NTNU
Norwegian University of
Science and Technology

Finding an Optimal Approach for Indexing DNA Sequences

Anders Brujordet

Master of Science in Informatics
Submission date: May 2010
Supervisor: Herindrasana Ramampiaro, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

There has been a lot of research and magnificent breakthroughs in the field of DNA since its discovery in 1953 by Francis H. C. Crick and James D. Watson. We have been able to gather information about many species and and their genes. We have seen the amount of available DNA grow to staggering heights in a relatively short amount of time. The extraction or sequencing of this data has gone from being a very laborious task to being performed by automated machines.

At one point we ended up with enormous amounts of data but not very good techniques for organizing and searching through this data. There has been several breakthroughs in regards to these problems both in speed and in precision. However there is more to be done and this thesis discusses, implements and tests a proposed approach to build a solution that addresses these challenges. This thesis will look at a technique that is simple but elegant and should aid in the quest of utilizing the vast amounts of knowledge that are stored in the already gather DNA.

We discover that our approach is indeed a valid approach but to truly aid in this quest for being able to fully search and utilize DNA further work is required, we have however laid the grounds for a precise, covering and fast DNA search system.

Contents

List of Figures

List of Tables

Acknowledgements

First off all I would like to thank my girlfriend Rutt Kine for steady support and motivation through a tough but immensly informative year which has allowed me to grow as a researcher and improve as a developer. I would also like to thank my fellow classmates at the Norwegian University of Science and Technology who have also been working within the field of bio-informatics. They have been a great source of inspiration and support. A specially I wish to thank my supervisor Heri Ramampiaro. His knowledge and experience in this field has been an invaluable resource and served as a guidance in the research and development of this approach.

Chapter 1

Introduction

1.1 The problem

Since the beginning of time, the human race has strived to understand the world and the creatures within it. One of the biggest mysteries has been our own slow but steady creation attributed to the art of evolution. A few decades a go The Human Genome Project[12] set out to decode the entire human genome. That is, to find all the building blocks needed to create a human being, which they did in 2003.

This is very valuable information, but the problem arises when trying to search and understand this data. First of all, the amount of data is extremely large, there is also a large amount of scrap data surrounding the useful data that we want. This is because as the DNA has developed there has been a lot of insertions, deletions and alterations. This has left damaged or uncomplete genes that are now switched off. In some cases though, this data might be valuable in regards to how the species in question have developed. Data-replication is also very common and makes it harder to understand what data is useful, and what is not. To be able to understand this data these problems must be addressed.

In the last decades DNA and its applications has received wast amounts of attention. Partly for its potential attribution in medicine and specific health improvement, but also for its very helpful characteristics regarding criminal investigations. Additionally there are probably countless undiscovered uses for the analysis of DNA regarding the science of evolution and the understanding of life it self. Imagine in awe or horror what the results would be if we could create species for specialized purposes or grow entire arms, legs or eyes for transplantations whenever and where ever we want.

The techniques used today are for the most part full text searching with modifications, which is very accurate, but not as fast as we would wish. These techniques are not optimal for approximated searches which is very important in DNA since it tends to evolve in different species, groups, and individual organisms. In a thesis by Aleksander Grande[8] it is shown that it is possible to use techniques from Content Based Information Retrieval[10] to index and search through DNA by generating images from DNA and searching through these. In this paper we will look at the data gathered by Grande and try to see if we can use n-grams to get the same or better results as he did without the cost of generating images. We will off course go more into detail on both our approaches in this thesis but it is safe to say that his approach has been one of our inspirations.

1.1.1 Motivations for improving DNA-search

DNA as evidence

During an investigation of a crime scene DNA can be used to find out where one person has been. Not long ago, blood on a crime screen only proved one thing; That someone had bled there. Now we can use DNA to pinpoint the person who bled. However this process takes about 30 days[15], from the day it is submitted. Now this is a very critical period in the investigation, where time is of the essence. If we could help speed up this process it could mean the difference between solving a crime and letting a criminal get away.

Research on evolution

If we could employ similarity searches on DNA we could look at the evolving genes in different species. Furthermore, we could find relations between different species and track down the origin of current genes that may have become obsolete. This will also help convince those who still believe that evolution is just a hypothesis and it would show how we relate to the rest of the world.

Pre-diagnosis

If we could analyze, search and compare DNA in a fast and simple way, there would be huge advantages in finding the genes that make some people more vulnerable to some diseases then others. This would enable doctors to start treatment or take preventive measures so that the patients may never become ill of the oncoming sickness. There is no doubt that this would prolong human life as far as dying from pre-disposed diseases goes.

Paternity test

We have all heard stories of men raising a child as their own only to find out that what they thought was their own child was in fact a result of an affair by the wife. This is an issue that is not very common to talk about, one way to help this situation would be to make paternity tests easy, cheap ,fast and thus more accepted.

Identification

If these searching techniques evolved even more, we could see DNA as a way of identifying a person. The biggest problem in this aspect would be to be able to analyze the DNA fast enough. Say to within a few seconds to be tolerable at the airport for instance. The biggest problem in this instance would be to overcome the problem of identical twins, they can actually have the exact same DNA.

Charting human migration

If we could utilize all the information within the DNA of large groups of people over large land areas we could be able to map human migration better then today. In this way we would learn a lot about how we evolved and adapted to different climates and locations. This approach could also be employed in animal research. In other words, within DNA lies many of the answers we have been seeking for centuries.

The biological factory

A darker side of DNA-research is concerned with growing limbs and creating specialized species and bacteria for certain purposes. Since we will not divulge into the moral aspects of these motivations we will simply note that this is also a branch that would benefit of better ways to search and deal with DNA.

1.2 Problem spesification

In 2009 Aleksander Grande wrote "A Multimedia Approach to Medical Information Retrieval"[8] and asked two major questions.

1. Is it possible to translate a DNA sequence to an image?
2. Is it possible to use this image in a content based information retrieval system to find DNA sequences.

Grande found that is is in fact possible to translate a DNA sequence into an image, and also managed to use these images in an CBIR setting to search through the DNA sequences as images. The drawback was that his approach was a bit expensive and not as fast or accurate as the search systems already available. We want to find out if it is feasible to use his approach in a way that could compete with the state of the art search algorithms for DNA currently available. However, we have seen that what Grande actually did was to utilize a Fuzzy-Search, which has shown to be very useful when searching through DNA. We will go more into detail later, but for now it can be explained as a search that retrieves documents that are not necessarily identical to the search term, but similar. To lay the foundation for this we will try to find the best way to utilize n-grams when creating our index. We also realized that there is not much to gain using images compared to simple text-search which is much cheaper. In addition we will look at DNA annotations, which are names and/or qualities of the proteins within the DNA, to filter the search. We will dive deeper into our approach later, when we have established some of the necessary background information. For now we will state the main questions in this thesis as the following:

1. How can we utilize n-grams to search through DNA.
2. Can this approach compete with the state of the art technology for searching through DNA?

1.3 Our scope

Since the questions posed by Grande have already been confirmed we will not question the possibility of his approach. On the other hand, n-gram-searching will be our main concern. As mentioned we will compare the performance of our approach to the state of the art DNA searching tools, and hope to be able to compete with these. If this approach is successfull it might be a very good starting point for doing evolutionary searching in DNA. That is to find links between species and genes. However this will be outside the scope of this thesis. Furthermore we will provide advice as to the use of annotations but this will not be a primary part of our approach. It is also usefull to note that our approach is not ment as a full implementation of a commercially benefitial system. On the other hand we wish to lay the foundations for a better way of searching

through DNA. The creation of the index using n-grams is only the first step in a long process which will require a lot of testing and tweaking to be able to compete fully with the commercially availeble DNA search systems. We will describe how we can compare our approach to these and point to improvements to our approach.

1.4 The author

I am with this thesis concluding my masters degree in information technology and I am guided by my supervisor Professor Heri Ramampiaro and thus will refer to the authors in plural. As mentioned this thesis is part of a masters degree and is written in the course of two university semesters at the Dept. of Computer and Information Science (IDI) with the Norwegian University of Science and Technology.

1.5 The structure of this thesis

Chapter 1

Chapter 1 contains the introduction and motivation for the thesis. It defines the scope and specifies the problem. We also show some of the possible applications of DNA searching and explain the intents of our approach.

Chapter 2

In this chapter we will provide the necessary theoretical knowledge to understand the rest of the thesis. It will for the most part just scratch the surface of the topics concerning our approach. We will also refer to articles and books going deeper into the subjects if the reader wants to get more information of the topics covered.

Chapter 3

Chapter 3 will take a look at the state of the art in DNA search systems. The major systems and other approaches that either relate directly to or has inspired our approach. We will also have a look at the technologies that we have based our research on or that have been used in our solution.

Chapter 4

This chapter will show how we implemented our approach and take a closer look at our algorithms. How we have chosen to tackle the challenges of searching through DNA. What problems we have met during the development and how we solved them. We will also share some of the code for these algorithms.

Chapter 5

Here we will look at our results. How good our searches are, and how they compare to each other and the state of the art DNA search systems. We will be going into detail and looking at the implications of our findings and how they will effect further work.

Chapter 6

At last, we will draw a conclusion based on the results found in chapter 5. We will also look at the tests and analyze the implications in regards to what can be improved and what should be researched in the future and summarize our thesis.

Chapter 2

Background research

In this chapter we will give the information necessary for the reader to fully understand the key elements in this thesis. We will give references to cited papers and articles for a more in depth reading and understanding of the topics and terms used. However this basic introduction to these topics should suffice. To be able to understand the concept of n-gram search we first need to look at information retrieval and what lies beneath this indexing method. Since this thesis concerns DNA and the biology surrounding it, we will give a quick introduction to this topic as well as having a brief look at Content Based Information Retrieval (CBIR).

2.1 Information retrieval

The term Information Retrieval(IR) is very broad and can be defined in several ways. Even pulling up your cellphone to find a phone number can be described as IR. But in our context the definition from the Cambridge University Press fits very well[10].

> "Information Retrieval(IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)"

What this means to us, is that we give input to a search system, and receive relevant output. For instance we supply a part of a DNA sequence of a protein found in dogs and receive the actual gene in return. Or we could supply DNA of a suspected criminal, and receive information stored about this individual within a DNA database. To summarize we have input, a search process, and output. Unknown to the user, the DNA in the search system is first indexed and prepared for searching. The naive way of doing this is to just search through every item in the search system and look for the search query. However this would take an enormous amount of time and resources. Instead we create indexes from the items in the search-system. For instance we could create an index of words, and each word in the index would point to a paper or article containing that word. We will look at all of this in more detail in the following chapter.

2.1.1 What is a document

To understand the process of searching we need to define the items within the search system. In information retrieval, this is the document and has been described as having the following characteristics[2].

1. There is materiality: Physical objects and physical signs only;
2. There is intentionality: It is intended that the object be treated as evidence;
3. The objects have to be processed: They have to be made into documents; and, we think,
4. There is a phenomenological position: The object is perceived to be a document.

This tells us that a document does not at all need to be a piece of paper. It should be an object, that documents something. So for instance the Aurora Borealis, or Northern light, is not a document. A picture of it however, is a document. To further the point, a lion is not a document, but a lion presented in a zoo is. In other words, a document has to not only document something, but the documentation has to be intentional in the words most naive way. In our context however, a document is defined as a document Id and a sequence of DNA containing an assortment of nucleotides in some meaningful order. From this we conclude that the intention of information retrieval is to find and retrieve the documents a user wants which is expressed by a query.

2.1.2 Document indexing

A naive approach would be to store many documents in a folder, and when someone searches for "evolution", go through all the documents in the folder and return those documents containing the word "evolution". Not only would it be resource demanding and time consuming but it would get worse for every document added to the folder. Instead we use an index to tell us what words or terms occur in which documents. A common data structure for storing this information is an inverted index. (see table 2.1) The terms are listed alphabetically, and the document within which the term occurs are listed ascending to the right. It is also common to store information regarding how often the term occurs in a given document. Additionally it is useful to know how often the term occurs in the entire document collection. This gives us an idea how important this term should be for selecting what documents to retrieve. If this term occurs in say 90% of all the documents, it does not really help us in selecting good documents to retrieve. Terms that occur very often and give little help with filtering out relevant documents are called stop-words. Since they do not give us very much information, we simple remove or ignore them when creating the index. Accents and spacings are also usually removed, since they give us very little information and can actually make redundant entries in the index.

