UNIVERSITY OF TARTU

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Institute of Computer Science

Computer Science Curriculum

Gerli Viikmaa

Detection of Near-Duplicates Using Error-Correcting Codes

Bachelor's Thesis (6 ECTS)

Supervisor: Sven Laur, PhD

Tartu 2014

Detection of Near-Duplicates Using Error-Correcting Codes

Abstract:

The detection of near-duplicate items from a large set is a problem faced in many fields. This paper constructs and analyses two algorithms for finding similar pairs from an input dataset. It shows that these algorithms are applicable and efficient in the domain of DNA sequences.

Keywords:

Duplicate detection, coding theory, algorithm design

Sarnaste elementide tuvastamine veaparanduskoodide abil

Kokkuvõte:

Sarnaste elementide tuvastamine suurest hulgast on probleem, mida esineb erinevates valdkondades. See töö konstrueerib ja analüüsib kahte algoritmi sisendhulgast sarnaste paaride leidmiseks. Näidatakse, et need algoritmid on sarnaste DNA järjestuste leidmiseks rakendatavad ja efektiivsed.

Võtmesõnad:

Sarnaste elementide tuvastus, kodeerimisteooria, algoritmide loomine

Contents

1 Introduction

As a motivating example consider the following problem: we have a large set of nucleotide sequences, which are 36 base pairs (letters) long. The number of sequences is in the order of magnitude 10^6. We have determined that there are very similar sequences in this set due to errors occurring during the dataset generation. More specifically, when the sequences are read by a machine, it sometimes mistakes one letter for another. These small errors clutter our dataset and we would like to fix them. The structure of the data has led us to believe that any two sequences with up to 3 differences in base pairs should actually be considered the same sequence. The problem stems from the fact that our dataset is so large that we cannot compare each sequence to another in reasonable time. We need some algorithm that finds the pairs of near-duplicates more efficiently.

This problem, known as near-duplicate detection, has been explored previously in the context of compressing sequence databases [2, 3]. However, our sequences are rather short and the restrictions on the types of errors allowed in the data are different, rendering those approaches inapplicable to our case.

Near-duplicate detection is also relevant in applications such as indexing web pages [8, 9], compressing information found in natural language texts such as tweets [6], image and video search on the web [7, 13], plagiarism and copyright infringement detection [7]. This topic is also relevant in the field of cryptography, with example applications in biometric authentication [5] and passwords that allow spelling errors [1].

This thesis explores the solution to this problem using coding theory as this approach has not been explored in mainstream literature.

This paper is structured in the following way. In Section 2, explanations are given for relevant concepts in coding theory (Section 2.1) and linear algebra (Section 2.2). Our problem is described in more detail in Section 2.3. The problem is formalized and a general solution is given is Section 3. Two different variants of the solution are analysed in Sections 3.1 and 3.2. The work is summarized and additional topics for research are provided in Section 4, followed by references. Appendix A contains some implementations of the methods described in this thesis. Appendix B contains a licence to allow reproduction of this thesis.

2 Background

This section aims to explain necessary concepts in coding theory, provides a few useful results from linear algebra related to finite fields and describes the problem dataset mathematically

2.1 Coding theory

Coding theory is a study of codes. The main terms as used in this field are explained here. The following mathematical definitions are based on [10].

Definition 1. An (n, M) *code* over a finite alphabet Σ is a subset $\mathcal{C}$ of size M of the space Σ^n. Each element of $\mathcal{C}$ is called a *codeword*. The parameter n is called the *code length* and the parameter M is the *code size*. Code size can also be expressed as $|\mathcal{C}|$.

Let us consider words in some alphabet that all have the same length. A code, in this context, is a set of words (known as codewords) that are used to represent a set of pre-agreed messages. The process of retrieving the corresponding codeword for a message is known as encoding. Each codeword corresponds to exactly one message.

Codewords can become corrupted, resulting in a new word. We are interested in the type of errors where some letters in the new word are different from the letters in the same positions in the codeword. More formally, we can give the following definition.

Definition 2. Let $\mathcal{C}$ be an (n, M) code over Σ and $\mathbf{c} \in \mathcal{C}$ a codeword. An *error* is the changing of some entry c_i in the codeword $\mathbf{c} = c_1c_2 \dots c_i \dots c_n$, resulting in a word $\mathbf{x} = c_1c_2 \dots x_i \dots c_n$ where $x_i \neq c_i$.

The number of errors can be given by the following measure:

Definition 3. The *Hamming distance* between two words $\mathbf{x}, \mathbf{y} \in \Sigma^n$ is the number of coordinates on which $\mathbf{x}$ and $\mathbf{y}$ differ.

Hamming distance between words $\mathbf{x}$ and $\mathbf{y}$ is denoted by $\mathtt{d}(\mathbf{x}, \mathbf{y})$. Hamming distance is a metric, meaning that it satisfies the following properties for every three words $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \Sigma^n$.

- Nonnegativity: $\mathsf{d}(\mathbf{x}, \mathbf{y}) \geq 0$. The distance is zero if and only if $\mathbf{x} = \mathbf{y}$.
- Symmetry: $\mathsf{d}(\mathbf{x}, \mathbf{y}) = \mathsf{d}(\mathbf{y}, \mathbf{x})$.
- The triangle inequality: $\mathsf{d}(\mathbf{x}, \mathbf{y}) \leq \mathsf{d}(\mathbf{x}, \mathbf{z}) + \mathsf{d}(\mathbf{z}, \mathbf{y})$.

The smaller the Hamming distance between two words, the closer they are to each other. A similar concept can be defined for a single word.

Definition 4. The *Hamming weight* of a word $\mathbf{x} \in \Sigma^n$ is the number of nonzero coordinates of $\mathbf{x}$.

Hamming weight of a word $\mathbf{x}$ is denoted by $\mathsf{w}(\mathbf{x})$. The previous two terms are related in the following way.

Proposition 1. *Given that Σ is an Abelian group, for every two words $\mathbf{x}, \mathbf{y} \in \Sigma^n$, the following equality holds*

$$\mathsf{d}(\mathbf{x}, \mathbf{y}) = \mathsf{w}(\mathbf{y} - \mathbf{x}).$$

Error-correcting codes are codes that are designed to withstand a number of errors occurring when a codeword gets corrupted. If the number of errors is not too large, the corrupted word can be fixed. The process of finding the closest codeword to a word is known as decoding.

Definition 5. A *nearest-codeword decoder* of an (n, M) code $\mathcal{C}$ over Σ is a function $\mathcal{D} : \Sigma^n \to \mathcal{C}$ whose value for every word $\mathbf{x} \in \Sigma^n$ is the closest codeword in $\mathcal{C}$ in terms of Hamming distance to $\mathbf{x}$. In case of a tie between two or more codewords, the first in the lexicographic ordering of the tying codewords is chosen.

Let it be noted that since each codeword corresponds to a message, by decoding a word it is also possible to reconstruct the message.

Decoders of a different type have also been studied but they are not relevant in the scope of this thesis. More information can be found in [10].

