Nucleotide Sequence Similarity Search Using Techniques from Content-Based Image Retrieval

Eivind Lysne

Master of Science in Informatics
Submission date: June 2015
Supervisor: Herindrasana Ramampiaro, IDI

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

Nucleotide Sequence Similarity Search Using Techniques from Content-Based Image Retrieval

Eivind Lysne

June 1, 2015

Preface

This thesis was written as part of a masters degree program in informatics at the Department of Computer and Information Science at the Norwegian University of Science and Technology (NTNU).

Acknowledgements

Simply, thanks to my Mom and Dad for advice and support, and my supervisor Heri Ramampiaro for taking the time to supervise this thesis from the other end of the globe.

Abstract

The amount of DNA data continues to increase exponentially as a result of high-throughput next generation sequencing. Current state-of-the-art tools for nucleotide sequence similarity search are not equipped to deal with this growth and new thinking is needed to tackle the rising scalability challenges.

This thesis investigates the experimental approach of translating DNA sequences into images and applying state of the art techniques from the field of content-based image retrieval to index and search the resulting images. The challenges of translating DNA sequences into images are discussed and two algorithms for image generation are proposed. We look into the different feature descriptors that are available and evaluate them in the context of the generated images. Lastly the approach as a whole is evaluated with the mean average precision metric using BLAST as the gold standard reference.

The results show that the proposed approach is not successful in approaching BLAST in retrieval performance, but offers a significant reduce in index sizes and thus better performance and scalability on large DNA databases.

Sammendrag

Mengden av tilgjengelig DNA-data fortsetter å øke eksponensielt som følge av nyere teknologi for høyhastighets DNA-sekvensering. Dagens standardverktøy for DNA-sekvenssøk er ikke skikket til å håndtere økningen, og det er bruk for nytenkning for å takle skaleringsutfordringene dette medfører.

Denne masteroppgaven undersøker en eksperimentell metode der DNA-sekvenser transformeres til bilder, hvorpå ”concept-based” bildesøksteknikker blir brukt til å indeksere sekvensene og utføre sekvenssøk. Oppgaven diskuterer utfordringene med å transformere DNA-sekvenser til bilder, og to algoritmer for transformasjon foreslås. Videre ser vi nærmere på tilgjengelige ”feature descriptors” for bildesøk og hvordan de fungerer med bilder generert av de foreslåtte algoritmene. Til sist evalueres metoden ved ”mean average precision” med BLAST som referansesystem.

Resultatene viser at den foreslåtte metoden ikke lykkes i å nærme seg BLAST i søkesensitivitet, men gir en betydelig reduksjon i indeksstørrelse, noe som vil bidra til raskere ytelse og større skalerbarhet for store DNA-databaser.

Contents

List of Figures

List of Tables

List of Acronyms

ANMRR	Average Normalized Modified Retrieval Rank
BLAST	Basic Local Alignment Tool
CBIR	Content-Based Image Retrieval
CEDD	Color and Edge Directivity Descriptor
CGR	Chaos Game Representation
DNA/RNA	Deoxyribonucleic Acid / Ribonucleic acid
FASTA	Fast All
FCTH	Fuzzy Color and Texture Histogram
HSB	Hue Saturation Brightness
HSP	High-scoring Segment Pairs
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
JCD	Joint Composite Descriptor
LIRe	Lucene Image Retrieval
MAP	Mean Average Precision
MNRO	Mean Normalized Retrieval Order
MPEG	Moving Pictures Expert Group
NCBI	National Center for Biotechnology Information
P@n	Precision at n
RGB/RGBA	Red Green Blue / Red Green Blue Alpha

1. Introduction

1.1 Motivation

Several modern research fields such as molecular and evolutionary biology, genomics and forensics are using DNA heavily in research. As next generation sequencing techniques become ever faster, the amount of DNA data amassed will continue to increase rapidly as well. As of march 25. 2015 the European Nucleotide Archive consists of over 530 million sequence entries containing a staggering 1,154,095,724,206 nucleotides, with the total volume of data doubling roughly every 10 months [1].

Large amounts of data become useless if it cannot be indexed and searched effectively, and a prerequisite for this is having relatively small indexes that fit in memory. However, state-of-the-art tools in the field of bioinformatics such as BLAST uses the entire sequences in the index, resulting in large indexes with limited scalability. Therefore alternative methods to indexing for sequence similarity search needs to be explored in order to tackle the exponential growth of data.

1.2 Problem specification

This thesis investigates an experimental technique of representing DNA sequences visually as images, and uses state of the art content-based image retrieval methods for indexing and retrieval of sequences. This concept has previously been shown to be feasible [2].

The main focus is to develop algorithms for transforming a DNA sequence into an image, capable of representing the structure of the sequence such that similar sequences results in visually similar images. We will also need to investigate which types of feature descriptors work best for the images produces by the algorithm. The goal is to achieve a performance improvement over the algorithms in [2], and hopefully approach the established state of the art in terms of sequence similarity search.

1.3 Thesis structure

This thesis is structured as follows: Chapter 2 will review the background theory and concepts related to this thesis, and other related studies.

Chapter 3 will be an overview of the state-of-the-art technologies for DNA sequence similarity search and CBIR, and a review of the limitations that exist.

In chapter 4 we will present the approach and explain in detail how it works and how it solves the problem specified in the above section. This chapter will also contain a description of the prototype system that was developed to demonstrate and evaluate the approach.

Chapter 5 is an evaluation of the approach. Therein we will outline the process of evaluating CBIR systems in general, common retrieval metrics, and how this particular evaluation was carried out. Then we will present the results of the evaluation.

Finally, in chapter 6 we discuss the significance and validity of the results and offer a conclusion on the findings in this thesis. Lastly, we will also discuss possibilities for future work in this area.

2. Background Theory

2.1 Related work

This thesis builds upon the work of another master thesis whose results are described in [2]. The authors introduces the idea the idea of applying CBIR techniques to DNA sequence similarity search, and presents two algorithms for generating images from DNA sequences, one using color histograms and one based on frequency spectrograms. Their results show the promise of the idea in the areas of performance and index size, but they also show a lack of sensitivity compared to BLAST. In the time since that thesis was written BLAST has had many new releases and the field of CBIR has evolved to the point that it renders the comparison somewhat obsolete, although it successfully demonstrates the potential of the approach.

2.2 Deoxyribonucleic acid, DNA

DNA is a molecule present in the cells of all living organisms which acts as a blueprint for the proteins that make up the organism. The information is encoded as sequences of the nucleotides *adenine*, *cytosine*, *thymine*, and *guanine*, usually represented by the letters *A*, *C*, *T*, and *G* respectively. Letters in groups of three form *codons*, which code for amino acids, which in turn form larger sequences coding for a particular protein. DNA sequences also contain regions that does not code for proteins. In some organisms, like humans, the non-coding parts make up the vast majority of the genome while the opposite is true for bacteria.

The structure of the DNA molecule was first identified by Watson and Crick in 1953 [3]. Although the existence of the DNA molecule had already been known for decades, the discovery of the double helix structure of was the key to understanding its function.

2.2.1 RNA

RNA is, like the DNA molecule, a nucleic acid and shares many similarities in structure. One important difference is that in RNA sequences the nucleotide thymine is replaced by uracil, meaning the same information is encoded with a slightly different alphabet.

RNA serves several biological functions such as transcription of DNA for building proteins. The RNA molecule is typically single-stranded but can occur as double-stranded in some cases, an example being the genome of certain viruses.

2.2.2 Sequence alignment

Sequence alignment is the process of taking two or more sequences and arrange them next to each other in rows, in order to identify regions of similarity between them. This technique is obviously very useful in sequence similarity search where the alignment of two sequences is the most common model of similarity between the two sequences.

Sequence alignment can be categorized into global and local alignment. Global alignment is most useful in comparing sequences with a high degree of similarity and comparable length. Local alignment was introduced to work better with sequences with isolated regions of similarity. Figure 2.1 is an illustration of the differences between global and local alignment between the same the same two sequences. Although the number of matches is identical, the global alignment has shorter matching regions as it tries to align the whole sequence.

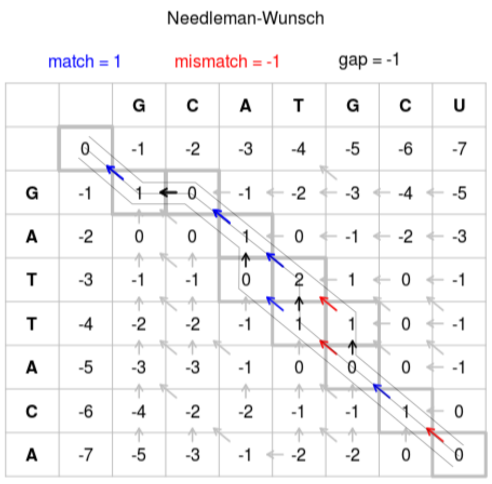
```
Global                          Local

GTGTACGCCAGAT                   GTGTACGCCAGAT
|  ||| ||| ||                     |||| |||||
G--TAC-CCA-AT                   --GTAC-CCAGA-
```

Figure 2.1: Global versus local alignment

Alignment of three or more sequences at the same time, called *multiple sequence alignment*, is a related problem with its own set of techniques and algorithms to address the additional complexity. These are generally not used in the case of sequence similarity search where pairwise alignment is more useful.

The Needleman-Wunsch algorithm

The Needleman-Wunsch algorithm [4] is an dynamic programming algorithm for finding an optimal global alignment of any two strings. It works by creating a *NxM* matrix where *N* and *M* is the equal to the length of the respective strings. All the cells of the matrix, each corresponding to a position in both strings, are then given a score with respect to a scoring scheme which can vary. An optimal alignment is then found by tracing a maximum path from the end corner of the matrix back to the starting row/column.



		G	C	A	T	G	C	U
	0	-1	-2	-3	-4	-5	-6	-7
G	-1	1	0	-1	-2	-3	-4	-5
A	-2	0	0	1	0	-1	-2	-3
T	-3	-1	-1	0	2	1	0	-1
T	-4	-2	-2	-1	1	1	0	-1
A	-5	-3	-3	-1	0	0	0	-1
C	-6	-4	-2	-2	-1	-1	1	0
A	-7	-5	-3	-1	-2	-2	0	0

Figure 2.2: Needleman-Wunsch[1]

Although the algorithm is costly, the original recursive algorithm being $O(n^3)$, it is still widely used when finding an alignment of optimal quality is important. Another approach was proposed recently by Chakraborty & Bandyopadhyay [5], which claims a considerable speed improvement over the Needleman-Wunsch algorithm.

The Smith-Waterman algorithm

Smith-Waterman [6] is a variant of the Needleman-Wunsch algorithm adapted for finding local alignments. It is, like Needleman-Wunsch, guaranteed to find an optimal alignment. The approach is largely the same except that the scoring is never negative, and backtracing is done from the highest scoring cell along the maximum path until a zero is encountered.

[1]https://commons.wikimedia.org/wiki/File:Needleman-Wunsch_pairwise_sequence_alignment.png

2.2.3 Sequence indexing

Data indexing is a requirement for performant, scalable search for any type of data. In the case of DNA data, classical string indexing data structures such as suffix trees, suffix arrays, and inverted indexes based on n-grams, offer very little compression for biological sequences [7, 8] and the resulting indexes are therefore too large to be feasible.

The Burrows-Wheeler transform is a reversible transformation which is widely used in compression. The transformation does not provide any compression itself, but makes the string more suitable for compression by rearranging the symbols into repeated runs of the same symbols. Indexing based on the Burrows-Wheeler transform has been applied to aligning short reads onto a reference genome as a part of the process of high-troughput sequencing [9]. Another study used Burrows-Wheeler transform indexing to speed up optimal local alignment in sequence similarity search [10]. The result was a huge speedup over Smith-Waterman for uncompressed sequences, but BLAST was still much faster.

