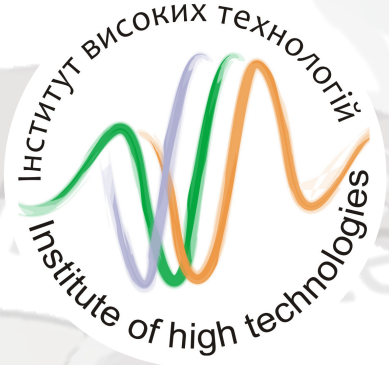
Taras Shevchenko Kyiv
Institute of High Technologies

B I O I N F O R M A T I C S



written by
Claes J. S Lindhardt

Coding a K-mer finder

softweare aided sequenzing

Rubber Duck	Christian D. Schiøtt
Co-Rubber Duck	T. C. Thane
Code Proof-Reader	Maksym Kryshtalov
Grammatical Proof-Reader	Lisbeth Sjælborg

0.1 Abstract

This article looks at how you can make the process involved with sequenzing DNA cheaper and faster primarily by creating new software. This kind of research have already been conducted at other universities around the world such as DTU[1] and US san diego[2]. However it seems no such thing have been done in Ukraine yet. Therefore it makes sense to replicate the process of creating a software which can be used and now understood by Ukranian biologists and bioinformaticians. Futhermore it is important to replicate research with diffrent methods and compare results to further proof the results.

Sequenzing of DNA is particularly useful for control of resistence in antibiotics, used in places such as the food indstry. As well as detecting vital sequenzes in genes. In this article we will look into dormant Tuberculosis sequenzes4.3 The aim of this article is to show how you can develop software to aid taking sequenzing into NGS(next generation sequenzing).

Traditionally when sequenzing DNA, you would first take the sample, then isolate it trough a media; duration one to two days. Following you would do some sub-typing or characterization(eg. gram stain or spp/ssp ID) taking up to a week. finally you would do some molecular typing(PFGE etc.).

This article aims at making tools for the sub-typing part by making small different tools for doing parts of the sub typing. The hope is to make it easier to collect data about the process of sequenzing as well as making it easier to have someone working with sequenzing the DNA and someone improving the tools for sequenzing. Allowing people to specialize for generating an overall better result.

However this is a hugemongus task, therefore this article will only focus on coding a K-mer finder aimed at bacteria, as it is a good start and will provide a good understanding for futher research in that direction.

DISCLAIMER: There is no claim that the research done in this article is the first time this kind of research is done, rather than scientific break through this report is aimed at giving researchers in Ukraine new or better tools for research and engineering as well as a comparison of research earlier done in this field.

[1]DTU have created a varity of tools for sub-typing , https://cge.cbs.dtu.dk/services/
[2]Professors from US san Diego, https://extension.ucsd.edu/courses-and-programs/bioinformatics

0.2 Conclusion

The software made in this report was at first able to find k-mer with mutation, and take out code sections reverse them and express their complementary nucleotides. Applications were found in finding dormant tubercolosis genes origin of replication in datasets, with a random search algorithm and a Gibbs search. The answeres this software provides should be taken into consideration by a biologist before further use. It can be used, however, as a time safer when looking for specific genes. It can give estimates that can potentially save you days of laboratory work. In order to assure it works properly the sections scanned should not be much more then 650 nucleotides long.

0.3 Suggested improvements

This section covers the things, that I either did not have the time or skill to cover and what might be interesting to research next, as this article is mostly re-construction of reasearch done other places, but in a Ukrainian context. Some of the things suggested here might already have been done. Just not as public Ukrainian research. I believe that it is important for a contry to replicate research done internationally in a local context. Not only to check the research and improve data on it, but also to improve local infrastructure.

First improvement would be a more intuitive GUI. It is clear that this is made by people with a background in biology and software, not design. It might be nice to set up a server in an extended period of time and let everyone use the software and try to get some user feedback. For this article I did however collect some user feedback. I had a person with no background in science trying it, plus one with a background in software, and one with a background in biology. None of them had any university education to assure that the software was not overcomplicated for the user experience.

It would also be nice to restructure the code so that it could read series of nucleotides from an other text file then the codefile it self. However it works very well when you just copy and paste the DNA into the file itself.

It would be nice if tools were made which could dig out the relevant bits of the nucleotide chains so that a biologist did not have to give an estimate at first before use.

Contents

1 Making the software

1.1 prequsition and product specifications

In this section I will examine excatly what is needed. This will be done by laying out which steps will be needed to get the program up and running. We will code 5 programs, which are as follows

1. The first one for estimating an ori(origin og replication), and expressing the complementary dna string.
2. Then one which can find the skew distance for estimating what part of the DNA it makes sense to look for ori's in. As well as taking mutations somewhat into acount by using hamming distance.
3. We will try to make a better estimate of the ori, by using a greedy motif search.
4. To improve the processing time, we will try to find the ori by a random search.
5. Finally to try to get the best result, we wilk do a gibbs sampling.