Example. A widely-used alphabet in data transmission is $\{0, 1\}$ – bits. Let us look at words in this alphabet of length 3. A possible code on such words is a repetition code where the codewords are created by repeating a bit three times:

$\mathcal{C} = \{000, 111\}$. This code can be used to transmit a single bit – 0 or 1 – by repeating it three times and sending 000 or 111 instead. Although we are adding to the size of the message (three bits instead of one), this allows us to detect and correct the flip of a single bit. Thus message transfer is more error-resistant.

For instance, if we wish to send the message 0, we encode it and transmit 000. If we were to receive 010, we can decode this back to the closest codeword 000 and successfully determine that the sent message was 0. However, if more than a single bit is flipped (e.g. we receive 011), the code will fail to determine the message - 111 is the closest codeword to 011.

Incidentally, the described code is not the most optimal code that can correct the flip of a single bit in binary string. Exist constructions for codes that are able to correct the same number of mutated bits but have less encoding overhead.

It is possible to quantify the number of mutations an error-correcting code can fix. In order to do that, it is first necessary to determine how far apart codewords of the code are.

Definition 6. The *minimum distance* of code $\mathcal{C}$ is the minimum Hamming distance between a pair of distinct codewords of $\mathcal{C}$. The distance $d_{\min}$ is given by

$$d_{\min} = \min_{\substack{\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C} \\ \mathbf{c}_1 \neq \mathbf{c}_2}} \mathsf{d}(\mathbf{c}_1, \mathbf{c}_2)$$

An (n, M) code with minimum distance $d_{\min}$ is called an $(n, M, d_{\min})$ code.

As a corollary, all pairs of distinct codewords are at Hamming distance at least $d_{\min}$ from each other.

Proposition 2. *Let $\mathcal{C}$ be an $(n, M, d_{\min})$ code over Σ. The nearest-codeword decoder $\mathcal{D} : \Sigma^n \to \mathcal{C}$ recovers correctly every pattern of up to $\left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$ errors.*

In the repetition code example above, the minimum distance of the code was 3, hence the code's ability to detect the flip of one bit.

Let us give an another interpretation of the above proposition. This requires us to introduce a new term and elaborate on its properties.

Definition 7. Let Σ be an alphabet and r be a nonnegative integer. The set of all words in Σ^n at Hamming distance r or less from a word $\mathbf{x}$ is called a *sphere* of radius r centered at $\mathbf{x}$. The number of words in a sphere is called the *volume* of a sphere.

Proposition 3. *If the size of the alphabet is q and words have length n, then the volume of a sphere of radius r is given as*

$$V_q(n,r) = \sum_{i=0}^{r} \binom{n}{i} (q-1)^i$$

Proof. Let's look at a sphere centered at $\mathbf{x} = x_1 x_2 \dots x_n$. All the words $\mathbf{y} = y_1 y_2 \dots y_n$ at distance i from $\mathbf{x}$ have the structure where there are exactly i coordinates where $\mathbf{x}$ and $\mathbf{y}$ differ. There are

$$\binom{n}{i}$$

different sets of those i coordinates. In each position j where $x_j \neq y_j$ there are $q-1$ different letters of the alphabet – all but x_j – that could be at position j in the word $\mathbf{y}$. Thus the number of words exactly at distance i from $\mathbf{x}$ is

$$\binom{n}{i} (q-1)^i.$$

Since all the words in a sphere are at an integer distance between 0 and r from the centre of the sphere, the total number of words in a sphere is

$$V_q(n,r) = \sum_{i=0}^{r} \binom{n}{i} (q-1)^i.$$

□

Proposition 2 can be interpreted geometrically in the following way. Spheres with radius $r = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$ that are centered at a distinct pair of codewords in $\mathcal{C}$ are disjoint. This is illustrated on Figure 1. The distance between the two codewords $\mathbf{c}_1$ and $\mathbf{c}_2$ is at least $d_{\min}$. Thus, if $\mathbf{x} \in \Sigma^n$ is a word contained in the sphere of radius r centered at codeword $\mathbf{c}_1$, the nearest-codeword decoder $\mathcal{D}$ will return $\mathbf{c}_1$

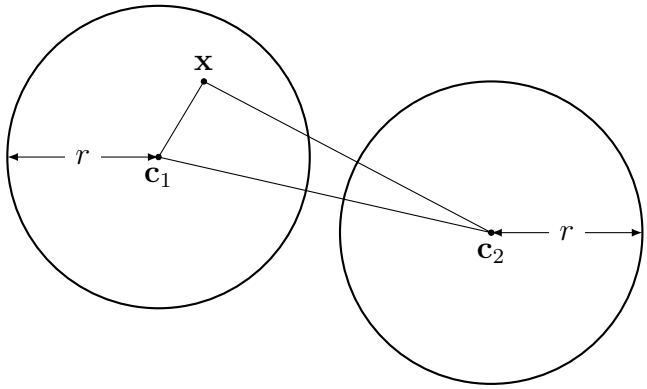

Figure 1: Two spheres with diameter $r = \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor$ (depicted as circles) centered at codewords $\mathbf{c}_1$ and $\mathbf{c}_2$.

when applied to $\mathbf{x}$.

If we center a sphere at each codeword, the total number of words contained in the spheres is bounded by the following.

Proposition 4. *For any $(n, M, d_{\min})$ code over an alphabet Σ of size q, the following inequality holds (*sphere packing bound*):*

$$M \cdot V_q\left(n, \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor\right) \leq q^n$$

Here the right-hand side is the number of all words in the space Σ^n where q is the size of the alphabet Σ. Thusly, for any code the sum of the volumes of the spheres centered at each codeword is at most the number words in the space Σ^n.

Definition 8. A code $\mathcal{C}$ that attains the sphere-packing bound is called *perfect*.

As follows, perfect codes are codes such that the spheres with radius r centered at each codeword cover the space Σ^n fully. This means that the code has the following pleasant properties. First, there is no word $\mathbf{x}$ in the space Σ^n such that

$$\forall \mathbf{c} \in \mathcal{C} : \mathsf{d}(\mathbf{x}, \mathbf{c}) > \left\lfloor \frac{d_{\min}-1}{2} \right\rfloor.$$

Second, there are no ties in decoding. Finally, these spheres divide the entire space Σ^n into equal-sized parts.

Unfortunately, it has been proven that only a few constructions of perfect codes exist. Additionally, the constructions are rather limiting in terms of their parameters n, q and $d_{\min}$.

Although a perfect code for an arbitrary set of parameters $n, q, d_{\min}$ may not exist, making the assumption that it does enables us to simplify the treatment in this cursory feasibility scan.

Generally speaking, exist various constructions for codes, most of which are not perfect. Examples of famous constructions are parity codes, repetition codes, Hamming codes, Reed-Solomon codes. More details can be found in [10]. Many constructions exist for codes having the following property.

Definition 9. A code $\mathcal{C}$ over a finite field $\mathbb{F}_q = \Sigma$ where q is the size of the field is called *linear* if $\mathcal{C}$ is a linear subspace of the vector space $\mathbb{F}_q^n$ over $\mathbb{F}_q$, that means that for any two codewords $\mathbf{c}_1, \mathbf{c}_2 \in \mathcal{C}$ and two scalars $\alpha_1, \alpha_2 \in \mathbb{F}_q$ the condition $\alpha_1\mathbf{c}_1 + \alpha_2\mathbf{c}_2 \in \mathcal{C}$ holds.

