DEGREE PROJECT IN ENGINEERING PHYSICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

Quantum Methods for Sequence Alignment and Metagenomics

OSCAR BULANCEA LINDVALL

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Quantum Methods for Sequence Alignment and Metagenomics

Oscar Bulancea Lindvall
oscar.lindbul@gmail.com

Quantum and Biophotonics
Department of Applied Physics
School of Engineering Sciences
Royal Institute of Technology, SE-106 91 Stockholm, Sweden

Sweden, 2019

Typeset in LaTeX

TRITA-SCI-GRU 2019:219

Graduation thesis on the subject of Applied Physics for the degree of Master of Science in Engineering Physics. The thesis was presented June 10, 2019 at 11.00 in Albanova University Center, Roslagstullsbacken 21, Stockholm.

Abstract

The analysis of genomic data poses a big data challenge. With sequencing techniques becoming both cheaper and more accessible, genomic data banks are expected to grow exponentially and surpass even the scale of astronomical data. The field of bioinformatics will therefore require more efficient methods of analysis. At the same time, quantum computing is being explored for its potential in areas such as chemistry, optimization and machine learning, and there has been recent interest in applying quantum computing in genomic analysis problems to achieve more efficient algorithms. This thesis aims to provide an overview of proposed quantum algorithms for sequence alignment, discussing their efficiency and implementability. Other quantum methods are also explored for their possible use in this field. An Ising formulation of the multiple sequence alignment problem is proposed, with qubit usage scaling only linearly in the number of sequences, and the use of this formulation is demonstrated for small-scale problems on a D-Wave quantum annealer. The properties of quantum associative memories are also discussed and investigated in the context of improving K-mer matching techniques in metagenomics, with simulations highlighting both their capabilities and current limitations.

Acknowledgements

This project was highly interdisciplinary and presented many challenges in understanding the relevant areas of both quantum computing and bioinformatics. It has been a great learning experience and I am thankful to all the people who have helped make this possible.

I would like to start by expressing my gratitude toward my supervisor Dr. Xiu Gu. I am deeply grateful for her guidance that helped me navigate this vast field of research and through the uncertain points of my work. I want to thank both her and Prof. Göran Johansson for allowing me to participate in this project, making me feel part of the team and for letting me be part of many interesting discussions.

I would also like to thank Per Sikora and Ebba Carbonnier for their contributions to my project. The meetings we had to discuss quantum applications in life sciences gave invaluable insight into bioinformatics and the possible uses of quantum computers, which undoubtedly helped shape my thesis.

I am also grateful to everyone at the department of applied quantum physics at Chalmers for letting me be part of such a fun and inspiring workplace, and for giving me so many ideas of techniques to explore during my work. I learned much about quantum computing and quantum technology from the many interesting seminars you provided.

And lastly, I want to give thanks to all my friends and family who have supported me through my years at university; my mother and father who have always cheered me on, and Eva Johansson and Mikael Bjurek who provided me with a second home while writing this thesis. Thank you all very much!

Contents

1 Introduction

In the future, genome analysis could provide targeted treatments and aid in understanding and preventing complex diseases. Ever since the start of the human genome project, the sequencing and analysis of a human genome have become increasingly faster and cheaper, opening up more possibilities for medical and research applications. Even when not used for diagnostic purposes, methods for the analysis and comparison of genomic structure contribute to research in diseases, medicine design and evolutionary developments, to name a few. However, the analysis of genetic sequences poses a big data challenge, with sequence data-banks increasing exponentially in size[1], the field is and will be in need of more efficient methods for analysis.

At the same time, the field of quantum computing is on the rise and is being explored for its potential use in chemistry, optimization and machine learning, and many others. Quantum computers are being researched by many around the world, and developed by companies such as IBM[2], Rigetti[3] and D-Wave[4], who are already offering public and commercial quantum computing platforms. With this, we are moving closer to the so called era of Noisy Intermediate Scale Quantum (NISQ) computers[5], where quantum devices on the scale of 50–100 qubits will be available, going beyond what any classical computer is capable of simulating and where it might be possible to demonstrate quantum supremacy. In recent years, there has been interest in how quantum computing could be applied in the field of bioinformatics to provide more efficient methods in analysis of genomic data and is being explored by many parts of the scientific community, as well as companies such as Entropica labs[6].

This thesis aims to provide an overview of methods proposed for quantum applications in bioinformatics, listing useful properties as well as current drawbacks or limitations. Areas and techniques which could be of use in this field in the years to come are also explored. In particular, the thesis explores quantum pattern-matching structures for genomic classification and quantum optimization for the use in sequence alignment.

In section 2, a general background is given for the relevant problems in bioinformatics and a brief introduction to quantum computing and its framework is given in section 3. This is followed by an overview of recently proposed quantum methods in section 4, focusing on the sequence alignment problems presented in section 2. The topic of quantum optimization for use in alignment is brought up in section 5 where an Ising formulation for the Multiple Sequence Alignment (MSA) problem is formulated in sections 5.1 and 5.2 and tested in sections 5.3 and 5.4. Section 6 goes on to present the field of quantum pattern-matching using Quantum Associative Memories (QuAMs), which in section 7 are adapted and investigated for their potential use in the field of metagenomics.

2 Bioinformatics

A brief introduction is given to the fields of bioinformatics that are encountered in this thesis and definitions of relevant problems are presented.

2.1 DNA-Structure

Deoxyribonucleic acid, or DNA, is a di-helical structure built out of a sugar-phosphate backbone, making out the helices, and bridging components bonding the two helices together, called nucleotides, as illustrated in fig. 1. The nucleotides differ in their base component of which there are cytosine (C), guanine (G), adenine (A) and thymine (T). These substances binds in pairs of A−T and G−C in the DNA structure, making a strand of DNA contain two complementary sequences. When divided into single-helical strands, i.e. ribonucleic acid (RNA), one sequence is called the sense sequence and is usually used for protein transcription, meanwhile the other, the anti-sense sequence, is used for DNA replication. In the analysis of DNA one typically speak in terms of the sense strand, and following this convention, DNA will be referred to as if only having one sequence. The DNA of an organism is contained in all of its cells containing a nucleus and is used for coding the structure and inner functions of the organism itself. The size of the entire genetic content of an organism, called the genome, vary vastly between the different kinds of creatures, with the vomiting bug (NORO virus) approximately consisting of seven thousand bases (having RNA and no DNA) and the human genome having on the order of $3 \cdot 10^9$ base pairs. The order of occurrence of the bases in DNA are used to code for the amino acid components in proteins. The amino acids and their order are critical for a protein's specified function. By inference, the order of DNA bases

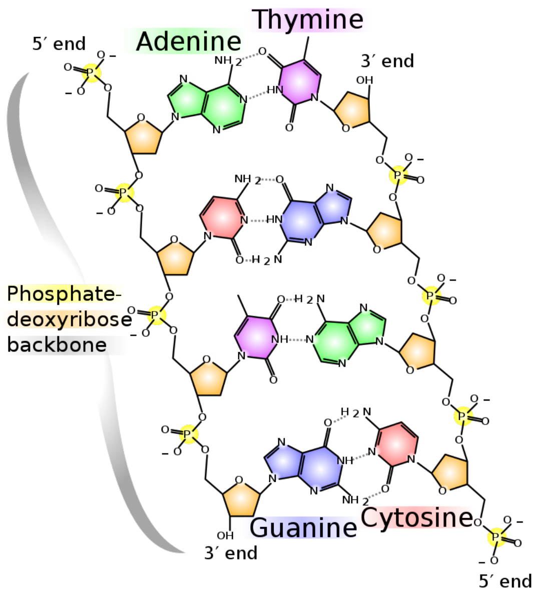


Figure 1: DNA structure (from Wikipedia[7])

in a specific sequence is used to decide the function and control of almost all bodily functions. It is this that makes the analysis of DNA sequences, in particular with the base order in mind, a cornerstone in understanding biological functions and differences between organisms. Finding sequences which code for a particular function, i.e. genes, and using the knowledge of those to infer functional properties, is an important field in both medicinal and more fundamental biological/biochemical research. DNA is highly varying, with changes in the structure, e.g. mutations or arbitrary insertion or deletions (indels) of base pairs, occurring both between generations and withing an organism's lifetime. This change is a mechanism for biological diversity, enabling the evolutionary process to occur. While such changes are capable of creating differing protein functions, it is not unusual for homologous DNA sequences, i.e. sequences with a common ancestry, to function in similar ways. If they are homologous but exhibit different behavior in their coded proteins, finding out what regions in the sequences cause the differing properties is both an interesting and challenging problem. Regions of similarity or knowledge of the particular mutations are both valuable forms of information in this field. Therefore, methods capable of highlighting and quantifying these properties have been developed for this problem, named the sequence alignment problem.

2.2 Sequence Alignment

The problem of sequence alignment is to measure and identify the similarity between string sequences. As explained in the previous section, analyzing the similarity between DNA-sequences may reveal not only evolutionary relations, but also functional relations in the associated proteins, giving it a central role in bioinformatics.

There are however different ways of measuring similarity between DNA-sequences, depending on what kind of comparison one is interested in making. In bioinformatics, the problem is therefore separated into two main categories, global and local alignments. In the latter problem, one is only interested in finding regions of similarity, possibly being substantially smaller than the length of the sequences involved. However, in the global alignment case, one is interested in the difference between the sequences in their entirety. The goal in global alignment is to find the least number of base mutations and indels explaining the deviation between the strings, typically treating it as an optimization problem where matching elements give a reward and mismatches along with indels give some kind of penalty. In this manner, it is similar to the problem of finding the minimal edit distance between strings, which is also used for analysis of natural languages and financial data. Although, the score for matching bases and penalties for edits and indels can vary depending on the type of alignment one wants to make, even to the point of making the penalty non-linear. The global alignment should therefore be considered a generalization of the edit distance problem.

2.2.1 Global Alignment

The problem of global, pair-wise, sequence alignment is defined as follows. One is initially given two sequences, $A = a_1, a_2, \ldots, a_n$ and $B = b_1, b_2, \ldots, b_m$, where the characters $\{a_i\}$ and $\{b_i\}$ are part of some common alphabet, e.g. {'A','T','C','G'} for DNA. The problem then consists of adding indel characters (usually added to the alphabet as a character such as "–") making new sequences $A' = a'_1, \ldots, a'_N$ and $B' = b'_1, \ldots, b'_N$ such that the pairing of characters between them optimizes some similarity score function $\text{sim}(A', B')$. Note that every alignment will consist of at least $|n - m|$ indels, making $N \geq \max(n, m)$, due to the difference in sequence length. Typically, this score function is a sum over the individual matching scores of the pairs of characters between the sequences, e.g.

$$\text{sim}(A', B') = \sum_{i=1}^{N} \text{score}(a'_i, b'_i), \tag{2.1}$$

where the score(a, b) function a reward or penalty depending on which of the scenarios

- $a = b$, matching bases,
- $a \neq b$, mismatching bases,
- a or b is an indel

is encountered. For example, say one chooses a score of $+1$ for each match and a -1 for a mismatch or indel, then the sequences ACGCT and CCAT would have an alignment

$$\begin{array}{lcccccc} A': & \text{A} & \text{C} & \text{G} & \text{C} & - & \text{T} \\ B': & - & \text{C} & - & \text{C} & \text{A} & \text{T} \end{array}$$

where vertical alignment indicates pairing of elements. According to our scoring scheme, the score is

$$\text{sim}\left(\begin{array}{cccccc} \text{A} & \text{C} & \text{G} & \text{C} & - & \text{T} \\ - & \text{C} & - & \text{C} & \text{A} & \text{T} \end{array}\right) = \overbrace{\text{score}(A, -) + \text{score}(G, -) + \text{score}(-, A)}^{-3}$$
$$+ \underbrace{2\,\text{score}(C, C) + \text{score}(T, T)}_{3} = 0,$$

which one can visually confirm is an optimal solution.

In this way, sequence similarity can be identified by interpreting mismatches as possible base mutations and indels as arbitrary insertions/deletions of nucleotides, respectively. A global alignment therefore gives a relatively indicative picture of how each sequence evolved, and detailed information over which parts the sequences could initially have in common. However, the alignment procedure is not only used when detailed alignment information is needed. It is common in many areas of bioinformatics to compare a genomic sequence to those contained in a database, and for the purpose of quantifying the similarity, the similarity function defined in global alignment is commonly used to quantify sequence similarity.

The method for classically finding the optimal alignment is called the *Needleman-Wunsch* algorithm, which makes use of dynamic programming. To give a perspective on the complexity of the alignment method, a brief description of the algorithm is provided.

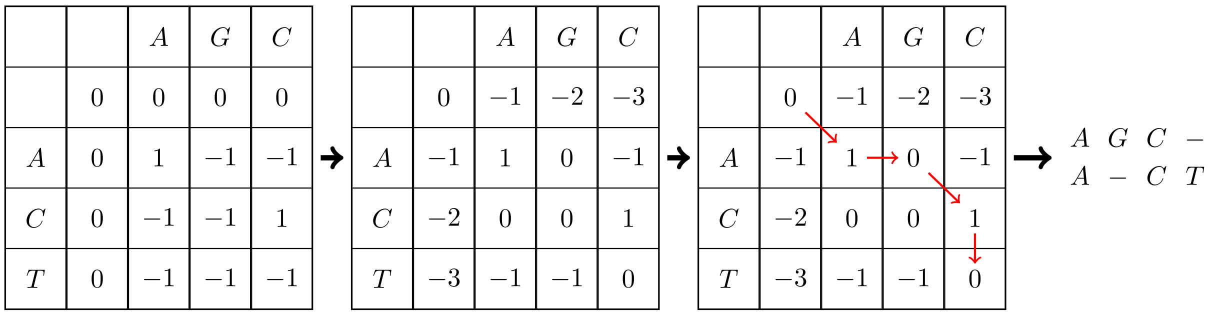
1. A so called substitution matrix, D, is constructed for the two sequences involved, consisting of the matching scores for the initial sequences, i.e. $D_{i,j} = \text{score}(a_i, b_j)$.
2. A cumulative score matrix, S, is constructed by the recursive scheme

$$S_{i,j} = \begin{cases} (i+j)g & \text{if i = 0 or j = 0} \\ \max\begin{pmatrix} S_{i-1,j-1} + D_{i,j}, \\ S_{i-1,j} + g, \\ S_{i,j-1} + g \end{pmatrix} & \text{otherwise} \end{cases}, \tag{2.2}$$

where g is the indel penalty. This recursive scheme maximizes the reward and minimizes the penalties by considering all possible indel insertions. If at position (i, j), the term $S_{i-1,j-1} + D_{i,j}$ is chosen, it means the score is maximized by simply letting the bases a_i and b_j be aligned, if any of the other terms are larger, then it is more favorable to insert an indel into the first or second sequence respectively. The score in $S_{n,m}$ is thus the score for the optimal alignment.

3. By backtracking the choices which created the score $S_{n,m}$, one then finds the optimal alignment configuration.

An example of running this algorithm for sequences $\{A, G, C\}$ and $\{A, C, T\}$ with a reward of 1 for matching, and a penalty of -1 for mismatch and indels, is shown in fig. 2. As it is described above, the algorithm has a worst case complexity of $O(nm)$ in both memory and time, due to the initialization and updating of the substitution matrix.



		A	G	C
	0	0	0	0
A	0	1	-1	-1
C	0	-1	-1	1
T	0	-1	-1	-1

		A	G	C
	0	-1	-2	-3
A	-1	1	0	-1
C	-2	0	0	1
T	-3	-1	-1	0

		A	G	C
	0	-1	-2	-3
A	-1	1	0	-1
C	-2	0	0	1
T	-3	-1	-1	0

$$\begin{array}{cccc} A & G & C & - \\ A & - & C & T \end{array}$$

Figure 2: An example of the steps in the Needleman-Wunsch algorithm, showing the substitution matrix construction, the cumulative score calculation and backtracking resulting in the displayed alignment, from left to right. The found path indicates in order of start to finish: 1. match, 2. indel in second sequence, 3. match, 4. indel in first sequence.

