NTNU
Norwegian University of
Science and Technology

A Multimedia Approach to Medical Information Retrieval

Aleksander Grande

Master of Science in Informatics
Submission date: June 2009
Supervisor: Herindrasana Ramampiaro, IDI

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

Abstract

From the discovery of DNA by Francis H. C. Crick and James D. Watson in 1953 there have been conducted a lot of research in the field of DNA. Up through the years technological breakthroughs has made DNA sequencing more faster and more available, and has gone from being a very manual task to be highly automated.

In 1990 the Human Genome Project was started and the research in DNA skyrocketed. DNA was sequenced faster and faster throughout the 1990s, and more projects with goals of sequencing other specie's DNA was initiated.

All this research of DNA led to vast amounts of DNA sequences, but the techniques for searching through these DNA sequences was not developed at the same pace. The need for new and improved methods of searching in DNA is becoming more and more evident.

This thesis explores the possibilities of using content based information retrieval to search through DNA sequences. This is a bold proposition but can have great benefits if successfully implemented. By transforming DNA sequences to images, and indexing these images with a content based information retrieval system it can be possible to achieve a successful DNA search.

We discover that this is possible but further work has to be done in order to solve some discovered issues with the transforming of the DNA sequences to images.

Keywords: DNA search, spatial data representation, content based information retrieval

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost i will thank my fiancee, Lene for her endless patience and ability to listen to my frustrations. My parents have also been very helpful, good advice and moral support from my mother and father when motivation was low has given me the vitality to continue working. I would also like to thank my supervisor Heri Ramampiaro for helping me choosing this thesis and for giving me good advice all through the process. A final, and big thank you to all of the guys at Gribb,Amund, Anders, Bjarne, Glenn, Hans Petter, Håvard, Håvard, Lars, Mats, Ole Georg, Robin and Øystein they have been great and their competence has been a great asset in times of trouble.

Chapter 1

Introduction

1.1 The problem and why should we solve it

The increased attention to bioinformatics and genome research has led to a higher demand of information technologies to support this research. In DNA searching today, the techniques used are mainly text-matching and full text indices. This proves to be a very precise search method, but is very computational intensive and slow.

The demand of new and more computational effective ways of searching in DNA sequences are crucial in order to bring DNA search out to the people. This thesis tries to use existing technologies in new ways, mainly multimedia retrieval techniques with genome data retrieval. There has been done very little research in this field.

The issues of searching in genome databases are many, they contain very high amounts of data, there is a lot of unnoticed data replication, and a lot of the useful data is surrounded by junk data. Techniques used today are not that good at approximate searches. An approximate search is a search where the system tries to find items that are not exactly like the query item, but similar. There are techniques in genome searching that use wild card searching and some that support it directly, but these are slow and exhaustive.

The nature of multimedia information retrieval, and especially content based information retrieval(CBIR), is that in order to work, they have to support approximate searching. In this thesis we[1] will try to use the properties of CBIR to my advantage and employ CBIR techniques to achieve

[1]Throughout the thesis, when it is referred to "we", it is referring to the author, this makes reading easier.

genome search in order to solve some big problems in genome searching.

CBIR is the field of searching in the content of non-text media, especially in images. In order to do a search through the content of non-text media, you have to use approaches that are good at approximation searches, mainly because a user never searches for the exact query, but something that is similar. So the search has to be fuzzy. Also CBIR systems are more useful the more images they can search through, so a good CBIR system can search approximately through large amounts of data, in a fast manner.

1.1.1 Applications and benefits

There are many applications for achieving better/faster approximate search.

Scalability of existing approaches

The two main approaches used today in DNA sequence comparison/retrieval namely, the variations of BLAST and FASTA, were developed in the very beginning of the "DNA-era". These are both approaches that uses heuristics for quick DNA sequence comparison with strings. They are reasonably fast and very precise, but they have serious drawbacks. The algorithms have not seen any major improvements since their release, and since they both rely on string comparison they have limited amounts of potential improvement. The only way to achieve higher performance is to enable the hardware they run on, to run faster. This fact combined with the increased speed in sequencing of DNA and the interest in DNA research, makes the amount of data to process very large.

If we can utilize the properties of CBIR to search in DNA or at least provide a CBIR search as a filtering step before passing it on to BLAST or FASTA to achieve higher performance.

Genome research fields

In genome research the ability to find DNA sequences that are related to each other is crucial in order to discover the function of a gene. The homology search is at the center of genome research. The homology search is a search where a scientist uses an existing DNA sequence containing a gene with a known function as a query to a DNA search. The search gives the scientist the possibility to find related genes and maybe undiscovered genes and map their functions.

CBIR used in a genome context shows very promising properties, that can allow speedy homology search, and a homology search that can provide new ways of finding related genes.

Disease discovery

There are many diseases and dysfunctions that are proven to be hereditary(diabetes, cancer, heart disease etc.), that these kinds of diseases are transferred through the DNA from child to their parents. If it is possible to increase the performance of searching and the broadness of the homology search, the search can be used on all newborn children. If every child that is born on a hospital has a full DNA search of their entire DNA sequence, prohibitive measures can be taken to deal with known diseases[2].

Police investigations

DNA is becoming more and more used in police investigation of crime scenes, it is taking over for the much used and proven fingerprint. But the time between an submission of a DNA sample and a result takes about 30 days at minimum[34], if we can decrease the time spent on a search for a suspect's DNA crimes could be solved faster. And if we could make a DNA search run on slower hardware it could be possible to do such a search in the field, by a police officer.

1.2 Problem specification

The title, "A Multimedia Approach to Medical Information Retrieval", is unspecific and not very descriptive. So some definitions are in order. We define multimedia approach as multimedia information retrieval or content based information retrieval. We can then continue with defining medical information retrieval and define it as DNA retrieval or, DNA search. So, in other words; using content based information retrieval to search in DNA.

Two central questions arise from the definition above:

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?*

[2]There are several moral implication with this, we mention this only as a possible application.

1.3 Scope

In order to answer the questions asked in Section 1.2 we chose to focus the scope of this thesis and make some limitations on the work done.

As stated both in Section 1.1 and Section 1.1.1 performance is used as a central argument to why we should seek new approaches to DNA search. But in this thesis we will not focus on measuring the performance of CBIR systems and comparing them with state of the art DNA search systems. We will focus on the task of using a general purpose CBIR system for searching in DNA data. Albeit, we have a notice of the performance with regards to data sizes and index sizes in Section6.5.

1.4 Thesis structure

The thesis is structured in 6 chapters;

Chapter 1

This chapter contains the motivation, and the problem specification for the thesis, and defines the scope in which we try to solve the problems defined.

Chapter 2

Chapter 2 gives the theoretical framework for being able to understand our work, and understand what this thesis tries to solve. We give some theory about information retrieval, evaluation of information retrieval systems, content based information retrieval, a brief introduction to biology and DNA, and last an introduction to bioinformatics.

Chapter 3

Chapter 3 deals with the state of the art both in DNA search and CBIR systems. We describe the FASTA and BLAST algorithms and also describe some popular indexing techniques for proficient storage of DNA sequence indices. The second half of the chapter deals with CBIR systems, it gives an overview of many popular CBIR systems and their properties.

Chapter 4

There are two main parts in Chapter 4, first we present the issues with DNA spatial data representation, and possible solutions for these problems, then we continue with describing the approaches we have selected as the most promising. We then continue with illustrating how we used LIRE to index and search the images generated from the proposed methods.

Chapter 5

The results are presented in Chapter 5, where we show how the different approaches performed, and compare them. We also try to evaluate the strengths and weaknesses of the approaches, and give a brief analysis of the results.

Chapter 6

In Chapter 6 we try to apply the results gathered in Chapter 5 to come to some conclusions on how our approaches work, and what we could have done differently.

Chapter 2

Background research

This chapter will present relevant research fields in information retrieval, evaluation of information retrieval systems, content based information retrieval, biology and bioinformatics.

In order to describe content based information retrieval systems and techniques, we need to look at the regular information retrieval system and terms used in this field. And to understand the different fields in bioinformatics we need a quick biology primer. The monolithic data amounts involved in DNA search is also illustrated.

2.1 Information Retrieval

The term Information Retrieval can be defined as: the part of computer science witch studies the retrieval of information from a collection of documents. Information retrieval aim at satisfying a user's information need, this need is expressed in a form of a query which the information retrieval system understands[7]. So, Information retrieval, or short IR, is the study of filling a user's information need.

The normal way of meeting the users need is to provide an interface where the user can ask or query for relevant information. There are many types of queries used, but the most used is keyword based, where the user provides some keywords in the query interface. And a list of relevant documents are presented to the user.

For a system to be able to do this, the collection of the documents must be searchable. A very naive way of searching in documents is of course to just scan through all documents and look for occurrences of the keywords specified, but as the amount of documents and the amount of information in these documents increases, this novel approach will take very much time to complete. Therefore indexes as been introduced. An index is basically a list of in which documents some piece of information occurs.[7]

2.1.1 The document

The term ”document” is central in information retrieval, and in a digital library context. In the digital library, an object is a document when it is[9];

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

In information retrieval we facilitate the retrieval of documents. But regularly the document is considered as a piece of information; a webpage, a book, an article, a image, a movie, a sound file etc.

All these things are considered as a document, and the goal of information retrieval is to give the user the possibility to retrieve the information stored in a digital library or a database in a quick and easy way without knowing the exact location of the document.

2.1.2 Indexing and searching in documents

Data retrieval, in the context of an IR system, consists mainly of determining which documents of a collection contain the elements specified in the user's query, however this is often not enough to satisfy the user's information need.[7]

To increase the efficiency of information retrieval, the data collection needs to be processed, we have to transform the collection so that it is searchable. This is called indexing.

In regular ,text based, information retrieval, where the data collection consist of textual documents ,the process usually consists of removing stop words. Stop words are words that have a high frequency across all documents in the collection, and therefore does not add any information that can be

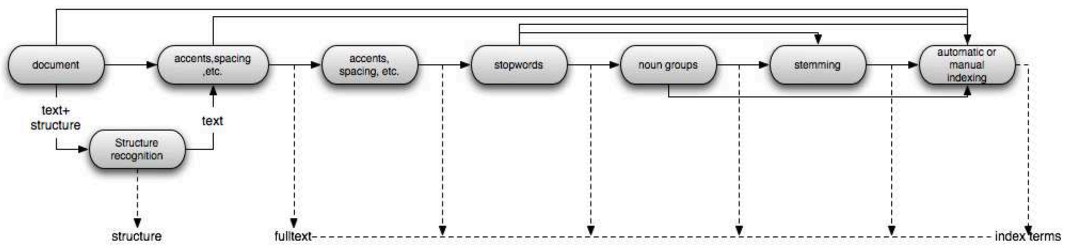


Figure 2.1: Overview of traditional document indexing.[7]

used to distinguish between documents. It enables the index to be greatly reduced in size and thus making it more efficient.[7].

Stemming [7] is an often used technique. Stemming consists of taking a word and removing prefixes and suffixes to it, i.e. "talking" becomes "talk", "fishing" becomes "fish". This enables the search engine to find documents that are related to the query terms.

The documents are then transformed in to a data structure for easy access and searching. There are three main techniques[7]; inverted files, suffix arrays and signature files. The details on how these techniques work will not be discussed, but the jist of it is that the processed documents are organized in such a way so that keywords from a query can point to which documents that specific keyword exists in, and usually where in that particular document the keyword appears in.

As outlined in Figure 2.1, the indexing process of regular textual documents consists of several (optional) steps, where each step either outputs some kind of information used directly in the indexing process and or outputs information to be used in later steps in the indexing process. In Figure 2.1 we see 7 steps;

1. *Document: this is the input to the indexing process.*

2. *Structure recognition: looks at the structural parts of the document, and extracts the structural properties of the document and information about these structures, and then passes the text to the indexing engine.*

3. *Accents, spacing, etc: removes characters with very low information value.*

4. *Stop-words: removes stop-words.*

5. *Noun groups: you can identify nouns, and take special care of these, for increased information in the index.*
6. *Stemming: reduce words to their grammatical root.*
7. *Automatic or manual indexing: based on the information extracted in the previous steps, create some kind of searchable index.*

The output from this process is a finished index, ready for use in a search engine[7].

2.1.3 Querying

Generally in an information retrieval system one or more query languages are used. A query language is basically a compromise between a user's need to express a demand for information and the system's ability to parse this language and make sense of it. It is the way the user asks the system for information. The query language is determined before the indexing starts, so that indexing methods and data structures used are optimized for a certain type of query language.

In text-based information retrieval systems, we generally have five different query languages; *keyword-based*, *single-word*, *boolean* and *natural language*.

The most known of these is keyword-based or single-word querying, the query language we know from most web search engines like Google, Yahoo and AltaVista. The user supplies one or more keywords, and the search engine processes these and returns the result set. The way results are presented in may vary, this is more closely discussed in Section 2.1.3.

Single-Word

Single-word queries are quite simply single words as a query. In both single-word queries and context queries, sentences are viewed as strings of characters. A word in the sentence is a consecutive sequence of words surrounded by delimiters or separators usually spaces, commas, periods, question marks etc. When the words are extracted from the documents they are added to the index.

The index then contain a link from a specific word and the document(s) it occurs in. It is also possible to have more specific information in order to locate the word inside the document.

When a user then searches for a word the process is quite simple; the system scans the list of words and then retrieve the documents it resides in. To support multiple words you use the same scheme for each word and the system then only retrieves the documents that are similar for all query words.

Context

A context query allows the user to specify a context where a certain word appears, or the context between a list of words. The queries are usually divided into two categories, phrase and proximity queries.

The phrase query is a collection of multiple single words queries, but their relative positions from each other are central.

The words in the phrase should be close together so if you search for the phrase "information retrieval is fun", and you find the word "information" at position 30 in a document, then the successive words must occur at position 31, 32 and 33.

For a proximity query the user specifies the context, for example the query can look something like; "information NEAR retrieval". This means that the search engine looks for documents where the word "information" appears in some distance from "retrieval". The distance can usually be specified by the user, but in our case the distance should be 1.

Boolean

The boolean query is the most basic query to process. The user specifies some words separated with boolean operators, usually AND, OR and NOT, that describes the document the user is seeking. If you have an index as specified in Table 2.1.3

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Anthony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Table 2.1: Example of index for supporting boolean queries[32]

Each of the words in the index specified in Table 2.1.3 have signatures, the word Brutus will have the signature 110100. This signature defines what documents it exists in, as shown in Table 2.1.3 it exists in ”Anthony and Cleopatra”, ”Julius Caesar” and ”Hamlet”, but not in the rest.

Since the user specifies queries in boolean operators, we can use these operations on the queries as well. Take the query ”Anthony AND worser”, translate that query to the signatures; 110001 AND 101110, apply the binary operator AND and the results are; 100000. This indicates that only ”Anthony and Cleopatra” contains both the words ”Anthony” and ”worser”.

The processing of boolean queries are very simple, you only use binary operators to calculate the results and these operations are possible to do entirely in hardware, thus yielding very high speeds.

Natural Language

Natural language, is the language you use to communicate, its what you say and what you write. You write some query in a natural language, i.e. ”Who wrote the book Introduction to Information Retrieval?”, then the query processor interprets this language and translates it into some query language that a computer understands, i.e. structured query language(SQL), and you actually get the answer to your question; ”Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze”.

Ranking

Ranking is the process of deciding what documents to return to the user, in a database with millions of documents there are often very many seemingly relevant matches for some query. I.e. the query ”information retrieval” in google has $4,150,000$ hits, but only the first 10 are presented to me, how to decide what documents to return?

For phrase and single-word queries an often used scheme is term frequency, for each document you count the occurrences of the query terms and sum them, and then sort the returned documents based upon the value that is returned. Also inverse document frequency, which is a measure on how many documents a term occurs in, can be used for ranking.

The ranking process can be greatly increased by using relevance feedback, this is a way for the system to map the actual relevance by asking the user which of the documents returned actually were relevant. And by storing this information, weights can be attached to a document in a certain query context, so when another user submits the same query, the system knows that certain documents are more relevant than others.[7]

Query expansion

Query expansion is used to help the user find other relevant documents than those returned by the initial result. This is a task where the search engine can analyze the top scoring returned documents in order to extract new query terms from these, and then query the collection again to produce a greater amount of returned documents and more relevant hits.

Another way is to use a thesaurus to extend the query with similar words and then perform the query. [7, 32]

2.2 Evaluation of information retrieval systems

Evaluation of IR-systems is crucial to satisfy the user's information needs. As stated in Section 2.1 the goal of information retrieval is to give the user the information the user wants. The evaluation of information retrieval is measuring how good we meet the user's need of information.

2.2.1 Test collections

In order to determine how good a certain query is we need to know what documents are relevant for that query. This is usually done by using a controlled test document collection. These test collection are collections created for evaluation, and benchmarking of information retrieval. They consist of a document collection, a set of queries and what documents are relevant for the different queries[32].

Cranfield

The Cranfield collection is considered as the first test-collection. It was created in the United Kingdom in the 1950s. It contains 225 queries with a lot of relevance statistics for the results and 1398 documents about aerodynamics.

TREC

TREC, or Text Retrieval Conference, is an initiative started by the American National Institute of Standards(ANSI) in 1992. It contains 1.89 million documents and 450 queries, which are documented with relevance statistics. It is considered by many as the de-facto standard for retrieval evaluation.

GOV2

GOV2 is also a initiative of the ANSI. It is a collection of 25 million web pages, and it is the largest web collection for research purposes.

CLEF

CLEF or Cross Language Evaluation Forum is a collection for testing cross language information retrieval. It has different test collection with relevance statistics for testing IR-systems against specific languages.

2.2.2 Precision and Recall

Precision and recall is a widely metric used for evaluating IR-systems, and gives a good measure on how a users information need is met.

Precision

Precision[7] (illustrated in Figure 2.2.2) is defined as the relationship between the number of relevant items retrieved and the total number of retrieved documents, or the percentage of the retrieved documents that are relevant. If I is defined as a certain information need(query). $|A|$ is defined as the total number of retrieved documents and $|Ra|$ is the number of relevant documents retrieved then precision is defined as: $Precision = \frac{|Ra|}{|A|}$

Recall

Recall[7] (illustrated in Figure 2.2.2) is a measure illustrating the relationship between the number of relevant documents which has been retrieved for a certain query and the number of relevant documents retrieved, in other words, the fraction between the number of relevant documents to the query and the relevant documents returned. If I is defined as a certain information need(or query), and $|R|$ is the number of relevant documents available in the database for I, and $|Ra|$ is the number of relevant documents retrieved for I, then recall is defined as: $Recall = \frac{|Ra|}{|R|}$

2.2.3 F-Measure(F1 score,F-Score)

The f-measure gives a value of the relationship between precision and recall, it is a weighted average of precision and recall. It is defined as;
$f = \frac{2(precision*recall)}{(precision+recall)}$

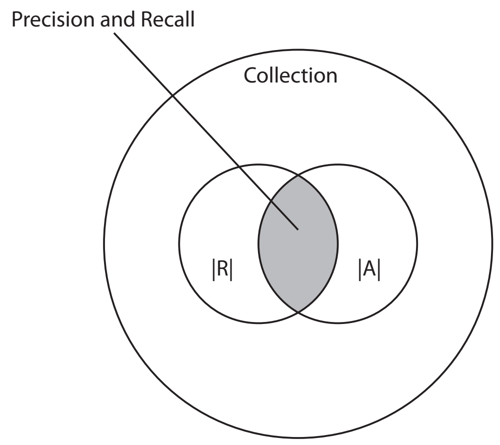


Figure 2.2: Illustration of precision and recall[7]

the resulting value gives a single point basis of evaluation, which can be good to have instead of multiple values of precision and recall for a range of queries.

2.3 Content Based Information Retrieval

Content based information retrieval (CBIR) is a collective term for information retrieval where the user query the actual content of the images,audio and video and not the metadata or some form of annotation. The techniques for images, video and audio are quite different and in this thesis we do not focus on sound and video, only images.

The way we search in the content of images is usually done is by performing an special kind indexing on the document collection. Where some properties of the image is chosen to be searchable, these properties are called features.

These properties are then extracted from all the images in the document collection and put into a searchable index.

When the user then wants to query the document collection, the user could do it in a couple of ways. Either supply a sample image as the query, which the system then extracts the same features from as stored in the index. Then these extracted properties are used as the query.

The other way is for the user to enter the desired properties of the image, it could be percentage of pixels in a certain color, it could be the roughness, the contrast, the regularity etc.

In some CBIR systems the shape of objects in the image are extracted as a features. If this information is present, the user could supply a query in the form of shapes and their colors, or the user can use a sketch interface to query the system.[1]

2.3.1 Feature extraction

Feature extraction is the process where the properties of the image that are selected for storing in the index is gathered and processed. The extraction is often performed on a processed image to retrieve distinct values that are easy to store, some of the most used features are; color information or color histogram, texture and shape features.[7]

Color Information

Color information (color histogram or color vector) is the most used feature, if we look at Table 3.3 most of the systems described use color information of some sort as a feature. It is easy to extract from images and gives fairly good results.

The actual information extracted is the color distribution of the image, it is usually represented as an n-dimensional vector where n is the wanted resolution of the color vector. Then the system iterates over all the pixels present in the picture and counts the colors in each pixel and adds this to the vector(see Figure 2.3). The vector is then added to the index.