Term	**Documents**
Africa	{1, 2, 6, 14, 16}
Banana	{1, 5, 6, 15, 19}
Darwin	{3, 4, 24, 26}
Dawkins	{3, 4, 22, 28, 29,36}
Fish	{1, 2}
Pizza	{7, 8, 9, 27, 29}

Table 2.1: An inverted index

To summarize we create the index by saving each term that is found within the document collection in a fast and simple data structure where we also store information

about within in which documents a term is found. This makes perfect sense when we want a fast and reliable way of searching through many documents and are looking for single terms. There are certainly other ways to organize the index. For instance tree and map structures, and even hash tables would speed up the search. For our purposes however the above description will suffice.

Improving the index

There are several ways of improving the index besides alternating the data structure for storing it. One way of making it more covering is to use stemming, and a widely used variation is Porter stemming.[1] Stemming helps the search engine to not only find exact matches to the search term, but also related terms. This is done by removing the prefixes and suffixes of the word. For instance horses becomes horse. Removing the prefix is not very common in english, but in german the derived form of a verb has a "ge" in front of it such as the derived form of "finden" which is "gefunden". Stemming is very effective but has one drawback, for instance pennies would be stemmed to penni, which should have been penny. Likewise gefunden becomes funden, but should have been finden. Lemmatization[1] however usually complies to doing things the right way. The only drawback with lemmatization would be that it makes the index creation slower because it will require more computing.

The terms that we are left with are stored as tokens in the index, we use a tokenizer or a token filter to generate tokens from the document collection and these tokenizers can use many features like stemming, stop-words, lemmatization and so on. When the documents have been tokenized we need a reasonable way of storing the index. There are several ways to do this, for instance inverted files, suffix arrays, and signature files. The way theses techniques work is by making terms available and pointing to the documents where they can be found. The specifics of these different techniques will not be relevant to understanding our problem and will therefore not be discussed but an example of an inverted index is shown in table 2.1.

As mentioned stop-words are also removed to reduce the index size and thus the search time. The argument for removing stop-words is simple. If you are searching for "Oil is money" and 99% of all the documents contain the term "is" then searching for this term will not help you to find the top most relevant documents. In stead the you should focus on "Oil" and "money" who for arguments sake are each just contained in 5% of all the documents. This will refine your search a lot more and the result with and without the term "is" will be close to the same, but the cost for including it would be higher. Thus, we remove stop-words.

To summarize this section the process of creating a text based index can be expressed in a 7 step algorithm[1] shown in table 2.2. For illustration of the document indexing process see figure 2.1:

N-Grams

Being the crucial part of our approach, n-grams are very relevant. The process of creating these is very straight forward. If we have the word "linux" and we create n-grams where $n = 3$ we get the tokens: "lin", "inu", "nux". Now for indexing regular text this is seldom the most efficient way since words can be of average size and each letter can have more then 26 different values if we ignore symbols and notations. This leaves us with a 4-gram that can have $26 * 26 * 26 * 26 = 456976$ different values, which is huge. DNA however have only 4 values, "A C T G" so we get only $4 * 4 * 4 * 4 = 256$

1	Add a document to the collection.
2	Extract the structural properties of the document and information about these structures.
3	Remove accents and spacings.
4	Remove stop-words.
5	Identify nouns to provide increased information in the index.
6	Reduce words to their grammatical root by stemming or lemmatization.
7	Now from the information gathered, create a searchable index.

Table 2.2: Basic indexing algorithm

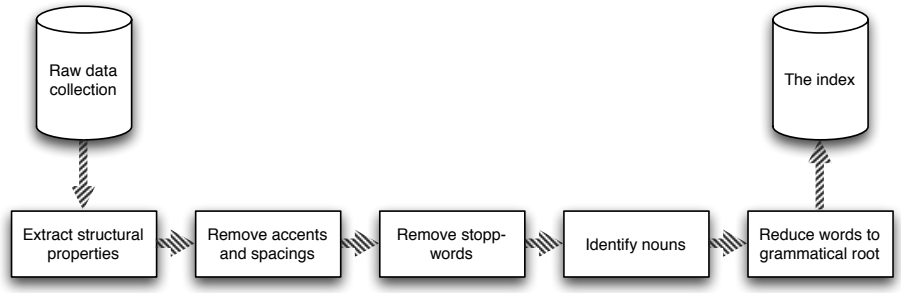

Figure 2.1: Standard Indexing

different values for a 4-gram. To summarize, what the n-gram actually does is to split up the entire text that we are indexing into grams of the size of n. So there will be very few complete words within the index. This means that we can not search in the exact same way that we would with a regular word-based index. Instead we must prepare the query to mach the syntax of the index. In other words, we must use an n-gram filter on the search query so that it can be compared to the terms in the index. However this is a small price to pay since the index itself will be very small when indexing DNA with n-grams as showed above.

This makes the n-gram very interesting for indexing DNA. And a specially the effect of different n-gram sizes is very interesting and will be looked at and discussed throughout this thesis. We will therefor test different sizes of n in the n-grams and try to establish how the size of the n-gram effects the overall performance of the search system. Since others have looked at n-grams in the context of DNA we will also show the differences between our approaches and compare our results.

2.1.3 Queries

Keyword query

We should also have a brief look at the way a search is handled. Only after the index has been created can a search query be posed. There are three query-languages that are widely used. The most common in public search engines like Google and Yahoo is keyword queries. The users presents one or more keywords which are run through the index where the appropriate documents are retrieved for each query. Then the results from each query search are AND'ed together so only results that contain both

queries are returned. This is however the naive approach. Before the query is processed however, an optimization is performed to ensure that the query does not result in work that does not have to be done. Like searching for two terms twice. (see figure 2.2) This query-language is very useful when you do not really know what you are searching for, such as when you are looking for news concerning a product and only want general information conserning this product.

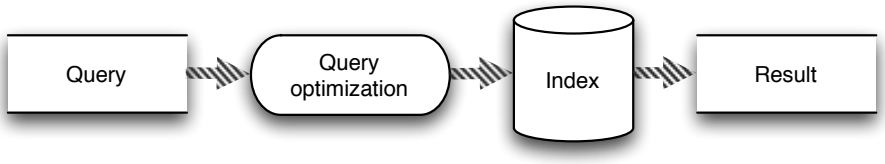

Figure 2.2: Query process

Context query

A context-query is used when the user wants to specify in what context a word appears. There are to variants of this query, *prashe-query* and *proximity-query*. A phrase-query is performed by not only matching words, but by matching them in an exact context to each other, such as "How to loose 10 pounds in one week". This is very useful if you know exactly what you are searching for. Proximity-queries are more defined. For instance you can search for "loose NEAR weight" this will return documents where "loose" is within some predefined distance from "weight". This on the other hand is useful if you have some idea of what you are searching for.

Boolean query

Boolean-queries are also very effective when the user is very familiar with the kinds of documents in the collection, and are therefore used in scientific collections for instance. The user will specify some terms separated by boolean operators such as this; "peter AND jackson NOT michael". This query will retrieve documents containing Peter and Jackson but exclude the once containing Michael. We can easily see which documents that are relevant from table 2.3.

Document	**peter**	**jackson**	**!michael**	**relevant**
African Artist	0	1	1	0
Lord of the rings	1	1	0	1
District 9	1	1	0	1
King Kong	1	1	0	1
Olsenbanden	0	0	0	0

Table 2.3: Boolean index

Natural language has also been used employed lately. One of the more peculiar uses of this concept is BussTuc[13]. we gave BussTuc the search phrase "When does the next buss for Lade leave Samfundet?" the answer was both accurate and impressive.

> "The station nearest to Lade is Lade allé 80 . Bus 4 passes by Studentersamfundet 1 at 4:12 pm and at 4:31 pm and arrives at Lade allé 80 , 32-33 minutes later . Bus 3 passes by Studentersamfundet 1 at 4:26 pm and arrives at Lade allé 80 , 32 minutes later . "

This may seem very intelligent, but what actually happens is that BussTuc translates the natural language into a query language such as SQL, and then runs the query against the index, stored in a database.[1]

2.1.4 Evaluation of information retrieval systems

It is crucial that we discuss how an information retrieval system is evaluated so that we can compare our solution to existing technologies. What we want to measure, is how good we are at providing the user with the documents he[2] is looking for. To figure out how well our search system works we will test it against *test collections*. These are document collections that are created to be used for testing search systems. A test collections will contain documents and a documentation of pre-made queries and what should be the respective result for these queries. We then test the queries in our own search system on these test collections and compare the results to the ones that are said to be "correct" according to the test collections's documentation.

Precision and recall

A short description would be that precision is retrieved and relevant terms divided by the retrieved terms, and recall would be the retrieved and relevant terms divided by the relevant terms (see figure 2.3) but lets go a little more in depth[1].

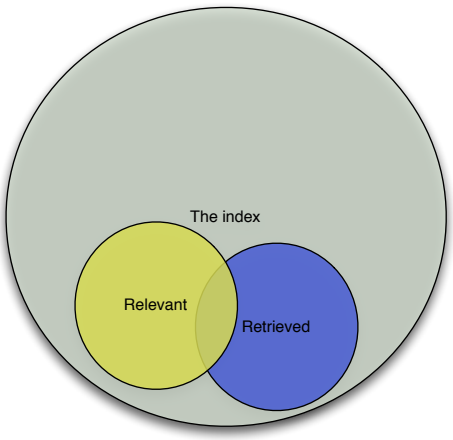

Figure 2.3: Recall and precision

Precision is used to define how accurate a search is. That is to say, how many of the documents retrieved are actually relevant to the query posed by the user. Precision is therefore usually expressed as a percentage of the retrieved documents which are relevant. If we let $|D|$ be the number retrieved documents $|R_R|$ will be the number of

[1] SQL is a language used for querying relational databases.

[2] We will throughout this paper refer to people in general as the masculine form for simplicity

documents retrieved that are relevant to our query. This gives us $Precision = \frac{|R_R|}{|D|}$ If precision is high it means that the documents we have collected are relevant to the query. A low precision value however tells us that we are retrieving unrelevant documents, that is, false positives.

Recall is used to define coverage of our search. That is to say, how many of the relevant documents in the collection that are actually retrieved. If we define the number of relevant documents retrieved as $|D_R|$. The number of relevant documents in the entire collections is denoted as $|R|$. Recall is then defined as $Recall = \frac{|D_R|}{|R|}$. If recall is high, we are finding most of the relevant documents. A low recall value however tells us that we are missing a lot of relevant documents.

F-measure is used to find the relationship between precision and recall. It will provide a weighted average between the two and is defined as $f = \frac{2(precision*recall)}{(precision+recall)}$. The argument for using this measure is that is gives a single point of evaluation instead of two separate indicators.

Fuzzy-Search

Now it is important to note that in some cases we might not know exactly what we are looking for. Say that we are searching for a gene found in humans, but we are searching through the DNA of other primates. Now this could be an excellent way of finding genes that have been subjected to evolution and changed slightly over the years from our fellow ancestor. In this case we would not be looking for exact matches at all, we would be looking for genes that are similar to human gene. For instance if you google[3] you will often be asked "Did you mean …", where the search engine guesses that you either misspelled the word or used it in the wrong context. That is a great benefit of fuzzy-searching[10], it allows you to make mistakes and helps you on the way towards your goal. We will not, however, go to deep into fuzzy-search but this would be an natural way to go after defining the best ways to utilize n-grams.

Index

The index size and search-time will also be important to evaluate the information retrieval system. In this thesis we will look at these two values but we will primarily focus on how our approach will scale, not the exact values. This is also in part true for the precision and recall but they will however be important when evaluating the approach as a whole, not only in scalability. It will also be interesting to see how many of the possible terms or grams will be used in regards to that there are a finite number of terms for each n-gram in our case since we are only indexing DNA with n-grams. It is very important that the index does not become larger then expected, this would indicate that something is wrong with the n-gram indexing process.

2.2 Content based information retrieval

We will briefly discuss the concept of Content Based Information Retrieval(CBIR) for the sake of argument since we are building some of our thought and ideas on the previous approach by Grande that utilized CBIR. The techniques from CBIR are usually employed in the search for sounds, pictures or the combination of the two, that is, video. Now the only part of CBIR that will affect our thesis is searching with pictures so we

[3]That is search the web using Google's search engine.

will ignore sound and video. CBIR is in many ways very similar to regular text-search, however since there are few words in pictures we need something else to create terms in our index. Instead of words we extract features from the picture.

