original.sam so that we can keep track of the unchanged file as well as any modifications we make to it.

```
$ minimap2 -ax sr Pa_16S.fna pa_reads/sub_SRR21376282_1.fastq
pa_reads/sub_SRR21376282_2.fastq > original.sam
[M::mm_idx_gen::0.005*0.49] collected minimizers
[M::mm_idx_gen::0.005*0.66] sorted minimizers
[M::main::0.006*0.64] loaded/built the index for 1 target sequence(s)
[M::mm_mapopt_update::0.006*0.64] mid_occ = 1000
[M::mm_idx_stat] kmer size: 21; skip: 11; is_hpc: 0; #seq: 1
[M::mm_idx_stat::0.006*0.64] distinct minimizers: 249 (100.00% are singletons);
average occurrences: 1.000; average spacing: 6.173; total length: 1537
[M::worker_pipeline::4.394*0.67] mapped 331126 sequences
[M::worker_pipeline::7.179*0.44] mapped 331126 sequences
[M::worker_pipeline::10.032*0.34] mapped 331126 sequences
[M::worker_pipeline::10.091*0.34] mapped 6622 sequences
[M::main] Version: 2.28-r1209
[M::main] CMD: minimap2 -ax sr Pa_16S.fna pa_reads/sub_SRR21376282_1.fastq
pa_reads/sub_SRR21376282_2.fastq
[M::main] Real time: 10.091 sec; CPU: 3.424 sec; Peak RSS: 0.408 GB
```

We now have a SAM file so we can finally take a look at the data that this assignment is concerned with! Let's take a look at the first four lines of the SAM file and quickly go over what we see. You can find a description of everything in SAM format in the documentation.

```
$ head -5 original.sam
@HD     VN:1.6  SO:unsorted     GO:query
@SQ     SN:Pa_16S       LN:1537
@PG     ID:minimap2     PN:minimap2     VN:2.28-r1209   CL:minimap2 -ax sr
Pa_16S.fna pa_reads/sub_SRR21376282_1.fastq pa_reads/sub_SRR21376282_2.fastq
SRR21376282.1333952     77      *       0       0       *       *       0       0
GGCAGCTTCAGGTGCGACTGGCTGAGCAGCAGCTCCACGCCGCTGTCCTCCAGCATGTAGGCCTGGCGCTCCTCGGGATACTC
CGGGTCCACCGGCACGTAGGCGCCGCCGGCCTTGAGGATCGCCATCAGGGCCACGACCATCTCGATGG
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFF,FFFFFFFFF rl:i:0
SRR21376282.1333952     141     *       0       0       *       *       0       0
GAGCGCGGGGTCGGTGCGGACCGCCTGGTGGGCGTGGCCATGGAGCGCTCCATCGAGATGGTCGTGGCCCTGATGGCGATCCT
CAAGGCCGGCGGCGCCTACGTGCCGGTGGACCCGGAGTATCCCGAGGAGCGCCAGGCCTACATGCTGG
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF rl:i:0
```

SAM files typically being with header lines that provide information about the file. These header lines begin with @. In this file we have two header lines: an overall header line "@HD", which includes information about the SAM format version (VN), the order with which alignments are sorted (SO), and how alignments are grouped (GO).

The next header line is a dictionary of the sequence names (SN) and their lengths (LN) against which reads were mapped. We only have one sequence in our reference fasta file so there's only one here.

The final header line gives information about software that has been used to manipulate this SAM file. Whenever you run a program that modifies a SAM file, the software will (or at least should) add a line to the header to record what was done. This line includes information like the name of the software (ID and PN), the version of the software (VN) and the command that was run (CL). This information can be helpful to check the details of how a specific SAM file has been modified. We'll see more of these lines added as we proceed in this document.

After the header are the alignment lines. The two shown above are a little bare for reasons that will become clear shortly. However, they still have all the mandatory fields of SAM format that were described in the main document.

The first column of each alignment indicates the name of the read. If you look at the two lines shown above, you will see that the read name is the same. This can happen for a few reasons. Firstly and most commonly, this corresponds to an alignment of the two mates of a single pair. As we saw when we looked at the FASTQ files above, each read in the first FASTQ file has a mate with the same name in the second file. Each of those reads will be mapped as a separate sequence and any alignments will get their own lines in the resulting SAM file. The second possibility when you see multiple lines with the same read name in column 1 is that a single read mapped to multiple locations. There are two cases in which this can happen which we will explore later: supplemental mapping and secondary mapping.

The second column is the FLAG column, which includes information about how the read mapped. In order to identify which of the flags are set, we need to convert the flag value to binary. Alternatively, for manual checking we can just copy the value into a handy website to tell us which flags are set. If we check the values of 77 and 141 on that website we see that these alignments both have the 1, 4, and 8 flag set and the first alignment also has flag 64 while the second has flag 128. Those flags tell us that the reads are both paired (flag 1) (which we already knew because these are Illumina paired reads), that the read is unmapped (flag 4), that its mate is also unmapped (flag 8), and that one is the first read in a pair (flag 64) and one is the second (flag 128).

Most of those flags aren't particularly useful to us right now. Most importantly the 4 and 8 flag are set which tell us that neither read is mapped. When a read and its mate are not mapped, that generally tells you that the fragment of DNA from which the reads were derived does not exist in your reference sequence. Here we are mapping reads to the 16S rRNA gene, which is a 1,500 base portion of the **P. aeruginosa** genome (which is 6.5 million bases), so it's not surprising if most reads don't map to our reference.

Filtering mapped reads

Before we view our mapped reads in IGV I just want to hold back for one more step. When we view mapped reads, we are (of course) not going to be viewing the unmapped reads like the two described above. You often won't care about unmapped reads and they are of no use to us here either. Therefore we can just remove them from our SAM file.

Manipulating SAM files like this is easy; we just use samtools. samtools is the most downloaded package on Bioconda. In fact the top three packages are all for working with SAM files and related data. Install samtools into your conda environment if you haven't already if you want to follow along with the next steps.

To filter a SAM file to include or exclude alignmets based on their flags we can use samtools view. Specifically, in our case we want to exclude unmapped reads. To do so we can exclude any alignments with the 4 bit set in their flag by using the `-F` option. We'll also use `-h` to include the header lines in the output:

```
$ samtools view -h -F 4 original.sam > mapped.sam
$ head -6 mapped.sam
@HD     VN:1.6  SO:unsorted     GO:query
@SQ     SN:Pa_16S       LN:1537
@PG     ID:minimap2     PN:minimap2     VN:2.28-r1209   CL:minimap2 -ax sr
Pa_16S.fna pa_reads/sub_SRR21376282_1.fastq pa_reads/sub_SRR21376282_2.fastq
@PG     ID:samtools     PN:samtools     PP:minimap2     VN:1.21 CL:samtools view -
hF 4 original.sam
SRR21376282.4691250     99      Pa_16S  1456    60      82M69S  =       1483    82
GGTTACCACGGAGTGATTCATGACTGGGGTGAAGTCGTAACAAGGTAGCCGTAGGGGAACCTGCGGCTGGATCACCTCCTTAA
TCGAAGATCTCAGCTTCTTCATAAGCTCCCACACGAATTGCTTGATTCACTGGTTAGACGATTGGGTC
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF,FFFFFFFFFFFFFFFFFFFFF NM:i:0
ms:i:164        AS:i:164        nn:i:0  tp:A:P  cm:i:8  s1:i:75 s2:i:0  de:f:0
rl:i:0
SRR21376282.4691250     147     Pa_16S  1483    52      55M96S  =       1456    -82
GGTGAAGTCGTAACAAGGTAGCCGTAGGGGAACCTGCGGCTGGATCACCTCCTTAATCGAAGATCTCAGCTTCTTCATAAGCT
CCCACACGAATTGCTTGATTCACTGGTTAGACGATTGGGTCTGTAGCTCAGTTGGTTAGAGCGCACCC
FFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFF NM:i:0
ms:i:110        AS:i:110        nn:i:0  tp:A:P  cm:i:2  s1:i:75 s2:i:0  de:f:0
rl:i:0
```

As you can see, we now have an extra header line that includes the command we ran. We also have different alignments at the top of our file. These alignments have much more information in them. Have a look at the table in the main document to see what each field means. We won't discuss all the fields here, but for now let's just have a look at what the two flags tell us.

The flags 99 and 147 tell us that these reads have flags 1, 2, and either 16 or 32, and either 64 or 128 set. Three of those flags are new to us: 2, 16, and 32. Flag 2 tells us that the read is mapped in its proper pair. What that means is that the read and its mate both map as you would expect given that they are from the same DNA fragment: they are oriented towards one another and appropriately close together. This flag is set using the same sort of process you used to identify primers that will yield an amplicon in your isPCR implementation. Flags 16 and 32 tell us which strand each read maps to.

Together these flags communicate that these reads mapped in a pretty much expected way. They are mapped how you would expect if they had come from a piece of your reference sequence or something quite like it (not making any statements about the actual alignement here - this just considers orientation and mapping status, not mismatches etc.). There are 4 flag column values that indicate normal mapping like this: 99, 147, 83, and 163. Have a look at which flag bits those correspond to as an exercise to make sure you understand why those would be "normal" mappings.

Now that we have removed unmapped reads we have acheived more than just moving some mapped reads to the top of the file. We have removed the majority of the alignments in our SAM file (because they had no value to us). When mapping to small reference sequences, filtering out unmapped reads can be a big saving in storage and analysis time.

```
$ wc -l *.sam
      836 mapped.sam
  1000003 original.sam
  1000839 total
$ du -h *.sam
356K    mapped.sam
333M    original.sam
```

Note that samtools provides flexible filtering based on the FLAG column. `samtools view` allows you to specify flags in a few different ways (see the docs for the list) and you can specify one or more flags. In the decimal format that flags are shown in the FLAG column, we could keep only reads that have a FLAG value of 99 with `-f 99` as `-f` only returns alignmets with all of the requested flags. `samtools view` offers four flag filtering options which cover all the different ways you might want to filter flags. I'll list them here with a short explanation and a summary of the logical operation in parentheses:

- `-f` KEEP alignments with ALL of the indicated flags (inclusive AND)
- `-F` REMOVE alignments with ANY of the indicated flags (exclusive OR)
- `--rf` KEEP alignments with ANY of the indicated flags (inclusive OR)
- `-G` REMOVE alignments with ALL of the indicated flags (exclusive AND)

Viewing read mapping in IGV

Before we can view our SAM file in IGV we need to perform one more step. IGV wants our SAM file to be sorted so that reads mappings are ordered by the location leftmost base in the reference they map to (i.e., the POS column of the SAM file). We can acheive that with `samtools sort`. The only special option we need here is `-O SAM` to tell `samtools sort` to output SAM format. Its default is to output BAM, which is a binary format equivalent of SAM which takes up less space and is quicker to work with programmatically. We'll stick with SAM format so that we can look at the file contents manually.