Now the last 3 of these programs does vertically the same so why would we make 3. We do that because as you will see it makes sense to have diffrent measures in diffrent situations, all 3 programs produces different results and it is the job of the biologist to decide which one to use when. As well as it is the job of a computer scientist or software engeneer to look at which one of the 3, there is time to run.

So how do we do that in practice?.
step 1:

DnaA is the protein which initiates the replication. It is telling the DNA-polymerase "bind here". Now typically, the DnaA box is also the most common occurring sequense of DNA. So our first program needs to start by finding the most common strings in a text. We are not sure which of the 'words' we find are the DnaA box. But in first step we are only concerned with finding the most common 'words'.

step 2:
Now imagine we made the function work which finds the most common K-mers. The thing that we want to do now is to figure out which of the 'words' or pieces of code are the DnaA boxes. One way of doing this is looking at complementary strings. How often do we se a common string followed by a just as common string. Like if the DnaA's are trying to tell the DNA-polimiarse "Start here, and stop here"

step 3: Improving step 1
When we have a functioning system like this, we will need to take mutations in the genome in to account. It might not be enough to simply find the most common occuring DnaA boxes. We will properly have to have a way to find them even though all of them have mutations.

step 4:
Okay, now that we got everything well working, and have got something we can go back to if the step we are at does not work, let us try to improve. Maybe we can search more effectivly with

a bit of randomness.

step 5:
Try to apply it to some bacteria that might contain dormant tuberculosis to test if it will be usefull for this purpose.

1.2 Psuedocode and Composition

So this is how the functions from the the Function Map5.1 was put together into usable. It is a very high level of abstraction so if something does not make sense or you want to replicate the code, I recommend that you look at the the Function Map, as this section only explains how the functions from there was put together into programs. If you are reading to understand how I got the results, you however only need this section.

Program 1: PatMatch
We make the assumption that a genome has one single ori(orgin of replication) and that it is represented as a sequence of nucleotides from the four-letter alphabet A, C, G, T. First focus will be on the simple case of finding ori in bacterial genomes, which consist mostly of a single circular chromosome. In this first program we will also assume that there is no mutation in the DNA we are working with.

```
#inputs: DNA, k = k-mer length,
#t = number of outpur k-mers,n = number of iterations
def KmerInverter():
        find the most common kmer of a certain length.
    #Remeber we assumed no mutations so it makes sense
    #to thingk that the most common will be the ori and terminator
        append those two kmers to list
        invert that list and express thier complementory nucleotides
        return the inverted complemented list
```

Program 2: SkewScan
This program is particularly useful when working with ecoli as the point of it is to narrow down the nucleotides that we search within 500 because if we search in a much bigger dataset, I found that the results start to become inacurate. If Skew Diagrams are new to you I wrote a whole section on it4.2.

```
#inputs: a hamming distance, a genome, optionally a nucleotide to look for,
#How many nucleotides you want it to find.
def SkewFinder():
        First find the minmum skew
    print the skew diagram.
    take the 500 nucleotides around that minimum
    find the most common kmers within the accepted hammingdistance.
    return the kmers found in the Ecoli or other round genome.
```

Program 3: MotifDosRFinder
Now we want to we want to try to become better at searching in the local area we can locate in ecoli bacteria with the skew diagram. So from now on we will only look at sequenzes of about a 450 nucleotids or the likes and imagine that that a such sequenze have been pulled out of the

DNA. A place to start with doing that could be a greedy motif search. Greedy motif search is covered in detail in the background theory subsection Statistics and randomness. 4.4.

```
# Input:  a DNA strings, k the length of kmers you want
#and t the amount of kmers you want this should correspond
#roughly to the amount of DNA string
# Output: GreedyMotifSearch(Dna, k, t)
def MotifDosFinder(Dna, k, t):
  -insiciate a list BestMotifs = []
    -loop trough the amount of DNA string given, in the length of the
        kmer we is trying to find.
        find the combination of kmers with the best score.
        return that list of kmers
```

Program 4:MotifDosRFinderRandom
Now we want to have as good results but find them faster so that we could potentially scan larger data set. To do this we will do a random search.

```
#inputs: DNA, k = k-mer length,
#t = number of outpur k-mers,n = number of iterations
def MotifDosFinderRandom(DNA, k, t,n):
        Randomly select a K-mer motifs.in each string
    findScore()

    while(i>N):
        Profile =findProfile(Motifs)
        Motifs = findMotifs(Profile, DNA)
                if findScore(Motifs) > findScore(bestMotifs)
                bestMotifs = Motifs
    return bestMotifs
```

The idea is to keep iterating like findMotifs(findProfile(findMotifs(findProfile(Motifs), DNA)), DNA). until we get a high enough score instead of saying it has to iterate a certain amount of times. It can be a little tricky, and this is explained in dept and with more background in the backgrund theory section4.4
. Now this is the main function. Then we will write a function that repeats this function either until it has the score, we want it to have, or until it has iterated a certain amount of times.