Subsequently, if $\mathcal{C}$ is defined over a finite field and $\mathcal{D}$ is the corresponding nearest-codeword decoder, it is possible to represent each word $\mathbf{x}$ as the sum of its nearest codeword and an error word ε:

$$\mathbf{x} = \mathcal{D}(\mathbf{x}) + \varepsilon.$$

The following section gives a definition and lists some properties of finite fields.

2.2 Finite fields

Let $\mathbb{F}_q$ denote a finite field of size q. This is sometimes also denoted as $\mathrm{GF}(q)$. The letters GF stand for *Galois field*.

Definition 10. A *field* $\mathbb{F}$ is a set of elements on which addition $(+)$ and multiplication $(\times)$ are defined. The operations must satisfy the following axioms for all elements $a, b, c \in \mathbb{F}$.

- Associativity

$$(a + b) + c = a + (b + c)$$

$$(a \times b) \times c = a \times (b \times c)$$

- Commutativity

$$a + b = b + a$$

$$a \times b = b \times a$$

- Distributivity

$$a \times (b + c) = a \times b + a \times c$$

$$(a + b) \times c = a \times c + b \times c$$

- Identity

$$a + 0 = 0 + a = a$$

$$a \times 1 = 1 \times a = a$$

- Inverses

$$a + (-a) = (-a) + a = 0$$

$$a \times a^{-1} = a^{-1} \times a = 1 \text{ if } a \neq 0$$

Let only a few results concerning finite fields be listed here.

Proposition 5. *If q is a prime, the field $\mathbb{F}_q$ coincides with $\mathbb{Z}_q$, the ring of integer residues modulo q.*

Proposition 6. *The size of any finite field is the power p^n where p is a prime and n a positive integer. For any prime p and positive integer n, there exists a finite field with p^n elements.*

When working with nucleotide sequences, the alphabet we are using is $\Sigma = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$. This can be transformed to use a finite field $\mathbb{F}_4$ with any correspondence between the elements of Σ and $\mathbb{F}_4$ since we are using Hamming distance which counts differences. We would like to use a finite field in some applications because of some properties that finite fields have. One of them is the fact that since fields are Abelian groups, Proposition 1 applies to them.

The field we are going to be using for DNA sequences is $\mathbb{F}_4$ which is a field with elements $\{0, 1, \alpha, \alpha + 1\}$ and addition and multiplication defined in Tables 1 and 2 respectively. [12]

Table 1: Addition table of $\mathbb{F}_4$

$+$	$\mathbf{0}$	$\mathbf{1}$	$\boldsymbol{\alpha}$	$\boldsymbol{\alpha+1}$
$\mathbf{0}$	0	1	α	$\alpha+1$
$\mathbf{1}$	1	0	$\alpha+1$	α
$\boldsymbol{\alpha}$	α	$\alpha+1$	0	1
$\boldsymbol{\alpha+1}$	$\alpha+1$	α	1	0

Table 2: Multiplication table of $\mathbb{F}_4$

$\times$	$\mathbf{0}$	$\mathbf{1}$	$\boldsymbol{\alpha}$	$\boldsymbol{\alpha+1}$
$\mathbf{0}$	0	0	0	0
$\mathbf{1}$	0	1	α	$\alpha+1$
$\boldsymbol{\alpha}$	0	α	$\alpha+1$	1
$\boldsymbol{\alpha+1}$	0	$\alpha+1$	1	α

A relatively useful property of $\mathbb{F}_4$ is that

$$\forall x \in \mathbb{F}_4 : x + x = 0.$$

This means that whenever we wish to subtract a value, we can add it instead.

2.3 Mathematical setup of our problem

As described in the introduction, the original problem was finding very similar pairs of sequences from a relatively large set of DNA sequences. As noted in the previous section, the size of the nucleotide alphabet is $q = 4$ so we can use the finite field $\mathbb{F}_4$ as our alphabet. The data we are interested consists of sequences that has length $n = 36$. In terms of the number of errors that we wish to correct, let us look at a few different scenarios. In the milder case, let us set $d = 1$ error. This means we need to use a code such that $d_{\min} = 3$. A construction for a perfect code exists for such parameters – the Hamming code. In a stronger case, we wish to correct errors of up to $d = 3$ which means that we need to use a code where $d_{\min} = 7$. Unfortunately, no perfect code exists for this configuration. However, it is still possible to use a code with a minimal coverage, i.e., a code such that the radius needed to cover all words with spheres centered at codewords is minimal.

Furthermore, since we know that the DNA sequences encode proteins, we can translate the DNA sequences into an amino acid sequence using a codon table. In that case our sequences shorten to $n = 12$ and the alphabet has 20 letters. However, since a finite field of size 20 does not exist, we must use an alphabet that is slightly bigger. The next prime power after 20 is 23, so we can use $q = 23$ and $\Sigma = \mathbb{F}_{23}$. The number of errors we are interested in fixing here is $d = 1$. Note that this situation is not equivalent to either of the cases for DNA sequences. This is due to the fact that changes in a single nucleotide may or may not change the translated amino acid. Similarly, changes in 3 different nucleotides may result in 0 to 3 changes in the corresponding peptide sequence.

Although in the following we will give general results for any parameter set, these are the sets we are interested in benchmarking.

3 All pairwise near-duplicates

If we have a set of N words, then the brute-force solution to finding all similar pairs requires comparisons of all possible pairs which is a total of

$$\binom{N}{2} = \frac{N(N-1)}{2}$$

pairs of words.

We are looking for an algorithm to rather quickly find all the near-duplicate pairs in a set of input words.

Definition 11. The set of *near-duplicates* of an input set $X \subseteq \Sigma^n$ with respect to distance d is a set of two-word sets

$$\mathcal{N}(X, d) = \{\{\mathbf{x}, \mathbf{y}\} : \mathbf{x}, \mathbf{y} \in X, \mathtt{d}(\mathbf{x}, \mathbf{y}) \leq d\}.$$

The set $\{\mathbf{x}, \mathbf{y}\}$ is called an *unordered pair*.

Note that since $\mathtt{d}(\mathbf{x}, \mathbf{y}) = 0$ only if $\mathbf{x} = \mathbf{y}$, all near-duplicates are at distance at least 1 from each other.

The task of finding all near-duplicates of a particular input set X is a rather hard one. A simpler task is to find a superset of near-duplicates which is smaller than the set of all pairs.

Definition 12. The set of *candidate near-duplicates* of an input set $X \subseteq \Sigma^n$ with respect to distance d is the set of two-element sets

$$\mathcal{NC}(X, d) \supseteq \mathcal{N}(X, d).$$

We can then calculate the exact distances between all of the pairs in this set and thus find pairs of words that are in our distance range. Although a trivial candidate set is the set of all pairs, it is our intention to minimize the size of the candidate set, leaving us with as few comparisons as possible.

