Brenden Kadota
260718799
COMP 561

Local Alignment of Probabilistic Genome with Burrows-Wheeler Transform

Introduction

Sequence alignment is often an important part of understanding the function of new genes. Aligning an undiscovered gene or protein to a known sequence gives researchers clues into the functions of the gene. Next Generation Sequencing (NGS) has made tremendous progress in terms of speed, read length, and throughput, along with a sharp reduction in per-base cost (Dijk et al., 2014). With the expansion of genome databases, finding homologies for protein or DNA sequences are becoming easier.

Currently, BLAST is the most popular alignment tool. BLAST preprocesses the genome database into k-mers, consecutive DNA sequences of length k, and saves the location of each k-mer. The query sequence is also processed into k-mers and query for identical k-mers in the database. Matches are linearly expanded and scored. High scoring matches are called seeds and are expanded using a comprehensive local alignment algorithm. BLAST is a quick and efficient tool to search a database of a discrete sequence. However, BLAST is unable to search through probabilistic genomes since it can not preprocess the genome into k-mers.

Read alignment tools such as BWA, BOWTIE, and BOWTIE2 have compressed the genome through the burrows wheeler transform (BWT). The BWT is a reversible permutation of text originally created for data compression (Burrows and Wheeler, 1994). The BWT is created by appending the character $ to the text being transformed. Character $ is lexicographically smaller than all alphabet in the text. Then all cyclic rotations of the text are placed into individual rows and sorted lexicographically. The resulting BWT is the last character of every cyclic rotation or the rightmost column (Figure 1a).

The BWT has a property known as last first (LF) mapping. The i^{th} occurrence of a character in the last column corresponds to the same character in the i^{th} occurrence in the first column (Ferragina and Manzini, 2000). This property allows the BWT to be quickly searched for a query. To search for a query, start at last character in the query at position X. Find the range for of occurrences of the character at X in the first column of the BWT matrix. Then count the occurrences of character at X - 1 in last column of the BWT. The character in the last column corresponds to the character before of the first. Because of LF mapping, the occurrences of the character at X-1 become the new range. Thus, you look in the first column for the character at X-1, but now constrained to the new range. The algorithm terminates when the range becomes zero, or when the substring is fully quired (Figure 1c).

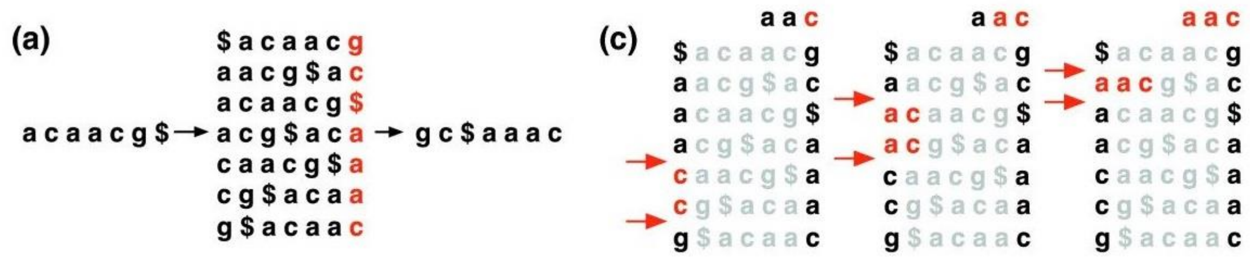

Figure 1. a) The Burrows-Wheeler matrix and transformation for 'acaacg' (c) repeatedly applying the last first (LF) mapping to recover the original text (in red on the top line) from the Burrows-Wheeler transform (in black in the rightmost column).
https://genomebiology.biomedcentral.com/articles/10.1186/gb-2009-10-3-r25

Although LF mapping can be used to find an exact subtext, it is insufficient to align substrings with mismatches or insertions. Here, I propose an algorithm that takes the seed and extend technique from BLAST, and LF mapping from BOWTIE to align query sequences against a probabilistic genome. The algorithm uses LF mapping to search for seeds that are expanded using the Smith Waterman algorithm. Further, once the LF mapping reaches the end, positions of low confidence are then explore to locate more seeds.

Methodology

The BWT of the genome was made by using the brute force method. However, due to memory space, all cyclic rotations of the genome were unable to be stored in a matrix as in figure 1. Therefore, a suffix array was created instead. The suffix array was created by creating an integer array of all the starting points in the sequence. The starting points were used as the start of suffixes to be sorted. The BWT character can then be extracted from the suffix array (SU). Since all the prefixes have been sorted, all cyclic rotations of the string have been sorted too. Then the last character of each cyclic string would be the character at SU(i) – 1. If i equals zero, then it returns the '$' character. Therefore, the ending character of each cyclic rotation is SU(i) – 1 which forms the BWT.