Program 5: GibbSampFinder
If you find this pseudocode insufficient for understaning the theory behind GibbSampling there is a section devoted to it in the background theory section4.4

```
#inputs: DNA, k = k-mer length,
#t = number of outpur k-mers,n = number of iterations
def GibbSampFinder(DNA, k, t, n):
        randomly select k-mers from each motif
        put them into a list
        for n: #Iterate n times
            -Select a random number from 1 to
            the total amount of motifs.

            -remove the kmer that is the selected number in the list

          - find the profile for the other kmers. based on that
            profile of probabilities take out the most likely
            from the removed motifs

           -put that k-mer in at the position you removed the kmer
           from before and do the same thing as many times as
           n tell you to. but now with the new k-mers.

        return BestMotifs
```

Now this is the main function then we will write a function that repeats this function either until it has the score we want it to have or until it have iterated a certain amount of times.

1.3 coding the software

This section shows how in specific the program was made in python if you intent to create your own program based on this article this section will probably not be as relevant to you as the psuedocode for the program section.

As I were coding I needed some sample sequenzed DNA to try all my functions on. I found some online[3]
. After having created the product specifications and pseudocode, it was not to difficult writing the actual code in pyhton.

The functions
If you look in the files something.py ans maybesomethingelse.py you will see all an example of all the functions written for this program. Under each function you might see a section commented out. Thede sections are alternatives to how the function could be coded. You can pick the one that suits the best for your purposes. To really fully understand the the functions I recommend playing around with the output section in each of the files commenting in and out different print of function and adjusting its parameters to see what the different functions does. If you want pseudocode for all of these functions you can look at the Funcion Map section5.1

[3]from this Phillip Compeau presentation that I found online.

The GUI
It can be a bit trickey to use as there is currently no GUI other then the commandpromt. Which you have to know how to operate to use the program. If you wish a GUI to be made please contact me at claeslindhardt@gmail.com and let me know. Then I will do my best to make a simpel one in HTML so the program can be operated from an Index file. in a browser tap.

2 Using the software

This part gives specifications on how to use the software developed in this article and the empiri of using it I gathered.

the best wat to acces the software is by downloading an IDE and have direct acces to the python code. I myself used Atom for coding it. Then I ran the files I made in the command promt. The idear is that you only have to change in the sections input and output to use these programs but that you do not have to code all the functions yourself.

2.1 user-manual Tsch k-mer finder

So there is 5 programs each wich can take a diffrent input and provide an output. Here is a walk trough on what wach of the programs take as input and what outpur the can generate. As well as a discription of what they might be used for

Program 1
Input: A DNA string, and the length of a kmer
Output: A kmer that might be the ori of that DNA string
it can be used for: For understanding and constructing the other 4 programs.

Program 2
Input: A genome of an ecoli, how many mutations you will allow on each kmer and the length of the kmer you want
Output: a skew diagram and a decent approximation of what kmer might code for the ori at the given length
it can be used for: Finding the Ori in Ecoli bacteria. Therby giving an idear of what to mortify

Program 3
Input: Genes as strings, the amount of genes, and the length of the diseared kmer
Output: Estimate of the ORI
it can be used for: Making a quite good but very slow estimate of the ORI

Program 4
Input: Genes as strings, the amount of genes, and the length of the diseared kmer
Output: Estimate of the ORI
it can be used for: Making a okay but very fast estimate of the ORI

Program 5
*Input:*Genes as strings, the amount of genes, and the length of the diseared kmer
*Output:*Estimate of the ORI

it can be used for: Making a quite rapid and rather decent estimate of what kmer might correspond to the ORI.

2.2 user-test respond

I order to make sure that the program was intuitive I had a person with a none science background try out the program.

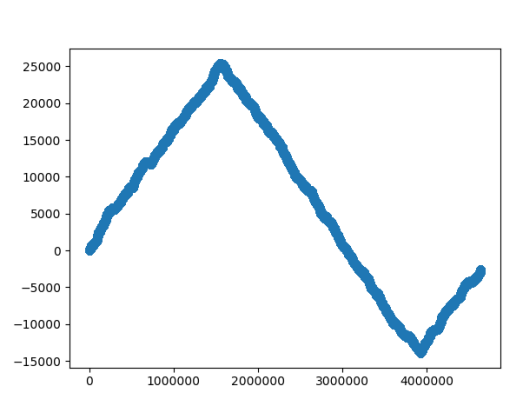
"The MotifDosRFinder program is easily accessable and requires only basic coding software. As a person with limited coding experience I found the program straightforward to use. Try this for playing around with DNA-sequences for educational purposes" - Trine Andreason

3 The Data produced

3.1 Sequencing Ecoli

EColi is a bacteria with typicallu over 4 milions nucleotides, still it is rahter small. But more importantly it is used a lot in microbiology. where you typicallu have to find its Ori expermentally, without haveing much clue to where it might be. But when searching for something a computer is usally your best friend. This case is no exception. Especially when we hace some biology background knowlagde. As I covered in the theory section4.2. We know that it is round and that from ori to terminator there is an increasing G-C diffrence and from terminator to Ori there is a decreasing G-C diffrence so if we do what is called a skew diagram we can get an overview of what these points might be. Now what is always tricky about skew diagrams is to find a way to do it efficiently. The program provided with this report can do it in a few minuts on a normal laptop.