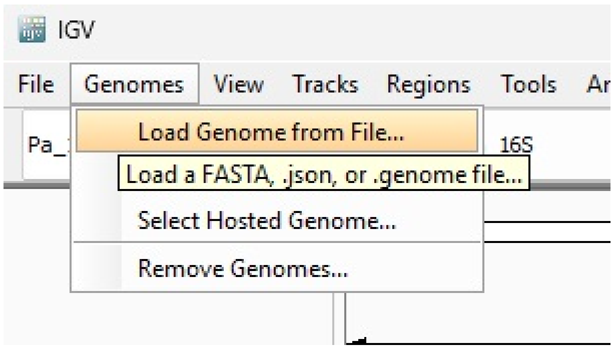
```
$ samtools sort -O SAM mapped.sam > mapped_sorted.sam
$ head -7 mapped_sorted.sam
@HD     VN:1.6  SO:coordinate
@SQ     SN:Pa_16S       LN:1537
@PG     ID:minimap2     PN:minimap2     VN:2.28-r1209   CL:minimap2 -ax sr
Pa_16S.fna pa_reads/sub_SRR21376282_1.fastq pa_reads/sub_SRR21376282_2.fastq
@PG     ID:samtools     PN:samtools     PP:minimap2     VN:1.21 CL:samtools view -h
-F 4 original.sam
@PG     ID:samtools.1   PN:samtools     PP:samtools     VN:1.21 CL:samtools sort -O
SAM mapped.sam
SRR21376282.1336028     163     Pa_16S  1       57      82S69M  =       1       70
GTGAATTCGAGAGTTTTATCTCTTTTTAAGAGAGTGCGATTGCTGAGCCAAGTTTAGGGTTTTCTCAAAACCCAAGCAGTATT
GAACTGAAGAGTTTGATCATGGCTCAGATTGAACGCTGGCGGCAGGCCTAACACATGCAAGTCGAGCG
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF NM:i:0
ms:i:138        AS:i:138        nn:i:0  tp:A:P  cm:i:7  s1:i:59 s2:i:0  de:f:0
rl:i:0
SRR21376282.878936      99      Pa_16S  1       60      42S109M =       1       128
TGCTGAGCCAAGTTTAGGGTTTTCTCAAAACCCGAGCAGTATTGAACTGAAGAGTTTGATCATGGCTCAGATTGAACGCTGGC
GGCAGGCCTAACACATGCAAGTCGAGCGGATGAAGGGAGCTTGCTCCTGGATTCAGCGGCGGACGGGT
F,F,FFFFFFF:F::F:FFFFF:FFFFFFFF:FFFFFF,FFFF,FF:FFFF:FFF:FFFFFFFF:FFFFFF:F:FF:FFFFF,
FFFFFFFFFFF:F:FFF,FF:FF,FFFF:FFFFFF:,FF,FFFFFFF:FFFFF:FFF::FFFFFF::F NM:i:0
ms:i:218        AS:i:218        nn:i:0  tp:A:P  cm:i:12 s1:i:116        s2:i:0
de:f:0  rl:i:0
```

Now you can see that the 4th column, POS, which is the leftmost position in the reference to which a read mapped, is 1 in the first two alignments shown (SAM uses 1-base counting). All the alignments in the file are now sorted by the POS column.

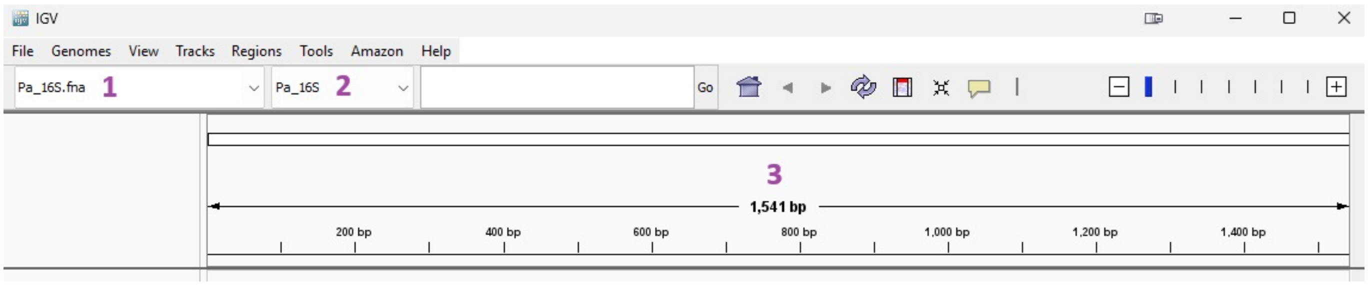
Now let's look at our mapping in IGV. We'll start by just opening the software and loading our data. Then I'll walk through the important elements of the interface and visualization.

One tip before we start: at least on windows, it helps to pin the directory with your reads to quick access. When you use the menu to open a SAM or fasta file in IGV, it doesn't typically remember which directory you last navigated to, but starts at your root directory. The quick access reduces the pain of navigating through your file system over and over.

Once you open IGV, the first thing you should do is load your data. You can load the reference sequence using the "Genomes" -> "Load Genome from File..." menu as shown in the below image. That will bring up a file explorer window that you should use to navigate to the location of your fasta file.

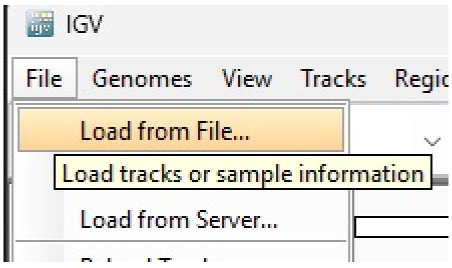


After loading the reference sequence you should see the information in the top portion of your window has updated. I've numbered the three visual elements that will have changed.

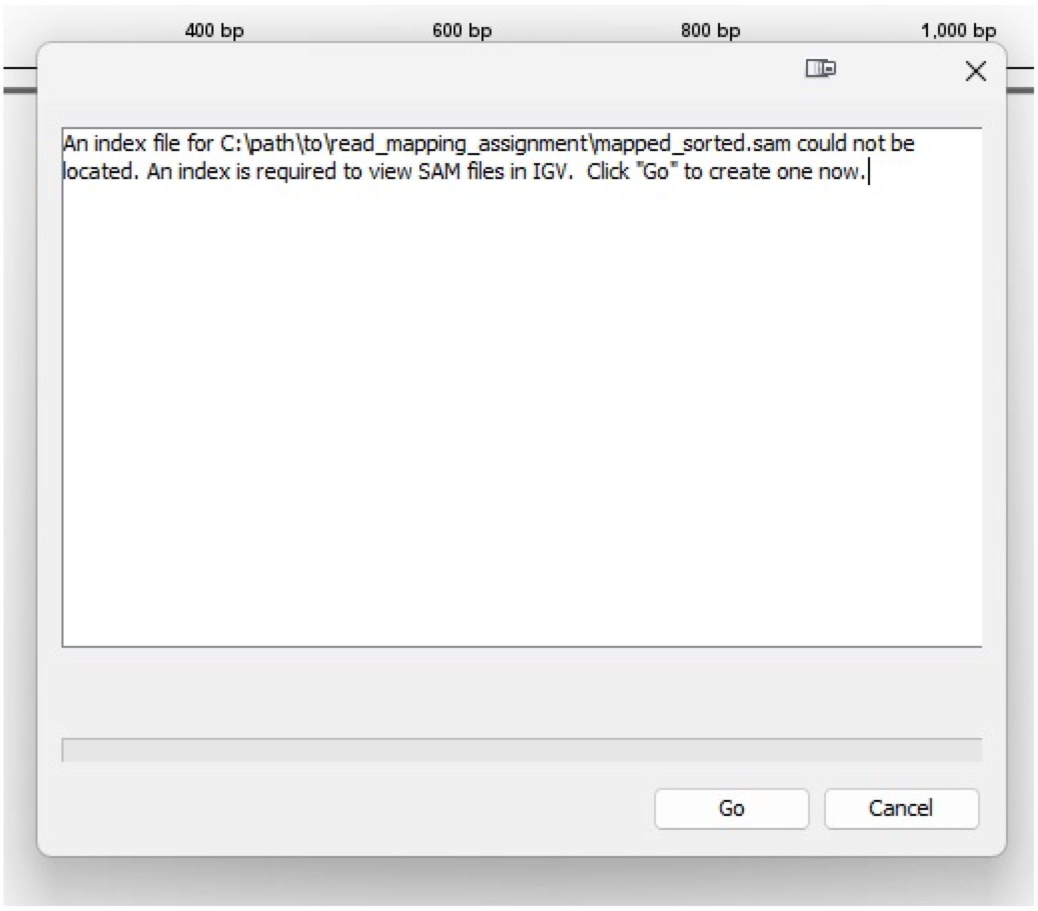


1. The name of the loaded "genome" file. IGV calls this a genome regardless of what your reference sequence actually is. In our case it is just a small region of a genome.
2. The sequence within your reference file that we are currently viewing. If we had more than one sequence in our file then we could use this dropdown menu to switch between each one. The sequence name in this box is simply the FASTA header of each sequence.
3. A navigation bar to show what range of positions are currently being viewed. You can zoom using the + and - buttons at the top right of your window or by clicking and dragging a range within this panel.

Next let's load the SAM file. Mapping data is loaded using "File" -> "Load from File..."


IGV
File Genomes View Tracks Regic
Load from File...
Load tracks or sample information
Load from Server...

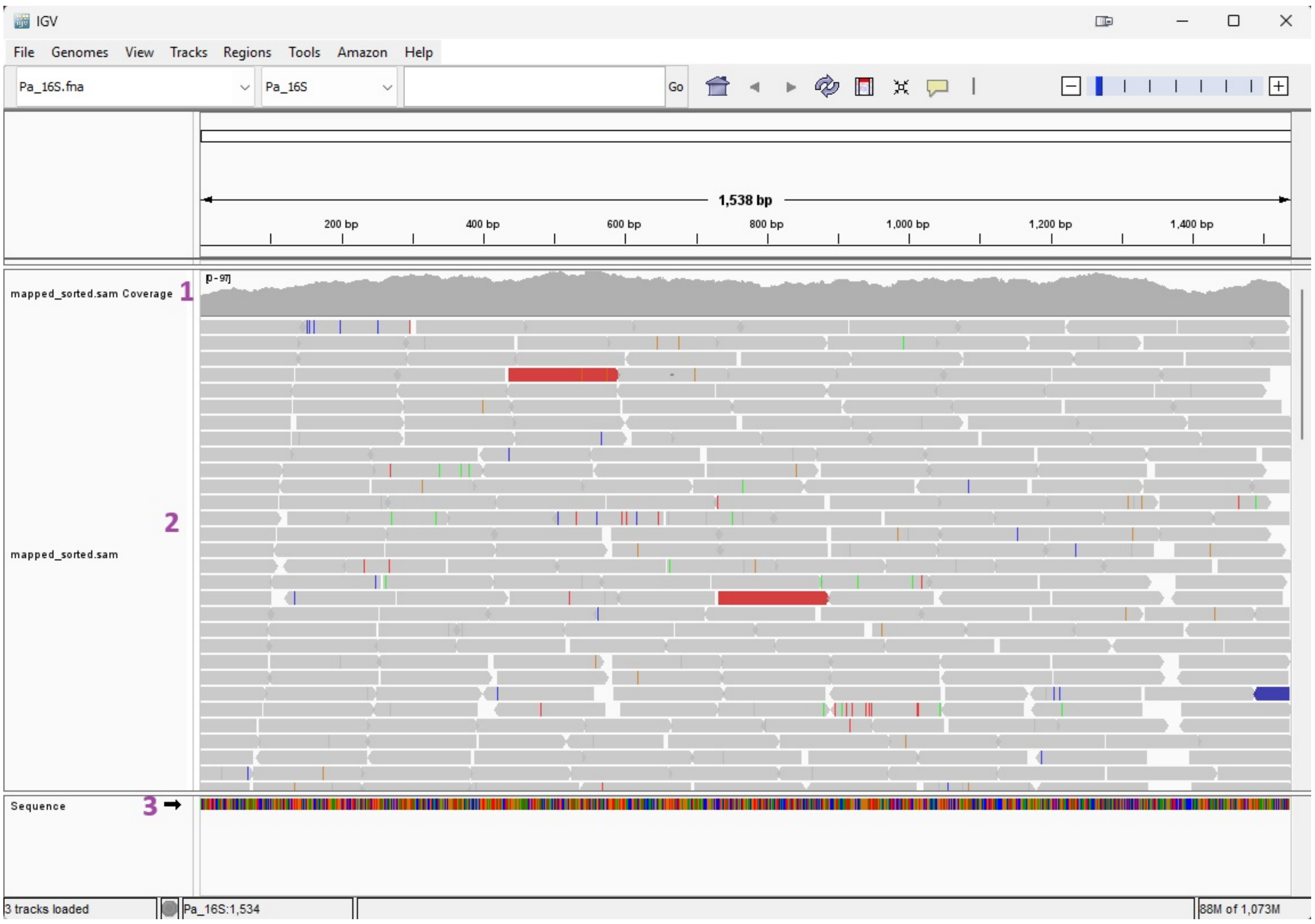
You then navigate to and select the sorted SAM file that we have just created. If you selected the right file then you should see a popup like the following



An index file for C:\path\to\read_mapping_assignment\mapped_sorted.sam could not be located. An index is required to view SAM files in IGV. Click "Go" to create one now.