2.2.1 Feature extraction

Feature extraction is the concept of using features in the picture to distinguish it from the rest and thus make the pictures available for searching. One common feature, *histograms*, will describe the distribution of different colors in a picture. One way of doing this is to divide the picture in tiles, and let the most common color in one tile represent the color of that tile. A major problem with this approach is that it does not take into account the spatiality of pictures and it will render two identical pictures with different colors differently. It could also interpret two different pictures as identical if the vector representing their colors are the same. The histogram will, as mentioned, return a vector describing the distribution of colors, for an illustration see figure 2.4. *Shape information* can also be very useful in some cases. This concept tries to divide the picture into shapes. For instance on could identify clouds, water or grass. The problem with this approach is that is hard to distinguish say a wolf from a dog, or even dog from a cow.

There is much to be said about CBIR but as we will not be employing CBIR in our approach but rather build on results from previous uses of it, we will leave it at this. We will however explain why we decided to walk away from CBIR in chapter 4.

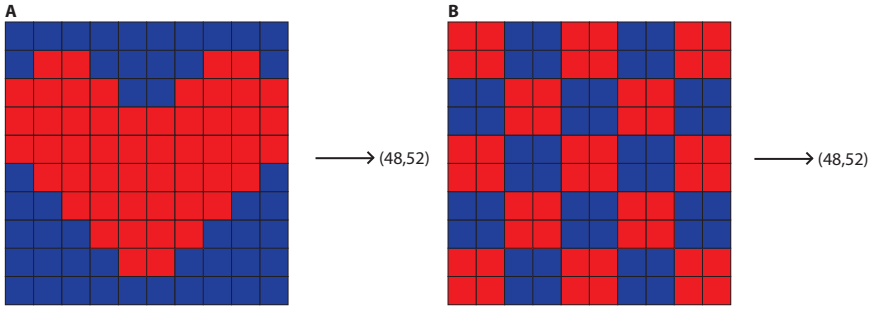

Figure 2.4: A histogram example

2.3 A short introduction to genomics

We will now provide the necessary knowledge about genomics to appreciate our thesis.

2.3.1 Deoxyribonucleic acid - DNA

Deoxyribonucleic acids are the building blocks that make up our genes and thereafter our genome. We rely on DNA to pass on our traits to our children. DNA research is therefore a great way to find links between species and to learn how we evolved into what we are today. There has also been shown[9] that deceases we are pre-disposed for can be reviled through our genes. Basically DNA provides a blueprint for creating living organisms, however we are far from fully understanding how this process works.

In any case, we are doing great progress, and hopefully this thesis may be a humble contribution to this quest.

The DNA molecules are stored within the cell-nucleus. Within the nucleus it is kept safe[4] from the more hostile environment within the cell. The DNA strain consists of nucleotides witch are glued to a sugar-phosphate, and makes up a twisted ladder better known as the double helix (see figure 2.5 borrowed from The National Library of Medicine, USA). Each step of this ladder is pair of nucleotides. A pair consist of either Adenine (A) and Thymine (T) or Cytosine (C) and Guanine (G). One major problem, as mentioned before, when analyzing DNA is that there is a lot of scarp data. That is to say, data that has no apparent meaning or value, but are simply remains of old genes or mutations. Other parts of the DNA make up codings for amino-acids which in turn make up proteins. The DNA required for this is called a gene[9]. This part of the genome is usually of the biggest interest, because it is this part that tells us something about the actual living individual. For the DNA-information to get outside the cell, mRNA is used. It is a temporary copy of half of the ladder in the double helix. Since A can only bind with T and C can only bind with G, only half the ladder is needed for replication of the DNA strain. If we find an A we know that a T follows as an A will follow a T. There are several other complexities surrounding this process but they are beyond the scope of this thesis since they have no immediate impact on our approach. Chromosomes are the housings of the DNA strains, at the time of pregnancy half the chromosomes come from the mother and half come from the father, they combine into 23 chromosome pairs. These are the containers for our DNA within the cell nucleus[9].

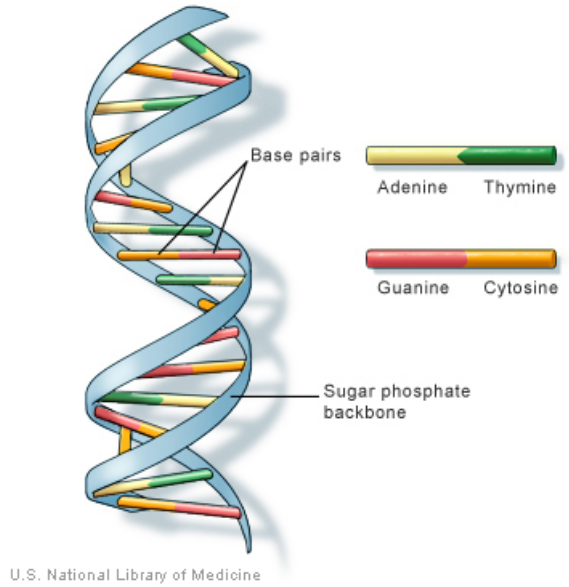

Figure 2.5: The double helix structure of DNA

[4]Well, since mutations from i.e radiation do happen, it is not completely safe.

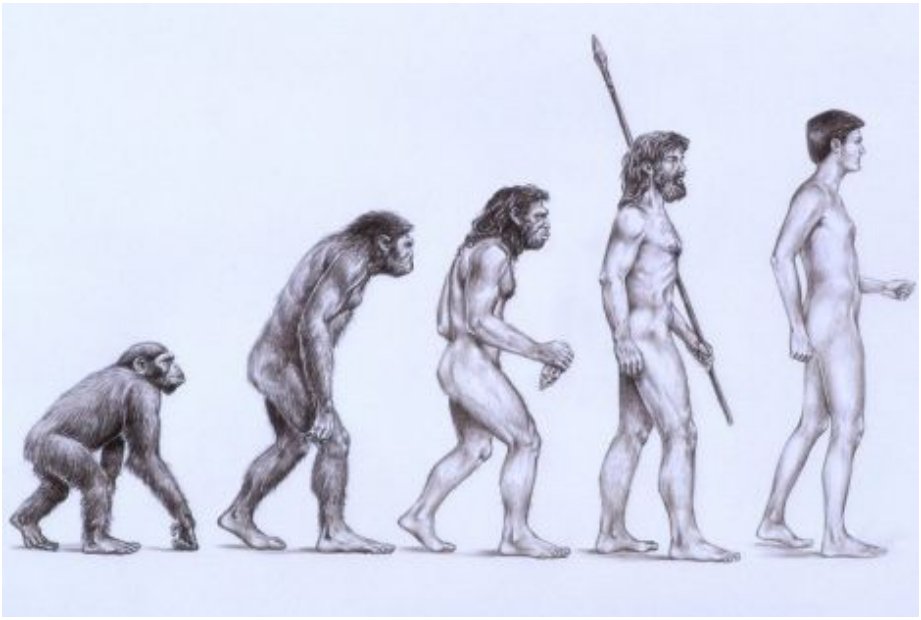

Figure 2.6: The evolution of man; Curtesy of shirtcity

2.3.2 Evolution

Charles Darwin stepped into dawn of a scientific understanding of the creatures on this planet when he published "On the Origin of Species", he suggested that all animals on this planet could be linked to one common ancestor. The amount of evidence for evolution has become staggering over the last decades[3], and we now look to DNA for more clues on how this process came about. As mentioned earlier, DNA is passed on from parent to child. And since DNA contains genes, thus genes are passed on in the same way. However some mutations will occur in the genes when passing them from parent to child, some of the features in the child will be different from that of the parents. Now this is what make evolution possible. The child's slightly altered genes can either make it more or less adapt to the environment. The individual with the best adapted genes, will in the end be the best fit to pass on its genes. Those genes that do not fit the environment however, will therefor, in the end, die. This is the cruel, but simple way that the wonder of evolution is. Also we can look at different species and see similar genes. If we know what one gene will do in one species, that can give us some idea of what that genes function is in another species. This is where we can employ homology searches. Where we look at the evolutionary distance between the two genes. That is, how many insertions, deletions or alterations have happened between the two genes.

We hope that we will be able to create the basis for fuzzy-searching in genes. This will allow us to see the links between species, and maybe find new ways for evolution to occur. It is for example suggested that viruses can carry genes across species.[3] Evolution is therefore one of the fields we hope to learn more about through this approach.

2.3.3 The genome projects

There are several genome projects trying to map up all the DNA of different species. This is important work that will puch us closer to understand life. Some are concerned with different species and some are concerned with different diseases. *The Human Genome Project*[12] is probably the most famous genome project. Is was started in 1990, and in 2001 they managed to get a rough mapping of the human genome. Then in 2003 the HGP managed to map the entire human genome. they had mapped 3,400,000,000 base pairs and they were sequenced. They estimate the accuracy to be around 99.99%

2.3.4 Genetic information retrieval

We have mentioned the huge amount of data stored in the human genome, and the very restricted alphabet that makes makes up the genome. It should become obvious that searching through such wast and homogenous data demands anything but a naive approach. As mentioned the human genome consists of 3,400,000,000 base pairs which ads up to 6,800,000,000 values. Even a human chromosome is almost 51,000,000 base pairs. These are awesome numbers, even in modern information retrieval. If we estimate that a regular text-document holds 400 words per page, a document with 100 pages will hold 40 000 words, as a typical large paper. If we again make a very naive estimate that each word has on average 10 letters, which should be far more then the truth, we only end up with 400,000 letters. Not even remotely close to the documents we are indexing in our case. If we could find an easy way of defining an entire gene, and could know that it would not be a part of any other gene things would be a lot easier. However, this would ignore the major difference between DNA and regular text. The words in the text have not been subjected to change and evolution throughout millennia. Therefore, we have a lot more work to do.

This means that when a researcher is working with DNA there is currently a lot of manual labour involved. Locating genes, determining their function and presence in other spices. There is no doubt that new and better tools are needed for this branch of biology to flourish. Tools that allow the scientists to be scientist and not do what computers could have done for them.

Protein databases

As we have shown, the nucleotide-databases contains wast amounts of data and a lot of it is repetitious and some is junk. However one could use a protein-database to find a description for or identify a gene. This is a lot faster then searching through nucleotide-databases. The drawback is that a protein-database is rarely as up to date, and relies more on human interaction. Because of this the error-ratio increases and keeping the database up to date requires more work. Also the data that shows inactive genes will not be present. But when a gene is located and more exploration is needed, a homology-search through a protein-database is very valuable. Also the protein-sequences will contain a lot more then the 4 characters of the nucleotides and will therefore not be as optimal for n-grams as nucleotides are. Thus we will not be employing protein-databases into our search approach.

Seaquence alignment

Sequence alignment is the notion of aligning two sequences and doing an approximate comparison. We talked earlier about the importance of the evolution of genes and thus

this is a valuable technique for identifying related genes. Our current approach will off-course deal with boolean queries but as mentioned before, we hope to lay the groundworks for a good approximation-search.

We start out by aligning the sequences where they seem to fit best, then we find the evolutionary distance. This is done, as mention earlier, by looking at how many insertions, deletions and mutations that has to be made to make the two sequences identical. There are several different ways of scoring this evolutionary distance. One approach is to give matches between the sequences a positive score, insertions and deletions a negative score and mutations a smaller negative score. This will give us a means of sorting the sequences from close to far from the original sequence that we are using as search input. There are two ways of aligning these sequences, global and local alignment[5] [10];

Global alignment looks at the complete sequence, and tries to align it with the alignment used a search input. For this to work, the two sequences need to be roughly the same length so that they can be aligned directly, side by side. This is obviously not always the case, but if it is, this is a very quick process.

Local alignment however is better at comparing sequences that have different lengths, which is much more often the case. The way that this approach works, is by not looking at the beginning and the end of the sequences, but rather focuses on a start position and a length. And in this way we can iterate over the sequence and find local alignments that would not have been found otherwise. Because of this local alignment is the most frequently used approach.

Glocal alignment is a hybrid between the two who seeks to utilize the best from each of the approaches. This approach attempts to find the best possible alignment which includes the beginning and end of one of the sequences.

It should also be noted that *heuristics* are also often used to filter out unwanted sequences. The way this works is by using an algorithm to calculate how relevant a sequence **probably** is, and thus discarding it if it is thought to be insignificant. A heuristic like this will not be able to give any true identifications, but should be able to discard unwanted sequences and in this way speeding up the process quite a lot.

2.3.5 Approaches to DNA search

By now it should be obvious that searching through DNA is not a simple task. It requires planning and some knowledge about the nature of DNA and one must align a wast amount of sequences. On top of all this, the process should be as cheap as possible, and fast. There are currently many more or less successful algorithms utilized to employ DNA searching, we will now present two of the most influential of these.