Figure 1: A Skew Diagram made from a 4.5 nucleotide long E.Coli Genome



The skew diagram above was produced in around 2 min. and gives a pretty clear indication of what the maximum and minimum skew value is. The second program SkewScan.py can then take the minimum region and look for the most common kmer with a range of 500 nucleotides around the minimum skew value. It takes mutations into account and if you provide it with the length you think the kmer is and how many mutations it may have it can find it within minutes.

Because it only takes minutes you can try it multiple times and prestty quickly get an idear of what it might look like. Then you can hopefully do more efficient lab work. You can also use one of the functions from the first program to find the complementary string if you want to see how it looks expressed.

3.2 Tuberculosis, this softwares applications.

In 2003, researchers discovered the **dormancy survival regulator (DosR)**, a transcription factor which regulates a lot of the genes whose expression drastically changes during the hypoxic conditions. It was still not clear how DosR regulated these genes. To try to reevaluate this puzzle, bio-researchers did a DNA array experiment. They found 25 genes with expression levels that noticeably changed in hypoxic circumstances. If we take upstream regions of these genes, each of which is 250 nucleotides long, we would like to find the “sequence” that DosR uses to control the expression of these genes.

To simplify, I selected just 10 of the 25 genes. Thereby I made a DosR dataset (test.txt). Then I tried to identify the motif using the motifs function I developed in 103.py.

the last of the programs develops could with give a decent estimate to what might code for the tubercolosis in these genes. Now I only used 10 of the 25 genes as that was all I could gain acces to. You would have to look at all the genes to provide results that could be used in the fields. But the method and algorithm developed in this article seems to be able to provide such results.

4 Some background theory

4.1 Genome replication

According to wikipedia "The term k-mer typically refers to all the possible substrings of length k that are contained in a string"

It is a process essential to cell replication. Watson and Crick wrote their landmark paper on the DNA double helix with a now-famous sentence:

"It has not escaped our notice that the specific pairing we have postulated immediately suggests a possible copying mechanism for the genetic material."

They theorised that the two strands of DNA unwind during replication. Each of the parent strsnds acts as a template for the synthesis of a new strand. Therefore, the replication process starts from a pair of complementary strands of DNA and ends with two pairs of complementary strands.

The nucleotides T and A are complements of each other, as are C and G. Having one strand of DNA and a supply of “free floating” nucleotides we can imagine the synthesis of a complementary strand on a template strand. This model of replication was proven by Meselson and Stahl(1958).

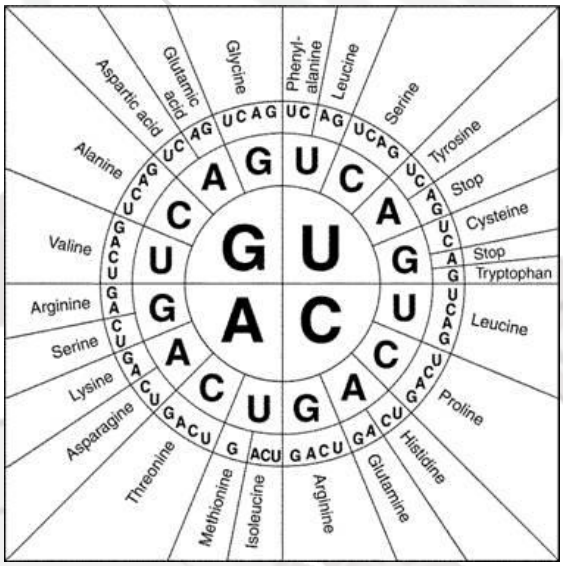
The beginning and end of a DNA shown are denoted 5’ (the five prime) and 3’ (the three prime).. Each DNA strand is read from 5’ -> 3’, and the complementary strand runs in the

opposite direction.

We think of a k-mer as a "clump" if it appears many times within a short interval of the genome.

The thing to know in Molecular Biology is "DNA makes RNA makes protein". The RNA transcript will be translated into the amino acid sequence of a protein, which performs a function in the cell
[4]

Figure 2: An overview over how RNA can form proteins



The code uses the translation of an RNA 3-sequenze (also called a codon) into one of 20 amino acids. 3 of the 64 RNA codons are stop codons, which stops translation. Each of the codon have their own letter as well.

Steve Kay used DNA arrays to show which genes in the plant Arabidopsis thaliana are activated at different times of the day(in year 2000). He then extracted the regions of close to 500 genes that showed the circadian behavior and looked for frequently appearing patterns these regions. So the ideas used in this report is far from new.