Just click "Go" and IGV will create the necessary file which will have the same name as your SAM file plus a .sai extension. Note that if you were viewing a BAM file you would need to create the index yourself using `samtools index <BAM file path>`.

After clicking "Go" you'll see a quick progress bar animation in the popup and then the popup will disappear to finally reveal our mapped reads!



I have numbered three key visual elements in this view that tell us important information about how our reads are mapped:

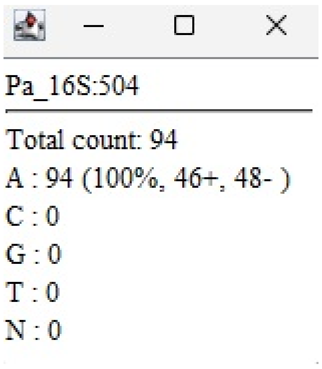
1. This bar plot shows the depth of coverage at each position in our reference sequence. Depth of coverage is a measure of how many reads map to each position in the reference sequence. That is what we tried to estimate above to determine how much to subsample our reads. We'll look at this bar plot in more detail below.
2. The horizonal grey (and other color) arrows below the depth bar plot are representations of individual mapped reads. Each arrow represents a single alignment of a read and corresponds to a single line in the SAM file. If you look closely at them (or zoom in a bit) you'll see that one end of each arrow is blunt and one end is triangular. This tells you which direction the reads was mapped. i.e., the arrow points in the mapped direction, so if the triangle is on the left, the read is mapped to the reverse strand of the reference. We'll talk about the red and blue arrows in a moment.
3. This stripe of color represents the sequence in the reference. If you zoom in all the way you will see this stripe change to bases, where each base has a different color associated with it. When zoomed out, each base is shown as a thin stripe of color as the letters would be too small to see.

Let's take a look at the first two those three elements in a bit more detail (that's all there is for the sequence in element 3).

Depth bar plot

You'll notice that the bar plot showing depth is not a straight line. Typically, when sequencing a genome or any other molecule(s) of DNA with Illumina sequencing, you fragment billions of copies of genome randomly and then sequence a random selection of those fragments (the process of processing the fragments during sample preparation is not 100% efficient so some fragments are lost). Because the process is random, you will always see randomness in the depth across your reference sequence. There are cases where you might have non-random differences in depth as welf. For example, the reference has one copy of a gene, but your reads came from an organism with two copies.

We can check the actual depth value at a single position by simply clicking on the bar chart. You will see a popup that shows the depth and what bases are present in the reads mapped to that position.



```
Pa_16S:504

Total count: 94
A : 94 (100%, 46+, 48- )
C : 0
G : 0
T : 0
N : 0
```

In the example popup I'm showing, there are 94 reads mapped to this position and all of them have an A at this position. Next to the A is the percent of reads with that base and the direction (forward: +, reverse: -) in which the reads mapped. You should expect to see a roughly even split of forward and reverse mapped reads for double stranded DNA so what we see here looks as we would expect.

It's hard to click positions precisely fully zoomed out, so you'll generally want to zoom in to interogate individual positions. If you just want to judge depth across the whole reference sequence, you can also just glance over the bar plot and use the Y-axis value range shown in brackets at the left of the bar plot.

Finally, you'll remember that we tried to subsample our reads to acheive a depth of about 20. As you can see in the bar plot, we missed that mark by about five times! If you map these reads to the **P. aeruginosa** genome we worked with for previous assignments, you probably would see an average depth of around 20 as the approach we took earlier is reasonable. The reason we are seeing such high depth here is because the 16S rRNA gene is often present in multiple copies in bacterial genomes. Indeed, you have already seen this - you got multiple amplicons from your isPCR module when amplifying 16S rRNA regions. What we are actually looking at in this IGV view is reads which came from probably five copies of the 16S gene (based on the ratio of expected to observed depth) all mapped to a single copy.

Alignments

The alignments are where our raw mapping data are being displayed and where most of the interesting information is to be found. We've already mentioned that reads can be shown as different colors so let's start with that. You can also read about how to interpret and control the alignment views in the IGV documentation.

Read color

Once we have mapped reads to a reference sequence, we gain information about how the two mates in each pair are related. Remember that the mates in a pair are two sequences which were derived from a single fragment of DNA. Each mate is the sequence from one end of that fragment of DNA towards the middle. If the fragment was short, then the two reads may overlap at the end, but if the fragment is long there will be no overlap. If there is overlap then we can guess at how long the fragment that was sequenced was. However, if there is no overlap then we have no way to infer how long the fragment was based just on the two reads. Once we map the reads to a reference we can simply count the number of bases between them to assess how long the fragment was (assuming the reads came from DNA with a sequence like our reference).

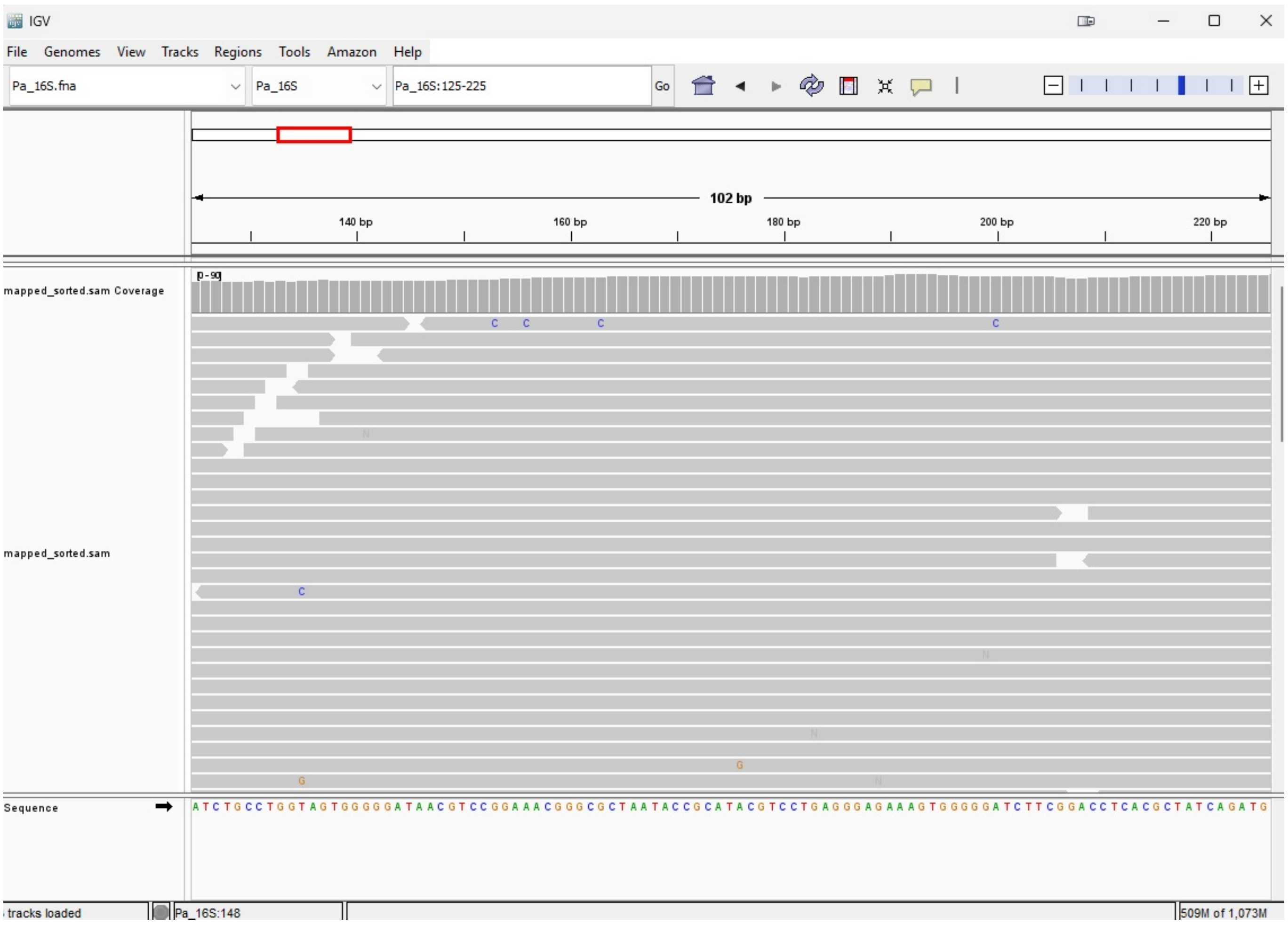
When performing the fragmentation step in sample preparation before Illumina, people typically try to aim for fragments that are a bit longer than the combined length of the reads they will generate. Overlapping reads are a waste of money as you are getting the same information twice. Therefore, getting fragments a bit longer than two reads maximizes the information per dollar.

IGV assesses the fragment size (called insert size by IGV because you "insert" the DNA between your adapter sequences during sample preparation) based on read mappings and then colors reads red if their fragment sizes are larger than expected and blue if the fragment sizes are smaller than expected. IGV does this when the reads are in the top or bottom 0.5% of fragment lengths, but you could also set expected fragment sizes manually. Grey reads mean their fragment size is within the expected range.