Compressed suffix arrays and the full-text minute-space index which is based on the Burrows-Wheeler transform was used in [7] and found to be feasible for use on entire genomes.

Research using n-gram- (sometimes referred to as q-gram in the literature) indexes with additional layer compression have seen decent results in compression, but sequence matching performance for longer queries degrades quickly [8, 11].

2.2.4 Alignment-free sequence analysis

Alignment free sequence analysis refers to any method of comparing sequences not based on sequence alignment. Such methods emerged both as a response to the heavy computational complexity of optimal alignment, and the issue of alignment after genetic recombination [12].

A novel method of DNA sequence analysis using histograms was presented in [13]. The authors compute the similarities of resulting histograms, and plot the resulting correlation matrices using different methods such as multidimensional scaling and directed graphs.

Signal processing techniques can be applied to a DNA sequence by converting it into into one or multiple discrete signals. Most useful in the case of DNA sequences are time-frequency representation and analysis. DNA spectrograms computed using short-time fourier analysis have been proposed in [14, 15, 16], and scaleograms using wavelet transforms was used in [17, 18].

Iterated maps and Chaos Game Representation

The use of an iterated map (called iterated function system in older literature) for representing DNA sequences graphically was first introduced in [19] proposing a Chaos Game Representation (CGR), which was inspired by a technique for creating fractals called the Chaos Game. A CGR of a DNA sequence is created by first creating a square with each corner labeled A,C,T and G after the four nucleotides. Then a point is created for each letter in the sequence halfway between the previous point and the corner labeled with the current letter, starting from the center of the square.

A significant number of studies have been done on the subject since, which are reviewed in [20], and interesting properties of CGRs have been uncovered. The most significant discoveries being that they are a scale-independent representation of a Markov probability table [21], and also a bijective mapping function. This means among other things that they can uncover local similarities between sequences. An example of a CGR is shown in figure 2.3. Notice the self-similar patterns which occur both in the larger and smaller quadrants of the image.

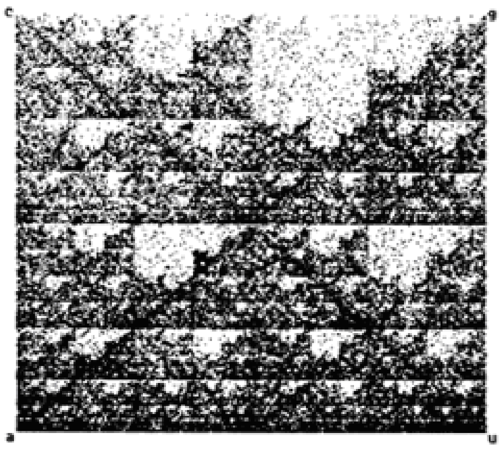


Figure 2.3: CGR of Human Beta Globin Region on Chromosome 11 [19]

2.3 Content Based Image Retrieval, CBIR

Content based image retrieval is the indexing and retrieval of images based on the low level features of an image, which includes color-, shape-, and texture information. As digital image collections grow in size the need for such systems which does not rely on textual metadata produced by humans for organizing and retrieval of images becomes ever greater. As a result, extensive research has been done on the subject in the last years [22, 23].

2.3.1 Feature extraction

Features and the extraction of said features play a huge part in CBIR. Feature extraction is the process of mapping an image to a vector in feature space. This is done as part of both the task of indexing images and querying an index using an example image, as can be seen in figure 2.4 which shows a general CBIR system. Image similarity can then be computed as the distance between feature vectors.

Feature extraction algorithms are the backbone of CBIR systems and are critically important to the performance of a system. Mapping an image into feature space requires computing a vector that sufficiently represents the low-level properties of an image without infeasible dimensionality.

2.3.2 Query by example

CBIR systems are in most cases *query by example*, meaning an image is used as a query with the intention of finding similar images rather than providing a textual query. A textual query would have to be converted to an appropriate feature vector, which is a highly complex and difficult problem to solve. CBIR systems are therefore arguably less user-friendly than traditional systems, especially for very simple use cases such as a user wanting to search for images of cats. In the case of query by example the user would have to provide an example image of a cat and possibly receive several results not containing cats, although this would depend on the system.

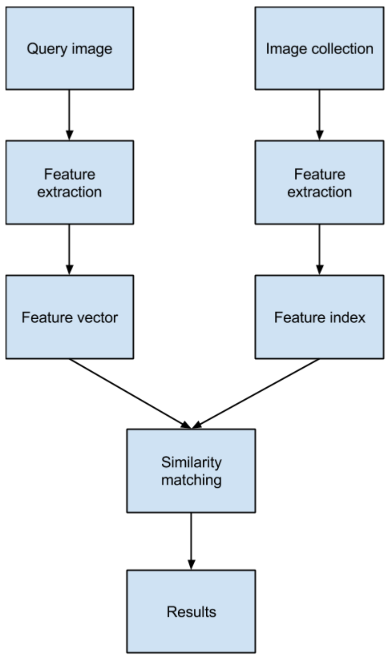


Figure 2.4: Simple illustration of a CBIR system

2.3.3 Compact composite descriptors

Compact composite descriptors is a term for feature descriptors combining multiple features in a single compact feature vector. A trivial layout example for such a descriptor is shown in 2.1. For most types of images using a combination of features will yield better results than relying on a single feature [24], e.g. a color histogram, considering the limited information it provides. It comes with the cost of increased dimensionality however, thus requiring a more compact representation of the individual features.

$$V = [\overbrace{v_1, v_2, .., v_k}^{colorfeature}, \overbrace{v_{k+1}, v_{k+2}, .., v_n}^{texturefeature}] \tag{2.1}$$

2.3.4 Medical image retrieval

In recent years the potential for using CBIR in medical image retrieval has been the subject of much research [23, 25], and a composite feature descriptor intended for radiology images was recently proposed in [26].

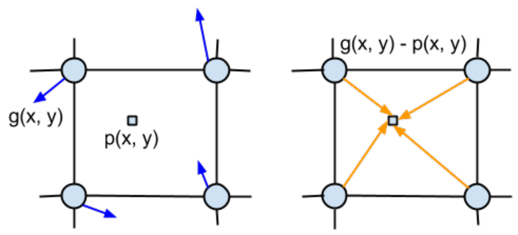


Figure 2.5: Perlin Noise pixel calculation

2.4 Procedural texture generation

Procedural texture generation is the process of generating textures programmatically, usually with the use of pseudorandom algorithms. Pseudorandom in this context refers to the property of appearing random, while still producing the same output given the same input.

Procedural texture generation is widely used in computer graphics for creating natural looking textures. Additionally, procedurally generated textures can be used as heightmaps for producing realistic, randomly generated terrain. Proceedural texture generation has recently seen a rise in popularity in computer games, with Minecraft being the most well known example. Some games, like the upcoming No Man's Sky, takes procedural generation to a completely new level using it to generate a whole universe.

2.4.1 Perlin Noise

Perlin Noise the classic algorithm for proceedural texture generation and was developed by Ken Perlin who first described it in 1985 [27]. It works by dividing space by a hypergrid (n-dimensional grid) and generating vectors of pseudorandom numbers for each grid point. The value of each point in the space is calculated by first computing the distance vector between the surrounding grid points and the point, then taking the dot product between the pseudorandom vector and the distance vector for the point for each surrounding grid point, and finally interpolating between the four values. Each point thus becomes a weighted average of the surrounding grid points with respect to distance. The process for the 2-dimensional case is visualized in figure 2.5.

2.4.2 Simplex Noise

Simplex Noise [28] is another algorithm developed by Ken Perlin to address some of the limitations of the Perlin Noise algorithm. The concept is exactly the same, but instead of dividing space by a hypergrid Simplex Noise divides spaces by *n*-simplices. The result is fewer vertices and thus lower computational complexity and higher dimensional scalability. Simplex Noise also reportedly does not suffer from the same visual artifacts that can be present in Perlin Noise. An example of a texture produced using Simplex Noise can be seen in figure 2.6b.

2.4.3 Worley Noise

Named after Steven Worley who introduced it in 1996 [29], Worley Noise takes a different approach than the two above. Feature points are placed randomly across the texture space. The value of a specific point in space is defined simply as the distance to the *n*th closest feature point. This creates a cell-like appearance similar to that of a Voronoi diagram, and Worley Noise is often referred to as Cell Noise because of it. Different effects can be achieved by utilizing different distance measures, i.e. utilizing manhattan distance would produce a different appearance than classic euclidean distance when using the same feature points. An example texture created by a Worley Noise function using euclidean distance is shown in figure 2.6c.

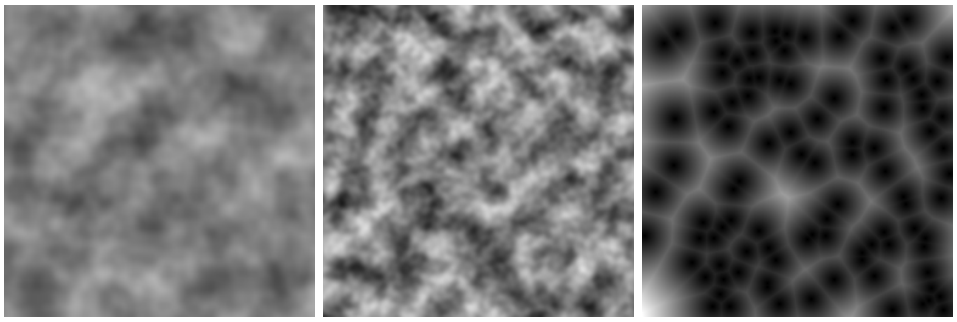

(a) Perlin Noise (b) Simplex Noise (c) Worley Noise

Figure 2.6: Comparison of different noise functions

3. State of The Art

3.1 Sequence similarity search

3.1.1 FASTA

FASTA [30] was originally a tool for protein sequence alignment, and was later extended to include functionality for nucleotide sequences. It is based on heuristics, meaning it is not guaranteed to find the optimal alignment, but fast enough to be feasible for large databases unlike optimal approaches based on dynamic programming.

An important legacy of FASTA is the FASTA file format, which has become a standard format for representing amino acid- and nucleotide sequences in text format. It is a very simple text based format and is easily parsable. A FASTA-file consists of one or more sequences separated by a single-line sequence header. The headers-lines are prefixed by the character '$>$' and contains unique identifiers and optionally a description of the sequence. Figure 3.1 shows an example taken from the data-set used later in the thesis.

3.1.2 BLAST

BLAST [31] is another heuristic based tool which has become the de facto standard for amino acid- and nucleotide sequence similarity search. It has lower time-complexity than FASTA as well as comparable sensitivity.

BLAST's heuristic search strategy consists of finding short matches that score above a certain defined threshold value, before extending them to produce local alignments.

The BLAST algorithm will first create a list of all the words of k-length in the sequence. The value of k will be different depending on whether the sequence

```
>ENA|AAAA02000001|AAAA02000001.1 Oryza Sativa Indica Group cultivar\
 93-11 chromosome 1 Ctg000001, whole genome shotgun sequence.
GGGACTCTCCAACGGCTCCCCGAGGAGCTCGAGAGGACGATTAAGTCATCCTCGAGGGAC
CTCGCCCGAGGAGCGGTGGAGCTCGTACTGGCGAGTTACCAGGCCAGGACCCCGACTTCT
CCCCATGGACGGCGCTGGACGAGTTCCCTCCCGGGACCGAGGACGGCGCGCGCGCGCAGG
.....
CCCCCGCGGCCCATCGGCCCTTCCCCGCGC
>ENA|AAAA02000002|AAAA02000002.1 Oryza sativa Indica Group cultivar\
 93-11 chromosome 1 Ctg000002, whole genome shotgun sequence.
CCGGGCCCTACCTCGCCTGCCAAGCGGAGGGCCTCGACCTCCCTGGCGGACGAGTCTAGC
GCGCCCTGCAGATCCGCGATGGTGTGTTCGGCAGCTGCGAGGCGGGCGGTTAAGTCGCTT
TCGCCCGCGGCCGCCCCGCCGCTGCGCGCCCTCGCGTCCAGCTCCTTCACCCGTGCCTCG
.....
```

Figure 3.1: Part of a FASTA-file

is an amino acid- or nucleotide sequence. For nucleotide sequences the default word length is 11. An example for $k = 3$ is shown in figure 3.2. The number of words generated from a sequence will be $(n-k)+1$, where n is the length of the sequence. The assumption is that a sequence with a good alignment to the query will contain many highly similar matches of short words.