Color information need a complete iteration over all the pixels in the image to extract information, but when it first is extracted and stored in the index, it is very fast to compute the distance between two color vectors.[7]

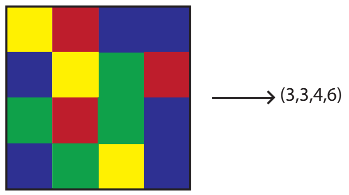


Figure 2.3: Image to vector conversion (yellow=3,red=3,green=4,blue=6)

One of the problems with using color information as a searchable feature is that it does not take in to account the spatial nature of images, it merely counts the pixel values of the images, so the results can be very ambiguous.

If the user supplies a sample image as a query, the images returned as results can be very visually different from the query image.

In Figure 2.4 two different images are displayed, one depicting a heart, and one with some squares. To a human they are totally distinct and very different from each other and a human sees no or little similarity between them(other than the similar colors). But, to the system they are totally identical. Albeit, the simplicity of the algorithm and the speed makes this a very popular approach in CBIR systems.

Some systems like WISE[54], also use localized color information to overcome these flaws, so the user can roughly specify the layout of the color in the images. This is done by dividing the image into sub-images and performing the color feature extraction on these sub-images.

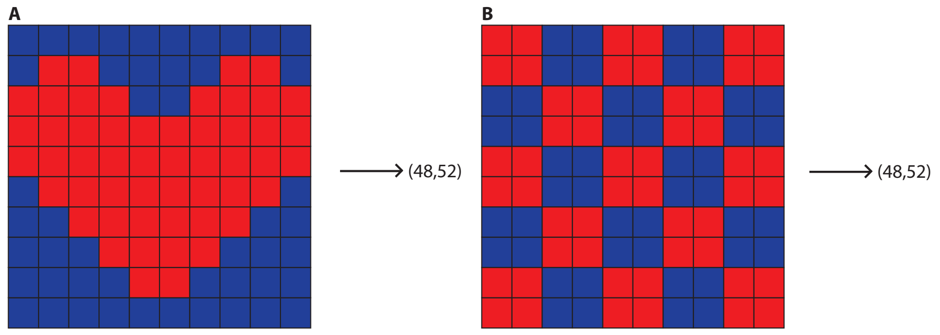


Figure 2.4: Two images that have equal color vectors but are different

Texture Information

Texture information is a good measure for measuring the similarity between two images. Because the features extracted are spatial, we overcome some of the problems of using color information. Texture is the tactile-visual properties of the image; the pattern of the image.

The texture can be extracted by various means, usually by thresholding a gray-scale version of the image. There are also versions where the image is transformed to the frequency plane[2] and the texture is studied as a signal. The metrics used for measuring texture are usually; coarseness, contrast, directionality, linelikeness and regularity.[12, 51, 24]

Coarseness is the difference in the variance between the highest gray values and the lowest, and the transition between these values, if they are very short the texture is more coarse, this can be sampled on a sub-image level across the image, or just at some regular interval to form a vector.

Contrast is the ratio between a certain pixel's color value and the average color value, and is usually sampled a number of times across the image, or from sub-images of the image.

Driectionality directionality is the measuring of lines and their angles in a texture, you first process the image to extract the lines, and then you measure their angle, and then you can produce a histogram of their angles, or a general angle which is the global texture direction.

Line-likeness is the measure of how the angle of the different lines are when compared with each other, you weight the lines with low similarity with negative weights and the ones with high similarity with high positive weights, then a scalar value is produced representing how much the lines are alike.

Regularity is computed by looking at sub-images, and tracking the patterns arising in directionality, line-likeness and coarseness and comparing the values from each sub-image with each other, and if they are similar you can determine a regularity in the texture.

Shape Information

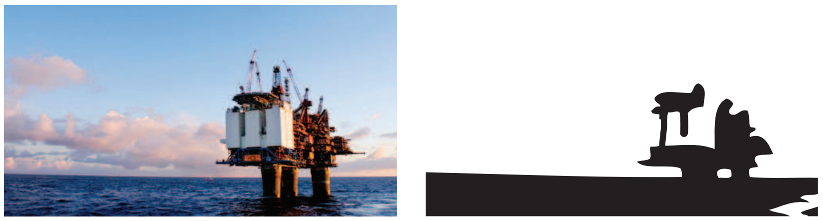
If you are able to segment the image and determine the shapes of the extracted regions, you have come a long way.

In ImageScape[25] this is done, and the system goes so far that it tries to find regions of water, sky and grass, so you can search images specifying regions with these properties.

Shape properties are hard to extract. And require lots of computation. If you search for a certain shape, the system should be able to search for a similar shape but in a different size, different rotation or different translation(position). There are two main approaches for extracting shape features;

Segmentation is a term used to describe the process of dividing the image into segments or regions based upon some common feature(texture or color). There are multiple algorithms for doing this, some popular are watershed, k-clustering and region growing. Then a tracer algorithm is run to extract these segments into polygons which can be easily compared to provide some measure of likeness.

In the example below, we see that we have two prominent segments extracted from the algorithm, and the shape of these can then be indexed as a set of polygons[18].



(a) Sample image of oilrig (courtesy of ntnu)

(b) Segmented image of oilrig

Figure 2.5: Example of segmentation

In *Edge detection*, we apply image processing filters to extract prominent edges of regions.

The most popular algorithms are, the Canny algorithm, the Nalwa algorithm, the Iverson algorithm, the Bergholm algorithm and the Rothwell algorithm. [18][29]. The details of these algorithms will no be discussed in this thesis. But the jist of the algorithms are that the prominent edges of the image are enhanced to such a degree that only they are visible in the image.

The lines extracted from the algorithms are then traced and represented as a vector of coordinates. Figure 2.6 shows a typical result of an edge detection algorithm.

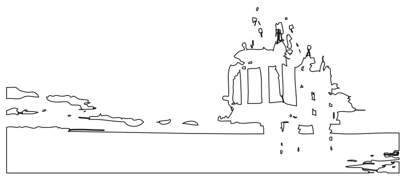


Figure 2.6: Edge detection on same image as in 2.5

2.3.2 Querying in CBIR

Querying content in images introduces some new issues; how do we query the information we have extracted from the images? Below we discuss some of the most popular methods of providing a query interface.

Query-By-Example

Query-by-example is a huge advantage in CBIR, and it is one of the goals that CBIR tries to achieve. When a user wants to search the CBIR system the user can just provide an image that is similar to the one desired. Then the system processes the query image. And uses the extracted features from the image to query the system.

Text

A text query in a CBIR system is quite different than from a regular IR system. It can be done by textually specifying some value of a feature extracted from an image. An example is, ”contrast:0.2”, or by smarter systems that actually can distinguish objects in the image, i.e. ”blue sky”.

The query text is then translated to a searchable feature, in the example, ”blue sky”, the color histogram can be set to have a very high value of blue in the top half part of the image.

Query by Sketch

The query by sketch interface is probably the most elaborate query scheme, both for the implementers of the system and for the user. It has high demands on the user to be able to draw what the user is looking for. The query interface must be easy to use, and the resulting query must be more flexible than query-by-example, because we must assume that the user are not able to create a sketch in a way that is as good as a regular image.

2.3.3 Partial Image Retrieval

When we talk about partial image retrieval it is in the sense of querying spatial features like texture and/or shape information. The process of indexing the document collection in such a way that makes this possible is not mundane.

In order for a system to query specific parts of an image, the image has to be segmented. There are many things to take into account when doing this; should the system segment the image regularly based on some constant? Should we look for regions of interest and then choose a segment size based upon the most prominent features in the image? Or should we just define that every image in the document collection is to be segmented into n segments?

All these different approaches gives different results, and have different needs with regards to computational power. There is no obvious answer, but specialized CBIR systems often select approaches to fit the system's need.

2.4 Genomics/Biology

In this section we give a primer on biology and genomics in order to be able to understand the motivation and the challenges involved in DNA search.

2.4.1 DNA

Deoxyribonucleic Acid, or DNA, is the building blocks of all known living organisms and some viruses. It is a program running inside them dictating the development and activity of the organism's body.[17]

The DNA molecule resides inside the cell-nucleus, where it is protected from the hostile environment inside the cell by the nuclear membrane. The primary function of the DNA is to be a vessel for heraditary trait, it basically provides the ability for different features to pass from one generation to the next in a certain specie, thus facilitating evolution.

A DNA molecule is a long linear molecule consisting of messages coded by four types of molecules called nucleotides. These are; adenine, cytosine, guanine and thymine or shorted down to A for adenine, C for cytosine, G for guanine and T for thymine and this makes up the alphabet of DNA.

The shape of the DNA molecule is the famous double helix. The 3D structure consists of the nucleotides which are connected to a sugar-phosphate backbone. An illustration of this spiral is shown in Figure 2.8.

In the molecule the two spirals intertwine to make up the double helix, and between these spirals the nucleotides form pairs, adenine with thymine and cytosine with guanine, are called base pairs.

Some parts of the DNA-molecule will contain information that code the creation of a certain protein. These are called genes and are the parts of the DNA molecule of interest to researchers, because they dictate the function of the organism.[44]

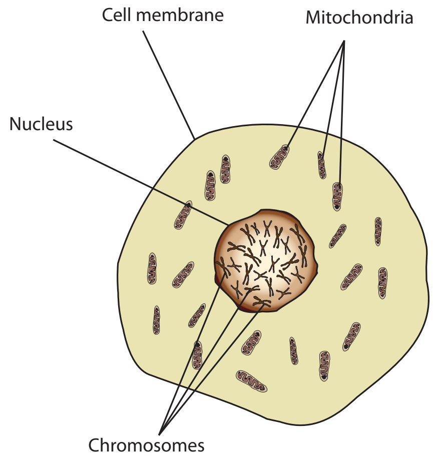


Figure 2.7: The cell

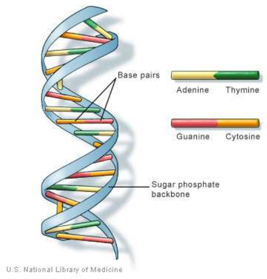


Figure 2.8: The double helix(courtesy of Lister Hill National Center for Biomedical Communications)

In order for a cell to produce a certain protein,enzymes in the cell nucleus "travel" along the backbone of the DNA molecule and read the nucleotides and make a temporary molecule called mRNA, this is a molecule that can travel through the nucleus' membrane, and out to the cell body, where the mRNA is used to put together the amino acids that make up the protein. There are 20 types of known amino acids and they can be combined into a vast amount of different proteins.

The DNA strands are organized in chromosomes. The chromosome contains about 60% protein and about 40% DNA. The mixture of DNA

Table 2.2: Brief history of the human genome project

Human Genome Project started	1990
Human chromosome mapping data repository created	1991
First detailed genetic-mapping achieved [1]	1994
First non viral genome fully sequenced	1995
Gene Map 98 released, contained 30,000 markers	1998
First human chromosome sequenced	1999
Human genome working draft published	2001
Completed sequencing the human genome	2003

and protein in the chromosome is called chromatine. The chomosome has a central core of protein where loops of chromatine attach. In the chromatine loops the DNA strands are wound around proteins(histones). These are believed to further loop and coil to make up the shape of the chromosome. The human body has 46 chromosomes or 23 chromosome pairs.[44]

2.4.2 The Human Genome Project

The Human Genome Project (HGP) was started in 1990 and aimed at mapping, the then estimated, positions of the around 100 000 genes of the human genome. In 1991, after a year, about 5,000,000 base pairs was mapped, estimated to be around 2 percent of the entire human genome. In 2001, a rough draft of the human genome was presented, and finally, in 2003 the HGP was declared as finished. By then 3,400,000,000 base pairs was sequenced with a accuracy of 99.99 percent[3, 37]

1

2.4.3 Evolution

Over time species change to adapt to their living condition[44], this is a genetic change and is the foundation of the diversity of all life.

The changes happen on a genetic level, and are changes on the nucleotide sequences in the DNA. The fact that two different species has evolved from a common ancestor is a well known fact, and a useful tool in biology.

[1]Genetic mapping or linkage mapping provides evidence of disease being transmitted from one generation to the next, it also provide helpful information about the position of a certain gene on the chromosome

If a scientist is working on mapping the functions of the DNA of some species, lets say a rice species, and is unable to determine the function of a certain gene. The scientist can look at a different specie of rice. If a sequence that is very similar to the sequence the scientist is studying is present in the other specie, it can be deduced that they have the same function.

This fact is crucial in the field of genome research, and crucial in the world of DNA search. The scientist perform what is called a homology search when looking for the gene in that other specie of rice. This is the bread and butter in DNA search.

When measuring likeness between two sequences the term evolutionary distance is used. It is the number of changes that has happened on the gene in form of substitutions, additions or removals of either, amino acids if we are studying protein sequences or nucleotides when looking at nucleotide sequences.

2.4.4 The Rice Genome Research Program

The Rice Genome Research Project (RGP)[41] was started in 1991, with the aims at sequencing different rice cultivars to foster the ability to create new cultivars that have higher yield, and can grow in new environments. The project was started by the Japanese Ministry of Agriculture, Forestry and Fisheries and is the leading contributor to the International Rice Genome Sequencing Project (IRGSP)[43], and together they completed the sequencing of the rice genome in 2004, its size was about 420,000,000 base pairs, relatively small compared to the human genome.

2.5 Bioinformatics

As mentioned in Section 2.4.4, there is put a lot of effort towards sequencing the human genome, and a lot has been achieved. One problem arises when having collected all this data; what to do with it?

There are vast amounts of data, as you can see from 2.3, the smallest human chromosome is nearly 51,000,000 base pairs, that is 103,922,194 distinct values in a sequence databank.

Not only the shear size of the chromosomes is a problem, the data is also very homologous. This presents large problems to bench scientist who wants to use this data in his or her work. There is just too much data to wade through it manually to look for regions of interest,instead there must be automated search tools that can do the labour intense jobs for the scientists.

When a scientist has located a gene at some location in the DNA and want to explore its function a homology search is in order. This is usually done in a protein database and not in a nucleotide database.

There are some simple reasons for this; the homologous nature of nucleotides (only four different types) gives a high probability when studying short sub-sequences that repeat themselves throughout the sequence, and that you get false positives when querying with short query-sequences.

As mentioned in Section 2.4.1, the protein sequences contains 20 different amino acids that gives a higher diversity and lower homology in the data than nucleotide databases, which in turn lowers the chances of false positives.

When querying a nucleotide database, the computational power needed is much greater. Since the data sets are many orders of magnitude larger. It increases not only the data stored in the database itself, but it also means that the query sequences given are much larger in a nucleotide database than in a protein database.

The results from a protein database search are much more easy to analyze since there has been a greater effort towards evaluating results in an protein database search than in results from a nucleotide database search. Therefore the theoretical frameworks for analyzing such results have been established.

One problem when querying protein databases is that the protein databases are never as updated as the nucleotide databases. This is due to the nature of the DNA-sciences. All DNA material that is sequenced is first stored as nucleotide-sequences. Since the conversion over to protein databases requires a lot of manual labour there can be significant delays, and errors in the protein databases[8].

The homology search gives a good indication on wether a given gene is present in some species, and it can also determine if that gene has mutated from one generation to another.

The search can be carried out in different ways. The user can supply wild-card characters in the query sequence, which tells the search engine that these characters can be replaced with any of the legal characters in the given alphabet(nucleotides or amino acids). Or the user can let the system automatically determine the similarity between two sequences without entering the wild-cards, this also means that you have to parameterize the allowed evolutionary distance between two studied sequences.

Human Chromosome	Sequence Length	Rice Chromosome	Sequence Length
1	245,203,898	1	58,438,333
2	243,315,028	2	49,173,368
3	199,411,731	3	44,894,443
4	191,610,523	4	42,702,029
5	180,967,295	5	39,417,126
6	170,740,541	6	42,019,208
7	158,431,299	7	40,173,439
8	145,908,738	8	39,854,202
9	134,505,819	9	31,509,377
10	135,480,874	10	28,385,764
11	134,978,784	11	35,887,929
12	133,464,434	12	33,460,123
13	114,151,656	-	-
14	105,311,216	-	-
15	100,114,055	-	-
16	89,995,999	-	-
17	81,691,216	-	-
18	77,753,510	-	-
19	63,790,860	-	-
20	63,644,868	-	-
21	46,976,537	-	-
22	49,476,972	-	-
X	152,634,166	-	-
Y	50,961,097	-	-

Table 2.3: Overview of the length of human and rice chromosomes (in base pairs) [28, 42]

2.5.1 Indexing and Searching in Genetic (DNA) Content

As illustrated earlier indexing of information is not trivial, it is a science in itself. It provides access structures that can drastically reduce the time needed to search trough immense amounts of information.

Indexing in a traditional sense, tries to reduce the amount of data to be searched and in a way extract the essence of the information into a small compact package that is faster to read. We do this by removing elements that are unnecessary, i.e. words that does not give the document any information.

It is not possible to remove as much data as we see fit without loosing the information stored in that data. The results from queries will be less relevant, and the overall usefulness of the IR system will deteriorate.

When dealing with DNA, the data is very heterogenous and its very hard to remove any data from the collection without making the querying worse. The information contained in a DNA sequence is mostly undocumented, processes like discovering protein coding regions, annotation and gene functions is used to document these.

There are many techniques[23] used to index and search DNA sequences. The most used are string based and uses what is called sequence alignment.

2.5.2 Sequence alignment

As mentioned above, a big motive for searching within genome sequence is to determine evolutionary relations between two studied sequences, or just to determine some gene function by doing a homology search. Sequence alignment has an excellent way of measuring and proving a relationship between two or more sequences. As mentioned in the Section 2.4.3, changes in the DNA happen all the time, some of the changes are persistent and some are not. These changes can be translated to insertion, deletion and replacement of nucleotides in the DNA-sequence.

The whole idea of aligning sequences is to shift one of the studied strings over the other to find the best matching position. There is almost no chance of having a perfect alignment, so we need approximate alignment. We first align the sequence on a position where it matches best, and then we look at the number of changes that has to be made on the query sequence in order to get a perfect alignment. These changes are insertions, deletions and replacement. The number of changes reflects the evolutionary distance.

By scoring the parts where the two strings match exactly with a positive score, and where they don't match with a negative score, a ranking system is achieved, and we can quickly sort sequences by their similarity.

There are several different algorithms who use alignment to find the best matches and they are discussed in more detail below.

Global vs local alignment

When doing alignments there are two; global and local alignment (you also have a hybrid between these two sometimes mentioned as glocal alignment).

Global alignment works by trying to match a complete sequence, this works only if the ,two or more, studied sequences are of roughly the same length. Then the system tries to align the sequence from start to end.

Local alignments can be used on sequences that differ very much in length. Here you try to align only parts of the sequence to prove some similarity. The hybrid method tries to align the start and the end of the sequences and then continue with local alignments throughout the rest of the sequence.

The most used method is local alignment. This lies with the fact that a sequence does not have a distinct starting or and end, it only has a length and a position on the DNAs strand.

So in order align two whole sequences with each other will not give any proficient results.

Also, when researching genes you are not looking for totally similar sequences, but rather sequences that can look quite dissimilar on the face but have some evolutionary relationship.

Heuristics

The Merriam-Webster's dictionary defines heuristic as "involving or serving as an aid to learning, discovery, or problem-solving by experimental and especially trial-and-error methods", and many algorithms use heuristics to increase their speed. This is usually done by generating some kind of support data structure which can, at an early stage, be used to filter out sequences,or regions of sequences, that are of little interest.

2.5.3 Searching

Searching in DNA is hard, the system must align a lot of sequences, and it must be done fast. We here present two of the founding approaches of DNA search.

Needleman-Wunsch The Needleman-Wunsch algorithm guarantees to make an optimal global alignment of two strings. It uses a similarity matrix to score each alignment in order to find the optimal sequence alignment.

The algorithm starts with creating a scoring table, where the first column is the first sequence to be compared. The first row consists of the second sequence to be compared. Thus the height is $Length(Sequence1)$ and width $Length(Sequence2)$.

The algorithm then creates a so called trace-back table of the same size, it is used to store the "path" through the scoring table that gives the optimum alignment. This is done by starting in the lower hand corner and following the pointers given in the trace-back table, and adding together the corresponding scores in the scoring table.

(a) The scoring table

		S	E	N	D
	0	-10	-20	-30	-40
A	-10				
N	-10				
D	-10				

(b) The trace-back table

		S	E	N	D
	↖	←	←	←	←
A	↑				
N	↑				
D	↑				

Table 2.4: The two initialized tables used in the Needleman-Wunsch algorithm

It then continues to initialize the scoring table by giving the top and left edge cells scores that ensures that we achieve a global alignment this differs with what kind of similarity matrix is used.

When the scoring table is initialized we start filling the scoring matrix. We calculate the score of each cell from a predefined scoring scheme. We then fill in the substitution matrix, by checking witch score up/diagonal/left is the lowest and then store a pointer to that cell. When all scores are computed and the trace-back table is full we simply follow the path to find the optimal alignment.

When traversing the trace-back table there are three possible pointers, a pointer that points; upwards, diagonal and left. If we encounter a diagonal pointer the two characters are perfectly aligned, if we encounter a upwards pointer a gap is introduced in the first sequence. If we encounter a left pointer, a gap is introduced in the second sequence.