Stripey reads

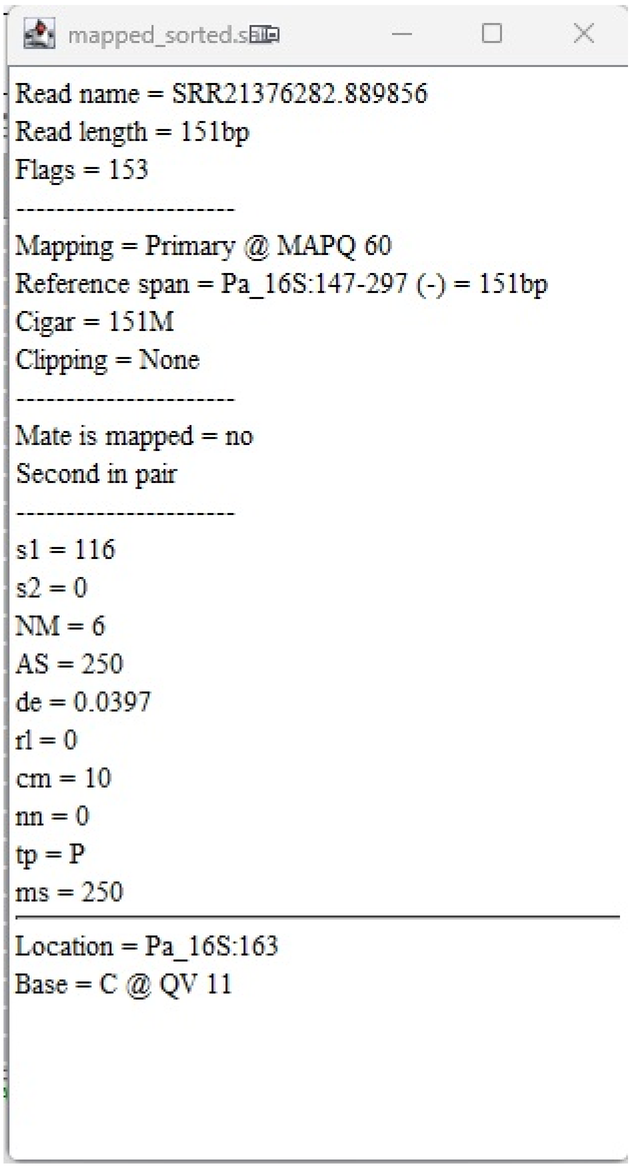
If you look closely, you'll see that some of the grey reads have thin stripes of color on them. These thin stripes indicate mismatches where the base in the read differs from the base in the reference. The color of the stripe indicates what the base is in the read. Generally, if you see a solid stripe from the top to bottom of your window where all your reads have a lined up stripe then that indicates that your reference and the DNA you sequenced actually differ at the shown position. In our case all the stripes are dispersed and the reads do not all agree on any differences relative to the reference. If we zoom in on a region we can take a closer look at some of these mismatches. I'll zoom in to about 125-225 to look at the blue stripes in the read on the top left of the above image.

Quick note: IGV does not (to my knowledge) offer the ability to change base colors to a color blind accessible palette. You will need to zoom in to the level where you can read bases to tell what the stripes are showing. If anybody finds a way to change the colors to fix this issue please let me know so I can add instructions to this document in future!



```
ATCTGCCTGGTAGTGGGGGATAACGTCCGGAAACGGGCGCTAATACCGCATACGTCCTGAGGGAGAAAGTGGGGGATCTTCGGACCTCACGCTATCAGATG
```

Looking more closely we can see that the blue striped in this read are actually C base calls. Let's click on one of the Cs to see more information about it. Clicking a read anywhere along the arrow will bring up a popup with all the SAM information for that alignment formatted in a readable way. At the bottom of that popup will be information about the specific base you clicked. I will click the third C in the top read, but I encourage you to click about a bit after reading this section to investigate the other mismatches in reads. The purpose of this document is just to orient you so you have the tools to explore after all!



```
mapped_sorted.s

Read name = SRR21376282.889856
Read length = 151bp
Flags = 153
----------------------
Mapping = Primary @ MAPQ 60
Reference span = Pa_16S:147-297 (-) = 151bp
Cigar = 151M
Clipping = None
----------------------
Mate is mapped = no
Second in pair
----------------------
s1 = 116
s2 = 0
NM = 6
AS = 250
de = 0.0397
rl = 0
cm = 10
nn = 0
tp = P
ms = 250
Location = Pa_16S:163
Base = C @ QV 11
```

Looking over the information in that popup, we can see that the FLAG value is 153. If you lookup that value online you'll see that it tells us that this read is mapped to the reverse strand, but its mate is unmapped. We expect to see that near the ends of our reference as we have mapped reads to a small part of the genome and so the mate of this read likely came from some DNA next to the 16S gene which is not present in our reference.

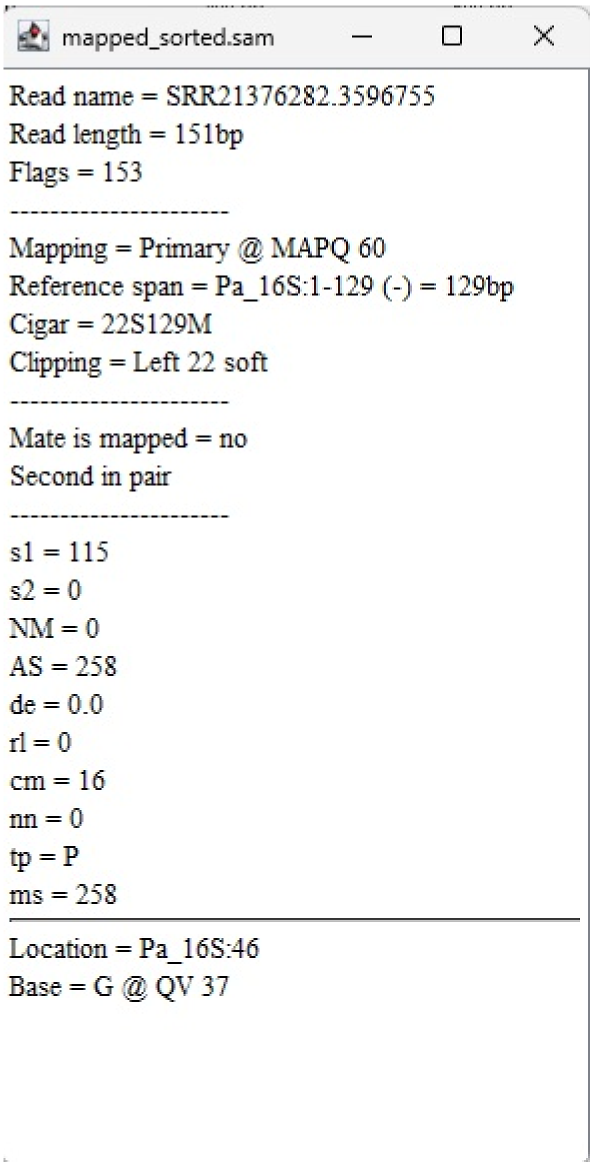
After the flags we see some other summary information about the mapping such as that this is a primary mapping and the cigar string is 151M (which means all 151 bases mapped without gaps or clipping).

Next is the tags column which includes information that is formatted according to a system defined by the mapping software. We'll ignore those data.

Finally we have the information about the base we clicked. We can see that the base is mapped to position 163 in the reference, that it is a C, and that the quality score for the base call is 11. A Q score of 11 corresponds to a 7% probability of error so that tells us (along with no other reads agreeing there is a C here) that this is likely a sequencing error and not really what the base was in the DNA this read came from.

Clipping

Next let's take a look at a read that mapped to the edge of our reference. Zoom out and then click on any read at the very left of the window. You should be able to find one with "S" in the CIGAR string quite easily.



```
Read name = SRR21376282.3596755
Read length = 151bp
Flags = 153
----------------------
Mapping = Primary @ MAPQ 60
Reference span = Pa_16S:1-129 (-) = 129bp
Cigar = 22S129M
Clipping = Left 22 soft
----------------------
Mate is mapped = no
Second in pair
----------------------
s1 = 115
s2 = 0
NM = 0
AS = 258
de = 0.0
rl = 0
cm = 16
nn = 0
tp = P
ms = 258
Location = Pa_16S:46
Base = G @ QV 37
```

As you can see in the above popup, this read has 22 soft clipped reads. Soft clipping means that the bases did not align to the reference sequence, but that they have been kept in the SAM entry of the alignment. Usually, soft clipping indicates that either the left of right end of the read does not match the reference, or that the read extends beyond the end of the reference. In our case the second situation has caused this soft clipping, but the first case is also something you might see. Crucial for your assignment is what I said about soft clipped bases being kept in the SAM entry. This read is 151 bases long, but the first 22 were soft clipped. That means that if you wanted to extract the base from this read that maps to the first base of the reference you would need to extract the 23rd base - the base after all the soft clipped bases. i.e., you should skip soft clipped bases in your code.

Explore before we continue

Next we're going to mess with our reference sequence to see how it changes our SAM file and IGV visualizations. Before we do, I strongly recommend you look around and click on some reads to make sure you are comfortable with what IGV is showing you. You can also clarify issues or confusions on Canvas. You can click and drag the alignment panel to pan laft to right, scroll to go up and down through the rows of reads (called the pileup), and use the + and - buttons to zoom.

Messing with our reference to learn more

Next we are going to make changes to our reference sequence and then remap our reads and view them in IGV. By making changes manually and then seeing what the result is in our data I hope that you will get a clear understanding of what the data are telling you when you look at alignments of real data. While we won't cover the biological meaning of the various changes we will make, you will at least understand, for example, what various cigar string characters tell you about how the reads mapped to a reference.

As we go through the different modifications we make here keep in mind that your assignment this week is going to involve writing code to extract the base in a read that maps to a specified position in the reference sequence. For each modification think about how the change we made to the reference will impact which position in the read corresponds to the reference sequence. The reason I am showing you how to use IGV is so that you can pick a couple of reads mapped to modified reference sequences and manually check the positions so that you understand how to use the CIGAR string to figure out exactly where in the reference each base in a read maps.

Deleting a base in our reference

Let's start by deleting a base in our reference. I will delete the 50th base (which is a T), but you can delete a different base if you prefer. Let's also change a base as well so we can see that stripe from top to bottom that I mentioned above. I'll change base 30 from T to C, but again you can change a different base if you like.

