Assignment 4: Colour Blindness

Subhojyoti Khastagir
SR. no.: 04-04-00-10-51-21-1-19446
M. Tech. - CSE
Department of Computer Science and Automation
subhojyotik@iisc.ac.in

1 Introduction

In this module, we take a look at the reason for red-green colour blindness at the level of genetics. We compare the genes responsible for colour sensors in the eyes with that of a reference sequence of genes, and find out the configuration of the gene responsible for the defect.

The gene responsible for the medium and long wavelength sensors in our eyes are present on the X chromosome. The two genes have a similar structure of exons, with around 15 positions of difference between them. Because of the similarity of the structures, defects may arise during crossing over of two X-chromosomes which result in a mix of the long and medium wavelength genes, or sometimes even one less or one more gene.

2 Data

We are provided with genetic sequence of the X-chromosome of around 150-million length which includes the genes responsible for the long and medium wavelength sensors of the eyes. We are also provided with the last column of the BWT string in a file which has around 100 characters per line. The mapping from the BWT to the reference sequence is also provided.

3 million reads of around length 150 are also provided. These reads are to be aligned with the reference string to find out the exact configuration of the gene.

3 Technique

Because of the enormous size of the reference sequence, it is difficult to naively iterate it and find matches. It is not impossible but requires huge computational resources to do so. Clever techniques and data structures are required to be able to make use of the properties of computing resources like temporal and spatial properties of RAM and cache memory of the processor.

3.1 Suffix Tree

We can use a suffix tree of the reference sequence and traverse the tree to find the matching locations of the given read in the reference string. This method would be a lot slower because of using pointers to jump around locations in the tree because of the properties of cache memory of the processor, which can take advantage of spatial repetitions. On top of that we require extra pointers to point back to the locations of matches as well as pointers to avoid having to repeat the traversal of the suffix tree in case of a mismatch. It can be improved with the Burrows-Wheeler Transformation (BWT)

3.2 Burrows-Wheeler Transformation

The Burrows-Wheeler Transform is an efficient method used often to compress strings. Using this method, we can find matches of reads in a long reference sequence while taking advantage of the spatial properties of a processor.

There are two kinds of queries: the "rank" and the "select" query which can be used to perform searches on the BWT of a string. Of these we only require the "rank" query to match the reads to the reference sequence. By searching sequentially in a "band" which is nothing but a contiguous portion of the BWT string, we can speed up the search process.

As far as the memory requirement is concerned, we do need extra storage to be able to perform the rank query. If the length of the reference sequence is n, then the storage requirement for storing the ranks is $O(4n)$. This can be reduced by using a technique known as Δ-milestone. In this technique, instead of storing the ranks of each and every character, we store the rank of every Δ-th character. In this way the storage requirement is reduced to $O(\frac{4n}{\Delta})$.

4 Implementation

The following functions were provided in the code template:

1. loadLastCol: This function takes the path to the file which stores the last column of the BWT and returns a list of lines in the file. Each line is a string of 100 characters in the BWT's last column. The reason behind using list of strings instead of a single string is that it enables the Δ-milestone to be easily calculated for $\Delta = 100$.
2. loadRefSeq: This function takes the path to the file which stores the reference sequence and returns a list of lines in the file. Here also, each line is a string of 100 characters and this makes it easy to calculate the Δ-milestone for $\Delta = 100$
3. loadReads: This function takes the path to the file which stores the reads and returns a list of strings where each string represents a read.
4. loadMapToRefSeq: This function takes teh apth to the file that containes the mapping from the BWT to the reference string. It returns a list of integers which represent the position of the corresponding character in the reference string.
5. MatchReadToLoc: This function takes a read and returns a list of possible locations where the read might match the reference sequence. In this function, the "N"s in the read are replaced with "A"s and the complement of the read is also considered as suggested in the provided problem statement.
6. WhichExon: This function takes a list of positions of match and returns the exons which correspond to the positions
7. ComputeProb: This function takes the number of matching reads for each exon and returns the probability of it being the four configurations (normal 3 genes, one gene mixed out of 3 genes, one gene mixed with one extra gene 4genes, one gene mixed with one less gene 2 genes)
8. BestMatch: This function takes a list of probabilities of the four configurations and returns the most likely configuration of the four