(a) The finished scoring table

		S	E	N	D
	0	-10	-20	-30	-40
A	-10	1	-9	-1	-29
N	-10	9	-1	-3	-13
D	-10	-19	-11	2	3

(b) The finished trace-back table,colored cells indicate the traversal

●		S	E	N	D
	↖ ▨	←	←	←	←
A	↑	↖ ▨	← ▨	←	←
N	↑	↖	↖	↖ ▨	←
D	↑	↑	↖	↖	↖ ▨

Table 2.5: Scoring and trace back table from Needleman-Wunsch

1	Create scoring table
2	Initialize scoring table (w/negative edge scores)
3	Calculate score for current cell
4	Update trace-back table
5	Repeat steps 3 and 4 for all cells
6	Traverse trace-back table

Table 2.6: The basic Needleman-Wunsch algorithm

Smith-Waterman The Smith-Waterman algorithm performs a local alignment, it works both on nucleotide and protein sequences, and it is based upon the Needleman-Wunsch algorithm and the algorithms are similar but they give quite different results.

As seen in the paragraph above about the Needleman-Wunsch algorithm, the edge cells of the scoring table is initialized with negative scores in order to achieve a global alignment of the two compared sequences. The Smith-Waterman initializes the the edge cells are initialized with 0, this opens up the possibility of making local alignments on the sequences.

The algorithm then continues to search for the cell with the highest score, and makes it the starting point for the backwards traversal of the trace-back table, it then continues to traverse until the score stops growing.

The algorithm can also be expanded to e.g. traverse from the 5 highest cells.

1	Create scoring table
2	Initialize scoring table (w/edge scores of 0)
3	Calculate score for current cell
4	Update trace-back table
5	Repeat steps 3 and 4 for all cells
6	Traverse trace-back table to find optimal alignment

Table 2.7: The basic Smith-Waterman algorithm

Chapter 3

State of the art

In this chapter we explore the different techniques that is used to day in genome search, both alignment algorithms, and data structures for storing indices of DNA sequences. By studying these methods we can shed light on the complexity involved in the most used approaches for achieving homology search in genomes. The FASTA and BLAST algorithms are discussed in detail.

3.1 DNA search as we know it

3.1.1 FASTA(FASTP)

The FASTA algorithm, derived from the FASTP algorithm, was developed by David J. Lipman and William R. Pearson[39, 40]. FASTA is an heuristic approach to sequence comparison. The algorithm starts with creating a lookup table of the query sequence. The creation of this lookup table is parameterized with a parameter, K. K defines the length of the entries in the lookup table. To create the table the query string is split into K-sized tuples. For protein databases it is generally 1 or 2, for nucleotide databases it can be up to 6.

In the lookup table all the occurrences of the tuples in the query sequence are stored. To generate an initial score of the different sequences a diagonal dot-plot is used to find consecutive matching K-tuples. And thereby can give a positive score for connected matching K-tuples and a negative where there are gaps.

The 10 best matched sequences are then forwarded to the next step, which is a closer rescan through the remaining sequences. Here the K value is

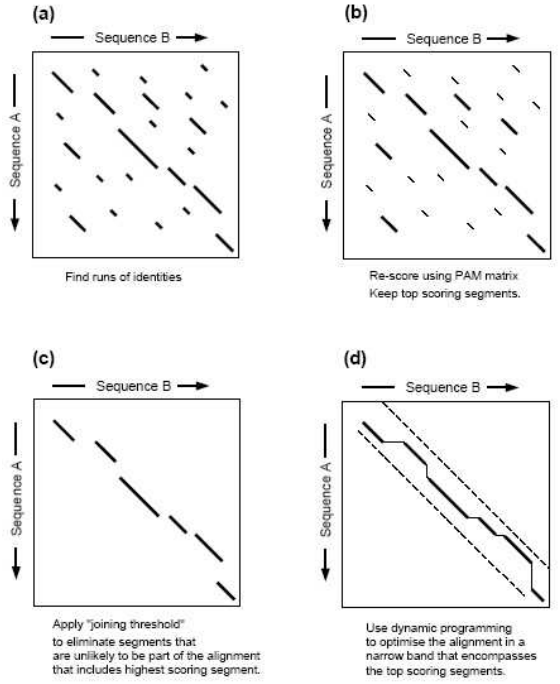

(a)
Sequence B
Sequence A
Find runs of identities
(b)
Sequence B
Sequence A
Re-score using PAM matrix
Keep top scoring segments.
(c)
Sequence B
Sequence A
Apply "joining threshold" to eliminate segments that are unlikely to be part of the alignment that includes highest scoring segment.
(d)
Sequence B
Sequence A
Use dynamic programming to optimise the alignment in a narrow band that encompasses the top scoring segments.

Figure 3.1: Overview of the dot-plot in FASTA, borrowed from [50].

often dismissed and the sequences are studied in more detail. The sequences are rescored using a scoring matrix (BLOSUM/PAM/Identity matrix).

The sequences with a higher score than the parameter S (cutoff value), they are forwarded to the next step, where we try to join misaligned sequences by approximating the alignment with gaps, then a banded Smith-Waterman algorithm[50] is used to calculate the optimal score for the different alignments.

3.1.2 BLAST

BLAST[4] or Basic Local Alignment Search Tool is an approach to increase the speed of sequence comparison with alignment. The BLAST algorithm works by first splitting the query sequence and the sequence to compare against into so called words. Words are sequence fragments of a given length.

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

Table 3.1: The basic FASTA algorithm

These words are then stored in a word list and compared with the subject-sequence in a substitution matrix i.e. BLOSUM[21]. A substitution matrix is a two dimensional matrix containing the probability of one amino acid being replaced by another, thereby giving a statistical chance of two words having some evolutionary relationship.

When querying BLAST, you give it a threshold parameter, T. This parameter defines how much a word have to match the subject-sequence in order to continue investigating the similarities. If the score is T or higher, the word size is extended in order to find high scoring segment pairs (HSP's). A HSP is a (local) aligned segment with no gaps, i.e. insertions or deletions, that has one of the top scores of the aligned sequences. This extension of the segment is continued until the HSP score begins to decrease. The score which the HSP's are rated with is, what BLAST calls, raw score. Here a simple scoring scheme is used, complete matches in the aligned range is given a positive score and gaps are given a negative score.

All the remaining HSP's that have a higher score than the S parameter, the cutoff score, is put in a list and the significance of the HSP score is evaluated.

This is usually done manually by a human, but to make it easier for the users of BLAST some filtering can be done to remove some of the hits. This is done by filtering out repetitive regions and regions with low complexity. The results are then verified by a human.

There is also a method of evaluating the result, by using the Gumbel extreme value distribution(Gumbel EVD), witch is a function that calculates the probability of a high scoring region appearing just by chance. If the HSP scores better than the EVD function it can, most probably, be considered as a real similarity.

There are also optimizations to be made in the input query, where the low complexity regions can be removed before further processing of the query continues.

1	Remove low complexity/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 3.2: The basic BLAST algorithm, *optional

The flavors of BLAST

The BLAST algorithm has some variations, its different settings where it is used[27, 5];

- BLASTN - searches through a specified *nucleotide* database and returns the most significant hits.
- BLASTP - searches through a specified *protein* database and returns the most significant hits.
- BLASTX - translates (six frame translation[1]) the *nucleotide* input to a *protein* sequence and searches a specified *protein* database and returns the most significant hits.
- TBLASTX - translates (six frame translation) the *nucleotide* input to a *protein* sequence and searches against a specified translated (six frame translation) database.
- TBLASTN - searches through a specified translated *nucleotide* database with a *protein* sequence.
- MEGABLAST - searches trough a specified database with multiple queries by concatenating the sequences.
- PSI-BLAST - searches through a database with a query sequence, then takes the most similar sequences from the query and generates a new sequence based on their most prominent features and uses the generated query to search the database again to find distantly related sequences.

[1]Frame translation is a way to determine the resulting protein sequence from looking at a nucleotide sequence

3.1.3 Indexing approaches

As described in Section 2.1.2 indexing techniques are widely used in regular information retrieval, it primarily does two things; relieve the search algorithms by decreasing the amount of information to be searched through and it provides the search algorithms with data structures that increases the performance of the search algorithms. As mentioned in Section 2.5.1 there are some problems with the removal of information, but we can decrease the sheer amount of data stored in the data structures, and we can optimize these structures in order to search faster. Below are some of the approaches to achieve indexing of DNA databases.

Suffix tree

The suffix tree[22] has some good properties when it comes to searching. As its name reflects, it uses the suffixes of a string to generate a tree. This property means that there is less data to be searched through, and by traversing the tree you can easily determine a match or a mismatch.

To create a suffix tree for a string S you begin by inserting an empty root node. Then you create a list of the suffixes in S, sort them by size (longest first). Continue with the top element of the list, and for each suffix you insert all characters of the current suffix into the tree by starting at the root node as the current node, and iterate through the tree. For each character of the suffix you check if the current node has any children containing that character, if it does, that node is then set to be the current node, if there are no node containing that character a new node is inserted and that node is set to be the current node.

When you reach the end of the current suffix the last node visited is set to be the index of that suffix.

To search a suffix tree T for a certain string S. Start with the root node as the current node, then you compare the first character,C of S with the child nodes of the root node, if C has a match in the child node then C is removed from S and that child node is then set to be the current node.

This is repeated until you have examined all characters in S, when the end of S is reached you know that T contains S. If you at some point have no child nodes of the current node that contains the current character then T does not contain S.

In order to make suffix trees work in a database setting, you will have to generate trees for all stored sequences, and for all queries to the database you would have to use the procedure described above.

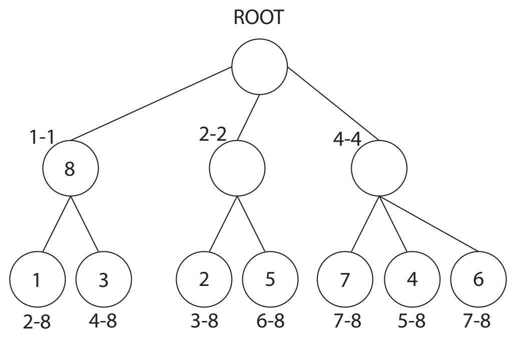


Figure 3.2: Example suffix tree of sequence: ACATCTTA

Q-grams

A q-gram[11, 33] or an n-gram, is a subsequence of length q of a given sequence. In order to explain DNA indexing with q-grams we have to define a new term, the edit distance. The edit distance is "the minimum number of edit operations (ie. insertions, deletions and substitutions) of single characters needed to transform the first string into the second"[11].

So, given a sequence S, you can get all its q-grams by sliding a "window" of length q over the entire sequence and stopping at the end, thus the number of q-grams is: $|S| - q + 1$.

X. Cao. et. al. proposes an indexing scheme where the database index consist of two levels. All q-grams that are possible to make in the sequences are devided into clusters. In the second level a transformation is applied to the sequences to form what they call c-signatures based on their q-grams. These c-signatures are then inserted into a tree structure called c-signature trees(c-trees). The qClusters are quite straight forward to build, first all the q-grams of the sequence is generated, then you gather them into groups based upon some likeness parameter.

The hash table is created with hash function whos parameters are both the q-grams themselves but also which qCluster it belongs to. Then a signature of each q-gram is created, the signature is a bitmap, where each bit in the bitmap represents what positions the different q-grams exist at.

To illustrate this we give an example taken from[11]: Take the sequence P="ACGGTACT". P's signature will then be [01 00 00 11 00 11 10 00](for q=2). If we then take a q-gram from P, "AC", we see that it occurs twice at position 0 and 5, thus those bits are set to 1. With the matematical formula;

$SDist(sig^c(S), sig^c(P)) = \sum_{i=0}^{k} -1|u_i - v_i|$

where v is the bits in the signature of sequence S and u is the bits in the signature of sequence P we can calculate the distance between the two signatures. The signatures of the sequence is then generated into c-tree, a rooted tree optimized for searching. We will not go into details regarding the construction of such a tree, it is similar to suffix tree building, we show an example is shown in 3.1.3.

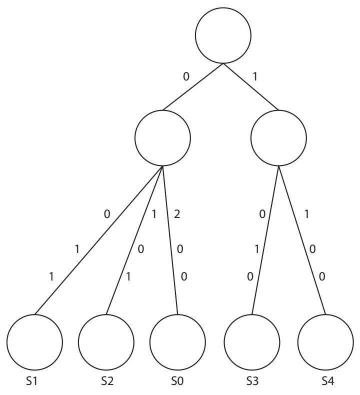


Figure 3.3: Signature tree for sequences: S0="ACGGT", S1="CTTAG", S2="ACGTT", S3="TAAGC" and S4="GACGT"[11]

The interesting part here is the query process, when given a query sequence S, a hash key is generated based on the q-grams present in the query string. The hash key is looked up in the hash table built in the indexing process, and the neighbors of the hash key in the hash table are gathered, and encoded. If their edit distance is within the specified range, the algorithm continues to store them in a candidate set C. This concludes the first level of filtering. In the second level, the signature of the query sequence is computed and the signature is then used to search through the signature trees built in the indexing process.

R-trees

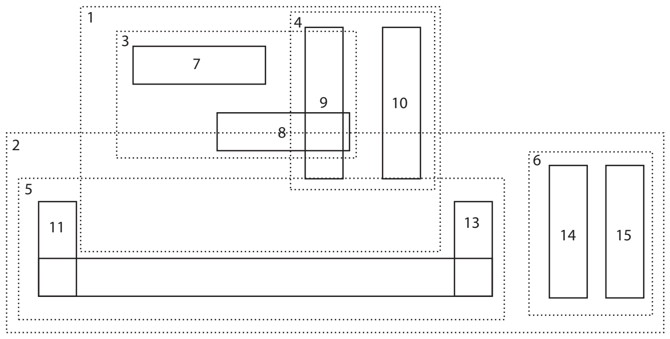
R-trees as well as suffix trees are well used data structures in information retrieval. While suffix trees are good for indexing sequential data, R-trees are constructed for indexing spatial data, or multi-dimensional data. A problem to overcome, is the one dimensional nature of the DNA strands, if we can represent the DNA sequence in a multi dimensional matter we can use R-trees as an index.

W.C Kim et. al.[19] proposes an algorithm for doing this. First you allocate a window of a specified size on every position of the sequence to be indexed. For each window a signature of that window is created. This signature is represented as a four dimensional rectangle. The following formula is used: $BS(W_i) = (([min_a, max_a], [min_c, max_c], [min_g, max_g], [min_T, max_T]), i)$ Where themin_x andmax_x represent the minimum and maximum occurrences of the nucleotide characters of the window to be indexed. For a sequence there are bound to be signatures that overlap partially or totally. Some of these sequences can be removed in order to save storage space.

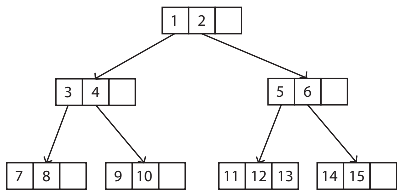
In an R-tree, a subtree compose a n-dimensional shape, and its size determines new insertions into the tree, also the R-tree has a limited number of child nodes (two or three are often used). When building the R-tree, you take the first window, this window becomes the root node. You then continue with going to the next window. If that window can be inserted under the root without making its bounding box bigger than the root node itself you insert it there, if the node itself becomes the root node. So the general algorithm for inserting nodes into the R-tree, you traverse the tree until you find all the nodes where there is free space for an insertion, and select the one that creates the smallest bounding box for that subtree.

W.C Kim et. al.[19] mention three different ways of searching in the R-tree, for an *exact match query* you construct a 4-dimensional query rectangle of size $([min_a, max_a], [min_c, max_c], [min_g, max_g], [min_T, max_T])$. The same is done for a *wildcard match query*, but here allmax_x variables are increased with the number of wild cards in the query sequence. For a `k-mismatch query`, a query where you tolerate a edit distance of k, the query rectangle is constructed by adding k to the max_x values, and subtracting k from the min_x values, if the min_x value becomes negative, it is set to 0.

We begin the actual query process by searching for all index rectangles who overlap the query rectangle and add them to a candidate set. A verification step is done for each of the candidates where that window is looked up and checked if it actually matches. If it does it is returned as a hit. W,-C Kim et. al.[19] also proposes a more complex index structure where the rectangles of the index are spread more eavenly in the vector space, this involves weighting the windows and extracting more properties of the windows to make them more unique, we will not go into those details here.



(a) Example spatial data



(b) Example r-tree of data

Figure 3.4: Example r-tree(inspiration from[45])

Wavelets

A wavelet[15] is a way of dividing a continuous signal into components in order to analyze them. You look at the genome sequence as a signal, for instance you take the sequence "a c a a t t a c t t a a t a g a a a a" and you view that as a continuous signal, and apply a wavelet transformation function to it, Elharti et. al. proposes the HAAR wavelet transform function which is represented by; $W_{jk}(x) = W(2^j x - k)$. By applying this transformation, we can use variance measuring methods to measure the similarity between two given sequences. We will not go into the details on how this is done, as it is far out of the scope of this thesis.

3.2 CBIR today

Here we will present a selection of current CBIR solutions in use, the features they extract, and how these features are used in a query context.

3.2.1 Alexandria Digital Library(ADL)

Developed by the University of California the ADL(now known as the National Geospatial Digital Archive) was made to be used with the vast datasets of geo referenced information[31], from maps created in medieval times to recent satellite imagery the ADL was constructed to make all this data browsable and searchable.

To search in the ADL, the user is presented a two dimensional map and with this map, the user zooms in/out and pans around to select a region of interest. The user can also select what kind of image he/she wants, and also have a textual query to filter out wanted resources.

The indexing of this database is fairly manual, there are few or none features extracted from the images and the query process is also manual, the user must browse around until the wanted resources are found. Whether or not this is a CBIR system, can be discussed, in[31] there are feature extraction tactics discussed, but as far as the user interface goes, there are no sign of possibilities for a user specified query within these features.

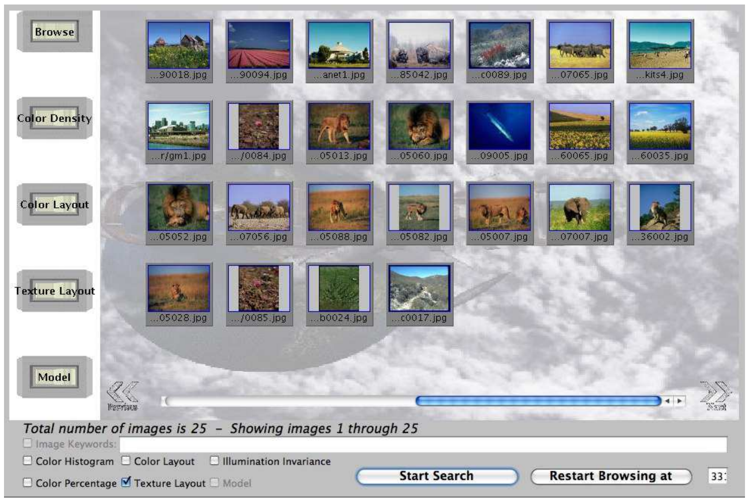
3.2.2 Content-Based Image Retrieval from Digital Libraries

The CBIRD[35] system was created at the school of computing science, Simon Fraser university in Canada. It is a true CBIR system, it uses a color histogram, color layout, and texture layout. When querying the database the users have interfaces for specifying the most prominent colors in the images, the texture layout and the color layout or you can search with the features of a specified image. The user can select anyone of the features, or none. When the search is complete the user is presented with a list of matches where the images are represented with thumbnails and can browse through the hits.

3.2.3 Diogenes

Diogenes[6] is a web search CBIR system for retrieving images of persons, portraits, group photos etc. It scans through web pages and looks for images with people in them, it then associates that page with the person's photo.

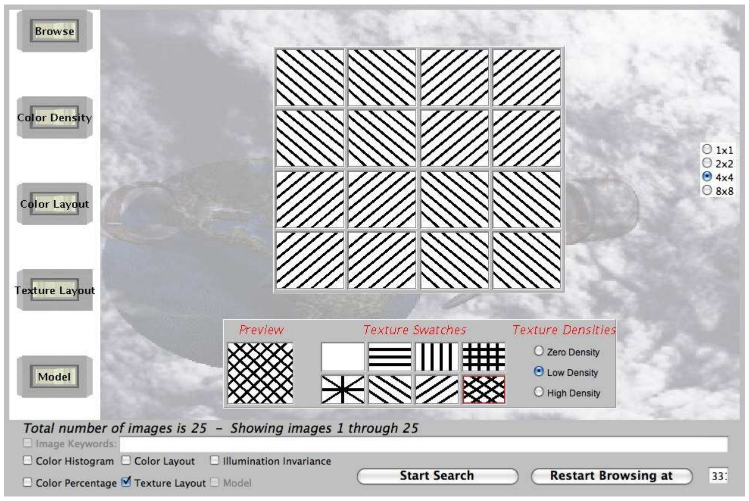
It uses two main features to accomplish this, the first one is a facial recognition algorithm which determines whether the image at hand is of a person. The second feature is the context of the webpage around the image, which is parsed from the HTML page. The system only supports textual queries for the persons name, and it then retrieves all photos indexed of this



Total number of images is 25 – Showing images 1 through 25

☐ Color Histogram ☐ Color Layout ☐ Illumination Invariance
☐ Color Percentage ☑ Texture Layout ☐ Model

(a) Search results



○ 1x1 ○ 2x2 ● 4x4 ○ 8x8

○ Zero Density ● Low Density ○ High Density

Total number of images is 25 – Showing images 1 through 25

☐ Color Histogram ☐ Color Layout ☐ Illumination Invariance
☐ Color Percentage ☑ Texture Layout ☐ Model

(b) Example texture search parameter

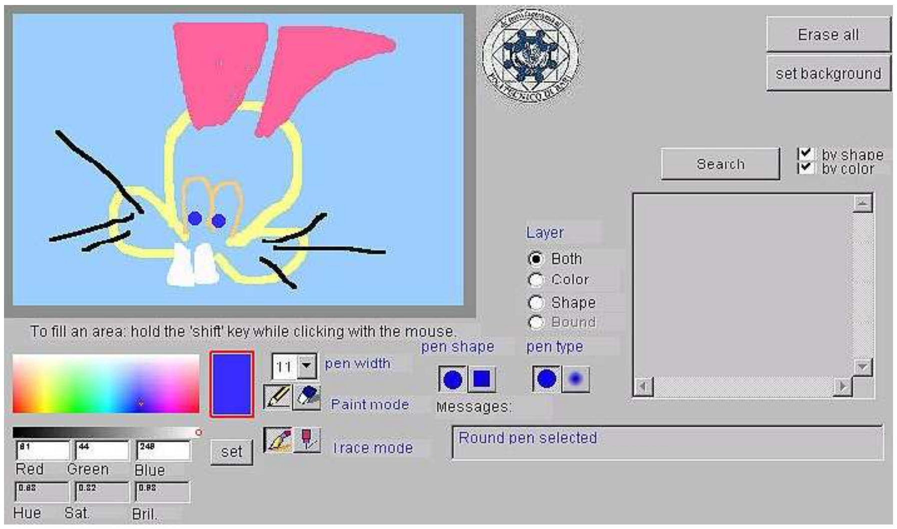
Figure 3.5: Screenshots from CBIRD

person. The results are then presented as a list with thumbnails sorted by relevance.

