A Computational Method of Bias Reduction in Allele Specific Expression Analysis

Reducing biases in RNA sequencing methods and mapping tools

Alyssa Ricketts, Joyce Zhou, Leo Brusa,
Sam Wolfson, Steven Bishop

A final report for the course
CSE 428 - Computational Biology Capstone

Paul G. Allen School of Computer Science and Engineering
University of Washington
June 2019

1 Abstract

Motivation: Identifying genes for which two alleles are expressed at different rates is important when studying differences in the human population. Allele-specific expression (ASE) analysis is the process of measuring the relative abundance of transcribed RNA in a cell line in order to approximate the level of expression of each allele. A common method of ASE analysis is to count the frequency of different bases at each single-nucleotide polymorphism (SNP) position.

Problem statement: Variant detection in RNA-Seq data is compromised by biased mapping of reads to the reference DNA sequence, as well as the skewing of relative RNA abundance due to sequencing variation dependence on the surrounding nucleotide sequences.

Approach: Our approach aims to correct for sequencing bias through the use of existing tools, i.e. `seqbias`, and correct for mapping bias by performing bias correction using four variants of a reference genome.

Results: We were able to significantly increase the R^2 correlation coefficient between reference and alternate read count of SNPs on the same gene in our bias-corrected data. On chromosome 20, correcting for bias increased R^2 by 0.297, and on chromosome 1, correcting for bias increased R^2 by 0.017.

Conclusions: We were able to successfully reduce bias in human RNA-Seq data on chromosome 20. Our results showed that the bias-corrected data had increased consistency of allele expression in a single gene. However, due to time constraints, we have not yet replicated these results on other chromosomes. This is a possible avenue for future work.

2 Introduction

While every human has up to two variants, or alleles, of each gene, these alleles are not necessarily expressed to the same extent. It is possible that one allele is completely inactive, one allele is expressed to a much greater extent than the other, or both alleles are expressed at roughly the same extent. There has been a great deal of interest in identifying genes for which the two alleles in an individual are expressed at different rates [4]. In order to understand human population differences, it is useful to characterize the functional variation within a single individual, such as differences of expression between alleles [10].

Allele-specific expression (ASE) analysis is the process of measuring the relative abundance of transcribed RNA in a cell line in order to approximate the level of expression of each allele. Typical ASE analysis seeks to capture the relative expression of a given transcript by counting the RNA-Seq reads carrying reference and alternate alleles over heterozygous single-nucleotide polymorphism (SNP) sites [2]. This is done by mapping RNA sequenced reads to a genome, counting the number of reads within a gene that match a reference sequence, and comparing this to the number of reads within the same gene that do not match the sequence. However, variant detection in RNA-Seq data is compromised by biased mapping of reads to the reference DNA sequence, as well as the skewing of relative RNA abundance due to sequencing variation dependence on the surrounding nucleotide sequences [6, 11].

2.1 Sources of Bias

In an ideal experiment, the number of RNA-Seq reads mapping to a position in the genome is simply a function of the RNA abundance [6]. However, during the sequencing process, some portions of the genome are covered more frequently than others due to the surrounding nucleotide sequence. This source of bias is called **sequencing bias**.

After sequencing the reads, a tool is used to map the reads to a reference genome. The reads map to the portion of the genome with the closest match to the read. However, as reads can be in either the forward or reverse direction and can contain SNPs, it is likely that reads will map differently when using different references of an organism. This source of bias is called **mapping bias**.

2.2 Reducing Bias

There are a number of methods available to reduce the bias in ASE analysis. To reduce sequencing bias, the package `seqbias`, created by Daniel Jones, can be used to create vectors that estimate the relative amount of bias present in each read start position in a set of reads. It uses a simple Bayesian network trained on a set of aligned reads and a reference sequence [6]. These values are used to adjust the counts that are recorded for each read in the downstream analysis.

To reduce mapping bias, reads can be mapped to four variations of a reference genome: the forward reference (the original), the reverse reference (the reference's reverse strand), the forward alternate (the original but including SNPs from the reads), and the reverse alternate (the alternate's reverse strand). The distribution of mapped locations varies between these different reference genomes. This reduces the mapping bias because it allows reads not containing reference SNPs to be mapped more accurately.

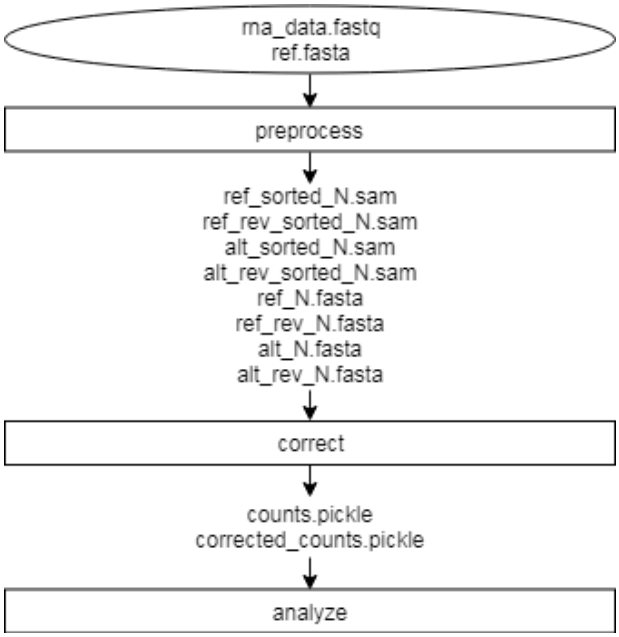

Figure 1: High-level view of the pipeline

3 Methods

The overall pipeline is summarized as follows (see **Appendix**). First, a human genome reference sequence and RNA-Seq data is obtained. In this instance, the human genome is obtained from the `GATK` resource bundle [5], and RNA-Seq data is obtained from the Sequence Read Archive [12]. This data is then run through the preprocessing pipeline, where a reverse reference sequence, alternate sequence, and reverse alternate sequence are created, and reads are mapped to each of these, creating four `SAM` files. Next, these `FASTA` sequence files and `SAM` read files are passed as inputs to `seqbias`, which exports a sequencing bias value for each read start position. Then, our `readcounter` script is run twice, once counting the number of reference and alternate SNPs seen in the reads when correcting for sequencing bias, and once counting the number of reference and alternate SNPs seen in the reads when not correcting for sequencing bias. Finally, the corrected and uncorrected counts are used to do a manual analysis of the effect of bias correction.