The following functions were implemented on top of the template functions:

1. getIndices: This function computes the Δ-milestones of the BWT sequence and returns a dictionary of the same
2. getBand: This function returns the band in the reference sequence where the read matches and the offset upto which it matches the reference sequence in case the whole read does not match
3. complementRead: This function returns the complement of the given read which is also required to compute the matching location
4. countMismatches: This function takes two strings and returns the number of character where they differ from each other

5. : getLocsOfRead: This function calculates the positions where the given read matches the reference sequence. The MatchReadToLoc function makes use of this function to find matching locations of the given read and its complement

The following global variables are declared:

1. INDICES: It is used to store the Δ-milestone in order to avoid repeating the calculation

The following libraries are used in addition to the ones provided in the template:

1. math: The math library is used in the binomial distribution to calculate Combination

The approach of the algorithm is as follows:

1. The provided template first loads all the files (BWT last column, reference sequence, reads and the map from bwt to reference sequence)
2. For each read, repeat till step 5
3. Using MatchReadToLoc function, find out the positions where the read mathches the reference string. This function calculates the matching position for the read as well the complement of the read
4. Using WhichExon function, find out which exons in the reference sequence correspond to the positions calculated in the previous step. If a read corresponds to both long and medium wavelength exon, then it is considered to contribute 0.5 to both, otherwise its contribution is 1 to whichever exon it matches
5. Maintain a count of the exon number where the reads match
6. Now we have a count of how many reads matches the exons
7. Using ComputeProb function, calculate the probability of the count being from one of the four gene configurations. The probability is calculated as a binomial distribution of the event of a read being from a long wavelength gene. The equation of binomial distribution is given by $\sum {}^{n}C_{r}p^{r}(1-p)^{(n-r)}$, where n is the number of total events, r is the number of favourable events and p is the probability of favourable event. Here, r corresponds to the number of reads that matches an exon from the gene of long wavelength, and p is precomputed from the configurations and hardcoded into the program.
8. Using the BestMatch function, decide which of the configurations is most likely one from the probabilities computed in the previous step.
9. Output is the most likely configuration of the four possible gene configurations

5 Compute Environment

The code was run on a server with Intel Xeon Platinum CPU of 2.3GHz base clock speed, little endian x86_64 architecture. It has 2 socket with 24 cores per socket and 2 threads per core, hence a total of 96 threads. It has 66 MiB of L3 cache, 48 MiB of L2 cache and 1.5 MiB each of data and instruction L1 caches. The RAM installed on the server is 252GB.

6 Analysis and inferences

6.1 Output of the program

The program gives the following counts of exons:

	Exon 1	Exon 2	Exon 3	Exon 4	Exon 5	Exon 6
Red	135	61	67	128	275	358
Green	135	185	75	123	328	358

The computed probabilities of the four configurations are:

Configuration 1	Configuration 2	Configuration 3	Configuration 4
0.00129	0.05343	0.10969	0.06182

From the probability values, it is clear that configuration 3 is the most likely configuraiton since it has the highest probability value

6.2 Time requirement

The code took a total of 1236 seconds which translates to 20.6 minutes of runtime

6.3 Space requirement

Since the code was run on a server with more than adequate RAM, it executed without a hiccup. The resident memory was around 11.3GB and the virtual memory was 11.6GB in size. However it could not execute on an average machine with 8GB of RAM and resulted in a python "MemoryError".

6.4 Inference

The time requirement was definitely reasonable, but the fact that it could not run on a computer with 8GB of RAM adds to the cons. Maybe making use of a better technique to read the file instead of simply reading in all the lines of the file may improve the situation.

7 Conclusion

In study of genetics, working with such ginormous sequences are very usual, hence knowledge of data analytics is required to help reduce the requirement of compute resources, such as designing techniques to massively parallelize the computations and perform them in a distributed fashion, or something more simple as using clever data structures like what we saw in this module. This not only makes it easier to perform the difficult computations, but also in a very feasible manner which avoids waste of time, resources or money.