Using Hadoop to Identify False Positives in Bacterial Strain Typing from DNA Fingerprints

Colin C. Adams
Department of Computer Science
California Polytechnic State University
San Luis Obispo, CA 93407
Email: ccadams@calpoly.edu

***Abstract*—Pyroprinting is a novel technique used by the Department of Biological Sciences to obtain "fingerprints" from the DNA of *E. coli* isolates in order to categorize them into strains. To determine the number of false positives that occur in the pyroprinting process, isolates with the same pyroprints needed to be sequenced to see if their underlying alleles match. If they do match, this shows they are indeed the same strain and are a true positive. If the alleles don't match, they are different strains and are a false positive. To do this 100 isolates with nucleotide identifiers were sequenced. Over five million sequences were then analyzed using a program implemented on Hadoop. This program provided a general indicator of the efficacy of pyroprinting by grouping the sequences into their respective isolate buckets and analyzing them to determine which were false positives. The Hadoop implementation proved to be reliable and highly scalable. This method of analysis is generally applicable to many areas within bioinformatics, as well as potential uses in other industries. The results from the experiment are still being analyzed to determine the frequency of false positives, and how this can inform the use of pyroprinting.**

***Keywords*—Hadoop, Distributed Computing, Pyroprinting, Bacterial Strain Typing, E. coli, DNA**

I. Introduction

Pyroprinting is a method of strain typing for bacteria. It was developed by the Department of Biological Sciences at Cal Poly, and uses fingerprints from each bacterium's genetic material to organize them into strains. The DNA fingerprint for the bacterium is called its pyroprint. This technique was developed as a cheaper, faster alternative to other bacterial strain typing techniques. Bacterial strain typing is important as a means of identifying the sources of water contamination. First the bacteria, namely *E. coli*, in a body of water is analyzed to identify the strain. Then, the host species can be inferred from information on which species are likely to host that particular strain. This is used to determine which species are contributing most to water contamination [1].

To develop pyroprinting as a strain typing method, the Department of Biological Sciences has collected *E. coli* samples, referred to as isolates, from known host species and obtained the pyroprints (discussed in Section II-B). These pyroprints have been compared to each other to determine which isolates are a part of the same strain [1]. However, thus far there has been no way to identify if this method creates false positives, where isolates have the same pyroprints and are deemed the same strain but in the underlying DNA of the isolates are different. The objective of this experiment is to measure the frequency of false positives which would indicate the efficacy of this method, and the objective of this paper is to explain how Hadoop was used within the experiment [4].

The rest of this paper is organized as follows: Section II explains pyroprinting in depth and the overall structure of the experiment, Section III summarizes the algorithm for analysis using Hadoop, and Section IV analyzes the performance of our implementation.

II. Background

A. E. coli DNA Structure

Escherichia coli DNA consists of a single circular chromosome. As seen in Figure 1, there are seven rRNA operons within the chromosome [1]. Each copy of the operon contains the *16S*, *23S*, and *5S* genes. Between each pair of genes is an internal transcribed spacer (*ITS*) region, with the *ITS* region between *16S* and *23S* being identified as the *ITS-1* region, and the *ITS* region between *23S* and *5S* being identified as the *ITS-2* region.

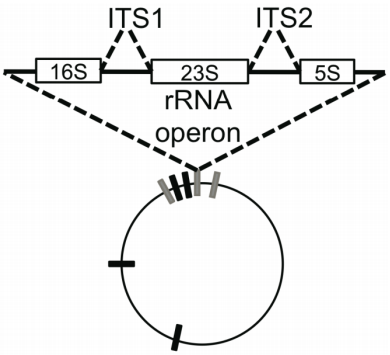

Fig. 1. A simplified diagram of the *E. coli* chromosome, displaying the seven copies of the *ITS-1* and *ITS-2* regions [1].

While these *ITS* regions are repeated seven times throughout the chromosome, each individual copy is not necessarily identical to the other copies. Each unique sequence that appears is identified as an allele, and an allele ratio can be obtained based on the number of unique alleles and the number of times each allele appears. If all seven copies of the *ITS-1* region within a particular isolate are identical, then this is an allele ratio of 7, as there is only one allele and it appears seven times. Whereas if there are four copies of one allele

and three copies of another allele, this is denoted as an allele ratio of `4:3`. If all copies are unique, this is an allele ratio of `1:1:1:1:1:1:1`. The full list of possible allele ratios can be seen in Table I.