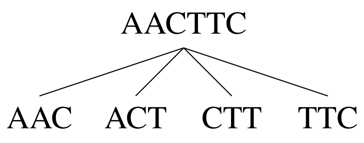


Figure 3.2: Word list generation

Secondly BLAST compares the word list to the database and assigns scores to each word based on a scoring scheme. Words scoring below a specified threshold are then dropped from the word list. The database is then searched for exact matches of the remaining words. BLAST attempts to extend any found matches into what is called High-Scoring Segment Pairs (HSPs), which can be further merged into longer alignments if appropriate. The generated HSPs that score above a certain cutoff value are kept and the rest are dropped.

3.1.3 BioJava

BioJava [32] is an open source bioinformatics library written in Java. It has several modules supporting among other things sequence alignment, sequence annotation, and working with and comparing biomolecular structures. It can also access web services such as the NCBI's BLAST service as well as parse most file formats used in bioinformatics.

3.2 Content-based image retrieval

3.2.1 MPEG-7

MPEG-7 [33] is an ISO/IEC standard from the The Moving Picture Experts Group. It defines a number of multimedia descriptors, many which are applicable to content-based image retrieval [22, 34].

3.2.2 Lucene and LIRe

Apache Lucene is a Java library for building information retrieval software. It powers many state of the art search engines such as Elasticsearch and Apache Solr, and is used by major companies like Twitter and Facebook to power their search functionality.

LIRe (Lucene Image Retrieval) [34, 35] is a Java library providing CBIR features on top of Apache Lucene, which by itself only provides on indexing and retrieval of textual data. It implements most state of the art feature descriptors from the recent MPEG-7 standard as well as the compact composite descriptors mentioned above.

3.2.3 Feature descriptors

CEDD

Color and Edge Directivity Descriptor [36] combines color and texture information together in a single histogram of 54 bytes, which is a relatively small size compared to most other composite descriptors. Another advantage of this descriptor is the low computational cost of extraction, meaning better performance for both indexing and retrieval.

Color information is extracted using a fuzzy rule-set, while texture information is extracted using the 5 digital filters defined by the MPEG-7 Edge Histogram Descriptor. Together they form a 144-bin histogram of which each bin is quantized to consist of only 3 bits.

FCTH

Fuzzy Color and Texture Histogram [37] is another descriptor that combines both color and texture information. The size of the descriptor is 72 bytes per image, making it slightly larger than CEDD but still suitable for large image databases.

FCTH uses the same approach as CEDD for extracting color information, but differs in case of texture extraction. Texture information is extracted by computing the Haar wavelet transform on blocks of the image and applying a fuzzy rule-set. The resulting histogram is 192-bins in size and quantized to 3 bits per bin.

JCD

Joint Composite Descriptor [38] is a combination of CEDD and FCTH taking advantage of the fact that the two descriptors use the same exact color extraction system. Combining the the two descriptors therefore means combining the texture information areas only, resulting in a manageable size gain over FCTH which is the larger one of the two.

Luminance Layout

Luminance Layout is a very simple descriptor intended for use for grayscale images. It works simply by scaling the image down to a very small size, and using the resulting image as the feature vector itself.

Gabor

The Gabor feature uses a set of linear filters with different orientations and frequencies to extract texture information from an image. It has been successfully applied to multiple computer vision tasks, such as face recognition, iris recognition and fingerprint matching.

3.3 Analysis and limitations

This thesis relies on the assumption that existing state of the art in CBIR is good enough for the approach presented in the next chapter. This section will therefore focus on Sequence similarity search and, more specifically, BLAST.

As mentioned briefly in chapter 1, BLAST produces rather large indexes since it has to index the whole sequences as a result of every character in the sequence being equally important. This is in contrast to traditional ”document” information retrieval where only part of the terms in a document needs to be indexed. Table 3.1 shows the size of the resulting blast index for three different sized data-sets. One can see that the size of the BLAST index is roughly 1/4-th of the size of the original FASTA-file itself, and this size difference is mostly due to converting from UTF-8 to a more compact binary representation.

Table 3.1: BLAST index sizes

# sequences	FASTA file size	BLAST index size	percent size
10	175004 B (0.167 MB)	44852 B (0.043 MB)	25.63
29392	313285943 B (299 MB)	82188282 B (79 MB)	26.23
50231	423889187 B (405 MB)	112899074 B (108 MB)	26.63

Since BLAST performs linear scans of the index having large indexes is a problem, particularly for very large data-sets in which case the index might not fit in the available memory which in turn would have a severe impact on performance.

4. Approach

In this chapter we describe two different algorithms for generating images from DNA sequences, and the system used for indexing and retrieval. We also highlight any design choices made, why they were made, and the resulting effect.

4.1 Image generation

A lot of research has been done in visualizing DNA sequences through various means with the purpose of making DNA sequences easier to analyze by humans. In this thesis however, images are to be analyzed by computers and this raises somewhat different concerns as computer vision works very differently from human vision.

4.1.1 Requirements

There are a couple of requirements that one should aspire to meet when designing such an algorithm. Firstly it needs to be deterministic, i.e. given the same DNA sequence it needs to produce the same exact image every time. Secondly given two similar DNA sequences the resulting images need to be visually similar in a way that can be detected by existing CBIR techniques. Thirdly, in order to match existing state of the art technology i.e. BLAST, sequences that are not highly similar but contain significant matching regions should have this reflected in the resulting images.

4.1.2 Challenges and considerations

Symbolic to numerical space translation

What we are formally doing is designing a function that maps from symbolic space with a small alphabet to numerical space.

$$f : \{A,C,T,G\} \to \mathbb{R} \tag{4.1}$$

This can be approached in several different ways, and additional information such as a the position of a symbol in the sequence and its neighboring symbols can be taken into account.

One of the algorithms in [2] uses the bijective mapping between the four letters of the nucleotide alphabet and the four channels of the RGBA color space.

$$\begin{aligned} f: \quad & A \mapsto R \\ & C \mapsto G \\ & T \mapsto B \\ & G \mapsto A \end{aligned} \tag{4.2}$$

The problem with this is that the alpha-channel is ignored by the LIRe library, which is also used in [2], and seems to be largely ignored by the field of CBIR as a whole. Using this trivial translation therefore results in a loss of information.

Image size normalization

The most intuitive way of designing an algorithm for our purpose would be to have image sizes vary with respect to sequence length, i.e. a longer input sequence would result in a higher resolution output image, and this is indeed the case for both algorithms in [2]. The algorithms that are presented later in this chapter by contrast produce images of constant resolution regardless of sequence length. This has some potential benefits.

Local alignment allows for sequences of potentially very different lengths to be compared without significant penalties. In CBIR however, images of very different sizes would need to be normalized either by scaling the images themselves or normalizing their feature vectors. By having normalization as a part of the image generation algorithm we can control directly how it occurs.

Table 4.1: Supported ambiguous nucleic acid codes

Code	Meaning
R	A, G
Y	C, T, U
K	G, T, U
M	A, C
S	C, G
W	A, T, U
B	C, G, T, U
D	A, G, T, U
H	A, C, T, U
V	A, C, G
N	A, C, G, T, U
X	masked
-	gap of unknown length

Considering the vast amount of potential data, a fast image generation algorithm is paramount to the viability of the approach as a whole. Having the image resolution depend on input sequence length gives the algorithm worse performance guarantees from an algorithmic standpoint than using constant image size. In practice this does not mean that the algorithm will be slower, especially for short input sequences, but having good performance guarantees makes an algorithm more practical.

Ambiguous symbols

DNA sequences have a four letter alphabet A, G, C, T. In the case of RNA T is replaced by U, but for our purposes they are interchangeable and can be treated as if they are the same letter. The FASTA format supports several ambiguous characters (listed in table 4.1) which can mean two or several different letters, or even an arbitrary gap in the sequence. This adds a lot of complexity, making it infeasible to support them in any application. The way BLAST deals with these codes is to either treat them as mismatches, or to treat them all as N which is supported by BLAST. BLAST does not support dashes and will reject any input containing them.

Following the example of BLAST the only ambiguity code to consider is N, and there are different ways to deal with it. One way would be to consider it part of the alphabet itself as a unique character. This conflicts with the meaning of N and requires the algorithm to support a larger alphabet, which can add complexity depending on the method used. Another way would be to remove the ambiguity by translating the Ns into unambiguous symbols that are part of the alphabet. This would introduce an unknown bias [13], since representing each case for each occurence of N would be intractable. The final method we considered, and the one that was used in [13], is to simply remove all occurrences of N. This is by far the simplest solution, but it relies on the assumption that ambiguity symbols are irrelevant in the context of sequence similarity search. Additionally, the sequences containing the symbol N will have their length reduced. We decided on removing any occurrences of N for its simplicity.

4.1.3 Value Noise algorithm

This algorithm is inspired by proceedural texture generation. It is essentially a derandomized version of an algorithm called Value Noise, which is itself a simplified version of Perlin Noise described in chapter 2. The difference lies in the use of single values instead of gradient vectors, which makes the calculations simpler. By letting the input sequence determine the values of the grid the algorithm becomes deterministic instead of pseudorandom.

First we create a square 2-dimensional grid, which width/height is a function of the length of the input sequence n, and a fixed value k which indicates how many letters of the sequence to be used for a single cell value. $k = 12$ was selected as the default value.

$$grid\_size = \lceil \sqrt{n/k} \rceil \tag{4.3}$$

Each grid cell consists of three values corresponding to the RGB channels. The values are computed from a sub-sequence of length k of the input sequence using the formula 4.4. For each occurrence of A, C, and T in the sub-sequence the corresponding channel is incremented by the value y. In the case of G, the R and

G channels are both incremented by $y/4$ to produce a more yellow color.

$$(r,g,b) = \sum_{i=0}^{k} f(s_i) \qquad ,f(x) = \begin{cases} (y,0,0) & \text{if } x = A \\ (0,y,0) & \text{if } x = C \\ (0,0,y) & \text{if } x = T \\ (\frac{y}{4},\frac{y}{4},0) & \text{if } x = G \end{cases}, \qquad \begin{matrix} x \in \{A,C,T,G\} \\ y = 1/k \end{matrix} \tag{4.4}$$

The grid is then used to produce an image of size 1024x1024 by expanding the grid to the size of the image, and computing the value of every pixel by interpolating between the nearest grid values. The grid's size depends on the sequence length and might not fit perfectly over the image, in which case the image will have a tiny black border along the right and bottom sides.

The last step consist of inverting the brightness of the image. This is done by first converting the RGB value to the HSB color space, inverting the B-channel which denotes ”brightness” in the HSB color space, and then converting the result back again to RGB. This creates marble-like textures in the image which seems to be better detected by feature extractors. The difference is shown in figure 4.1.