2.2.2 Local Alignment

In contrast to the global alignment, local alignment methods try to find an alignment for the most similar regions of two sequences, even if the alignments of other regions are disregarded entirely. As such, the problem is similarly defined as in section 2.2.1, except for the modification of the similarity score (2.1). The sum is not taken over the entire alignment configuration, but is instead optimized over all different subsequences. Such a procedure is useful in finding conserved domains between different species of genomes and for finding hidden motifs, i.e. reoccurring sequences that tend to be biologically significant.

There exists qualitative, but also simple, methods of finding such alignments, called dot matrix methods. Such methods are based on constructing the so called dot matrix. It is similar to the substitution matrix made in the Needleman-Wunsch, but is simpler in the sense that matching base pairs are only marked and not assigned a score. The dot matrix for the example sequences ACCGT and TCCGTACC is constructed as in fig. 3, and the desired local alignments can be found visually by identifying diagonal lines of relevant lengths. It is effective for visualizing the similarities when the noise levels are low, and is effective in finding matching recurrences between sequences. However, it is more fitting as a visual aid, and more robust methods are needed for large-scale data.

The *Smith-Waterman* algorithm is a modification to the Needleman-Wunsch method, using dynamic programming to find one or more local alignments between two sequences. It works as follows:

Step 1 As in the Needleman-Wunsch method, the substitution matrix, D, is created.

Step 2 The cumulative score matrix, S, is also similarly constructed, however with slightly modified rules.

$$S_{i,j} = \begin{cases} 0 & \text{if } i \leq 1 \text{ or } j \leq 1 \\ \max \begin{pmatrix} S_{i-1,j-1} + D_{i,j}, \\ S_{i-1,j} + g, \\ S_{i,j-1} + g, \\ 0 \end{pmatrix} & \text{otherwise} \end{cases}, \tag{2.3}$$

	T	C	C	G	T	A	C	C
A						●		
C		●	●				●	●
C		●	●				●	●
G				●				
T	●				●			

Figure 3: Dot matrix of ACCGT and TCCGTACC.

Here, the score never takes negative values, which is important as negative scores are interpreted as failing with global alignment. In local alignments, it is not as interesting to include failing parts of the alignment and therefore the zero scores act as markers for suitable local alignment separators.

Step 3 And finally, a traceback is performed, although starting in the matrix cell of highest score, until a cell with a zero score is reached. The path taken gives the best local match between the sequences. The traceback process may also be repeated if other local similarities are desired.

As in the global alignment, methods relying on dynamic programming requires the creation and updating of the substitution matrix, which consists of $\mathcal{O}(nm)$ operations in time and memory. Although having the same complexity, the Smith-Waterman is slightly more expensive, as one also has to find the cell with highest score before the traceback.

2.2.3 Multiple Sequence Alignment

So far, we have only considered pair-wise alignment, i.e. alignment between two sequences. However, in bioinformatics, it can be of interest to find similar regions or a global alignment of more than two sequences. Such an alignment can highlight conserved sequences between different generations of genes, as well as mutations. Both are important traits for understanding the genetic and functional evolution between homologous genes. Multiple sequence alignment (MSA) is more complex than the two-sequence case, as the optimal solution for the multiple alignment can often be different than the optimal alignment for each pair of sequences. There is also no unique way of quantifying the alignment score between more than two sequences.

There is however, a scoring scheme that is commonly used called the *sum-of-pairs* (SP) scoring, where the score for each alignment column in the configuration is evaluated as the sum of the pair-wise score for all element pairs in the column. The problem is defined as starting with L sequences $\{S_1, \ldots, S_L\}$, each with a number of characters $S_i = s_{i,1}, s_{i,2}, \ldots, s_{i,n_i}$. The sequences are then modified by insertion of indels. Assuming they are modified to have the same number of characters N, the similarity score is defined as

$$\mathrm{sim}(S_1, \ldots, S_L) = \sum_{i=1}^{L} \mathrm{columnscore}(s_{1,i}, s_{2,i}, \ldots, s_{N,i}), \tag{2.4}$$

which with the SP scoring would be formulated using a pair-wise scoring function, as

$$\mathrm{sim}(S_1, \ldots, S_L) = \sum_{i=1}^{L} \sum_{j=1}^{L} \sum_{k=j+1}^{L} \mathrm{score}(s_{j,i}, s_{k,i}). \tag{2.5}$$

In this way, multiple alignment can be seen as a generalization of the global alignment problem and can in fact be solved with a generalization of the Needleman-Wunsch algorithm using an L-dimensional substitution matrix. Although this would cause a memory and time complexity which is $\mathcal{O}(N^L)$ if no optimization or heuristic is used. It has actually been shown that multiple sequence alignment with the SP scoring scheme is NP-complete[8].

With the lack of efficient methods yielding the optimal solution, most algorithms for MSA are based on heuristic approaches. Progressive methods, as they are named, constructs the alignment by a step-by-step approach using the pair-wise alignments. Starting from the most similar sequence pair, the MSA alignment is made by appending one sequence at a time, guided by a strategy using the similarity distance between all pairs as input[9]. Clustal and T-Coffee are among the most popular progressive alignment methods[10, 11]. Another type of approach consist of genetic algorithms, which rely on optimization methods in finding the best alignment. Different types of alignments are considered based on where indels are placed, and the candidate alignments are randomly modified according to preset rules conjectured for the evolutionary process of genomes, and then selected or discarded depending on how the alignment affects an objective function.

2.3 Metagenomics

Metagenomics is the study of the genetic makeup of samples taken directly from their native environments. Early genomic analysis mostly used artificially grown cultures, but with the advance of sequencing technology, it was shown that such cultures contained less than one percent of the diversity in the original sample[13]. Methods were therefore developed to analyze samples taken directly from the corresponding environment, as to conserve this diversity. Metagenomic analysis is capable of measuring the microbial content in the given sample, as well as what metabolic processes are possible within the sample environment. Such methods present new opportunities in a variety of fields, such as ecology and medicine, where the bacterial and viral content have a vital role. In agriculture, knowledge of how the microbial content in the soil affects plant growth could lead to more effective disease detection and more efficient farming practices. In environmental studies, the microbial content and their metabolic processes give insight into how well an ecosystem, such as a lake, is capable of handling pollution. And in medicine, metagenomics is being used to research the connection between the human microbial gut content and bowel diseases[14], and has been a successful method for finding connections between certain microbial infections to the development of cancer, which could be used to improve treatment methods[15].

A metagenomics project typically follows the workflow of fig. 4. A sample is extracted from the environment one wishes to study and is thereafter processed to filter out unwanted content if necessary. The process of pre-filtering is often applied when the sample comes from a host that is not the target of the study. If the volume of the DNA content one wants to research is low, it is amplified by artificial means. Then, after isolating the DNA content, extraction of the sequence base pairs is performed by sequencing techniques. A modern and commonly used group of sequencing methods are that of next-generation (high-

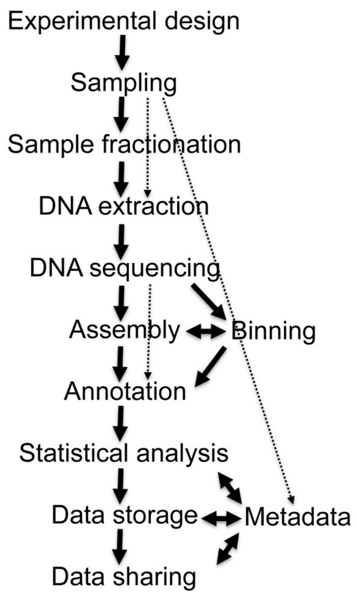


Figure 4: Flow diagram of a typical metagenomics project. Taken from [12].

throughput) techniques. In such procedures, one starts by fragmenting the DNA content by chemical means and then identifies sequence fragments of typically small length (e.g. an Illumina sequencer, producing ∼150 base pairs), but in parallel, producing a large volume of genomic data. Combined with strategies for fragmenting and post-processing long sequences, e.g. Shotgun sequencing[16], one can nowadays sequence up to 18,000 human genomes each year[17]. What one does with the obtained genomic data then depends on the purpose of the study. If the goal is to find the sequence and analyze its structure, one would use sequence assembly methods, which reconstructs the contained sequence by combining the sequenced fragments. This thesis will however focus on another type of goal, which is taxonomic classification.

In some of the areas mentioned above, one is not as interested in the exact structure of the DNA found, but rather if it can be classified to belong to certain organisms or genes. This is achieved by binning techniques, effectively sorting each read into some taxonomic category and at the end of the analysis drawing some conclusion based on the binning distribution. There are several tools which use different strategies for binning, but in this thesis, attention is turned to methods specializing in so called "K-mer" based techniques. In particular, focus is given to the method of Kraken and its binning technique. Kraken[18] uses a database of reference sequences, making up its taxonomic categories. However, the

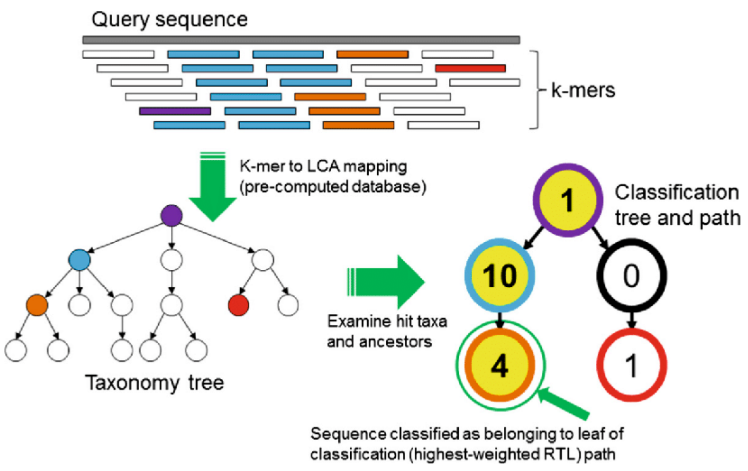

Query sequence
k-mers
K-mer to LCA mapping
(pre-computed database)
Taxonomy tree
Examine hit taxa and ancestors
Classification tree and path
1
10
0
4
1
Sequence classified as belonging to leaf of classification (highest-weighted RTL) path

Figure 5: The Kraken sequence classification algorithm. To classify a sequence, each K-mer in the sequence is mapped to the lowest common ancestor (LCA) of the genomes that contain that K-mer in a database. The taxa associated with the sequence's K-mers, as well as the taxa's ancestors, form a pruned subtree of the general taxonomy tree, which is used for classification. Taken from [18].

database is not structured around the sequences themselves. Each sequence is identified by a characteristic K-mer, a subsequence of length K. The binning technique used by Kraken, illustrated in fig. 5, then consists of extracting all K-mers contained in the given query sequence, which could be produced by a sequencer, and searching the database with the K-mers as search queries. If the K-mer is contained in the database, then a taxonomic tree, representing the evolutionary relationship between the reference K-mers, is updated accordingly. After processing every query K-mer, the taxonomic tree is processed to give a conclusion as to what the taxonomic identity of the query sequence should be. Kraken and similar methods are however not without flaws as they, for example, rely on exact matching of their queries. While exact matching can be made highly efficient for the typical K-mer length of $K = 31$ or less, sequencing errors caused either by erroneous DNA cloning or readout, result in a reduced identification accuracy. And while similarity-based matching can be incorporated to dampen the effect of these errors, it is computationally expensive to implement in a database search.

The process of Kraken and other methods making use of this "K-mer counting" taxonomic classification approach could provide good starting points for quantum improvements in metagenomics. The biggest restriction placed on universal quantum computing for use in genomic application is the limitation on the number of qubits. Most methods rely on encoding the entire DNA, RNA or protein sequence into a bit string (i.e. a single state) or a quantum structure of similar size. Considering the length of a typical gene or genome, which could be several thousands or millions of base pairs, this seems infeasible at the modern scale of genomics. However, with the short-length K-mers used in these methods, they are much more feasible to consider running on quantum computers within the years to come. In section 7, research is done on quantum methods that could be suitable for improving the search procedure in these binning methods, mostly focusing on the scenario of Kraken.

3 Quantum Computing

Quantum computers are devices much like classical digital computers, with the vital difference that information is stored and processed using the elementary two-level quantum system, the *qubit*

$$|q\rangle = \alpha |0\rangle + \beta |1\rangle . \tag{3.1}$$

The possibility of quantum superposition, letting $|q\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ or any choice of α and β adhering to $|\alpha|^2 + |\beta|^2 = 1$, opens new doors for parallel processing, and the growth of the state space through the tensor product

$$|q_1, \dots, q_n\rangle = (\alpha_1 |0\rangle + \beta_1 |1\rangle) \otimes \cdots \otimes (\alpha_n |0\rangle + \beta_n |1\rangle) \in \mathbb{C}^{2^n}, \tag{3.2}$$

being exponential in the number of bits, could reduce resource requirements. But this has to be balanced with the nature of quantum mechanics that limits the flexibility of quantum algorithms, involving the probabilistic state measurement and collapse. The linearity of the theory is also the cause of several limitations. The no-cloning theorem[19] in particular prevents the copying of information with quantum states, in the sense of perfectly and deterministically replicating an arbitrary and unknown quantum state $|\psi\rangle$ through a unitary U as

$$U |\psi\rangle \otimes |\phi\rangle = |\psi\rangle \otimes |\psi\rangle . \tag{3.3}$$

Despite these limitations, theoretical uses for quantum computations have been investigated in the areas of prime factoring and unordered search where the famous Shor's algorithm promises exponential speedup compared to classical prime factoring if implemented correctly and Grover search is able to retrieve a desired element from an unordered database of N items in $\mathcal{O}(\sqrt{N})$ database look-ups, although this speedup would depend on the specific problem and type of search, which is discussed further on. Quantum computers are also expected to improve results in simulations of chemical systems, in optimization and machine learning.

3.1 Computational Model and Implementation

The quantum algorithm will have to be tailored and expressed in a model for the hardware it is applied within. For the qubit, one can in essence utilize any two-level system or even a restricted multilevel one, e.g. anything from the nuclear spin of an atom, a photon's optical cavity to advanced solutions such as a quantum states of a superconducting system. There are however pros and cons with each implementation, particularly for the system's stability in regards to decoherence and the time of performing operations on the qubits. This determines the number of one-qubit operations that can be performed on the same qubits before the state needs to be reset, and table 1 lists relevant values for various implementations, which are summarized in [20]. The listed implementations would differ in the types of operations that are implementable, in how we as experimenters would interact with the system and how the qubits themselves would interact with each other. The way to perform quantum computations would have to be modeled accordingly, and among the many computational models, the so called *quantum circuit model* is commonly used, and supported by most publicly available quantum hardware platforms, as the IBMQ[2] and Rigetti[3]. Although the reader should note that this model treats universal quantum computers, being universal in the sense of being able to run or simulate any quantum device. There are other forms of quantum computations, such as quantum adiabatic computation which is performed in D-Wave's systems, but those are generally limited in the algorithms they can run.

The quantum circuit model is an extension to the classical Boolean circuit model of computation, allowing the use of qubits, reversible and unitary operators, and measurements, based on the following assumptions.

1. The quantum computer is coupled to a classical device that is responsible for measurement registration and processing. One can also assume it is capable of controlling basis state preparation and the application of quantum operators.

System	Coherence time (s), τ_Q	Operation time (s), τ_{op}	$\eta_{op} = \tau_Q/\tau_{op}$
Nuclear spin	$10^{-2} - 10^{8}$	$10^{-3} - 10^{-6}$	$10^{5} - 10^{14}$
Electron spin	10^{-3}	10^{-7}	10^{4}
Ion trap (In$^+$)	10^{-1}	10^{-14}	10^{13}
Electron $-$ Au	10^{-8}	10^{-14}	10^{6}
Electron $-$ GaAs	10^{-10}	10^{-13}	10^{3}
Quantum dot	10^{-6}	10^{-9}	10^{3}
Optical cavity	10^{-5}	10^{-14}	10^{9}
Microwave cavity	10^{0}	10^{-4}	10^{4}

Table 1: Estimates of decoherence times τ_Q and operation times τ_{op} in seconds, and the feasible number of operations that can be performed on the same quantum state, η_{op}, for various qubit implementations[20].

