COMP561 Final Project Research Report

Alignment Algorithm Design for Probabilistic Genomes

Instructor: Mathieu Blanchette

Agnes Liu 260713093

11 December 2019

1 Introduction

In the research field of genome analysis, sequence alignment is a widely used methodology for cross comparison of the genomic profiles of organisms. It is vital in detecting functional and structural homology and evolutionary origins amongst various species. (Mount 2006) In 1990, Altschul et al. have designed the Basic Local Alignment Search Tool (BLAST) for the purpose of quickly identifying alignments with optimal local similarity that align queries of interest against given genomes. It is now one of the most broadly used tools in the study of bioinformatics. However, this traditional approach considers only a fixed genome: the optimal local similarity computed by BLAST is thus based on the assumption of an entirely given genome. This creates some limitation for the functionality of BLAST since healthy viable individuals can have single nucleotide polymorphisms (SNPs) and it would be burdensome to get comprehensive information of genomes of interest individually. Since SNPs are highly variant amongst individuals within species, it would be very useful to consider sequence alignment problems based on a probabilistic genome which gives highest probabilities for genotypes with highest frequencies in population. Therefore, this project is aiming at designing an implementation of BLAST based on a probabilistic genome. This implementation would be more flexible in computing the optimal local alignment scores and could be more biologically realistic.

2 methodology

Traditional BLAST consists of 3 major steps: identifying high scoring words, scanning the genome for un-gapped hits, and extending with gapped alignments for total alignment score comparison. (Altschul 1990) Due to the uncertain feature of datasets considered by this project, the developed algorithm consists of 2 steps only: identifying high scoring regions (HSR) and alignment expansion. Underlined words are names of the functions.

2.1 Identifying HSR

To identify high scoring regions, first break down the query to overlapping words by a given word length.

Then align each word against the probabilistic genome to identify regions with high scoring potentials. The locate function aligns a word against the genome by adding absolute e-based log values of the probability for each of the word's nucleotides. For each word, regions of the genome with highest similarities would return a positive value with very low magnitudes (eg. For word length = 4, |4*ln(0.9)| = 0.4214). locate sets a threshold level for words considered to be highly probable. In the current case, it has a maximum threshold x = (word length-1)*ln(0.7).

After execution of locate, the program moves on to alignment, which performs 3 types of filtering towards the word libraries to narrow down candidate choices. First filtering eliminates candidate positions at each word-specific library that neither overlap with neighbouring word candidates nor with a significantly optimal locate score. As alignment proceeds, it iteratively updates alignment scores for genomic positions with high overlapping patterns in neighbouring words. Different from locate score, the alignment score subtracts locate scores for neighbouring words to obtain most negative values. To eliminate false positive given by eliminating less optimal words with high locate score magnitude, a second filtering is applied at each iteration to discard less optimal positions according to another threshold. The alignment function returns 2 lists recoding the genomic indexing and alignment scores for a significantly smaller set of best HSR candidates.

2.2 Alignment Expansion

The expansion method is a simple modification for the Needleman Wunch algorithm that computes alignment score for each candidate: it uses 1 as matching score, ln(P(nucleotide_in_query)) as score of mis-match. In addition, expansion also counts number of mutations for future candidate comparison, which is done by evaluate that chooses max-scoring candidate with least mutation counts. In the end, the program reports the start index of optimal alignment position as well as count of mutations at this region.

3 Results

3.1 Running Time

The program included some minor designs for optimizing the running time. First, like in the traditional BLAST, the function splitq splitting query into list of words is a run-time optimization approach that significantly fastened the filtering of candidates as word lengths are in general significantly smaller than query lengths. Second, when building up the database with input fasta and probabilistic files, the program creates index keys of high probability positions for each nucleotide. This approach optimized time complexity in 2 ways: 1. Increased run time for locate by narrowing size of positions to traverse for each entry in the word, and 2. Shrank the library sizes for all the words, thus saved time for alignment iterations. Another thing to notice is that, in the python data structures, numpy performed significantly better in running time than pandas even with the demand of converting nucleotide to program-defined equivalent integer indexes for each query entry.

On original Genome	**Avg Length of word library**	**Avg Length of final list**	**Running time (s)**
Query 1	30	5	141.5
Query 2	36	12	145.6
Query 3	41	12	149.3
With Index Keys	**Avg Length of word library**	**Avg Length of final list**	**Running time (s)**
Query 1	8	5	20.5
Query 2	9	5	20.1
Query 3	13	6	21.9

Table 1. The comparison of Running Time with Brut force Traversal vs. Key traversal: Query1,2,3 are queries with no mutation, 4 single point mutations and multiple mutations with lengths>1 respectively. With the aid of index keys the running time for the program is reduced up to 6/7, and on the mean time, average lengths of word libraries are greatly reduced.

