DS 202: Project Report

Minimizers

By: Subhasree Majumder

Indian Institute of Science, Bangalore

28th May, 2021

1 Introduction

Pattern matching is one of the many problems of bioinformatics and a number of exact matching and approximate matching algorithms have been designed to solve the problem. Pattern matching finds its applications heavily in protein, nucleic acid both DNA and RNA sequence matching. One such heuristics of approximate matching follows the 'seed-and-extend' matching wherein k-length kmers known as seeds are extracted from string sequences and stored. When query sequences arrive, similar kmers are extracted from them and exact k-mer matching is done with the database sequence. The matched k-mers are extended using colinear matching to get approximate matching as shown in figure [1].

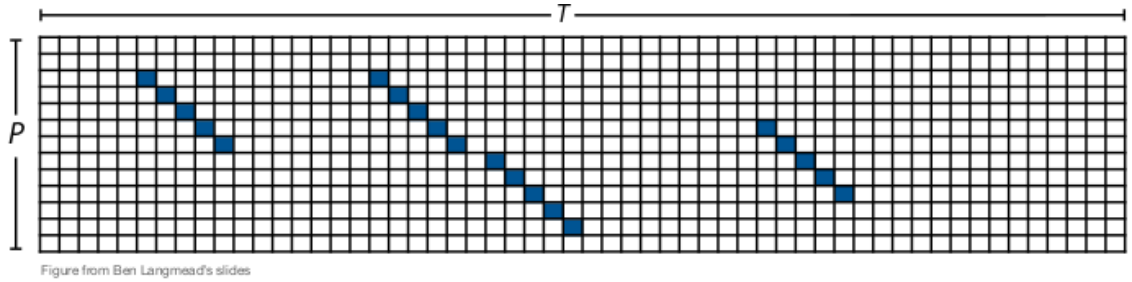

Fig 1: 'Seed-and-Extend' using k-mers

But storing all the seed k-mers might improve the aprroximate matching of query sequence to reference sequence but it also comes with a high space complexity. So instead of storing all k-mers, a subset of them can be stored as was first proposed by Roberts et al. [1] in a concept called minimizers. For example, given the length of string Ti

This project explores the concept of minimizers, startegies to store them, parameters affecting minimizer performance and applications of minimizers. Finally, scope of research has been discussed.

2 Minimizers Algorithm

In order to reduce k-mer storage, we can store a subset of k-mers, called minimizers but what should be the strategy of selecting that subset? One strategy could be storing every k-th position k-mer so that each letter is covered once. But the problem here is that the two sequences containing common subsequence starting at pi and pj respectvely may not have a common k-mer stored if p_i-p_j is not a multiple of k. Further we can apply other stragegies:

2.1 Interior Minimizers

We may choose a subset of k-mers such that they satisfy the following property:
Property 1: If two strings have a significant match then atleast one of the minimizers chosen from one will also be chosen from the other.
We look at a window of w consequtive k-mers and select the lexigographically (ordering) smallest k-mer as the minimizer. If more than one smallest k-mer, all of them are stored as minimizers. These minimizers are called (w,k)-minimizer. Each window w covers (w + k -1) letters. In the following example, which has been borrowed from Roberts et al paper [1], the alphabet of the string uses digits instead of letters. We can see in Fig 1, a sliding window of lengt 4 and k-mer length of 3.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sequence	2	3	1	0	3	2	1	0	1	2	3	3	1	0	1
	2	3	1	**0**	**3**	**2**									
		3	1	**0**	**3**	**2**	1								
window			1	**0**	**3**	**2**	1	0							
with				**0**	**3**	**2**	1	0	1						
minimizer					3	2	1	**0**	**1**	**2**					
in						2	1	**0**	**1**	**2**	3				
BOLD							1	**0**	**1**	**2**	3	3			
								0	**1**	**2**	3	3	1		
									1	**2**	**3**	3	1	0	
										2	3	3	**1**	**0**	**1**

Fig 1: Minimizer with w=4,k=3
Credits Roberts et al. 2004

But there lies a problem with this approach and that is gaps of characters in the sequence string that are not covered by minimizers. Minimizers in two adjacent windows can differ by atmost w positions because the first window's minimizer could be at the beginning kmer and the adjacent window's minimizer is at the wth i.e. the last kmer position. So gap size is atmost w-k. Therefore So setting w<=k ensures no gap occurs between minimizers except the first and ending w-1 characters. This happens when at the start of the sequence if the mnimizer of first window starts at the last kmer in w and consecutive windows do not have minimizers that cover these w-1 letters. Similarly this stuation might happen at the end of the sequence also.