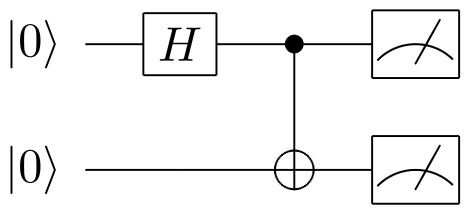


Figure 6: Circuit model of operation creating the entangled state $|\psi\rangle = (|00\rangle + |11\rangle)/\sqrt{2}$ and subsequently destroying it by measuring both qubits, producing either the $|00\rangle$ or $|11\rangle$ state with equal probability. Evaluated left to right.

2. The circuit acts on registers, i.e. collections of n qubits, with a predefined computational basis, written as the $|0\rangle$ and $|1\rangle$ states in correspondence to the 0 and 1 states of a classical bit.
3. Any one state of the computational basis can be prepared in at most n steps.
4. A gate, i.e. a quantum operation, can be applied to any group of qubits in the circuit, and there exist a *universal* set of gates that can be implemented by the hardware.
5. A measurement can be performed at any point in time, on one or more circuit qubits.

A demonstration of a quantum circuit is given in fig. 6, showing an initial state preparation in the $|00\rangle$ state, standard operations (see table 2) creating an entangled state and finishing with measurement on both qubits.

3.2 Quantum Gates

Quantum algorithms would in their simplest forms be applications of arbitrary unitaries, finishing with measurements, but in actual use one is limited to applying unitary operations implementable by the hardware. Table 2 lists standard and commonly used quantum gates and their effects, which are composed of mostly one-qubit gates with the exception of the CNOT, SWAP and TOFFOLI gates. Limited by the use of these gates, one is required to examine the topic of gate decomposition, i.e. separating a gate acting on d qubits into a series of one- or two-qubit gates, and gate set *universality*. In Boolean logic circuits, the AND, OR and NOT gates are together said to be universal, in the sense that they can be used to construct any other Boolean operation. Similarly, in quantum computing there exists operators, with a standard choice being the Hadamard, CNOT and T gates, which can be used to approximate any one- or two-qubit unitary operation to arbitrary accuracy. The choice of a universal gate set is not unique, but the existence of one is vital to prove the computability of operations in the given model. In the quantum

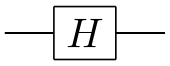
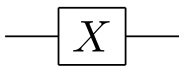
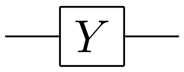
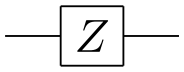
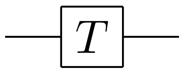
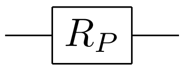
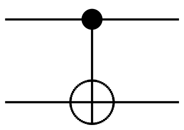
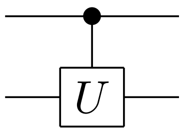
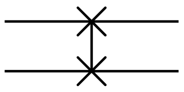
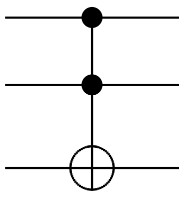
Name	Circuit notation	Effect
Hadamard		$H\|0\rangle = \frac{\|0\rangle + \|1\rangle}{\sqrt{2}}$ $H\|1\rangle = \frac{\|0\rangle - \|1\rangle}{\sqrt{2}}$
Pauli-X		$X\|0\rangle = \|1\rangle$ $X\|1\rangle = \|0\rangle$
Pauli-Y		$Y\|0\rangle = i\|1\rangle$ $Y\|1\rangle = -i\|0\rangle$
Pauli-Z		$Z\|0\rangle = \|0\rangle$ $Z\|1\rangle = -\|1\rangle$
T, or $\pi/8$-gate		$T\|0\rangle = \|0\rangle$ $T\|1\rangle = e^{i\frac{\pi}{4}}\|1\rangle$
P rotation $P \in \{X, Y, Z\}$		$R_P(\theta) = \cos\left(\frac{\theta}{2}\right) - i\sin\left(\frac{\theta}{2}\right)P$
Controlled-NOT, or CNOT		$\text{CNOT} = \|0\rangle\langle 0\| \otimes I + \|1\rangle\langle 1\| \otimes X$
Controlled-unitary CU, U one-qubit gate		$CU = \|0\rangle\langle 0\| \otimes I + \|1\rangle\langle 1\| \otimes U$
SWAP		$\text{SWAP}_{q_1,q_2}\|q_1q_2\rangle = \|q_2q_1\rangle$
Toffoli, or multi-controlled NOT		$\text{TOFFOLI}_{q_1q_2,q_3}\|q_1q_2q_3\rangle$ $= \|q_1q_2\rangle \otimes \begin{cases} X\|q_3\rangle, & \|q_1q_2\rangle = \|11\rangle \\ \|q_3\rangle, & \text{otherwise} \end{cases}$

Table 2: Standard and typically implementable simple quantum gates and their circuit notation.

circuit model, it is assumed that the hardware is capable of implementing at least one such universal gate set, such that in theory, all unitary operations are constructable.

On the topic of decomposition however, it has been shown that an arbitrary n-qubit unitary can be decomposed into at most $\mathcal{O}(n^2 4^n)$ one-qubit and CNOT gates[20]. This is certainly not an efficient number of operations, but it is merely an upper bound, and there are unitaries which are possible to decompose more efficiently. But there are also unitaries which cannot be decomposed efficiently, and should be considered in the design of quantum algorithms. This also relates to the preparation of an arbitrary quantum state, which has been shown to be inefficient in the sense of using an exponential number of simple gates. This is easily realized as the preparation of an arbitrary state $|\psi\rangle$ from some initial state $|0,\dots,0\rangle$ is the same as designing an arbitrary unitary, mapping the zero state to $|\psi\rangle$.

3.3 Grover Search

Grover search has become an important part of many newly developed quantum algorithms, for its possible speedup and adaptability. As many of the methods presented here are Grover-inspired, a brief introduction of the algorithm is given.

Grover search is performed as described in algorithm 1, and consists of three main components. The first is the preparation of the search state $|\psi_0\rangle$ in the uniform superposition over all states. Such an operation can be achieved by applying a Hadamard gate on each qubit,

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle = \left(\bigotimes_{i=1}^{n} H_i \right) |0_1, \dots, 0_n\rangle \,, \tag{3.4}$$

as it was seen in table 2 to transform each computational basis state into an equal superposition of both states. The other main parts are the phase inversion operator $I_O |i\rangle = (-1)^{O(i)} |i\rangle$ and the Grover operator

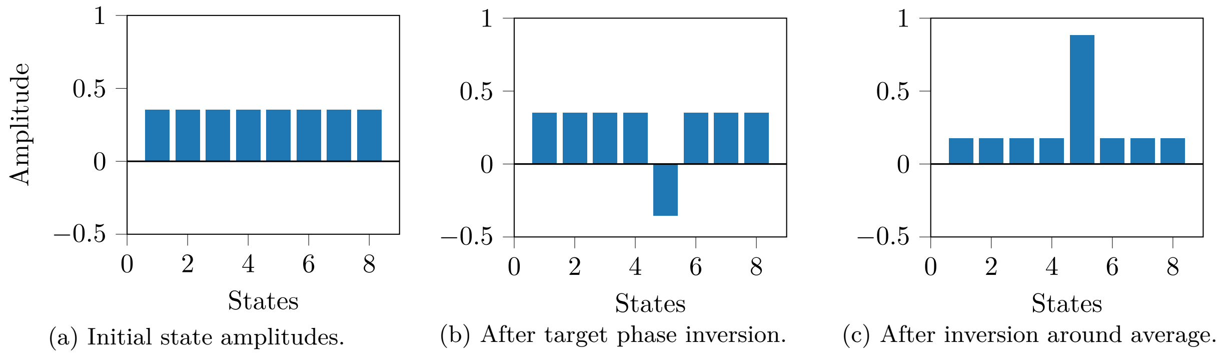


Figure 7: One iteration of Grover search with one marked state among 8 items.

$G = 2|s\rangle\langle s| - I$. The former operator inverts the phase of all states satisfying an oracle function O by $O(i) = 1$. All states not satisfying the oracle takes $O(i) = 0$. After phase inversion, the Grover operator, otherwise called the diffusion operator can be applied as

$$G = 2|s\rangle\langle s| - I = \left(\bigotimes_{i=1}^{n} H_i \right) \left(|0_1, \dots, 0_n\rangle\langle 0_1, \dots, 0_n| - I \right) \left(\bigotimes_{i=1}^{n} H_i \right) , \tag{3.5}$$

where the middle operator can be decomposed by $2n$ uses of X-gates and an n-bit Toffoli gate, making the entire operator decomposable to $\mathcal{O}(n)$ simple gates. The intuition behind this operator is to "invert amplitudes around the mean" causing states with inverted phase to become peaked, as illustrated in fig. 7, giving high probability of measurement at an optimal number of iterations found to be $\mathcal{O}(\sqrt{2^n})$. While algorithm 1 demonstrates the case of assuming only one state is marked, the method has been generalized to taking an arbitrary initial state[21], having an arbitrary number of solutions which can be counted before execution[22], and allowing for a more arbitrary phase change induced by I_O to reduce the number of iterations[23]. The algorithm can be implemented efficiently, up to the construction of the phase inversion operator that is dependent on the oracle. The oracle is a way of making the method adaptable to any

Algorithm 1 Grover search with $N = 2^n$ elements with one solution

```
1: $|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$          ▷ Initialize database state
2: for for i=1 to ROUND $\left(\frac{\pi}{4}\sqrt{N}\right)$ do
3:     Set $|\psi\rangle = I_O |\psi\rangle$          ▷ Mark solution state
4:     Set $|\psi\rangle = G |\psi\rangle$          ▷ Invert around average
5: end for
6: Measure $|\psi\rangle$
```

problem, but if it is inefficient to evaluate by quantum computations for a particular problem, then so is the Grover search.

3.4 Quantum Annealing

Quantum annealing is a form of adiabatic computation meant to solve unconstrained optimization problems, and is implemented in D-Wave quantum devices. The problem the quantum annealer solves is to find the ground state of a given Hamiltonian H_{opt}. The quantum annealing technique is to let the state of the device evolve adiabatically from an easily constructed state, described as the ground state of an operator H_0. Qubit interactions are scaled with time toward the final Hamiltonian H_{opt}. Based on the normalized time parameter $s = t/T \in [0, 1]$, where T is the total annealing time, the total system Hamiltonian is described as

$$H = A(s)H_0 + B(s)H_{opt}, \tag{3.6}$$

with $A(s)$ and $B(s)$ as functions that can be chosen depending on the implementation, but that satisfy $A(0) = 1 - B(0) = 1$ and $A(1) = 1 - B(1) = 0$. The adiabatic theorem of quantum mechanics states that if the annealing time T is long enough, or rather $A(s)$ and $B(s)$ vary slowly enough, then the system will remain in the ground state of the total Hamiltonian throughout this evolution, ending up in the desired ground state of H_{opt}[24]. In practice this technique is not perfect and will rely on repeating such an adiabatic evolution and measurement, but the measurement result with the largest probability should be the ground state if done correctly. One can compare this to simulated annealing in which the solution states (not necessarily quantum) are explored through a temperature-dependent random walk, and similarly sampled.

In implementing these techniques in real devices there are however physical and technical limitations to what can be achieved. In a quantum annealing device, qubits are meant to be acted upon by external forces as to develop the interactions desired in the problem formulation. However, the implementable interaction strength is typically limited and the connectivity in the quantum device is constrained to what is called the *chimera graph*, representing the supported connectivity of the device. When encoding the problem onto the quantum annealer, one therefore requires an embedding onto the chimera graph, i.e. a reformulation of the problem using the interactions which are available. This could require scaling interactions and usage of more than one physical qubit to encode one logical (problem) qubit, as demonstrated in fig. 8.

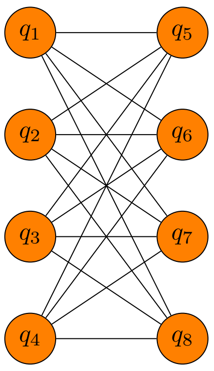
4 Quantum Methods for Alignment

One advantage of quantum algorithms lies in the inherent parallelism in quantum superposition, and interference among such superposed states. This is what provides the speedup in Grover search and other algorithms such as the quantum Fourier transform[26]. The properties of these algorithms that utilize this parallelism have therefore become popular starting points for newly developed methods, and this section introduces some of the methods recently proposed for the use in the sequence alignment or the string matching problem.

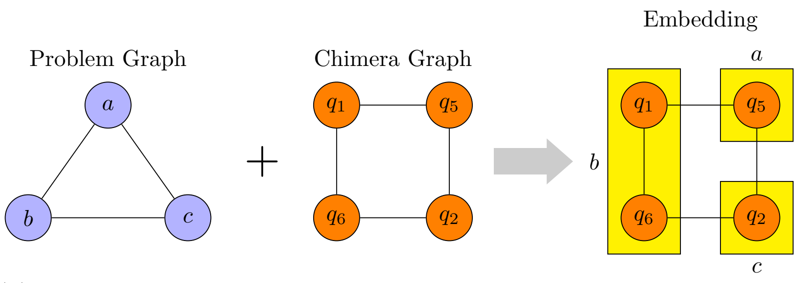
4.1 Grover-Based Methods

4.1.1 Hamming Distance Lookup

One search algorithm to be adapted to genomic matching was proposed by Hollenberg[27], which makes use of a Grover search procedure to find the database position of matching sequences. What is considered



(a) Chimera graph example of unit cell of D-Wave2000Q. Having qubits numbered 1 to 8.



(b) Embedding of three-qubit loop onto chimera graph in fig. 8a. Using two nodes to represent one physical qubit is required.

Figure 8: Chimera graph unit cell of D-Wave[25] and chimera graph embedding example.

is a database of genomic sequences, $D = \{R_1, \ldots, R_N\}$ and a reference sequence $r = r_1, \ldots, r_M$ to be searched for in the database. The physical interpretation of this database could be either a collection of genes from a real database, or a list of subsequences of length M in some sequence of length $N + M$. The method was proposed for matching protein sequences, which in comparison with DNA have generally smaller sequences albeit a larger alphabet (20 amino acids versus 4 DNA base pairs), but it is possible to apply to either type of sequence. His method consists of entangling two registers Q_1 and Q_2 with database sequences and indices, respectively, in which the latter could encode the subsequence position in its origin. The database state would be constructed as

$$|D\rangle = \frac{1}{\sqrt{N-M+1}} \sum_{i=1}^{N} |B_i\rangle \otimes |i\rangle \,, \tag{4.1}$$

where $|B_i\rangle$ in Q_1 holds the bit representation of sequence R_i and $|i\rangle$ the index state. After database construction, the register Q_1 is transformed according to the states' difference to the reference, using a series of CNOT gates with the reference bits acting as controls. This would change the sequence states $|B_i\rangle$ into what he calls "Hamming states", $|\bar{B}_i\rangle$, showing the differing sequence and reference elements as qubits in the $|1\rangle$ state. This is then used to construct an oracle in Grover search, marking the states having all zeros, or in terms of the Hamming distance function T, having $T = 0$. The phase inversion operator in Grover search would then be designed as

$$I_S |\bar{B}_i\rangle = \begin{cases} -|\bar{B}_i\rangle & T(i) = 0 \\ |\bar{B}_i\rangle & \text{otherwise} \end{cases} \,, \tag{4.2}$$

and the diffusion operator as $I_H = 2|H\rangle\langle H| - I$, acting around the Hamming state

$$|H\rangle = \frac{1}{\sqrt{N-M+1}} \sum_{i=1}^{N} |\bar{B}_i\rangle \otimes |i\rangle \,, \tag{4.3}$$

on register Q_1. If these operators are applied while knowing the number of marked states k in the database, for which there are strategies to calculate[22], then one of those solutions will be found in $\mathcal{O}(\sqrt{N/k})$ Grover iterations. This solution would then mark the position in the database or sequence, where the reference is located.

