CS CM121 Project 2:
Pseudoalignment Implementation

By Aydin Karatas

Due 25 Mar 2023

§1 Introduction

Given a transcriptome and RNA-seq reads, map each read to the transcriptome. We must identify the equivalence class of each read (i.e. possible transcripts the read originated from), find out how frequently each equivalence class appeared, and report the size (number of transcripts) of each equivalence class. Possible problems include taking into account the possibility of mapping to the reverse compliment and correcting for sequencing errors in the reads.

1.1 Zip File Contents

The provided zip file, `CM_CM121_Project_Code.zip`, is a folder of the following files.

- `project.py` and `project.R` contains the code for this project
- `chr11_transcriptome.fasta` and `reads.fasta.gz` are the inputs for `project.py`
- `ref_equiv_classes.csv` and `ref_statistics.txt` are the expected outputs of `project.R`

1.2 Usage Instructions

Input the following command in a UNIX-based terminal to produce the preliminary file `read_obs.csv`.

```
python3 project.py chr11_transcriptome.fasta reads.fasta.gz 31 > read_obs.csv
```

`project.py` creates a k-mer index from `chr11_transcriptome.fasta` and maps reads from `reads.fasta.gz` to the index. 31 designates the k-mer size. Input the following command in a UNIX-based terminal to produce the output table for this project, summary statistics, plots used in this report.

```
Rscript project.R read_obs.csv > statistics.txt
```

`equiv_classes.csv` is the output table for this project, containing information on equivalence class sizes and occurrences. `statistics.txt` contains summary statistics on equivalence classes for all reads, correctly-mapped reads, and incorrectly-mapped reads (the distriction between the the is discussed in **§3.2**). `all_noi.png` and `err_noi.png` are the plots used in this report.

§2 Methods

Below outlines the process for finding the equivalence class for each read and producing the file preliminary `read_obs.csv`. The **Results** section will discuss how all this data was analyzed to produce `equiv_classes.csv` and `statistics.txt`. All scripts were run through the Mac Terminal of a MacBook Air (M1, 2020). `read_obs.csv` was produced in less than 4 minutes.

2.1 Creating k-mer Index

Transcript names from the transcriptome file were stored in a hash table as keys; the values being the forward sequence (from the transcriptome file) and reverse complement sequence (generated from the forward sequence). The k-mer index is formed by storing the k-mers of all sequences as keys; the values were the transcripts that each k-mer was found in. k-mers that appear multiple times in one transcript only displayed that one transcript name once. See Figure 1 below for a visual pipeline. This process took around 1 minute.

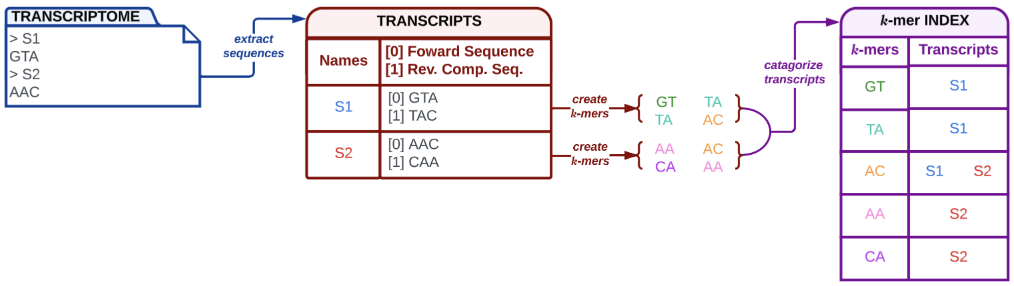


Figure 1: Process for creating k-mer dictionary with $k = 2$.

2.2 Hash Table Pseudoalignment

The kmers of a given read were create based on the k-mer length. Using the index made in **§2.1**, we attempt to find the set of transcripts associated with each read k-mer. Read k-mers that did not exist in the index were assigned a null set. To find the equivalence class, we took the intersect of the sets of transcripts for all k-mers. This process took around 12 seconds per 100,000 reads of size 100.