TABLE I. LIST OF ALL POSSIBLE ALLELE RATIOS

Allele Ratio	Number of Alleles
7	1
6:1	2
5:2	2
4:3	2
5:1:1	3
4:2:1	3
3:3:1	3
3:2:2	3
4:1:1:1	4
3:2:1:1	4
2:2:2:2	4
3:1:1:1:1	5
2:2:1:1:1	5
2:1:1:1:1:1	6
1:1:1:1:1:1:1	7

B. Pyroprinting

From each *E. coli* isolate two pyroprints can be obtained, one from the *ITS-1* alleles and one from the *ITS-2* alleles. The pyroprint is obtained by simultaneously pyrosequencing all seven copies at once [1]. The product of this process is a vector of real values, and while the original alleles and allele ratio cannot be reproduced from it, it is a "fingerprint" of them, shown in Figure 2. Changes in alleles themselves, or in the ratio of alleles, are reflected in the resulting pyroprint.

These pyroprints are then used for strain typing. Two isolates are deemed to be the same strain if both their *ITS-1* pyroprints are similar enough to each other, and their *ITS-2* pyroprints are similar enough to each other, where similar enough is determined based on Pearson correlation, as explained in [1].

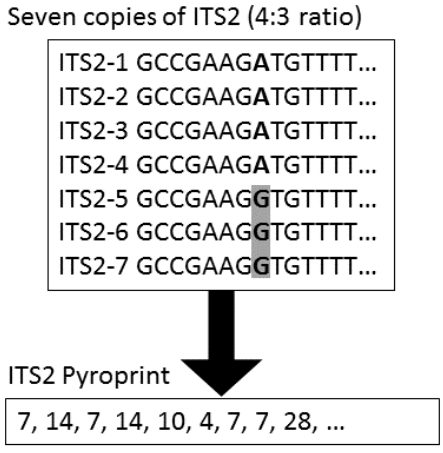

Fig. 2. A simplified diagram of the inputs to the pyroprinting process and the result from the process [1].

In this context, a false positive occurs when two isolates have matching pyroprints, but have different alleles or allele ratios. This would cause the two isolates to be falsely concluded to be the same strain, when in fact the underlying DNA is different.

C. Experiment Overview

In order to identify false positives, and obtain a general frequency rate, a series of steps were taken. First, multiple *E. coli* strains were chosen as being of particular interest, and 100 *E. coli* isolates from those strains were identified for use in the research. Next, a PCR was run on each isolate to amplify its *ITS* regions, including a region primer identifying which region it came from. This PCR also attached a barcode, identifying which isolate the sequence originated from, and a known adapter to the beginning of each sequence. The resulting DNA was mixed together to form one sample, which was then sequenced using an Illumina process. The sequences from the Illumina process were then processed and analyzed by a program built on top of Hadoop, which identified the alleles and allele ratio for both *ITS* regions for every isolate. Then the identified alleles and allele ratios for each isolate within each strain could be used to calculate the frequency of false positives within the pyroprinting process.

This paper is primarily interested in the computational analysis that was completed using Hadoop.

III. IMPLEMENTATION

The core analysis program is implemented on the distributed computing framework Hadoop. hadoop is itself an implementation of the MapReduce, a programming model for processing large data sets [2]. The input to the program is a set of DNA sequences with a size on the order of millions, and the desired output is the set of isolates along with their allele ratio and alleles for each *ITS* region.

The analysis is broken up into two separate Hadoop jobs. The first identifies which isolate and *ITS* region each input sequence originated from, finds the allele within the sequence, and counts the frequency of that allele within that particular isolate and *ITS* region. The second job groups all alleles from the same isolate and *ITS* region together, and performs a statistical test to determine the proper allele ratio, based on the raw frequency of each allele.

A. First Job

First, the mapping function runs on each input sequence. This mapper attempts to parse the sequence to get three important pieces of information: the barcode, the region primer, and the allele. The first step in the parsing process looks for the known adapter and uses this as a reference point to find the other information. Next, it looks for the barcode, to identify the isolate, and then the region primer to identify which *ITS* region it came from. The allele starts immediately after the primer, however the length varies and is defined based on the pyroprint process. To find the end of the allele a theoretical pyroprint is run, using the dispensation sequence for the particular region defined in [1], and the last nucleotide used in the pyroprint is the last nucleotide of the allele.

A number of issues can occur during this parsing process. First, the adapter may not have existed in the input sequence. Other possibilities are that the read started partway through the barcode, the barcode that was read was not a barcode used in the experiment, neither region primer existed in the sequence, or the read was not long enough to finish the pyroprint. If any

of these issues occurred then the appropriate validity code, other than "Valid", is assigned to the sequence. The properties that were unable to be parsed are left empty.