3.1 Preprocessing

The pipeline begins by preprocessing the data. The first step in the preprocessing is generation of the reverse strand of the reference sequence using the script `generate_reverse_genome` that we developed. This script reverses each chromosome, substituting the matching base pair at each index. We use `HISAT2` to map our reads to both the forward and reverse reference, resulting in an unsorted `SAM` file. We then use `samtools` to sort these reads. The sorted reads and reference sequence are then passed through `bcftools` to find SNP locations and store them in a `VCF` file. The `VCF` and the reference sequence are then passed through `GATK` to generate an alternate sequence from these reads. The sorting and mapping process is repeated for mapping reads to the alternate and reverse alternate sequences. Finally, these mapped reads and sequences are filtered for a specific chromosome of interest.

3.2 `seqbias`

`seqbias` is a package designed to assess and adjust for technical bias in RNA sequencing datasets [6]. It takes a reference `FASTA` and an RNA-Seq `BAM` file and trains a Bayesian network classifier on the provided aligned reads. It then uses this to evaluate the per-position sequencing bias, based on the surrounding

sequence. The model considers user-specified intervals, which can be modified to cover various positions in the genome or chromosome of interest. `seqbias` then exports the intervals that were covered, and a vector of sequencing bias values for each read start position in the corresponding intervals. For our purposes, we use one interval with a length of the chromosome to ensure coverage of each position.

3.3 `readcounter`

`readcounter` is the primary tool we wrote and its function is to count the number of reads from each allele. The tool's design is based off of `allelecounter` [2], a similar script; however our implementation includes the ability to factor bias values into the counts. There are two ways to run `readcounter`: without sequencing bias correction and with sequencing bias correction. Both methods fundamentally solve the same problem of counting reference and alternate SNPs found in the provided reads, but more preprocessing and internal computation must be done in order to correct for bias. Therefore, we examine each independently.

Regardless of method, `readcounter` requires at least three inputs: a reference sequence in `FASTA` format, the mapped, sorted RNA-Seq reads in `SAM` format, and finally a `VCF` containing SNP positions within the chromosome. `readcounter` must then parse the input files to produce a human-readable format before the reads can be counted.

`readcounter` begins by first parsing the `FASTA` index file corresponding to the reference sequence. It maps each chromosome to its start position in the file (in bytes), the number of bases per line, and the number of bytes per line. This allows for easy access to different read start positions within the sequence without reading the entire sequence into memory.

The second step in `readcounter` is to extract the SNP positions stored in the `VCF`. This maps SNP positions on each chromosome to a list of possible alternate SNP bases, ignoring any indels.

Finally, before any counting begins, `readcounter` parses the RNA-Seq reads stored in the `SAM`, returning a list of objects containing the read start position, the chromosome onto which the read was mapped, the parsed read in a pseudo-`PILEUP` object, a flag indicating whether the read contained any SNPs that matched the alternate, and a set of SNP positions covered by the read. It is important to note that reads that did not cross any SNP positions were ignored, and that it is possible for a read to cross a SNP position but not contain any SNPs.

pseudo-PILEUP

```
'start':50465726
'chrom':'chr20'
'seq':'...a..'
'alt':1
'snps':(50465729)
```

Figure 2: Example of a single line in the pseudo-PILEUP object.

The pseudo-`PILEUP` object is inspired by the `PILEUP` format used by `samtools` [8], where matches to the forward reference are stored as a "`.`", matches to the reverse reference are stored as a "`,`", and matches to the alternates are stored as their bases in upper or lower case to indicate a forward or reverse sequence match, respectively. Our format differs in that mismatches are stored as "`*`", which are later ignored in the counting process, and that no symbol is needed to denote the start and end of a read, given that each object contains a single read.

3.3.1 Without Bias Correction

When run without bias corection, `readcounter` tallies up the number of reads that match the reference sequence and the number of reads that match the alternate sequence, for each SNP position. It does so by iterating over the pseudo-`PILEUP` objects. For each read, it first checks if that read is from the alternate allele, indicated by the "alt" flag provided in the pseudo-`PILEUP` object. It then iterates over each position in the read, and checks if the current position is listed in the set of SNP positions, again provided in the pseudo-`PILEUP` object. If the current position is not a SNP position, it is ignored and `readcounter` moves on to the next position in the read. If the current position is a SNP position, and the read is from the alternate, then `readcounter` increments the count at this position on the alternate allele by one. If the current position is a SNP position, and the read is not from the alternate, then `readcounter` increments the count at this position on the reference allele by one. `readcounter` then returns the count of reference reads and alternate reads at each SNP location.