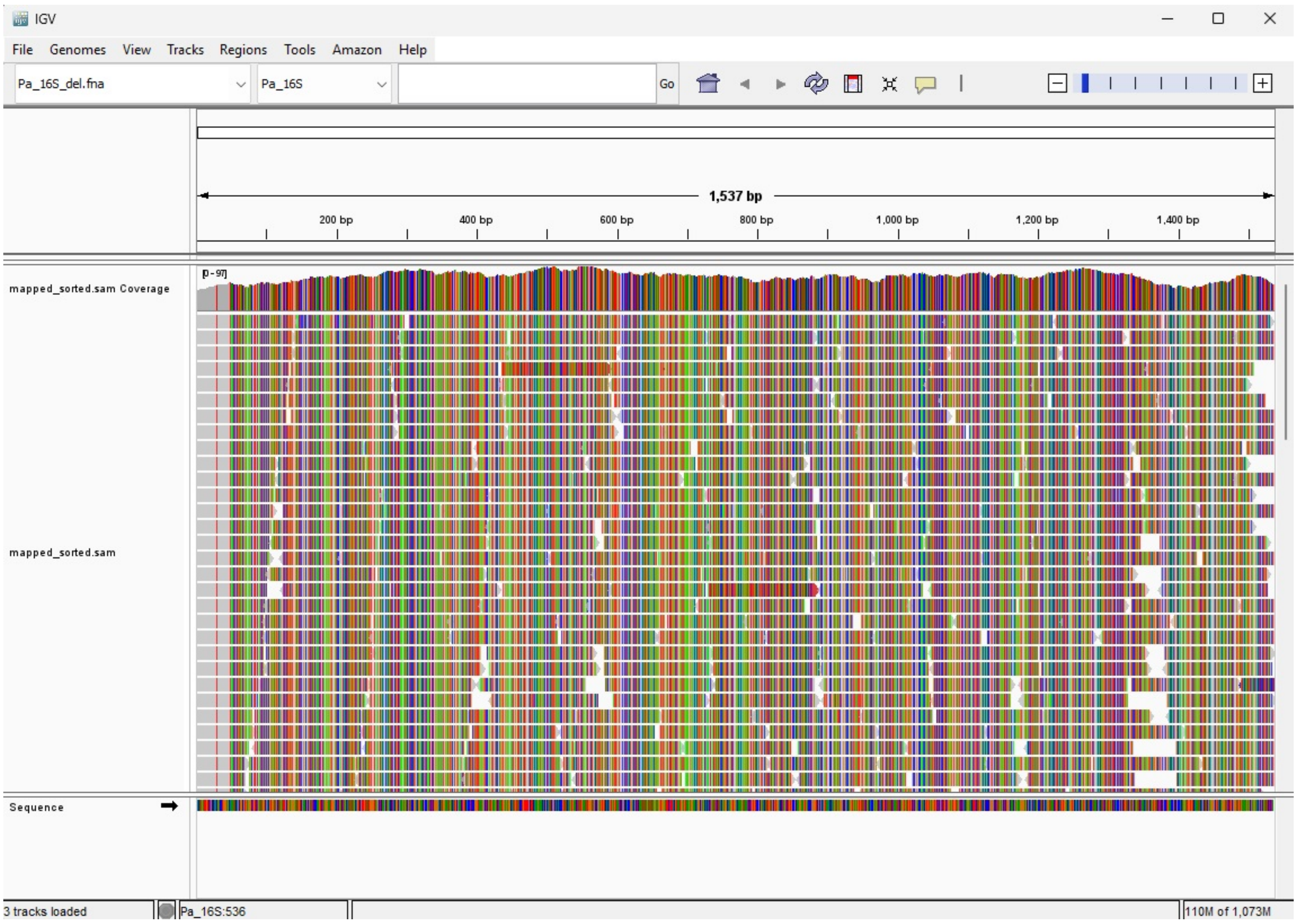
I won't provide my modified reference here as I want you to go in and make the changes yourself if you are following along. That is an important part of connecting the changes we are making to the results we see.

Now let's remap. You may have noticed earlier that I was writing SAM files from the output of `samtools` and `minimap2` using `>` redirection. In fact, `samtools` can read and write to stdout and stdin so we can do all the mapping, filtering, and sorting in a single pipeline. I'll name my modified fasta file Pa_16S_del.fna and then process with the following command:

```
minimap2 -ax sr Pa_16S_del.fna pa_reads/sub_SRR21376282_1.fastq
pa_reads/sub_SRR21376282_2.fastq | samtools view -h -F 4 | samtools sort -O SAM >
del_mapped.sam
```

Now we can view the mappings in IGV. You can either open a new IGV session or open the new "genome" and SAM file in the window you were already using.

Before we get to the mapping data of our reads against the deletion, I want to quickly show you a mistake you are likely to make. We didn't (or at least I didn't) change the fasta header of the new reference. That means that the SAM file of our original mapping and the new SAM file have the same sequence name in the reference column. IGV does not attempt any kind of check to confirm the alignment is correct before showing it to you. It just shows what the SAM file tells it was the alignment. That means if you ever accidentally load reads that we mapped to a different reference than the one you have loaded and the two references (the one you mapped to and the one you accidentally opened in IGV) have the same sequence IDs, you will see something like this:



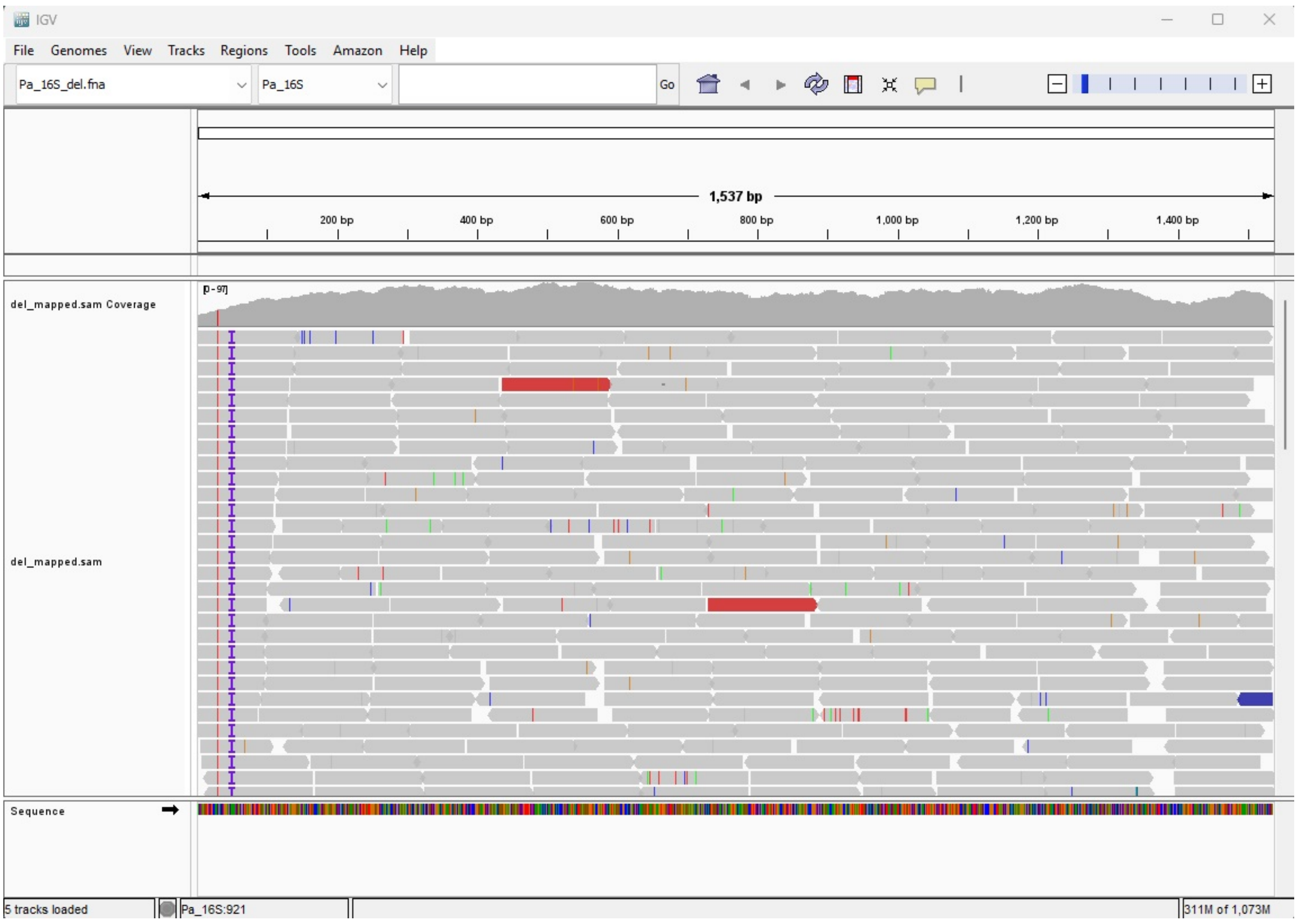
If you ever see a wall of rainbow where almost every single column is a stripe of color it always means that you have not loaded the correct combination of reference and SAM (i.e., the reference that the SAM file shows read mappings for).

The reason we see this wall of rainbow in the above image is because the cigar string tells IGV that there is an insertion in the read so the correspondence of read and reference after that insertion is no longer correct. The POS value is also impacted by the deletion in the reference, so reads mapped to the right of that point are shifted 1 base relative to the reference that is shown. IGV doesn't confirm that the alignment in the cigar string make sense. It just shows it and colors the bases according to its own rules.

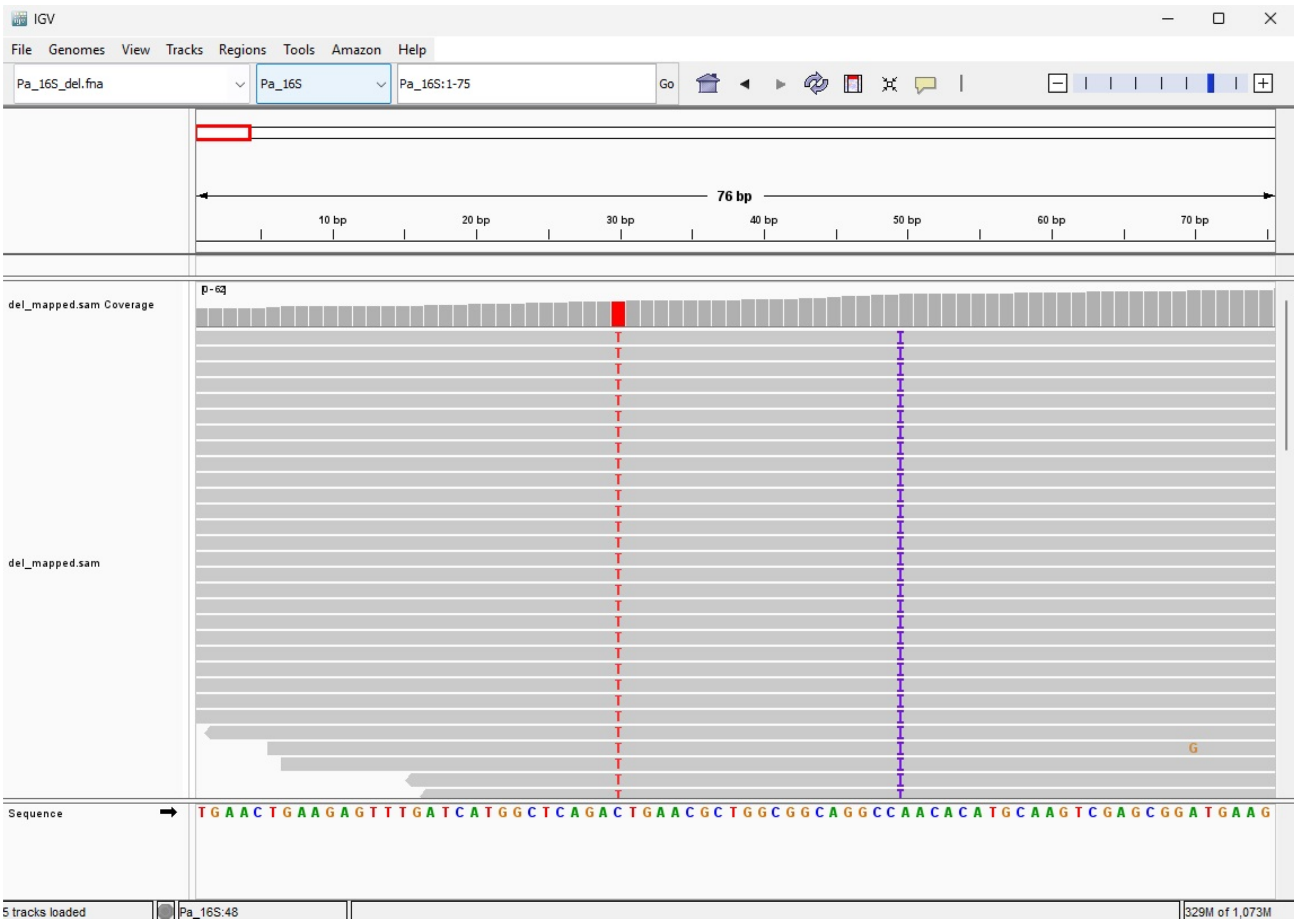
In other cases you can see this because there is zero correspondence between the reads and the reference. If you use an assembly from something like unicycler, which assigns numbers starting at 1 as the entire contig header (i.e., the whole fasta header is just ">1"), then mapping reads to two unicycler assemblies can produce SAM files that would be viewable in IGV to see this effect.