After parsing, the mapper outputs the key as the validity, barcode, region, and allele. The value is simply the original sequence. This setup ensures that all inputs with the same barcode, region, and allele are grouped into the same reducer, allowing the frequency of that allele can be summed.

The reducer receives the validity, barcode, region, and allele as the key, and a list of original input sequences associated with that key. If the validity code is anything other than "Valid", then the reducer will output all of this information as an object to a "Trash" output, so that the sequences are available for investigation later and are separate from the valid reads. Otherwise, if the validity code is "Valid", then the number of input sequences is summed. For these valid reads, the reducer will output the key as the barcode and region, and the value as the allele and count. This allows all of the alleles with the same barcode and region to be grouped together in the second Hadoop job.

B. Second Job

The second job takes as input all of the "Valid" output from the first job, but not the "Trash" output. As the input already has the key defined to group the alleles based on the barcode, or isolate, and *ITS* region, the map step does not need to make any modifications. Thus an identity mapper is used.

Once in the reducer, the frequencies of all alleles are summed to get the total number of reads from the particular isolate and region. Any allele with a frequency less than a certain portion of the whole will be deemed "Mutated". This cutoff portion is configurable, but is usually set to between 1% and 4%. An allele that only appears once in the chromosome should still maintain 1/7th of the total reads, therefore any allele with a frequency significantly less than that must not be a true allele from the chromosome. A likely possibility is that a mutation occurred somewhere in the process, either in the amplification PCR or in the Illumina sequencing process, that caused this allele to appear. These "Mutated" alleles are written to their own output, separate from the results.

For all remaining alleles, those that are not deemed "Mutated", further analysis is performed to determine the proper allele ratio. Any allele ratio with the same number of alleles as are present could potentially be correct. For example, if there are two non-mutated alleles present, the possible allele ratios are `6:1`, `5:2`, and `4:3`, as seen in Table I.

For each potential ratio, a Chi-square test for independence is used with the observed and expected counts to determine if it is plausible to be the correct ratio. The alleles are sorted in descending order by count, and each allele is then assigned a value from the potential ratio. The expected counts are then calculated using the value from the potential ratio and taking that portion of the summed allele frequency. The test is then run with a significance level of 0.05. If the null hypothesis is rejected, then it is known with 95% confidence that this is not the correct allele ratio. The desired outcome is that the null hypothesis is rejected for all of the potential ratios except for one, which would then become the presumed ratio. In the case that there is only one allele, and therefore the only possible ratio is `7`, this is presumed to be the correct ratio without running a statistical test, as this would have zero degrees of freedom. If the null hypothesis was rejected for all possible ratios, or it was not rejected for multiple ratios, then the allele ratio cannot be determined.

After the allele ratio is found, the results are output containing the catalog number, barcode, region, allele ratio, and the list of alleles. The list of alleles also includes the count and part of the ratio associated with each allele. The catalog number identifies which strain the isolate is a part of. This mapping is defined in a configuration file which is loaded into Hadoop using a `Distributed Cache`. Each reduce only outputs a single result, but the total output from the system will contain a result object for each *ITS* region of every isolate.

The alleles and allele ratios can then be compared for isolates within the same strain (with the same catalog number), to determine whether or not a false positive exists within that strain.

IV. Performance

At the beginning of the project, the expected input size was 500 thousand sequences. This led to the exploration of an in-memory, sequential version as a potential viable option. This version was implemented using Java, with heavy reliance on the Stream API. Using the Stream API allowed the general processing pipeline of MapReduce to be replicated in a single machine implementation [3][2]. With the Stream API two versions were implemented, sequential and parallel, where the parallel version used a parallel stream to distribute the process between multiple cores. A machine with a four core Intel i7 processor and 8GB of RAM was used to run performance metrics for these two implementations, and these can be seen below in Figure 3. Each data point is an average execution time from three trials. Both versions were run with the Java Virtual Machine configured with 2GB of maximum heap space, and completed successfully with input sizes up to 1.5 million sequences, but ran out of memory with 1.75 million. As seen in the graph, the parallel version performs better with all input sizes except 1.5 million, where being close to the memory limit creates a bigger slow down.