4.2 Skew Diagrams

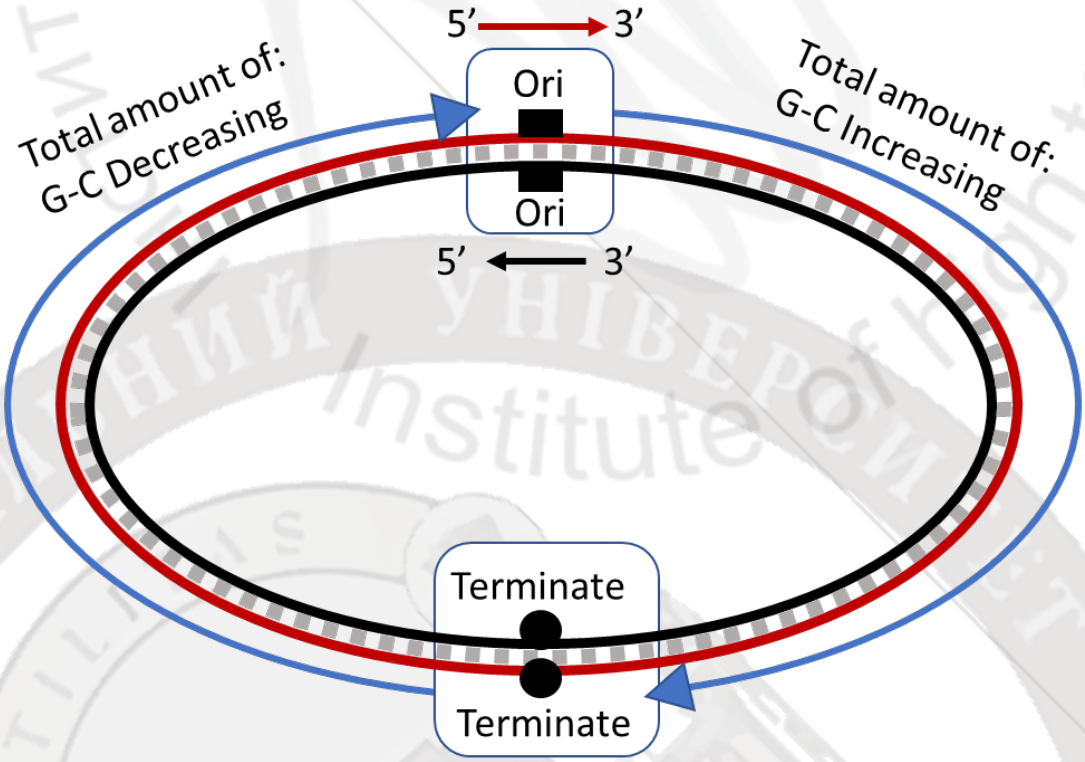
We think of a k-mer as a "clump" if it appears many times within a short interval of the genome. We might want to split large genomes into clumps of lets say 500 nucleotides as we know that the ori(orgin of replication), changes troughout the genome. By searching each blog we can find the most commonly occurring k-mer in that arrear. We want an Input: A string Genome, and integers k(how long a k-mer), L(the clump length), and t(the minimum amound of times the k-mer must appear in the clump).
and then we want the output: All distinct k-mers forming (L, t)-clumps in Genome. At first we might get a long list of k-mer which occur 3 times in 500 nucleordie clups if we try to locate ori in E. coli for example. But it is not quite time to give up yet, because it is here that the skill of

[4]Figure 2 is this Webside.

a bioinformatician comes to play in special. At this point biologists as well as software engeneers usally give up.

But we are not going to do so! We will use our knowlegde of the amount of C nucleotides occuring in typical Bacterical DNA in relation to the Amount of G Nucleotides.

Figure 3: The G-C difference over an round DNA string



If we look at the model, we can see that generally the diffrence between G and C decreases when we are on our way from terminator to ori. And increases when we are on our way from ori to terminator. If we were to make a diagram over this measure we could probably see where to search. So we want to write a program which identifies what we call a skew diagram and the takes the maximum and minimum and search that region of the sequenze. So that we only search a selected part of the gene. We want to take care of mutations as well, A way to do this is hamming distance that compares two strings and says how many differences they have. If the score is lower than a number we set on hamming distance we can say that they are the same. Then we accounted for mutation, and we can adjust it to any degree we would like.

We made a skew diagram from a e coli gene that looks like this. bla bla bla..

4.3 Tuberculosis

The Mycobacterium tuberculosis bacterium (MTB) is cause of over a million deaths every year. Although the spread of TB(Tuberculosis) have been lowered with the help of antibi-

otics, strains that resist all known treatments are appearing. MTB is successful as a pathogen because it can persist in humans for decades without causing disease. MTB can lie dormant within the host's body and may or may not reactivate at a later time. The size of potentially latent infections makes it very tricky to control TB epidemics. We are therefore interested in finding out what makes the disease latent and how MTB activates itself within a host. - This is all not very new.

It is not clear why MTB can stay active for such long periods or how it may survive during latency. Resistance of active TB to antibiotics implies that MTB can possibly have the ability to stop expression of most genes and stay 'sleeping'. We call Hibernation in bacteria Sporulation duo to the fact that many bacteria form protective and metabolically 'sleeping' spores that can survive in very rough conditions.