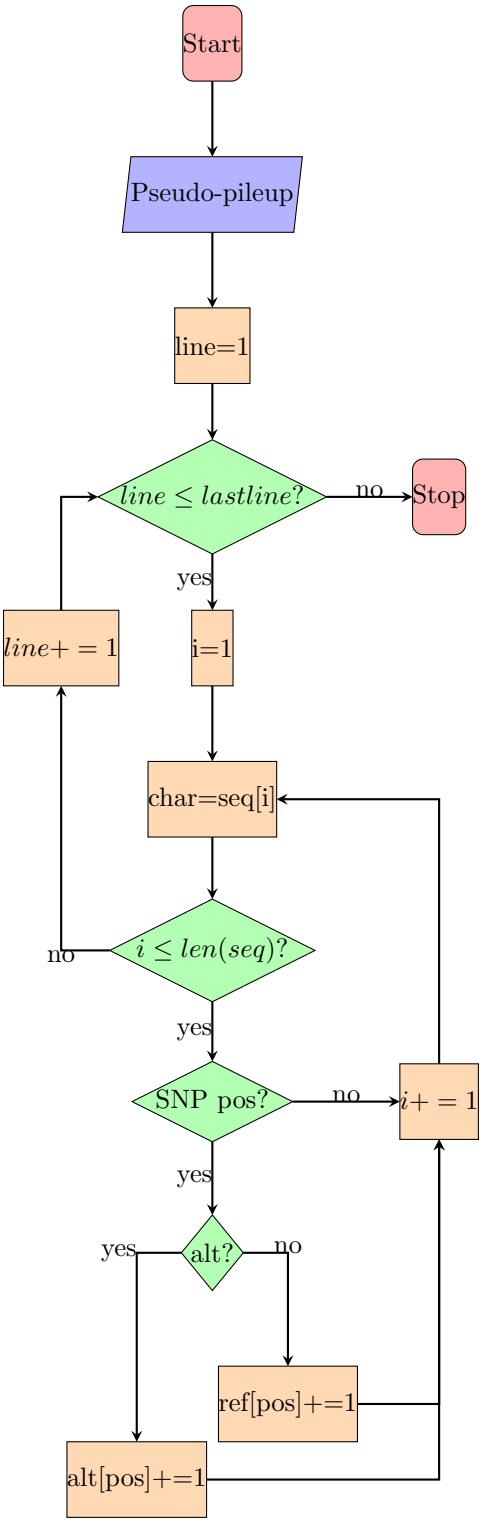

Figure 3: Logic for counting reference and alternate SNPs without correcting for sequencing bias.

3.3.2 With Bias Correction

When run with correcting for sequencing bias, `readcounter` has 11 inputs: the same three that it takes without bias correction, as well as the outputs from `seqbias`, namely, four bias value vectors and four interval vectors corresponding to the aforementioned bias vectors. The first steps `readcounter` takes with bias correction are the same as without, including parsing the `FASTA`, `VCF`, and `SAM`. When it is time to count reference vs. alternate SNP positions, this is where the two methods differ.

From the four bias value vectors and four interval vectors, each position in the intervals is mapped to its corresponding bias value. This is possible as each interval in the intervals vector provides a start and end position of the interval, which corresponds to a single column in the bias value vectors. `readcounter` iterates over each row in the intervals file and each column in the bias values file. It uses the start position provided from the interval, and then maps each position, from start to end, to each bias value in the current bias values column, in order. This process is completed four times, once for each of the bias value and interval inputs, creating four separate dictionaries mapping read start position to bias value: forward reference dictionary, reverse reference dictionary, forward alternate dictionary, and reverse alternate dictionary.

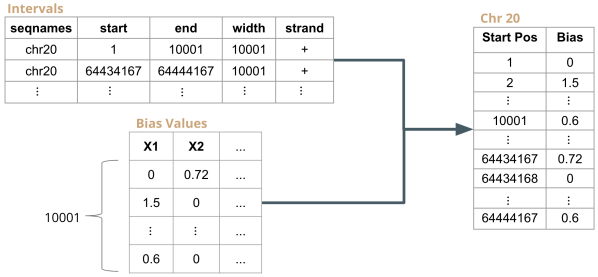

Figure 4: Translation from seqbias output to internal representation

The bias values for each read start position are then used to count the number of occurrences of each allele seen in the reads. This is again done by iterating over the pseudo-`PILEUP` objects. For each read, it first checks if that read is from the alternate allele, indicated by the "`alt`" flag provided in the pseudo-`PILEUP` object. Next, it obtains the first character in the sequence provided by the pseudo-`PILEUP` object, which corresponds to the read start position. If the read is from the alternate, and the read start position character is '`A`', '`C`', '`G`', '`T`', or '`.`', this indi-

cates the read start position nucleotide matched the alternate sequence on a forward read. `readcounter` then indexes into the forward alternate dictionary, and obtains the bias value that corresponds to this read start position. If the read is from the alternate, and the read start position character is '`a`', '`c`', '`g`', '`t`', or '`,`', this indicates the read start position nucleotide matched the alternate sequence on a reverse read. `readcounter` then indexes into the reverse alternate dictionary, and obtains the bias value that corresponds to this read start position. If the read is from the reference, indicated by the "`alt`" flag being false, and the read start position character is a '`.`', this indicates the read start position nucleotide matched the reference sequence on a forward read. `readcounter` then indexes into the forward reference dictionary, and obtains the bias value that corresponds to this read start position. If the read is from the reference and the read start position character is a '`,`', this indicates the read start position nucleotide matched the reference sequence on a reverse read. `readcounter` then indexes into the reverse reference dictionary, and obtains the bias value that corresponds to this read start position.

The abundance of a region can be more accurately assessed by normalizing the read count values, therefore dividing each position by its predicted bias [6]. Thus, we calculate the normalized count value by dividing 1 by the bias value for the start position, obtained from the dictionary:

$$\text{normalized count} = \frac{1}{\text{bias value}}$$

A bias value of 0 is unexpected as each position is assigned a value if at least one read begins at this position, and we only use read start positions when calculating the normalized count value. However, this is possible due to sequencing read errors or mapping errors, which can cause a read start to not map to any position. Thus, a bias value of 0 for a read start position is unusual but does occur occasionally. In these situations, instead of dividing by 0 to obtain an undefined normalized count value, we simply ignore reads that have a read start position bias value of 0.