3.2 Accuracy

3.2.1 Accuracy For Substitution

Substitution is the most common type of genetic variation as it causes fewest alternation towards biological processes. If comparing genomes of the same species, SNPs across individuals can be viewed as substitution. (Morin et al. 2004) With word length < (query length)/5, this implementation has outputted decent optimality in prediction results regarding queries with substitutional mutations:

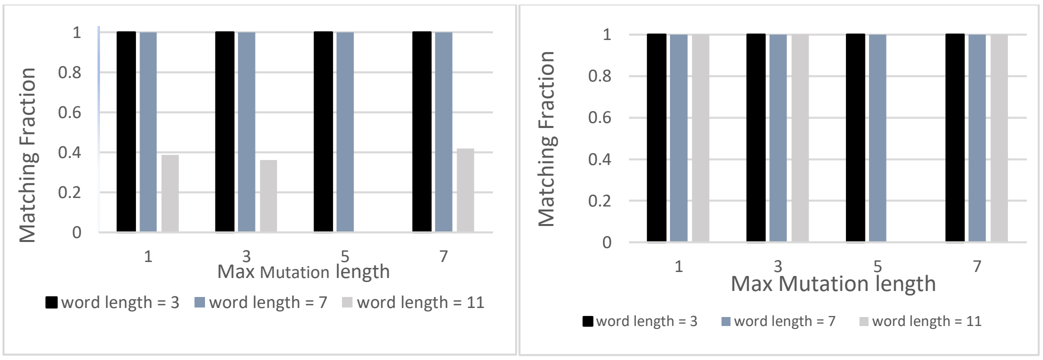

Figure 1. The Fraction of Matching Entries over Query lengths for substitutionally mutated queries: left, % of matching genomic positions with single-site mutated queries non-mutated sites. Right, % of matching genomic positions with doubly mutated queries non-mutated sites. X axis showing maximum mutation lengths at single site. Black bars used word lengths = 3, blue bars with lengths = 7, grey bars =11.

Figure 1 illustrates the aligning performances for queries with 1~2 mutation spots of maximum lengths 1, 3, 5, and 7 respectively. From the graph it is easy to observe that this program has a relatively good performance unless used with wordlength > 20% of the query length.

3.2.2 Accuracy For Insertion/Deletion

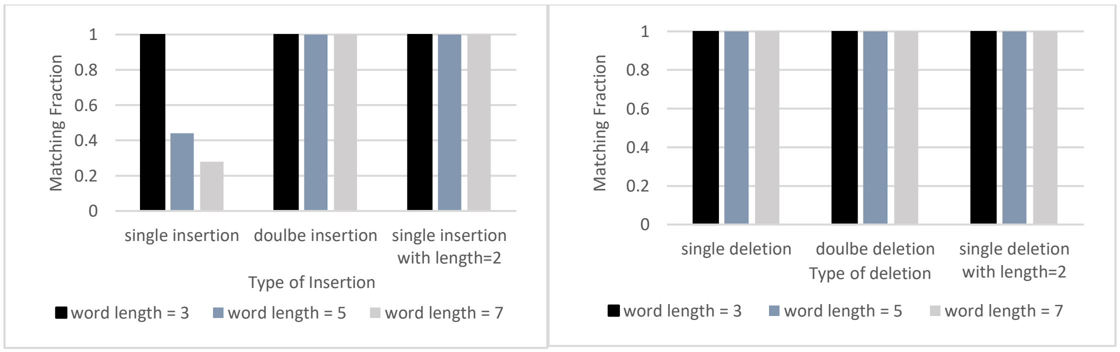

Figure 2. The Fraction of Matching Entries over Query lengths for in/del mutations: left, % of matching genomic positions with single-site mutated queries non-mutated sites. Right, % of matching genomic positions with doubly mutated queries non-mutated sites. X axis showing maximum mutation lengths at single site. Black bars used word lengths = 3, blue bars with lengths = 5, grey bars =7.

Figure 2 showed aligning performances for queries with 1~2 insertion/deletion spots that shifts a sub-sequence of them. In general the model performs well; however, problem still exists as word length choice gets larger.