The two algorithms described in this section are both based on the same idea of hashing multiple times. The general scheme is the following. Let $\mathcal{H}$ be a set of hash functions. Pick a hash function $h \in \mathcal{H}$ and group all of the input words

in set X based on the output of this function. We will then form unordered pairs of all the words in one group. All these pairs will be added to the candidate set $\mathcal{NC}(X, d)$. We will repeat this process with all the hash functions in set $\mathcal{H}$. This process should give us a smaller set of candidates than the set of all pairs. Our task is to construct this set of hash functions.

We are interested in finding the smallest set of functions for the described algorithm such that for any input set all the near-duplicate pairs are covered but also the least amount of false near-duplicate candidates are reported. This should help us to minimize the number of necessary comparisons we will have to make to find the actual near-duplicates.

Two different hash function sets will be constructed in the following subsections. We will analyse the resulting constructions in terms of the size of the candidate set, giving both a lower and an upper bound to the set size.

3.1 Baseline method

Before describing the method based on coding theory, let us set the baseline with the following simple solution based on the idea of locality-sensitive hashing [4] which has also been used for near-duplicate detection.

Denote the set of indices of words in Σ^n with $\mathcal{I} = \{1, 2, \ldots, n\}$. The construction will use the following type of hash functions.

Definition 13. A *substring function* is a function $h_I : \Sigma^n \to \Sigma^k$ where $0 < k \leq n$ and $I = \{i_1, i_2, \ldots, i_k\} \subseteq \mathcal{I}$ is a set of indices such that

$$h_I(\mathbf{x}) = x_{i_1} x_{i_2} \ldots x_{i_k}.$$

Let X be a set of words. Let $\mathbf{x}, \mathbf{y} \in X$ be near-duplicates with respect to distance d, that is $\{\mathbf{x}, \mathbf{y}\} \in \mathcal{N}(X, d)$. Hence the two words are different in some positions $\Delta = \{i_1, i_2, \ldots, i_{d'}\}$ where $d' \leq d$. We would like to have a set of hash functions $\mathcal{H} = \{h_I : I \subseteq \mathcal{I}\}$ such that

$$\exists I \subseteq \mathcal{I} : h_I(\mathbf{x}) = h_I(\mathbf{y}).$$

This requirement ensures that for all pairs of near-duplicates $\{\mathbf{x}, \mathbf{y}\}$, at least

one of the hash functions will use a set of indices where the two words are the same and if it does, the output of the hash function will be the same.

Let there be $\ell = |\mathcal{H}|$ hash functions. To simplify the analysis, let us use a set of hash functions $\mathcal{H} = \{h_{I_1}, h_{I_2}, \ldots h_{I_\ell}\}$ such that all coordinates will be used exactly once for some hash function. This is formalized in the following two equations.

$$\bigcup_{i=1}^{\ell} I_i = \mathcal{I}$$

$$\forall i, j : i \neq j \Rightarrow I_i \cap I_j = \varnothing$$

We will consider as candidate pairs all such pairs where the output of at least one hash function was the same. For an input set X, this can be expressed more formally as

$$\mathcal{NC}(X, d) = \{\{\mathbf{x}, \mathbf{y}\} : \exists h \in \mathcal{H} : \mathbf{x}, \mathbf{y} \in X, h(\mathbf{x}) = h(\mathbf{y})\}.$$

One way of defining the functions is by splitting the words into $d+1$ consecutive substrings of length $k = \frac{n}{d+1}$ where n is the length of the words and d is the threshold of near-duplicates. For simplicity, let us assume that n is a multiple of $d+1$. According to the pigeonhole principle, if two words have d or less differences, at least one of the $d+1$ substrings will be equal. Thus the algorithm requires $\ell = d + 1$ hash functions.

The functions will be the following.

$$\begin{aligned} h_1(\mathbf{x}) &= x_1 x_2 \ldots x_k \\ h_2(\mathbf{x}) &= x_{k+1} x_{k+2} \ldots x_{2k} \\ &\vdots \\ h_\ell(\mathbf{x}) &= x_{dk+1} x_{dk+2} \ldots x_n \end{aligned}$$

Let us give some bounds on the size of the candidate near-duplicate set. If the words all come from an alphabet Σ which contains q letters, then each of these functions has q^k possible outputs. Each of these outputs has a corresponding bucket.

Definition 14. A *bucket* for hash function h is the set

$$\mathcal{B}_{\mathbf{y},h} = \{\mathbf{x} : h(\mathbf{x}) = \mathbf{y}\}.$$

In case we are not interested in any particular function h, the notation for a bucket can be shortened to $\mathcal{B}_{\mathbf{y}}$.

Each function splits the input dataset between its buckets. For each bucket $\mathcal{B}_{\mathbf{y}}$, there are q^{n-k} different inputs $\mathbf{x} \in \Sigma^n$ that will result in the same output $\mathbf{y}$. Hence, the maximal number of words that can fall into the bucket is q^{n-k}:

$$\forall \mathbf{y} \in \Sigma^k : |\mathcal{B}_{\mathbf{y}}| \leq q^{n-k}.$$

Since candidates are formed by pairing up all the elements in a bucket, the number of candidate near-duplicates for an input set X for this algorithm is

$$|\mathcal{NC}(X, d)| = \sum_{h \in \mathcal{H}} \sum_{\mathbf{y} \in X} \binom{|\mathcal{B}_{\mathbf{y},h}|}{2}.$$

For a single function each word can only fall into a single bucket, hence for an N-element input set the following condition must hold:

$$\forall h \in \mathcal{H} : \sum_{\mathbf{y} \in X} |\mathcal{B}_{\mathbf{y},h}| = N.$$

For an input set X containing N elements, the size of $\mathcal{NC}(X, d)$ is maximal when for all functions some buckets are full and the rest are empty.

The number of full buckets for a single function is then

$$\frac{N}{q^{n-k}}.$$

All of these will contribute $\binom{q^{n-k}}{2}$ unordered pairs to the candidate set and there are $d + 1$ functions. This can be summarized by the following lemma.

Lemma 1. *Let $X \subseteq \Sigma^n$ be an N-element input set. The maximal number of candidate near-duplicates with respect to distance d arising in the baseline scheme*

	$\lvert\mathcal{NC}_{\min}(X,1)\rvert$	$\lvert\mathcal{NC}_{\max}(X,1)\rvert$	$\lvert\mathcal{NC}_{\min}(X,3)\rvert$	$\lvert\mathcal{NC}_{\max}(X,3)\rvert$
DNA	$\left(\frac{N}{6.9\cdot 10^{10}} - 1\right) N$	$6.9 \cdot 10^{10} N$	$\left(\frac{N}{1.3\cdot 10^{5}} - 2\right) N$	$3.6 \cdot 10^{16} N$
Peptide	$\left(\frac{N}{6.4\cdot 10^{7}} - 1\right) N$	$6.4 \cdot 10^{7} N$	-	-

Table 3: Bounds to sizes of candidate near-duplicate sets for N-element input set X for three different scenarios. The numbers for peptides are given with $q = 20$.

is

$$|\mathcal{NC}_{\max}(X,d)| = (d+1) \cdot \frac{N}{q^{n-k}} \cdot \binom{q^{n-k}}{2} = \frac{d+1}{2}\left(q^{n-k} - 1\right) N$$

where q is the size of the alphabet Σ and $k = n/(d+1)$.