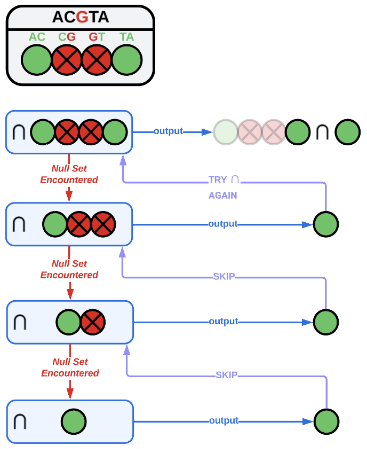
2.3 Handling Sequence Errors

Taking the intersect for a given read, as outlined in **§2.2**, often resulted in a null set. To exclude sequencing errors, we recursively found the shortest series of k-mers with a successful intersect; each recursive call removed the final k-mer until [1] finding success or [2] removing all k-mers.

We then wanted to skip all sequential k-mers that contain the perceived base error. In case [1], the next k number of sequential k-mers were skipped. We believe that the last base of the k-mer we just omitted was erroneous, so we will skip all k-mers containing this base. In case [2], the first k number of k-mers of the read were skipped because we do not know which base was erroneous.

The algorithm kept tracking of how many skips to make when returning to previous recursive calls. After the perceived-to-be-erroneous k-mers were skipped, the algorithm took the intersect between the current series of valid k-mers (this series would have no k-mers in case[2]) and the next k-mer after the skipped k-mers(this k-mers would automatically be the new valid series of k-mers case[2] if it is not a null set itself). Under case [1], we added the k-mer to the current series if the intersect was successful. Else, skip the k-mer in question and the next k-1 k-mers to initiate skipping errors again. This process of adding and skipping k-mers was repeated until completely returning from the first function call. See Figure 2 for a visual pipeline.

Figure 2: Backtracking and skipping example error G ($k = 2$).



§3 Results

The preliminary file `read_obs.csv` contained the equivalence classes for each read and was feed into `project.R`. The `R` portion of our pseudoalignment pipeline calculated the number of times a specific equivalence class was mapped to and the size of each unique equivalence class, outputting the file `equiv_classes.csv`, `statistics.txt`, and the plots shown below. This process took less than 7 seconds.

3.1 Summary Statistics

A total of 15,402 unique equivalence classes were identified. Here were the summary statistics for all equivalence class sizes and occurrences as found in `statistics.txt`:

	Min.	**1st Qu.**	**Median**	**Mean**	**3rd Qu.**	**Max.**
Size:	0.00	1.00	2.00	3.883	5.000	59.000
Occ.:	1.00	2.00	8.00	83.27	38.00	29881.00

Of the 1,282,526 reads in the data, a total of 726,812 reads (56.67%), mapped to an equivalence class of size 1; 166,431 reads (12.97%) mapped to an equivalence class of size 2; 51,782 reads (4.04%) mapped to an equivalence class of size 3. There were 29 cases ($2.26 * 10^{-5}\%$) of reads having a null set as its equivalence class. Figure 3 graphs the relationship between mapped reads and equivalence class size.

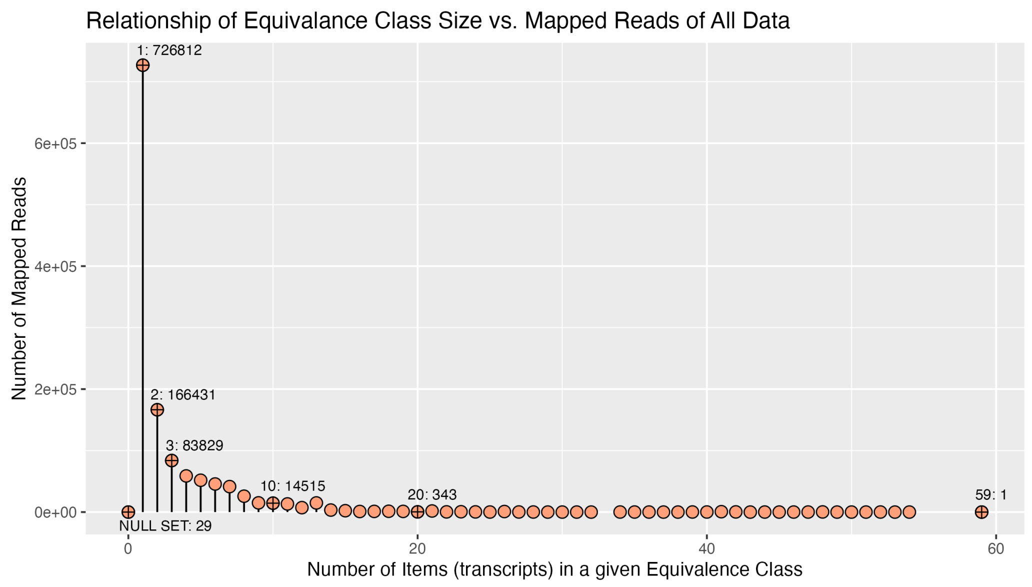


Figure 3: Size of equivalence classes produced from pseudoalignment compared to the number of mapped reads for an equivalence class size.