In the first phase, the algorithm uses FM indexing to find seeds. The algorithm starts at the end of the query and uses FM indexing. A seed is found when the algorithm reaches a point where only one position in the genome is found using LF mapping. Thus, a seed is an exact matching of the query sequence to the genome. The algorithm then saves the seed to be expanded in phase 2. To account for probabilistic and genuine differences between the query sequence and the genome the algorithm uses a backtracking approach. After a certain number of exact matches, the algorithm saves the lowest confidence character at each position. When the algorithm reaches a seed, the seed is stored and the algorithm backtracks a to the position with the lowest confidence and explores with the lowest confidence character there instead. The number of exact matches and backtracking steps can be modified to increase seed numbers. Further, if the query has not been completely explored, the algorithm then starts at the end of the smallest seed.

The second phase of the algorithm is the expansion phase. During the expansion phase the seed position in the genome is locally aligned to the query with the Smith Waterman algorithm. The Smith Waterman algorithm is first done on the seed location and the query sequence. Then the table is dynamically expanded if the largest value in the table is 10 nucleotides away from the end of the table. Thus, the Smith Waterman algorithm can find insertions or deletions of at most length 10. Further, the query and genome sequence are then reversed to locate local alignments in the opposite direction. The reversed sequence is then added to the forward sequence to obtain a local alignment in both directions.

The Smith Waterman algorithm also used a different scoring scheme. The score of a match is the confidence value at that nucleotide in the genome. The score of insertions, deletions and substitutions at that position would then be the negative score of the confidence at that nucleotides position. Therefore, higher confidence regions in the genome have a higher weight than lower ones. When the alignment is found, the average probability of the sequence is also calculated by taking summing the confidence scores at each genomic nucleotide and dividing by the length of the genomic sequence.

Results

The BWT was performed on a portion of chr22 of the predicted BoreoEutherian ancestor sequence. The BWT was stored in a text file containing 600,000 nucleotides and probabilities with the size of 4,611kb, 1.3 times bigger than the sequence file and the probability file. The BWT was computed in $O(n \log n)$. Further, the BWT was created in 2.4 seconds. An occurrence table was stored for speed and ease of calculation with the size of 16,011kb. The occurrence table can be omitted and calculated from the BWT if needed.

To find seeds, the algorithm was run with an exact match parameter of five and allowed for five single nucleotide polymorphism (SNP). Seeds were then expanded with the Smith Waterman Algorithm. The Smith Waterman Algorithm stopped when the maximum value in a cell was 10 cells away from the end of the table or the dynamic programming table could not expand due to query or genome size.

I tested 1000 query seeds from the BoreoEutherian genome. The query seeds were of length 30, 50, 100. Query seeds were created by selecting a random section on the genome and taking a query from there. Further, the confidence of the sequence was used to select the nucleotide randomly. At all positions in the query a random number between 0 and 1 was selected. If the random number is above the confidence of the nucleotide at a position, a random new nucleotide would be created.

	Length 30	Length 50	Length 100
Correct Match (n = 1000)	791	848	893
Time Elapsed (average)	6.4s	6.7s	9.8s

Table 1. Number of correct matches and time elapsed for queries of length 30, 50, 100

Query sequences of length 100 were the best at finding the correct match but took on average 3.1 seconds longer to find (Table 1). This is due to more seeds being expanded from different substrings of the query sequence. Thus, it is more likely that one of the seeds will contain the correct location of the query sequence. There is a trade-off to more seeds as query sequences of length 100 run the longest compared to the other lengths. Query sequences were found to be incorrectly aligned when the SNP occurred in the first 5 nucleotides. Further, the algorithm would incorrectly align query sequences if the query had low probability SNP.

	Length 30	Length 50	Length 100
Correct Match (1 insertion)	101	202	308
Correct Match (2 insertions)	49	114	119
Correct match (1 Deletion)	324	472	696
Correct match (2 Deletions)	0	0	7

Table 2. Number of correct matches for deletions or insertions in the query

Next, I tested the algorithm performance with insertion and deletion mutations in the query. An insertion mutation was adding random nucleotides to the query while a deletion mutation removed them. The query was made the same as before, but insertion or deletion mutations of length 6 was applied to the query sequence randomly. The same parameters as before were used for the algorithm.