Now to the actual data.

Once you load the SAM file for the reads mapped to the reference with a deletion you should immediately spot the difference from the earlier mapping. Now there is a stripe of color and a stripe of Is from top to bottom in our pileup.

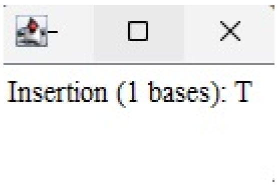


Let's zoom in and have a closer look at those stripes



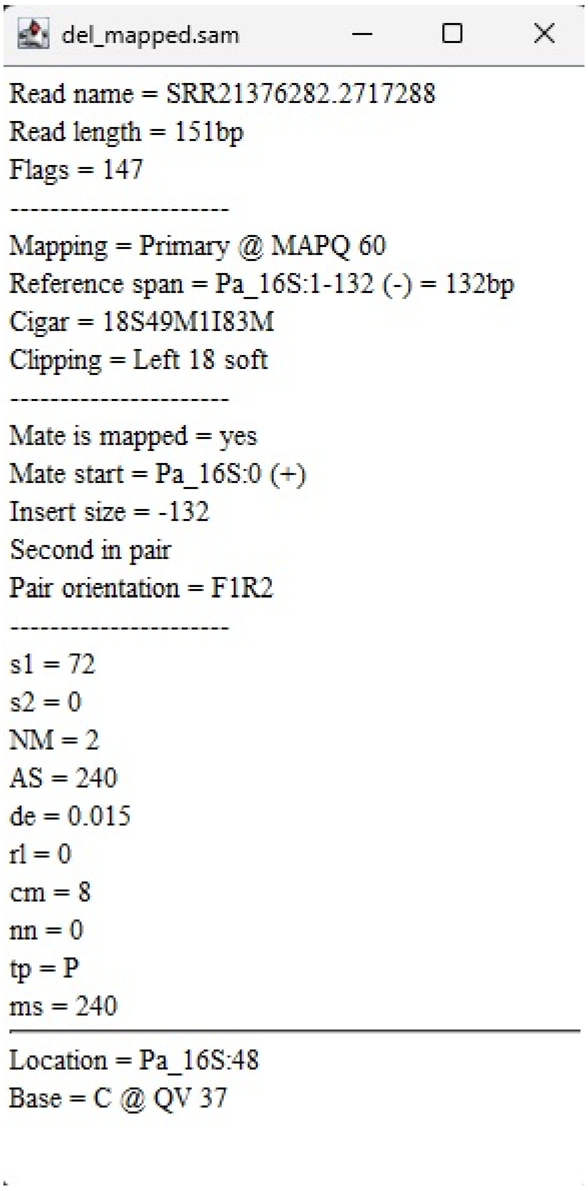
At this zoom level we can see that the depth bar plot has a red bar at position 30 and that all of the reads have a T there. That T corresponds to the position where I changed a T in the reference to a C. This sort of pattern (a vertical strip of the same base in every read) is what it looks like when there is a real SNP (single nucleotide polymorphism, i.e., different base in two sequences) between your reads and your reference.

We can also see a vertical strip of purple Is. This corresponds to the deletion I made in the reference sequence. If we click on one of those Is we can see what IGV is showing us.



```
Insertion (1 bases): T
```

So the I stands for insertion. That makes sense. However, IGV is calling this an insertion when what I did is ***delete*** a base in the reference. If we click a read with an I in it at a location away from the I we can see that the I shown by IGV corresponds to an I in the CIGAR string.



```
del_mapped.sam

Read name = SRR21376282.2717288
Read length = 151bp
Flags = 147
----------------------
Mapping = Primary @ MAPQ 60
Reference span = Pa_16S:1-132 (-) = 132bp
Cigar = 18S49M1I83M
Clipping = Left 18 soft
----------------------
Mate is mapped = yes
Mate start = Pa_16S:0 (+)
Insert size = -132
Second in pair
Pair orientation = F1R2
----------------------
s1 = 72
s2 = 0
NM = 2
AS = 240
de = 0.015
rl = 0
cm = 8
nn = 0
tp = P
ms = 240
Location = Pa_16S:48
Base = C @ QV 37
```

So the read I clicked has a CIGAR string of 18S49M1I83M which means 18 soft clipped bases, then 43 matching bases, 1 inserted base, and finally 83 matching bases. If we head our SAM file we will see that other alignments have similar CIGAR strings. Below I'll just grep to remove the header and cut the CIGAR column in the interest of space

```
$ head del_mapped.sam | grep -v "^@" | cut -f6
42S49M1I59M
61S49M1I40M
52S49M1I49M
47S49M1I54M
24S49M1I77M
```

You might find it confusing that the CIGAR string has an I when we deleted a base in the reference. However, by deleting the base in the reference, we have created a situation in which there is an extra base in the reads compared to the reference. If we didn't know that there was a base missing in the reference then we wouldn't know whether this extra base was an insertion in the reads (or rather the DNA they came from) or a deletion in the reference. Often, this lack of knowledge of what happened is handled by calling gaps "indels", which is just insertion and deletion merged into one word. In the case of SAM CIGAR strings, indels are simply reported with respect to the contents of the read. So if there is an extra base in the reads that is missing in the reference, then the base is said to be an insertion in the reads.

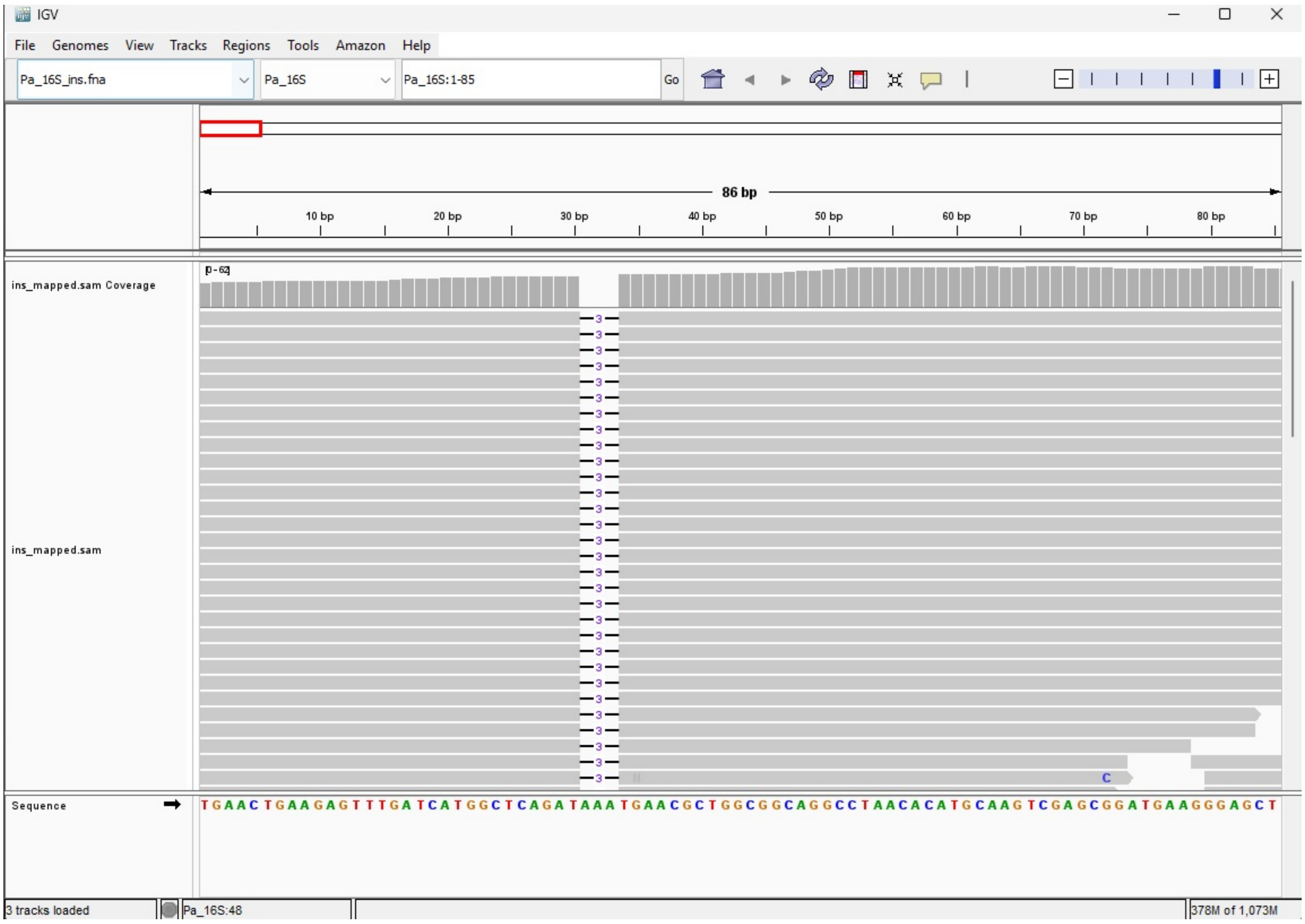
Inserting bases into our reference

Given what we just saw when we deleted a base in our reference, you might already have a good idea what you will see when we insert a base into the reference. You're probably right, but we'll go through the exercise anyway. Remember to keep in mind how these modifications would impact the correspondence of each base in the reads to positions in the reference.

I'll modify our original 16S sequence to have an extra AAA after the 30th base this time and save that as Pa_16S_ins.fna. I'm making these modifications near the start of the file so that it changes CIGAR strings in the first alignments in the SAM file so we can see the modifications in the head of the SAM file, but you could make your changes elsewhere if you prefer.

```
minimap2 -ax sr Pa_16S_ins.fna pa_reads/sub_SRR21376282_1.fastq
pa_reads/sub_SRR21376282_2.fastq | samtools view -h -F 4 | samtools sort -O SAM >
ins_mapped.sam
```

Again zooming in on the left of the reference you can see something different this time. Instead of Is there are dashes with a number in between. This is how IGV shows the presence and size of a deletion in your reads.



That deletion corresponds with the 3D in CIGAR strings of reads mapped to that location

```
$ head ins_mapped.sam | grep -v "^@" | cut -f6
82S30M3D39M
42S30M3D79M
61S30M3D60M
81S30M3D40M
52S30M3D69M
```

A hint to help you think about how to handle these in your code: looking at that IGV display and one of the first reads in the SAM file (which has the 3D in its cigar string), what would you return if asked for the base in the read that corresponds to the 31st position in the reference? What about position 32, 33, and 34? Which positions do those correspond to in the read? i.e., how would a deletion in the reference change which base position in the read corresponds to a position in the reference? How does that differ from the correspondence if there are no deletions or insertions (this is the crux of the assignment method `base_at_pos`).