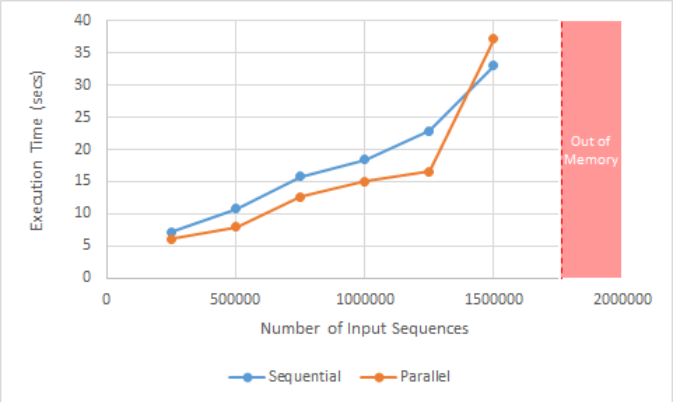

Fig. 3. A graph showing the execution time of the sequential and parallel single machine implementations on input sets of varying sizes.

Later in the project, the decision was made to use a more extensive sequencing process that would yield a much higher number of sequences, between 10 and 20 million. From preliminary testing with the sequential version it was clear it could not handle input sizes of that scale, and this led us to the MapReduce implementation using Hadoop, discussed in detail in Section III. The real sequences that were received from the sequencing process only contained 5 million sequences, instead of the estimated 10 to 20 million. The performance for this implementation can be compared to that of the single machine implementations in Figure 4, with execution times for the Hadoop implementation on input sizes up to 5 million. The Hadoop implementation was run with a cluster in Google Cloud Compute Engine and made up of one master node and three worker nodes. Each node has 2 vCPUs and 8GB of memory. The input data was stored in Google Cloud Storage, as opposed to the typical use of the Hadoop Distributed File System. The results show that with this cluster configuration the Hadoop implementation is significantly slower for the small input sizes that the single machine implementations could also analyze. However, the Hadoop implementation continues to perform even with much larger input sizes, running 5 million sequences in 3 minutes and 42.7 seconds on average. It was also found that 20 million sequences could be analyzed in 7 minutes and 10.2 seconds on average, and it is likely that significantly larger data sets could be handled easily even with the current cluster configuration.

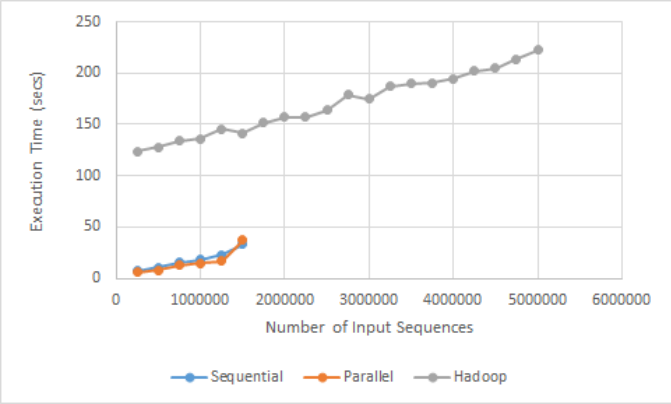

Fig. 4. A graph showing the execution time of both the Hadoop and single machine implementations on input sets of varying sizes.

V. Conclusion

Our method allows for the analysis of pyroprinting as a process in a reliable, scalable manner. It results in a detailed summary from which further analysis can be performed in various directions for many potential findings. This method is generally applicable to the finding of false positives in any DNA fingerprinting process, but can be applied to many areas within bioinformatics.

The largest benefit from this project for our research is insight into the general occurrence of false positives within pyroprinting. The data is still under analysis, but this will inform how we use the pyroprinting process in the future, and potentially help us discover how the process can be improved.

VI. Acknowledgements

First and foremost, I would like to thank Skyler Gordon, the lead on the project as a whole. He was my main resource for understanding the problem domain and coordinating my contributions with the general project. The project never would've happened without him. I would also like to thank Dr. Alex Dekhtyar, my advisor, for all the assistance he gave in helping me find the project and work through the problem. Thanks are also due to Dr. Michael Black, Dr. Christopher Kitts, and Dr. Jennifer VanderKelen. Finally, I would like to acknowledge Zach Zhang, for allowing me to adapt his theoretical pyroprinting code for use in this project.

References

[1] Michael W. Black, Jennifer VanderKelen, Aldrin Montana, Alexander Dekhtyar, Emily Neal, Anya Goodman, and Christopher L. Kitts. Pyroprinting: A rapid and flexible genotypic fingerprinting method for typing bacterial strains. *Journal of Microbiological Methods*, 105:121 – 129, 2014.

[2] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[3] Kishori Sharan. *Beginning Java 8 Language Features: Lambda Expressions, Inner Classes, Threads, I/O, Collections, and Streams*. Apress, Berkely, CA, USA, 1st edition, 2014.

[4] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc., 1st edition, 2009.