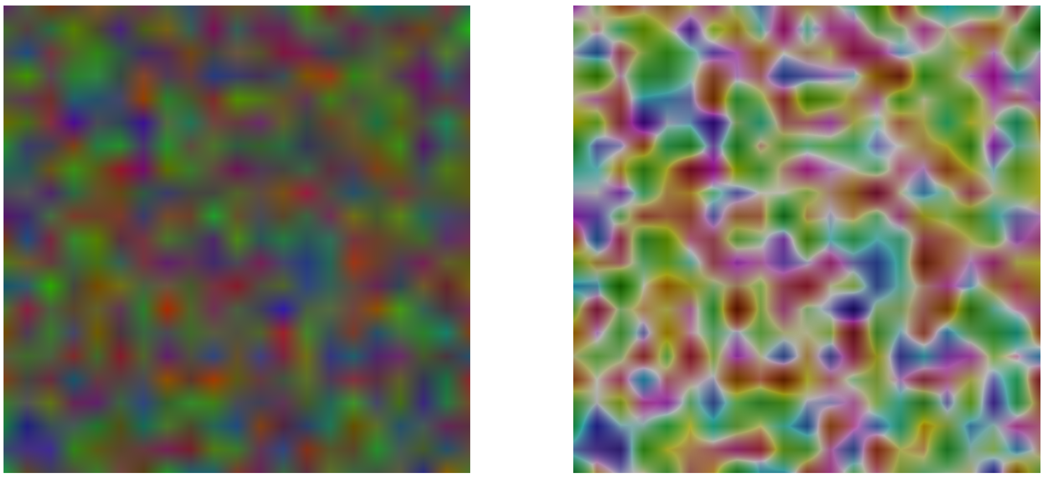

(a) Without inverted B-channel (b) With inverted B-channel

Figure 4.1: Before and after inverted B-channel

Word size and grid expansion

Each cell in the grid has its RGB value determined by a word in the DNA sequence of length 12. The word length affects the grid dimensions, which in turn affect the level of detail in the resulting image. A word length of 12 was chosen to closely match the word length used by BLAST in its initial steps which is 11, but ultimately kept as default since it was found to provide a good trade off value for long versus short sequences. This might not be the case for any data set however, in which case the word length can be tuned to a higher or lower value.

In most cases a DNA sequence will not fit perfectly in a symmetrical grid. There are two simple ways to solve this: Either have the grid too small and fit all the extra symbols into the last cell, or have a larger grid and keep the last cells empty. Figure 4.2 shows the two cases for a simple case with word length 4. In the first case more than half of the sequence gets pushed into the last cell, which means that those symbols contribute less to the resulting image's appearance. In the latter case, a portion of the bottom left of the image will simply be white (after inverting the brightness). Choosing to always use the larger grid seems to be the better solution as it does not bias part of the sequence.

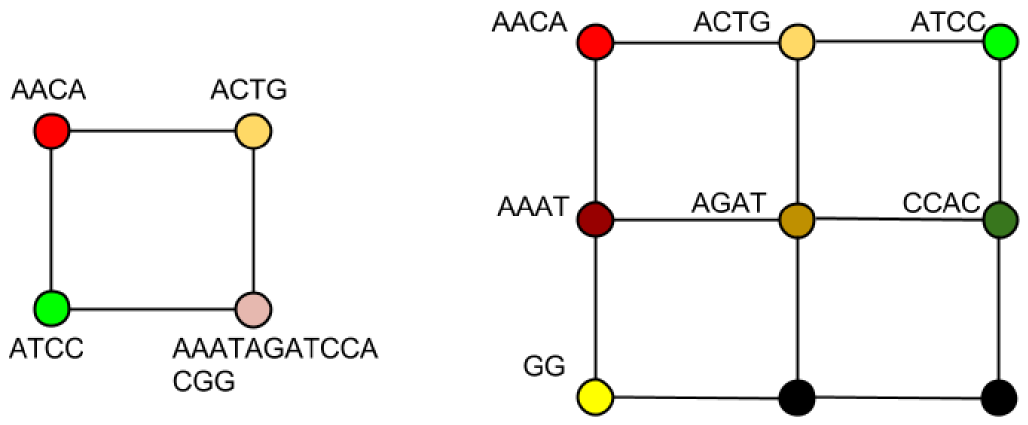


Figure 4.2: Grid expansion

Interpolation function

There are several different interpolation functions which usually offer different trade-offs between visual quality and computational cost. We considered two well known methods for use in this algorithm.

The fastest method by far is bilinear interpolation, which is the 2-dimensional version of linear interpolation. The function for linear interpolation 4.5 is beautifully simple and can be extended to two dimensions by first interpolating in one direction and then the other.

$$lerp(a,b,f) = (1-f)*a+f*b \qquad f \in [0,1] \tag{4.5}$$

Cosine interpolation is the other method we considered. It is much slower to calculate, but can be sped up by utilizing memoization on the function $g(x)$, caching the results.

$$cos\_interp(a,b,f) = (1-g(f))*a+g(f)*b \qquad \begin{array}{l} f \in [0,1] \\ g(x) = (1-\cos(x*\pi))*0.5 \end{array} \tag{4.6}$$

Figure 4.3 provides a side by side comparison of two images computed using the different interpolation methods. Bilinear tends to produce grid-like visual artifacts which are clearly visible in 4.3a, but not present in the image with cosine interpolation (4.3b). In our case visual quality is not a factor, but the choice of interpolation function will have a huge impact on performance. We therefore chose to use bilinear interpolation to maximize speed.

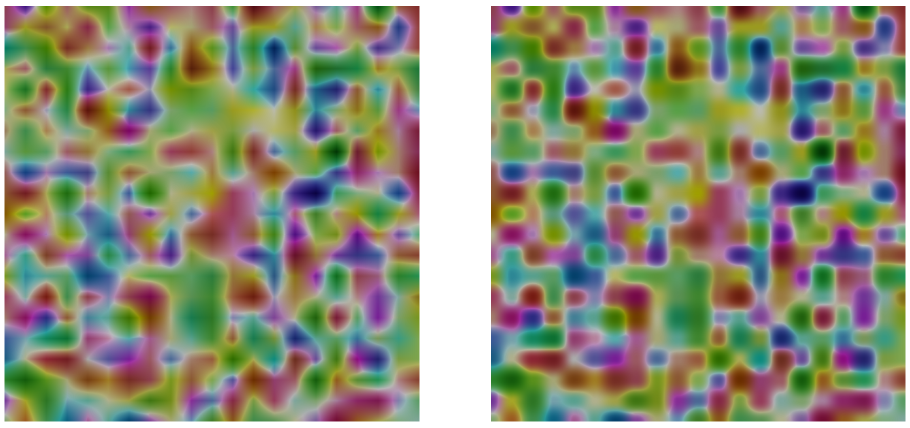

(a) Bilinear interpolation (b) Cosine interpolation

Figure 4.3: Interpolation comparison

Implementation

The algorithm is reasonably fast, being linear ($O(n)$) in the size of the input sequence, but has quite expensive per-pixel operations that can make it very slow on large image sizes. A Java implementation is shown in listings 1, 2 and 3.

Listing 1 Java source code for noise image generation algorithm

```java
public static final int IMG_SIZE = 1 << 10; // 1024
public static final int CELL_SIZE = 12;
private double grid2ImageUnits;

public BufferedImage generateImage(String sequence) {
    BufferedImage img = new BufferedImage(
        IMG_SIZE, IMG_SIZE, BufferedImage.TYPE_INT_RGB);
    double[][][] grid = makegrid(sequence);
    grid2ImageUnits = (IMG_SIZE - 1) / ((grid.length - 1) * 1.0);
        for (int y = 0; y < grid.length - 1; y++)
            for (int x = 0; x < grid.length - 1; x++)
                expand(x, y, grid, img);
    return img;
}
```

4.1.4 Chaos Game Representation algorithm

The second algorithm developed in this thesis is based on the Chaos Game Representation of DNA sequences, as explained in chapter 2.

The process of generating a CGR of a DNA sequence is illustrated in figure 4.4. We begin with a 2-dimensional space where each corner is labeled with one of the four nucleotides, and the space is divided into four equal quadrants.

Then we iterate through the sequence, and for each nucleotide we create a point halfway between the previous point and the corner labeled with the same letter as the current nucleotide. Observe that each point falls within the borders of the current nucleotide's quadrant, but near the quadrant of the previous nucleotide. This creates a pattern for repeated subsequences which becomes visible for longer sequences. Equation 4.7 shows the recursive mathematical definition for generating the points, where n is the sequence length and s_i is the ith symbol of the sequence.

$$P(x_i, y_i) = \sum_{j=0}^{n} \frac{f(s_i) - P(x_{i-1}, y_{i-1})}{2}$$

$$\textit{where} \qquad f(z) = \begin{cases} (0,0) & \text{if } z = A \\ (1,0) & \text{if } z = C \\ (0,1) & \text{if } z = T \\ (1,1) & \text{if } z = G \end{cases} \tag{4.7}$$

The last step is to further divide the space into a number of bins. The number of points that falls into each bin is counted and used to construct a grayscale heatmap, where darker shades indicate a higher concentration of points. This is illustrated in figure 4.5. As a result very long sequences does not make the image completely black, but instead produces a gradient effect.

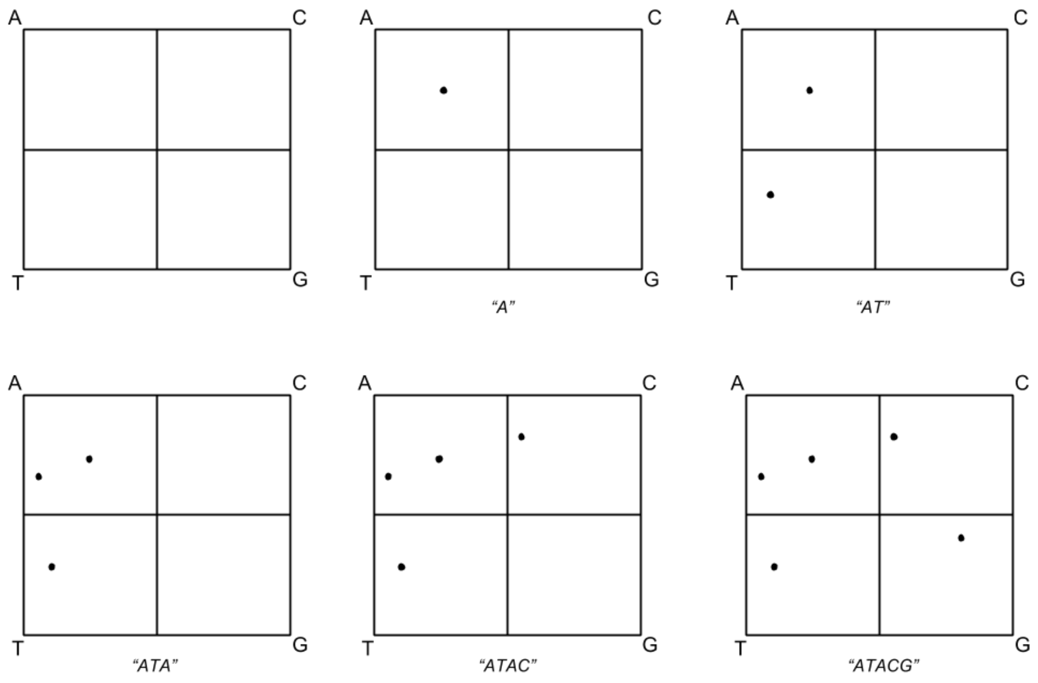


Figure 4.4: CGR of DNA sequences

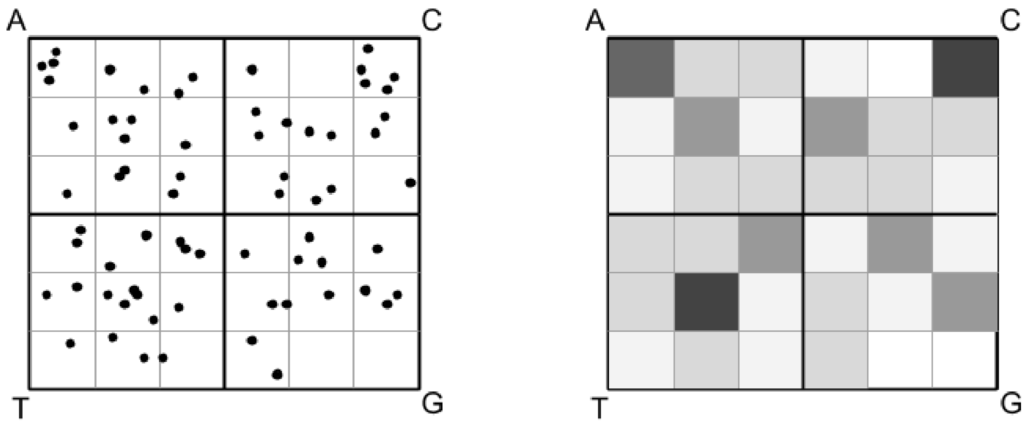


Figure 4.5: Heatmap contruction from CGR

Image resolution and number of bins

Significant variation in sequence length creates a problem for this method. Long sequences need a high resolution to avoid appearing as just noise, and shorter sequences fail leave any visible mark unless the resolution is low. In addition, the image size needs to be the same for all sequences for the system to compare them accurately.