`readcounter` then iterates over each position in the read, and checks if the current position is listed in the set of SNP positions, again provided in the pseudo-`PILEUP` object. If it is not a SNP position, it is ignored and moves on to the next position in the read. If it is a SNP position, and the read is from the alternate, then it increments the count at this position on the alternate allele by the normalized count value for the read start position. If it is a SNP position, and the read is not from the alternate, then it increments the count at this position on the reference allele by the normalized count value for the read start position. Thus, each SNP position in a given read is incremented by the normalized count value calculated for only the read start position. It then returns the count of reference reads and alternate reads at each SNP location.

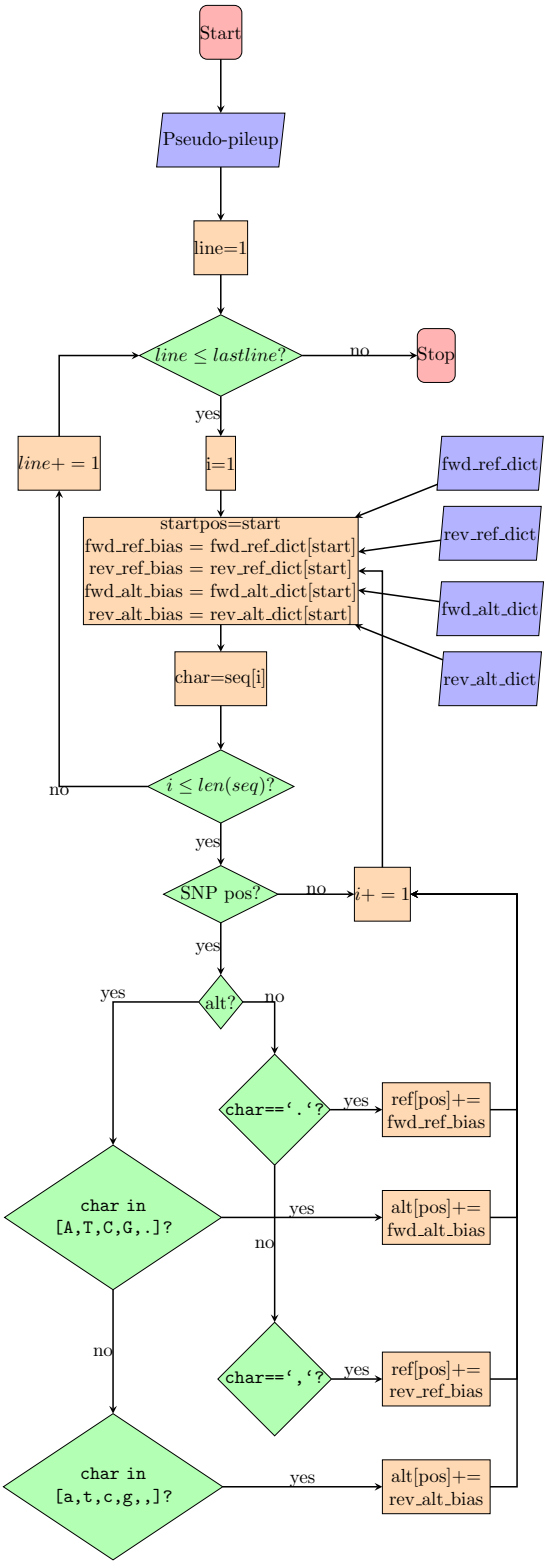

Figure 5: Logic for counting reference and alternate SNPs when correcting for sequencing bias.

3.3.3 Output

The `readcounter` was first run without providing the "bias" flag as input, indicating the script should simply count the total number of times reference and alternate alleles are seen by incrementing the count for each SNP position by 1 each time a read covers it. The readcounter was then run a second time, with the "bias" flag provided, indicating that the normalized count values should be used.

The counts of reference and alternate alleles were grouped by chromosome, and then identified by position within a chromosome. A Python class called `GenomeSNP` was implemented to keep track of these counts. After running `readcounter`, we used the Python `pickle` library to write the completed `GenomeSNP` object containing the reference and alternate counts to a file for use in analysis later on in the pipeline.

The tool Ensembl [14] and the corresponding Python library `pyensembl` were then used to match up SNP positions with the genes that contain them and to annotate the `GenomeSNP` class with gene information for each SNP.

So, our final processed data looks like this:

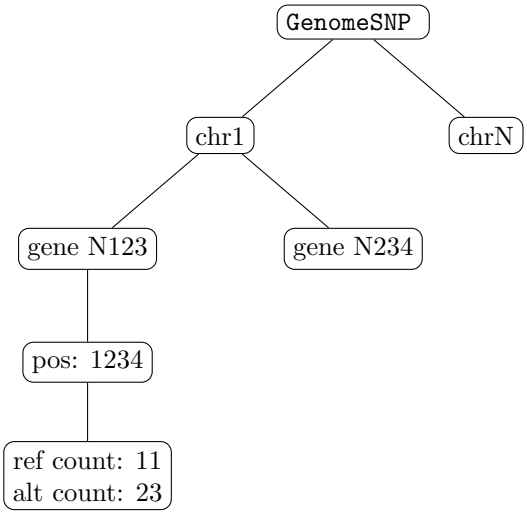

Figure 6: Breakdown of the `GenomeSNP` class

(where each chromosome contains genes, and each gene has multiple positions, each of which contains a count of the base from the reference and from the alternate). Recall that two of these objects are generated: one from the uncorrected read data, and one from the bias-corrected read data.

3.4 Analysis

We hypothesized that, within a single gene, the ratio of alternate count to reference count should be consistent across all SNP positions in the gene; that is, the linear correlation between the reference counts of SNPs and alternate counts of SNPs should be high within a gene. We used an R^2 test of linear regression to determine whether the bias correction improved the linearity of the data, and graphed the improvements. We also provide graphs of the reference and alternate counts for one gene, to provide background for the type of improvement that we hypothesized.

4 Data

We used two data sources for the analysis. The first is the human reference genome, which is the hg38 assembly in `FASTA` format [5]. This is the most up to date standard reference genome and is commonly used in computational biology studies.