Needleman-Wunsch

The Needleman-Wunsch algorithm was published in 1970 in the Journal of Molecular Biology, by the two professors Needleman and Wunsch. Their algorithm performs a global alignment of two sequences and was the first applications of dynamic programming with regards to biological sequence alignment. It is widely used in bio-informatics to align nucleotide or protein sequences because it promises to find the optimal global alignment of two sequences.

The algorithm uses a scoring-matrix to score the aligned characters which is initialized with negative scores. The first column reflects the first sequence to be compared first and the first row reflects the second sequence. Therefore the height will reflect the

[5] There is also a hybrid between these two, often referred to glocal alignment.

length of the first sequence and the width will reflect the length of the second sequence. Then a path is created through a traceback table that gives the optimal alignment which is then used with the similarity matrix to calculate the similarity score. The trace-back begins at the lower right hand-side and finds the cheapest way to the upper left hand side.[11] The basic layout of the algorithm is presented in table 2.4

1	Create a scoring table
2	Initialize the scoring table with negative values
3	Calculate score of the current cell
4	Update the traceback table
5	For all cells, repeat 3 & 4
6	Traverse the traceback table

Table 2.4: The Needleman-Wunsch algorithm

Smith-Waterman

The Smith-Waterman was also published in the Journal of Molecular Biology, but 11 years later then the Needleman-Wunsch algorithm. The Smith-Waterman algorithm differs mainly from the Needleman-Wunsch in that it uses local alignment in stead of global alignment. It to uses a scoring-matrix, but initializes it with zero values unlike the Smith-Waterman that uses negative values. This opens for doing local alignments. It even promises to find the optimal local alignment. The algorithm starts with the highest value in the scoring-matrix. It then uses a trace-back from that position until it reaches a value with zero or the (0,0) position of the matrix. Except from the different initialization the basic algorithm is the varies similar to Needleman-Wunsch. (see table 2.5) There are several accelerated versions or implementations of Smith-Waterman. For instance FPGA and SSE who use specialized computer chips and GPU using Graphical Processing Units. The latter approach has seen a lot of popularity with many algorithms and technologies in the last decade, even the Playstation 3 by Sony has been used as a specialized unit for speeding up the process.

There are several motivations for using local alignment. One motivation is that a sequence that has been through a lot of evolution will have many insertions and deletions, thus many of the regions will be very different rendering a low value for global alignment. With local alignment one can value the alignments that are in fact similar and ignore those that are not. Another motivation is that Smith-Waterman guarantees to find the optimal alignment. However the drawback of Smith-Waterman is that it is relatively slow and demanding. It requires $O(mn)$ time and space. For this reason, BLAST which we will look at in chapter 3, has largely replaced the Smith-Waterman algorithm. BLAST however does not guarantee an optimal alignment.

1	Create a scoring table
2	Initialize the scoring table with zero values
3	Calculate score of the current cell
4	Update the traceback table
5	For all cells, repeat 3 & 4
6	Traverse the traceback table to find optimal algnment

Table 2.5: The Smith-Waterman algorithm

Chapter 3

Technologies

In this chapter we will present technologies that we see as inspirational or that we wish to compare our approach to. These are mostly similar to our approach but in general they employ n-grams in a far more complex fashion than we do. It is our intent to show that it should not be necessary to alter the n-gram so much to get the results that we want. A simple implementation of the n-gram indexing will provide a more stable ground for building a feature complete system and this will help keep the approach as generic as possible. When there is a simple, fast and accurate indexing approach in the bottom that has been shown to perform well it is a lot easier to create specialized filters and retrieval technique on top. This is how we hope to lay the grounds for a fuzzy search that can also be completely accurate but still be fast enough to compete with the state of the art search systems. We will also show what technologies we base our thesis and project testing on. However, we have tried to rely on as few technologies as possible. This is merely based on the fact that we do not expect to compete with the state of the art technologies out of the box, but show find the advantages and disadvantages of our approach in respect to these other technologies. So what we intend to show eventually is how our approach will perform and what can be done to bring it to a full featured state.

3.1 Related work

3.1.1 FASTA

FASTA or "FAST-All"[14] because it will work with any alphabet, is based on FASTP which was designed especially for protein-search. FASTA is very fast thanks to its heuristic approach. It begins by creating a lookup table of the query sequence. The length of the entries in the lookup table is defined by the parameter K. The length of K varies, but for nucleotide databases it can be up to 6, compared to protein databases who normally just use K of 1 or 2. This lookup table stores all occurrences of the tuples in the query sequence and a diagonal dot-plot is used to find matching K-tuples. An example of the dot-plot are shown in figure 3.1 which is borrowed from [17] This process goes on to find possible alignments in a fast an straight forward manner. When possible matches are found they are forwarded to an implementation of the Smith-Waterman algorithm which does the hardest work.

An accurate calculation of similarity statistics helps biologists to decide if alignments are coincidental or signs of homology. FASTA uses local sequence alignment to find

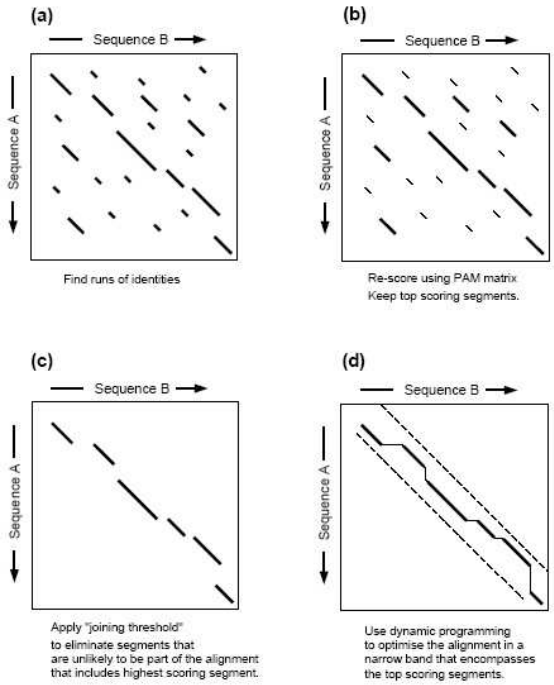

Figure 3.1: The FASTA dot-plot

matches. As mentioned, FASTA first looks at word-to-word matches of a certain length and patterns regarding hits. This marks potential hits before it initiates its second step, an optimized search using Smith-Waterman. There is a four step process of calculating 3 scores which describes the sequence similarity results, these are[14] shown in table 3.1.

1	Identify the regions of highest density in each sequence comparison.
2	Use the scoring matrix to rescan these regions and include only those that are contributing to the highest score.
3	If an alignment has regions scoring higher then a cutoff value, try to join it with others to create an approximate alignment with gaps. Before reporting the best scoring alignments, penalize each gap.
4	Use Smith-Waterman algorithm to calculate an optimal score for alignment.

Table 3.1: The Sequence similarity algorithm

Now this constitutes the process of finding similarities as mentioned the process is completed by employing the Smith-Waterman algorithm. An outline of the basic FASTA algorithm is presented in table 3.2 where S is the cut-off value.

Precision/Recall and query execution times for FASTA, BLAST and Cafe are presented in figure 3.5 and figure 3.6 and further discussed and compared in the section concerning Cafe.

1	Create a lookuptable of length K.
2	Scoring with diagonal dot-plot.
3	Rescore the 10 highest scoring sequences.
4	Join the misaligned sequences with a score higher than S.
5	Use the Smith-Waterman algorithm to calculate optimal score.

Table 3.2: The FASTA algorithm

FASTA format

The FASTA format or Pearson-format is used by many DNA search-approaches. It is a text-based representations of sequences and a sample sequence is shown in table 3.3. There are many tools available for converting DNA-sequences from one format to another. However, we will not go into detail about this, but we will be using the FASTA format for our indexing.

>EMBL_CDS:BAB79009.1 Mus musculus (house mouse)
caggtccaactgcagcagcctggggctgagcttgtgaagcctggggcttcagtgaagctgtcctgca aggcttctggctacaccttcaccagctactggatgcactgggtgaaacagaggcctggacgaggcct tgagtggttggaaggattgatcctaatagtggttttactaagtacaatgagaagttcaagagcaagg ccacactgactgtagacaaaccctccagcacaccctacatgcagctcagcagcctgacatctgagga ctctgcggtctattattgtgcaagatacgattattactacggtagttactttgactactggggccaa ggcaccactctcacagtctcctcagccaaaacaacacccccca

Table 3.3: The FASTA format

3.1.2 BLAST

The Basic Local Alignment Search Tool (BLAST) [4] is a bioinformatics algorithm for comparing DNA and other biological sequence information. BLAST allows for approximate searches with a threshold value T. Such that one can set a threshold for when a sequence no longer is a valid resemblance to the query. In this way, one can search for a gene commonly found in pigs, and look for a similar gene in humans. IF we know the meaning of that gene in pigs, we could use this data to understand better how the similar gene works in humans. It is easy to see why BLAST often is used to look at evolutionary relationships between species. The main reason why BLAST is one of the most widely used search systems in bioinformatics is that it values speed over sensitivity. This is a good approach when dealing with such wast amounts of data as we do when searching in DNA sequences. The drawback, is as mentioned, that BLAST cannot guarantee to find optimal alignments.

BLAST uses a word based heuristic that locates matches between two sequences by using the notion of High-scoring Segment Pairs (HSP). BLAST will assume that high scoring sequence alignments exist and search for them. This approach is based on the segment pairs algorithm from Smith-Waterman. The big advantage of using this approach is that the statistics from the segment pairs algorithm is very well understood, so we can easily interpret the data from it and calculate probability for maximal segment pair alignment. The drawback is that the alignments can not contain gaps but BLAST is faster then FASTA but has about the same sensitivity. Just as FASTA BLAST also uses

the FASTA-format but can also use the genebank format. The output can be delivered in HTML, text or XML among others, but for the web version of BLAST HTML is obviously the default format. An outline of the basic algorithm can be seen in table 3.4 where step 1 is optional.

1	Remove low complexity or repeating regions.
2	Split the query sequence into words.
3	Search for words that have a higher score than T.
4	Repeat for all words in the query sequence.
5	Extend the matching words to create the HSP's.
6	Generate list with HSP's scoring higher than S.
7	Manual or automatic evaluation of HSP list.

Table 3.4: The BLAST algorithm

There are several flavors of BLAST [4] where different settings are used. *BLASTN* and *BLASTP* are used for searching through nucleotides and proteins, respectively, and returns the most significant hits. *BLASTX*, *TBLASTX* and *TBLASTN* all use translation of either input or the target database to improve the search with either proteins or nucleotides. *MEGABLAST* uses multiple queries by concatenating the sequences. *PSI-BLAST* first queries the database, then uses the feedback to improve the search and find distantly related sequences by generating a new sequences based on the most prominent features.

3.1.3 Q-Grams

Q-Gram indexing

A project at the University of Singapore[5] showed one way to utilize q-grams[1] in DNA-search. In their approach they created hash-signatures from q-grams. The idea is that there are a finite number of q-grams for a given q-size. Thus one can create a has map that contains a signature of bits, 0 or 1 to determine of a given q-gram is located within a document. This makes searching very fast, but it also creates a great deal of overhead within the indexing process. The other drawback is that searching for evolutionary relations will be restricted to what is statically set before the index is generated. This makes the approach fast but less general and less dynamic and harder to extend for other purposes then what was intended, the latter is not necessarily a drawback. In addition they were very concerned with the edit distance which they define as such:

> "The edit distance between two sequences is defined as the minimum number of edit operations (i.e., insertions, deletions and substitutions) of single characters needed to transform the first string into the second. ed(S, P) is used to denote the edit distance between sequence S and P."

This approach is in many ways closely related to ours, but the employ a lot of extra overhead to reduce index size and lookup time. These are not crucial points in our approach. We feel that these issues will not be heavily enough compromised by using plain n-grams, but this will be better covered in chapter 6 and we will look closer at their results in chapter 5.

[1] Q-grams and n-grams are the same but we will use the term they have used in there paper when referring to their work.

Q-Gram filtering

At the University of Bielefeld in Germany a similar project was completed in 2005[16]. Their main idea was to;

> "..devise an efficient filter for identifying the regions between A and B that may contain an E-match"

A basic explanation is that they use q-gram-filters to look for similarities between short regions of the sequences. Here an E-match is an approximate match where a threshold number must be satisfied. If the threshold is satisfied a closer inspection is performed. They use a matrix to store the information they gather and an example from their paper is shown in figure 3.2. It is an 8 × 2-parallelogram in the edit graph between the sequences A = TACATGTCAGTT and B = GACTGGCAGC. For q = 2, there are four q-hits within the parallelogram and $dist(,) = 2$.