Next, let us provide a lower bound to the size of the candidate set. The size of $\mathcal{NC}(X,d)$ is minimized when hashing spreads the words uniformly over all the buckets, resulting in as few words as possible in a single bucket. As a reminder, a single function has q^k different buckets. For simplicity let us assume that N is divisible by q^k. Then the number of words in a single bucket is

$$\frac{N}{q^k}$$

which will contribute

$$\binom{\frac{N}{q^k}}{2}$$

pairs to the candidate set. Since there are $d+1$ hash functions, the total number of candidate pairs is given by the following lemma.

Lemma 2. *Let $X \subseteq \Sigma^n$ be an N-element input set. The minimal number of candidate near-duplicates with respect to distance d arising in the baseline scheme is*

$$|\mathcal{NC}_{\min}(X,d)| = (d+1) \cdot q^k \cdot \binom{\frac{N}{q^k}}{2} = \frac{d+1}{2} N \left(\frac{N}{q^k} - 1\right)$$

where q is the size of the alphabet Σ and $k = n/(d+1)$.

For any set of input sequences X, $|\mathcal{NC}_{\min}(X,d)| \leq |\mathcal{NC}(X,d)| \leq |\mathcal{NC}_{\max}(X,d)|$.

The sizes of $\mathcal{NC}_{\min}$ and $\mathcal{NC}_{\max}$ have been tabulated in Table 3 for the scenarios described in Section 2.3. The bounds give a rather wide range for the actual size

of $\mathcal{NC}$. We can see that in the best case the number of comparisons we will have to make is a lot smaller than in the brute-force solution. This is promising, since under the assumption that words in the input dataset are sampled randomly, the lower bound is much more likely to occur. The upper bound provided here is very improbable in practice.

The prototype implementation of this scheme in Sage can be seen in Appendix A.1.

3.2 Method based on error-correcting codes

This section provides the second construction for a set of hash functions. Let $\Sigma = \mathbb{F}_q$ be a finite field and $\mathcal{C} \subseteq \mathbb{F}_q^n$ an $(n, M, d_{\min})$ code. Let the threshold for near-duplicates be d such that

$$d = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor > 0.$$

Let $\mathcal{D}$ be a nearest-codeword decoder of this code. Let us assume that the code is perfect.

Let us construct a set of hash functions $\mathcal{H}$. Let the first hash function in this set be the decoding function $\mathcal{D}$. The buckets will now correspond to codewords, since the decoding function assigns the nearest codeword to any word in the space Σ^n. All buckets will have the form

$$\mathcal{B}_{\mathbf{c}} = \{\mathbf{x} : \mathcal{D}(\mathbf{x}) = \mathbf{c}\}.$$

Since the code we are using is perfect, all of the words in bucket $\mathcal{B}_{\mathbf{c}}$ will be in the sphere centered at $\mathbf{c}$. Since all the words in a sphere are relatively close together, let the set $\mathcal{NC}(X, d)$ be populated by forming unordered pairs of all the words in a single bucket.

However, this one hash function may not yet have found all the pairs at distance up to d - there might be pairs that fell into different spheres at hashing. See Figure 2 for an illustration. So we need to shift all the codewords to cover the rest of the candidate near-duplicate pairs.

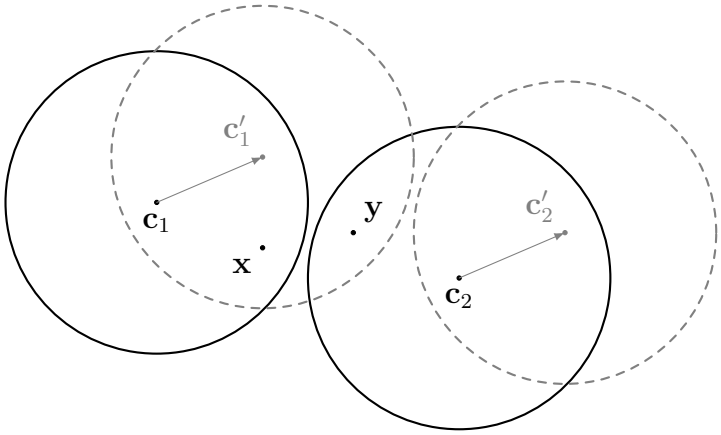

Figure 2: A pair of near-duplicate words that decode to different codewords are decoded to the same codeword in a shifted code.

Definition 15. Given an $(n, M, d_{\min})$ code $\mathcal{C}$ over a finite field $\mathbb{F}_q$, the *shifting* of this code by a vector $\varepsilon \in \mathbb{F}_q^n$ is the operation of creating a new code $\mathcal{C}_\varepsilon$ such that

$$\mathcal{C}_\varepsilon = \{\mathbf{c} + \varepsilon : \mathbf{c} \in \mathcal{C}\}.$$

Let it be noted that even if $\mathcal{C}$ is a linear code, the resulting shifted code is not linear in the case

$$\forall \mathbf{c} \in \mathcal{C} : \mathbf{c} + \varepsilon \neq \mathbf{0}.$$

Additionally, in the case of $\varepsilon = \mathbf{0}$, the shifted code is equal to the original:

$$\mathcal{C}_\mathbf{0} = \mathcal{C}.$$

Let us denote the nearest-codeword decoder of $\mathcal{C}_\varepsilon$ by $\mathcal{D}_\varepsilon$. Since $\mathcal{C}$ is defined over a field, shifting codewords by a vector ε is equivalent to shifting all words by a vector $-\varepsilon$. Thus we can construct decoding of the shifted code as

$$\mathcal{D}_\varepsilon(\mathbf{x}) = \mathcal{D}(\mathbf{x} - \varepsilon).$$

Let X be an arbitrary input set. The task of constructing the set of hash functions can be reworded into finding a set of shifts $\mathcal{E}$ such that each near-duplicate

pair is decoded into the same codeword for at least one code $\mathcal{C}_\varepsilon$:

$$\forall\{\mathbf{x},\mathbf{y}\} \in \mathcal{N}(X,d) : \exists\varepsilon \in \mathcal{E} : \mathcal{D}_\varepsilon(\mathbf{x}) = \mathcal{D}_\varepsilon(\mathbf{y}).$$

On the one hand, the more shifts there are, the more candidate pairs we will get. On the other hand, there need to be enough shifts so that all near-duplicate pairs will always be caught by some shifting. In the following, we will first give a lower bound to the number of shifts in set $\mathcal{E}$. Assuming that pairs of near-duplicates contributed to the near-duplicate set are disjoint between distinct shifts, the lower bound will have the form

$$|\mathcal{E}_{\min}| = \frac{|\mathcal{N}(\Sigma^n,d)|}{|\mathcal{C}|\ |\mathcal{N}_s(\Sigma^n,d)|}$$

where $|\mathcal{N}(\Sigma^n,d)|$ in this context is the number of all near-duplicates in Σ^n, $|\mathcal{C}|$ is the number of codewords in code $\mathcal{C}$ and $|\mathcal{N}_s(\Sigma^n,d)|$ the number of near-duplicates in a sphere with radius d.

Let us prove the following lemma.