The next data source comes from a human RNA sequencing study on the SRA database. The particular study is the GSM484895: GM12878_RNA-Seq_Rep1 (SRR038448) study, in which RNA was sequenced using Illumina to study the binding sites of RNA Polymerase II in humans of different ancestry [12]. The data that was taken from this experiment were the raw, unmapped reads in `FASTQ` format. The cells from the study's RNA sequencing contained no treatment.

5 Results

We ran our pipeline on chromosome 20, as it was the smallest chromosome with good sequencing data. This allowed us to run our tool in reasonable time. We saw meaningful improvement in the consistency of reads across a single gene in our corrected data. We used the R^2 metric described in the Analysis section, and we present our results in table 1 and figure 9. On average, we obtained a 0.293 increase in R^2.

To better understand what this R^2 metric means, consider just the gene TMSB4XP6 on chromosome 20. The uncorrected counts are shown in figure 7, and the corrected counters are shown in figure 8. Note that the linear regression line much more closely represents the corrected data (and the R^2 value is correspondingly higher).

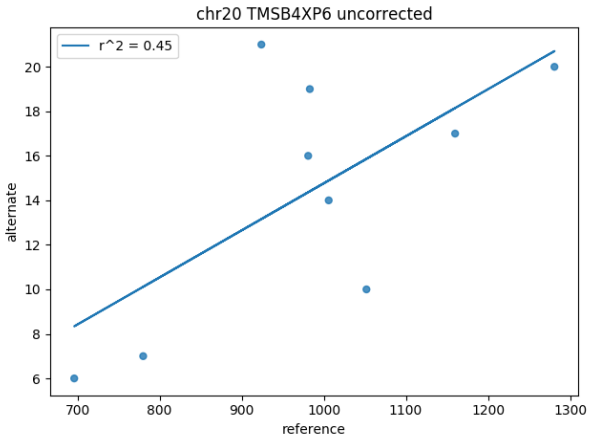

Figure 7: Uncorrected counts for chromosome 20, gene TMSB4XP6

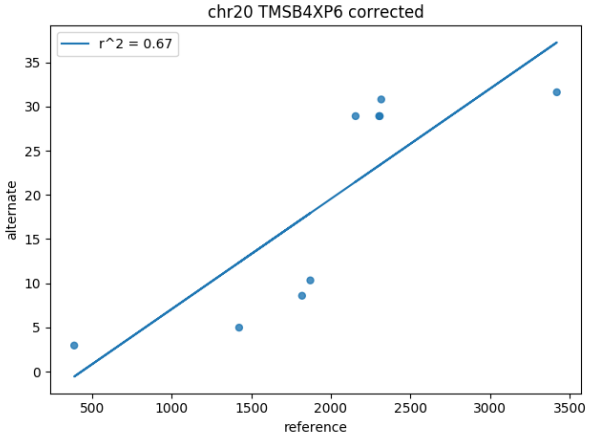

Figure 8: Corrected counts for chromosome 20, gene TMSB4XP6

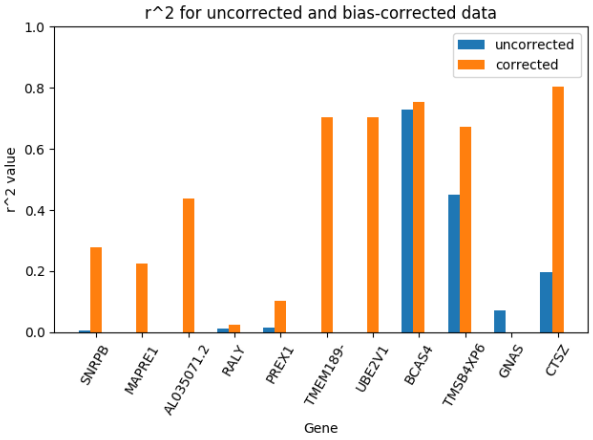

Figure 9: R^2 values for uncorrected and corrected counts on chromosome 20.

Gene	Number of SNPS	Uncorrected R^2	Corrected R^2	ΔR^2
SNRPB	5	0.006	0.278	0.272
MAPRE1	7	0.000	0.226	0.226
AL035071.2	6	0.000	0.437	0.437
RALY	7	0.012	0.026	0.014
PREX1	9	0.014	0.103	0.089
TMEM189-UBE2V1	5	0.000	0.703	0.703
UBE2V1	5	0.000	0.703	0.703
BCAS4	13	0.728	0.755	0.027
TMSB4XP6	9	0.451	0.672	0.221
GNAS	20	0.073	0.000	-0.073
CTSZ	7	0.198	0.805	0.607

Table 1: Uncorrected and corrected R^2 values for genes on chromosome 20.

We also performed the same analysis with genes on chromosome 1, and we present the results in table 2 and figure 10. We obtained similar results for many of the genes in chromosome 1: finding that the bias correction did improve the linear correlation between reference and alternate alleles within a gene. On average, we obtained a 0.017 increase in R^2. However, there were a few genes—for instance AL031281.2—where the bias correction actually caused significant *decrease* in the linear correlation. Due to time limitations we didn't have a chance to explore why this occurred, but it presents an opportunity for future study. This improvement in R^2 is also much more modest than the improvement we obtained on chromosome 20, which merits further investigation.

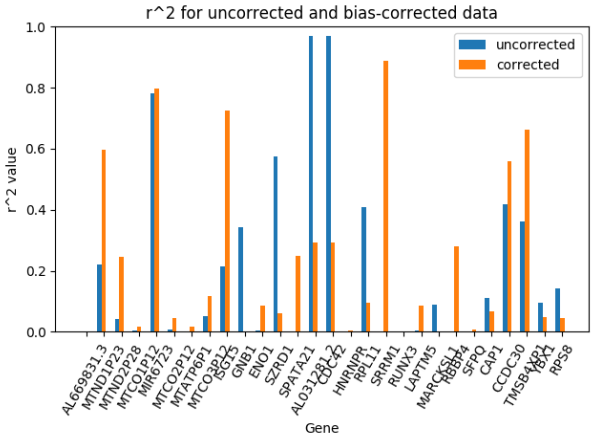

Figure 10: R^2 values for uncorrected and corrected counts on chromosome 1.