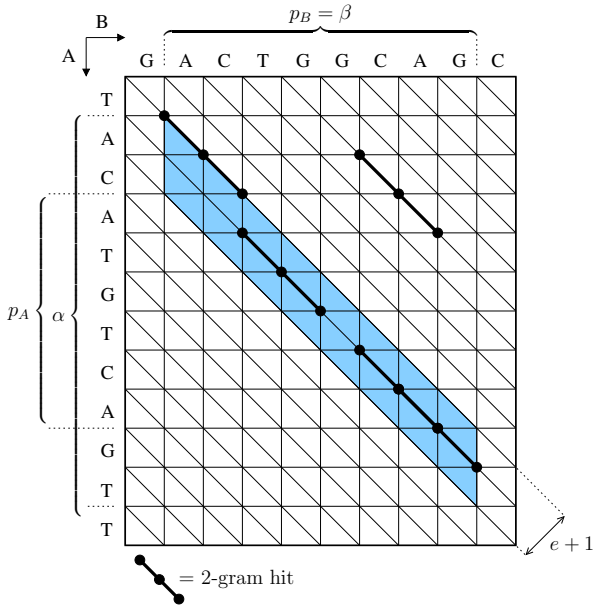

Figure 3.2: An 8 × 2-parallelogram

A lot of their efforts are laid into tweaking threshold values for faster indexing and searching while still keeping recall and precision acceptably high. Precision and recall are not however listed in a way that can be used to exactly compare it to our result. It should be noted though that they seemed to be able to compete with BLAST in sensitivity, that is in our terms precision. But as with our approach this is not a full flexed commercial system. It is an idea that they have show to be very useful, and a filter of such sort could be very valuable in our approach as well. However this is something that should be looked at in future work and as they noted to; ".. increase the interval of practically usable values of □ by allowing mismatches in the q-grams." As we see from their results the q-gram performs very well in regards to that their queries are of length 1000. Since our queries are only a few bases long we feel that 500ms for a 500MB dataset should be a target for us to be able to compete with this approach. When it

comes to precision and recall this project showed that they are equal to BLAST which is pretty good.

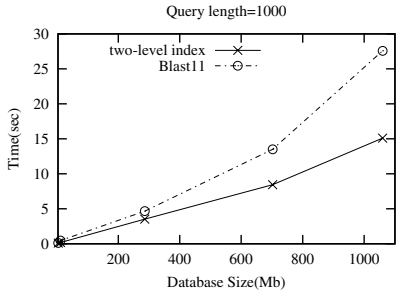

Figure 3.3: Q-gram query execution times

3.1.4 GenomeLIRE

As mentioned Aleksander Grande's thesis "A Multimedia Approach to Medical Information Retrieval" [8], looked at a way to index and search through DNA using Content Based Image Retrieval(CBIR). He let a sequence make up a pixel in an image by letting each nucleotide A, C, an G represent Red, Green, Blue and Alpha values. By tweaking this technique with scaling and filtering he managed to get pretty good results. This approach relies mainly on color histograms. This means that the image is divided into segments and given an average value of each color. This will then be very useful for finding sequences that are similar, and therefor god at evolutionary searches in DNA. But not very good when employed to find exact matches. An example of a DNA generated image from 25,000 base pairs can be seen in figure 3.4.

What was found about this approach is that it is much faster then the regular approaches such as FASTA and BLAST. It also showed to be better at fuzzy searching then these other technologies. This is the part that we want to build on in out approach. However the problem that was that it was hard to determine the exact relationships between two sequences. This is an important piece of information that we aim to be able to do better with n-gram searching while still keeping the possibility open for fast fuzzy searching. But as mentioned before the fuzzy part of our approach is left for further work. Another drawback of this approach is that it is hard to compare sequences of very different sizes because they will produce images of very different sizes. There were presented some improvements to this issues, but it is not clear if this would actually fix the current drawbacks. As for precision and recall they were not essential since it is very hard to determine these values for this approach. But they are included in the appendix and they do not compete with the state of the art DNA search systems. For further information the thesis is available at http://daim.idi.ntnu.no. In this thesis however query execution times was not presented so they will not be a part of the comparison to our results.

3.1.5 Cafe

The authors of Cafe[19] worked to find a way to predefine likely sequence alignments to reduce query evaluation costs. They managed to reduce this cost by 40%-90%[18]

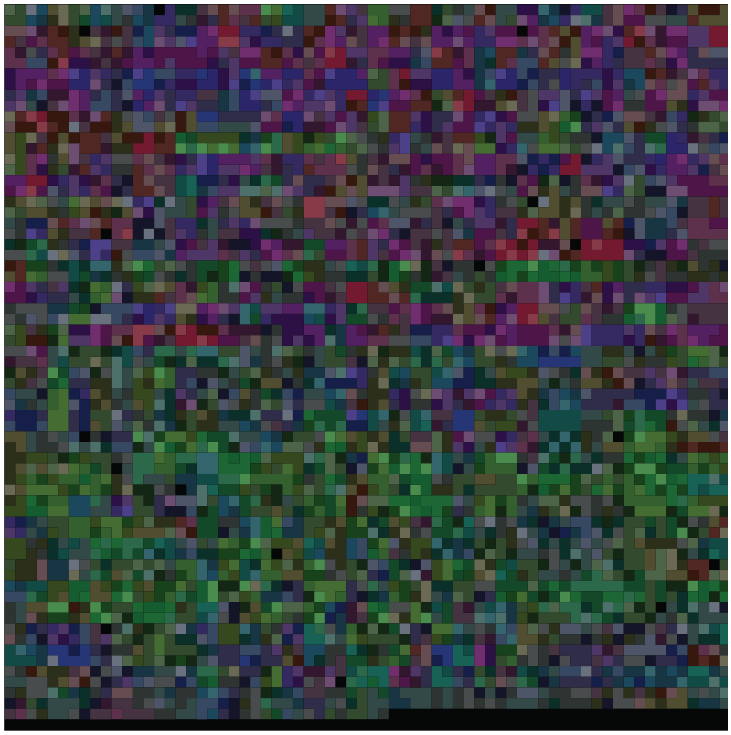

Figure 3.4: An image generated from a DNA sequence

by providing a framework for subsequent local alignment. CAFE employes an inverted index to find a subset of local sequences that show some similarity to the query sequence. It is shown that CAFE is faster but has a little bit lower precision than BLAST when searching for *very* similar sequences. It should also be noted that it was very difficult to find papers describing the details of the inner workings of Cafe as the project seems to be discontinued.

Instead of testing the performance of BLAST, FASTA and Cafe we used the comparisons made by the author of Cafe. As we saw in the graph in chapter 3 (see figure 3.5) they tested how recall and precision developed and found the following; When precision was above 95% recall was as low as 70% and as high as 80% for all three of the systems. They ran tests against the PIRSF collection with 1 834 queries. We used the same solution for looking at query execution times. Figure 3.6 shows the query execution time for BLAST, Cafe, and FASTA and averaged over 41 queries. They actual sizes of the queries are not discussed in detail but the shortest one is of 58 bases so it is clear that since our collection is maximum 500MB and our queries only a few bases long we should strive to have a query execution time below 500ms to be able to compete with the best of these approaches in regards to time consumption.

3.2 Technologies in our approach

For the general development we have used Eclipse an open-source IDE (Integrated Development Environment). We have employed plugins for LaTeX which is a markup

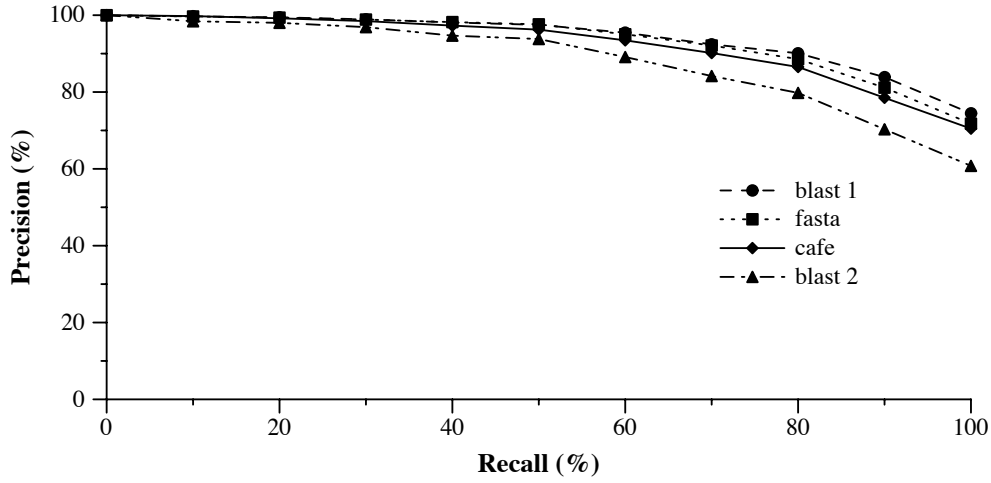

Figure 3.5: A precision recall comparison of BLAST, FASTA and Cafe

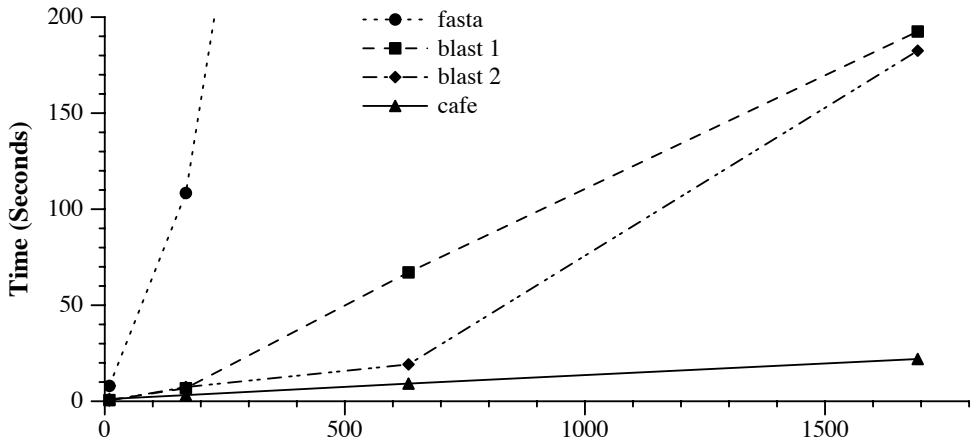

Figure 3.6: A query execution time comparison of BLAST, FASTA and Cafe

language used to create documents, articles and books. Subversion takes care of version-control and backup. It makes it easy to roll back to previous branches of the projects and keeps it safe in case of computer crashes. Our entire setup was hosted on a Linux machine running Ubuntu 9.04 - server edition. As for that, most of the development was done using Mac OSX. In addition OmniGraffel was the software we used to create graphs and diagrams in this thesis.

3.2.1 Lucene

Lucene is a project hosted by apache[6] and they claim their place in the world as such;

> "Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform."

This is quite covering, as Lucene is indeed an open-source search engine library. We have chosen to do all our programming in Java to make the cooperation with Lucene as good as possible. Java is general-purpose, concurrent, class-based, and object-oriented programming language initially developed by Sun. However some of the parsing of the FASTA formatted DNA was done using a few simple python-scripts. One of the big advantages of lucene is that is used in many fields of research, so there are many available plugins and extensions. It is also easy to extend many of the algorithms employed by Lucene so it is well suited to be used in the further work of this approach. Lucene is free in the sense that it is open-source and we will distribute our project as such. Lucene is widely recognized as fast, reliable and the most frequently used java implementation of a full-text search engine. Because there is so many developers on the Lucene project, documentation and support is not hard to get. By using the irc[2] server freenode, we quickly got help from the lucene support and development channel. This made development a lot easier then having to deal with a support-team over the phone for instance.

3.2.2 The power of Python

Since we used DNA retrieved from the European Bioinformatics Institute[3] we had a lot of DNA but no queries with predefined results that we could test our search against and find precision and recall. For this we utilized pyhton scripting. We predefined 30 queries for each test and had the python-script walk through the entire DNA-collection and find every document containing the query. Since this is a very naive and straight forward approach it took a long time, but gave us invaluable information when testing our approach. With these correct answers to our predefined queries the testing of our approach became a trivial matter of comparing our results to the results from the python-script.