Using a 2-dimensional array of bins one can simulate a lower resolution with a higher resolution image. Figure 4.6 illustrates the effect of different bin sizes on the same sequence. Notice how the pattern disappears in the lowest resolution image (4.6a), and is barely visible at the highest resolution (4.6d).

There are two clear ways to handle this: The number of bins could be variable, i.e. a function of the length of the sequence, or one could try to find a resolution that would work best with the most sequence lengths. We found that a $128x128$ number of bins (like in figure 4.6b) seemed to work well with the data set, and so chose to use this number. The image resolution was set to a fixed $512x512$.

Implementation

The image generation algorithm described above is very fast, and has low memory requirements. Generating the heatmap is linear in sequence length ($O(n)$), and producing the image itself is just an iteration over each pixel. The source code for the Java implementation of the algorithm is shown in listings 4 and 5.

4.2 Indexing and retrieval system

This thesis relies on using existing state of the art CBIR solutions for indexing and retrieval of generated images.

4.2.1 Software

A prototype system was developed for testing and evaluating the approach described in this chapter. It was written in Java using the OpenJDK[1] 7 open source implementation. Along with the Java Standard Library, the following additional libraries were used: LIRe[2] 0.9.5 for CBIR functionalities which also bundles Apache Lucene[3] 4.10.2, and JOpt Simple[4] for parsing command line arguments.

4.2.2 Architecture and usage

The architecture for this system is shown in figure 4.7, including arrows showing the different control paths for indexing and retrieval. It closely resembles the architecture of any basic information retrieval system with the exception of the Image Generator module.

The system features a simple command-line interface which can be run on a headless server. Listing 6 shows the commands for the three supported tasks, namely image generation, indexing, and retrieval.

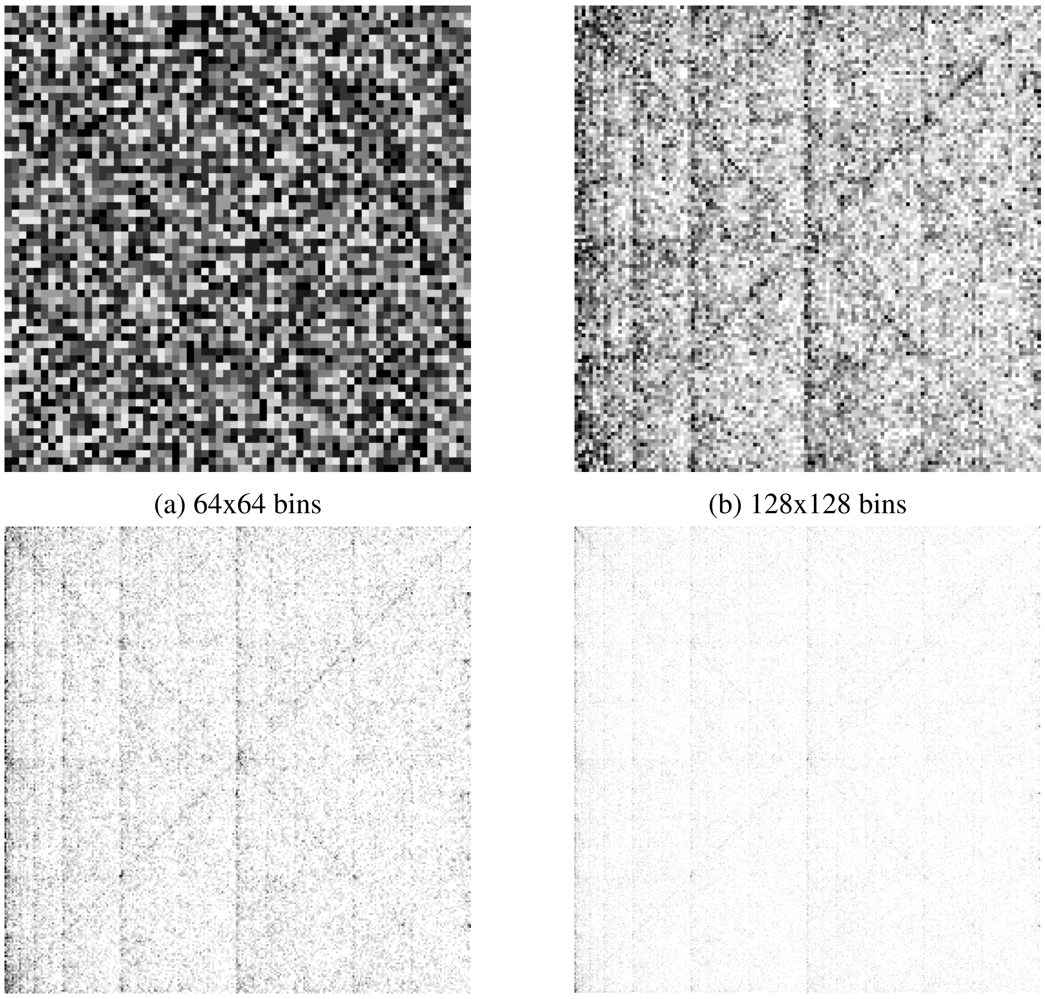
[1]http://openjdk.java.net/
[2]http://www.semanticmetadata.net/lire/
[3]http://lucene.apache.org/core/
[4]http://pholser.github.io/jopt-simple/

Listing 2 Java source code for noise image generation algorithm cont. 1

```java
private double[][][] makeGrid(String sequence, int gridLength) {
    int gridSize = (int) Math.ceil(Math.sqrt((double)
     ↪  sequence.length() / CELL_SIZE));
    double[][][] grid = new double[gridSize][gridSize][3];
    double d = 1.0 / CELL_SIZE;
    int i = 0, j = 0;
    for (int k = 0; k < sequence.length(); k++) {
        char base = sequence.charAt(k);
        switch (base) {
            case 'A': grid[i][j][0] += d; break;
            case 'C': grid[i][j][1] += d; break;
            case 'T': grid[i][j][1] += d; break;
            case 'G':
                grid[i][j][0] += d;
                grid[i][j][1] += d;
                break;
            default: continue;
        }

        if ((k+1) % CELL_SIZE == 0 && k < sequence.length() - 1) {
            j++;
            if (j >= gridSize) {
                j = 0;
                i++;
            }
        }
    }
    return grid;
}
```

Listing 3 Java source code for noise image generation algorithm cont. 2

```java
private void expand(int x0, int y0, double[][][] grid,
 ↪  BufferedImage img) {
    double f = 0.0, g = 0.0;
    double gradientFactor = 1.0 / grid2ImageUnits;
    double[] rgb = new double[3];
    for (int y = 0; y < grid2ImageUnits; y++) {
        int y1 = y0 + 1;
        for (int x = 0; x < grid2ImageUnits; x++) {
            int x1 = x0 + 1;

            double r1 = lerp(grid[y0][x0][0], grid[y0][x1][0], f);
            double g1 = lerp(grid[y0][x0][1], grid[y0][x1][1], f);
            double b1 = lerp(grid[y0][x0][2], grid[y0][x1][2], f);
            double r2 = lerp(grid[y1][x0][0], grid[y1][x1][0], f);
            double g2 = lerp(grid[y1][x0][1], grid[y1][x1][1], f);
            double b2 = lerp(grid[y1][x0][2], grid[y1][x1][2], f);
            rgb[0] = lerp(r1, r2, g);
            rgb[1] = lerp(g1, g2, g);
            rgb[2] = lerp(b1, b2, g);

            img.setRGB(
                (int) Math.ceil(x0 * grid2ImageUnits + x),
                (int) Math.ceil(y0 * grid2ImageUnits + y),
                invertBrightness(doubleToIntRGB(rgb)));

            f += gradientFactor;
            if (f >= 1.0) f = 0.0;
        }
        g += gradientFactor;
        if (g >= 1.0) g = 0.0;
        f = 0.0;
    }
}
```


(a) 64x64 bins

(b) 128x128 bins

(c) 256x256 bins

(d) 512x512 bins

Figure 4.6: Result of different bin sizes for a sequence 50,145 nucleotides long

Listing 4 Java source code for CGR image generation algorithm

```java
public static final int IMG_SIZE = 1 << 9; // 512
public static final int N_BINS = 128;
public static final int STEP = IMG_SIZE / N_BINS;

public BufferedImage generateImage(String sequence) {
    BufferedImage image = new BufferedImage(IMG_SIZE, IMG_SIZE,
     ↪  BufferedImage.TYPE_INT_RGB);
    int[][] bins = createBinsFromSequence(sequence);
    for (int y = 0; y < N_BINS; y++) {
        for (int x = 0; x < N_BINS; x++) {
            double gray = 1.0 - bins[y][x] * 0.1;
            if (gray < 0.0)
                decr = 0.0;
            int rgb = doubleToIntRGB(gray)
            for (int y0 = 0; y0 < STEP; y0++)
                for (int x0 = 0; x0 < STEP; x0++)
                    img.setRGB(x * STEP + x0, y * STEP + y0, rgb);
        }
    }
}
```

Listing 5 Java source code for CGR image generation algorithm cont.

```java
private int[][] createBins(String sequence) {
    int[][] bins = new int[N_BINS][N_BINS];
    int xA = 0, yA = 0;
    int xC = IMG_SIZE - 1, yC = 0;
    int xT = 0, yT = IMG_SIZE - 1;
    int xG = IMG_SIZE - 1, yG = IMG_SIZE - 1;
    double center = (IMG_SIZE - 1) / 2;
    double currentX = center, currentY = center;

    for (int i = 0; i < sequence.length(); i++) {
        char base = sequence.charAt(i);
        double dX, dY;
        switch (base) {
            case 'A':
                dx = (xA - x) / 2;
                dy = (yA - y) / 2;
                break;
            case 'C':
                dx = (xC - x) / 2;
                dy = (yC - y) / 2;
                break;
            case 'T':
                dx = (xT - x) / 2;
                dy = (yT - y) / 2;
                break;
            case 'G':
                dx = (xG - x) / 2;
                dy = (yG - y) / 2;
                break;
            default: continue; // Skip all N's
        }
        currentX += dx;
        currentY += dy;
        bins[((int) (y / STEP))][((int) (x / STEP))]++;
    }
}
```