Lemma 3. *In the space Σ^n the number of near-duplicates with respect to distance d is*

$$|\mathcal{N}(\Sigma^n,d)| = \frac{q^n}{2} \cdot \sum_{i=1}^{d} \binom{n}{i}(q-1)^i$$

where q is the size of the alphabet Σ.

Proof. Fix an arbitrary word $\mathbf{x}$ from Σ^n. The number of words at Hamming distance i from word $\mathbf{x}$ is

$$\binom{n}{i}(q-1)^i.$$

This is due to the fact that any word $\mathbf{y}$ at distance i from $\mathbf{x}$ differs from $\mathbf{x}$ at exactly i positions out of n.

Since near-duplicates are at distance 1 to d from each other, the number of near-duplicates containing $\mathbf{x}$ is

$$\sum_{i=1}^{d} \binom{n}{i}(q-1)^i.$$

Since there are q^n words in Σ^n and the pairs we are counting are unordered, the total number of near-duplicates is

$$|\mathcal{N}(\Sigma^n, d)| = \frac{q^n}{2} \cdot \sum_{i=1}^{d} \binom{n}{i}(q-1)^i.$$

□

Next, let us find the number of near-duplicate pairs in a sphere. We could simply estimate it from above with the total number of pairs we can form of all the words within a sphere. This estimate is, sadly, quite far off and we would like to have a stricter bound. In order to do that, let us prove the following lemmas.

Lemma 4. *In a full sphere with radius d the number of near-duplicates $|\mathcal{N}_s(\Sigma^n, d)|$ with respect to distance 1 is*

$$|\mathcal{N}_s(\Sigma^n, 1)| = \binom{n}{1}(q-1) + \binom{n}{1}\binom{q-1}{2} = \binom{n}{1}\binom{q}{2}.$$

Lemma 5. *In a full sphere with radius d the number of near-duplicates $|\mathcal{N}_s(\Sigma^n, d)|$ with respect to distance 2 is*

$$|\mathcal{N}_s(\Sigma^n, 2)| = \binom{n}{1}\binom{q}{2} + \frac{1}{2}\binom{n}{2}(q-1)^2(2n(q-1) + q^2 - 2q + 4).$$

Proof. The near-duplicate pairs $\{\mathbf{x}, \mathbf{y}\}$ with threshold d inside a sphere of radius d centered at vector $\mathbf{c}$ can be split into the following disjoint groups, each of which can be easily counted. Let us accompany the following list with an example in $\mathbb{F}_4$ where $\mathbf{c} = (0, 0, 0, 0, 0)$. Assuming that $d = 2$, these are the groups.

(a) Pairs containing the vector $\mathbf{c}$ (for example $\mathbf{x} = \mathbf{c} = (0, 0, 0, 0, 0)$ and $\mathbf{y} = (1, 0, 0, 0, 0)$).

(b) Pairs where both vectors are at distance a ($0 < a \leq d$) from the vector $\mathbf{c}$ and are at distance b ($0 < b \leq d$) from each other (for example $\mathbf{x} = (1, 0, 0, 0, 0)$ and $\mathbf{y} = (\alpha, 0, 0, 0, 0)$ for $a = 1$, $b = 1$).

(c) Pairs where one vector is at distance a_1 ($0 < a_1 \leq d$) from $\mathbf{c}$, the other at distance a_2 ($0 < a_2 \leq d$) from $\mathbf{c}$ where $a_1 \neq a_2$ and are at distance b

$(0 < b \leq d)$ from each other (for example $\mathbf{x} = (1, 0, 0, 0, 0)$ and $\mathbf{y} = (1, 1, 0, 0, 0)$ for $a_1 = 1, a_2 = 2, b = 1$).

Let us find all these numbers. The number of pairs in group (a) is simply

$$V_q(n, d) - 1 = \sum_{i=1}^{d} \binom{n}{i} (q-1)^i. \tag{1}$$

If we consider a codeword $\mathbf{c}$, all the points in the sphere centered at $\mathbf{c}$ are at distance up to d from $\mathbf{c}$. The only word at distance 0 is the codeword itself. Thus each element in the sphere (except for $\mathbf{c}$) forms a near-duplicate pair with $\mathbf{c}$.

The number of pairs in group (b) is

$$\frac{1}{2}\binom{n}{a}(q-1)^a \sum_{k=0}^{\lfloor \frac{b}{2} \rfloor} \binom{a}{k}\binom{n-a}{k}(q-1)^k \cdot \binom{a-k}{b-2k}(q-2)^{b-2k} \tag{2}$$

given that we have defined the binomial coefficient in such a way that

$$\text{if } m > n, \text{ then } \binom{n}{m} = 0.$$

This formula stems from the following components. First, there are $\binom{n}{a}(q-1)^a$ words at distance a from the codeword $\mathbf{c}$. Let $\mathbf{x}$ be an arbitrary word from Σ^n. All words $\mathbf{y} \in \Sigma^n$ differ from $\mathbf{x}$ in a combination of the following two ways.

(1) At a position i where $x_i \neq c_i$, y_i is such that $y_i \neq x_i$ and $y_i \neq c_i$. Thus $\mathbf{x}$ and $\mathbf{y}$ are at the same distance from $\mathbf{c}$ and there is one difference between $\mathbf{x}$ and $\mathbf{y}$.

(2) At a position i where $x_i \neq c_i$, $y_i = c_i$; for another position $j \neq i$ where $x_j = c_j$, let $y_j \neq c_j$. Thus again the distance from $\mathbf{c}$ is the same but in this scenario there are two differences between $\mathbf{x}$ and $\mathbf{y}$.

If the difference between $\mathbf{x}$ and $\mathbf{y}$ is b, there can be 0 to $\lfloor b/2 \rfloor$ differences of type (2) since each of this type of difference adds 2 to $\mathsf{d}(\mathbf{x}, \mathbf{y})$. Let us denote the number of this kind of differences by k. Thus the number of differences of type (1) is $b - 2k$.

If there are k differences of type (2), there are $\binom{a}{k}$ different combinations of k indexes i such that $x_i \neq c_i$. At each of these positions c_i must be equal to

y_i. Additionally, there are $\binom{n-a}{k}$ different combinations of k indexes j such that $x_j = c_j$. At each of these positions in $\mathbf{y}$ there can be anything but c_j. As such, there are $(q-1)^k$ different ways that those positions could be filled in $\mathbf{y}$.

As there were k differences of type (2), there need to be $b-2k$ differences of type (1) for the distance between $\mathbf{x}$ and $\mathbf{y}$ to be b. There are $a-k$ positions i in $\mathbf{y}$ such that they are not different in terms of a type (2) difference and $x_i \neq c_i$. Hence, there are $\binom{a-k}{b-2k}$ different ways of choosing $b-2k$ such positions in $\mathbf{y}$. At each of these positions there can be a value that is different from both x_i and c_i, so there are a total of $(q-2)^{b-2k}$ ways those positions could be filled in $\mathbf{y}$.