[2] Internet Relay Chat, often wrongfully just know as mIRC
[3] http://www.ebi.ac.uk/

Chapter 4

Implementation

4.1 The main idea

The main idea of our approach is to take advantage of the homogeneity of DNA. Since the data has so little variations and only has 4^n possible values where n is the size of the n-gram, the index will be relatively small. And a small index is usually a fast index, which makes sense when you want to search through wast amounts of data. What we wanted to test a specially is which size of the n-grams will give us the best results. To evaluate this we look at how different n-grams perform with different dataset sizes. We also enabled filtering on annotations, as it is sometimes useful to to filter on what species or type of gene you are looking at. However, the annotations are merely a suggested feature and not the main characteristic of our approach.

As we have shown in chapter 3, there has been done other studies on similar approaches, but usually with a lot of extra overhead in the use of n-grams. These systems have also had a tendency to be very specialized. Looking at ways to improve certain aspects of DNA-search. What we are trying to do is to create an overall good base for DNA-search. Our approach looks at n-grams and we will suggest several steps to be taken to optimize and extend our approach in the future.

4.2 Problems and decisions

During the development of our approach we ran into a few problems. We did however manage to solve them either by rethinking our future path or implementing better alternatives to our previous algorithms employing an evolutionary method of development. As mentioned we received very good help from the developers of Lucene.

4.2.1 The search for our approach

As mentioned earlier a lot of our preliminary research was based on the findings of [8]. We also looked into many of the numerous articles written on the subject of DNA-search such as those presented in chapter 3. As a matter of fact we started out trying to develop Grandes approach further, however we met a few obstacles. First of, the process of turning DNA-sequences into images left only one major feature to be extracted that could be fully utilized. This was the color histogram and gave a good approximation search. As a special case this is very often valuable, but value of a fast boolean search

can never be underestimated and thus we decided to leave the approximation search as future work. Since we want to be able to do more than just approximate search. Instead we used some of the information gathered by Grande when constructing our approach and algorithms to lay the groundworks for an extensible way of searching with different degrees of accuracy and utilizing annotations to filter the results. This is where we found that n-grams seemed to be the best suited way of indexing DNA.

Annotations

After doing some test-coding, we found that the filtering of annotations was very naive and straight forward, it would be a standard text search that is matched to the result from the sequence search. Because of the natural low complexity of this feature it is not heavily emphasized in this thesis, but is merely suggested as a good way of filtering the search-results. One should also note that if implemented properly it should not affect recall, but enhance precision if anything, this should however be tested.

4.2.2 Memory restrictions

We soon discovered that memory is a huge issue when indexing wast amounts of DNA. This was no surprise, but a fair month was spent trying to streamline the indexing process to allow the entire dataset to be indexed without running out of heap-space[1]. The solution was twofold, first of the dataset was split into smaller more manageable files by a simple python-script. This allow Lucene to index smaller files and to write back to disk more often and thus flushing the heap more frequently. In addition a custom indexing method was written to allow loose handling of the index, so that the entire dataset did not have to lie in main memory. This is in fact the part that would perform the flushing of the heap so that it would not grow larger then the allocated memory space.

4.2.3 The different formats

There are several formats for storing DNA-sequences. Both XML and structured text-files are popular, we have previously showd an example of structured text being the FASTA-format. An example of an XML-format can be seen in table 4.1. The structured text format GenBank is very popular but it has a lot of overhead compared to the FASTA-format. (see table 4.2) First of we tried writing parsers for the different formats, but it turned out that there are several tools available for converting between formats. Since the actual parsing of input data formats is not a part of the approach we decided that this should be placed outside of the scope of this thesis. This led us to stick to the FASTA-format since it is well documented, simple and thus very easy to read and analyze. There is also more then enough data available in the FASTA-format.

4.2.4 Testing

Since we used DNA-sequences that had no test sets, that is datasets with the pre-ordered results of given queries, we had some problems defining precision and recall at first. However we created a simple python-script that created queries with pre ordered test sets for us. Also testing the query-results from our approach against the test set proved difficult in the beginning. This was primarily because Lucene does not allow the listing of all found results to a query, it simply does not have a built in method for this. In stead

[1] The memory dedicated to the program when running

<species> Ecol K12 W3100 </species>
<chromosome> Ecol K12 W3100 </chromosom>
<sequence> AACCTGCGGAAGGATCATTACCGAGTGCG GGTCCTTTGGGCCCAACCTCCCATCCGTG TGTTGCTTCGGCGGGCCCGCCGCTTGTCG GCCGCCGGGGGGGCGCCTCTGCCCCCCGG CGGAGACCCCAACACGAACACTGTCTGAA AGCGTGCAGTCTGAGTTGATTGAATGCAA GCCCGTGCCCGCTCTATTGTACCCTTCAA CAATGGATCTCTTGAACTGCTGTTCCGGC <sequence>

Table 4.1: An XML format

GXP DEFINITION	170357 743 bp DNA loc=GXL 141619|sym=TPH2|geneid=121278|acc=GXP 170357| taxid=9606|spec=Homo sapiens|chr=12|ctg=NC 000012|str=(+)| start=70618393|end=70619135|len=743|tss=501,632| homgroup=4612|promset=1|descr=tryptophan hydroxylase 2| comm=GXT 2756574/AK094614/632/gold; GXT 2799672/NM 173353/501/bronze
ACCESSION	GXP 170357
BASE COUNT	216 a 180 c 147 g 200 t
ORIGIN	TTGATTACCT TATTTGATCA TTACACATTG TACGCTTGTG TATAAATGTG TACAACTATT AGTTATCCAT AAAAATTAAA GGTTTAAGCA TTCAGCAGTG CTGATCTTTC TTAAATTATT GCACAAAATC TTTGAATTCA CAATTGCTTA AAGACTGAGG TTGAGAGATG AGAGAACTAA CGTCAGAGGA TAGATGGTTT TTATGTATTG TTCTCCACCA CCCCCGCCCA AAAAGCTACT ACTATGAGCA CAGATAACCC CAGGCTTCAG GTCTGTAATC CAGAAATGAG TTTCTTTCTA ATCAGTCTTG CATCAGTCTC

Table 4.2: The GenBank format

we had create a custom collector, run through the results and store their document identifier in a hash map for later comparison. We suffered the same memory problems here as with the indexing until we solved it using the same technique employed in the indexing process and some nice pointers from the developers of Lucene. Another part of the testing that proved difficult was measuring the query execution time. As mentioned we had to customize the query process a bit to get the information we wanted, and this led to increasing the query time a lot. The solution was simply to run all queries twice, on simple search which was timed and one that aids in the measuring of precision and recall.

4.3 Algorithms

The algorithms was developed using an evolutionary approach. We did not really know what to expect from Lucene nor did we know how the index would behave whilst indexing so large amounts of data. As described above we ran into a few problems along the way. This is how we solved them and implemented our benchmarking system.

4.3.1 Indexing

The indexing process was divided into two main procedures and the code for the indexing can be viewed in the appendix. The filtering of the input and the selective tokenization of that input. First of we looked at the the data line by line. The Annotations are indicated by a ">" as the first character, and the DNA-sequence will then follow after a newline and continue until a new annotation is met in the form of a ">". The data is read line by line from the dataset so that we will not have to keep the entire document in memory while we are indexing. We add the annotations to the index as a field called "Annotations" and sequences in a field called "Sequences". These two fields together with a document Id constitutes a document in our index. Now that these lines of input are filtered into their respective fields, each field is tokenized with a separate tokenizer. The annotations are tokenized as simple text where some common stop-words are removed. The document Id is stored as it is, and the sequences are tokenized by an N-Gram tokenizer and a lowercase filter for consistency. For a basic outline see figure 4.1. When the indexing is done, we run Lucene's Optimize method on the index. Thus it is compressed and rearranged to better meet speed requirements. This algorithm can be summarized as follows:

```
1  while((line = doc.readline() )!= null):
2      if ">" in line:
3          annotaionField += line
4          if(sequenceField != ""):
5              annotationField.StandardTokenize()
6              sequenceField.NGramTokenize(gramSize)
7              document.addField(annotationField)
8              document.addField(sequenceField)
9      else:
10         sequenceField += line
```

Figure 4.1: Indexing of sequences using n-grams

Since we were going for a elegant and simple system that should be the base for a more in depth and specific search-system, we tried to keep the architecture of the system as simple as possible. We have devised the architecture into general modules and an outline is presented in figure 4.2. The document collection is the input of the

system. The documents are passed to the indexer which in turn creates an index by using specific tokenizers for each field as described in chapter 3. The index can then be searched by the searcher, which is in fact implemented as a search controller. The bench marker gets its input from a python script. But since this is not a part of the actual system it is not included in the figure. But in short the python script creates 30 queries with correct results for each query for n-grams of size 3,4 and 5. This data is then used by the bench marker to query the index via the searcher, and precision and recall is calculated along with other statistical data which will be presented in chapter 5.

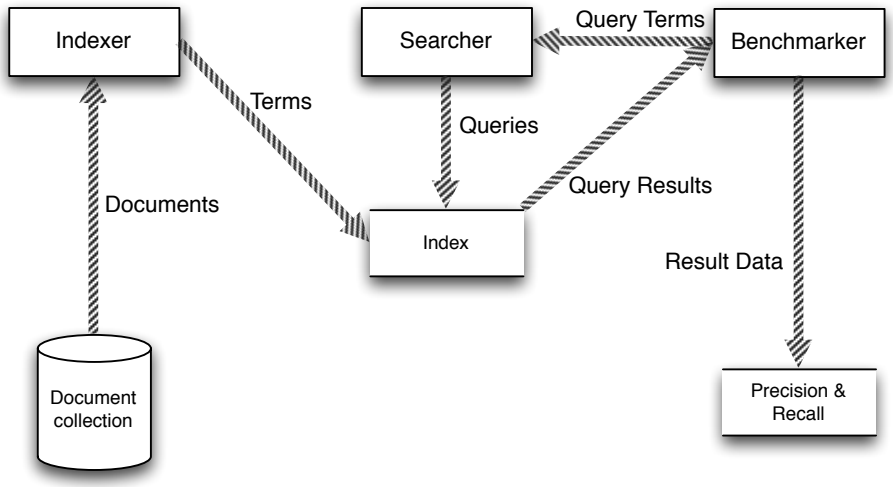

Figure 4.2: The basic architecture of our system

4.3.2 Searching

We primarily used Luke[7] for searching, since it works out of the box and has an nice graphical interface. Luke fitted us well since it is also open source and easy to use. However some of the tests and searches required us to write a custom search method since we need to do some manual labor to get the information we wanted from the index. However for all other purposes it is very straight forward to use Luke as it just parses the input to the built in keyword searcher in Lucene and traverses the result. Luke also gives some statistics concerning query time and such, but this was not our main concern in this thesis. However speed is the area where we will suggest the most of the improvements to be made in further work.

4.3.3 Testing

As mentioned we had to create our own test sets, this was a challenge but when completed it turned out to be a great strength. It let us run our own, predefined test and we were able to look at the most used and least used terms in the index before we decided on what terms to use in the queries. The main part of the python-script looked like this:

```
1 for line in inputFile.xreadlines():
2     if(">" in line):
```

```
        if(foundTerm):
            docCounter += 1
            docList += annotation
            foundTerm = False
        annotation = line
    else:
        if(query in line):
        foundTerm = True
```

We looked for the ">" marker of the annotation, stored it, and returned it if the following sequences contained the query we were searching for. Since we had the advantage of predefining the format of the test-results the benchmarking was a lot easier. The flow of the algorithm was this:

```
def runTest(queries):
    for query in queries:
        benchmark(query)

def benchmark(query):
    searchResultsHashMap = searcher.doSearch(query)
    testResultsList = testset.getResults(query)

    for result in testResultList:
        if result in searchResultHashMap:
            correctFindings++
        else:
            falseNegatives++
        totalFindings++
```

When we have found the totalFindings, falseNegatives, and correctFindings, it is very straight forward to find precision and recall. This gives us a good and clear way to compare our approach against other existing solutions. The query execution time was also measured and gives a pointer as to how fast the system will perform. We have also gathered information regarding index sizes and required time for indexing although this is not crucial information at this time. In addition we have looked at how many DNA N-Grams are gathered compared to the amount of documents contained within the index. All of this will give us a solid image of the performance of our approach and where to go from here and if our approach is viable at all.

4.3.4 The construction of our algorithms

As mentioned in the beginning of this chapter we have employed an evolutionary approach. By this we mean to say that we started out with a naive approach where we just tried different algorithms and stuck with the ones that served us best. This also means that we had to occasionally step back and rethink our approach and algorithms. However we felt that since this part of DNA searching is not very well explored a specially in regards to high precision searching, and thus found that a evolutionary approach was the best suited and it has served us well.