Reversing part of our reference

That's it for the CIGAR string characters. You now hopefully have a good idea of what each one tells you about how your reads align to your reference. Now let's consider two FLAG bits that probably aren't clear yet: secondary and supplemental. We'll look at supplemental first.

Whenever you see a supplemental alignment there will also be a primary alignment for the same read. What supplemental means is that most of a read maps somewhere, but a smaller part of that read also maps somewhere else. We are going to create a situation where that happens by reversing the middle of our reference sequence. For the mappings we've done so far we haven't had any supplemental alignments. We can check by counting the number of alignments with 2048 in their FLAG value. You can either filter the SAM file and pipe to `wc -l` or `samtools view` has a `-c` option to report the number of reads.

```
$ samtools view -f 2048 -c mapped_sorted.sam # read count with 2048 in flag
0
$ samtools view -F 2048 -c mapped_sorted.sam # read count without 2048 in flag
832
```

I'm not sure if there's a fancy and simple Bash one-liner to invert part of a DNA sequence. Instead I'm just going to use `seqtk` to extract the first 200bp, then extract and reverse complement the next 1000bp and then extract the last 300bp in three steps in a code block and manually merge them. You can follow along with my commands or come up with something yourself. If you can do it in a clearer way let me know!

```
$ { seqtk subseq Pa_16S.fna <(echo -e "Pa_16S\t0\t200"); seqtk subseq Pa_16S.fna
<(echo -e "Pa_16S\t200\t1200") | seqtk seq -r; seqtk subseq Pa_16S.fna <(echo -e
"Pa_16S\t1200\t1537"); } > all_seqs.fna
$ echo ">Pa_16S" > Pa_16S_rev.fna
$ grep -v ">" all_seqs.fna | tr -d "\n" >> Pa_16S_rev.fna # tr -d to clean up the
line breaks
$ echo >> Pa_16S_rev.fna # to add in a final line break
```

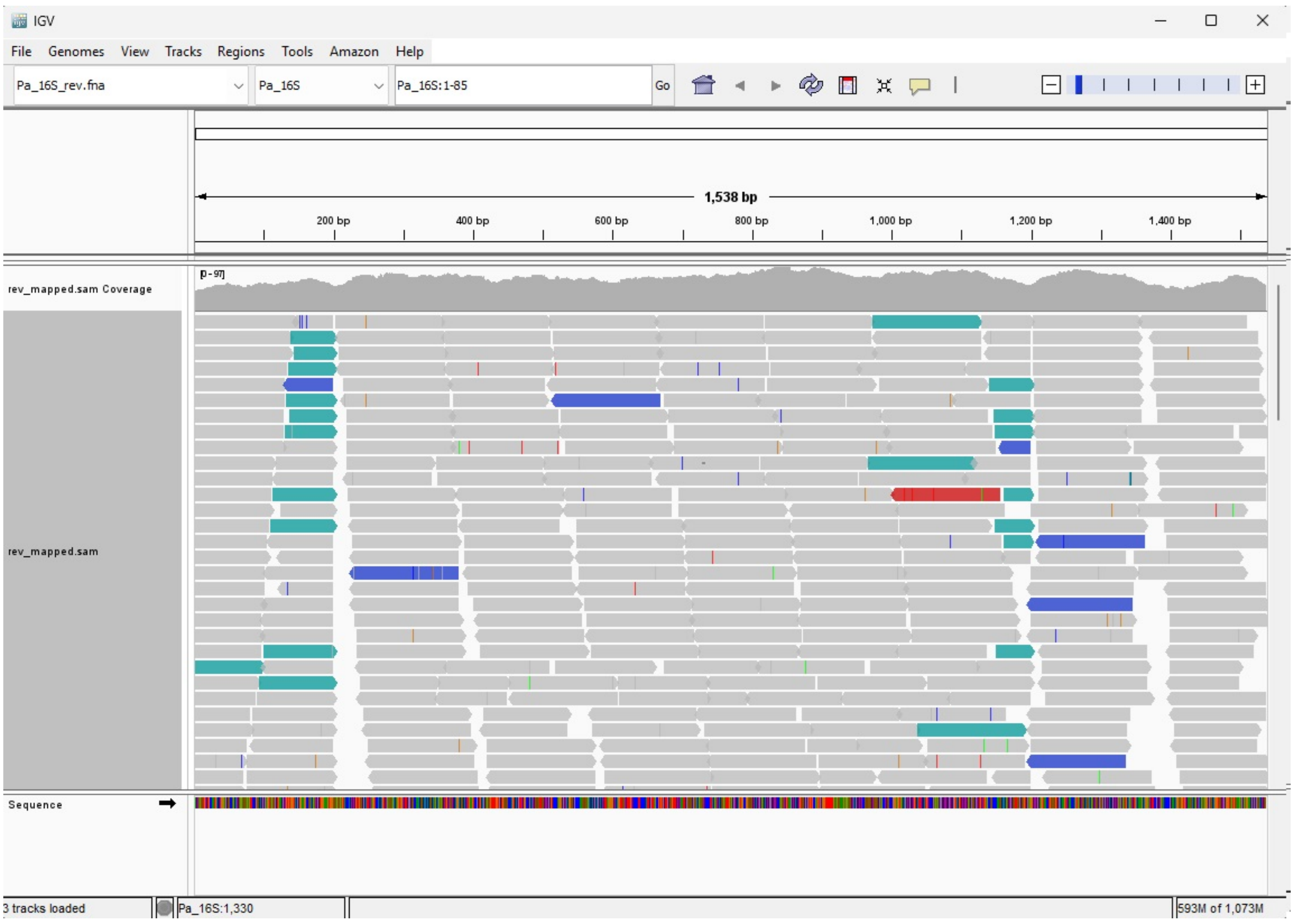
While fasta format tolerates weird line breaks, IGV complains if they are present so I added in that `tr -d` and lone `echo` above to tidy up the output file.

```
minimap2 -ax sr Pa_16S_rev.fna pa_reads/sub_SRR21376282_1.fastq
pa_reads/sub_SRR21376282_2.fastq | samtools view -h -F 4 | samtools sort -O SAM >
rev_mapped.sam
```

Now we should have some supplemental mappings. Drum roll...

```
$ samtools view -f 2048 -c rev_mapped.sam
76
```

Great! Let's take a look in IGV and see if we can spot them.

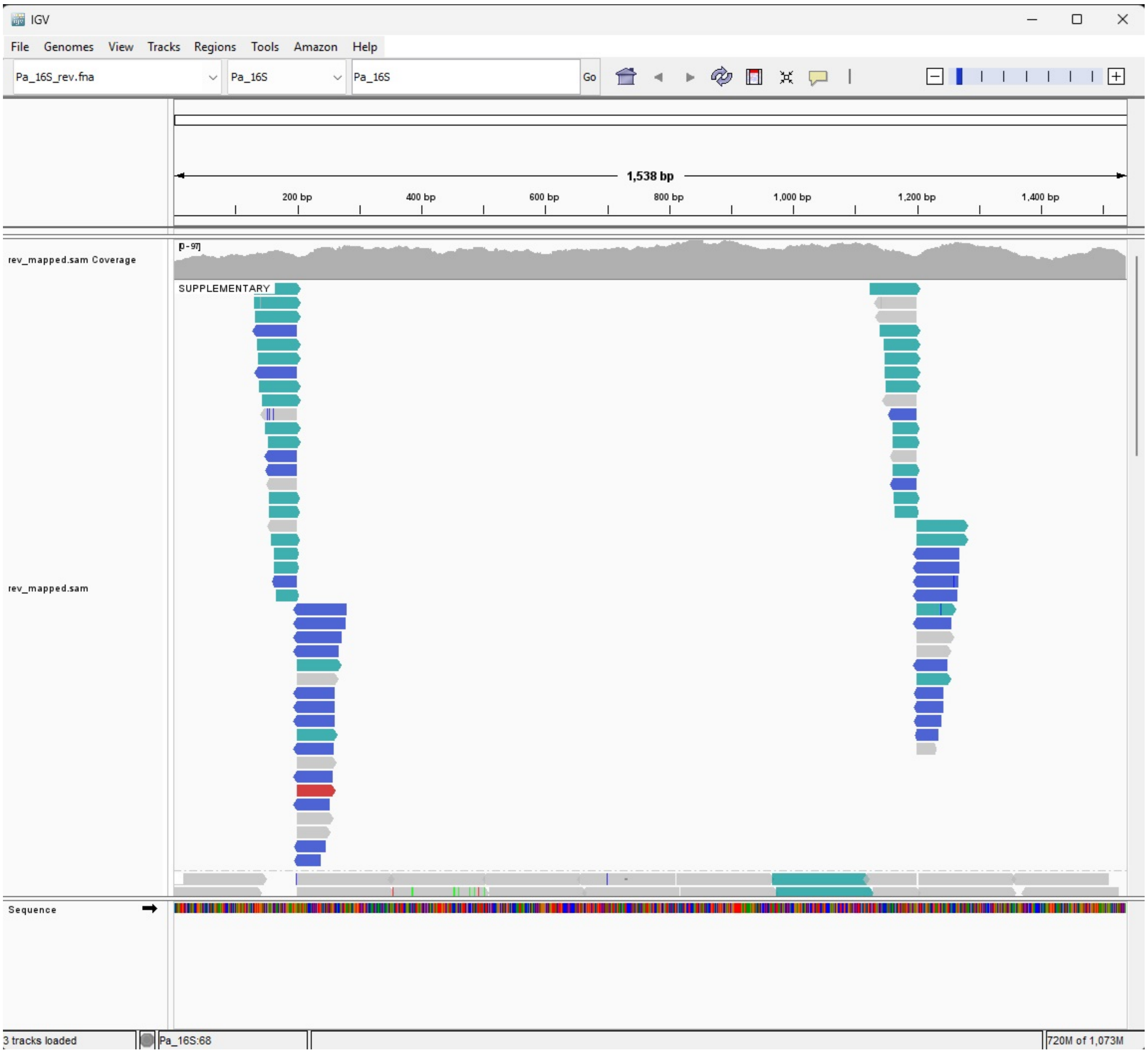


You can see a couple of big differences between this pileup and the previous ones we've been looking at. First is the teal (blue-green) arrows. Many of those are the supplemental mappings, but really that color means that both mates of a pair of reads map in the same direction (which is unexpected if the reads were from sequencing towards the middle of a DNA fragment from each end, but what you see when an inversion has occurred). You'll also see a wall of read-ends at 200 and 1200. What I mean by that is that if you put your cursor at the gap between the two reads that in the top row that meet at 200 and then scroll down to the bottom of the pileup, you'll see that there is a gap there all the way down. No reads map across the position 200 to 201 boundary. The same is true at position 1200.

This read mapping pattern is diagnostic of an inversion or some other structural difference in the DNA between the reference and the reads. Structural variants are a common situation in which you will see supplemental read mappings because we reversed a large portion of the reference. We therefore moved what used to be position 201 in the reference to position 1199. A read that used to map from 150-230 now maps from 150 up to position 200 and from 1169 to 1199 as well.