3.2.4 DrawSearch

DrawSearch[46] is a web-based search util, exploring the possibilities of query-by-drawing/query-by-example. It indexes shape, color and texture features. The color features are calculated by splitting the image into 4 smaller images and calculating the color features from these four images. The shape information is obtained by calculating fourier descriptors and the texture information is obtained by Gaussian Markovian Random Fields.[52]

The query process in DrawSearch3.6 is really interesting, the user is presented with a user interface where the user can sketch an image that looks somewhat like the desired image and this image is then processed and used in the query. The results are then displayed in a list with thumbnails ordered by similarity.



To fill an area: hold the 'shift' key while clicking with the mouse.

Round pen selected

Figure 3.6: The query input in DrawSearch(illustration from http://aa-lab.cs.uu.nl/cbirsurvey/cbir-survey/node14.html)

3.2.5 Excalibur Visual RetrievalWare(EVRW)

Excalibur[53] is a library for indexing in a CBIR system. It has an impressive list of feature extraction methods; color, shape, texture, brightness, color structure and aspect ratio. The system is based upon neural network methods, and automatically indexes binary patterns in digital information. There are little information about the specifics on EVRW.

3.2.6 Fast Object Color-based Query System(FOCUS)

FOCUS[30] was developed at the Department of Computer Science, University of Massachusetts. The images are indexed by extracting a color histogram feature. Each image is divided into blocks of 100x100 pixels, and each of those blocks have their color histogram in the HSV color space, and some clever math is employed to extract all the unique peaks of all the blocks. These peaks are then combined into a list and makes up the image's index. All the index elements are stored in a B+ tree, with their hue value as the primary key. The querying consists of a user selecting an entire existing image from the database, or the user can create a custom query by selecting a region of interest in one of the images in the database. The user is then presented with a list of all the images matching the query, ordered by relevance. The user can now select to perform a refined query, where the returned images are queried on the 100x100 pixel blocks initially created, and thereby spatial color information can be matched.

3.2.7 Finding Regions in the Pictures(FRIP)

Developed at the Yonsei University in Korea, FRIP[26] is designed to segment regions in images. When an image is segmented, the different regions in the image is then processed to extract, color, texture and shape features. The texture is extracted by using a biorthogonal wavelet frame[26], which then the direction is extracted from. The shape features are extracting centroid-boundary distances(see Section2.3.1).

The color feature is the average color value represented in RGB color space. When querying, the user selects an image from the database as the basis of the query, the system then segments the image into distinct regions, and then the user selects the region of interest which is then used as the query, the relevant images are then retrieved and shown in a list sorted by similarity to the query.

3.2.8 ImageRover

ImageRover[47] is developed at the Computer Science Department at Boston University. ImageRover uses texture and color features to query images. The system divides the image into five sub-images where the color histograms are calculated, and the texture orientation is calculated using steerable pyramids. The user is prompted with a text input, where keywords are entered.

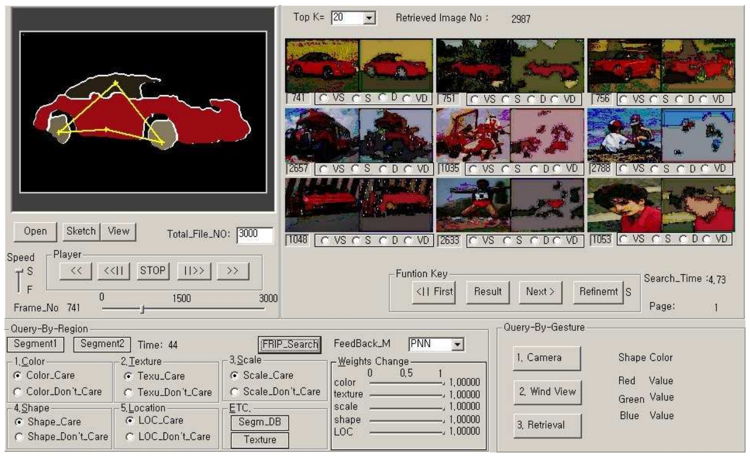


Figure 3.7: Shows a query and the result presentation of FRIP (from http://cvpr.yonsei.ac.kr/cvpr/frip/version.html)

Relative images are then retrieved, and the user selects images that are similar to what the user is looking for, then those images make up the query.

3.2.9 ImageScape

ImageScape[25], developed at the Department of Computer Science at Leiden University is another search by skecthing system. The system extracts the edges prominent in the image, and stores these as vectors, in addition to average color information, gradients and areas of turbulence are stored. The user queries the system by sketching an image, the user is instructed to just draw the contours of the image he/she is looking for. The user can also drag certain icons onto the canvas to illustrate i.e. water/tree/face 3.8. The user the gets the results as a list sorted by relevance.

3.2.10 Leiden 19th Century Portrait Database(LCPD)

The LCPD[36] is also developed at the Leiden University, and its main goal is to index portraits. The main feature here is the color values. The system has multiple indexes where the user selects what index to use. There are one index for the intensity of the pixels in the image, there is one binary (threshold) index, and there is one gradient index. There are also variations on these different indexes, where some of the indexes(projected indexes) has reduced index-sizes, by not applying a pixel-per-pixel index, but rather using

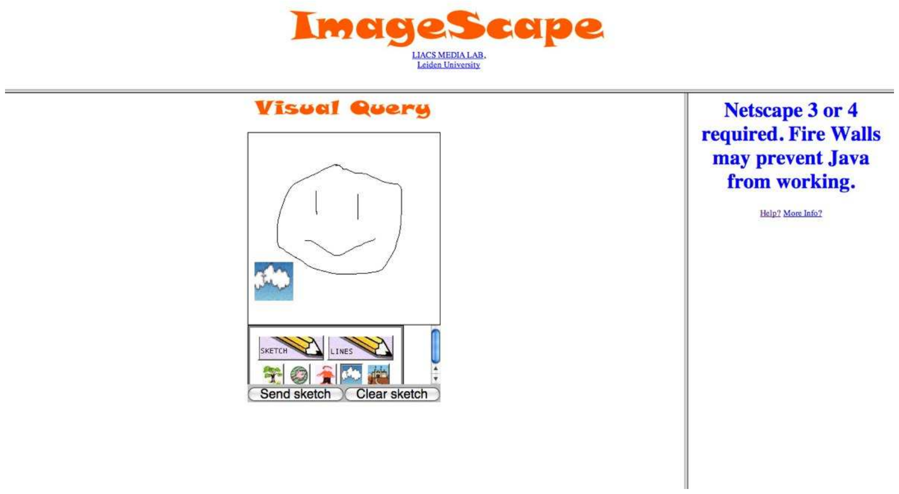


Figure 3.8: Shows the sketch window from ImageScape

larger portions of the image. The query process consist of the user selecting the desired search index, and then browsing to an image to select one that is fitting as a query. 13 of the matching results are returned plus the search image, and you can continue the search process by clicking one of the result images.

3.2.11 Lucene Image Retrieval(LIRE)

Lucene Image Retrieval is an open source project, and is an extension to Lucene[20] search engine. It extends the document type in Lucene to not only include text-documents, but also images. Tt indexes images with a color histogram, color layout, and edge outline information. As it is a programming library, Lucene does not provide any user interfaces for querying and indexing, but it provides API's for interfacing with it from Java. It is easy to implement and can be included in a project with little extra code.

3.2.12 Multimedia Analysis and Retrieval System(MARS)

MARS system was developed at the Department of Computer Science at the Univiersity of Illinois, and then development continued at the Department of Information and Computer Science at the University of California.[38] It supports color, texture and shape features, and also textual annotation of the images. The color is represented in HSV color space by histograms, the texture is represented by one coarseness histogram and one histogram for the direction of the texture. There is also a scalar value representing the contrast. The system supports layout information of these features, in order to do this, the image is divided into 25 sub-images, and the features are extracted from all these images. The user queries by a boolean query, and the feature values are specified either by using an example image, or by setting the values directly with for instance a color selecting control.

3.2.13 Multimodal Information Retrieval System(MIR)

MIR[49] has a more traditional approach to searching, it employs text mining techniques and natural language processing to analyze captions of images, in order to determine the content of the image. It also uses face detection to further study the content of the image, if any faces are detected, the detected faces are removed from the image and a color histogram is computed from the rest of the image. The user queries the system either with a text string, an image topic which is selected from a predefined list, or with an image.

3.2.14 Picasso

Picasso[10], developed at the Visual Information Processing Lab at the University of Florence, uses color regions and shapes and the relationships between them to measure image similarity. It segments the image based on the shape, and then measures the colors in these regions. It repeats this process on the image at different resolutions. The user can query the system in three ways, either by drawing a shape(outline) and then paint the shapes with colors (this can also be done automatically with a sample image). Or the user can sketch and image and lastly the user can select a sample image from the database, and use the texture from that image to query the system with.

3.2.15 PicHunter

PicHunter[13] developed by NEC Research Institute, it is a quite novel search algorithm based upon color histograms in different color spaces. The user queries with an example picture, but the smart part about the system is the relevance feedback. It looks at the action of the user when the initial results are presented and when the user marks images as relevant the system then uses these images to further refine the search. This way the system learns from the user, and this knowledge is put back in the system, so the system gets smarter and smarter.

3.2.16 IBM Query By Image Content(QBIC)

Developed by IBM Alamaden Research center, computes average color vectors, and color histograms in Lab, Munsell, RGB and YIQ color space, the color histograms are created by averaging the colors in groups of 3xn pixels. The system also computes texture direction, contrast and coarseness. The system also supports shape features, where the area of the shapes in the image are calculated, the central axis of the shapes are determined, also properties as circularity and irregularity are indexed. The query process in QBIC is either by selecting an existing image in the database, or the user can sketch an image, and there are interfaces for constructing texture and color queries by selecting swatches from a library.

3.2.17 Similarity-based Multimedia Retrieval Framework

Developed by Center for Geometry, Imaging, and Virtual Environments at the University of Utrecht in the Netherlands, the Similarity-Based Multimedia Retrieval Framework, SMURF indexes color histograms, texture and quite advanced shape information. The system is capable of polygon matching, and they have a test suite where hieroglyphs are indexed as an example and it retrieves them with great success. The query process is simple, you start out with a random selection of images and select one as your query source and the results are then shown, this query process can be repeated as many times as the user wants.

3.2.18 Octagon Java CBIR System

Octagon is a java CBIR system, developed by Jukka Viitala. It indexes images based with color histograms and structure, the documentation is somewhat lacking, and does not go into details on how the images are indexed. The interesting thing is the query system, the user can either query by selecting an existing image in the database, but can also supply an image. Before the user queries the system, the user can tweak the balance on how the color and structure feature is weighted, by moving a slider you can choose to prioritize either one of them more than the other.

3.2.19 Gnu Image Finding Tool

Gnu Image Finding Tool or GIFT, was developed at the University of Geneva under the name Viper. It is now maintained by the Free Software Foundation and is an open source CBIR framework. it supports textual annotation, color histogram and average color. The query interface is highly implementation specific, but GIFT supports an open XML-based protocol for easy front-end implementations.

3.2.20 VisualSEEk

VisualSEEk[48] is developed at the Image and Advanced Television Lab at Colombia University in New York, USA. The system segments the images into regions with similar color compositions, and for each of these regions their positions and sizes are marked and some sub-region color information is also stored. When a user queries the system, he creates regions of the desired size, and then specifies the color of these regions, the regions can also overlap.

3.2.21 WebSeer

WebSeer[16] is developed at the Department of Computer Science at the University of Chicago and is a CBIR system for web content. It crawls the internet in search of images, and it tries to separate photographs and drawings. The images are separated by measuring the numbers of colors used in the image, and gradients and saturation of the color. Photographs are then processed with face detection and they employ text filtering techniques to extract keywords, captions and other textual information about the image. The user queries by supplying keywords, also properties as how many faces in the image.

3.2.22 Wavelet Image Search Engine(WISE)

WISE[54] is developed at the Department of Computer Science at Stanford. It uses wavelet transforms to store color layout information extracted from the image. The image is scaled to 128x128 pixels, and transformed to a custom color space. It then uses wavelet transforms on all pixels, and uses the properties of this wavelet transform to only extract some of the wavelets and stores these as the index. The user can query the system either by an example image, or by sketching.

3.2.23 Overview and discussion

In Table 3.3 the features of the CBIR systems described above are listed. To fit the table on one page we had to condense the text a bit so we start with giving some explanations to the abbreviations used in the table headings;

- *Text* is textual information, context of appearance etc.
- *Tex.dir.* is texture direction
- *Tex.prop.* is texture properties, that is properties other than direction or layout.
- *Tex.lay.* is texture layout
- *Clr.hist.* is color histogram
- *Avg.clr.* is average color
- *Clr.lay.* is color layout
- *Shp.prop.* is shape properties.
- *Shp.vec.* is shape vectors, that is complete shapes are indexed
- *Face.reco.* is facial recognition.

The table shows that color information, both as color histogram, average color and color layout is prominent in all systems. Texture properties are also used alot. The table also illustrates the diversity of CBIR systems in use today, and how specialization of CBIR systems is possible, also it illustrates that no CBIR system uses all features, and this is for a simple reason; the more features you have to extract, index and search through the slower the system is.

Table 3.4 shows the different possibilities of query interfaces, most of the systems require that the user uses an image that is already indexed.

	Text	Tex.dir.	Tex.prop.	Tex.lay.	Clr . hist.	Avg.clr.	Clr.lay.	Shp.prop.	Shp.vec.	Face.reco.
ADL	X									
CBIRD			X	X	X		X			
Diogenes	X									X
DrawSearch			X	X	X	X	X			
EVRW		X	X	X	X	X	X			
FOCUS					X	X	X			
GIFT	X		X		X	X				
FRIP		X	X	X		X		X	X	
ImageRover		X		X			X			
ImageScape			X	X		X			X	
LCPD					X	X				
LIRE			X		X	X				
MARS	X	X	X	X	X		X			
MIR	X				X	X				X
Picasso			X			X	X	X	X	
PicHunter					X					
IBM QBIC		X	X		X			X		
SMURF			X		X			X	X	
Octagon	X	X	X		X					
VisualSEEk						X		X	X	
WebSeer	X				X					X
WISE					X		X			

Table 3.3: Overview of the features extracted in CBIR systems

	Text	Image from database	User supplied image	Sketch	Browsing
ADL	X				X
CBIRD		X	X		
Diogenes	X				X
DrawSearch				X	
EVRW	N/A	N/A	N/A	N/A	N/A
FOCUS		X			
GIFT		X	X		
FRIP		X	X		
ImageRover	X	X			
ImageScape				X	
LCPD		X			X
LIRE	N/A	N/A	N/A	N/A	N/A
MARS	X	X	X		
MIR	X				
Picasso		X		X	
PicHunter		X			
IBM QBIC		X			
SMURF		X			
Octagon	X	X			
VisualSEEk				X	
WebSeer	X				
WISE		X	X	X	

Table 3.4: Overview of the different query methods in CBIR systems

Chapter 4

Construction of Algorithms

As mentioned in the introduction, the goal of this thesis is to apply CBIR techniques to DNA sequence comparison, and try to achieve a performance boost. The main idea here is to use an out-of-the-box CBIR system to index a set of images generated from DNA nucleotide sequences.

This involves two steps; creating an algorithm for visualizing the information stored in a nucleotide sequence as an image. Basically translating the one dimensional sequence into a two dimensional picture, where the values of the sequence represents a color value in the RGB color space. And then index these images in a general purpose CBIR system.

4.1 Generate images based on genome-sequences

The sequence translation proposes some challenges to solve and some considerations that have to be made.

4.1.1 Challenges

The image generating process proposes many challenges that have to be solved. The first one is the homogeneity of the information stored in a nucleotide sequence, the nucleotide sequence only has a alphabet of four characters. Item by item a nucleotide sequence can seem pretty uniform and repetitive. The second challenge is to take this uniform information and try to represent it in such a way that we create differences that are traceable(in a CBIR system). The third, and a very hard, challenge is the differences in size of the sequences, to be able to match sequences, you have to have a somewhat similar size. The final and hardest challenge is to encode the information in such a way that you can discover sub-sequence matches.

Encoding information

The encoding of the information is the core of the algorithm. The key thing is to figure out how we can translate Adenine, Cytosine, Thymine and Guanine to color values and represent this in an image.

First, we need an image that is substantially different from the other images generated, and the differences has to be represented in such a way that they can be measured in a CBIR system. As mentioned in Section 3.2.23 the most used features of CBIR systems are texture and color histograms, so in order to be able to succeed in retrieving images at least these two features has to be prominent in the images. The color histogram is the easiest thing to overcome, if you directly map the nucleotides of groups to RGB values and these directly represent the contents of the sequence, the histogram will be quite descriptive of the sequence.

Problems do arise with using the color histogram. As mentioned in 2.3.1, the histogram does not take into account the position of the pixels. One other problem is the error rate of sequencing combined with evolutionary distance, the histograms for two sequences that should be matched will be quite different, and therefore may not match in the CBIR system. But, with some fuzzy matching, it can be a good indicator for the similarity of two images.

The texture of the images when blindly translated from nucleotide values to RGB values will be very uniform, due to the small alphabet of values in the sequences. We have to find a way to translate the values of the sequence in order to generate a texture that is descriptive of the actual content of the sequence.

One solution is to post process the images with a threshold (binarization) filter, where the pixels with lower color intensity than a specific threshold value will be set to zero and all pixels with higher values will be set to the maximum allowed value in the specific color space. We can also use a contrast enhancing filter to simply increase the higher color values and decrease the color values. One also have to consider other features that can be used in the indexing of these images. Shape and color layout are two that can produce great results.

Creating differences

As mentioned above, the color histograms can be similar for two different images, so solely to depend on color histograms are not enough. The algorithm must be able to have regions which are significantly different. And that can be detected by studying a histogram. Such regions in nucleotide sequences are in such a low level, that they are very hard to efficiently detect using a CBIR system. And as the size of the sequences grow the small irregularities in the sequence become more and more distant when viewed as an image in two dimensions.

DNA spectrography[14] proposes some techniques that can be used. DNA spectrography tries to use spatial visualization techniques in order to detect protein coding regions. And they are able to create such differences that they actually succeed in discovering these regions. This knowledge is transferrable to our approach, and can be used to create an algorithm for visualizing DNA sequences.

Differing sizes

The size differences in a database of nucleotide sequences are staggering, out of our selected test database5.1 we saw that the average length of the sequences was about 9000 bp (basepairs), the smallest one is 239 bp and the longest sequence was 227420 bp. Figure 4.1 contains a scatterplot where the huge differences in sequence length are illustrated. These differences poses some issues. We have to try to create images that are able to be compared unaffected by the size of the sequence.

There are some possible solutions to this problem, but none of them are optimal. Since it is impossible to increase the size of a sequence in a way that does not affect the comparison. We can try to reduce the size of the large ones to normalize the sizes. As we see it, there are two different approaches, either you can remove base pairs *before* the sequence is translated to an image, or you can scale the image *after* the translation has happened. Either way you loose information, and you can loose information that is essential for two sequences to be considered similar(DNA wise).

One advantage of scaling the image after the translation, is that you can scale the image down, or you can scale the image up. And the results will be similar across the sequence database. Another approach is to use a run-length encoding scheme where you try to use lossy text-compression methods, and then process the output of that encoder.