Chapter 5

Testing and results

Here we present our test and the results from our n-gram indexing approach. We will show how our solution performs while the data to be indexed grows and how the different n-gram sizes will perform under these conditions.

5.1 The testing

We then created a test method that took all the test-results and processed them, did the required queries, stored the precision and recall for each query and the took the average of the 30 queries we ran on each n-gram. This was done over for 4 different ascending test set sizes, and also for each of the n-gram sizes 3,4 and 5. In other words 360 queries were used to test our approach, which we feel should be enough to give a glimpse into the performance of the different n-grams as the test-set size increases. It should also show the difference of performance between the selected n-gram sizes and gives a view of the average query execution times for each size of n-gram. The n-gram sizes of 3,4 and 5 were selected because they would keep the index as small as possible and hopefully keep the search fast and accurate. A larger n-gram of say size 6 would start to contradict the intentions of our approach and create a relatively larger index.

In addition we noted the indexing times and index-sizes. This data is however a little bit less reliable and frankly not an important part of our approach since this is what we have intended for further work. The indexing time can not be trusted completely because the general strain on the computer can not be accounted for as there were other processes running at the time. For this to be accurate a dedicated computer should be used for all the indexing and benchmarking. The index size however is more reliable and gives a clear indication on how the n-gram size affects the index size. But since we used the optimize function in lucene your results may vary since lucene does a few tweaks to reduce the size and search time. As a side note, time consumption time for the queries was not measured. The reason for this is two fold. First of all we had limited time to set up the queries, testing, research and so forth. Secondly it was hard to compare this data with results from other tests on other systems since we would need their queries and their test data .And in addition, this is as mentioned a concern we will address in further work.

5.1.1 Our testset

As described in chapter 4 we created our own test-sets. This is a long process, but not very difficult. It also has great advantages since we are thus allowed to choose the format of the tests and what queries that should be used. It also lets us have a better understanding of the test set and discover possible biases in the queries. The DNA-sequences used for testing was collected from the European Bioinformatics Institute, and were stored in the FASTA-format. We stored the test results in text files containing the the id of the document that was stored in the annotation. This gave us files with test results for each query that we then compared to our own query-results. As shown we then used these test to find precision and recall. In all the graphs in this chapter we compare query execution time and precision/recall to the amount of documents in the index. The actual size of the data set used to create the index compared to the number of documents can be viewed in table 5.1.

Index nr	Documents	Size
1	115 891	120MB
2	244 193	240MB
3	348 056	365MB
4	506 157	550MB

Table 5.1: Table showing the raw data size used to create each index

5.2 How to evaluate the results

Precision and recall will be the main benchmarks for our approach. As mentioned it is the de facto standard for comparing search system, also in DNA searching. Although some systems like the CBIR approach of Aleksander Grande which we have mentioned throughout this thesis did not have a full comparison of these values, all other approaches that we have referred to did in fact produce these values. The query execution times are also important in the evaluation of the system. We will as mentioned have look at the indexing time and size as well. However, since we have not implemented a specialized n-gram indexer for DNA but rather used a novel approach, the results of the precision/recall and query execution times are much more important in our case. If we can get good results with this naiv approach, there is a lot of room for improvement on indexing time and size by specializing the n-gram-indexer for DNA.

We would be very happy with precision and recall lying at 95% and above and query execution time should be below 500ms [19], this will place us up with the best of the current technologies for searching in DNA and show that we have a strong base for exploring the world of searching DNA further. The major focus of our thesis is to find if n-grams are useful in DNA-search and if it is, what size of the n-gram would be the best. It would also be interesting to see if a different size of n-gram is more useful with different dataset sizes. We decided to test only 3,4 and 5 sized grams. This is mainly because 2 grams would be so small that indexing and search process would take very long. Keep in mind that to search this index you would have to split up the search string into n-grams and do a boolean search where all the n-grams are AND'ed together. Furthermore n-grams larger then 5 would make our index so large that we will loose the benefits we are looking for in the first place.

5.3 The results

5.3.1 Main results

Precision

After the tests had run to completion we analyzed the results and placed them into graphs showing precision, recall, and indexing-times. As mentioned indexing-times were not crucial and not completely accurat but are presented more as a guideline. First off we looked at precision (see figure 5.1). Precision is very important because it tells us how much of the results returned by our search that is relevant to the posed query. Returning many many unrelevant results will give a low precision, returning only relevant results will give a precision of 100% which would be optimal. However a lower precision could be ignored if recall was high and we had a good ranking method. This would leave all the most relevant results visible to the user. As we see from our graph the precision is close to perfect when using 3-grams. We also see that 4-grams give almost only relevant results, but when we try to use 5-grams we get a bit more unrelevant results. (see table 5.2) However it should be noted that all the n-gram sizes performed well in regards to precision.

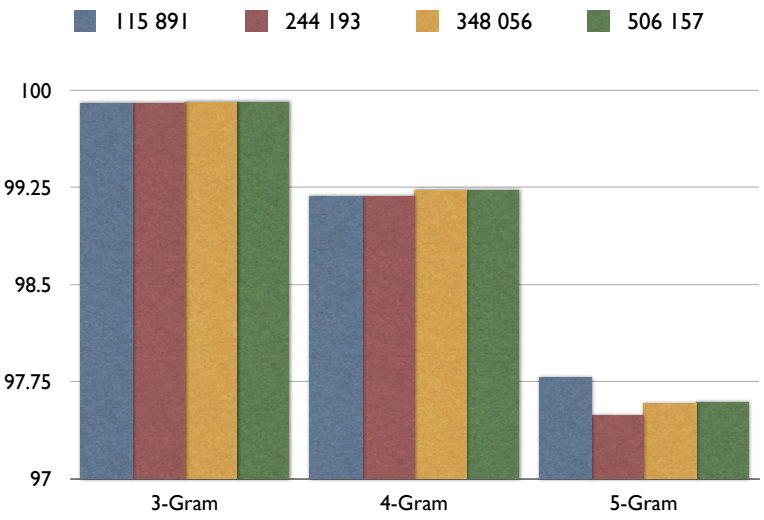

Figure 5.1: Precision

Recall

It is when we start to look at recall we really see some difference between the sizes of the n-grams. (see figure 5.2) 3-grams are clearly very good and is collecting close to all of the relevant documents and the performance is almost constant as the number of

Nr of Docs	3-Gram	4-Gram	5-Gram
115 891	99.90%	99.18%	97.78%
244 193	99.90%	99.18%	97.49%
348 056	99.91%	99.23%	97.58%
506 157	99.91%	99.23%	97.59%

Table 5.2: Table showing percision

documents increase. The 4-grams are very much the same, but again the 5-grams are falling behind. (see table 5.3) When we look at how the performance of the n-grams are scaling it is hard to find a pattern because of the relative small changes. But there is an indication that when the number of documents grow precision improves on the cost of a lower recall value. This is however what was expected but also in this case the overall performance of the different sizes of n-grams was very good. A low recall value would be very damaging to our approach since it would leave out relevant results and thus not lay a good ground for future work.

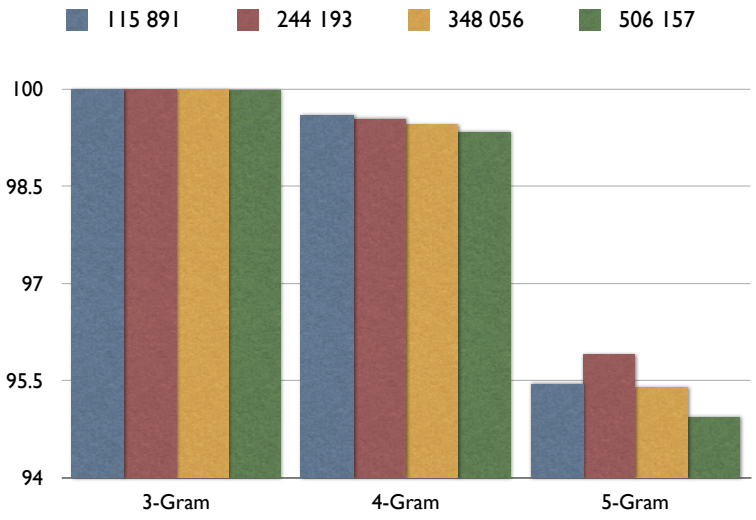

Figure 5.2: Recall

Query execution time

We stated earlier that we wanted to keep below 500ms for any given query in our approach. This is based on the data presented in the related work and from a user oriented standpoint. Nobody wants to search today and retrieve the results tomorrow. The query execution time is measured from the nanosecond the query is posed untill the 50 top

Nr of Docs	3-Gram	4-Gram	5-Gram
115 891	99.99%	99.60%	95.45%
244 193	99.99%	99.54%	95.90%
348 056	99.99%	99.45%	95.39%
506 157	99.98%	99.34%	94.93%

Table 5.3: Table showing recall

ranking hits are retrieved and the results can be viewed in figure 5.3 and the actual values are showed in table 5.4. These numbers are the average of 30 queries for each n-gram size and for each index size so a total of 360 time measurements was performed to gather the result.

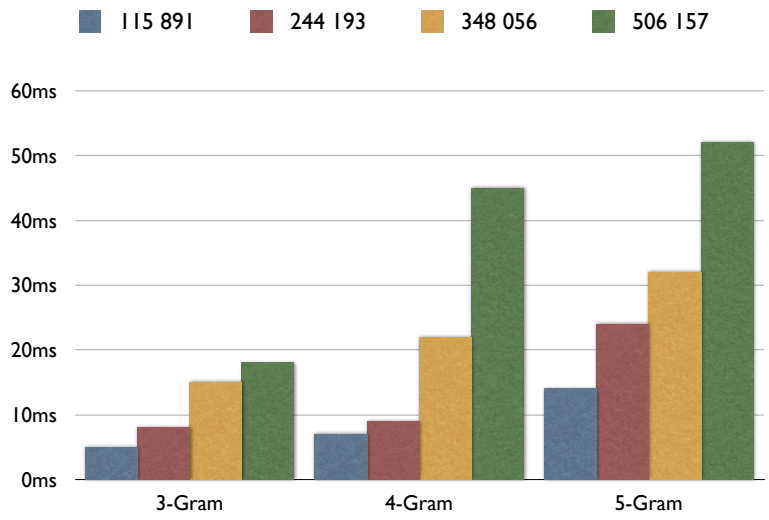

Figure 5.3: Query execution times

Nr of Docs	3-Gram	4-Gram	5-Gram
115 891	5ms	7ms	14ms
244 193	8ms	9ms	24ms
348 056	15ms	22ms	32ms
506 157	18ms	45ms	52ms

Table 5.4: Query execution times

As we see the query execution time grows rapidly when we increase the index size.

In this case there is also a big difference between the n-gram sizes. The 5-gram basically has its lowest execution time equal to the highest of the 3-gram. Further more all three n-gram sizes seem to have about the same factor of increasing query execution time when the index size increases. Also as expected the 4-gram lies as a mean average between the two outer points 3-gram and 5-gram. Not unexpected though the increment for each rising index size is not uniform but it gives a clear guideline as to the scalability of the system.

5.3.2 Guideline results

The indexing time and size are, again, not really an essential part of our research but it is included as a guiding factor and to help give a better image of how the system performs.

Indexing time

As mentioned earlier the data concerning the indexing-time is only useful as a guideline. The actual time it took to index depends heavily on the computer used for the indexing, but the the development of time consumption as the number of documents grow can be very useful because it shows some pattern of development. As we can see, the indexing-time grows quite a lot. (see figure 5.4). This suggests that there might be possible to do some improvements on how the index is built, since Lucene does not seem to optimize for the fact that there are a lot of repetition in the terms being added. Also it should be noted that almost 40% of the indexing time is spent on optimizing the index which should not be necessary with so homogenous data.