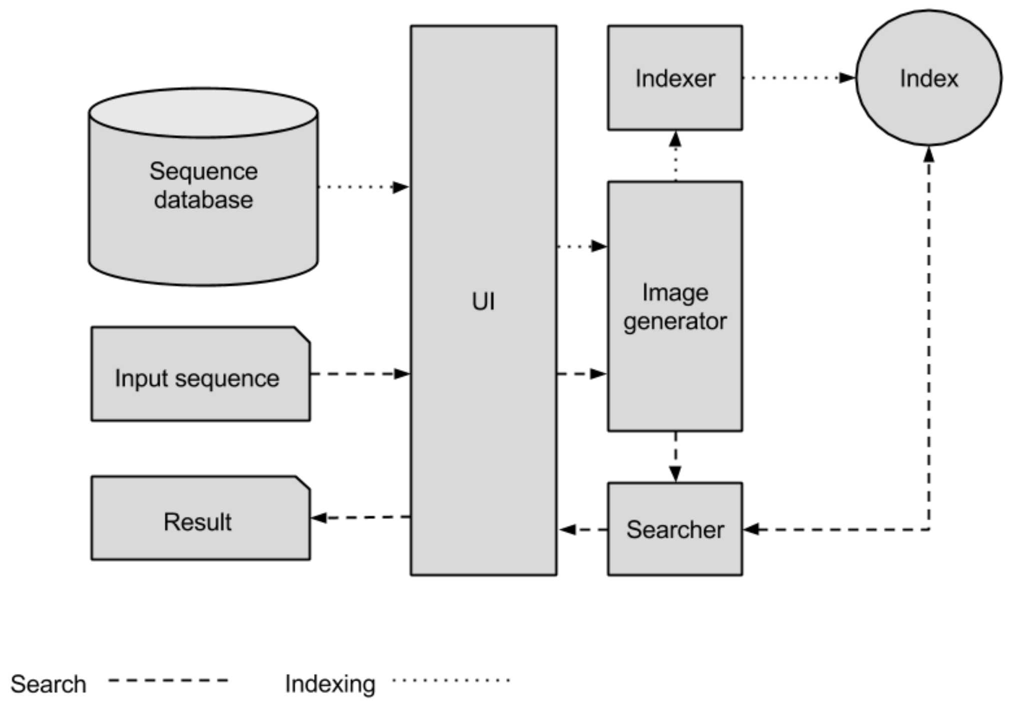


Figure 4.7: Block diagram of the system

Listing 6 Usage of prototype system

```
# Generate images from FASTA-file in output directory
$ java -jar <jarfile> generate <db fasta-file> <output-dir>

# Create index from image directory in output directory
$ java -jar <jarfile> index <image-dir> <output-dir>

# Search for the query in with index
$ java -jar <jarfile> search <query fasta-file> <index-dir>
```

5. Evaluation

In this chapter we present an evaluation of the approach described in the previous chapter. This includes selection of the data set, presentation and selection of retrieval metric(s), and how the evaluation itself was performed and the results.

5.1 The data-set

The data-set we selected for use in the evaluation was the Oryza Sativa Indica Group Whole Shotgun Genome Sequence Set from The European Nucleotide Archive [39], consisting of 410,675,192 basepairs among 53,325 sequences. The main motivation for using this data-set was that it had been previously used for the same purpose in [2], wherein it is described as being a particular challenge due to high variation in sequence length and therefore particularly useful for evaluating the system. This also applies to this thesis as we are tackling the same problem.

5.1.1 Ground truth

Information retrieval systems are normally evaluated using specialized test collections. These collections supply a set of queries and a *ground truth* for each query, which in the case of image collections would be the set of images known to have visual similarity to the query image or relevance in the case of a textual query. In the case of DNA sequence data, there exists no such collection with an established ground truth to use as reference. Therefore we will be using BLAST as the gold standard reference system and adopt the results from BLAST searches as ground truth. This method is flawed however, since BLAST is based on heuristics and not guaranteed to return optimal results for a query. The implications of this will be discussed in the next chapter.

5.2 Evaluation metrics

5.2.1 Precision and recall

Precision and recall are the two classic metrics used for evaluating information retrieval systems. They are defined by the following equations, where A is a subset of the total document collection known to be relevant for the query, and B is the subset of documents retrieved by the query.

$$Precision = \frac{|A \cap B|}{|B|} \tag{5.1}$$

$$Recall = \frac{|A \cap B|}{|A|} \tag{5.2}$$

Simply put, precision is a measure of the *exactness/accuracy* of the retrieved result while recall is a measure of the result's *completeness*. They have a somewhat inverse relationship, meaning that there is usually a trade-off between the two, although theoretically perfect precision and recall are not mutually exclusive. In real world systems one is often more important than the other. A common example is standard web search, where users rarely browse beyond the first page of results. Higher precision is therefore a desired quality while good recall is less important, if not impossible due to the quantity of data on the web.

F-measure

Precision and recall are not very informative in isolation, and are often combined into a single measure such as F-measure, which takes the harmonic mean between the two. The following equation shows the formula for weighted F-measure, where β is a positive real number and P and R denotes precision and recall. $\beta = 0.5$ and $\beta = 2$ are two common values which weights in favor of precision and recall respectively.

$$F_\beta = (1+\beta^2) * \frac{P * R}{(\beta^2 * P) + R} \tag{5.3}$$

P@n

In the case of ranked retrieval it is possible to measure the precision taking only the top ranked results into account. This measure is called *Precision at n*, commonly abbreviated P@n, and can be much more informative than base precision in evaluating such systems as low-ranked false positives no longer negatively affect the precision score. In addition this helps soften the relationship between precision and recall since the number of retrieved documents no longer affects the precision as much, provided the relevant documents are generally ranked higher.

Limitations

Although sufficient for simple classification tasks, precision and recall are not considered appropriate measures for evaluating ranked retrieval systems simply because they do not take the rankings into account. This is also true for P@n which only somewhat mediates the problem. They do, however, serve as important building blocks from which more complicated evaluation metrics are constructed.

5.2.2 Mean Average Precision, MAP

Mean Average Precision is the most commonly used metric for evaluating ranked information retrieval systems. It provides a single number for evaluation of a set of queries, making it highly useful in evaluating system performance.

It is based on the Average Precision metric, which is the average of the P@n values at each rank of n that has a relevant document. The MAP of a set of queries is simply the arithmetic mean of the average precision for each query, as shown in the equation below where Q is the set of queries being evaluated and $AVGP(q)$ is the average precision measured for the query q at each relevant result.

$$MAP(Q) = \frac{\sum_{q \in Q} AVGP(q)}{|Q|} \tag{5.4}$$

5.2.3 Average Normalized Modified Retrieval Rank, ANMRR

ANMRR was proposed in the MPEG-7 standard especially for evaluating CBIR systems, and is used as the evaluation criterion for all their color core experiments [40]. It is based on the Average Retrieval Rate, which is a measure of how many images of the ground truth set is retrieved among the top k retrievals relative to the size of the ground truth set for the query. Naturally this introduces a bias toward queries with smaller ground truth sets, which is a problem when evaluating several queries with varying sizes of ground truth sets. To eliminate this problem ANMRR introduces several steps of normalization to produce what they call the *Normalized Modified Retrieval Rank (NMRR)* before the average is computed.

A detailed explanation of the metric can be found in [40].

5.2.4 Mean Normalized Retrieval Order, MNRO

A very recently proposed evaluation metric, MNRO, is another metric specifically designed for CBIR systems that includes both precision and recall in a single metric. Its main advantage is that it is also accurate for scaled up versions of the system, unlike MAP and ANMRR which does not take into account the size of the image database and the effect this would have on retrieval performance.

Mean Normalized Retrieval order is explained thoroughly in [41].

5.2.5 Selecting a retrieval metric

In order to properly select an appropriate evaluation metric for a system, one needs to understand the system to be evaluated. In this case it is a CBIR system, although it is highly specialized and the desired retrieval properties are not the same as a standard CBIR system. The question is whether the evaluation would benefit from using a metric designed for evaluating CBIR systems, such as ANMRR or MNRO, over the tried and true Mean Average Precision.

Since the evaluation results in this thesis are unlikely to ever be compared with any other CBIR system, we chose to use Mean Average Precision as it is both much simpler than the other two and more likely to be used by other sequence similarity search systems, making future comparisons easier.

5.3 The sample sets

We decided to use two sample sets of different sizes for the evaluation, one with 100 samples and one with 1000. The smaller one would be used for evaluating the different feature descriptors and the larger for the evaluation of the approach as a whole. Both sample sets were chosen randomly from the entire data-set. Figure 5.1 shows that the distribution of sequence lengths among the two sample sets and the data-set is similar, and the samples are therefore representative of the data-set as a whole.

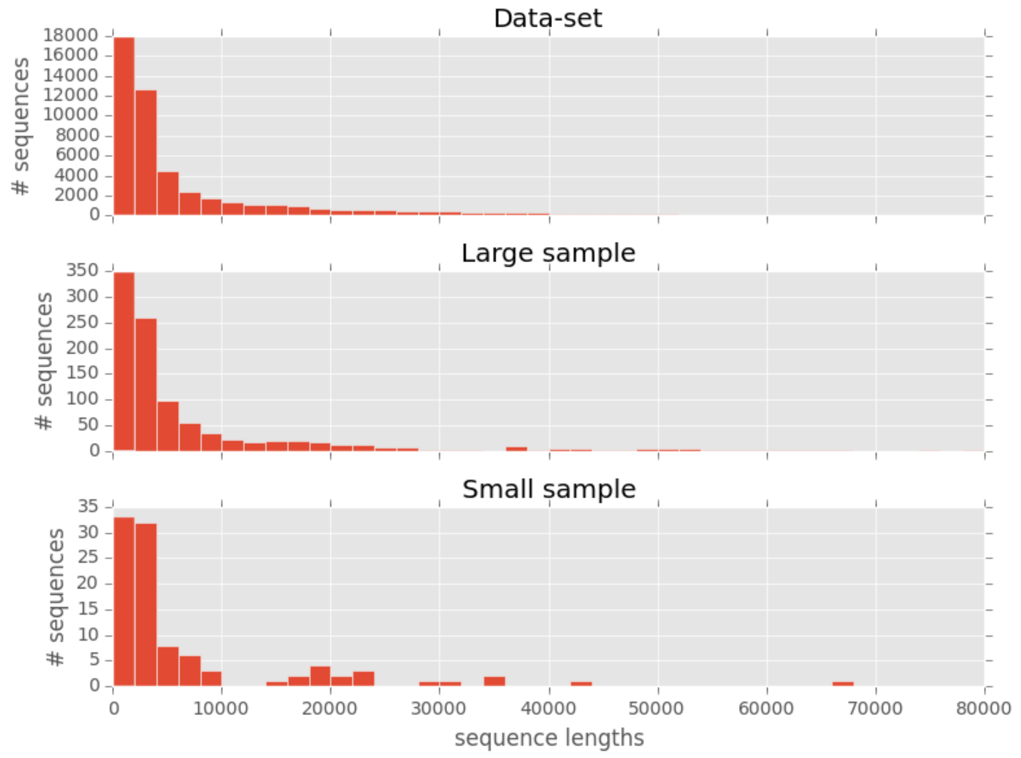


Figure 5.1: Distribution of sequence lengths

Next a ground truth set for each of the sample sequences was established using the result from a BLAST search using the sample as query. We decided to limit the sizes of the ground truth sets to focus only on the most relevant hits. This was done by filtering the ground truth sets to include only the top 20 highest scoring alignments. If two or more alignments had the exact same score we included them all, but counted them as only one entry toward the 20 entries limit.

It makes sense to focus only on the most relevant hits since we are using a binary relevance metric, i.e. each item is either relevant or non-relevant with no degrees of relevancy. Moreover, the similarity models in sequence alignment and CBIR are so different that it is safe to assume that lower scoring alignments reported by BLAST will more often than not be viewed as dissimilar by the approach in this thesis.

The command used for the BLAST searches is shown in figure 5.2. The version of BLAST used was 2.2.30+

```
$ blastn -db <db-name> -query <fasta-file> -outfmt 10
```

Figure 5.2: BLAST command

5.4 Results

5.4.1 Feature descriptors

First we needed to determine which feature descriptors worked best for the images generated by both approaches. We excluded any descriptors not implemented in LIRe, but since LIRe supports most state-of-the-art descriptors this was considered as only a minor downside. The other important criterion was index size and computation speed.