While this formulation finds an exact match, an extension is suggested to allow for fuzzy matching, i.e. similarity-based matching, by repeating the Grover search and each time searching for a new specific Hamming distance $T = n$, as is done in attempts to use Grover search for optimization[28, 29]. Taking this into account, one would in the worst case check all possible Hamming distances, making the worst case complexity $\mathcal{O}(rM\sqrt{N/k})$, with r being a certainty measure in the subroutine to count the number of marked states. However, this method was proposed as a starting point in quantum sequence alignment and does not discuss more practical aspects of its implementation. One thing to note in his approach is the transformation of database states into Hamming states, which greatly simplifies the construction of the phase inversion operator. For the $T = 0$ case, it is easily constructed by a multibit Toffoli gate, together with X and Hadamard gates as shown in fig. 9.

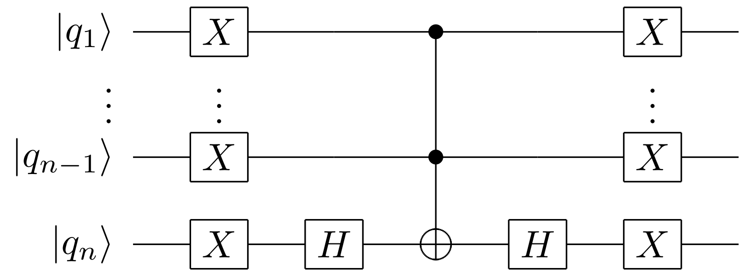


Figure 9: Phase inversion circuit for the exact matching $T = 0$ case.

As the transformation is done over the superposition of database entries and therefore in parallel, it effectively summarizes the comparison of database elements in one simple operation, which cannot be done classically. The efficiency of finding a solution when performing exact matching might be superior to the classical case, assuming the database is initially unsorted. However, if it were an extensive genomic database, then it is likely for there to be a sorting strategy in place, reducing the complexity of classical exact searching to at least $\mathcal{O}(M \log N)$ through binary searching. Although, were the database constructed from a genomic sequence without initial pre-processing as in the local alignment problem, it would essentially keep its advantage when using small reference patterns, in comparison to the Smith-Waterman algorithm and other exact string matching methods, having complexity of $\mathcal{O}(M + N)$ or $\mathcal{O}(NM)$ [30]. But the Smith-Waterman algorithm would also acquire approximate matches, for which this method would keep its advantage only if

1. the new oracle for checking $T = n$ could be implemented efficiently,
2. and the database construction of (4.1) could be done in sub-polynomial time.

If these conditions cannot be fulfilled, then either oracle call or memory reconstruction after a collapsing measurement would cause the search over one Hamming distance to be $\mathcal{O}(N)$ at the least, effectively ridding the method of its advantage.

4.1.2 Conditional Oracle Search

Another technique inspired by Grover search is that of Mateus and Omar[31, 32] for closest pattern matching using an oracle-based recall procedure. While the method is not directly targeted toward genetic pattern search, the adaption is straightforward. The idea of the method is to amplify the states in a superposition, representing the positions and elements in a sequence. This is executed through a Grover-inspired search procedure, but with an amount of oracles equal to the size of the sequence alphabet Σ, executed with a pattern of length M, $p_1 \dots p_M$, and a search sequence of length N, with characters $w_1 \dots w_N$ according to algorithm 2. Here, the operators U_{p_j} act upon a state $|i\rangle$ by inverting the phase if $w_i = p_j$ and G is the usual Grover operator $G = 2|\psi_0\rangle\langle\psi_0| - I$. What phase operator is applied is chosen conditionally on the reference element being processed and one would essentially need M number of oracles. By the repeated inversion of phase for random positions in the reference pattern, and amplification through the Grover diffusion operator, the state storing a subsection of the search sequence containing M' matching elements would on average be amplified to a probability of $\frac{M'}{M}$. The author states that the total run time of this algorithm is $\mathcal{O}(M \log^3(N) + N^{3/2} \log^2(N) \log(M))$ in the number of elementary quantum operations, which is faster than the classical counterpart $\mathcal{O}(MN^2)$, calculated by the authors when counting the number of Boolean gates that would be used in the classical circuit.

The proposed method seems promising in its complexity, although its usage requires the allocation of M registers capable of holding all N positions of the search pattern, yielding a qubit requirement of at least $M\lceil\log_2(N)\rceil$. This is a rather modest usage for general patterns, and setting a limit of ∼100 qubits for future hardware it would not be unreasonable to have sequences on the size of ∼100-1000 base pairs

Algorithm 2 Closest pattern matching with oracle-based search[31]

```
1: choose $r \in \left[0, \lfloor\sqrt{N-M+1}\rfloor\right]$
2: set $|\psi_0\rangle = \frac{1}{\sqrt{N-M+1}} \sum_{k=1}^{N-M+1} |k, k+1, \dots, k+M-1\rangle$
3: for $i = 1$ to $r$ do
4:     choose $j \in [1, M]$ uniformly
5:     set $|\psi\rangle = I^{\otimes j-1} \otimes U_{p_j} \otimes I^{\otimes M-j} |\psi\rangle$
6:     set $|\psi\rangle = (G \otimes I^{\otimes M-1} |\psi\rangle$
7: end for
8: set $m$ to the result of measurement on the first component of $|\psi\rangle$.
```

and a reference of ∼10. It might be unsuited for usage on the scale of the human genome, but could be applicable for small scale problems, if its speedup is relevant at such a scale. Although the performance would need to be investigated further, before any conclusion of its applicability can be reached. Another thesis provides a more extensive survey of the aforementioned methods, providing detailed circuit usage analysis and simulation implementations, to which the more interested reader is directed[33].

4.1.3 String Matching and Edit Distance

Techniques have also been suggested for solving the problem of exact string matching and edit distance, both directly related to certain scenarios of the alignment problem. Among those is the string matching algorithm in [34] which achieves a complexity of $\tilde{\mathcal{O}}(\sqrt{N} + \sqrt{M})$ in time with $\tilde{\mathcal{O}}$ meaning up to logarithmic factors of the sequence length N and reference length M. This method uses a technique called *deterministic sampling* and a probabilistic oracle capable of checking for a mismatch at a certain position of the search sequence, of which it is proposed to implement using Grover search. Another algorithm works on approximating edit distance between strings, which for some score functions is equivalent to global alignment. The achievement of this algorithm is to approximate the edit distance in subquadratic time, reportedly in $\mathcal{O}(N^{1.858})$ or $\mathcal{O}(N^{1.781})$ depending on the approximate factor and assuming both strings are the same length N. Both algorithms are heavily reliant on the efficient implementation of certain oracles used in Grover search, but also on statistical methods which would be too involved to cover. The interested reader is therefore directed to the respective papers[34, 35].

4.2 Fourier-Based Methods

An algorithm based on the quantum Fourier transform was invented by Schutzhold[36] and interpreted for the purpose of local sequence alignment by Prousalis and Konofaos[37], which performs analysis of dot matrices, mentioned in section 2.2.2. Assuming a dot matrix $A_{ij} \in \{0, 1\}$ is given for a pair of sequences of length N and M, with the goal being to identify nearly diagonal lines in this matrix, one encodes the dot matrix into quantum memory as the state

$$|\psi\rangle = \frac{1}{\sqrt{\sum_{i=1}^{N}\sum_{j}^{M} A_{ij}}} \sum_{i=0}^{N-1}\sum_{j=0}^{M-1} A_{ij} |i\rangle \otimes |j\rangle = \frac{1}{\sqrt{\rho NM}} \sum_{k=1}^{\rho NM} |z_k\rangle \,, \tag{4.4}$$

with ρNM being the number of ones in the matrix A_{ij} and $z_k = (Ni+j)_k$ denoting the linear index position of the kth non-zero point. The intuition behind creating this state is that a line of certain inclination and significant length in the matrix causes a periodic spike when viewing the matrix A_{ij} as a function of the linear index $A(z) = A(Ni + j)$. By performing Fourier analysis on such a function, the relevant features of the line could be extracted from the peaks of the obtained spectrum. The quantum advantage then lies in the exponentially more efficient application of the Fourier transform that is the Quantum Fourier Transform (QFT)

$$QFT(|z\rangle) = \sum_{\omega=0}^{NM-1} e^{2\pi i \frac{z\omega}{NM}} |\omega\rangle \,, \tag{4.5}$$

which can be performed in $\mathcal{O}(\log_2^2(NM))$ simple operations[26]. However, as one is interested in merely measuring this state to obtain information on the frequency peaks, the algorithm can be optimized to only use $\mathcal{O}(\log_2(NM))$ simple operations. Iterating this transform for a number of times Ω to get representative data of the spectrum, the inclination and width of the line can be extracted by analysis borrowed from Laue diffraction. The location of the pattern however, cannot be found by this analysis alone. This would require splitting up the dot matrix and repeating this procedure for each part, until one is certain to have isolated any line that was previously discovered.

So in summary, if one would use a simple interval search strategy for locating a line, this method could run in $\mathcal{O}(\Omega \log_2^2(NM))$ simple operations. Additionally, the bit usage is only that required to encode the dot matrix in a quantum state, which would be merely $\log_2(NM)$ qubits. Essentially, one could use sequences on the order of $2^{50} \approx 10^{15}$ base pairs in this method if allowing for $\sim$100 qubits on a future quantum computer. What one should note however, is that it is not certain how to prepare the state (4.5) efficiently. The authors more or less state this as a black-box operation, BB, acting as

$$BB\left(|i\rangle \otimes |j\rangle \otimes |0\rangle\right) = |i\rangle \otimes |j\rangle \otimes |A_{ij}\rangle\,, \tag{4.6}$$

for which one could initialize the state as a uniform superposition of all $|i\rangle\,|j\rangle$ states, which can be done in $\log_2(NM)$ Hadamard gates, and then perform a postselection, i.e. deciding measurement, on the third register. If the measurement result is the state $|1\rangle$, the state should be prepared as in (4.5). While Schutzhold[36] suggests a quantum optical implementation of this blackbox, it has not yet been tested and it is unclear how such an operator should be implemented efficiently on other quantum computing platforms. Another difficulty is the Fourier analysis to be performed after obtaining frequency peak measurements. As the dot matrix is not very likely to contain just one simple and straight line, the spectrum analysis would have to be sufficiently robust to extract the dominant patterns, despite other noise-inducing patterns being present. Therefore, the method would be promising to use for local alignment purposes, but further development is required both in the analysis and implementation before being ready to be practically applicable.

4.2.1 Multiple Alignment Algorithm

Apart from the Grover-based and Fourier-based pattern and string matching algorithms, there has been a suggestion of an algorithm to solve the multiple alignment problem using only unitary operations at its core. This algorithm, proposed by Iriyama and Ohya[38], works as presented in algorithm 3. Having L sequences of length N, assuming that gaps have been added at the ends to make their lengths uniform, the algorithm would use L registers, $\{Q_i\}^L$, of length N to represent the alignment and two additional registers holding a threshold score $|s\rangle$, and a single qubit b to be used for computation. Alignments are then represented by letting positions at which a character is placed have the value $|1\rangle$ and indel positions $|0\rangle$, one example being

$$\begin{array}{llllllll} S_1: & A & T & C & G & - & - \\ S_2: & A & - & G & T & - & - \\ S_3: & T & G & - & A & A & - \end{array} \longrightarrow \begin{array}{l} Q_1: |111100\rangle \\ Q_2: |101100\rangle \\ Q_3: |110110\rangle \end{array}\,.$$

The algorithm would then use various simply implemented unitaries for permutation of these bits, such as to permute the indels in the alignment. Once enough permutations have been performed, a black-box unitary U_C calculates the score of the states and compares it to the chosen threshold score s. If the score of the state is larger, then b is for that superposed state set to $|1\rangle$, and otherwise stays as $|0\rangle$. If then a postselection is performed on the bit b and the resulting state $|1\rangle$ is obtained, then a measurement on the registers $\{Q_i\}^L$ gives one of the desirable permutations.

Depending on the threshold, this method could produce an approximate or optimal alignment, and would reportedly do so in $\mathcal{O}(NL)$ operations while using $NL + \log_2(s) + 1$ qubits. The number of qubits may be too large to be run on a universal computer and for the scale of genomic sequences, but the complexity is at least sub-exponential in the number of sequences and therefore low in comparison to other methods for multiple alignment. It is however unclear what factors are accounted for in calculating this complexity. The authors propose using a very advanced black-box unitary for the latter part of their algorithm and provide explicit details neither of the decomposition nor of their obtained complexity. In the worst case, the design of their unitary will be exponential in the number of bits, making the complexity exponential as well. The method was however proposed as an initial suggestion for algorithms to tackle

Algorithm 3 Quantum Multiple Sequence Alignment

```
 1: Prepare initial state $|\psi\rangle = \left(\bigotimes_{i=1}^{L} Q_i\right) \otimes |s\rangle \otimes |0\rangle$ according to given sequences and indel positions
 2: Set $|\psi\rangle = \bigotimes_g H_g |\psi\rangle$                ▷ Create uniform superposition over all indel qubits
 3: for every sequence $i = 1$ to $L$ do
 4:     For every gap $g_j$, make a pair with random character $c_j$                ▷ Choose permutations
 5:     Set $|\psi\rangle = \prod_{\text{all pairs } (g_j, c_j)} CX_{g_j, c_j} |\psi\rangle$                ▷ Permute characters $c_j$ with chosen gaps $g_j$
 6: end for
 7: Apply $U_C$ to registers $\{Q_i\}^L$, $U_C |\psi\rangle = |f(Q_1, \dots, Q_L)\rangle \otimes |s\rangle \otimes |0\rangle$.    ▷ Calculate score function of
    permutations
 8: Apply $U_S$ to marking bit $|b\rangle = |0\rangle$
 9: Measure $|b\rangle$
10: if $|b\rangle = |1\rangle$ then
11:     Measure registers $\{Q_i\}^L$
12: end if
```

this alignment problem and shows the capacity of quantum parallelism given efficient unitaries. In future endeavors, one could perhaps provide an improved version of this method or such an efficient unitary implementation.

5 Ising Formulation of Multiple Sequence Alignment

The field of quantum optimization has attracted substantial amounts of attention during the recent years. Quantum annealing devices are being realized by companies such as D-Wave, and new optimization methods such as the Variational Quantum Eigensolver (VQE)[39] and the Quantum Approximate Optimization Algorithm (QAOA)[40] are being developed and studied for the purpose of solving hard problems both in the field of chemistry and in the realm of NP-hard combinatorial problems[41]. Especially among the NP-complete problems, many have been translated into finding the minimum energy of an Ising Hamiltonian[42], with hopes of finding efficient solutions or accurate approximate solutions through quantum methods. Within the field of bioinformatics, the problem of multiple sequence alignment was presented in section 2.2.3, which had been shown to be NP-complete if one uses the SP alignment scoring scheme and which method of exact solving typically uses space exponential in the number of sequences. As such, it could benefit from future methods for quantum optimization.

There have been announcements of using quantum hardware to solve the MSA problem more efficiently with quantum annealing software[43], and prototype tools have been developed to perform the encoding of MSA problems into Ising formulations, to run on quantum annealing hardware[44]. However, to the extent of this author's knowledge, there is a lack of published work describing such translations and demonstrating its capabilities. Therefore, in this thesis, such an Ising formulation is derived and its correctness is demonstrated through simulations and quantum hardware execution.