4 Discussion

4.1 Optimality

With query length = 50, this implementation is guaranteed to find the optimal local alignment if no mutation was introduced. If mutations do exist, in the case of substitution, if the majority of mutations do not cluster at the first 1/10 of the query, this program is still able to identify the global optimal solution, regardless of the length of a continuous mutated region. In addition, if trade-off the running time for optimality by choosing small word lengths, this program would be able to identify global optimal solution even with the mutations of insertion and deletion.

4.2 Choice of Parameters

Since this program implementation relies largely on filtering threshold, its performance could be legitimately improved by adding situation-dependent calculations towards the thresholds as well as introducing iterative approaches in some steps.

For example, since the program performance can be significantly altered by magnitude of word length, it can be modified to execute itself several times with narrowing word lengths and compare the results from each word-length based iteration. The probability threshold for word matches in locate method was also set intuitively: although it logically makes good sense, but in-depth probability calculation can be added to improve the optimality. Lastly, in the election of final solution from the best candidates, evaluate was executed recursively based on increasing mutant length-to-query length ratios. This recursion can be further updated by applying more realistic mutant length-gene ratios concluded by biological literatures.

4.3 Implementation Concerns

Although being guaranteed with certain level of optimality, this program does contain limitations regarding its implementation. One of the apparent errors it might produce is the offset of the stored starting position on the genome when insertion/deletion is introduced to the query: because the program only takes records on the smallest index of the matching region, the solution might be skewed by insertion or deletion since they shift a whole subsequence off.

A potential fixation of this error could be: instead of storing the first position only, alignment stores the exact positions of the start of the current word, and create another list to record part of the query this proposed region is matched to, and perform bi-directional expansion in the Needleman Wunch algorithm instead of the current mono-directory expansion.

More importantly, since genes are observed to have longer lengths than the tester query length, I should test for longer queries where more issues might be revealed.

5 References

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

Brookes, A. J. (1999). The essence of SNPs. *Gene*, 234(2), 177–186. doi: 10.1016/s0378-1119(99)00219-x

Morin, P. A., Luikart, G., Wayne, R. K., & Group, T. S. W. (2004). SNPs in ecology, evolution and conservation. Trends in Ecology & Evolution, 19(4), 208–216. doi: 10.1016/j.tree.2004.01.009

Mount, D. W. (2006). *Bioinformatics: sequence and genome analysis*. Cold Spring Harbor, NY: Cold Spring Harbor Laboratory Press.

6 Appendix

Table A3.1: queries used in run time test

query 1: CAACTAACCACCACCCCTGTCTCCACTCACCGGAACAGAGACTCCCCCAG
query 2: CACCTAACCACCTCCCCTGTCTGCACTCACCGGAACAGAGACTCCACCAG
query 3: CACATAACCACCTCCCCTGTCTGCACTCACCGGAACCTCTGAGCCACCAG

Table A3.2.1: queries assessed in figure 1, mutation spots in red

Original query:
CAACTAACCACCACCCCTGTCTCCACTCACCGGAACAGAGACTCCCCCAG
With 1 mutation spot:
CAACTAACCAACACCCCTGTCTCCACTCACCGGAACAGAGACTCCCCCAG
CAACTAACCAAATCCCCTGTCTCCACTCACCGGAACAGAGACTCCCCCAG
CAACTAACCAGGTGGCCTGTCTCCACTCACCGGAACAGAGACTCCCCCAG
CAACTAACCAATCATGATGTCTCCACTCACCGGAACAGAGACTCCCCCAG
With 2 mutation spots:
CAACTAACCAACACCCCTGTCTCCACTGACCGGAACAGAGACTCCCCCAG
CAACTAACCAAATCCCCTGTCTCCACTCACCGACACAGAGACTCCCCCAG
CAACTAACCAGGACCCCTGTCTCCGGAGGCCGGAACAGAGACTCCCCCAG
CAACTAACCACCACTGATGTCTCCACTCACTACCTGTAGAGACTCCCCCAG

Table A3.2.2: queries assessed in figure 2, insertion colored & deletion as '_'

Insertion
CAACATAACCACCACCCCTGTCTCCACTCACCGGAACAGAGACTCCCCCAG
CAACTAACCGACCACCCCTGTCTCCACTCACCTGGAACAGAGACTCCCCCAG
CAACTAACCACCACACCCCTGTCTCCACTCACCGGAACAGAGACTCCCCCAG
Deletion
CAACTAAC_ACCACCCCTGTCTCCACTCACCGGAACAGAGACTCCCCCAG
CAACTAACCACCACC_CTGTCTCCACTCACCGG_ACAGAGACTCCCCCAG
CAACTAACCACCACCCCTGTCTCCACTC_ _CGGAACAGAGACTCCCCCAG