The Value Noise algorithm was designed for composite descriptors, taking into account both color and texture- or shape information. We selected CEDD, FCTH and JCD for further evaluation of this approach. The CGR algorithm produces grayscale images, and so descriptors storing color information would essentially have wasted space. Based on this we selected two descriptors intended for grayscale images to use in further evaluation: Luminance Layout and Gabor.

Index size

In order to compare the descriptors, we created three different indexes of the dataset using each of the descriptors for one index. The resulting index sizes can be seen in table 5.1.

Value Noise Surprisingly, FCTH produces the smallest index followed by CEDD and JCD. CEDD is the smallest descriptor of the three and should therefore produce a smaller index than FCTH. This result is most likely due to and optimization in the feature vector serialization implementation in the LIRe library, where any trailing zeroes in the feature vector are removed before writing the index. Since the color extraction is the same this indicates that the FCTH texture extraction algorithm produces the trailing zeroes.

Table 5.1: Descriptor index sizes

Image algorithm	Descriptor	Index size
Value Noise	CEDD	4.5 MB
	FCTH	2.7 MB
	JCD	6.2 MB
CGR	Luminence Layout	3.2 MB
	Gabor	15 MB

Table 5.2: Descriptor computation cost

Image algorithm	Descriptor	Total millis	Avg millis
Value Noise	CEDD	105591	105
	FCTH	112786	112
	JCD	223812	223
CGR	Luminence Layout	18942	18
	Gabor	62606	62

Chaos Game Representation Here we see that the Gabor descriptor index is more than four times bigger than the Luminance Layout index, which is odd considering the Gabor feature vector should not be that much larger. Looking at the source code for the LIRe implementation of that Gabor feature extraction reveals that the cause is poor byte serialization.

Computation cost

An important factor is the computation cost, which will add up considerably when indexing large data-sets, and slow down the systems response time for each query. The computation cost was measured by computing the feature vectors for 1000 images, once for each descriptor. The results are listed in table 5.2.

Value Noise The computation speed for CEDD and FCTH are roughly equal with CEDD being only slightly faster. We can assume that the total computation cost will grow approximately linearly with the number of images, so if we consider the average computation cost per image for our entire data-set of 53,325 images, the total difference between them in computation time is 6.2 minutes, which is not very significant. JCD takes twice as long as the other two, which is longer than expected since it only needs to perform an additional texture extraction compared to the other two. A closer look at the LIRe source code reveals that the JCD implementation first computes both the CEDD and FCTH histograms and merges them afterwards. This means that the same color information is extracted twice, which is highly inefficient. An optimized version would probably only take about 60 % longer than CEDD or FCTH to compute.

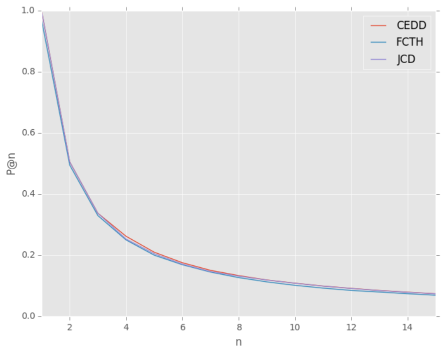
Chaos Game Representation Both feature descriptors are very fast to compute, but the Luminance Layout descriptor is still way faster as it only requires scaling the images and no other costly operations.

Retrieval performance

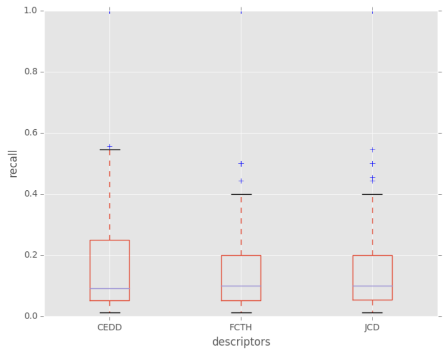
Next we wanted to compare the retrieval performance of the different descriptors for images produced by the algorithms in this thesis. Compared to real world images the images generated from DNA sequences in this thesis all have a substantial degree of similarity, enough that we can make a reasonable assumption that the retrieval performance differences between the descriptors will hold for any data set.

For this test we used the smaller sample set with 100 sequences. We searched for each sequence once for each descriptor, for both the Value Noise- and CGR algorithms. From the results we then computed the average P@n values for $n = 1$ through 15, and the average recall values across each query for all the descriptors. The results for the Value Noise generated images are shown in figure 5.3 and the results for the CGR generated images are shown in figure 5.4.

Looking at the results for the Value Noise algorithm the P@n values are nearly identical for all 15 values of n with CEDD having slightly higher values, especially at $n = 4$. For the recall values we can see that the median of CEDD is slightly lower than FCTH and JCD, but CEDD has a slighly higher distribution of values than the other two. Based on these results we selected to use the CEDD descriptor in the next evaluation. For the CGR algorithm the results are quite clearly in favor of the Luminance Layout Desctiptor which outperforms the Gabor descriptor in both precision and recall, so naturally we selected the Luminance Layout descriptor for use in further evaluation.

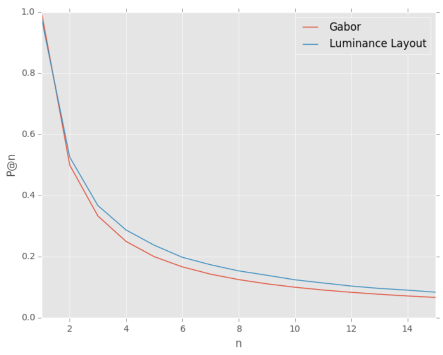


(a) Average P@n plot

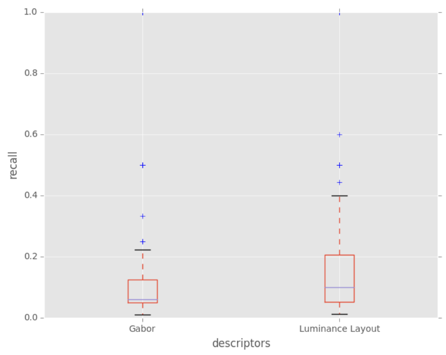


(b) Average recall box plot

Figure 5.3: Plots for descriptor retrieval performance test for Value Noise algorithm



(a) Average P@n plot



(b) Average recall box plot

Figure 5.4: Plots of descriptor retrieval performance test for CGR algorithm

5.4.2 System evaluation

After determining the best performing feature vectors we needed to perform a larger scale evaluation of my two approaches and compare the results to the results from [2], as one of the stated goals of this thesis is to improve upon the earlier work done in that paper. We opted to test against the best performing algorithm they present, which is the simple RGBA algorithm with 700 % upscaling. We also implemented their frequency-based algorithm, but my conclusion is that it is simply too slow to be practical for any significant sequence length.

Mean Average Precision and recall

For this test we used the larger sample set consisting of 1000 samples queries. The largest ground truth set had 272 entries so we set the maximum number of hits returned by LIRe to be 300. We also set a distance threshold to 20.0 to remove any low scoring hits from the results. Table 5.3 shows the results. As can be clearly seen there is little to no difference between the algorithms both in MAP and recall, although the Value Noise- and CGR algorithms both have a minuscule edge in average recall, and the CGR algorithm falls slightly behind in MAP. The CGR algorithm also return the maximum number of 300 hits for all queries, which is more than the other two.

Table 5.3: MAP and recall values

Image algorithm	**MAP**	**Avg recall**	**Avg # hits**
Value Noise	0.179	0.220	261.812
CGR	0.175	0.222	300
RGBA upscaled	0.179	0.216	257.643

6. Conclusion

6.1 Discussion

6.1.1 Reliability of results

As previously stated, the results presented in this chapter are derived using BLAST as a reference. Therefore they do not give a reliable general view of the retrieval performance of the approach but rather how well it performs relative to BLAST. However the lack of real-world test sets for DNA data has led to BLAST often being used as ground truth or *gold standard* in research proposing alternative algorithms for sequence similarity search [2, 42]. Considering BLAST's role in bioinformatics as the de facto standard for sequence similarity search, we argue that the results are still valuable as a measure of general retrieval performance with the drawbacks in mind.

6.1.2 Interpretation of results

If we look at table 5.3, the results unequivocally show an overall low retrieval performance relative to BLAST. All three algorithms return a large number of hits scoring under the distance threshold, but the recall values show that they are mostly non-relevant hits. Looking at the recall values, and assuming most ground truth sets are about 20 entries in size since this was set as the cut-off number for the samples, we can estimate that the average number of relevant hits returned for each query is only between 4 and 5. MAP takes ranking of results into account which considering the low score indicates that several non-relevant hits also rank higher than the few relevant hits returned. This is also supported by the steep initial decline of the P@n plots in figures 5.3 and 5.4 for the smaller sample set. These also show that the algorithms that the precision at $n = 1$ is 1.0 for all instances, which surprisingly is not always true for BLAST. This is a slight indication that the approach could be better or equal to BLAST for retrieving identical or near identical sequences.

6.1.3 Limitations of approach

There are two different factors we could attribute the results to, namely limitations in the image generation algorithms or limitations in the CBIR techniques used.

Since we are comparing whole images we are in essence doing something more akin to global alignment instead of local alignment, which is what BLAST performs. The idea behind having normalized image sizes was to make the comparisons less penalized by highly different sequence lengths. This seems to yield a slight gain in the number of hits returned over the RGBA upscaled algorithm, which has varying image sizes dependent on sequence length, but they seem to be mostly non-relevant hits. Therefore despite the normalized image sizes, both of the algorithms in this thesis along with the RGBA upscaled algorithm, highly prefer sequences of similar lengths, which is a significant limitation considering the intended use case.

The translation step between sequence and image introduces a level of ”fuzziness” to the system. This is an unavoidable trade-off for the increased speed and smaller indexes, but can be made smaller by having a good algorithm. The Chaos Game Representation algorithm has the least fuzziness of the algorithms evaluated, it being a bijective mapping from the nucleotide alphabet. The second translation from image to feature space introduces yet another layer of fuzziness which depends entirely on the feature descriptor. From the results it seems that even with an algorithm that introduces little fuzziness, the second translation introduces so much that it evens out.

We have shown the reduced index size achieved by using CBIR techniques compared to BLAST, but indexing speed is another matter. Taking into account the preprocessing step of converting sequences into images the time it takes to index a large number of sequences is significantly higher. Although not required, storing the images on disk could be desirable, e.g. to eliminate the need for this step should re-indexing be necessary. This could require several gigabytes of disk space depending on the number of images and bit depth per image.

6.2 Summary

In this thesis we have developed two different algorithms for translating DNA sequences into images. We have indexed the resulting images using state of the art software and techniques from Content-Based Image Retrieval and created a prototype system for sequence similarity search. The system has been evaluated using BLAST as a gold standard reference.

6.3 Main conclusion

Although the results show very little improvement over the previous work done with this particular technique, there is still definite promise in using CBIR techniques with sequence similarity search. The reduced index sizes and search times could make up for the reduced sensitivity in certain cases, especially when searching for highly similar sequences. With even better image generation algorithms and more advanced CBIR techniques we might be able to rival the retrieval performance of BLAST in the future, though we are not quite there yet.

6.4 Future work

A lot of work could still be done to further release the potential of the ideas demonstrated in this thesis and its predecessor. One example would be to develop image generation algorithms that better captures the underlying structure of the DNA sequences. Many of the techniques from the study of alignment-free sequence analysis could be worth exploring in this regard.

Another interesting topic would be the development of specialized feature descriptors, as most of today's state of the art descriptors are quite general and designed to work on many different types of images. Images produced by transforming DNA sequences will have certain characteristics and similarities depending on the algorithm that might be exploitable for creating better descriptors. Introducing other techniques from information retrieval and machine learning such as clustering, relevance feedback, and training sets could potentially be used as well to improve results.