In Ising formulations of combinatorial optimization problems[42], one writes the objective function of the problem, which is assumed to be expressible as a function taking a string of classical bits $z = z_1, \dots, z_N$, describing the configuration of the solution,

$$f_{opt} = f(z_1, \dots, z_N), \tag{5.1}$$

and translates it into a Hamiltonian taking a number of quantum spin operators

$$H_{opt} = H(\sigma_1^z, \dots, \sigma_N^z, \sigma_1^x, \dots, \sigma_N^x), \tag{5.2}$$

such that finding the ground state solves the original problem. The only criteria then being that the Hamiltonian eigenvalues $\{\lambda_i\}$ correctly reflect the costs of the various problem configurations, or that the minimum eigenvalue is given for the spin values representing the optimal solution to the problem. However, most optimization problems are defined with a set of rules for valid configurations, which are expressed as constraints on the objective function. These are usually formulated as *hard constraints*, in which the optimization procedure is forced to respect these conditions while solving, thus always keeping them satisfied. Unfortunately, quantum annealing hardware is generally not capable of respecting hard constraints. They must therefore be reformulated as *soft constraints*, which the optimizer is allowed to violate, but given a penalty for doing so. As such, one introduces a penalty Hamiltonian, H_P, and formulates the total Hamiltonian as

$$H_{tot} = AH_{opt} + BH_P \tag{5.3}$$

where A and B are real constants. The design of this Hamiltonian should ideally be such that H_P is positive semidefinite, having positive eigenvalues for all states being invalid solutions. The coefficients A and B are chosen such that the penalty is sufficiently severe that a change in spin, violating the conditions, can never yield a total negative energy. If one can identify ΔH_{opt} as the largest reward, and ΔH_P as the smallest penalty, from changing a single spin, then it suffices to fulfill

$$B\Delta H_P \geq A\Delta H_{opt} \iff \frac{B}{A} \geq \frac{\Delta H_{opt}}{\Delta H_P}. \tag{5.4}$$

In the MSA problem formulation, there is a natural objective function, the SP score (2.5), which will be the main Hamiltonian representing the problem. However, the SP score itself does not enforce any conditions on the configuration of the sequence base elements. Also, for any configuration of the alignment, the base pairs are required to respect the order of the initial sequence, and there should also not occur any mixing of base pairs between sequences. These are conditions to be enforced in addition to the optimization. Moreover, one may have to enforce the existence of each base pair, depending on the type of encoding being used. To summarize, the following kinds of Hamiltonians have to be designed for the MSA problem

1. The SP score Hamiltonian, H_{SP}
2. The order Hamiltonian, enforcing the sequence order in the alignment, H_O
3. The validity Hamiltonian, H_{val}, which penalizes whatever states may be invalid for the given encoding.

5.1 Maximum Trace Formulation

A natural consideration in designing the score Hamiltonian, is the existing mapping of MSA to the Maximum Trace (MT) problem formulated as a graph problem[45]. In the MT formulation of the MSA problem, to put it simply, you are given a fully connected undirected weighted graph of nodes, each supposed to represent a base pair among the sequences. The solution consists of choosing edges (forming a trace) that maximizes the total weights, while making sure that the chosen edges, i.e. element pairs, respect the sequence order. This is achieved by introducing directed edges between the nodes in the same sequences, and reformulating the problem as forming a trace containing no cycles[46].

Looking at this formulation from a practical standpoint, a trivial choice of encoding would be to let each edge be represented by a spin $s_{i,j}$. Then the score Hamiltonian could be constructed as a sum, weighting each edge appropriately,

$$H_{SP,trace} = \sum_{i,j} w_{i,j} \left(\frac{s_{i,j} + 1}{2} \right). \tag{5.5}$$

But this kind of encoding would quickly require large amounts of spins, as there would have to be one spin per possible alignment pair. This would, for L sequences of largest length N, grow as $\mathcal{O}(N^L)$, quickly becoming infeasible for the scale of modern annealing devices. Furthermore, to enforce the ”no cycles” condition, auxiliary bits encoding a node order would be needed, making the formulation even more expensive. Looking at the encoding cost of this formulation, one realizes that if the encoding should incorporate even the most extreme alignment possibility (e.g. every sequence aligning at opposite ends), then the maximum trace formulation of H_{SP} is required. Although, if one could restrict the alignment opportunities to those which are less extreme, then the spin usage can be considerably reduced. This is what is proposed in the “Column alignment” formulation.

5.2 Column Alignment Formulation

In this formulation, the alignment is viewed less as "pairing" of base pairs, and more in terms of "aligned placement". Here, a number of columns C is allocated for base pair/indel placement, and all elements placed in the same column are then decidedly aligned, as illustrated in fig. 10. Such a method would only

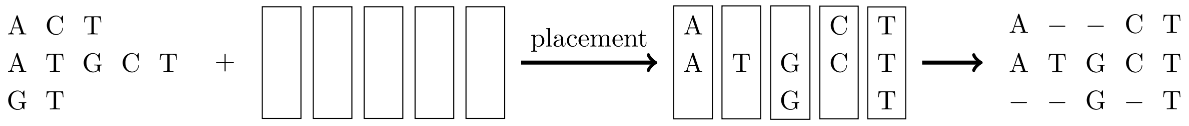


Figure 10: Alignment through column placement. Sequence elements are aligned by placement in pre-allocated columns, respecting the original order. Empty positions indicate indels.

require an encoding capturing the placement of base pairs into mentioned columns. Thus, letting a base pair in sequence s, at position n in the original sequence, have a spin $s_{s,n,i}$ indicating whether the base pair is placed in column i or not, is all that is required to encode the problem. Note however, that we shall not use the direct spin variables $s_{s,n,i} \in \{-1, 1\}$, but their Boolean counterpart $x_{s,n,i} = \frac{1+s_{s,n,i}}{2} \in \{0, 1\}$, and corresponding spin operators which shall be used interchangeably with the variables for convenience. While this restricts the alignment configurations to those achievable with the allocated columns, the spin usage is also limited to the amount of base pairs and their placements. Specifically, having C columns with L sequences of lengths $\{n_1, \ldots, n_L\}$, the spin usage S is

$$S = C \sum_{i=1}^{L} n_i, \tag{5.6}$$

which, with the maximum number of base pairs N, can be seen as $S \in \mathcal{O}(CLN)$, i.e. being polynomial in size of the problem parameters. Although it may be more common for MSA problems to choose the number of columns as $N + G$, with G being a number of extra gaps allowing for more flexible alignment configurations. Then the bit usage would be $\mathcal{O}(L(N^2 + NG)) = \mathcal{O}(LN^2)$ with G expected to be small in comparison to the size of the largest sequence. Note in particular that the size requirements in terms of spins is merely linear in the number of sequences, in contrast to the classical space requirements being exponential.

In constructing the SP score Hamiltonian, it is assumed that matching scores for each pair of relevant sequence elements are given as a weight matrix $\omega_{(s_1,n_1,s_2,n_2)}$ according to some defined matching scores and that there also exists an indel matching penalty g. Then, it can be realized that this Hamiltonian will consist of two main terms, the term responsible for rewarding matches and penalizing mismatches, and the term for indel matching which will have to designed differently due to the spins only encoding the positions of base pairs. For the former term, a sufficient design can be given as

$$H_{SP,1} = \sum_{s_1<s_2} \sum_{n_1,n_2} \sum_{i} \omega_{(s_1,n_1,s_2,n_2)} x_{(s_1,n_1,i)} x_{(s_2,n_2,i)}. \tag{5.7}$$

The summation is performed first over all pairs of sequences (without double counting), then over all pairs of elements in the current chosen sequences. The final sum is taken over all columns, adding a term weighted by the reward/penalty for the specific base pairs, which is appropriately equal to the weight if these base pairs are placed in the same column, and 0 otherwise.

Assuming a simple linear gap penalty for the alignment scoring, the Hamiltonian penalizing the alignment with gaps, if $g \neq 0$ and constant, can be written as

$$H_{SP,2} = g \sum_{s_1,s_2} \sum_{n_1} \sum_{i} x_{(s_1,n_1,i)} \left(1 - \sum_{n_2} x_{(s_2,n_2,i)} \right), \tag{5.8}$$

where the summation is again taken over all sequences, s_1 and s_2, and the following summations over sequence elements n_1 and n_2 in sequences s_1 and s_2 respectively. Since the chosen encoding merely captures the placement of base pairs, the only way to describe the alignment with an indel would be having no alignment with anything from the other specific sequence, which is captured by the term $1 - \sum_{n_2} x_{(s_2,n_2,i)}$.

However, one should note that such a term is valid only for the case of $\sum_{n_2} x_{(s_2,n_2,i)} \leq 1$, i.e. when only one base pair per sequence can occupy a specific column, for otherwise it would become beneficial for the optimizer to match with indels. This condition will be enforced by a term constructed below, however one can comment that the term could be squared, as

$$\left(1 - \sum_{n_2} x_{(s_2,n_2,i)}\right)^2, \tag{5.9}$$

which would instead increase the penalty for such invalid placements. However, it would make (5.8) non-quadratic and therefore require simplification, as only quadratic forms are supported on certain hardware, e.g. D-Wave's annealers.

As discussed before, a validity Hamiltonian, which fixes the conditions of valid placement is necessary. The conditions for the spin configuration to be valid in the sense of being able to represent an alignment configuration are

1. That each base pair exists in exactly one column (assuming one does not allow for explicit deletion of base pairs)
2. That a column position for a specific sequence is occupied by a maximum of one base pair.

The first condition can be enforced by

$$H_{val} = \sum_{s,n}\left(1 - \sum_{i} x_{(s,n,i)}\right)^2, \tag{5.10}$$

where the first sum is over every sequence and contained base pair. The term can be seen to be positive for when base pairs are missing placements ($\forall i, x_{(s,n,i)} = 0$), and the penalty increases quadratically with the deviation from valid configurations. The second condition is more concisely described together with the term enforcing the sequence order. Assuming that complete deletion of base pairs is not allowed in the alignment, one can restrict the placements of the base pairs in each sequence by

$$H_O = \sum_{s,n}\sum_{i\leq j} x_{(s,n,j)}x_{(s,n+1,i)}, \tag{5.11}$$

making the alignment both respect the sequence orders (for $i \neq j$), and limiting the placement onto the same column (for $i = j$). The intuition behind this Hamiltonian is illustrated in fig. 11, where it can be

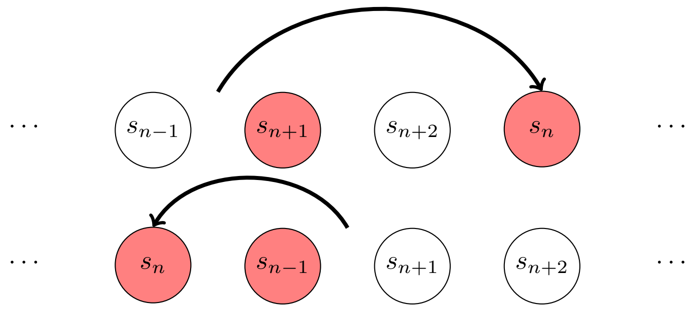


Figure 11: Illustration of order violation, and elements which interaction gives a penalty (red) according to (5.11). Both examples illustrate the penalty interaction activating when the placement order does not follow the sequence order. The above example shows that only interaction to nearest sequence element activates the penalty, even if order is violated for more than two elements (s_{n+1} and s_{n+2} both before s_n).

seen that an alignment featuring a sequence element at an earlier position will give a penalty if it is placed farther to the right of a later occurring base pair. One can note how it does not increasingly penalize even though non-neighboring base pairs are placed out of order. However, it is sufficient that a penalty is given by only one interaction.

To summarize, the Hamiltonian solving the problem of multiple sequence alignment by column placement of the involved sequence elements is given with (5.7), (5.8), (5.10), and (5.11), and weights according to (5.3), is

$$\begin{aligned} H = A &\sum_{s_1<s_2}\sum_{n_1,n_2}\sum_{i}\omega_{(s_1,n_1,s_2,n_2)}x_{(s_1,n_1,i)}x_{(s_2,n_2,i)} \\ + Ag &\sum_{s_1,s_2}\sum_{n_1}\sum_{i}x_{(s_1,n_1,i)}\left(1-\sum_{n_2}x_{(s_2,n_2,i)}\right) \\ + B &\sum_{s,n}\left(1-\sum_{i}x_{(s,n,i)}\right)^2 \\ + B &\sum_{s,n}\sum_{i\le j}x_{(s,n,j)}x_{(s,n+1,i)}. \end{aligned} \tag{5.12}$$

The only thing that remains is choosing the appropriate Hamiltonian coefficients according to (5.4). In (5.12), the case where the validity and order terms have the same coefficient has been chosen for simplicity. Inspecting (5.12), it can be seen that changing a spin from a valid to an invalid state, would incur a non-zero penalty of at least B from validity and order preserving terms. Meanwhile, the largest reward one might get from the SP score Hamiltonians depends on the match matrix ω and how the spin is flipped. The match matrix can contain both negative and positive elements, and the gap penalty is also capable of contributing with positive and negative values depending on the type of violation. Therefore, a more detailed analysis is required.

First, isolate the contributions of an arbitrary single spin $x_{(c,m,k)}$, which can be achieved by constricting indices in the sums. This gives one constricted term for every spin factor, and can be written

$$\begin{aligned} H_{SP,(c,m,k)} = A &\sum_{s_1<s_2}\sum_{n_1,n_2}\sum_{i}\delta_{(s_1,n_1,i),(c,m,k)}\omega_{(s_1,n_1,s_2,n_2)}x_{(s_1,n_1,i)}x_{(s_2,n_2,i)} \\ + A &\sum_{s_1<s_2}\sum_{n_1,n_2}\sum_{i}\delta_{(s_2,n_2,i),(c,m,k)}\omega_{(s_1,n_1,s_2,n_2)}x_{(s_1,n_1,i)}x_{(s_2,n_2,i)} \\ + gA &\sum_{s_1,s_2}\sum_{n_1}\sum_{i}\delta_{(s_1,n_1,i),(c,m,k)}x_{(s_1,n_1,i)}\left(1-\sum_{n_2}x_{(s_2,n_2,i)}\right) \\ - gA &\sum_{s_1,s_2}\sum_{n_1}\sum_{i}\delta_{i,k}\delta_{s_2,c}x_{(s_1,n_1,i)}\sum_{n_2}\delta_{n_2,m}x_{(s_2,n_2,i)}, \end{aligned} \tag{5.13}$$

which by straightforward index constriction and symmetry arguments can be simplified to

$$\begin{aligned} H_{SP,(c,m,k)} &= Ax_{(c,m,k)}\sum_{s,n}\omega_{(c,m,s,n)}x_{(s,n,k)} \\ &+ gAx_{(c,m,k)}\left(L-2\sum_{s,n}x_{(s,n,k)}\right), \end{aligned} \tag{5.14}$$

Now, imagine the spin $x_{c,m,k}$ is flipped, i.e. changed by an amount $\Delta x \in \{-1,1\}$. The corresponding change in the SP Hamiltonian will then be

$$\Delta H_{SP} = A\Delta x\left[gL+\sum_{s,n}(\omega_{(c,m,s,n)}-2g)x_{(s,n,k)}\right]. \tag{5.15}$$

Choosing the sign convention such that $A>0$, then $g>0$ according to its definition as a penalty and the greatest reward is obtained for the minimal (most negative) value of ΔH_{SP} possible for the values of Δx, which with N_{tot} as the total number of base pairs, is bounded according to

$$\Delta H_{SP} \ge A\min\begin{bmatrix} -N_{tot}\max\limits_{s_1,n_1,s_2,n_2}(2g-\omega_{(s_1,n_1,s_2,n_2)})-gL, \\ gL+N_{tot}\min\limits_{s_1,n_1,s_2,n_2}(\omega_{(s_1,n_1,s_2,n_2)}-2g) \end{bmatrix} = A\Sigma. \tag{5.16}$$

It also becomes clear from this expression, since $L \leq N_{tot}$, that ΔH_{SP} can be negative for an arbitrary spin flip, as long as the matching reward is chosen appropriately non-positive. Knowing this, the coefficients of the objective Hamiltonian terms can be chosen according to

$$|\Delta H_P| \geq |\Delta H_{SP}| \iff B \geq A\,|\Sigma|\,. \tag{5.17}$$

However, this bound is somewhat conservative since the estimation made the assumption that all spins $x_{(s,n,k)} = 1$, which is an unlikely scenario. Therefore, there could be a more heuristic but practical bound to be chosen if interaction strength needs to adapt to the limitations of the solving device.