For insertions and deletions (indels), query sequences of length 100 had the best sensitivity in locating the proper region of the query sequence (Table 2). However, the sensitivity for all three query lengths is much lower than queries without indels. This may be due to FM indexing going down wrong subsequence paths when indels are present, thus leading to wrong seed point. This occurs when the query before the insertion or deletion is in a second part of the genome as well. Also, as the number of insertions increased, the sensitivity of the algorithm decreased. This can be expected as there are now two places where the FM indexing can diverge. Further, the sensitivity for deletions was higher than insertions. One reason for the lower sensitivity for insertions is because the algorithm could start in inserted area. Thus, leading to a wrong seed with the prefix of the inserted area.

Discussion

The algorithm present in this report is an efficient algorithm in aligning query sequences that have SNP mutations along the sequence. It can locate the proper query sequence in short 30 nucleotide sequences with a sensitivity of 0.791. Further, if you increase the sequence length the sequence sensitivity increases to 0.893 with the trade-off of a 3.2 second runtime. This increased sensitivity is due to when a longer query is traversed through FM indexing, the range becomes smaller since there are fewer substrings. Thus, when you backtrack, you are more likely to backtrack to the proper character. However, in shorter sequences, there are multiple spots in the genome with the substring, therefore backtracking becomes harder.

For aligning queries with insertions and deletions the algorithm falls short. The algorithm is not sensitive to queries that have insertions or deletions. It locates the correct sequence with insertions with a sensitivity of 0.101 in short sequences of length 30 and 0.308 in sequences of length 100. However, for deletions the algorithm performs better with a sensitivity of for sequences of length 30 and 0.696 for sequences of length 100. This increased sensitivity is because if the insertion or deletion occurs after a long query subsequence then the FM range will be smaller. Thus, when the insertion or deletion occurs, the FM index will terminate and thus a seed will be set in that position. Therefore, a long query subsequence is needed before the insertion or deletion to properly align the query to the correct spot in the genome.

One solution to the poor insertion and deletion detection sensitivity is to change the seed conditions. By recording seeds when 5 sequences are found may increase the sensitivity of detecting insertions and deletions during the gapped extension phase. Further, increasing the parameter in number of substitutions would likely give a better result. Another possibility is to use FM indexing forwards and backwards. One downside to this approach is two BWT would need to be created. Both proposed changes increase sensitivity at the expense of running time.

Currently, the algorithm proposed is a fast sequence alignment tool for queries with substitutions in them. However currently the algorithm shows every alignment for each seed found in the genome. Therefore, a way of distilling alignments to only present the most interesting alignment is needed. Further, an algorithm called SMID uses disturbed sequence alignment to speed up local alignments. Further, reducing memory by computing the occurrence table will be needed when using the presented algorithm on bigger genomes.

The algorithm presented offers a novel strategy on aligning query sequences with a probabilistic genome. The algorithm is a good choice if the query has high similarity to a part of the genome of interest. Further, the sensitivity of the algorithm can be increased by increasing the number of seeds taken in a range or by increasing the number of backtracking steps. However, the algorithm struggles to achieve and adequate sensitivity for queries with insertions or deletions in them. As probabilistic genomes become more prevalent, this algorithm is a useful tool to align sequences with high similarity to the genome.

Refences

Burrows M, Wheeler DJ. A block-sorting lossless data compression algorithm, Technical report 124 , 1994Palo Alto, CADigital Equipment Corporation

Altschul, S. (1990). Basic Local Alignment Search Tool. *Journal of Molecular Biology*, *215*(3), 403–410. doi: 10.1006/jmbi.1990.9999

Dijk, E. L. V., Auger, H., Jaszczyszyn, Y., & Thermes, C. (2014). Ten years of next-generation sequencing technology. *Trends in Genetics*, *30*(9), 418–426. doi: 10.1016/j.tig.2014.07.001

Ferragina, P., & Manzini, G. (n.d.). Opportunistic data structures with applications. *Proceedings 41st Annual Symposium on Foundations of Computer Science*. doi: 10.1109/sfcs.2000.892127

Langmead, B., Trapnell, C., Pop, M., & Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, *10*(3). doi: 10.1186/gb-2009-10-3-r25

Langmead, B., & Salzberg, S. L. (2012). Fast gapped-read alignment with Bowtie 2. *Nature Methods*, *9*(4), 357–359. doi: 10.1038/nmeth.1923

Pearson, W. R. (1990). [5] Rapid and sensitive sequence comparison with FASTP and FASTA. *Methods in Enzymology*, 63–98. doi: 10.1016/0076-6879(90)83007-v