Latent MTB is often associated with Hypoxia, or oxygen shortage. Researchers have found that MTB becomes dormant in low-oxygen environments, maybe with the trick that the lungs the host will recover enough and spread the disease later. As MTB have a seemingly remarkable ability to survive for years without oxygen, it seems an advantage if we were to identify the genes responsible for the incitations of the latent state under hypoxic conditions. Biologists might be interested in finding a transcription factor that "feels" the shortage of oxygen and commence a genetic program which affects the expression of many genes, allowing MTB to adapt to hypoxia.this is not only true for MTB[5]. But this article looks at MTB as there is good datasets to find on the internet on MTB, which we can use our software on.

4.4 Statistics and randomness

So as we need to use a **random search**, we should also explain what it is. If we have our motifs, that we have gotten by finding the most common k-mers with a certain length and mutation.

Motifs			
T	C	G	G
c	C	G	G
a	C	G	G
T	t	G	G
a	a	G	G
T	t	G	G

Then we can make a count of the nucleotide, that occurs most times per row. If we then divide that by the number of rows we get the profile or the probability of each nucleotide taking up that position in the row

Counted				
A:	2	1	0	0
T:	3	2	0	0
G:	0	0	6	6
G:	1	3	0	0

Prob. Profile				
A:	2/6	1/6	0/6	0/6
T:	3/6	2/6	0/6	0/6
G:	0/6	0/6	6/6	6/6
G:	1/6	3/6	0/6	0/6

[5]Clinical practice: Herpes zoster., https://www.ncbi.nlm.nih.gov/pubmed/23863052 and https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4789101/

As you can see some of our probabilties are now 0 which is going to be a problem if we want to give them a score later on. In the 18'th Century, mathematician Pierre-Simon Laplace gave an estimation on the probability that the sun will not rise tomorrow. His estimate was 1/1826251. Back in the 18'th century it is possible that people did not take him to seriously. However they did not do random searches as much as programmers today have to do. We discovered that we generated quite a few possibilities of 0, when we did the random search. This might become a problem. To fix this we use **Laplace's Rule of Succession**. In the our case, we will make a pseudocount. By simply adding 1 (or some other small number) to each element of our count. In this way we will have a small chance of the events that had a probability of 0 before

Now that we have solved our problem of 0 chances we can get back to randomnizing our algorithm. Okay, so if we imagine that we now discard our precious previous algorithm that gives a pretty good estimate of the k-mer og a certain length and now instead just select random k-mers with that length, then we can find their score and profile. At first this will not give us a very good result as we selected something completely random. But if you look closely at the previous figure you can see that we can actually reconstruct the motifs from the profile and the profile from the motifs. So we can now iterate between the two, we can write that in pseudo code.

```
def MotifDosFinderRandom(DNA, k, t,N):
        Randomly select K–mer motifs.in each string
    findScore()

    while(i>N):
        Profile =findProfile(Motifs)
        Motifs = findMotifs(Profile, DNA)
                if findScore(Motifs) > findScore(bestMotifs)
                bestMotifs = Motifs
```

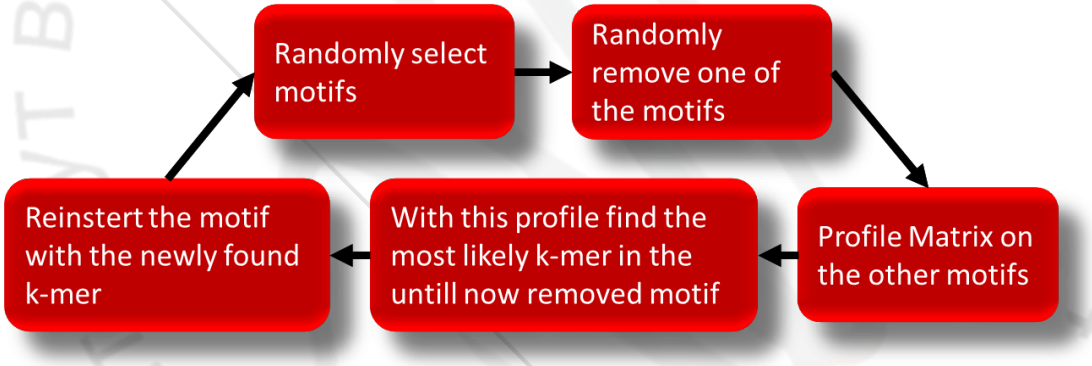
The idea is to keep iterating like findMotifs(findProfile(findMotifs(findProfile(Motifs), DNA)), DNA) until we get a high enough score instead of saying it has to iterate a certain amount of times. You could also set it to iterate until you find a certain score. However then runtime will be a bit harder to predict. When you have a certain number of iterations you can still use the big O'notation.