2.2 End Minimizers

So we saw that even without gaps, minimizers might leave out from covering w-1 characters in the sequence ends. If two sequences match each other on their ends such that the length of the match is less than w+k-1. Then there is a possibility that there are no common (w,k) minimizer for the common portion of the string.

We define a (u,k)-end-minimizer where u is the size of window and u varies from 1 to some max size v. The end minimizers are anchored to the end.

But again end minimizers become sparse in the interior of the string and do not cover all characters.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Sequence	2	3	1	0	3	2	1	0	1	2	3	3	1	0	1	1
	2	**3**	**1**													
	2	**3**	**1**	0												
	2	3	**1**	**0**	**3**											
	2	3	1	**0**	**3**	**2**										
	2	3	1	**0**	**3**	**2**	1									
	2	3	1	**0**	**3**	**2**	1	0								
	2	3	1	**0**	**3**	**2**	1	0	1							
	2	3	1	0	3	2	1	**0**	**1**	**2**						
	2	3	1	0	3	2	1	**0**	**1**	**2**	3					
	2	3	1	0	3	2	1	**0**	**1**	**2**	3	3				
	2	3	1	0	3	2	1	**0**	**1**	**2**	3	3	1			
	2	3	1	0	3	2	1	**0**	**1**	**2**	3	3	1	0		
	2	3	1	0	3	2	1	**0**	**1**	**2**	3	3	1	0	1	
	2	3	1	0	3	2	1	0	1	2	3	3	1	**0**	**1**	**1**

Fig 1: k-end Minimizer with k=3 and u=1,2...l-k+1
where l is length of string
Credits Roberts et al. 2004

2.3 Mixed Minimizers

We club together (w,k)-minimizers with (u,k)-end minimizers for u=1,...,w-1 at both ends of the string. As w<=k, every character will be covered by minimizer. The ends of the string will also be covered by end-minimizer. Thus it guarantees that strings with matches at the end will also have common minimizers.

2.4 Possible number of minimizers

Minimizers in two adjacent windows covering w+1 k-mers can differ in their minimizers in two ways only:

Before that, we note: every k-mer is equally probable of being picked up as minimizer. So if there are w+1 kmers, then each minimizer has 1/(w+1) probability of being picked up as minimizer.
Now,
a) If the minimizer in the previous window is on leftmost end. Then in all pobabilty the next minimizer picked up will be different because the overlap between the picked minimizer and k-mer of next window will become k-1.

0 3 2 1 0 vs 3 1 **0 3 2**
3 2 1 0 1 1 **0 3 2** 1
0 3 2 1 0

b) If the minimizer picked up in previous window is the rightmost kmer and the minimizer in next window was picked from w+1th k-mer then the minimizers differ because it was smaller.

3 2 1 **0 1**
2 1 **0 1 2**

Thus we see that minimizers can differing if either the 1st k-met or the (w+1)th k-mer was picked up in two consecutive windows. Thus the probability of minimizers differing in two consequtive windows is 2/(w+1). Thus on an average 2/(w+1) of all kmers are minimizers.

3. Applications of Minimizers

3.1 Sampling suffix arrays using minimizers

Suffix arrays are important data structure for indexing genome sequences. But storing large suffix arrays can be memory intensive. The work by *Grabowski et al.* 2014 [2] which has come up with an algorithm to sample suffix array based on minimizers, was able to search patterns of length 50 by 10% faster time over plain suffix array while the minimizer based suffix array only sampled 5.3% of the total suffixes. Thus a huge time and space optimization has been observed using minimizer based suffix array. The algorithm has been depicted to below in section 3.1.2.
In order to match a pattern P with text T, We calculate minimizer in P[1...w] i.e. prefix of the pattern. Next we do a Binary search the suffix P[j...|P|] in the SA_new Finally we verify each match with the truncated (j-1) prefix of the pattern P.