6 Discussion

This project was a great learning experience for all of us, and the scope of our project was quite appropriate. There are still many things that we wish to do, but outstanding problems really just come down to a lack of computing resources.

6.1 What We Learned

The first major thing that we learned was the technical skills required to work with DNA and RNA sequencing. This included learning many of the various file formats, namely `VCF`, `FASTA`, `FASTQ`, `SAM`, and `BAM`, as well as learning how to use many of the common tools used in sequencing analysis. Additionally, we worked directly with Daniel Jones, the author of `seqbias`, which was a major piece of our pipeline.

We also learned some biology, mainly sources of bias in ASE analysis, including mapping and sequencing bias.

6.1.1 Tools

Our first endeavor was into a package known as `GATK`, or the `Genome Analysis ToolKit`. Through poor documentation and fragile code, we managed to extract a few key tools that would be useful to our downstream analysis. However, in the end we decided that other toolkits would work better for most of our purposes and only ended up using the `FastaAlternateReferenceMaker` (which depends on `CreateSequenceDictionary`) to generate the alternate genome.

Gene	Number of SNPS	Uncorrected R^2	Corrected R^2	ΔR^2
AL669831.3	187	0.001	0.001	0.000
MTND1P23	10	0.222	0.597	0.375
MTND2P28	27	0.044	0.247	0.203
MTCO1P12	65	0.003	0.016	0.013
MIR6723	9	0.782	0.798	0.016
MTCO2P12	25	0.007	0.047	0.039
MTATP6P1	34	0.001	0.018	0.018
MTCO3P12	19	0.051	0.117	0.066
ISG15	5	0.214	0.725	0.510
GNB1	11	0.344	0.000	-0.343
ENO1	17	0.005	0.087	0.082
SZRD1	7	0.576	0.060	-0.516
SPATA21	5	0.000	0.249	0.249
AL031281.2	6	0.971	0.292	-0.679
CDC42	6	0.971	0.292	-0.679
HNRNPR	5	0.000	0.004	0.004
RPL11	11	0.409	0.094	-0.315
SRRM1	6	0.000	0.890	0.890
RUNX3	11	0.000	0.000	0.000
LAPTM5	23	0.003	0.086	0.083
MARCKSL1	8	0.089	0.002	-0.087
RBBP4	6	0.000	0.280	0.280
SFPQ	6	0.000	0.009	0.009
CAP1	7	0.111	0.066	-0.044
CCDC30	13	0.417	0.560	0.143
TMSB4XP1	7	0.363	0.662	0.300
YBX1	13	0.095	0.049	-0.047
RPS8	8	0.141	0.046	-0.095

Table 2: Uncorrected and corrected R^2 values for genes on chromosome 1.

One of these other toolkits we found is known as `samtools`, which is a very efficient and easy to use tool that mainly processes `SAM` and `BAM` files, but also includes useful tools to process `FASTA` files. We use this tool to sort our mapped reads, to index our `FASTA` files, and to convert back and forth from `SAM` and `BAM`.

A closely related toolkit to `samtools` is `bcftools`, which is mainly used to process `VCF` files. We only use it for one thing, and that is to create a `VCF` using `FASTA` and `SAM` files.

Additionally, we took a look at a toolkit called `HISAT2`, which is used to map reads to a reference genome. We use `HISAT2` to create an index of each genome and to map our reads to each of these indexed sequences.

Finally, Daniel Jones showed us how `seqbias` is able to reduce sequencing bias by altering the weight of each read start position in the specified interval. We learned that this is a very effective tool for reducing bias.

6.1.2 Sources of Bias

Although many of us came into the course with very little knowledge of biology, we were able to learn enough in order to come up with a solution for two of the major sources of bias in ASE analysis. The first of which is mapping bias, which seems pretty obvious based on our prior knowledge as computer scientists and engineers. However, what was significantly more subtle was the source of sequencing bias, which required quite a lot of research to understand. Our journey led us to the prior knowledge needed to more fully understand what the purpose of `seqbias` is to our pipeline, and how to apply it to the rest of our tools.

6.1.3 Dealing with Big Data

None of us had dealt with big data so much until we had to deal with tens of gigabytes of files that more or less had to be loaded into memory all at once if we wanted to be efficient. We ran into a Python error that most of us had never seen before, which was a `MemoryError`, saying that we had run out of memory. Usually memory is not an issue with programs nowadays (unless you're Google Chrome), so we had to deal with creating index files in order to more easily look up resources in the large files.

6.2 Future Steps

Although we were able to get some data fully through our pipeline, there was a lot more data and different ways of interpreting the data that would have been interesting to look at. This could have been made possible by optimizing our code. It currently takes around two hours to run `readcounter` with bias correction. This is not particularly ideal for when we want to figure out how to process data in a different way. Many places in our code could easily be parallelized, but then we would need to greatly reduce memory usage.

Additionally, we were pretty last minute getting the results we wanted, so not many of our stretch goals were looked at. This includes looking at reads that don't match the reference or the alternate sequences. Currently these are ignored as they are treated as read errors, but what would have been good to look at is if there were a lot of these that matched as this could be a sign of either a mapping error or neither of the alleles matched the reference or each other to begin with.

One more thing that we wanted to look at was to see if we could further reduce mapping bias. We take into account mapping bias by running `seqbias` on all four sequences, but if we were to also run `readcounter` with all four sequences then we would be able to reduce mapping bias there as well. However, it would have been difficult to interpret the results without further computational analysis.

Another thing that would be interesting would be to see how quality scores affect the bias values. Currently we ignore quality scores, but it would have been interesting to see if the places with lower quality scores were more likely to be amplified by the bias counts as there are probably very few of them relative to other reads.