So how the hell can we find anything useful, when we started with something completely random? Well part of the truth is that it is not completely random, as DNA strings are not completely random, so when we select k-mers randomly it results in a uniform expected biased profile. This biased profile should correspond somewhat to the k-mer we are looking for
Gibbs sampling, is what we finally used for this pogram. Now why would we use Gibbs sampling when we have a working randomized algorithm?. Because it is more 'soft'. When it comes to selecting it only changes one of the selected motives per iterations. With the randomized search algorithm there is a chance that we come very close to the correct result and then disgard it, because the random motifs search can process so fast.

So what is Gibbs sampling?

As this graphic shows, it is a way of finding k-mers one motif at the time. As you iterate through this loop the k-mers found in each motif will be found with larger and larger certainty. But there is a very little chance of it throwing away something which is close to something right.

Figure 4: A model of the Gibbs Sampler algorithm



At the same time it does not take to long to process a Gibbs search. So we can do it on our own laptops.

However, even this algorithm have its limits, and should be used along side ChIP sequenzing and the likes to assure that it is not applied on a too long string of nucleotides.

The Gibbs sampler might find a local optimum solution, which is in contrast to a global optimum. Since GibbsSampler explores only a minor subset of solutions, it can possibly get caught up in a local optimum. Therefore, we should run it more than one time.

5 Appendix

5.1 Function Map

Here you will see a description of all the functions that where made in this program. the code made in this article was made in python and the pesudo code is affected a little by that. For example when you see def and then indented lines it just means that it is a function much like in matematics you give it an input, it then does something with that and returns an output. So after def you will see the name I gave the function in my code.

5.1.1 Program 1

We need 5 functions for this program.
This first function finds how often a given k-mer occurs in a text

```
#Input:1: A string with the DNA sequenze, eg. AATTGA
#2: the DNA sequenze we want to count for, eg. AAT
#OutPut: The amount of times that input 2 occurs in input 1
def CountPatterns(Input1, Input2):
        set variable count to 0
        for every sequenze with the lenght of input 2 in input 1:
                if i is equal to input 2:
                Make count one bigger
    return count
```

This function counts how many times the same k-mer of a given length occurs.

```
#Input:1: A string with the DNA sequenze
#2: k-mer length
#OutPut: a dictonary with how often kmers occur in your input sequenze
def Occurences(Input1, Input2):
    Make a dictonary that can store a k-mer as a key and how
    often it occurs a value.

    for every sequenze of a certain length in input 1:
        if the sequenze = a key in count:
                add one to the value of that key
    return the dictionary
```

Now we want a function which gives us all the k-mers of a given length which occurs the most times.

```
#Input:1: A string with the DNA sequenze
#2: k-mer length
#OutPut: the kmers which occurs the most times
def MostOccurentkmers(Input1, Input2):
    Make a list []
    call Occurences(give it Input1, Input2 as arguments )
    Find the maximum value of the values Occurences
    Iterate trhough the dict Occurences returns:
        if the value to a key = the maximum value:
                append that key to the list you made
    return the list you made
```

This function reverse a string and gives you the complmentory string.

```
#Input: A sequenze as a string
#OutPut: a reversed string with complementory nucleotides expressed
def ComplementAndReverse(Input1):
    revPattern =reverse the pattern.
    for each nucleotide/letter in Input1:
        if letter == 'A':
            revPattern.append('T')
        else if letter =='T':
            revPattern.append('A')
        else if letter == 'G':
            revPattern.append('C')
        else if letter == 'C':
            revPattern.append('G')
    return revPattern
```

This function tells you where the pattern you insert occurs in a sequence.

```
#Input:1: A string with the DNA sequenze
#2: a k-mer
#OutPut: the kmers which occurs the most times
def MatchPattern(Input1, Input2):
```

```
    make a list (positions = [])

    for every sequenze of a certain length in input 1:
        if the sequnze of that length = input 2:
                Append the starting positon of the kmer to the
            list you made
    return the list you made(positions)
```

5.1.2 Program 2

To make this program we are going to need 8 functions. The first one will be for importing the DNA sequenze of the gene and the last one for plotting it. Now these are done very diffrent in different programming languages so it will be easier for people to google how to do that depending on language, so we will only go trough 6 functions. Note however, when plotting the length of the sequenze, the x axes and the y axes should be the G-C difference. One of the functions for this program is used in the last one as well, so to see CountPattern watch the pseudocode for the last program

This first function should find how many times a given nucletide appears.

```
#Input:1: A string of a Genome
#2: The symbol to look for
#OutPut: the kmers which occurs the most times
def nucCounter(Genome, symbol):
    -Make a dict for when you call CountPattern
    -make a variable n = to the length of input 1 the Genome
    -Make an ExtedGenome where you add the first half of the genome
    to the end of the genome


    Call PatternCount(with input 2, and input 1)

    for i in range(1, to n):
        -Set the current dict value = to the previous
        array[i] = array[i-1]

        if ExtendGenome[i-1] == symbol:
            array[i] = array[i]-1
        if ExtendGenome[i+(n//2)-1] == symbol:
            array[i] = array[i]+1
    return array
```