3.1.2 Algorithm to sample suffix array:

-Pass a sliding window of length w over T
-Calculate the lexicographically smallest substring of length p in each window (the minimizer)
-Leftmost smallest string is chosen incase of ties
-Position of minimizers are the positions of the sampled suffix
-Lexicographically sort the sampled suffixes

3.2 Fast and Low memory compaction of de-bruign graphs using minimizers

De-bruign graphs represent the information from a set of reads. Given a set of reads R, every distinct k-mer in R forms a vertex of the graph and an edge connects two vertices if they have a k-1 suffix-prefix overlap. Compaction of debruign graphs help in data reduction wherein long unitigs are compacted into single vertices. A unitig is a path $p = (x_1,...,x_m)$ over m≥vertices such that |p|=1 or 1<i<m and out and in degree of x_i is 1 and the in-degree of x_m is 1 and out degree of x_1 is 1.We call two strings u and v compactable, if u is the only in-neighbour of v and v is the only out-neighbour of u. The compaction operation replaces u and v with $u.^{k-1}v$. However compaction is a memory intensive step. As we need to load the entire graph into memory, compact them iteratively until no further compaction is possible. R Chikhi et al [3] came up with an elegant algorithm BCALM2 to compact de bruign graphs using a minimizer hashing technique.

The algorithm works in three stages. Given a set of k-mers representing the de bruign graph, the first stage deals with distributing k-mers into buckets. Here the buckets are designned such that

if $|\Sigma|$ is the size of alphabet, then upto $|\Sigma|^l$ where l is the length of l-mer. After the buckets being decided, k-mers will be distributed to the buckets based on lmm where lmm(x) is the l-minimizer of k-1 prefix of x and rmm is the l-minimizer of k-1 suffix of x.

3.2.1 BCALM2 Algorithm 1

Input: set of k-mers K from reads.
1. for all x ∈ K do
2. write x to F(lmm(x))
3. if lmm(x) ≠ rmm(x) then
4. write x to F(rmm(x))
5. for all parallel i ∈ {1,...,4^l}
6. Run compactBucket (F(i))

Second stage includes compaction of the buckets separately and parallely. The kmers in a bucket such that its lmm(x) ≠ rmm(x) reside in more than one bucket and are thus marked as lonely in the compact bucket stage.

3.2.2 BCLAM2 CompactBucket(i)

1. Load F(i) into memory
2. U <- compact u and v in F(i) to u**.**$^{k-1}$v
3. for all string u ∈ U do
4. Mark u's prefix as lonely (**.**) if i ≠ lmm(u)
5. Mark u's suffix as lonely (**.**) if i ≠ rmm(u)
6. if u's prefix and suffix are not lonely then
7. output u
8. else
9. Place u in reunite file

Final stage includes reunification and gluing of unitigs together. Here the string outputs from the compact bucket stage having lonely ends are reunited with matching strings having lonely end with same k-mer. The reunite stage removes duplicates and the output is maximal unitigs. BCALM2 compact debruign graph construction took 76 mins and 3 GB memory on human genome sequence surpassing SOA tools Abyss-P (6.5h,89GB).

4 Experiments

Proposed Experiment 1: Comparative study of parameters window size w and k

I conducted the following experiment, results have been tabulated in table []. I wanted to understand the effect of window length w and k-mer length k on the performance of alignment of query sequence to the indexed reference sequence. The reference dataset was a 4 million bp Ecoli dataset collected from [7]. The query sequences used were ecoli fasta reads from MinION instrument with average length of 3000 base pairs. I have used minimap2 , which is a fast sequence mapping and alignment tool based on minimizer [4]. Both command line and python wrapper known as mappy are available and I have leveraged both for the current experiement.

I have done mainly three types of analysis as is evident from the table: w<k, w=k, w>k and subsequently analysed the number of reads that could be aligned to the reference sequence, noted their mapping quality and the runtime of the total execution. Mapping quality also denoted as mapQ in the table is defined as the following empirical formula []:

$$mapQ = 40.(1- f_2/f_1) . min\{1,m/10\}.log f_1$$

where m is the number of k-mers on the primary chain, f_1 is the chaining score and f_2 is the scoring chain of the second best chain.