Sub-image matching is another feature that should be explored in order to facilitate matching shorter sequences with longer sequences such as in local alignment.

Bibliography

[1] Guy Cochrane et al. “Facing growth in the European Nucleotide Archive”. In: *Nucleic Acids Research* 41.D1 (2013), pp. 30–35.

[2] Heri Ramampiaro and Aleksander Grande. “DNA Sequence Search Using Content-Based Image Search Approach”. In: *5th International Conference on Practical . . .* (2011), pp. 191–199.

[3] F. H. C. Watson, J. D.; Crick. *Molecular Structure of Nucleic Acids*. 1953.

[4] S B Needleman and C D Wunsch. “A general method applicable to the search for similarities in the amino acid sequence of two proteins.” In: *Journal of molecular biology* 48 (1970), pp. 443–453.

[5] Angana Chakraborty and Sanghamitra Bandyopadhyay. “FOGSAA: Fast Optimal Global Sequence Alignment Algorithm. Suplementary Material”. In: *Scientific reports* 3 (2013), p. 1746.

[6] T.F. Smith; and M S Waterman. “Identification of Common Molecular Subsequences”. In: *J. Mol. Biol.* 147 (1981), pp. 195–197.

[7] Wing-kai Hon Tak-wah Lam Wing-kin Sung. “Practical Aspects of Compressed Suffix Arrays and FM-index in Searching DNA Sequences”. In: *Proceedings of the Sixth Workshop on Algorithm Engineering and Experiments and the First Workshop on Analytic Algorithms and Combinatorics* (2004), pp. 31–38.

[8] Francisco Claude et al. “Compressed q-gram indexing for highly repetitive biological sequences”. In: *10th IEEE International Conference on Bioinformatics and Bioengineering 2010, BIBE 2010* (2010), pp. 86–91.

[9] Ben Langmead et al. “Ultrafast and memory-efficient alignment of short DNA sequences to the human genome.” In: *Genome biology* 10.3 (Jan. 2009), R25.

[10] T. W. Lam et al. “Compressed indexing and local alignment of DNA”. In: *Bioinformatics* 24.6 (2008), pp. 791–797.

[11] Min-soo Kim et al. “n-Gram / 2L : A Space and Time Efficient Two-Level n-Gram Inverted Index Structure”. In: *Science And Technology* (2005), pp. 1–16.

[12] Susana Vinga and Jonas Almeida. “Alignment-free sequence comparison - A review”. In: *Bioinformatics* 19.4 (2003), pp. 513–523.

[13] António M Costa, José T Machado, and Maria D Quelhas. “Histogram-based DNA analysis for the visualization of chromosome, genome and species information.” In: *Bioinformatics (Oxford, England)* 27.9 (May 2011), pp. 1207–14.

[14] D. Anastassiou. “Frequency-domain analysis of biomolecular sequences”. In: *Bioinformatics* 16.12 (Dec. 2000), pp. 1073–1081.

[15] Anca Bucur et al. “Alignment method for spectrograms of DNA sequences.” In: *IEEE transactions on information technology in biomedicine : a publication of the IEEE Engineering in Medicine and Biology Society* 14.1 (Jan. 2010), pp. 3–9.

[16] Nevenka Dimitrova, YH Cheung, and Michael Zhang. “Analysis and visualization of DNA spectrograms: open possibilities for the genome research”. In: *Proceedings of the 14th annual . . .* (2006), pp. 1017–1024.

[17] M Altaiski, O Mornev, and R Polozov. “Wavelet analysis of DNA sequences.” In: *Genetic analysis : biomolecular engineering* 12.5-6 (Mar. 1996), pp. 165–8.

[18] J a Tenreiro Machado, António C Costa, and Maria Dulce Quelhas. “Wavelet analysis of human DNA.” In: *Genomics* 98.3 (Sept. 2011), pp. 155–63.

[19] H J Jeffrey. “Chaos game representation of gene structure.” In: *Nucleic acids research* 18.8 (1990), pp. 2163–2170.

[20] Jonas S Almeida. “Sequence analysis by iterated maps, a review.” In: *Briefings in bioinformatics* 15.3 (2014), pp. 369–75.

[21] J S Almeida et al. “Analysis of genomic sequences by Chaos Game Representation.” In: *Bioinformatics (Oxford, England)* 17.5 (2001), pp. 429–437.

[22] Ritendra Datta, Jia Li, and James Z. Wang. “Content-Based Image Retrieval: Approaches and Trends of the New Age”. In: *Proceedings of the 7th ACM SIGMM international workshop on Multimedia informatioan retrieval (MIR)* (2005), pp. 253–262.

[23] Henning Müller et al. “A review of content-based image retrieval systems in medical applications-clinical benefits and future directions.” In: *International journal of medical informatics* 73 (2004), pp. 1–23.

[24] Safinaz Mustapha and Hamid a. Jalab. “Compact composite descriptors for content based image retrieval”. In: *Proceedings - 2012 International Conference on Advanced Computer Science Applications and Technologies, ACSAT 2012* (2013), pp. 37–42.

[25] Ceyhun Burak Akgül et al. “Content-based image retrieval in radiology: Current status and future directions”. In: *Journal of Digital Imaging* 24.2 (2011), pp. 208–222.

[26] Savvas a. Chatzichristofis and Yiannis S. Boutalis. “Content based radiology image retrieval using a fuzzy rule based scalable composite descriptor”. In: *Multimedia Tools and Applications* 46.2-3 (2009), pp. 493–519.

[27] Ken Perlin. “An image synthesizer”. In: *ACM SIGGRAPH Computer Graphics* 19.3 (1985), pp. 287–296.

[28] Ken Perlin. “Chapter 2 Noise Hardware”. In: *SIGGRAPH Course Notes* (2001).

[29] Steven Worley. “A cellular texture basis function”. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH ’96* (1996), pp. 291–294.

[30] W R Pearson and D J Lipman. “Improved tools for biological sequence comparison.” In: *Proceedings of the National Academy of Sciences of the United States of America* 85.April (1988), pp. 2444–2448.

[31] S F Altschul et al. “Basic local alignment search tool.” In: *Journal of molecular biology* 215.3 (Oct. 1990), pp. 403–10.

[32] Andreas Prlić et al. “BioJava: An open-source framework for bioinformatics in 2012”. In: *Bioinformatics* 28.20 (2012), pp. 2693–2695.

[33] Thomas Sikora. “The MPEG-7 visual standard for content description - An overview”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 11.6 (2001), pp. 696–702.

[34] Mathias Lux. “LIRE: open source image retrieval in Java”. In: *Proceedings of the 21st ACM MM* (2013), pp. 843–846.

[35] Mathias Lux and Sa Chatzichristofis. “Lire: lucene image retrieval: an extensible java CBIR library”. In: *Proceeding of the 16th ACM international . . .* (2008), pp. 1–3.

[36] Savvas a. Chatzichristofis and Yiannis S. Boutalis. “CEDD: Color and edge directivity descriptor: A compact descriptor for image indexing and retrieval”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5008 LNCS (2008), pp. 312–322.

[37] Savvas a. Chatzichristofis and Yiannis S. Boutalis. “FCTH: Fuzzy Color and texture histogram a low level feature for accurate image retrieval”. In: *WIAMIS 2008 - Proceedings of the 9th International Workshop on Image Analysis for Multimedia Interactive Services* (2008), pp. 191–196.

[38] S Chatzichristofis, Y Boutalis, and Mathias Lux. “Selection of the proper compact composite descriptor for improving content based image retrieval”. In: *Proc. of the 6th IASTED International Conference*. Vol. 134643. 2009, p. 064.

[39] Rasko Leinonen et al. “The European Nucleotide Archive.” In: *Nucleic acids research* 39.Database issue (Jan. 2011), pp. D28–31.

[40] B S Manjunath et al. “Color and texture descriptors”. In: 11.6 (2001), pp. 703–715.

[41] Savvas a. Chatzichristofis et al. “Mean Normalized Retrieval Order (MNRO): A new content-based image retrieval performance measure”. In: *Multimedia Tools and Applications* 70 (2014), pp. 1767–1798.

[42] H. Hauswedell, J. Singer, and K. Reinert. “Lambda: the local aligner for massive biological data”. In: *Bioinformatics* 30.17 (2014), pp. i349–i355.

A. Code Listings

Listing 7 DNA sequence translator interface

```java
public interface DNAToImageTranslator {

    public BufferedImage generateImage(String sequence);
}
```

Listing 8 Static utility functions

```
public class Util {

    public static int doubleToRGB(double d) {
        return ((((int)(d * 255 + 0.5)) & 0xFF) << 16) |
                (((int)(d * 255 + 0.5)) & 0xFF) << 8)  |
                (((int) (d * 255 + 0.5)) & 0xFF);
    }

    public static int doubleToRGB(double[] rgb) {
        return ((((int)(rgb[0] * 255 + 0.5)) & 0xFF) << 16) |
                ((((int)(rgb[1] * 255 + 0.5)) & 0xFF) << 8)  |
                (((int) (rgb[2] * 255 + 0.5)) & 0xFF);
    }

    public static void saveImage(BufferedImage image, String path)
     ↪ {
        try {
            ImageIO.write(image, "png", new File(path));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Listing 9 Indexing implementation class

```java
public class Indexer {

    private File indexDir;
    private File imgDir;
    private Class<? extends LireFeature> descriptor;

    private Indexer(File indexDir, File imgDir) {
        this.indexDir = indexDir;
        this.imgDir = imgDir;
    }

    public Indexer(File indexDir, File imgDir, Class<? extends LireFeature> desriptor) {
        this(indexDir, imgDir);
        this.descriptor = desriptor;
    }

    public void start(int numThreads) {

        if (!indexDir.exists() && !indexDir.mkdir())
            return;
        else if (!imgDir.exists())
            return;
        else if (numThreads <= 0)
            numThreads = 1;

        ParallelIndexer parallelIndexer = new ParallelIndexer(
                numThreads, indexDir.getAbsolutePath(), imgDir.getAbsolutePath()) {
            @Override
            public void addBuilders(ChainedDocumentBuilder builder) {
                builder.addBuilder(new GenericDocumentBuilder(descriptor));
            }
        };
        parallelIndexer.run();
    }
}
```

Listing 10 Search implementation class

```
public class Searcher {

    private File indexDir;
    private Class<? extends LireFeature> descriptor;
    private int maxHits;
    private IndexReader indexReader;
    private ImageSearcher imageSearcher;

    public Searcher(File indexDir, Class<? extends LireFeature>
     ↪  descriptor, int maxHits, boolean indexInRAM) {
        this.indexDir = indexDir;
        this.descriptor = descriptor;
        this.maxHits = maxHits;
        readIndex(indexInRAM);
    }

    private void readIndex(boolean indexInRAM) {
        if (!indexDir.exists() || !indexDir.isDirectory())
            return;

        try {
            Directory directory = FSDirectory.open(indexDir);
            if (indexInRAM) {
                indexReader = DirectoryReader.open(new
                 ↪  RAMDirectory(directory, IOContext.READ));
            } else {
                indexReader = DirectoryReader.open(directory);
            }
            imageSearcher = new GenericFastImageSearcher(maxHits,
             ↪  descriptor);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```java

    public HashMap<String, Float> search(BufferedImage img) throws
     ↪  IOException {

        if (indexReader == null || imageSearcher == null)
            return null;

        HashMap<String, Float> hits = new HashMap<String,
         ↪  Float>(maxHits);

        ImageSearchHits searchHits = imageSearcher.search(img,
         ↪  indexReader);
        for (int i = 0; i < searchHits.length(); i++) {
            String fileName = searchHits.doc(i)
                .getValues(
                        DocumentBuilder.FIELD_NAME_IDENTIFIER)[0];
            hits.put(fileName, searchHits.score(i));
        }

        return hits;
    }
}
```