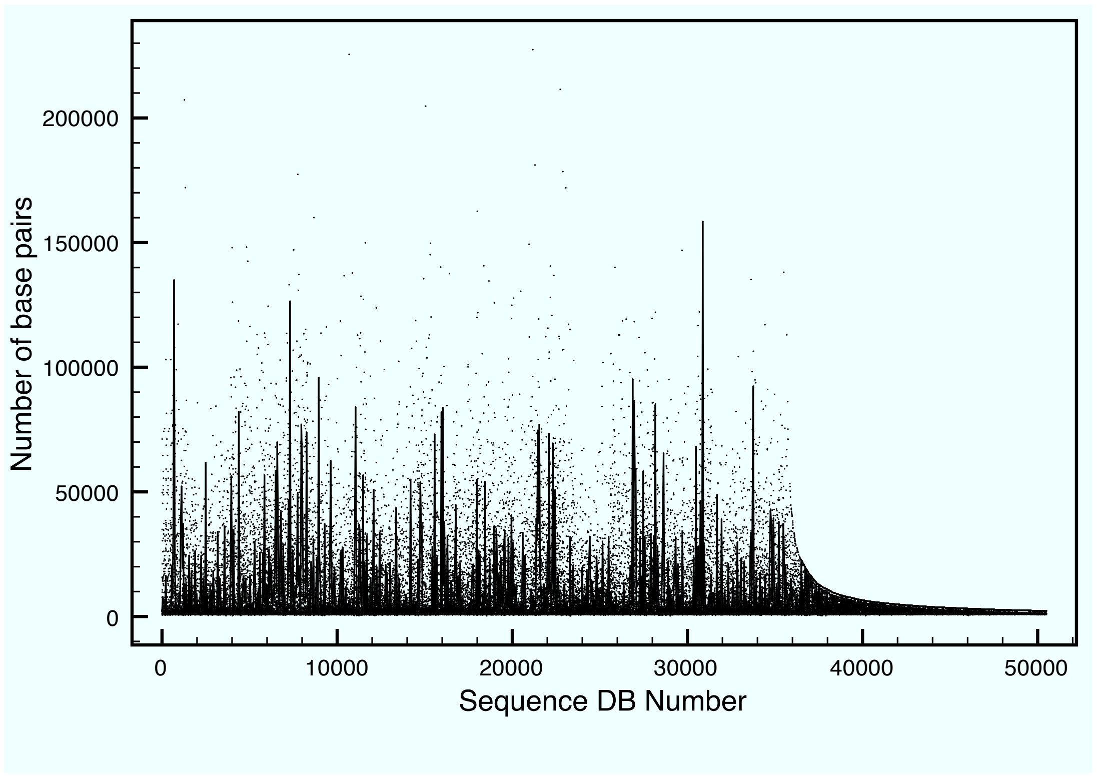


Figure 4.1: Plot of the sequence sizes in the dataset

Achieving sub-sequence matching

The hardest challenge to overcome is to enable the CBIR system with sub-sequence matching. Say that you have a 200 bp sequence you want to run through the system, and in a 10,000 bp long sequence that particular query-sequence exists. Well, looking at CBIR systems in general, one way to accomplish this is to use some kind of shape feature, and translate the image so that these shapes are uniform regardless of the size of the image. It can also be solved with layout color or texture information, but this will ruin performance figures.

4.1.2 Considerations

There are some considerations to be made, before designing the algorithms and deciding on what CBIR system we have to use.

Image dimensions

The image dimensions are hard to determine, the translation from a one dimensional list of values to a two dimensional matrix of pixels. Since you have no logical width parameter on the image you just have to set one. But the width of the image will affect the appearance of the image a great deal, so you have to do something that is uniform and that can work for sequences of all lengths.

As far as we can conceive there are two options. The first is trying to derive a static width parameter from the data set you are working on, or use some function of each sequence. One approach is to use the average length of the data, and then you can apply a function to that value to derive some value for the width of the image. The second options is to use the length of the sequence to be translated and apply a function to that. Either way you will get images that are very different in shape, and, most likely, you will have no control over the visual aspects of how the width of the sequence affects the visual presentation.

We decided to go ahead and use the square root of each sequence, this gives us the same aspect ratio for all images, and thus enables me to scale them uniformly.

What to emphasize

Should the system be able to facilitate weighting of one single nucleotide type? In that case we have to export four images for every sequence, and it will have a major impact on the speed of the system, both when indexing sequences but also in the retrieval process, regardless of which CBIR system we use. Another possibility is to have some vectors that are user adjustable when creating the images, so that the nucleotides can be individually adjusted for prominence in the resulting picture.

Nucleotide or protein sequences

In order to construct an algorithm we had to make a choice on whether to support nucleotide sequences or protein sequences. There are both technical and non-technical implications that point us to the selection of using nucleotide sequences.

On the technical side, we chose nucleotide sequences because they are easier to visualize, due to the limited set of values, we can simply map the different values to each of the four color components in the RGBA color space and use that as a basis for the image representation. Also it gives us a signal we can use with the frequency methods used in DNA spectrography.

On the non-technical side, we have the argument that nucleotide sequences are available at a earlier stage due to the time it takes to annotate and translate them, so if we can provide a broader and faster search with nucleotide sequences its a obvious choice.

However, there are some drawbacks. The limited set of values in nucleotide sequences compared to the 20 amino acids in protein sequences has some positive effects on the simplicity of the translation algorithm, but it gives us less diversity in the color spectrum. Another drawback of the uniformity of a nucleotide sequence, is when it is translated to a color value, the color values produced are inhibited by the low diversity of the nucleotide sequence, so a comparison of color histograms will be hard, if not impossible.

Choosing a CBIR system

One of the goals of this thesis is to use a general purpose CBIR system to test the possibilities of DNA sequence comparison in a multimedia context. There are multiple systems out there as discussed in Section 3.2. One requirement for us was that the system must support texture feature extraction and color information. Also the system should have fairly high performance (there is no need for a system that does not outperform existing technologies). The performance of a CBIR system is a function of how many features the system extracts, the complexity of the features and how many features are used when searching.

As stated in Section 3.2.23 most of the CBIR systems studied support a subset of the possible feature, and as mentioned above we need texture features and color features. The choice fell on the Lucene image retrieval system, it is well documented, easy to use and is written in Java. Also it has a low set of features that makes it speedy, and it supports texture and color feature extraction. Lucene also has parameterization on the different features, so you can fine tune the system with weights on what features to emphasize.

4.1.3 Methods

Since there are little or no existing work in this field of computer science, we did not have any research to base our work on. So we tried to look at research fields in spatial data representation and from DNA spectrography, to achieve a visualization of DNA nucleotide sequences. The process is somewhat based on trial and error to see what can be achieved.

Technologies used

We four main technologies for the implementation of this system.

LIRE[20] was used for the feature extraction and generation of the index for all the pictures, LIRE also includes a search class for searching with an image as example.

Java was chosen because LIRE is written in Java, and it is the easiest language to interface against LIRE with. Also there are many bioinformatics library written in Java witch are open source and readily available, the NCBI(National Center for Biotechnology Information) BLAST web service for example is a Java implementation.

MySQL was used as a data storage device for the nucleotide sequences, since the nucleotide databases are distributed as text flat files, they are slow to traverse and makes testing a hassle since you must iterate through a lot of text that is, in essence, unnecessary for us. So instead we parse the file once, and store it in a MySQL database. In this way we can easily extract the nucleotide sequences we need.

BLAST2 was used on a Linux platform, it is released by NCBI and is quite faster than the web service that NCBI provide.

4.2 The algorithms

I here propose two algorithms for generating images. The first is a novel approach where the nucleotides are directly translated to pixel values, the second algorithm is more advanced and uses a fourier transformation on the frequency of the nucleotides to calculate the frequency spectrum values of the nucleotides, and a formula used in DNA spectrography[14] for translating the spectral values to RGB color space.

4.2.1 The "RGBA" approach

This approach uses a grouping of 10 nucleotides to calculate each pixel. And for each for every occurrence of either adenine, cytosine, thymine or guanine a value, k is added to the red, green, blue and alpha channel of the pixel. k is calculated from dividing the maximum value possible in the color space (255) by 10, so that no pixel value will exceed the allowed value. Next when these pixels has been generated the width and height is calculated by $\sqrt{length(pixels)}$. The simplicity of the algorithm is its main advantage, it is very quick, and is implemented with very few lines of code, and it is also very memory efficient as you only have to have parts of the genome sequence in memory.

A huge drawback is that the size of the resulting image is a direct function of the length of the sequence, so it can be hard to do a good comparison between two images. Experiments with scaling of the input were done, an algorithm for a pseudo-lossy-compression algorithm was devised, but the resulting sequence was so unlike the original sequence that they were impossible to compare at an image-level. The pseudocode for the algorithm is written in Algorithm 4.2.1

Algorithm 1 ConvertDNAToImageRGBAApproach(DNASequence S)

```
1: Input: a dna sequence of length n
2: Output: an array of RGBA values
3: dnaseq = input;
4: n = length of dnasequence
5: col = array of length n/10
6: count = 0
7: w = 10;
8: k = Round(maximum_value_of_color_spacew)
9: for $i = 0$ to $n$; i = i + w do
10:    for all S = subsequence(i*w,w) do
11:       for all C as char in S do
12:          if C = "A" then
13:             color.R += k
14:          else if C = "C" then
15:             color.G += k
16:          else if C = "T" then
17:             color.B += k
18:          else if C = "G" then
19:             color.A += k
20:          end ifcount ++
21:       end for
22:       col[count] = color
23:    end for
24: end for
```

The algorithm takes a genome sequence as an input, and calculates a constant k, by dividing the maximum value allowed in each color channel in the selected color space with the constant w. w is the desired "window" size, it is how many nucleotides you want to represent with one pixel. Then the algorithm iterates over the entire sequence in blocks of size w. An iteration is made over each of these blocks, checking the nucleotides, if you encounter adenosine, the red component of that blocks respective pixel is increased with k, for cytosine the green component is increased, for thymine the blue component is increased, and for guanine the alpha component (representing the transparency) is increased. This results in a pixel with a valid color intensity for the color space, representing that block.

Example images of two gene sequences are shown in Figure 4.2. As we see, their respective sizes are quite different, and not easily comparable with any extracted feature without normalizing the feature descriptors.

First of all in Figure 4.2 b, there are no prominent features of that image. The pixel values appear chaotic, like confetti, and also there are no significant texture feature to extract. The actual implementation in Java can be seen A.6

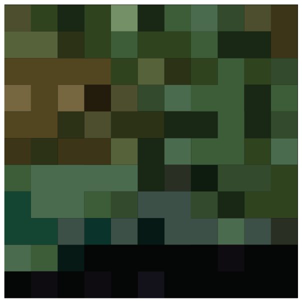
4.2.2 RGB algorithm with scaling of images

Due to some of the problems stated in Section 4.1.1, both with very different lengths of the nucleotide sequences, and the chaotic appearance of the images, a second algorithm was devised.

The basis of the algorithm is the same as the RGBA approach, but after converting the nucleotide sequences to RGBA values, the images are uniformly scaled, with a common percentage. For the testing we used a 300% scale and a 700% scale. The scaling was done to get larger representations of the color information extracted with the CBIR system, and hopefully better results.

4.2.3 The "frequency" approach

The frequency approach is by far the most complex and computational intensive, it is inspired by the algorithm described in [14], and takes its basis upon translating a nucleotide sequence into four discrete signals $S_1 \dots S_4$. This signal is obtained by first initializing four lists with the same length as the nucleotide sequence with 0, one for each nucleotide value. Then iterating through the nucleotide sequence $i = 0 \dots n$, on each iteration you check which nucleotide is on that position, and tag that nucleotides' list at that position with 1 (Algorithm 2). When the iteration is done, the results


(a) Image of short sequence

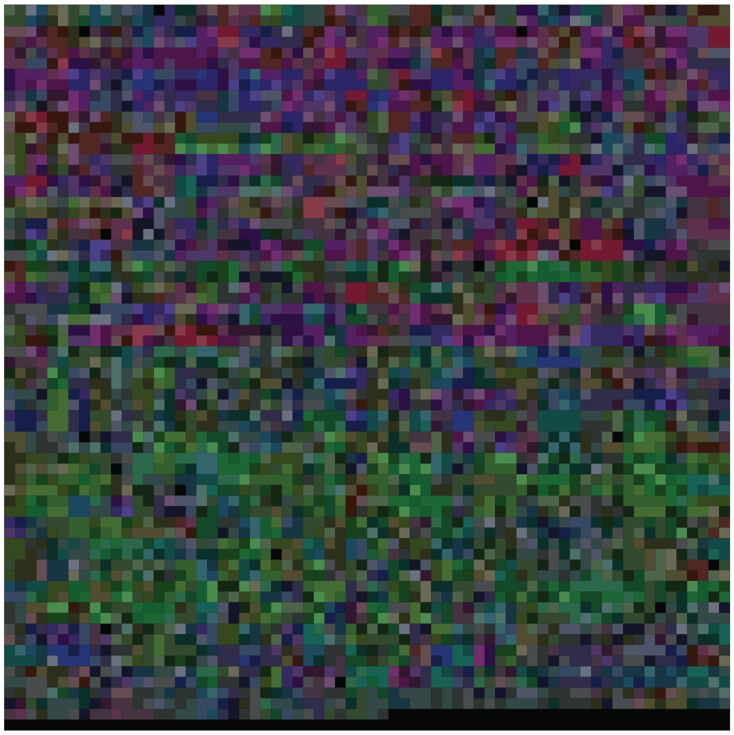

(b) Example of long sequence

Figure 4.2: Sample images from the RGBA approach. a) is 1000 bp long, b) is 25,000 bp long)

are four lists with values of 0 and 1, depicting the frequency of the different nucleotides in the sequence

We then define a value W which is the window size, this window we

Algorithm 2 generateBinaryCount(DNASequence S)

```
n = length(S)
bis = $[1 \ldots 4]$
bis[1] = $[1 \ldots n] = 0$
bis[2] = $[1 \ldots n] = 0$
bis[3] = $[1 \ldots n] = 0$
bis[4] = $[1 \ldots n] = 0$
for $i = 0$ to $n$ do
  $char = S[i]$
  if $char = "a"$ then
    bis[1] = 1
  else if $char = "c"$ then
    bis[2] = 1
  else if $char = "t"$ then
    bis[3] = 1
  else if $char = "g"$ then
    bis[4] = 1
  end if
end for
return bis
```

slide over the genome sequence as shown in 4.3.

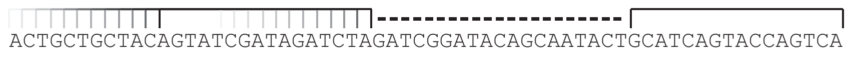

ACTGCTGCTACAGTATCGATAGATCTAGATCGGATACAGCAATACTGCATCAGTACCAGTCA

Figure 4.3: The window sliding over a sequence, $w = 16$

For each of these windows we calculate the discrete fourier transform (Algorithm 3), for each of the nucleotides.

Algorithm 3 1D-FourierTransform(Signal M)

```
1: for $u = 0$ to $M - 1$ do
2:    $F[u].real = 0$
3:    $F[u].imag = 0$
4:    for $x = 0$ to $M - 1$ do
5:       $F[u].real = F[u].real + (F[X] * COS(2 * \pi * u * x * x/M))$
6:       $F[u].imag = F[u].imag$ - $(F[X] * SIN(2 * \pi * u * x * x/M))$
7:    end for
8:    $F[u].real = F[u].real/M$
9:    $F[u].imag = F[u].imag/M$
10: end for
11: return F
```

We then iterate over the results from this window. and for each of the nucleotide frequencies we calculate the spectrum component with the formula;

$spectrum = \sqrt(F.imag^2) + (F.real^2))$.

We then use a formula from [14], which allows us to, from the spectrum values, calculate a RGB color space value, that formula is defined as;

$Color = ((adenine_weight_color * spec_adenine) + (thymine_weight_color * spec_thymine) + (cytosine_weight_color * spec_cytosine) + (guanine_weight_color + spec_guanine))$.

There are weights for each nucleotide in each color, these weights allows us to tweak the resulting image on emphasizing different colors or different nucleotides. We repeat this process for all the possible windows of size w on the whole sequence. The result is a set of pixels depicting the frequency of the different nucleotides.

Algorithm 4 ConvertDNAToImageFrequencyApproach(DNASequence S)

```
$W = 120$
$B = generateBinaryCount(S)$
$N = length(B[1])$
$pixels = [1 \dots floor(((N - W) + 1)/2)]$
//the color weights for adjusting the image
$awr, awg, awb, twr, twg, twb, cwr, cwg, cwb, gwr, gwg, gwb = 1$
$count = 0;$
for $i = 1$ to $N - W$ do
    $A = 1D - FourierTransform(subsequence(B[1], i, W))$
    $C = 1D - FourierTransform(subsequence(B[2], i, W))$
    $T = 1D - FourierTransform(subsequence(B[3], i, W))$
    $G = 1D - FourierTransform(subsequence(B[4], i, W))$
    for $k = 1$ to $length(A)$ do
        //calculate the spectrum values of the elements
        $spec_a = \sqrt{(}(A[k].imag^2) + (A[k].real^2))$
        $spec_c = \sqrt{(}(C[k].imag^2) + (C[k].real^2))$
        $spec_t = \sqrt{(}(T[k].imag^2) + (T[k].real^2))$
        $spec_g = \sqrt{(}(G[k].imag^2) + (G[k].real^2))$
        //calculate the RGB values
        $pixels[count].R = ((awr * spec_a) + (twr * spec_t) + (cwr * spec_c) + (gwr + spec_g))$
        $pixels[count].G = ((awg * spec_a) + (twg * spec_t) + (cwg * spec_c) + (gwg + spec_g))$
        $pixels[count].B = ((awb * spec_a) + (twb * spec_t) + (cwb * spec_c) + (gwb + spec_g))$
        $count++$
    end for
end for
return pixels
```

Figure 4.4 shows a result sequence processed with the algorithms in the frequency approach, there are prominent features that are more suited for extraction.

The nature of the algorithm, with its sliding window (Figure 4.3), exploits the shifting of this window to make significant areas of a nucleotide sequence, that in themselves can be fairly short, appear more prominent.

In the picture there are bright lines, across the image, these are regions of high frequency of all the nucleotides so that the color values of the pixels in these areas are higher.

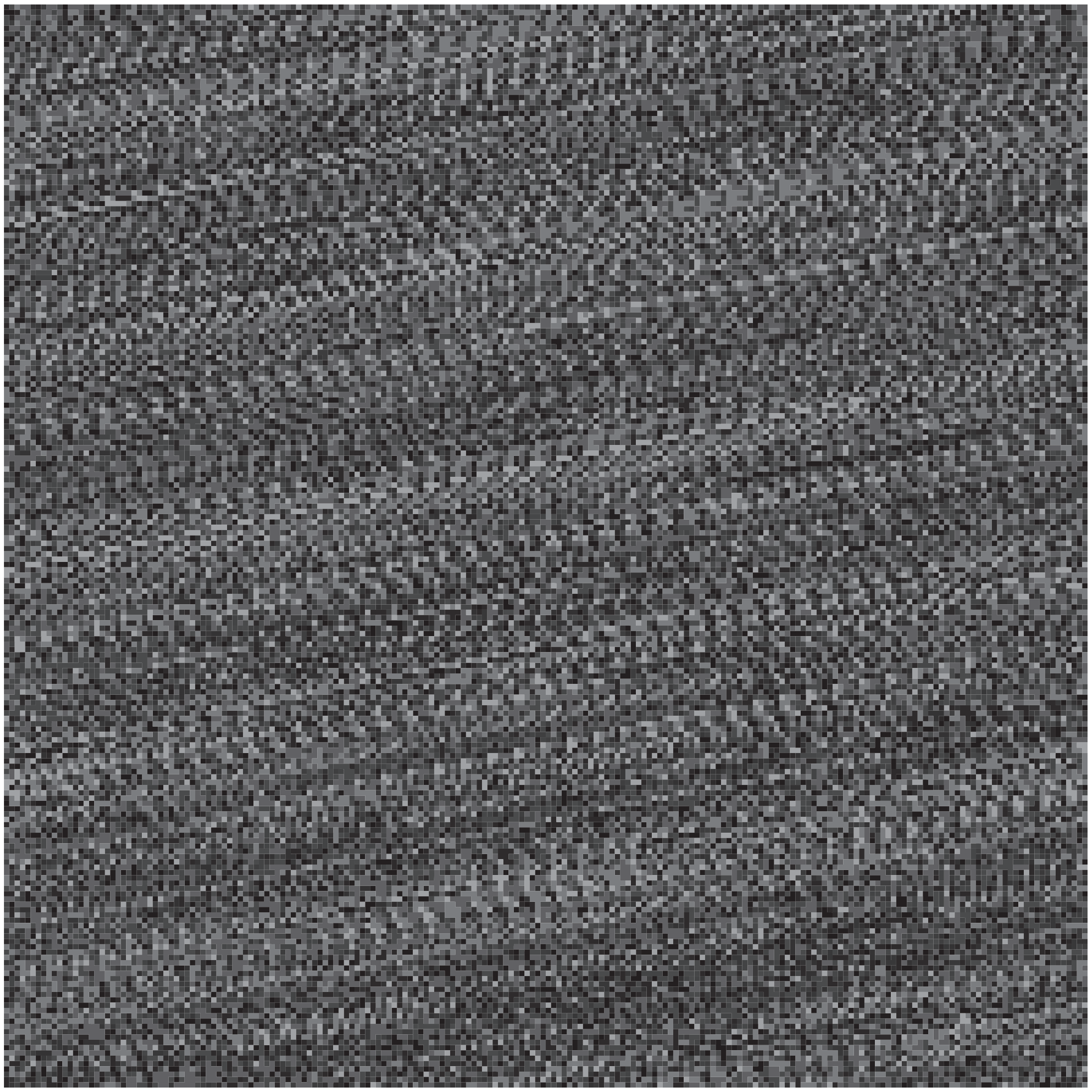


Figure 4.4: Example image of a nucleotide sequence translated with the frequency approach

One problem with this approach is the angle of the prominent lines and our approach to calculate the size of the image. The size of the image is directly derived from the length of the sequence. This means that if a user supplies a sequence that is equal to an indexed sequence, only a bit shorter, the angle of these lines, and their position in the image will also change. From a CBIR systems point of view, this differentiates the images substantially, especially if line-likeness or texture direction is used as a feature. The true implementation in Java can be seen in, code listing A.5

4.3 Indexing the generated images through a CBIR system

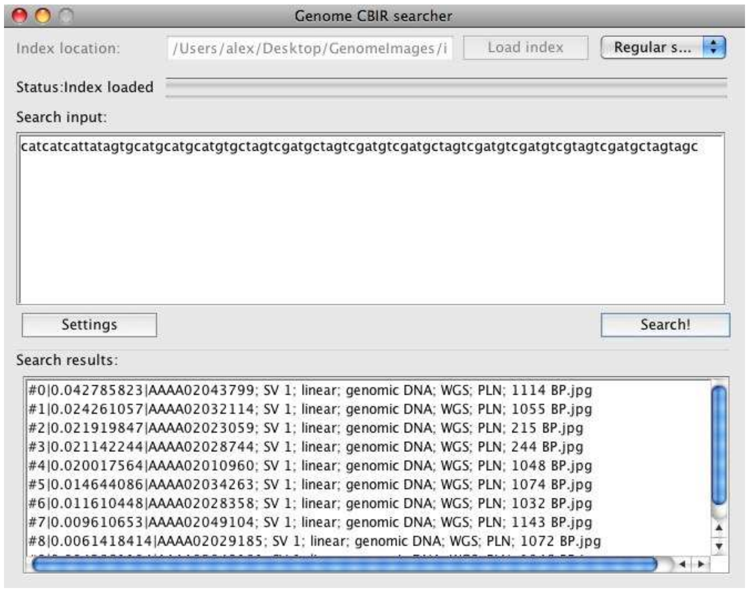
The chosen CBIR system for the task of indexing was LIRE. To begin indexing we constructed two modules, one module reads a flat-file of nucleotide sequences and parses this file and wrote them to a SQL database. The module also has a system for reading the database and processing the sequence input to images and writing these images to disk. The second module indexes the images with LIRE, and it has a Java Swing user interface for querying the database and visualizing the query sequences with the results from LIRE.