Addionally, the number of differences of type (2) can range from 0 to $\lfloor b/2 \rfloor$, each of which corresponds to a different type of pair of $\mathbf{x}$ and $\mathbf{y}$. Finally, since we are interested in the number of unordered pairs and in the previous manner we counted each such pair twice - once for $\mathbf{x}$ and once for $\mathbf{y}$, the total number of pairs as described in (b) is given in equation (2).

If $d = 1$, we can split the pairs in a sphere into only two groups: (a) and (b) – the third group is empty. Thus, if we replace $d = 1$ in equation (1) and the only combination of parameters $a = 1, b = 1$ in (2), we get the sum in Lemma 4.

If $d > 1$, we need to calculate the number of pairs in group (c) as well. Since we are looking to find the number of unordered pairs, w.l.o.g. we can only look at cases where $a_1 < a_2$. Let $\mathbf{x}$ be the word at distance a_1 and $\mathbf{y}$ the word at distance a_2 from the codeword. Since in this case one word is farther from the codeword than the other, we need to introduce a new type of difference.

(3) At a position i where $x_i = c_i$, y_i is such that $y_i \neq x_i$. Thus $\mathbf{y}$ is farther away by one difference from $\mathbf{c}$ than $\mathbf{x}$ and there is one difference between $\mathbf{x}$ and $\mathbf{y}$.

For the pairs in this group, there have to be $a_2 - a_1$ differences of this type. The rest $b-(a_2-a_1)$ differences are formed similarly to the previous group of differences of types (1) and (2). Thus, the number of pairs in group (c) is given by the formula

$$\begin{aligned} &\binom{n}{a_1}(q-1)^{a_1} \cdot \binom{n-a_1}{a_2-a_1}(q-1)^{a_2-a_1} \cdot \\ &\cdot \sum_{k=0}^{\left\lfloor \frac{b'}{2} \right\rfloor} \binom{a_1}{k}\binom{n-a_2}{k}(q-1)^k \cdot \binom{a_1-k}{b'-2k}(q-2)^{b'-2k} \end{aligned} \tag{3}$$

where $b' = b - (a_2 - a_1)$.

As these groups cover all possibilities of near-duplicate pairs to exist within a sphere, to get the total number of near-duplicates, we need to add up the numbers of each group as given in equations (1), (2), (3) with all possible combinations of distance parameters a and b. □

Example. With $q = 4, n = 5, d = 1$ we can use the $[5, 3, 3]$ Hamming code (see [10] for the construction). For that we know that the lower bound is

$$\frac{|\mathcal{N}(\mathbb{F}_4^5, 1)|}{|\mathcal{C}|\ |\mathcal{N}_s(\mathbb{F}_4^5, 1)|} = \frac{7680}{64 \cdot 30} = 4$$

but empirically we have managed to find only sets of shifts with size 7. A possible set of shifts that can be used as $\mathcal{H}$ is

$$\begin{array}{lll} (0,0,0,0,0), & (1,0,0,0,0), & (\alpha,0,0,0,0), \\ (\alpha+1,0,0,0,0), & (0,1,0,0,0), & (0,0,0,1,0), \\ (0,0,0,0,1). & & \end{array}$$

See Appendix A.3 for the usage of these shifts in the context of the code method.

The question we were also trying answer is what is the number of candidate near-duplicates that this scheme will provide us with? For each shift the candidates will be formed by pairs of words in the same bucket. Given an input set X and a set of shifts $\mathcal{E}$, the number of candidates is then

$$|\mathcal{NC}(X, d)| = \sum_{\varepsilon \in \mathcal{E}} \sum_{\mathbf{c} \in \mathcal{C}_\varepsilon} \binom{|\mathcal{B}_{\mathbf{c}, \mathcal{D}_\varepsilon}|}{2}.$$

Similarly to the baseline, we would like to give upper and lower bounds to the size of the candidate set.

First, let us give the upper bound for the size of $\mathcal{NC}(X, d)$ for an input set X containing N words. Similarly to the baseline, the size of $\mathcal{NC}(X, d)$ is maximal when some of the buckets are full and the rest are empty.

Since the buckets are spherical and one sphere can hold $V_q(n, d)$ elements, the number of full spheres is

$$\frac{N}{V_q(n, d)}.$$

Between the elements in a sphere we will have to make

$$\binom{V_q(n,d)}{2}$$

comparisons. If the number of necessary shifts is $|\mathcal{E}|$, the total number of candidate pairs is given by the following lemma.

Lemma 6. *Let $X \subseteq \Sigma^n$ be an N-element input set. The maximal number of candidate near-duplicates with respect to distance d arising in the code scheme is*

$$|\mathcal{NC}_{\max}(X,d)| = |\mathcal{E}|\frac{N}{V_q(n,d)}\binom{V_q(n,d)}{2} = \frac{|\mathcal{E}|}{2}(V_q(n,d)-1)N$$

where q is the size of the alphabet Σ, $|\mathcal{E}|$ is the number of hash functions and $V_q(n,d)$ is the volume of a sphere set in Σ^n with radius d.

Also similarly to the baseline, the size of $\mathcal{NC}(X,d)$ is minimal if words are spread equally between spheres. For a single shift, the number of buckets is $|\mathcal{C}|$, thus the number of words in each bucket is

$$\frac{N}{|\mathcal{C}|}$$

which form

$$\binom{\frac{N}{|\mathcal{C}|}}{2}$$

pairs that are added to the candidate set. If the number of necessary shifts is $|\mathcal{E}|$, the total number of candidate pairs is given by the following lemma.

Lemma 7. *Let $X \subseteq \Sigma^n$ be an N-element input set. The minimal number of candidate near-duplicates with respect to distance d arising in the code scheme with code $\mathcal{C}$ is*

$$|\mathcal{NC}_{\min}(X,d)| = |\mathcal{E}|\ |\mathcal{C}|\ \binom{\frac{N}{|\mathcal{C}|}}{2} = \frac{|\mathcal{E}|}{2}\left(\frac{N}{|\mathcal{C}|}-1\right)$$

where $|\mathcal{E}|$ is the number of hash functions.

Since the number of functions in this scheme is not obvious, it is not as straightforward to tabulate the bounds for our scenarios. We can, however, calculate two types of values: first, bounds for the candidate set if the number of shifts is given by the lower bound of the size of the shift set; and second, the maximal size for the shift set for this method to be more effective in terms of number of candidate near-duplicates compared to the baseline.

As a reminder, the upper bound for $\mathcal{NC}(X, d)$ for the baseline method was

$$|\mathcal{NC}_{\max}(X, d)| = \frac{d+1}{2}(q^{n-k} - 1)N.$$

In order for this method to be more effective, the number of candidate near-duplicates must be smaller for this method. Thus, by comparing the upper bounds, the number of shifts has to satisfy the following condition:

$$|\mathcal{E}| < \frac{(d+1)\left(q^{n\frac{d}{d+1}} - 1\right)}{V_q(n, d) - 1}.$$

For example, if $d = 1$ then the condition can be simplified to the form

$$|\mathcal{E}| < \frac{2q^{\frac{n}{2}}}{n(q-1)} - 1$$

For our scenarios, the condition has the following numerical values. If $q = 4$ and $n = 36$, for values of $d = 1, 2, 3$: $|\mathcal{E}| < 1.3 \cdot 10^9, |\mathcal{E}| < 1.5 \cdot 10^{11}, |\mathcal{E}| < 3.6 \cdot 10^{11}$ respectively.