5.3 Simulation Results

To demonstrate the correctness of the constructed Hamiltonian (5.12), it is given as input to a simulated annealing sampler implemented in the Ocean SDK package, provided by D-Wave[47]. To use the Ocean software, the problem has to be formulated in QUBO form, i.e.

$$H = \vec{x}^T Q \vec{x}, \tag{5.18}$$

where Q is a matrix containing the coupling constants and $\vec{x}$ is a vector of spin operators. As per design, our Hamiltonian can be written in such a form without simplification, and is thus straightforward to use. Section 10.3 shows the code written in Python that displays the construction of the Hamiltonian, the calculation of the matching matrix and input into the D-Wave Ocean SDK function, which is used in following tests.

The alignment score is calculated using costs as summarized in table 3. As written, the number of alignment columns is chosen as the number of characters in the longest sequence, N_{max}. Given the costs

Parameter	Value
Match cost	-1
Mismatch cost	1
Gap penalty, g	0
Number of columns, C	N_{max}

Table 3: Parameters used in alignment tests

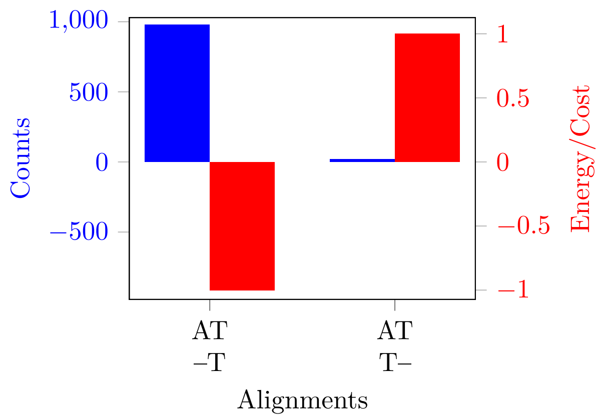
and rewards for matching, setting the first Hamiltonian coefficient to $A = 1$ in (5.12), then according to (5.16) the penalty coefficient can be chosen as

$$B = g(2N_{tot} + L) + N_{tot} \max \omega, \tag{5.19}$$

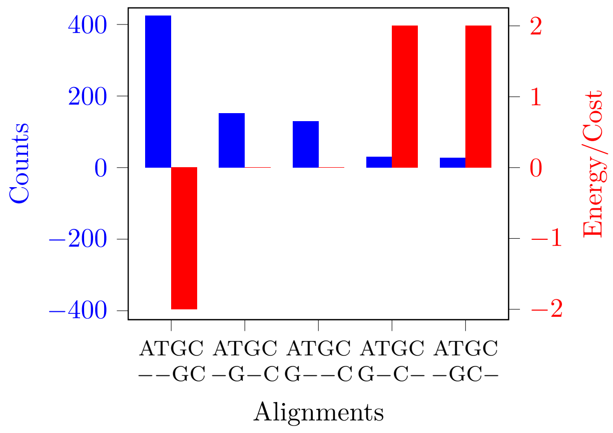
which for $g = 0$ reduces to $B = N_{tot} \max \omega$. $g = 0$ was chosen because of this bound which is seen to be lower than for the $g \neq 0$ case, and could therefore be more suitable for running on quantum hardware limited in the interaction strength.

In fig. 12, the amount of samples obtained from the simulated annealer of different alignments, along with the score of each alignment, are shown for various sequences. The first plot, fig. 12a, shows that the Hamiltonian is constructed correctly for the most trivial case and that the annealer can find the optimal solution. This stays true for increasingly larger and more difficult alignment problems, in the sense of longer and more numerous sequences. One can note how in fig. 12c the annealer manages to almost equally sample the two existing optimal solutions and give them majority, despite there being no alignment with all matching base pairs, in contrast to fig. 12b. An alternative would have been to violate the initial sequence order, however that scenario was not among any of the most populated results, showing the correctness of the penalty Hamiltonian. In figs. 12d and 12e, the number of sequences are increased to test the validity beyond pair-wise alignment and it can be seen that the Hamiltonian energies correspond to the correct SP scores for the parameters in table 3 and that the optimal solution is the most populated here as well. Although one relevant comment is that both invalid spin configurations and alignments violating sequence order were present during the sampling, but that their counts were far too negligible to include among the results in fig. 12.

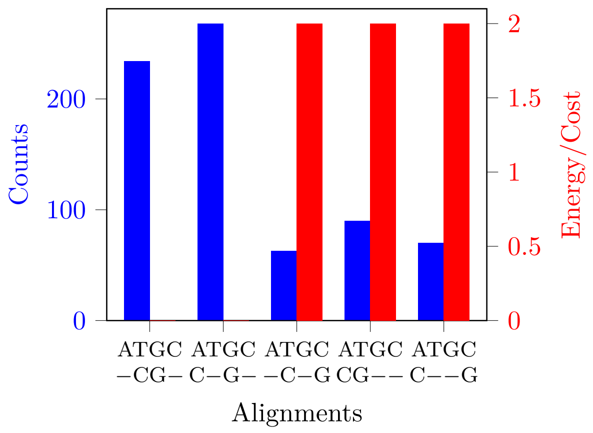
The constructed Hamiltonian has been shown to encode the MSA problem correctly for a collection of trivial test cases, as used by a simulated annealer. Solving larger problems is not attempted, as the purpose



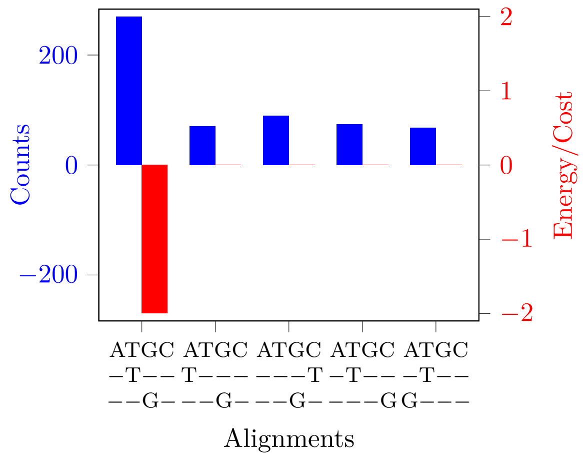
(a) Alignment results for sequences {AT, T}.



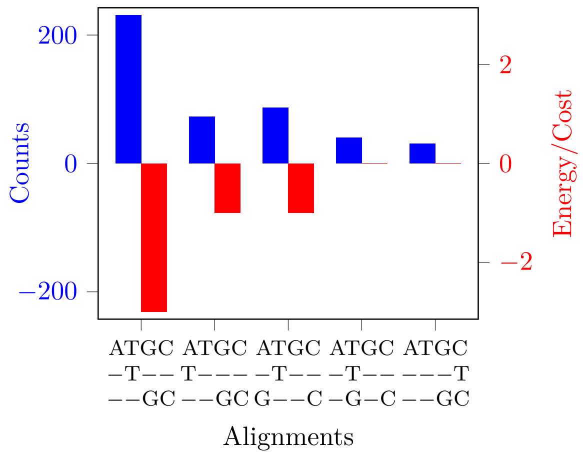
(b) The five most occurring alignments for sequences {ATGC, GC}.



(c) The five most occurring alignments for sequences {ATGC, CG}.



(d) The five most occurring alignments for sequences {ATGC, T, G}.



(e) The five most occurring alignments for sequences {ATGC, T, GC}.

Figure 12: Alignment results as gathered by simulated annealing with the constructed Hamiltonian, measuring 1000 samples. Blue bars (left) showing the amount of obtained samples corresponding to the given alignment, and red (right) showing corresponding Hamiltonian energy, i.e. the alignment score favoring lower values.

of these tests were to demonstrate the correctness of the Hamiltonian. The aim of this construction is not to attempt solving complicated alignment problems or going to the limit of combinatorial optimization using Ising Hamiltonians, it is merely to showcase one such construction as a first step in future research on this topic. However, the question remains as to how well the Hamiltonian formulation functions on real quantum hardware.

5.4 D-WAVE Annealing Results

In this section, the results of running the MSA Ising formulation on a D-wave quantum annealer will be showcased. In these tests, the same scoring parameters as in table 3 and choice of coefficients (5.19) will be used in constructing the Hamiltonians. However, instead of supplying the QUBO formulated problem to a simulated annealing sampler in the Ocean software, it will be given to one of D-waves quantum annealing processors, the DW2000Q system. As is briefly mentioned in section 3, a D-Wave's quantum annealer is built to support quadratic quantum interactions across a pre-constructed network of spins, called the chimera graph and as such, the network of interactions in the specific problem needs to be embedded onto this graph. For these test cases, it was not attempted to do this by hand. Instead, an automatic, although heuristic, embedding software provided in the Ocean SDK package was used. The code is shown in section 10.3 with the same script as in previous simulations, except for the change of an execution parameter.

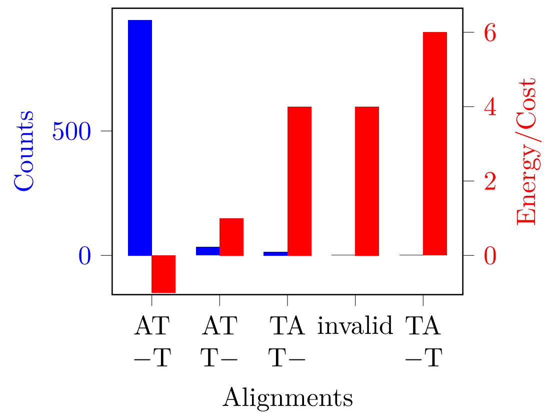
In fig. 13, results of sampling from the D-Wave machine is shown for various sequences. Starting lightly from the sequences {AT, T}, in fig. 13a, the samples are shown to be centered at the optimal alignment as expected. There are however some invalid and order violating terms being measured which were not present in simulations, although negligible. After the most trivial sequences were tested, a test of the sequences {ATC, T} was performed. The result of this test is not shown in any plot as the quantum annealer failed to sample even a single valid configuration among the 1000 samples taken. Therefore, accepting two alignment columns to be the limit for the formulation on a real machine, attention was instead turned to testing the limit in the number of sequences. Figures 13b to 13d together show alignment results for an increasing number of sequences. Figures 13b and 13c display excellent results in both having a minimum number of suboptimal results and having no or a tiny amount of order violation. In fig. 13d however, the limit is reached. While it obtains an adequate amount of counts for the optimal alignment, it is outvoted by some invalid spin configuration. It should be noted that as there are several possible invalid configurations, the "invalid" category is calculated as the maximum among all such results, and there could therefore be more counts in this category altogether, than what is shown in fig. 13d.

To investigate reasons for failure in detail in the long sequence case and in fig. 13d, falls outside the scope of this thesis. However, some obvious reasons which could be mentioned are low interaction coupling precision in the annealing hardware, and unsatisfactory embedding onto the chimera graph. The former cause is because of the hardware limitations of the quantum annealer, limiting the range of coupling constants which can be encoded. When a problem requires interaction beyond this limit, they are scaled down to fit the limited range[25]. In doing so, the hardware precision can cause weaker interactions to be neglected or reversed, especially if the problem requires both weak, precise and strong interactions simultaneously. As for the latter cause, the interactions used in the final embedding may differ from what is given as input, not only due to the previously mentioned scaling, but also because of how the interactions are divided onto the chimera graph. The difficulty of finding an embedding also increases with the number of spins to couple. Table 4 summarizes the necessary range of interactions to encode the

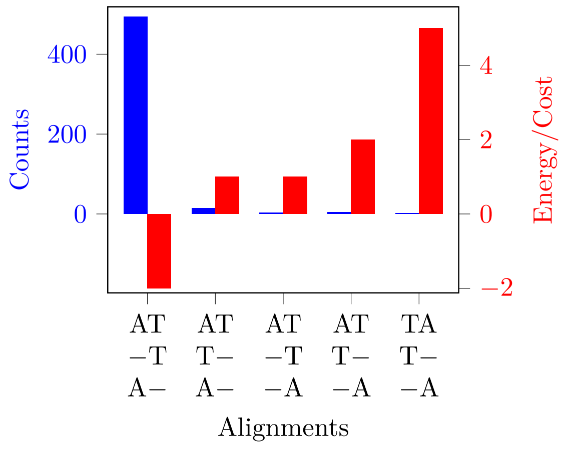
Test	Sequences	min $\|J\|$	max $\|J\|$	Spins
1	AT, T	1.0	10.0	6
2	ATC, T	1.0	8.0	12
3	AT, T, A	1.0	14.0	8
4	AT, T, T, A	1.0	18.0	10
5	AT, T, A, T, A	1.0	22.0	12

Table 4: Minimum and maximum interaction coefficients used in the MSA Ising formulation for the sequences tested on a D-Wave annealer.

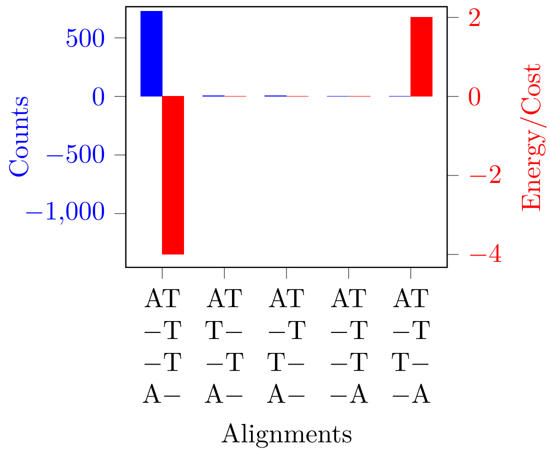
alignment problems that have been tested, together with the total spin usage in encoding the alignments.



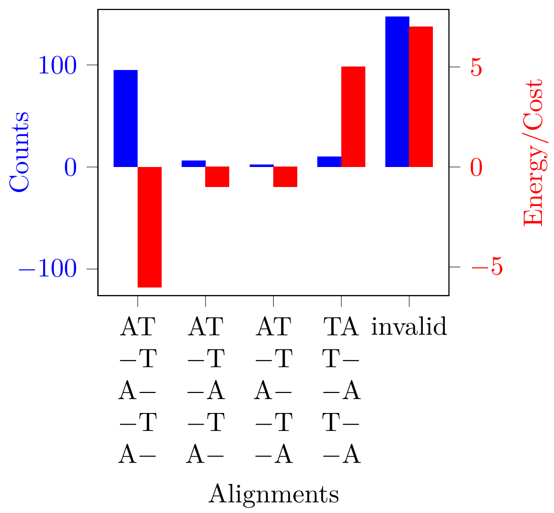
(a) The five most occurring alignments of {AT, T}.



(b) The five most occurring alignments of {AT, T, A}.



(c) The five most occurring alignments of {AT, T, T, A}.



(d) The five most occurring alignments of {AT, T, A, T, A}.

Figure 13: Alignment results as sampled from a D-Wave quantum annealer, using 1000 samples. Blue bar (left) shows the amount of samples corresponding to the given alignment, and red (right) shows corresponding Hamiltonian energy, i.e. alignment score favoring lower values.