We see for both increasing window size w while keeping constant k-mer length, and increasing k-mer length while keeping window size w as constant, both number of read hits decrease and runtime increases. Thus increasing length of k-mer and window size increases execution time and alignment quality.

However, we also see that when window size w<= k , the k-mer length, both number of hits and mapping quality is good which reduces significantly while runtime increases when w>k. This has been confirmed in the paper [1] that shows with w>k, a number of gaps between minimisers are formed, thus not covering efficiently the reference sequence leading to the observed loss in mapping quality. But we can observe that when w=k, the runtime is small compared to other two cases of w<k and w>k.

We also observe that very small window size (w=3) and very large window size with respect to k-mer length (w=100 and 200) are not only pulls up the runtime but also affects the quality of mapping produced.

Thus we can conclude that using a medium sized, comparable window size w and k-mer length k such that w<=k, will give quality alignments and small runtime.

Proposed Experiment 2: To demonstrate 2/(w+1) theoretical bound on number of minimizers generated.

I also tried to demonstrate the 2/(w+1) of all kmers being sampled as minimizer as discussed in section [2.4] using minimap2. However minimap2 provides only number of minimizers present in the chain constructed while mapping the query string to reference sequence, the field being ‘cm’. But to compare the minimizers actually generated to the number of miimizers sampled from the reference sequence, this attribute would not work.

Type of Analysis	w	k	Number of read hits	mapQ=0	mapQ≤50	mapQ>50	Runtime (sec)
Constant w Increase k w<k	3	10	127	10	1	116	1.69
	3	15	123	6	0	117	0.38
	3	20	119	2	0	117	0.443
	3	28	117	0	0	117	0.338
Increase w Constant k w<k	5	20	118	1	0	117	0.283
	8	20	117	0	0	117	0.25
	10	20	117	0	0	117	0.248
	15	20	117	0	0	117	0.255
w=k	8	8	122	9	1	112	21
	15	15	117	0	0	117	0.25
	20	20	117	0	1	116	0.24
	50	50	114	0	0	114	0.29
Constant w Increase k w>k	100	5	116	5	1	110	245
	100	10	114	0	3	111	0.245
	100	15	115	2	3	110	0.377
	100	28	113	11	8	94	0.499
Increase w Constant k w>k	50	28	111	0	4	107	0.36
	100	28	113	11	8	94	0.49
	200	28	94	12	8	74	0.58
	255	28	80	7	12	61	0.58

Table 1: Experiment on minimizer performance w.r.t. window size and k-mer size

5 Future Scope

A number of work has been undertaken to improve the performance of minimizers. Marcais et al [] proposed various ordering schemes that could outperform traditional lexicographic ordering of kmers in minimizer window scehmes. Jain et al [] propsed a weighted minimizer sampling algorithm that down weighted frequently occuring minimizers reducing false-positive match rate. Thus improving minimizer schemes, designing new minimzer strategies and finding new uses of minimizers can be said as the future directions for this topic.

References

[1] Roberts, Michael, et al. "Reducing storage requirements for biological sequence comparison." Bioinformatics 20.18 (2004): 3363-3369.

[2] Grabowski, Szymon, and Marcin Raniszewski. "Sampling the suffix array with minimizers." International Symposium on String Processing and Information Retrieval. Springer, Cham, 2015.

[3] Chikhi, Rayan, Antoine Limasset, and Paul Medvedev. "Compacting de Bruijn graphs from sequencing data quickly and in low memory." Bioinformatics 32.12 (2016): i201-i208.

[4] Li, Heng. "Minimap2: pairwise alignment for nucleotide sequences." Bioinformatics 34.18 (2018): 3094-3100.

[5] Marçais, Guillaume, et al. "Improving the performance of minimizers and winnowing schemes." Bioinformatics 33.14 (2017): i110-i117.

[6]Jain, Chirag, et al. "Weighted minimizer sampling improves long read mapping." *Bioinformatics* 36.Supplement_1 (2020): i111-i118.

[7] Ecoli Dataset Accessed 10th May, 2021 http://s3.climb.ac.uk/nanopolish_tutorial/ecoli_2kb_region.tar.gz