4.3.1 The indexing process

The process of indexing the nucleotide sequences consists of two steps, the first is generating the images, and the second is indexing these images with LIRE.

Generating the images

A simple test bench was set up to quickly test new algorithms, where we constructed a common interface, which if implemented, that allows for more generic testing of the algorithms. If any class implements this interface it can act as a nucleotide to image generator. We then created a class that reads the database and processes all the sequences to images and saves these images to disc. We used the filenames to store information about the accession number and the number of base pairs, so its easy to determine a hit if you have the accession number for a query sequence. We did this process for the four algorithms described above, and saved the images to different folders.



Index location: /Users/alex/Desktop/GenomeImages/i

Status:Index loaded

Search input:

```
catcatcattatagtgcatgcatgcatgtgctagtcgatgctagtcgatgtcgatgctagtcgatgtcgatgtcgtagtcgatgctagtagc
```

Search results:

```
#0|0.042785823|AAAA02043799; SV 1; linear; genomic DNA; WGS; PLN; 1114 BP.jpg
#1|0.024261057|AAAA02032114; SV 1; linear; genomic DNA; WGS; PLN; 1055 BP.jpg
#2|0.021919847|AAAA02023059; SV 1; linear; genomic DNA; WGS; PLN; 215 BP.jpg
#3|0.021142244|AAAA02028744; SV 1; linear; genomic DNA; WGS; PLN; 244 BP.jpg
#4|0.020017564|AAAA02010960; SV 1; linear; genomic DNA; WGS; PLN; 1048 BP.jpg
#5|0.014644086|AAAA02034263; SV 1; linear; genomic DNA; WGS; PLN; 1074 BP.jpg
#6|0.011610448|AAAA02028358; SV 1; linear; genomic DNA; WGS; PLN; 1032 BP.jpg
#7|0.009610653|AAAA02049104; SV 1; linear; genomic DNA; WGS; PLN; 1143 BP.jpg
#8|0.0061418414|AAAA02029185; SV 1; linear; genomic DNA; WGS; PLN; 1072 BP.jpg
```

Figure 4.5: The user interface of our application

Indexing with LIRE

For indexing the images with LIRE, we created a simple class. Its input is a folder(the input), and a file path which is the location of the index. And we used the default *SimpleAnalyzer* included in the distribution of LIRE to analyze the images. For each of the algorithms we created a separate index. The code for this can be seen in code listing A.2

4.3.2 Searching

Searching was implemented with two modules, one module that included the user interface shown in Figure 4.5 and one for batch processing the algorithms for testing. The batch processing module included a method for manipulating the query sequences to create evolutionary distance between the query sequence and the sequence we want to find. The implementation can be seen in code listing A.3

The application

In Figure 4.5 the user interface for our implementation of the search with LIRE is shown. It is quite simply a window in which the user can load an arbitrary index from disk (we have no user interface for generation of an index, it must be done with Java code). When this image is loaded the user can specify a query with a nucleotide sequence. When the search button is pressed, the nucleotide sequence in the input box is processed and generated into an image, and sent to the LIRE search engine. The results are then parsed and returned to the user in form of a list-box. There is also a settings menu, where the user can tweak the weighting of witch features to emphasize.

Chapter 5

Evaluation and Results

The evaluation of our proposed algorithms are crucial to determine if this is in fact a possible combination. Our testing consist of parallel testing of each algorithm, with the same data set and the same queries and comparing the results with BLAST.

5.1 The test data set

When we selected our data set, we looked at its size in number of sequences and the lengths of the sequences. This was to get an average looking database. We selected a databank from EMBL, the WGS (whole genome shotgun, a sequencing technique) release of Oryza sativa (indica cultivar-group), the genome of a rice specie.

5.1.1 The genome data

The actual genome data is a good test subject. Because we have high variance in sequence length, as shown in Figure 4.1, the diversity of the lengths are clearly illustrated. And in Section 4.1.1, we state that the shortest sequence was 239 base pairs and the longest was 227420, this will put our algorithms to the test to see if the cope with the diversity in the genome world.

This database was then converted with the algorithms presented in Section 4.2 for the RGBA scaling approach we post processed the images with 300% scaling and 700% scaling.

5.1.2 The queries

The queries was selected randomly by using the random() function of MySQL in the query;

```
SELECT accesion_number,sequence_data
FROM genomeEntity
ORDER BY RAND() LIMIT 50;
```

. The genome sequences returned was written to a file, which we later loaded in our batch query processor. We used the same 50 queries for all the approaches.

5.1.3 Evolutionary distance and sequence length

To really test the approaches we devised a modification function. This functions transforms the input before querying the system. The function takes a nucleotide sequence S and an integer n as parameters and then proceeds to change n nucleotides in S to a different nucleotide. This function simulates evolution, and with the easy parameterization we can tweak the values to find the threshold evolutionary distance that our system can endure.

5.1.4 Reference system

In order to test the proposed system, we constructed a batch query processor which uses the National Center for Biotechnology Information's web service for querying their database with BLAST. All the queries was processed through this, it took roughly a day to process all the 50 queries. Since BLAST is the de facto standard, these results gives us a baseline in which we can compare the results from our approach with.

5.2 Evaluating a CBIR system in a genome context

As mentioned in section2.2 there are two measures used in information retrieval, precision and recall. But, in order to calculate these values, we need a measure of the total number of relevant documents. This proves to be harder than first imagined. Finding all relevant genome sequences to a query is near impossible, what we did was to query BLAST with few parameters to get as many hits as possible returned, and used these hits as our relevant document collection for all the queries.

We had a problem with the number of returned results from LIRE, at a certain level in the result list, the score was so low that the returned images basically does not have anything in common. So we determined a common cutoff value to produce a more realistic computation of precision and recall, the cutoff value was determined to be 15, this both hurts and inflicts on the credibility of the values.

5.3 The results

Here we present the different results, starting with the number of hits on the different algorithms. The number of hits is compiled from first querying the NCBI BLAST server, and then querying the respective LIRE indices with the same queries.

5.3.1 A note about precision and recall

In Section 2.2 we describe precision and recall as the most used measure when evaluating information retrieval systems, but since neither BLAST nor FASTA does guarantee the best alignment we cannot include the values for precision and recall. They do not give any meaning since we cannot be sure that the returned hits from BLAST (our reference set) are relevant or not. So these result are not presented as the result in the thesis itself but as a curiosity it will be included as a part of the appendix.

5.3.2 Number of hits per mutation

Our result set is measured by querying the system with 50 queries, and for all queries we had 10 levels of mutation, and in each an increasingly amount of the nucleotides was changed. And a measure is calculated on the performance of the approaches on how many correct hits they have compared to BLAST.

RGBA approach

With the standard RGBA approach we did not have any high hopes, due to the small size of the images and the problems with distinguishing low-level features. In Figure 5.1 the results of the algorithm is shown, and the simple algorithm performed fairly well. And it has good tolerance to evolutionary distance. But it only had an average of 12 hits per 50 queries, this is a hit percentage of 24, which is under a quarter of the expected hits.

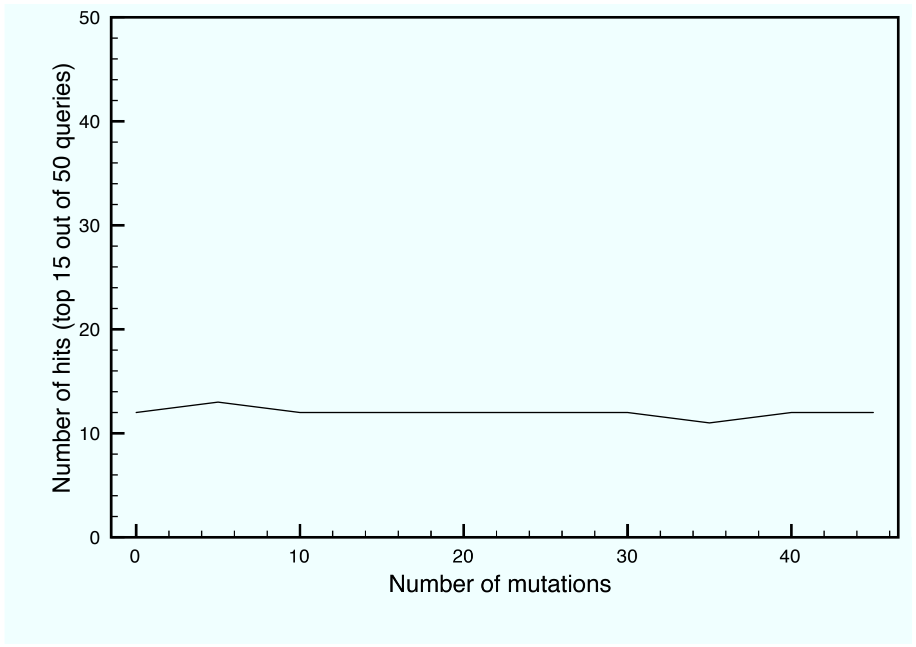


Figure 5.1: Number of hits on different levels of evolutionary distance

RGBA scaled 300%

In the first scaled index, we see a decrease in number of hits(fig: 5.2), compared to the regular RGBA approach. With only an average of 5,4 hits of 50 queries. This equals a 10.8 hit percentage. In other words, very low performance, but it still manages to find some of the sequences. We thought that scaling the images would help on the issue with low-level feature extraction of small images, but we may have mistaken.

RGBA scaled 700%

A surprising result from the second approach, with 700% scaling we see a definite rise in number of hits, the results from this approach led us to create a new index to test if we scaled the images even more, if we get even better results. As illustrated in Figure 5.3 the average hits per 50 queries is 15.8, which is a 29.2 percent increase from the 300% scaling approach, and a 31.6 hit percentage compared to the reference system.

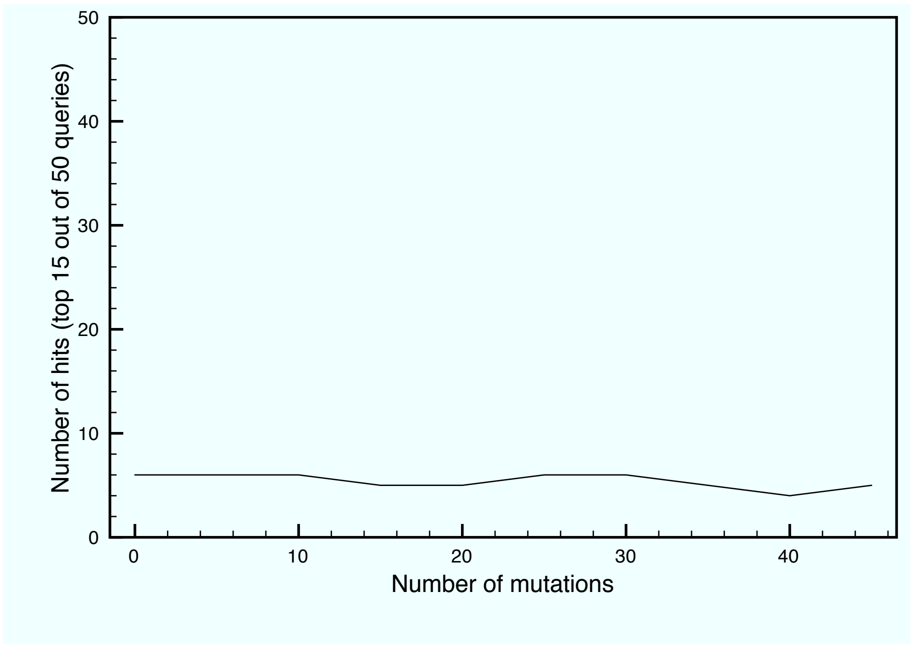


Figure 5.2: Number of hits on different levels of evolutionary distance

RGBA scaled 1000%

As shown in Section 5.3.2, scaling of the images 700% gave a significant rise in number of hits, and stable results across the mutations. But scaling of the images even more did not increase the number of hits, rather it decreased. Illustrated in Figure 5.4, we have an average of 10.7 hits per 50 queries, and a hit percentage of 21.4 across all queries.

Frequency approach

The frequency approach is the most elaborate and produces the most significant images compared to the other approaches. And as shown in Figure 5.5 it outperforms all the other approaches at least at the low range of evolutionary distance. With the initial average of 23 and 18 hits per 50 queries, its far better than the others. But we see that the hits quickly decreases with the increase of evolutionary distance, and it ends up with an average of 10.6 and an average percentage of 21.5. But it still is promising with the

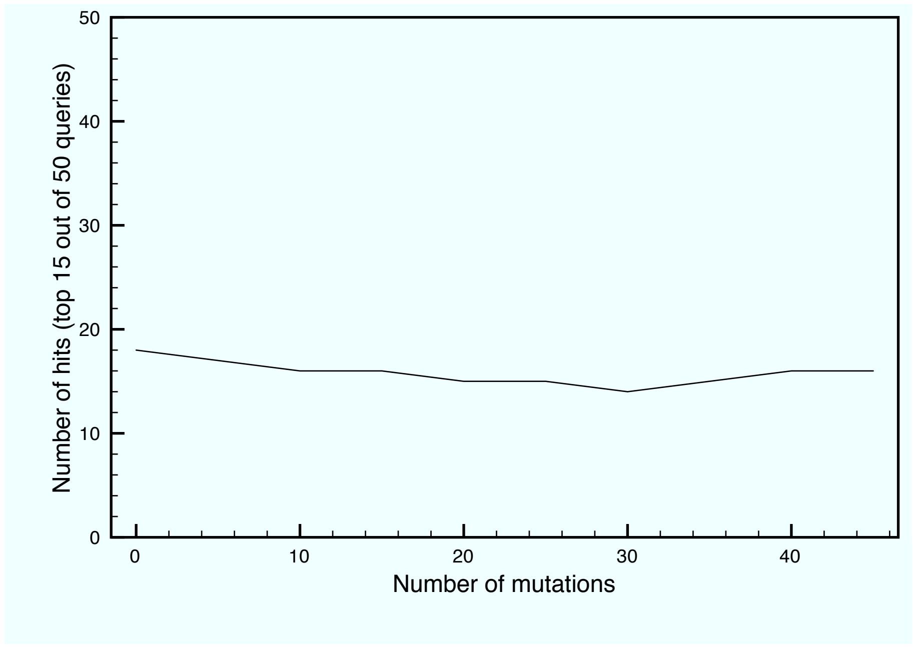


Figure 5.3: Number of hits on different levels of evolutionary distance

high hit-counts in the lower evolutionary distances.

Average number of hits compared

In Figure 5.6 a comparison between the proposed approaches is shown, and we see on the low scale of mutations that the performance of the frequency approach is far better than the others. But after 10 mutations the number of hits quickly drop.

This illustrates the issues with the algorithm, that central defining characteristics of the image changes very quickly with only a small amount of the nucleotides changes. This can to some extent be avoided by selecting lower values of the window-width parameter, but it illustrates the weaknesses of the algorithm very good.

As for the RGBA approach we see a good stability across the levels of mutation. All though the levels are low, they are stable. With the scaled RGBA approaches we see an increase in number of hits between a 300% scaling and the 700% scaling, and the 700% scaling outperforms the regular

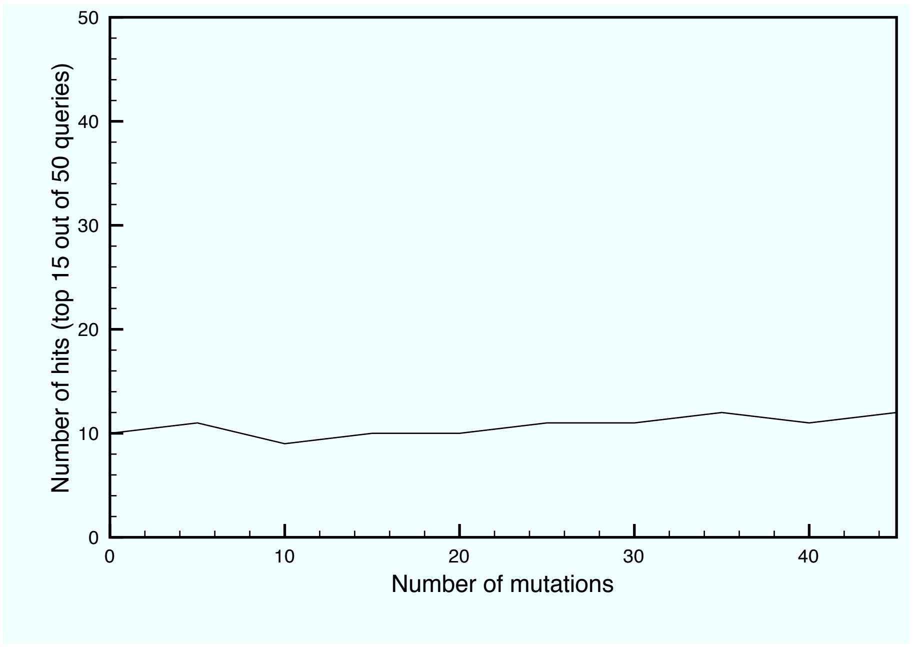


Figure 5.4: Number of hits on different levels of evolutionary distance

RGBA approach. But when we reach a 1000% scaling the numbers of hits drops across the board, and we can conclude that scaling of the images are not linear with the increase of hits.

5.3.3 Number of hits across sequence length

As stated in section 5.3.2, there are some varying results on the different approaches. But we do not say much about the distribution of the hits in relation to the actual query sequence. Figure 5.7 shows the number of hits compared with the length of the sequences.

It seems like the longer sequences in the data set, clearly gets better results, and especially with the fact that there are only 7 sequences with a length of over 20,000 bp. This means that the approaches generally favors long sequences. We have not specified graphs for all the separate approaches, but the trend was the same for all the approaches.