This function calculates all of your skew values

```
#Input:1: A string of a Genome
#OutPut: A list with skew values
def SkewCal(Genome):
        -Make a list to store the skew values
    -Append 0 as the first element of the list.
```

```
    #Iterate over the length of the whole genome
    for i in range(len(Genome)):
        #print('i-1:',skew[i])
        if Genome[i]=='A' or Genome[i] == 'T':
            Append the same value as the prviously added in the list
        else if Genome[i]=='G':
            Append the prviously added value in the list +1
        else if Genome[i]=='C':
            Append the prviously added value in the list -1
    return
```

This following function finds the minimum skew

```
#Input:1: A string of a Genome
#OutPut: A list with minimum skew values
def MinimumSkew(Genome):
    make a list to store the output
    call the SkewCal()
    find the minimum values from the list Skew Cal returns
    and add them to a list
    return the list with minimum skew values
```

The following function calculates the hamming distance

```
#Input:1: The first of the string you want to compare
#2: The second of the string you want to compare
#OutPut: the kmers which occurs the most times
def HammingDist(Input 1, input 2):
    Make a hamming distance variable and set it to 0

    for i in the length of one of the two inputs:
        if element i in input 1 is not = to elment i in input 2:
            make the hamming distance variable 1 bigger
    return the hamming distance variable
```

This final function of this program make an approximation of the ori kmer within the hamming distance

```
#Input:1: A string of a Genome
#2:the Pattern to compare with
#3: the accepted hamming distance
#OutPut: the kmers which occurs the most times
def EstimateMatch(Text, Pattern, d):
   make a list that you can add the positions of the hamming matches to
    n = len(Text)
    k = len(Pattern)
    for i in range(n-k+1):#itereate trough the genome looking at each
    #potential patter
        if HammingDist(Input 2, the potential part of your genome) <= d:
            then append it to the list you made to start with
    return the list with positions
```

5.1.3 Program 3

This program shall eventually could find the ori only given the DNA, the length of the kmer and how many DNA strings there is to find a kmer in. for that we need the following functions. If a function is being called in this program that you cannot find try to look in one of the other programs.

The first function count how often a letter occurs in a position of the kmers given as input.

```
# Input:  A set of kmer and thier Motifs
# Output: An overviev over how many times a nucleotide holds a position
def Count(Input):
    -Make a dictionary
    -Give it the four nucleotides as keys(ATGC)

    -make a list for each key with the length of the input kmers
    use 0 as a placeholder

    for loop that iterates trough all of the kmers
        letter for letter:
        if A letter is equal to a key then add 1 to the position
        of the list which holds the same position in the list
    return the dictonary that you have made.
```

This function find the profile from a set of motifs

```
# Input:  A list of kmers Motifs
# Output: the profile matrix of Motifs, as a dictionary of lists.
def Profile(Motifs):
    Call Count(with motifs)
        then divide each element in all of the value lists
    in the dictonary with the total amount of motif

    return the updated dictonary
```

The following function finds the kmer that would make most sense to compose from the profile of some motifs.

```
# Input:  A set of kmers Motifs
# Output: A consensus string of Motifs.
def Compus(Motifs):
        make a string we can store the final kmer in
    call the Count(Motifs) function

    Loop trough the count values
        if a valueble is bigger then any other for its position:
                append that letter to our string
    return the string kmer we have mede
```

the next function will give os a score we can use as an indication on how simmilar our diffrent kmers are. A lot like the hamming distance that was done in programe 2. It can be a bit tricky to wrap ones head around how the score is calculated so I made this tabular in the hopes it woud

seem more intuitive. the lowest row being the score for each coloum. To get the total score of all the motifs we simply add that up.

Motifs			
T	C	G	G
C	C	T	G
A	C	G	C
T	T	G	G
T	A	A	G
2	2	1	1

```
# Input:  A set of k-mers Motifs
# Output: The score of these k-mers.
def Score(Motifs):
    inisiate the variable score = 0
    call the Compos(Motifs) function
    iterare trough every string in motifs letter by letter
            if Motifs[i] is not = to the consensus[j] string:
                add one to score
    return score
```

Now istead of just calulating the profile, we will calculate the the probability of a kmer string from a profile

```
# Input:  String Text and profile matrix Profile
# Output: Prob(Text, Profile)
def Prob(Text, Profile):
    initiate variable p = 1
    range through characters:
    find the largest possability in each coloumb
        multiply that by our iniciated variable and update p
    return p
```

so If we have a kmerstring, the length of it, and a profile we need a function to tell us what kmer would be the most probable that is what this function is fore.

```
# Input:  String Text, an integer k, and profile matrix Profile
# Output: ProfileMostProbablePattern(Text, k, Profile)
def ProfileMostProbablePattern(Text, k, Profile):
        initiate a variable BestKmer
        loop trough the input text
        call the FrequencyMap(Text, k)
        find the maximum value in the frequency map
        take out the kmer with the maximum value
        if the the kmer found have a higher value then prvious best value
                set the current kmer = BestKmer
    return BestKmer
```