Finally, we would like to look at relatively obvious genes, such as eye or hair color, and see if after we correct for bias if the result shows a noticeable more accurate representation of the person the reads came from.

7 Summary

Our results showed an increase in r^2 value, indicating a stronger correlation between the alternate and reference counts after bias correction (see **Results**). This matched our hypothesis that alternate and reference counts on the same gene would have the same ratio regardless of SNP position (see **Analysis**). Essentially, our results demonstrate that our tool partially corrects for bias as after bias correction, the ratios became more similar.

Briefly recall the two types of bias that are introduced into RNA-Seq data: sequencing bias, which is introduced due to certain portions of the genome being more likely to be covered than other portions, and mapping bias, which is introduced due to differential mapping based on reference genome.

Having used `seqbias` to create bias-correction vectors, we were able to fully correct for sequencing bias present in the RNA-Seq data by normalizing the bias value for each read. This accounts for the majority of our bias correction as, although we used all four sequences when correcting for bias in `readcounter`, the parsed `SAM` reads do not account for all possible mappings; all the reads only come from the forward reference mapped `SAM`. Therefore, mapping bias was only partially corrected (see **Discussion** for more details), which may have been one reason as to why the increase in r^2 value was not as large as we had expected.

Ultimately, by reducing bias in the RNA-Seq data, our tool can be used to perform allele-specific expression analysis more reliably than other tools, such as `allelecounter` [2], which do not account for any form of bias present in the input data.

8 Group Reflection

8.1 Organization

As a group, we discussed our goals at our weekly meetings and assigned tasks to team members in order to maintain a productive momentum. We all ini-

tially focused on learning background material, such as sources of bias in allele-specific expression and possible methods for reducing said bias. As the project moved along, we began filling specific roles: Steven focused on the preprocessing stage, Leo focused on parsing the RNA-Seq reads, Alyssa focused on the bulk of the bias correction, Sam focused on the plotting and analysis stage, and Joyce focused heavily on organizing our GitLab repository and pipeline.

In order to stay on track, we created a broad set of goals we planned to achieve by certain weeks, e.g. we had planned to identify allele-specific expression in generic datasets by our sixth week and fully run our entire pipeline with bias correction by our eighth week. Although we did not achieve these goals by our initial deadlines, having an organized structure helped us maintain a positive working environment.

8.2 Challenges

One of the largest challenges our group faced was working with the data. Our main issues lied in finding reasonable data to use for our project, understanding the various file formats commonly used in bioinformatics, and handling big datasets. Initially, we spent a substantial amount of time trying to understand the various files, their formats, what they represent, how they are used, and how they are related. These include `FASTA`, `FASTQ`, `VCF`, `SAM`, and `BAM` files. Upon understanding these file formats, we then faced the challenge of finding data that would be appropriate for our purposes.

After finding reasonable data to work with, we encountered challenges unique to working with large data. The data we worked with was on the order of magnitude of 10 GB. This required us to find an external source outside of GitLab to host our data; we ended up using the CSE Google Drive, and zipping our data files before uploading. Additionally, we iteratively needed to optimize our code when it ran unreasonably slowly, as well as removing large data structures such as dictionaries. One workaround we found to processing these large datasets was to use Steven's desktop at home, so we were able to run our scripts during the day and overnight without bogging down our personal laptops.

8.3 Mistakes

The biggest mistake we made in this project was trying to use `GATK` for the first three weeks. We found that there was a very problematic lack of documentation and the code was very fragile. Newlines at the end of files would cause the programs to crash after many minutes of running, which was pretty frustrating. However, we were able to find many other tools that did the same thing as `GATK`, but much more efficiently. Committing to `GATK` for too long was a major setback, but we were still able to learn about the different file types and what steps we needed to take.

Another mistake we made was using the wrong data. It turned out we were using "really good data for some other application", which resulted in us losing almost a day in processing time. We were able to quickly resolve this and after one more day we had good results.

A non-technical mistake we made was getting off to a slow start. Many of us didn't know where our project was going so we ended up spending the first few weeks not doing much. However, after we got rolling we were able to better manage our time and make some massive progress.

8.4 Implementation

Overall we are satisfied with our implementation choices. We used Python which made everything very easy to write, but we paid the price when running the code as Python is not the fastest language. This ended up being a slight problem, but overall wasn't too detrimental as the tools we already were using took a long time to run. We were able to start the scripts running overnight and they would be done running (or an error would manifest) by morning.

Additionally, we structured our code very modularly which made optimizations very easy to make. We had standard input and output formats that allowed us to work independently on different portions of our tool as we knew what to expect of each module. For instance, one of the major things that we were able to optimize was the parsing of a `FASTA` file. We initially were storing the entire sequences in memory, but later optimized this away by using the `FASTA` index file instead. The way we had structured our code made this very easy and we only had to change the one relevant function.

References

[1] Christopher G Bell and Stephan Beck. "Advances in the identification and analysis of allele-specific expression". In: *Genome Medicine* 1.5 (2009), p. 56. DOI: 10.1186/gm56.

[2] Stephane E. Castel et al. "Tools and best practices for data processing in allelic expression analysis". In: *Genome Biology* 16.1 (2015). DOI: 10.1186/s13059-015-0762-6.

[3] Ana Conesa et al. "A survey of best practices for RNA-seq data analysis". In: *Genome Biology* 17.1 (2016). DOI: 10.1186/s13059-016-1047-4.

[4] Jacob F. Degner et al. "Effect of read-mapping biases on detecting allele-specific expression from RNA-sequencing data". In: *Bioinformatics* 25.24 (2009), pp. 3207–3212. DOI: 10.1093/bioinformatics/btp579.

[5] Data Sciences Platform @ Broad Institute. *Resource Bundle Reference materials for human analysis*. URL: https://software.broadinstitute.org/gatk/download/bundle.