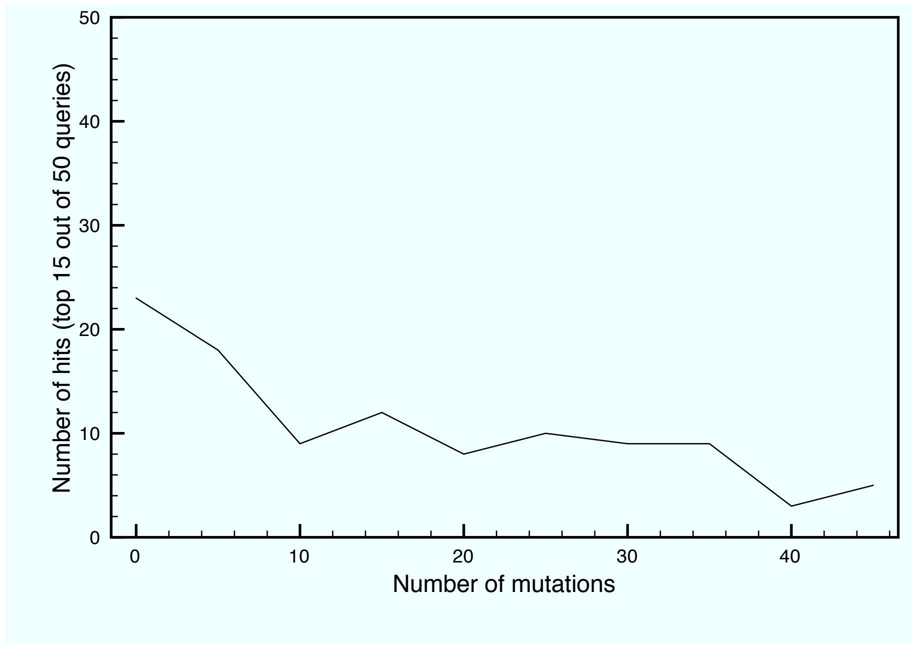


Figure 5.5: Number of hits on different levels of evolutionary distance

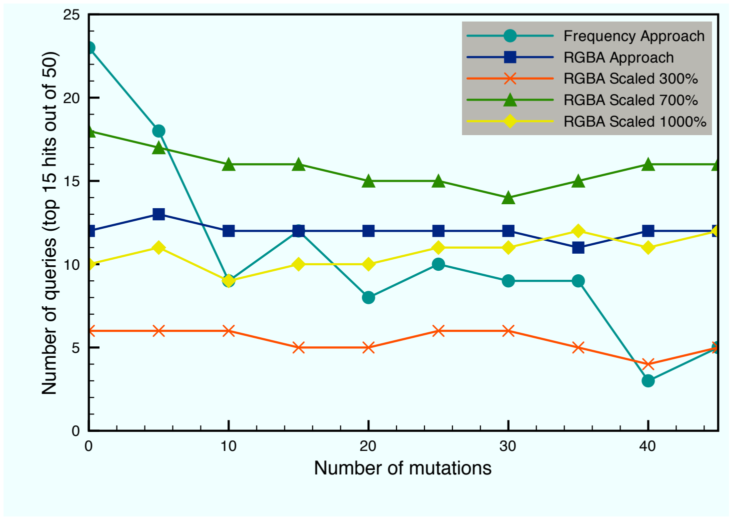


Figure 5.6: Number of hits on different levels of evolutionary distance

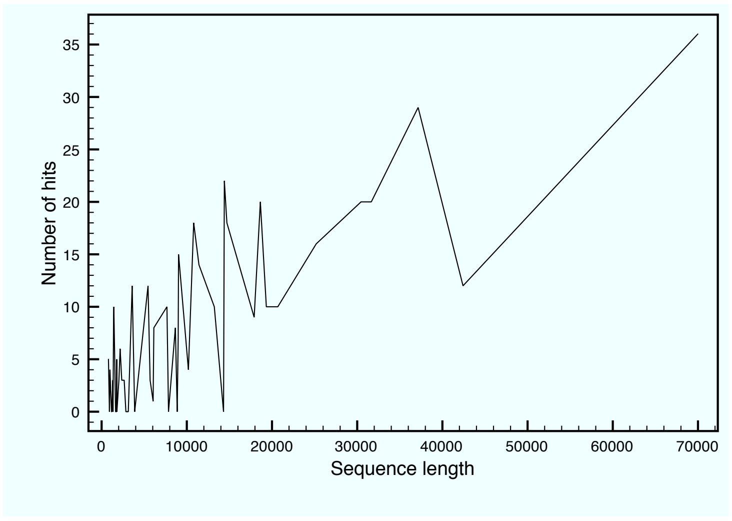


Figure 5.7: Number of hits compared to sequence length

5.3.4 Homogeneity vs Hits

We also made a measure on how the homogeneity of the sequence affects the number of correct hits returned, it is shown in Figure 5.8. The homogeneity measure was created by counting the number of the different nucleotides. And given that we define a heterogenous sequence as a sequence that contains 25% adenine, 25% cytosine 25% thymine and 25% guanine we add all the percentages deviating from this. If a sequence contains 20% adenine, 30% cytosine, 10% thymine and 50% guanine the homogeneity measure will be 50, we subtract 25 from all percentages and take the absolute value from the subtraction and then add them together. This gives us a simple measure on how the diversity of the query sequence is.

The measure does not take into account where in the sequence the nucleotides appear.

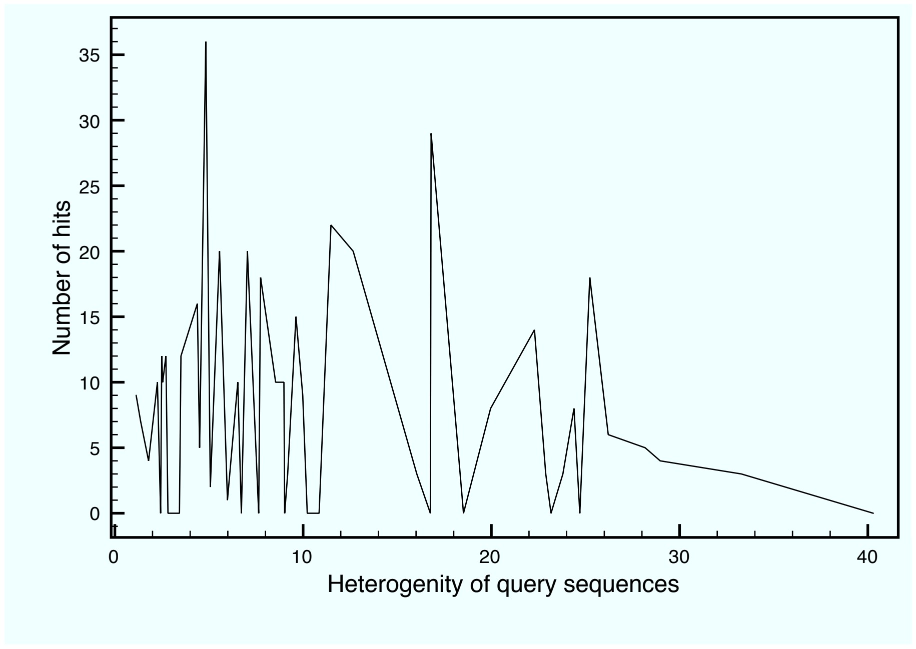


Figure 5.8: Number of hits on different levels of heterogenity

Chapter 6

Conclusion

In this chapter we present our conclusions, based upon the findings in the evaluation chapter.

6.1 Research questions

In the beginning of this thesis we asked two questions, the first was "*is it possible to translate a DNA sequence to an image?*". The answer for this is clearly, yes. As presented in chapter 4 there was two main algorithms presented that could translate a DNA sequence to an image.

The second question was, "*is it possible to use this image in a content based information retrieval system to find DNA sequences?*", this answer to this question is a bit faceted. We did succeed to create images and index these images in a CBIR system, and we were able to search this system with a genome sequence and get results.

The significance of these results are the central point in answering the question. As mentioned we presented two different algorithms for translating the nucleotide sequences to images. The most novice approach simply translated blocks of n-nucleotides to a pixel value and added that to an image. This approach performed far better than first imagined, and it performed even better with some post-processing of the images (especially with 700% scaling).

The second approach was much more complex, and was a bit slower in the translation process than the first, it outperformed the novice approach on low evolutionary distances between the subject sequence and the query sequence, but the number of hits quickly dropped.

The both approaches show great promise, especially the frequency approach, but it has some quirks that have to be worked out before implementing a system to "compete" with the currently used nucleotide searching systems. The frequency approach is too sensitive to changes in the nucleotide sequences(mutation), and also it has an expected sensitivity of genome sequence length. The novel RGBA approach is not as sensitive to mutation as the frequency approach, but did not preform nearly as good as the frequency approach.

The results from all approaches show that, yes, it is possible to retrieve genome sequences, translated to an image, with a general purpose CBIR system.

6.2 Implications

As we have stated, it is in fact possible to query a CBIR system with images generated from DNA sequences, so, what implications does this make?

First of all it shows that there is a field of DNA searching that shows some potential, by default a CBIR system should outperform the popular DNA search methods in terms of speed.

Secondly, the fuzziness of the search in a CBIR system is far greater than the ones in BLAST/FASTA by default, and to achieve this fuzziness in those systems you have to trade it off with performance.

6.3 Drawbacks

Although we can achieve broader searches with higher performance with a CBIR approach, there are some drawbacks. One of the clear drawbacks is the lack of distinct distance measures between two sequences in our approach, and this is quite central in determining if there are some relationship between two sequences.

6.4 Limitations

In the approaches outlined in this thesis there are some clear limitations and issues which are not addressed. The first issue, is the way we calculate the image dimensions, $\sqrt{sequencelength}$, especially in the frequency approach, this will change the image quite much when you adjust the length of the sequence. In both approaches it will change in the texture feature dimension, but in the frequency approach it will also change in the color feature dimension.

Another issue is the favoritism of longer DNA sequences versus short ones, in the results presented in Chapter 5 we showed a graph displaying this. And this is a huge limitation of the approaches, but with some further work it can most probably be solved.

6.5 Contribution

As shown, genome search in a CBIR system is a viable concept, and it challenges the current techniques, and the current usages of genome searching systems, this in it self is a contribution. If one could create a system that performs as well (in terms of finding the genes) as the current methods, but only faster and with better broadness of the search one could in fact make such systems more available to run on non-specialized hardware with low performance and it still is quite fast. One huge advantage is also the index sizes, in a BLAST/FASTA index you have to store the sequences in their entirety, this results in huge indexes and thus slower speeds. In our test data the BLAST index was a total of 521 Megabytes the corresponding index in the CBIR system was 25 Megabytes, this gives greater speeds but also greater portability of the systems.

6.6 Future work

Since this thesis only deals with the possibility of using CBIR in a DNA search context there are some ”wrinkles” to iron out, and for further development of this approach we have some thoughts.

6.6.1 Sequence length normalization

As mentioned in Section 4.1.1, the homology in size between the sequences has been a problem. This is a issue that should be addressed, and as we see it there are two possible solutions for solving this problem, input normalization or output normalization.

Input normalization is the process of reducing the size of the large input sequences to the system, by means of a lossy compression algorithm. This will give less differing sizes of the resulting images.

Output normalization will try to do the same thing, but instead of reducing the size of the actual DNA sequences we scale the larger images down to a more normalized size.

6.6.2 Post processing of images

Post processing can make great improvements on the uniqueness of the images created, especially with our frequency approach. Typical post processing algorithms will include image filters like, thresholding, bleeding, blurring and noise reduction.

Thresholding will reduce areas of low color values. This would probably fit nicely with the frequency approach, and especially compared with bleeding.

Bleeding will make the prominent regions of a certain color become bigger. Blurring will also reduce some noise aswell as the typical noise reduction algorithms like sobel masks.

6.6.3 Sub-picture matching

The ability to match subsequences in images is also something that has to be explored before publishing a finished system, this means that you have to figure out a new technique for calculating the image dimensions, and maybe a different process when submitting a query DNA sequence.

6.6.4 Custom CBIR systems/Multiple CBIR systems

One should also look into CBIR systems with other feature descriptors and other techniques for feature extraction in order to get the best results, or one could in fact construct a custom CBIR system, based upon findings in this report, that can tackle the issues in a better way than a general purpose CBIR system such as LIRE.

6.6.5 Other translation approaches

The most important work is to take the experiences made in this report and use these to construct new algorithms for the translation of the DNA sequences, and maybe look into also translating protein sequences.

6.6.6 Measure performance

A rigorous performance measuring between CBIR systems in a DNA context and the popular BLAST/FASTA methods must be made to be able to conclude if a DNA CBIR system is viable. But this can only be done if the CBIR system is able to compete with the state of the art systems with regards to precision.

Bibliography

[1] retrievr - search by sketch. `http://labs.systemone.at/retrievr/`, 2009.

[2] Mausumi Acharyya and Malay K. Kundu. Efficient rotation invariant feature extraction for texture segmentation - via multiscale wavelet frames.

[3] Acm.org. Communications of the acm, vol 34. 1991.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, October 1990.

[5] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schffer, Ro A. Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res*, 25:3389–3402, 1997.

[6] Yuksel Alp Aslandogan and Clement T. Yu. Diogenes: a web search agent for person images. pages 481–482, 2000.

[7] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. 1999.

[8] Andy Brass. Searching dna databases for similarities to dna. `http://bioinformatics.oxfordjournals.org/cgi/reprint/14/4/349`, 1998.

[9] Michael K. Buckland. What is a document? *Journal of the American Society for Information Science*, 48(9):804–809, 1997.

[10] Visual Querying By, A. Del Bimbo, M. Mugnaini, P. Pala, and F. Turco. Visual querying by color perceptive regions. pages 1241–1253, 1998.

[11] Xia Cao, Shuai Cheng Li, and Anthony K. H. Tung. Indexing dna sequences using q-grams. pages 4–16, 2005.

[12] C. H. Chen, L. F. Pau, and P. S. P. Wang, editors. *Handbook of pattern recognition & computer vision*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1993.

[13] Ingemar J. Cox, Matt L. Miller, Thomas P. Minka, Thomas V. Papathomas, and Peter N. Yianilos. The bayesian image retrieval system, pichunter: Theory, implementation and psychophysical experiments. *IEEE transactions on image processing*, 9:20–37, 2000.

[14] Nevenka Dimitrova, Yee Him Cheung, and Michael Zhang. Analysis and visualization of dna spectrograms: open possibilities for the genome research. pages 1017–1024, 2006.

[15] Abdelmoula Elharti and Donna M. Kocak. Comparing dna sequences using wavelets. *Mathematics and Applications of Data/Image Coding, Compression, and Encryption III*, 4122(1):138–148, 2000.

[16] Charles Frankel, Michael J Swain, and Vassilis Athitsos. Webseer: An image search engine for the world wide web. 1996.

[17] genome.gov. Dna fact sheet. `http://www.genome.gov/25520880`, 2009.

[18] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1992.

[19] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. *SIGMOD Rec.*, 14(2):47–57, 1984.

[20] Erik Hatcher and Otis Gospodnetic. *Lucene in Action (In Action series)*. Manning Publications, December 2004.

[21] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A*, 89(22):10915–10919, November 1992.

[22] Ela Hunt, Malcolm P. Atkinson, and Robert W. Irving. A database index to large biological sequences. pages 139–148, 2001.

[23] Ela Hunt, Malcolm P. Atkinson, and Robert W. Irving. Database indexing for large dna and protein sequence collections. *VLDB J.*, 11(3):256–271, 2002.

[24] Net Library inc. *mage databases : search and retrieval of digital imagery / edited by Vittorio Castelli, Lawrence D. Bergman.* 2003.

[25] Ramesh Jain, Michael S. Lew, Michael S. Lew, Michael S. Lew, Michael S. Lew, Kim Lempinen, Nies Huijsmans, and Nies Huijsmans. Webcrawling using sketches. 1997.

[26] B. Ko and H. Byun. Frip: a region-based image retrieval tool using automatic image segmentation and stepwise boolean and matching. *Multimedia, IEEE Transactions on*, 7(1):105–113, Feb. 2005.

[27] Ian Korf, Mark Yandell, and Joseph Bedell. *BLAST*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.

[28] Oak Ridge National Library. Chromosome faqs. `http://www.ornl.gov/sci/techresources/Human_Genome/posters/chromosome/faqs.shtml`, 2009.

[29] T. Sanocki M. Heath, S. Sarkar and K.W. Bowyer. A robust visual method for assessing the relative performance of edge-detection algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 19, No. 12*, pages 1338–1359, 1997.

[30] Bruce A. Draper Madirakshi Das, Edward M. Riseman. Focus: Searching for multi-colored objects in a diverse image database. 2001.

[31] B.S. Manjutah. Image browsing in the alexandria digital library project.

[32] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval.* Cambridge University Press, July 2008.

[33] Gonzalo Navarro and Ricardo Baeza-yates. A practical q-gram index for text retrieval allowing errors. *CLEI Electronic Journal*, 1:http://www.clei.cl., 1998.

[34] Illinois Goverment News Network. Provide law enforcement critical forensic evidence results even more quickly and efficiently. `http://www.illinois.gov/PressReleases/ShowPressRelease.cfm?RecNum=4661&SubjectID=49`.

[35] Ze nian Li, Osmar R. Za, and Bing Yan. C-bird: Content-based image retrieval from digital libraries using illumination invariance and recognition kernel. pages 361–366, 1997.

[36] Michael Lew Nies and Michael S. Lew. Content based image retrieval: Klt, projections, or templates. pages 27–34, 1996.

[37] ornl.gov. Frequently asked questions about the hgp. `http://www.ornl.gov/sci/techresources/Human_Genome/faq/faqs1.shtml`, 2009.

[38] Michael Ortega, Yong Rui, Kaushik Chakrabarti, Sharad Mehrotra, and Thomas S. Huang. Supporting similarity queries in mars. pages 403–413, 1997.

[39] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A*, 85(8):2444–2448, April 1988.

[40] William R. Pearson. Rapid and sensitive protein similarity searches. `http://bioinformatics.oxfordjournals.org/cgi/reprint/14/4/349`, 1985.

[41] Rice Genome Research Program. Introduction to the rice genome research project. `http://rgp.dna.affrc.go.jp/E/rgp/rgpintro.html`, 2009.

[42] International Rice Genome Sequencing Project. Status of irgsp. `http://rgp.dna.affrc.go.jp/cgi-bin/statusdb/irgsp-status.cgi`, 2009.

[43] The International Rice Genome Sequencing Project. The international rice genome sequencing project home. `http://rgp.dna.affrc.go.jp/IRGSP/index.html`, 2000.

[44] P. Raven, G. Johnson, S. Singer, and J. Losos. *Biology.* McGraw-Hill, 2004.

[45] Steven M. Rubin. *Computer Aids for VLSI Design, second edition.* 1994.

[46] E. Sciascio, M. Mongiello, and M. Mongiello. Content-based image retrieval over the web using query by sketch and relevance feedback. pages 123–130, 1999.

[47] Stan Sclaroff, Leonid Taycher, and Marco La Cascia. Imagerover: A content-based image browser for the world wide web. pages 2–9, 1997.

[48] John R. Smith and Shih fu Chang. Querying by color regions using the visualseek content-based visual query system. pages 23–41, 1996.

[49] Rohini K. Srihari. Automatic indexing and content-based retrieval of captioned images. *Computer*, 28(9):49–56, 1995.

[50] M. Sternberg. *Protein Structure Prediction: A Practical Approach.* Oxford University Press, Inc., New York, NY, USA, 1997.

[51] H Tamura, S Mori, and T Yamawaki. Texture features corresponding to visual perception. *IEEE Trans. on Sys, Man, and Cyb*, (8), 1978.

[52] P. Vacha and M. Haindl. Illumination invariants based on markov random fields. *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, Dec. 2008.

[53] Remco C. Veltkamp and Mirela Tanase. Content-based image retrieval systems: A survey. 2000.

[54] James Ze Wang, Gio Wiederhold, Oscar Firschein, and Sha Xin Wei. Wavelet-based image indexing techniques with partial sketch retrieval capability. pages 13–24, 1997.