What can be observed from the failed test cases 2 and 5 is that too large a range in coupling strength is unlikely to be the cause as test 2, which failed completely in finding a solution, had the smallest range of all. Instead, what both failures seem to have in common is the spin usage, which is simultaneously the same and the largest among all tests. This indicates that an inefficient chimera graph embedding could be the reason for failure, however it is difficult to draw a conclusion with the relatively few tests performed.

6 Quantum Associative Memories

Quantum Associative Memories (QuAMs) are quantum memory structures developed under the umbrella of neural networks. As Grover search and other quantum computational ideas began to emerge, the computational learning community tried to develop quantum structures making use of the storage and parallelism capabilities that quantum mechanics seemed to offer. The particular structure discussed here is an associative memory, or otherwise called content-addressable memory in the sense that an element of the memory or database is returned based on its similarity to a query, along with any possibly related information (illustrated by fig. 14). Therefore, the memory could be used for several pattern-matching purposes, making use of the unique interference of quantum mechanics to find matches in the memory. The

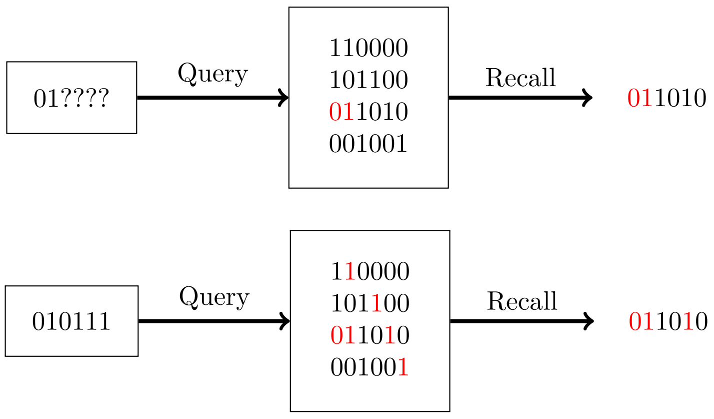


Figure 14: Examples of associative memory recall. Query elements can be accessed based on partial information (above), or based on similarity to query pattern (below).

quantum associative memories which shall be reviewed here have also been studied for their application in track pattern matching in particle detectors[48] and bioinformatic purposes[33], although focusing on the genetic assembly problem in sequencing.

6.1 Grover-Based Quantum Associative Memory

6.1.1 Ventura Model

The QuAM data structure was first proposed by Ventura and Martinez[49], who saw the potential of Grover search as a retrieval mechanism for a completing memory. The first part of the procedure is construction of the quantum memory. For this, Ventura and Martinez[50] proposed an algorithm capable of creating a uniform superposition state of the memory states, using a number of operations linear in the number of memory entries. This algorithm was later reused, generalized and simplified by following works[49, 51]. A description of the algorithm as written in [51] is shown in algorithm 4, but a simple demonstration will be given, with explanation of each operation.

To start with, three main registers are allocated for use; the input register, to hold the query state, the memory register to hold the memory states and the auxiliary register to be used in computations. The quantum state for storing n-bit patterns, with input qubits $\{p_i\}_{i=1}^{n}$, two auxiliary qubits u_1 and u_2, and memory qubits $\{m_i\}_{i=1}^{n}$, is structured as

$$|\psi\rangle = |p_1, \ldots, p_n; u_1, u_2; m_1, \ldots, m_n\rangle . \tag{6.1}$$

If one is aiming to store the first of N patterns $\left|p^1\right\rangle = \left|p_1^1, \ldots, p_n^1\right\rangle$ into the memory register, it is assumed the qubits are initially put into the state

$$\left|\psi_0^1\right\rangle = \left|p_1^1, \ldots, p_n^1; 01; 0_1, \ldots, 0_n\right\rangle, \tag{6.2}$$

by trivial initialization operators on the input and auxiliary register. To ease the understanding of this procedure, the reader should be aware that the auxiliary bit u_2 acts as a flag for stored $u_2 = 0$ and processing states $u_2 = 1$. First, the state is acted on by a series of 2-bit Toffoli gates to copy the pattern p^1 from the input in the empty memory register, i.e.

$$\left|\psi_1^1\right\rangle = \prod_{j=1}^{n} \mathrm{TOFFOLI}_{p_j u_2, m_j} \left|\psi_0^1\right\rangle = \left|p_1^1, \ldots, p_n^1; 01; p_1^1, \ldots, p_n^1\right\rangle, \tag{6.3}$$

given that $u_2 = 1$ also. Thereafter, the operation

$$\left|\psi_2^1\right\rangle = \prod_{j=1}^{n} X_{m_j}\, \mathrm{CNOT}_{p_j m_j} \left|\psi_1^1\right\rangle, \tag{6.4}$$

is applied, which has the effect of giving all bits of a state in the memory register the value 1, if that state was identical to the input state. This is inevitably what happens for our state, after applying (6.3) for $u_2 = 1$. Then, the first auxiliary bit is flipped for the all-one memory state

$$\left|\psi_3^1\right\rangle = \mathrm{TOFFOLI}_{m_1 \ldots m_n, u_1} \left|\psi_2^1\right\rangle, \tag{6.5}$$

marking the state for further processing. After doing this, the auxiliary register is operated on by the controlled unitary

$$\left|\psi_4^1\right\rangle = CS^{p}_{u_1 u_2} \left|\psi_3^1\right\rangle, \tag{6.6}$$

with the unitary

$$S^{\ell} = \begin{pmatrix} \sqrt{\frac{\ell-1}{\ell}} & \frac{1}{\sqrt{\ell}} \\ \frac{-1}{\sqrt{\ell}} & \sqrt{\frac{\ell-1}{\ell}} \end{pmatrix}. \tag{6.7}$$

At this point, the auxiliary bit state $|u_1 u_2\rangle = |11\rangle$, after application of (6.6), will split into two parts $|u_1 u_2\rangle = \frac{1}{\sqrt{p}}|10\rangle + \sqrt{\frac{p-k}{p}}|11\rangle$, one which will store the processed pattern ($u_2 = 0$), and another on which to continue input processing ($u_2 = 1$). Finally, doing the inverse of operations (6.3), (6.4) and (6.5), one obtains

$$\left|\psi^1\right\rangle = \frac{1}{\sqrt{p}}\left|p^1; 00; p^1\right\rangle + \sqrt{\frac{p-1}{p}}\left|p^1; 01; 0_1, \ldots, 0_n\right\rangle, \tag{6.8}$$

The input register can then be reset with a new pattern and the process repeated from (6.3) to (6.8) for each pattern. Doing so, although modifying (6.6) for the current (kth) pattern as

$$\left|\psi_4^k\right\rangle = CS^{p+1-k}_{u_1 u_2}\left|\psi_3^k\right\rangle, \tag{6.9}$$

it can be seen that the final result is

$$\left|\psi^N\right\rangle = \left|p^N; 00\right\rangle \otimes \left(\frac{1}{\sqrt{N}} \sum_{i=1}^{N} \left|p^i\right\rangle\right) = \left|p^N; 00\right\rangle \otimes |M\rangle, \tag{6.10}$$

where the memory register state is in an equal superposition of the patterns, as desired. Having constructed the memory state, summarized in algorithm 4, the next step is retrieval of the matching memories.

The mechanism for memory retrieval uses the potential of Grover search to apply its search algorithm only to specific parts of binary strings in a database. The algorithm would then ideally find a substring in the database that matches the query, and by having stored the entire strings in quantum memory, the rest of the string would follow through quantum entanglement of the involved registers. To give a detailed description, focusing on the memory retrieval procedure, a substring oracle I_τ is constructed which inverts the phase of a chosen state τ, which is given by the query and acts on a chosen subset of register qubits. As an example, suppose there are five register qubits, $\{q_1, q_2, q_3, q_4, q_5\}$, and the query could be given as

Algorithm 4 QuAM Memory construction

```
1: procedure MEMORY CONSTRUCTION($p^1, p^2, \dots, p^N$)
2:     Initialize registers $|\psi\rangle = |p_1, \dots, p_n; u_1, u_2; m_1, \dots, m_n\rangle \to |0_1, \dots, 0_n; 0, 1; 0_1, \dots, 0_n\rangle$
3:     for every pattern $p^k$ do
4:         $|p_1, \dots, p_n\rangle \to |p_1^k, \dots, p_n^k\rangle$          ▷ Load pattern into input register
5:         for every qubit j do
6:             $\mathrm{TOFFOLI}_{p_j u_2, m_j} |\psi\rangle$          ▷ Duplicate pattern into memory
7:             $X_{m_j}\,\mathrm{CNOT}_{p_j m_j} |\psi\rangle$          ▷ Transform memory state into bit difference state
8:         end for
9:         $\mathrm{TOFFOLI}_{m_1 \dots m_n, u_1} |\psi\rangle$          ▷ Flip work bit for term to add to memory
10:        $CS_{u_1 u_2}^{N+1-k} |\psi\rangle$          ▷ Put the pattern in memory storage
11:        $\mathrm{TOFFOLI}_{m_1 \dots m_n, u_1} |\psi\rangle$          ▷ Restore work bit
12:        for every bit j do
13:            $\mathrm{CNOT}_{p_j m_j} X_{m_j} |\psi\rangle$          ▷ Restore memory state to normal form
14:            $\mathrm{TOFFOLI}_{p_j u_2, m_j} |\psi\rangle$          ▷ "uncopy" pattern from memory
15:        end for
16:    end for
17:    return $|M\rangle = |m_1, \dots, m_n\rangle$
18: end procedure
```

$|\tau\rangle = |?0?11\rangle$, with *wildcard* characters on position 1 and 3, meaning the search is not performed on qubits q_1 and q_3. That implies that I_τ would be constructed as

$$I_\tau |q_1, q_2, q_3, q_4, q_5\rangle = \begin{cases} -|q_1, q_2, q_3, q_4, q_5\rangle, \text{ if } |q_2, q_4, q_5\rangle = |011\rangle \\ \ \ |q_1, q_2, q_3, q_4, q_5\rangle, \text{ otherwise} \end{cases}. \tag{6.11}$$

In the Grover search, the standard Grover operator, as presented in section 3,

$$G = |s\rangle\langle s| - I, \tag{6.12}$$

with the uniform superposition $|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=1}^{2^n} |i\rangle$, is used to evolve the states after marking. These two operators are applied to a superposition consisting of the states representing the bit strings in memory. Thus far, the procedure is not much different from that of a Grover search on an arbitrary initial state, which has been shown to have a limited success probability[21]. What distinguishes the presented QuAM from a normal Grover search, is how the problem of limited success probability is addressed. An extra phase rotation

$$I_P = I - 2P_M, \tag{6.13}$$

using the projector onto the memory states $P_M = \sum_{m \in M} |m\rangle\langle m|$, replaces the oracle at the second Grover iteration. This phase rotation inverts the phase of all states in memory, without concern for if it is marked or not and this supposedly reinforces the retrieval of marked memory states by making the non-marked non-memory (spurious) states have the same amplitude sign as the non-marked memory states. After the application of this memory-based phase inversion, the Grover search is performed as usual for a determined number of iterations T. In total, the Ventura model retrieval operator is

$$R = \left(\prod_{i=1}^{T} G I_\tau \right) G I_P G I_\tau. \tag{6.14}$$

For this method, it is shown that the proposed associative memory has a maximal probability of retrieving a marked pattern

$$P_{max} = 1 - (N - p - r_0)\left|\ell_0 - \overline{\ell}\right|^2 - (p - r_0)\left|\ell_1 - \overline{\ell}\right|^2, \tag{6.15}$$

where N is the total number of basis states, p the number of stored states, r_0 and r_1 being the number of marked states in memory and outside of memory, respectively[49]. l_0 and l_1 are the total amplitude of the unmarked spurious and non-spurious states, respectively, evaluated after line 9 in algorithm 5. The

Algorithm 5 Ventura model retrieval

```
1: procedure Retrieval($\tau$)
2:     Construct phase inverter for partial search $I_\tau$
3:     Construct Grover operator $G = |s\rangle\langle s| - I$
4:     Construct memory phase inverter $I_M = I - 2\sum_{m\in M} |m\rangle\langle m|$
5:     Initialize state $|\psi\rangle \to |M\rangle = \frac{1}{\sqrt{N}}\sum_{m\in M} |m\rangle$
6:     $I_\tau |\psi\rangle$                ▷ Invert phase for matching non-wildcard characters
7:     $G |\psi\rangle$
8:     $I_M |\psi\rangle$                ▷ Invert phase for all memory states
9:     $G |\psi\rangle$
10:    for i from 1 to $T$ do                ▷ Grover iteration phase
11:        $GI_\tau |\psi\rangle$
12:    end for
13:    Measure $|\psi\rangle$ in computational basis
14:    return $|\psi\rangle$
15: end procedure
```

barred quantity, $\bar{\ell}$ is correspondingly the average amplitude of unmarked spurious states in the initial superposition. The authors also show that to get as close to this maximum probability as possible, one should iterate the Grover search according to

$$T = \text{ROUND}\left[\frac{\frac{\pi}{2} - \arctan\left(\frac{\bar{k}}{\bar{\ell}}\sqrt{\frac{r_0+r_1}{N-r_0-r_1}}\right)}{\arccos\left(1 - 2\frac{r_0+r_1}{N}\right)}\right], \tag{6.16}$$

where $\bar{k}$, in correspondence to $\bar{\ell}$, is the average amplitude of marked states after applying the initial procedure of the search, and ROUND is the rounding function to closest integer. An additional remark is that one can determine $\bar{k}$ and $\bar{\ell}$ as

$$\bar{k} = \frac{8(p-2r_1)}{N}\left(1 - \frac{p+r_0}{N}\right) + \frac{r_1}{r_0+r_1} \text{ and} \tag{6.17}$$

$$\bar{\ell} = \frac{4(p-2r_1)}{N}\left(\frac{N+p-r_0-2r_1}{N-r_0-r_1} - \frac{2(p+r_0)}{N}\right) - \frac{p-r_1}{N-r_0-r_1}, \tag{6.18}$$

and as such, information of the amount of marked states in and outside of memory (r_0 and r_1) is all the extra information needed to maximize the retrieval success probability.

6.1.2 Distributed Queries

The work of Ventura and Martinez has been developed further by Ezhov, Nifanova and Ventura[52], who aimed at making the QuAM structure capable of *fuzzy matching*, in addition to its pattern completing properties. Their approach mainly consists of extending the concept of the marking oracle I_τ, allowing it to not only invert the phase of a single state, but provide arbitrary phase change to states according to a chosen matching priority.

A step by step comparison with the method of Ventura and Martinez starts with the construction of the memory state. While the extension in this work allows for the same type of memory state structure as in section 6.1.1, the authors propose constructing the memory with an *exclusion rule*, initializing the memory state as

$$|\not{M}\rangle = \frac{1}{\sqrt{2^n - N}}\sum_{m\notin M} |m\rangle, \tag{6.19}$$

i.e. the complement of the actual memory. This has been shown to not only decrease the probability of irrelevant retrieval[53], but also reduce the amount of information required to calculate the amount of search iterations, which is why it will be the only choice considered henceforth.

When it comes to the search procedure, the article introduces new types of phase change and diffusion operators, O_Q and D. The phase change is defined through a query state $|Q\rangle = \sum_i b_i |i\rangle$ as

$$O_Q = I - 2|Q\rangle\langle Q|, \tag{6.20}$$

such that it inverts the phase about the given state. The query form proposed by Ezhov was a binomial distribution on the Hamming distance between patterns, i.e.

$$b_i = \sqrt{a^{d_H(q,p_i)}(1-a)^{n-d_H(q,p_i)}}, \tag{6.21}$$

for a query center q with n bits and parameter $0 < a \leq \frac{1}{2}$.
The diffusion operator is chosen as

$$D = 2\left|\cancel{M}\right\rangle\!\left\langle\cancel{M}\right| - I, \tag{6.22}$$

which in relation to the Grover operator G that inverts around the average, instead inverts the amplitudes around the overlap to the exclusive memory state. With these operator choices, the retrieval process consists of a standard Grover evolution procedure, with the total retrieval operator defined as

$$R = \prod_{\tau=1}^{\Lambda} DO_Q, \tag{6.23}$$

as summarized in algorithm 6.