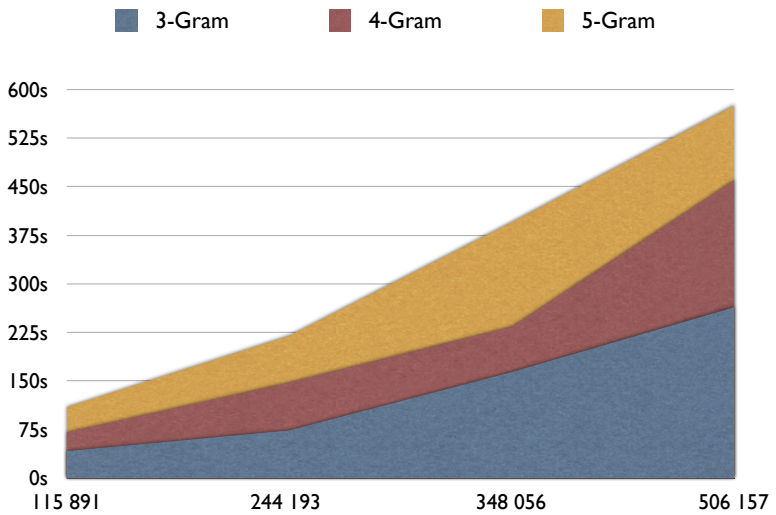

Figure 5.4: Indexing time

Index size

When we looked at the index size in the form of DNA-terms per index. We saw that the maximum number of terms for each n-gram was reached even with the smallest number of documents. (see table 5.5) This suggest that we are utilizing the actual size of the index devoted to storing the DNA-terms since the index-size was generally within a few percent of the original data. In other words, the index is very small. When we indexed with annotations the annotations took approximately 80% of the original data size, which might suggest that one should look into how helpful the annotation filter really is, and if it can defend taking up such a large space. However, we have left this up to others as we have a limited amount of time to do our research.

	3-Gram	**4-Gram**	**5-Gram**
Possible terms	64	256	1024
Actuall terms	64	256	1024

Table 5.5: Found terms for each n-gram size

Chapter 6

Conlusion

6.1 Our Results

6.1.1 The numbers

From the results in chapter 5 we have seen that the performance of the n-grams will wary not only between each other, but also as they are used on different amounts of documents. We have seen that 3-grams generally perform very well with both precision and recall. They are both close to 100% even as the number of documents were increased. However the 4-grams showed a increase in precision and a decrease in recall as more documents were added. Finally the 5-gram show the the most variation, and also the lowest values of both precision and recall. In the case of the 5-grams it was also very hard to see how the recall and precision was effected by a larger number of documents. The query execution times however was much better then what we hoped for. We saw that we were never near reaching our own limit of 500ms. This limit was based on the fact that the systems we compare our selves to used larger queries but consequently used a lot more time performing the queries. We thus took an calculated assumptions as to what results they would have had with smaller queries. This fact makes this comparison very hard since different papers have used different sizes of queries to test for query execution time. Therefore this comparison should be explored more in depth with larger queries of different sizes in further work to establish more precise data. We used queries of about 5 base pairs as compared to 1000 and 58 in other papers which is quite an enormous difference. As shown in chapter 5 we peaked at 52ms as the raw data size for creating the index reached 500MB which is very good. The best of the n-gram sizes however never rose above 18ms which is and outstanding result. It should although be noted that we should perform bigger queries over larger datasets to see how this approach will hold against huge datasets. For now, these results are at least very promising.

When we looked at indexing time and sizes we saw that the indexing time grows pretty rapidly but, as mentioned earlier this was mostly because of the optimizing that Lucene performed after adding the documents to the index. Therefore the time can probably be shortened, but one would then have to look at what effect this will have on performance. It would however not affect precision and recall since nothing is added or removed from the index during the optimization process. Furthermore the index sizes was pretty much as excepted. All possible terms was quickly added to the index, and it did not grow with any significance as the number of documents increased.

6.1.2 Comparison to related work

As we saw in the graph in chapter 3 (see figure 3.5) the authors of the Cafe paper tested how recall and precision developed for FASTA, BLAST and Cafe and found that the three different approaches all scored almost identically. When precision was above 95% recall was as low as 70% and as high as 80% for all three of the systems and in comparison, our 3-gram approach gave 99% precision and 99% recall. Even our 5-gram-index did better then all of the three approaches. When we looked at query execution time we saw that we performed better then the other approaches that we have compared our work to. This is as mentioned a great motivation for continued work on this approach and to bringing it even closer to a full flexed DNA search system. But one should keep in mind that these are calculated comparisons and further exploration is needed.

Certainly, there are other reasons why a specially BLAST and FASTA are the de facto standard for gene-search today. These results only tell us that as far as precision/recall and query execution time goes, we did better then what was expected and what other systems did.. Other qualities that are needed and expected will be addressed in further work. For all intents and purposes of this thesis however, we consider our approach to be a success. As mentioned, the road to a complete and commercially beneficial search-system is long and hard, this is merely the first step and there is still much to learn from the established approaches to DNA searching.

6.1.3 Implications

These numbers tells us that for one, the 3-grams are very ideal when it comes to precision and recall. Their performance is stable while the size of the test set size increases and does not show tendencies to falter. In addition they will keep the index small and has the smallest indexing-time required. We also saw that 4-grams did very well, but it is hard to find a tradeoff for using 4-grams when 3-grams give better performance, a smaller index, and faster indexing time. By the same argument 5-grams does not seem suitable, when so substantially outperformed by 3-grams in all regards. They could not compete with 3-grams in any of the considered areas. We should also note that since DNA datasets are so wast, we should look at all possible ways to reduce both indexing size and time while keeping recall and precision as high as possible. This leaves us to conclude that 3-grams are superior in all ways that we have looked at, but further work to establish this claim will be suggested.

6.1.4 Drawbacks

Our system has shown great results in regards to precision and recall. However searching in this system is all but intuitive. One would have to parse the queries into n-grams and use a boolean AND operation on the result. Rendering only the documents which contain all the search terms in the result. Furthermore the index there has not been taken any steps to optimize the index for such homogenous data. The biggest drawback as we see it is that there is no sense of spatiality between the terms. This means that you can not search for “ATCG near AATG” which would be very useful when looking for a continuous gene. It would reduce the boolean operations necessary and reduce or remove the need for filtering the queries with an n-gram filter.

6.1.5 Contribution

We have shown that improvements are needed in this area of bio-informatics, and that a fast reliable search within DNA has many motivations. Furthermore we have shown that our approach can compete with current state of the art technologies, a specially in regards to precision, recall and index-size. We have done so using a very naive and straight forward approach compared to the very complex and memory consuming approaches that are used today. We regard this in it self as contribution and hope that our work can further contribute as stable and accurate base for a full flexed state of the art DNA search system in the future.

6.2 Future Work

6.2.1 Finding the needle in the haystack

What we do not know as of yet, is the ease of use between the three n-gram values. That is to say, precision and recall tells you if the system returns the results that are correct according to the query posed. It does not however tell you if the results are what the user wanted. This means that we do not know how good this approach is to get exactly what you want. We therefor suggest that test be made and steps taken to ensure that the searching process is intuitive and uses some sort of feedback system. This would allow the system to learn how a users searches and what the user wants when searching in a particular way. A system is utterly useless if it can not find what the user is seeking, no matter how good recall and precision are.

6.2.2 Distance weighting

Another issue is as mentioned that the n-grams have no information regarding where in the document it is located or what other n-grams it is close to. Since the input query has to be split into n-grams when searching, we could get a lot better results and ranking if we knew the distance between the n-grams so that n-grams that are close to each other can be wighted higher then those who are far apart. This might even remove the need for using a n-gram filter on the query.

6.2.3 Recoding

An approach close to the one used in q-grams would be very interesting as an extension of our approach. Lets say that since there is a finite number of n-grams dependent solely on the size of n, each n-gram could be translated into a number or a hash. Reducing the size of the index and also helping when collaborating and sharing DNA. This might also create extra unwanted overhead but is worth looking into.

6.2.4 Query time

It would be very beneficial to look at the query time as the query size grows and find ways to further reduce it. This would require an extensive amount of testing and would probably also call for a specialized index to keep the time as low as possible.

6.3 Summary

We have shown the motivation for our approach and presented the necessary background information needed to fully understand our work. Furthermore we have presented competing technologies and their performance results. We have presented our approach and our results and compared them to the state of the art DNA search systems. The results showed that our approach can indeed compete with the compared approaches and we have given our thoughts on the road ahead. With this we hope that our work can be a humble contribution to the field of bio-informatics.

Bibliography

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Pearson Education Limited, 1999.

[2] Michael K. Buckland. What is a 'document'? *Journal of the American Society of Information Science 48, no. 9*, 2007.

[3] Richard Dawkins. *The Greatest Show on Earth: The Evidence for Evolution*. Free Press, Transworld, 2009.

[4] Lipman DJ et al. `http://www-math.mit.edu/~lippert/18.417/papers/altschuletal1990.pdf`, 1990. The BLAST algorithm.

[5] Xia Cao et al. Indexing dna sequences using q-grams. *Springer Berlin / Heidelberg*, 2005.

[6] Apache Software Foundation. `http://lucene.apache.org/`, 2010. The Lucene project webpage.

[7] Apache Software Foundation. `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC280013/?tool=pmcentrez`, 2010. The Luke procjet webpage.

[8] Aleksander Grande. A multimedia approach to medical information retrieval. *NTNU*, 2009.

[9] Arthur M. Lesk. *Introduction to Bioinformatics*. Oxford University Press, 2002.

[10] Cristopher D. Manning. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[11] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 1970.

[12] U.S. Department of Energy and the National Institutes of Health. `http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml`, 1990. A project to map the entire human genome.

[13] Norwegian University of Science and Technology. `http://www.idi.ntnu.no/~tagore/busstuc/`, 2010. A buss oracel.

[14] W R Pearson and D J Lipman. `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC280013/?tool=pmcentrez`, 1985. The FASTA algorithm.

[15] Pat Quinn. http://www.illinois.gov/PressReleases/ShowPressRelease.cfm?RecNum=4661%&SubjectID=49, 2006. From the office of the governor.

[16] Kim R. Rasmussen, Jens Stoye, and Eugene W. Myers. Efficient q-gram filters for finding all □-matches over a given length. *Universit at Bielefeld, Germany*, 2005.

[17] M. Sternberg. Protein structure prediction: A practical approach. *Oxford University Press*, 1997.

[18] Hugh E. Williams. An indexed approach to searching genomic databases. *Dept CS RMIT Melbourne*, 1998.

[19] Hugh E. Williams and Justin Zobel. Indexing and retrieval for genomic databases. *Springerlink*, 1998.

Appendix A

Code listings

```
static void indexDocs(IndexWriter writer, File file) throws IOException {
  if (file.canRead()) {
    if (file.isDirectory()) {
      String[] files = file.list();
      // In case of IO error
      if (files != null) {
        for (int i = 0; i < files.length; i++) {
          indexDocs(writer, new File(file, files[i]));
        }
      }
    } else {
      System.out.println("adding " + file);
      try {
        String sequence = "";
        String annotation = "";
        String docId = "";
        String line = "";
        BufferedReader buffer = new BufferedReader(new FileReader(file));

        while ((line = buffer.readLine()) != null){
          if(line.contains(">")){

            if(annotation.equals("")){
              annotation = line.replace(">EMBL_CDS:", "");
              docId = annotation.split(" ")[1];
            }
            else{
              Document doc = new Document();
              doc.add(new Field("DocId", docId, Store.YES, Index.NOT_ANALYZED ));
              doc.add(new Field("Annotation", annotation, Store.YES, Index.ANALYZED ));
              doc.add(new Field("Sequence", sequence, Store.NO, Index.ANALYZED ));
              writer.addDocument(doc);
```

```
                sequence = "";
                annotation = line.replace(">EMBL_CDS:", "");
                docId = annotation.split(" ")[1];
                }

            }
            else{
              sequence += line;
            }
          }

          Document doc = new Document();
          doc.add(new Field("DocId", docId, Store.YES, Index.NOT_ANALYZED ));
          doc.add(new Field("Annotation", annotation, Store.YES, Index.ANALYZED ));
          doc.add(new Field("Sequence", sequence, Store.NO, Index.ANALYZED ));
          writer.addDocument(doc);

          buffer.close();

        } catch (FileNotFoundException fnfe) {
          System.out.println(fnfe);
        }
      }
    }
}
```

```
public class NGramAnalyzer extends Analyzer {

  @Override
  public TokenStream tokenStream(String fieldName, Reader reader) {

    TokenStream stream = null;
    Tokenizer tokenizer;

    if(fieldName.equals("Annotation")){
      stream = new CustomWhitespaceTokenizer(reader);
      stream = new StandardFilter(stream);
      stream = new BiomedTokenFilter(stream);
      stream = new LowerCaseFilter(stream);
    }
    else if(fieldName.equals("Sequence")){
      tokenizer = new NGramTokenizer(reader, 4, 4);
      stream = new LowerCaseFilter(tokenizer);
    }
    return stream;
  }

}
```