3.2 Limitations

The simulated reads data provided information on the true transcript for each read. When searching for the true transcript of each read in the equivalence class determined by the algorithm, we found that the algorithm had a success rate of 68.92%; 398,585 reads were, therefore, mapped incorrectly. Figure 4 is another graph of mapped reads against equivalence class size but for incorrect mappings.

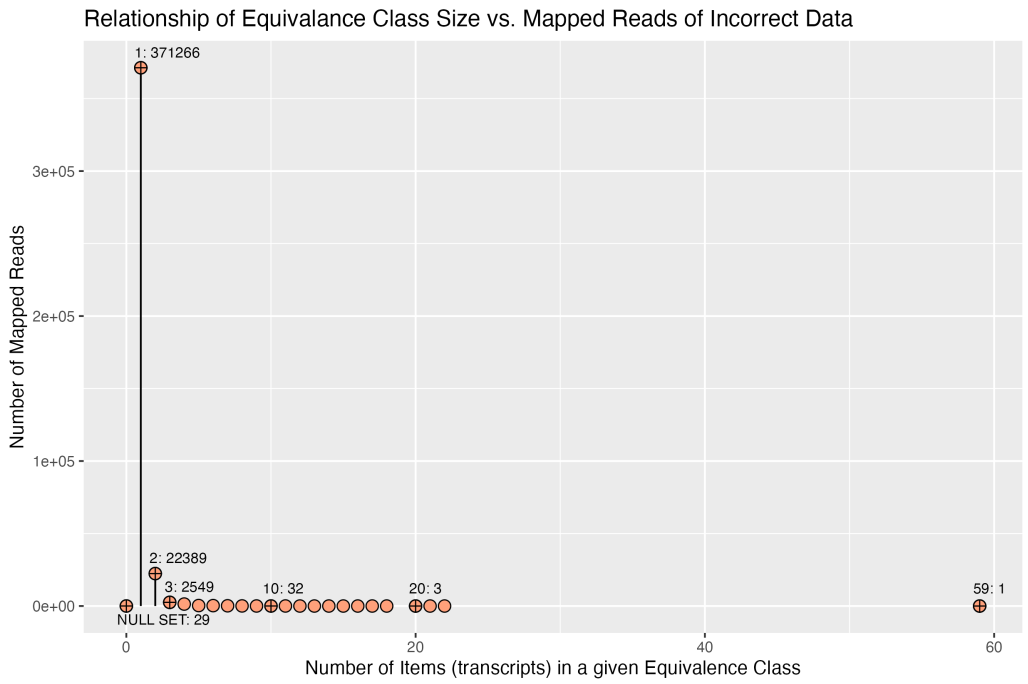


Figure 4: Equivalence class sizes of reads that mapped incorrectly.

It is here that we found a major limitation of this algorithm. Of the incorrect mappings, 371,266 reads (93.15%) mapped to an equivalence class of size 1. This result suggested that 51.08% of all reads that mapped to an equivalence class of size 1 were incorrect. This maybe due to sequencing errors that are neither null sets themselves nor result in a null set intersect. The algorithm could only determine the location of an possible error by checking for null set intersections. For example, `read 1` should map to transcript `ENST00000410108`, but instead mapped to `ENST00000382762`. Looking at the 70 sets of transcripts for `read 1`'s k-mer,

40 contained `ENST00000410108`;
47 contained `ENST00000382762`;
31 contained both transcripts.

It was possible that the sequences of these transcripts were similar enough to be found in multiple sets together. An error in `read 1` could cause it to look more like `ENST00000382762`, causing the backtracking correction to identify the wrong transcript. The algorithm was prone to error, with about a fifty-fifty change, when dealing with equivalence classes of size 1.