Appendix A

Code Listings

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.LinkedList;
import org.apache.lucene.*;
public class Indexer {
 private String fileFolder = "";
 private String output;

 public Indexer(String folder,String output){
  this.fileFolder = folder;
  this.output = output;
 }
 //builds the index with the images specified in the path of the constructor
 public void buildIndex() throws IOException {
        String filesPath = fileFolder;
   //get the path of all the jpg images from a folder(a method we constructed)
        String[] fileNames = getFilesInFolder(fileFolder,"jpg");
        String indexPath = output;
   //create document builder
            DocumentBuilder builder = DocumentBuilderFactory.
                getExtensiveDocumentBuilder();
   //create index writer with the simple image analyzer
            IndexWriter iw = new IndexWriter(indexPath, new SimpleAnalyzer(),
                true);
            for (String identifier : fileNames) {
                // Build the Lucene Documents
                Document doc = builder.createDocument(new FileInputStream(
                    filesPath+identifier), identifier);
                // Add the Documents to the index
                iw.addDocument(doc);
            }
   //do some magic on the index for optimization
            iw.optimize();
   //close the image writing
            iw.close();

 }
}
```

Listing A.1: An example of indexing with LIRE

```java
import java.awt.image.BufferedImage;
import java.io.IOException;
import org.apache.lucene.*;
public class LireGenomeIndex {
 private String indexpath;
 private PreProcessingAlgorithm alg;
 private IndexReader reader;
 private float colorHistogramWeight = 0.1f;
 private float colorDistributionWeight = 0.1f;
 private float textureWeight = 1.0f;
 private int numberOfHits = 100;

 public LireGenomeIndex(String path, PreProcessingAlgorithm algorithm){
  indexpath = path;
  alg = algorithm;
 }
 public LireGenomeIndex(String path, PreProcessingAlgorithm algorithm, float
     histogram, float distribution, float texture){
  indexpath = path;
  alg = algorithm;
  colorHistogramWeight = histogram;
  colorDistributionWeight = distribution;
  textureWeight = texture;
 }
 public void loadIndex()throws IOException{
  reader = IndexReader.open(indexpath);
 }
 public String[] search(String query){
  String[] results = null;
  BufferedImage queryImage;
  ImageSearcher searcher;
  ImageSearchHits hits;
  queryImage = alg.processInput(query);
  searcher = ImageSearcherFactory.createWeightedSearcher(numberOfHits,
      colorHistogramWeight, colorDistributionWeight, textureWeight);
  hits = null;
  try {
   hits = searcher.search(queryImage, reader);
  } catch (IOException e) {
   e.printStackTrace();
  }

  results = new String[hits.length()];
  if(numberOfHits > hits.length()){
   numberOfHits = hits.length();
  }
  for (int i = 0; i < numberOfHits; i++) {
    results[i] = hits.score(i) + " - "+ hits.doc(i).getField(DocumentBuilder.
        FIELD_NAME_IDENTIFIER).stringValue();
  }
  return results;
 }
}
```

Listing A.2: Our LireGenomeIndex implementation

```java
import java.io.IOException;

public class LireGenomeIndexSearcher {
 private LireGenomeIndex[] indexes;

 public LireGenomeIndexSearcher(String[] indexpaths, PreProcessingAlgorithm[]
     algs){
  indexes = new LireGenomeIndex[indexpaths.length];
  for(int i = 0; i < indexpaths.length; i++){
   indexes[i] = new LireGenomeIndex(indexpaths[i],algs[i]);
  }
 }
 public void loadIndex() throws IOException{
  for(int i = 0 ; i < indexes.length;i++){
   indexes[i].loadIndex();
  }
 }

 public String[][] search(String input){
  String[][] resultlist = new String[indexes.length][100];

  for(int i = 0; i < indexes.length;i++){
   resultlist[i] = indexes[i].search(input);
  }

  return resultlist;
 }

}
```

Listing A.3: Our LireGenomeIndexSearcher implementation

```java
public BufferedImage Process(BufferedImage image){
 LinkedList<Color> pixels = new LinkedList<Color>();
 float[][] res = generateBISOfWholeSequence();
 float[] a,c,t,g = null;
 FloatFFT_1D fft = new FloatFFT_1D(window);
 double awr,awg,awb,cwr,cwg,cwb,twr,twg,twb,gwr,gwg,gwb = 1;
 int length = res[0].length;
 awr = cwr = twr = gwr = gwg = awb = cwb = twb = gwb = 10;
 int iter = res[0].length-window;
 int numberofpixels = (int) Math.floor(((jepp+1)*window)/2);
 int sizeofimage = (int) Math.floor(Math.sqrt(numberofpixels));
 image = new BufferedImage(sizeofimage,sizeofimage,BufferedImage.TYPE_INT_RGB);
 int y = 0;int x = 0;
 for(int i = 0; i < iter+1;i++){
  a = getSubBIS(res[0],window,i);
  c = getSubBIS(res[1],window,i);
  t = getSubBIS(res[2],window,i);
  g = getSubBIS(res[3],window,i);
  fft.realForward(a);
  fft.realForward(c);
  fft.realForward(t);
  fft.realForward(g);
  for(int j = 1; j < Math.floor(a.length/2);j++){
   //calculate spectrum values
   double spec_a = Math.sqrt((a[(2*j)+1]*a[(2*j)+1])+(a[(2*j)]*a[(2*j)]));
   double spec_c = Math.sqrt((c[(2*j)+1]*c[(2*j)+1])+(c[(2*j)]*c[(2*j)]));
   double spec_t = Math.sqrt((t[(2*j)+1]*t[(2*j)+1])+(t[(2*j)]*t[(2*j)]));
   double spec_g = Math.sqrt((g[(2*j)+1]*g[(2*j)+1])+(g[(2*j)]*g[(2*j)]));
   //calculate color values
   double c_r = 255%Math.abs((awr*spec_a)+(twr*spec_t)+(cwr*spec_c)+(gwr+spec_g)
        );
   double c_g = 255%Math.abs((awg*spec_a)+(twg*spec_t)+(cwg*spec_c)+(gwg+spec_g)
        );
   double c_b = 255%Math.abs((awb*spec_a)+(twb*spec_t)+(cwb*spec_c)+(gwb+spec_g)
        );
   Color pix = new Color((int)c_r, (int)c_g, (int)c_b);
   image.setRGB(x, y, pix.getRGB());
   x++;
   if(x == sizeofimage){
    x = 0;
    y++;
   }
  }
 }
 return image;
}
```

Listing A.4: Our frequency approach implementation in java

```java
public float[] getSubBIS(float[] ar,int window, int offset){
 float[] ret = new float[window];
 try{
  System.arraycopy(ar, offset, ret, 0, window);
 }catch(Exception e){
 }
 return ret;
}
private float[][] generateBISOfWholeSequence(){
 LinkedList<GenomeEntity> list = seq.getList();//seq is a class member
 int len = list.size()*10;
 float[] a = new float[len];float[] c = new float[len];float[] t = new float[len
      ];float[] g = new float[len];float[][] res = new float[4][len];
 int counter = 0;
 for(int i = 0; i < list.size(); i++){
  GenomeEntity item = list.get(i);
  String subseq = item.decode();
  for(int j = 0; j < subseq.length();j++){
   char ch = subseq.toLowerCase().charAt(j);
   a[counter] = 0;
   c[counter] = 0;
   t[counter] = 0;
   g[counter] = 0;
   if(ch == 'a'){
    a[counter] = 1;
   }else if(ch == 'c'){
    c[counter] = 1;
   }else if(ch == 't'){
    t[counter] = 1;
   }else if(ch == 'g'){
    g[counter] = 1;
   }
   counter ++;
  }
 }
 res[0] = a;res[1] = c;res[2] = t;res[3] = g;
 return res;
}
```

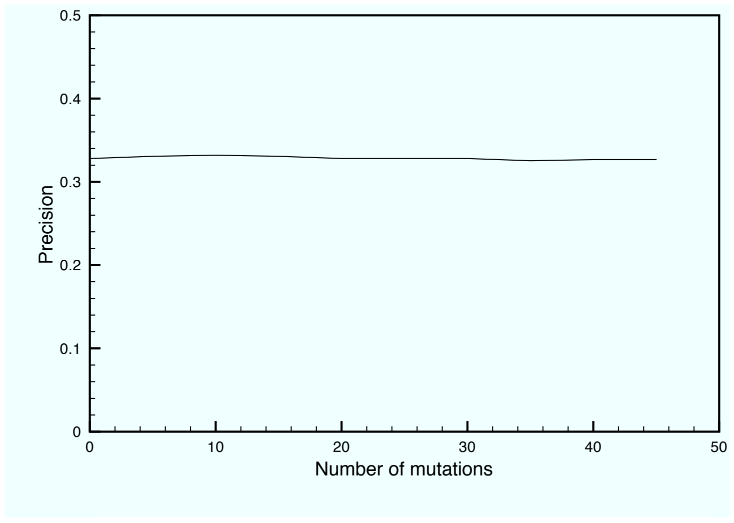
Listing A.5: Our frequency approach implementation in java(continued)

```java
public BufferedImage Process(BufferedImage image) {
 LinkedList<GenomeEntity> list = seq.getList();
 for(int i = 0; i < list.size();i++){
  String part = list.get(i).decode();
  Color tmp = getColorFromString(part);
  int y = (int) Math.floor(i/image.getWidth());
  int x = (i-(y*image.getWidth()));
  image.setRGB(x, y, tmp.getRGB());
 }

 return image;
}
private Color getColorFromString(String input){
 int r = 0,g = 0,b = 0,a = 0,colorIncrement = 0;
 if(input.length() == 0){
  return new Color(0,0,0,0);
 }
 colorIncrement = Math.round(255/input.length());
 for(int i = 0; i < input.length();i++){
  char nuc = input.charAt(i);
  switch(nuc){
  case 'a':
   r += colorIncrement;
   break;
  case 'c':
   g += colorIncrement;
   break;
  case 't':
   b += colorIncrement;
   break;
  case 'g':
   a += colorIncrement;
   break;
  }

 }

 return new Color(r,g,b,a);
}
```

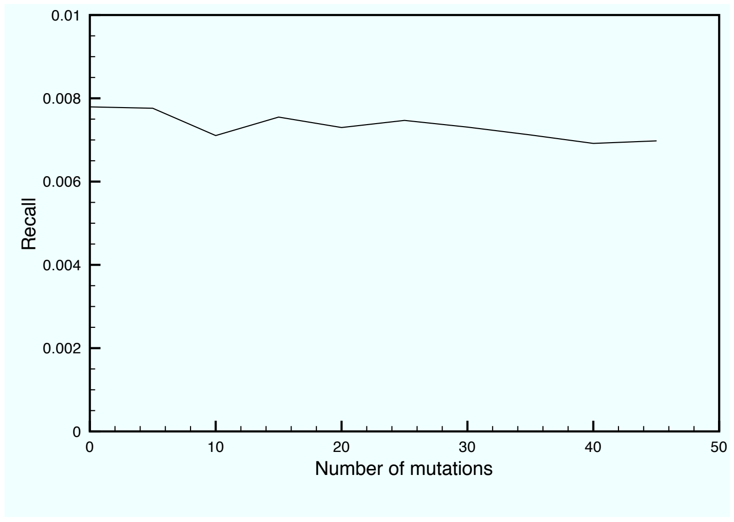
Listing A.6: Our RGBA approach implementation in Java

Appendix B

Precision/Recall Graphs

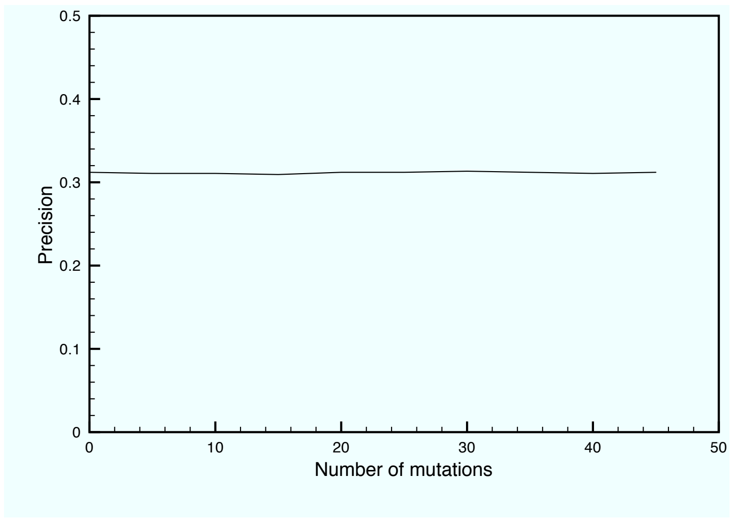


(a)

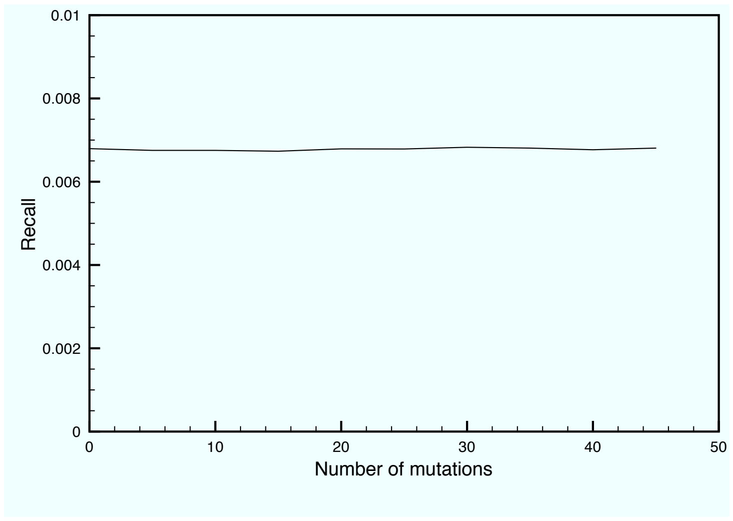


(b)

Figure B.1: Precision(a) and recall(b) values for the RGBA approach

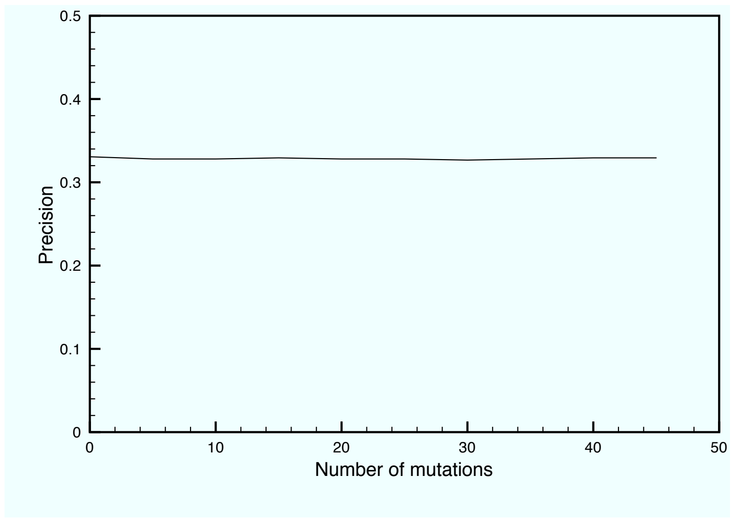


(a)

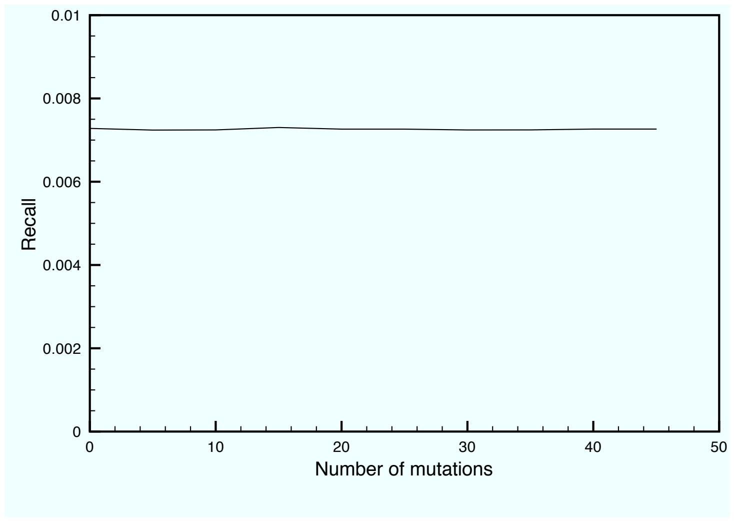


(b)

Figure B.2: Precision(a) and recall(b) values for the RGBA approach scaled 300%

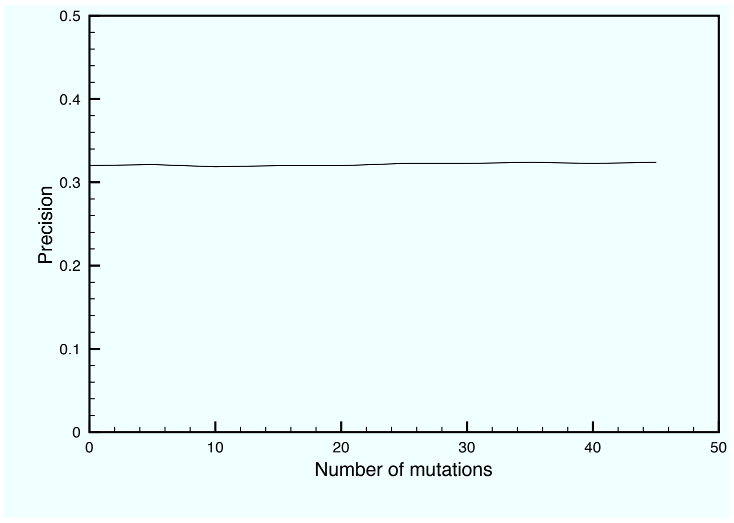


(a)

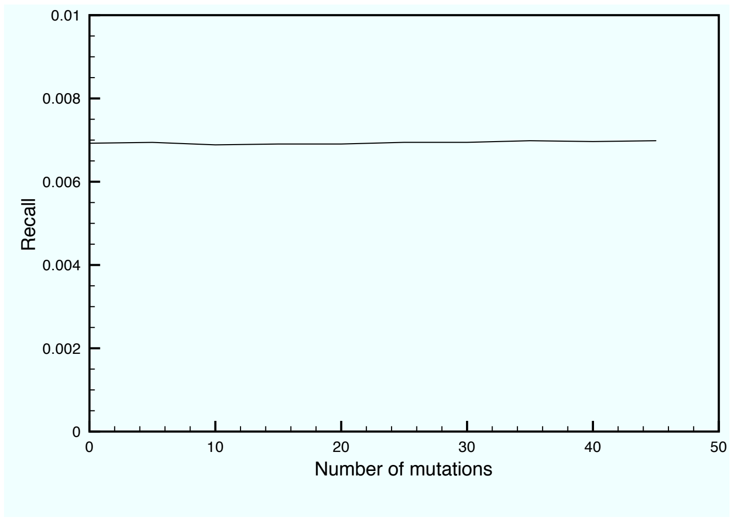


(b)

Figure B.3: Precision(a) and recall(b) values for the RGBA approach scaled 700%

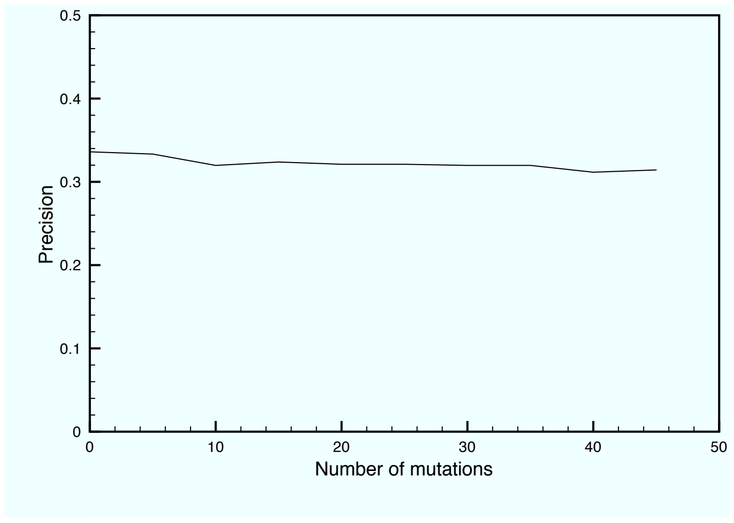


(a)

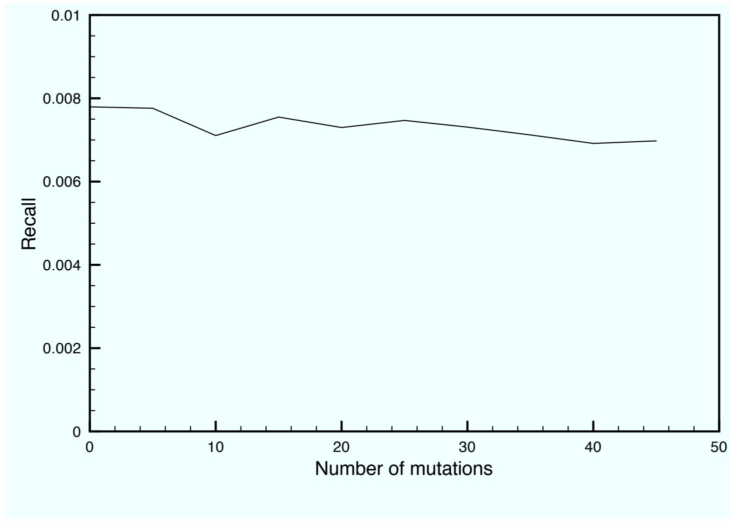


(b)

Figure B.4: Precision(a) and recall(b) values for the RGBA approach scaled 1000%

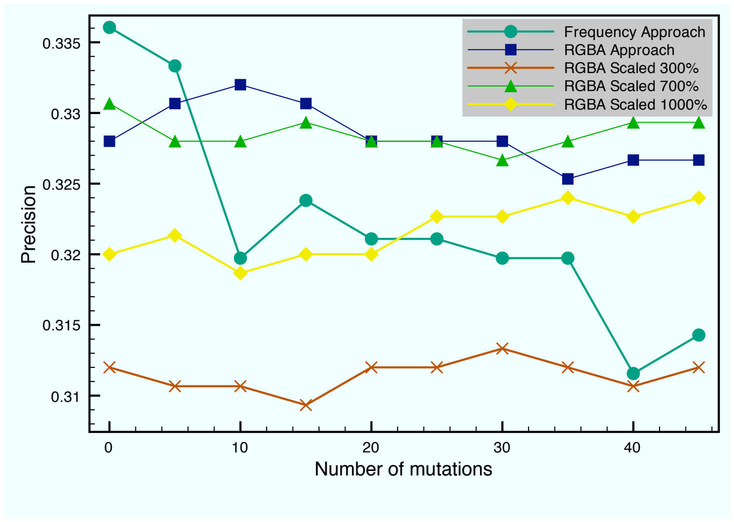


(a)

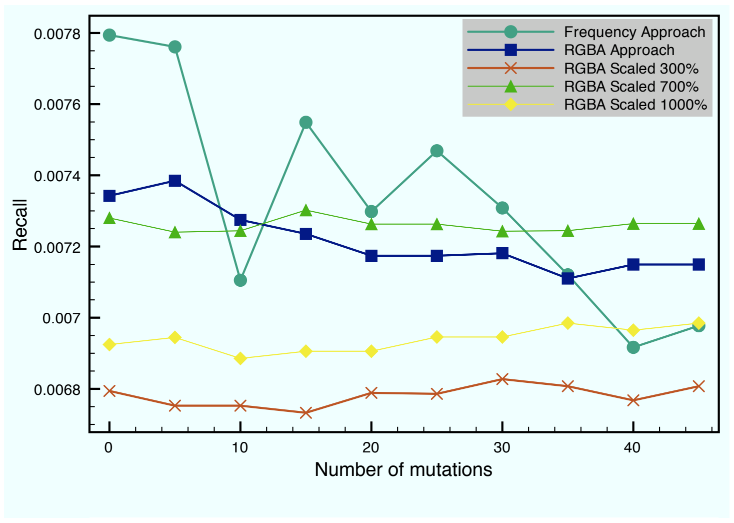


(b)

Figure B.5: Precision(a) and recall(b) values for the frequency approach



(a)



(b)

Figure B.6: Precision(a) and recall(b) values compared for all of the approaches