[6] Daniel C. Jones et al. "A new approach to bias correction in RNA-Seq". In: *Bioinformatics* 28.7 (2012), pp. 921–928. DOI: 10.1093/bioinformatics/bts055.

[7] Julian C Knight. "Allele-specific gene expression uncovered". In: *Trends in Genetics* 20.3 (2004), pp. 113–116. DOI: 10.1016/j.tig.2004.01.001.

[8] H. Li et al. "The Sequence Alignment/Map format and SAMtools". In: *Bioinformatics* 25.16 (2009), pp. 2078–2079. DOI: 10.1093/bioinformatics/btp352.

[9] Zhi Liu et al. "Comparing Computational Methods for Identification of Allele-Specific Expression based on Next Generation Sequencing Data". In: *Genetic Epidemiology* 38.7 (2014), pp. 591–598. DOI: 10.1002/gepi.21846.

[10] J. Rozowsky et al. "AlleleSeq: analysis of allele-specific expression and binding in a network framework". In: 7.1 (2014), pp. 522–522. DOI: 10.1038/msb.2011.54.

[11] Mazdak Salavati et al. "Elimination of reference mapping bias reveals robust immune related allele-specific expression in cross-bred sheep". In: (2019). DOI: 10.1101/619122.

[12] *Sequence Read Archive : NCBI/NLM/NIH*. URL: https://trace.ncbi.nlm.nih.gov/Traces/sra/?run=SRR038448.

[13] David L. A. Wood et al. "Recommendations for Accurate Resolution of Gene and Isoform Allele-Specific Expression in RNA-Seq Data". In: *Plos One* 10.5 (2015). DOI: 10.1371/journal.pone.0126911.

[14] Daniel R Zerbino et al. "Ensembl 2018". In: *Nucleic Acids Research* 46.D1 (2017), pp. D754–D761. DOI: 10.1093/nar/gkx1098.

Appendix

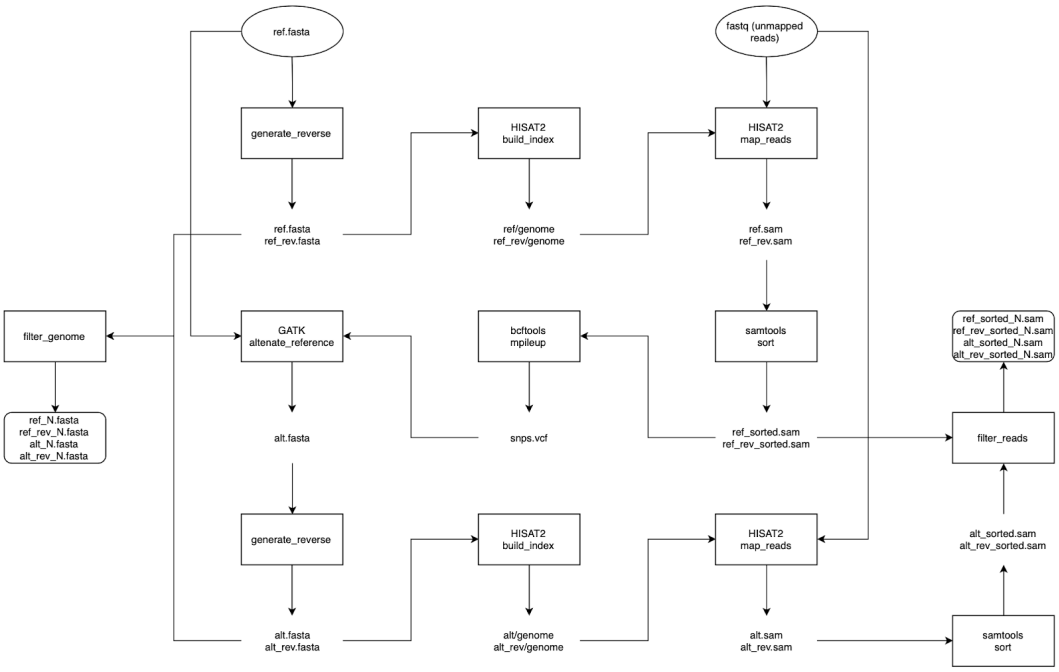

Figure 11: Preprocessing the data

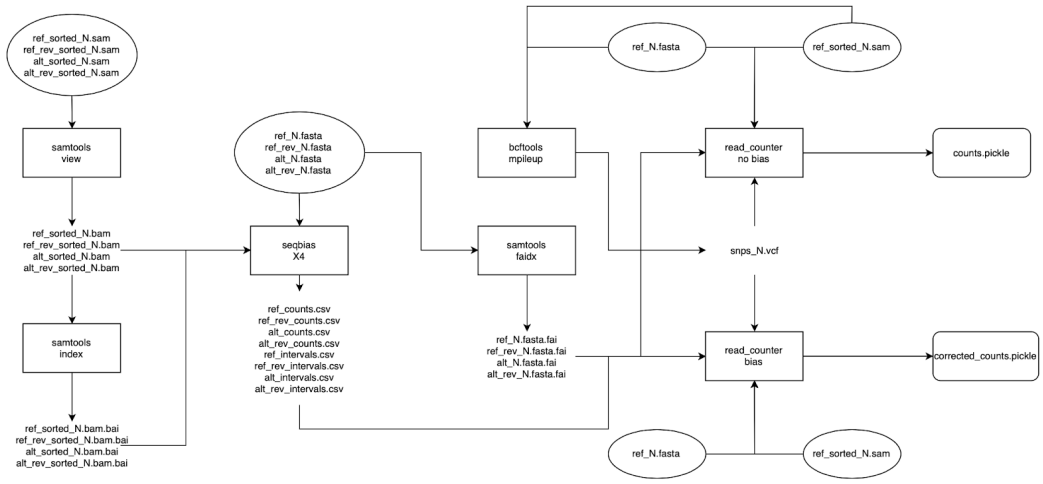

Figure 12: Reducing bias