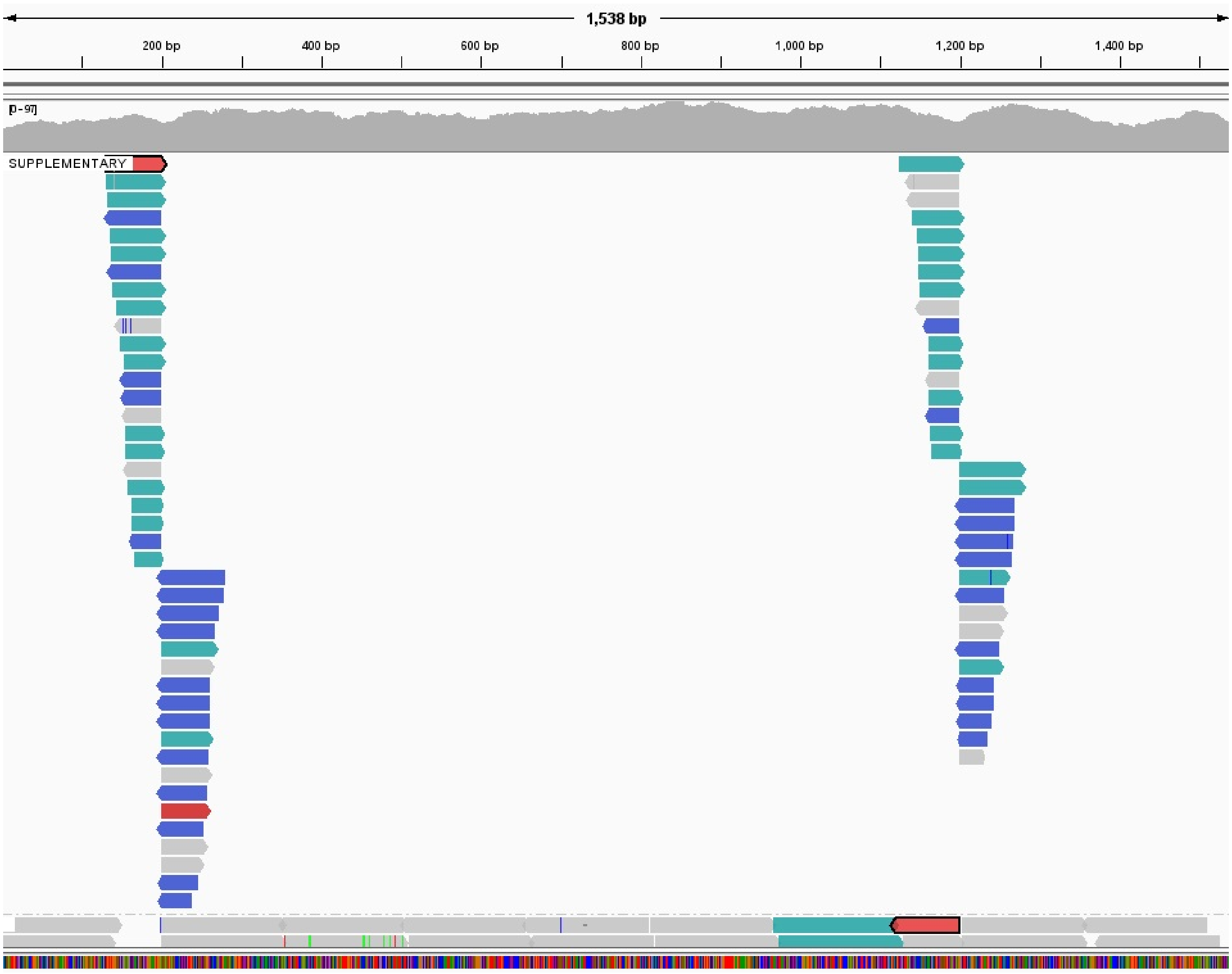
When mapping software finds that different parts of the same read could map to different locations, the longer or higher identity (fewest mismatches with reference) alignment is called the primary alignment and the shorter or lower identity alignment is called the supplemental alignment.

It's quite hard to tell that that is what is happening in the above image. However, if you right click in the alignment view, click "Group alignments by" -> "supplementary flag" then you will see all the supplementary alignments at the top of the pileup and then the primary alignments below.



Next you can see where the primary alignments for each are mapped. If you click on a supplementary alignment, you will see a "Supplementary Alignments" section in the popup. That tells you where the primary alignment is for the same read. This section have the same name if you click on the primary mapping, which is quite confusing. However, this section tells you where the other mapping is so if you click on primary it points you to the supplemental.

If you don't want to have to go hunting for the primary alignment, you can also right click an alignment and click "Go to Mate". For normal (primary only) reads, this just highlights the two mates in a pair of reads. For reads with supplemental alignments it highlights both the two mates in the pair and any supplemental alignments. The highlighting is making all the reads red with a black border. Clicking this will scroll your window weirdly. You can zoom a bit to reset the view to span the whole reference and you should then see the highlights. Below I'm showing the first supplemental read in the pileup and its primary mapping in one window. The mate is below out of view.



If you get sick of looking for reads in IGV you can also easily grep for read names in the SAM file to see all the alignments for the pair of reads with a certain ID. Generally, command line exploration of SAM files is quicker once you've got the hang of what the SAM data is telling you.

If you look at the CIGAR strings of supplemental alignments, you'll see that they have hard clipping, while their primary mappings have soft clipping. Both soft and hard clipping means that the clipped portion of the read isn't part of the alignment being described by this line in the SAM file, but have a look at a line in the SAM file with hard clipping and you'll see that hard clipped bases are not included in the SEQ column. That is an important thing to note when thinking about how to extract the base call at a certain position in a read with soft clipping vs hard clipping.

Duplicating our reference

Finally, let's force `minimap2` to give us some secondary mapped reads. Secondary mapping correspond to a flag value of 256 and are basically another place that a read could map that is less good than the primary mapping. These are distinct from supplemental alignments because supplemental aligments don't overlap their primary alignment. Secondary alignments can be the entire read.

Given that secondary alignments are just worse alignments of the whole read, we can modify our reference to cause those alignments by simply duplicating the whole thing and introducing some mismatches. I did it using a quick python script to replace every 50th base in the second copy with its complement base with the following script:

```
with open("Pa_16S.fna") as fin:
    seq=fin.read().strip()

rev_comp = {
    "A":"T",
    "T":"A",
    "C":"G",
    "G":"C"
}

new_seq = []
for i, base in enumerate(seq.split()[1]):
    if i % 50 != 0:
        new_seq.append(base)
    else:
        new_seq.append(rev_comp[base])

outseq = seq + "".join(new_seq) + "\n"

with open("Pa_16S_dup.fna", "w") as fout:
    fout.write(outseq)
```

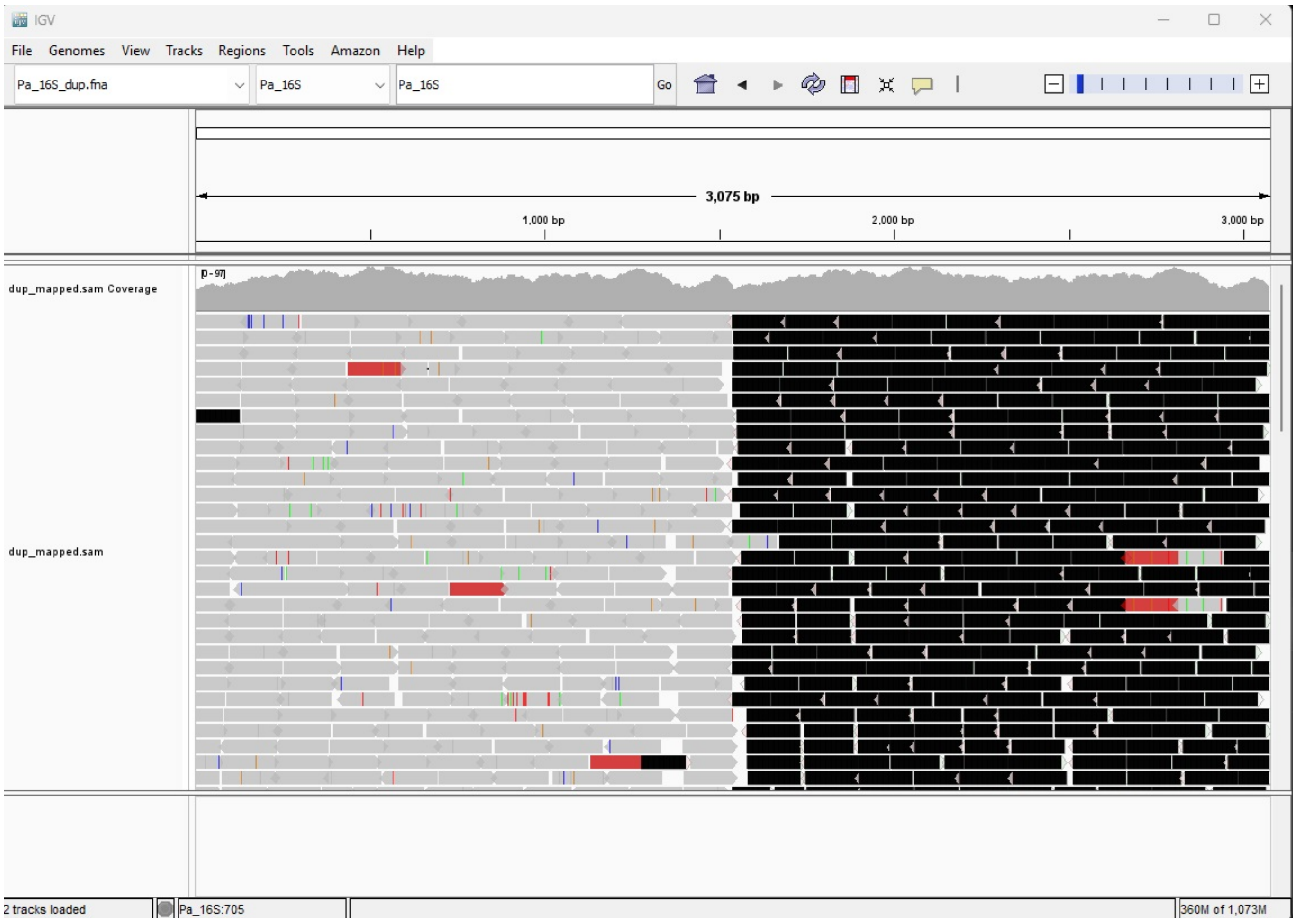
We then map again with a slight modification to what we've used before: `minimap2` won't output secondary alignments for Illumina data unless you ask it to with `--secondary=yes` so we need to add that option.

```
$ minimap2 -ax sr --secondary=yes Pa_16S_dup.fna pa_reads/sub_SRR21376282_1.fastq
pa_reads/sub_SRR21376282_2.fastq | samtools view -h -F 4 | samtools sort -O SAM >
dup_mapped.sam
```

We then get a lot of secondary alignments

```
$ samtools view -f 256 -c dup_mapped.sam
821
```

These are pretty obvious in IGV



Secondary alignments indicate that some sequence is present in two, non-identical copies. If the copies were identical then you would instead see two primary alignments for each read where the mapping score is 0 (because the mapping software can't choose which one to use). These non-identical copies of sequence can be gene duplications or things like that so if you are specifically looking for those you might want to look for secondary alignments. Otherwise, it is quite common to filter them out or to never include them in the mapping in the first place (`minimap2` doesn't include them unless asked to).