If $q = 20$ and $n = 12$, for values of $d = 1$: $|\mathcal{E}| < 5.6 \cdot 10^5$.

So although the exact set of shifts is unknown, the bounds for the size of this set are rather large for this method to be effective. We are hopeful that a set of such shifts exist.

We can conclude that, at least in the worst case scenario for sufficiently large q and n this method is better than the baseline.

4 Conclusion

This paper proposed and studied two methods for finding near-duplicates in a large collection. The methods were defined over an abstract data collection and analysed with the application domain of DNA sequences in mind. Upper and lower bounds were given to the efficiency of both of the algorithms when compared to the brute force method. Both algorithms were declared viable and faster alternatives to the brute force method. The algorithm based on coding theory was shown to be promising and we hypothesized that it is the more efficient method on large input spaces.

Since this was only the first cursory glance at these algorithms, there are plenty more questions that could be answered in relation to this topic. The method based on coding theory could be further analysed. An implementation could be developed with restraints on time and memory consumption based on modern computer architecture. Empirical tests on real biological datasets could be conducted. Since the currently proven bounds are relatively lax, stricter bounds could be proven for datasets drawn from fixed distributions. Either of the methods could be developed further and alterations to the schemes could be proposed and analysed. For example, there may be alternatives to the parts in both schemes where all pairs were formed out of elements in all buckets. Since we can theoretically list all the neighbors of a word, we might gain in efficiency by taking this information into account. More specifically, we are looking for a fast way to find neighbouring buckets for a given input word $\mathbf{x}$ in bucket $\mathcal{B}$ that could contain near-duplicate pairs containing $\mathbf{x}$.

Overall, this study provided a relatively brief overview of a couple of algorithms that can be used in speeding up a computation that has applications in many fields.

5 References

[1] G. V. Bard. Spelling-error tolerant, order-independent pass-phrases via the Damerau-Levenshtein string-edit distance metric. In *Proceedings of the fifth Australasian symposium on ACSW frontiers-Volume 68*, pages 117–124. Australian Computer Society, Inc., 2007.

[2] M. Cameron, Y. Bernstein, and H. E. Williams. Clustered sequence representation for fast homology search. *Journal of Computational Biology*, 14(5):594–614, 2007.

[3] N. M. Daniels, A. Gallant, J. Peng, L. J. Cowen, M. Baym, and B. Berger. Compressive genomics for protein databases. *Bioinformatics*, 29(13):i283–i290, 2013.

[4] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[5] Y. Dodis, L. Reyzin, and A. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In *Advances in cryptology-Eurocrypt 2004*, pages 523–540. Springer, 2004.

[6] C. Gong, Y. Huang, X. Cheng, and S. Bai. Detecting near-duplicates in large-scale short text databases. In *Advances in Knowledge Discovery and Data Mining*, pages 877–883. Springer, 2008.

[7] Y. Ke, R. Sukthankar, and L. Huston. Efficient near-duplicate detection and sub-image retrieval. In *ACM Multimedia*, volume 4, page 5, 2004.

[8] J. P. Kumar and P. Govindarajulu. Duplicate and near duplicate documents detection: A review. *European Journal of Scientific Research*, 32(4):514–527, 2009.

[9] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[10] R. Roth. *Introduction to coding theory*. Cambridge University Press, 2006.

[11] W. A. Stein et al. *Sage Mathematics Software (Version 6.1.1)*. The Sage Development Team, 2014. http://www.sagemath.org.

[12] Wikipedia. Finite field — Wikipedia, the free encyclopedia, 2004. http://en.wikipedia.org/w/index.php?title=Finite_field&oldid=606530008 [Online; accessed 14-May-2014].

[13] X. Wu, A. G. Hauptmann, and C.-W. Ngo. Practical elimination of near-duplicates from web video search. In *Proceedings of the 15th international conference on Multimedia*, pages 218–227. ACM, 2007.

Appendix

A Code examples in Sage

Validation of theoretical results and implementations of schemes were done using Sage, an open-source mathematics software system [11].

A.1 Baseline method

The following is a simple implementation for the baseline method described in section 3.1.

In this example, the analysed sequences are randomly drawn from $\mathbb{F}_4^5$. Additionally, a simple scheme for the case where n is not divisible by $d+1$ has been implemented in the function `find_pairs`.

```
from collections import defaultdict
from itertools import combinations

def f(coordinates):
    return lambda x: tuple(x.list_from_positions(coordinates))

def find_pairs(sequences, distance, space=space):
    ## First, create the hash functions
    n = space.degree()
    # k - the length of the (longer) subsequences
    k = ceil(n/(distance+1))
    n_short = k * (distance+1) - n
    hash_functions = []
    start = 0
    for i in range(distance+1):
        length = (k-1) if i < n_short else k
        hash_functions.append(f(range(start, start+length)))
        start += length
    ## Second, fill buckets for each function
    for fun in hash_functions:
        buckets = defaultdict(list)
        for sequence in sequences:
            hash = fun(sequence)
            buckets[hash].append(sequence)
```

```
            ## Form unordered pairs for each bucket
            for hash in buckets:
                for pair in combinations(buckets[hash], 2):
                    yield pair

field = GF(4, 'a')
dimension = 5
N = 100
sequences = random_matrix(field, N, dimension)

distance = 1
for pair in find_pairs(sequences, distance):
    print pair
```

A.2 Neighbors of a single word

The following demonstrates the retrieval of a single vector's neighbors in the vectorspace $\mathrm{GF}(4)^5$

Notice how Hamming distance is calculated as per the property 1.

```
field = GF(4, 'a')
space = VectorSpace(field, 5)

word = space.random_element()

neighbors = []
for vector in space:
    distance = (word + vector).hamming_weight()
    if distance == 1:
        neighbors.append(vector)
```

A.3 Finding candidate near-duplicates for $q = 4, n = 5, d = 1$

The candidates are found by the function `find_pairs`.

```
from collections import defaultdict
from itertools import combinations


def find_pairs(sequences, code, shifts):
```

```
    for shift in shifts:
        spheres = defaultdict(list)
        for sequence in sequences:
            hash = code.decode(sequence-shift)
            spheres[tuple(hash)].append(sequence)
        for hash in spheres:
            for pair in combinations(spheres[hash], 2):
                yield pair

field = GF(4, 'a')
code = codes.HammingCode(2, field)
V = code.ambient_space()
shifts = [V((0,0,0,0,0)), V((1,0,0,0,0)), V(('a',0,0,0,0)),
    V(('a+1',0,0,0,0)), V((0,1,0,0,0)), V((0,0,0,1,0)),
    V((0,0,0,0,1))]

N = 100
sequences = random_matrix(field, N, code.length())
for pair in find_pairs(sequences, code, shifts):
    print pair
```

B Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, Gerli Viikmaa (date of birth: 17 February 1992),

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:

   (a) reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and

   (b) make available to the public via the web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright, of my thesis **Detection of Near-Duplicates Using Error-Correcting Codes**, supervised by Sven Laur,

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, 14 May 2014