Algorithm 6 Memory retrieval with distributed query

```
 1: procedure Retrieval
 2:     Construct exclusive memory $\left|\cancel{M}\right\rangle$
 3:     Construct query operator $O_Q = I - 2\left|Q\right\rangle\!\left\langle Q\right|$
 4:     Construct diffusion operator $D = 2\left|\cancel{M}\right\rangle\!\left\langle\cancel{M}\right| - I$
 5:     Initialize search state $\left|\psi\right\rangle \to \left|\cancel{M}\right\rangle$
 6:     for i from 1 to $\Lambda$ do
 7:         $DO_Q\left|\psi\right\rangle$
 8:     end for
 9:     Measure $\left|\psi\right\rangle$ in computational basis
10:     return $\left|\psi\right\rangle$
11: end procedure
```

The contribution of the distributed query is demonstrated in the analytic derivation of state amplitudes. It is shown that state amplitudes in the search state $\left|\phi\right\rangle = \sum_i a_i \left|i\right\rangle$, can be written on the form

$$a_i(\tau) = A_i \cos(\omega\tau + \delta_i), \tag{6.24}$$

as a function of the number of iterations τ. The angular frequency of amplitude oscillation ω is decided by the overlap between memory and query state, as

$$\omega = 2\arcsin\left(\left\langle Q\middle|\cancel{M}\right\rangle\right). \tag{6.25}$$

Meanwhile the maximum amplitudes, A_i, and phase shifts, δ_i, are given by the details of the distributed query state, memory state $\left|\cancel{M}\right\rangle$ and initial conditions $\left|\psi^0\right\rangle$. More specifically, they are given as

$$A_i = B\frac{\sqrt{(b_i - m_i\sin(\omega/2))^2 + \left(\frac{m_i}{2\left\langle Q\middle|\cancel{M}\right\rangle}\sin\omega\right)^2}}{\cos(\omega/2)} \quad \text{and} \tag{6.26}$$

$$\delta_i = \phi - \frac{\omega}{2} + \arccos\left(\frac{b_i - m_i\sin(\omega/2)}{\sqrt{(b_i - m_i\sin(\omega/2))^2 + \left(\frac{m_i}{2\left\langle Q\middle|\cancel{M}\right\rangle}\sin\omega\right)^2}}\right), \tag{6.27}$$

with query amplitudes, b_i, and memory amplitudes, m_i, where B and ϕ are parameters to be found from the initial state before iterations. It can be noted that the amplitudes are non-zero even for states not in memory, meaning the retrieval of spurious states is a possibility. Also note how having no overlap between query and the exclusive memory $\langle Q|\cancel{M}\rangle = 0$ causes the expression to become ill-defined, and in fact makes the retrieval process (6.23) into an identity operation, and thus completely meaningless for retrieval. Therefore, this method requires a certain amount of mismatch between query and memory, and in fact, it is what determines the speed of retrieval, due to expression (6.25). Although the more significant quality is observed for the states which are in memory, i.e. $m_i = 0$. For those states, the quantities (6.26) and (6.27) simplify to

$$A_i = B\frac{b_i}{\cos(\omega/2)}, \tag{6.28}$$

$$\delta_i = \phi - \frac{\omega}{2}, \tag{6.29}$$

showing that maximal amplitudes become proportional to the associated query amplitudes. This allows the query to become a description of the retrieval priority, as query components with low or zero amplitude will retain low or zero amplitude in memory retrieval.

As for the number of iterations, Λ, it is clear from the form of the solution (6.24) and (6.25), that prior knowledge on the similarity between memory and query is required to know this value exactly. However, the authors argue that given that the number of patterns in memory is small compared to the number of all possible patterns, 2^n with n as the bit string length, one can use the approximation

$$\omega = 2\arcsin\big(\langle Q|\cancel{M}\rangle\big) \approx 2\arcsin\left(\frac{1}{\sqrt{2^n}}\sum_{i=1}^{2^n} b_i\right). \tag{6.30}$$

Assuming that the angular oscillation frequency is known, then the fact that the amplitudes of states in memory would be on the form

$$a_i = A_i \sin(\omega\tau), \tag{6.31}$$

because of their initial condition of $a_i(\tau = 0) = 0$ for the choice of exclusive memory, makes it certain that the maximum probability of memory retrieval is obtained at

$$\Lambda = \text{ROUND}\left[\frac{2\pi}{\omega}\left(\frac{1}{4} + \alpha\right)\right] \tag{6.32}$$

where α is an arbitrary integer to be chosen such that Λ is as close to an integer as possible.

6.1.3 Further improvement

Further improvements upon the distributed query model were proposed by Njafa, Nana and Woafo[54]. They modified the retrieval procedure by applying an additional operator I_M, similarly to what was originally done in section 6.1.1. With algorithm 7 showing the proposed modified procedure, the authors let I_M be either:

1. A memory-based state inversion similar to that in section 6.1.2,

$$I_M = I - 2P_M, \tag{6.33}$$

   where $P_M = \sum_{m\in M} |m\rangle\langle m|$ is the projector onto the memory state subspace.

2. A query-based phase shift, different from that of O_Q. Although in their paper, they limited themselves to a simple change in parameters in the query proposed by Ezhov, Nifanova and Ventura, (6.21).

While the second option was shown through numerical simulations to reduce the number of iterations to reach maximum correct retrieval probability, the first option appeared to significantly improve this probability.

The state evolution during the iterative phase of the procedure is still the same as in section 6.1.2, and thus the expressions (6.26) and (6.27) should still hold. But in contrast to initially having the exclusive memory state, as in the Ezhov model, if one applies the procedure of algorithm 7 up until the iterative phase from line 11, one obtains the "pre-iterative" initial state

$$|\psi^0\rangle = |\cancel{M}\rangle + 2\langle Q|\cancel{M}\rangle (2P_M - I)|Q\rangle. \tag{6.34}$$

Algorithm 7 Improved retrieval with distributed query

```
1: procedure RETRIEVAL
2:     Construct exclusive memory $|\cancel{M}\rangle$
3:     Construct query operator $O_Q = I - 2|Q\rangle\langle Q|$
4:     Construct diffusion operator $D = 2|\cancel{M}\rangle\langle\cancel{M}| - I$
5:     Construct additional operator $I_M$
6:     Initialize search state $|\psi\rangle \to |\cancel{M}\rangle$
7:     Set $|\psi\rangle = O_Q|\psi\rangle$
8:     Set $|\psi\rangle = D|\psi\rangle$
9:     Set $|\psi\rangle = I_M|\psi\rangle$
10:    Set $|\psi\rangle = D|\psi\rangle$
11:    for i from 1 to $\Lambda$-1 do
12:        Set $|\psi\rangle = DO_Q|\psi\rangle$
13:    end for
14:    Measure $|\psi\rangle$ in computational basis
15:    return $|\psi\rangle$
16: end procedure
```

One thing to note, is that the state is now dependent on the projection of $|Q\rangle$ onto the memory subspace. Because of this, the amplitude of memory states in this initial state is non-zero. While this apparently improves the probability of successful retrieval, the iteration estimation done in the Ezhov model is no longer applicable. Furthermore, the authors do not discuss the issue of knowing the number of iterations, apart from stating that they use a choice of

$$\Lambda = \text{ROUND}\left[\frac{2\pi}{\omega}\left(\frac{1}{4}+\alpha\right)\right] - 1, \tag{6.35}$$

with the same quantities as defined in section 6.1.2.

Although, following the analysis done by Ezhov et al. [52] one can use (6.34) to find the exact analytic expressions of the memory states. As an exclusive memory is used, the simplified expressions (6.28) and (6.29) hold. To determine the iterations, one mainly needs the initial phase ϕ, which according to [52] is determined by the initial state $|\psi_0\rangle$ before iteration, by

$$\phi = \arctan\left(\frac{1}{\sin\omega} - \frac{1}{\tan\omega} - 2\frac{\langle\psi_0|\cancel{M}\rangle}{\langle\psi_0|Q\rangle}\langle Q|\cancel{M}\rangle\right), \tag{6.36}$$

which with the initial state (6.34) can be written

$$\phi = \arctan\left(\frac{1}{\sin\omega} - \frac{1}{\tan\omega} - \frac{2-4|\langle Q|\cancel{M}\rangle|^2}{4\langle Q|P_M|Q\rangle - 1}\right). \tag{6.37}$$

If this quantity is known, then the optimal number of iterations can be determined from (6.24) as

$$\Lambda_{opt} = \text{ROUND}\left[\frac{\alpha\pi-\phi}{\omega} + \frac{1}{2}\right], \tag{6.38}$$

with an arbitrary positive integer α.

6.1.4 Efficiency Considerations

Memory construction for the initial model would require $\mathcal{O}(Nn)$ simple gates for N patterns of bit length n, while using $2(n+1)$ bits for the registers, not counting any eventual extra work bits for efficient implementation. But as the newly developed methods make use of an exclusive memory, one could question the efficiency of the initially proposed algorithm due to the fact that one needs to process $2^n - N$ states instead, making the worst case gate usage $\mathcal{O}(n2^n)$. Although, research has been done on techniques of state deletion[55, 56, 57], where a number of orthogonal states $|\tau_1\rangle, \ldots, |\tau_m\rangle$ are deleted from an arbitrary superposition state $|\gamma\rangle$. It could prove more efficient to fill the memory completely and delete undesired states, than to fill the memory one state at a time. As described in [56], the method is Grover search inspired, with iteration operator

$$S = -UI_0U^{-1}I_m, \tag{6.39}$$

where U is the state construction operator, $|\gamma\rangle = U|0_1, \ldots, 0_n\rangle$, and I_0 and I_m are operators changing the phase of the all-zero state and of states not to delete, respectively. (6.39) is then applied for a certain number of iterations which depends on the number of states being deleted and the specific change in phase caused by I_0 and I_m. However, as long as $N \leq \frac{3}{4} \cdot 2^n$, then the paper shows that the states could be deleted in just one application of S, by using phase matching techniques[58] developed for improving Grover search. Using this method, one could start from the uniform superposition state, which is easily constructed as

$$|s\rangle = \prod_{i=1}^{n} H_i |0_1, \ldots, 0_n\rangle, \tag{6.40}$$

where H_i acts on the ith qubit, meaning the U in (6.39) consists of only n Hadamard gates. As the I_0 operator could be made through a, somewhat advanced but efficiently implementable, n-bit controlled unitary gate, the main difficulty in using this method lies in the I_m operator. But if that operator could be applied efficiently or even sub-exponentially in the number of bits, it could improve memory construction efficiency for many practical cases, while also keeping the bit usage down to those needed for storing the n bit memory states (not including eventual extra bits for computations).

The time efficiency of state retrieval, is decided mainly by how efficiently the involved operators can be implemented, and how many iterations have to be performed for optimal retrieval probability. Construction of arbitrary operators is a general issue for most quantum algorithms, and the retrieval algorithms for the associated memories are no exceptions. The operations in the original version by Ventura and Martinez do not deviate from Grover search in any significant regard and as such is not as challenging to construct apart from the marking oracle. However, it is a different story for the methods based on distributed queries and onward. These methods include generalizations of the Grover search operations, allowing more arbitrary phase change during the evolution. Operators allowing such evolution are more complex, and while they are explicitly known in their unitary matrix form, decomposition could result in highly inefficient circuits. An alternative technique which could be applicable is given by a trick for density matrix exponentiation used in quantum tomography techniques[59]. Using k copies of a state $|\psi\rangle$, one can approximate the application of $\exp(-i|\psi\rangle\langle\psi|)$ on a state $|\phi\rangle$ by repeated application of

$$\mathrm{tr}_\psi \left(e^{-\frac{i}{k}S} |\psi\rangle \otimes |\phi\rangle\right) \tag{6.41}$$

where S is the SWAP gate acting on the $|\psi\rangle$ and $|\phi\rangle$ states, of which exponentiation can be done efficiently. Creating appropriate states $|\psi\rangle$ one could implement arbitrary phase changes based on the eigenvalues of $|\psi\rangle\langle\psi|$, but k would none-the-less scale as $k \sim 1/\epsilon$ with ϵ as the error and put more demand on the efficiency of state preparation.

As has been seen in sections 6.1.2 and 6.1.3, the number of iterations needed for retrieval is dependent on information about the overall match between the given query and the memory. When the query consists of a single basis state, then strategies suggested for Grover search with multiple solutions[22] could be attempted to obtain this before retrieval. However, in the methods adapted for fuzzy matching, it is more difficult, as the entire overlap between query and memory is needed. While there are methods for estimating the overlap between states, such as the swap test[60] used in quantum machine learning, and quantum amplitude estimation[61], they tend to be either purely statistical and require many copies of the states, or make use of oracles and other hard-to-construct black box operators. The approximation (6.30) proposed by Ezhov, Nifanova and Ventura is therefore an important factor in the practical usage of these methods, as only a priori information would be used. Following the approximation, the number of iterations would be

$$T \in \mathcal{O}\left(\frac{1}{\omega}\right) = \mathcal{O}\left(\frac{1}{\arcsin\left(\frac{1}{\sqrt{2^n}}\sum_{i=1}^{2^n} b_i\right)}\right), \tag{6.42}$$

from (6.32), making the query distribution the deciding factor. However, as it is formed as a quantum state, and therefore bounded by $\sum_i b_i^2 = 1$, the amplitude sum is bounded such that

$$T \leq \mathcal{O}\left(\frac{1}{\arcsin\left(\frac{1}{\sqrt{2^n}}\right)}\right) \approx \mathcal{O}(\sqrt{2^n}), \tag{6.43}$$

in the large n case. Still, it is not certain that the proposed approximation will be satisfactory for practical applications. Regrettably, the improved method of section 6.1.3 requires even more knowledge of the memory and query similarity, as is understood by the non-trivial overlap dependence in (6.34), which is not as easily approximated. While the authors propose what appears to be a heuristic approach, (6.35), for calculating the number of iterations, and reportedly have applied it with adequate results in their following works[62], it is not obvious how well it would function in practice, and especially on other forms of queries than those investigated so far.

When considering the case of distributed queries, there is also the issue of failing, in the sense of retrieving spurious states or memories with low priority. As the algorithm itself provides no way of checking whether a spurious or undesired state has been retrieved, caution is required. And while the analytic expressions in section 6.1.2 can describe the theoretical amplitude of states, it is difficult to analyze how the amplitude of spurious states will behave when implementing approximations and heuristics or using other kinds of queries than first suggested. One would ideally want a query prioritizing only the desired kind of states, using a query with no spurious overlap $\langle Q|\cancel{M}\rangle = 0$, but this causes the retrieval to fail completely, with no state change being done, thus forcing the query to be designed to have overlap with spurious states. This problem is at the core of the proposed methods, and for any query used, the efficiency of the approximations and heuristic would need to be evaluated.

6.2 Trugenberger Quantum Associative Memory

During the development of associative memories such as the Ventura model, Trugenberger proposed an alternative retrieval mechanism[51, 63]. In his approach, an application of unitary operators designed according to a certain metric, creates a superposition state with amplitudes corresponding to the similarity in the state and the query, according to the given metric. In his works, a Hamming distance metric is used as an example, but the method could be extended to include other metrics as well. For constructing the quantum memory state, the method presented in section 6.1.1 for the Ventura model is used.

Having constructed the memory state, the retrieval process begins by again using three registers. The first two registers are the same as in the memory construction, an input register $|p_1, \ldots, p_n\rangle$ and the memory register $|m_1, \ldots, m_n\rangle$, but lastly there is a register with auxiliary bits $\{b_i\}_{i=1}^{B}$ used in computations. This register can include previous auxiliary bits to save on qubit usage. For simplicity in demonstration, it is assumed that $B = 1$ initially. The retrieval begins by loading the query pattern, $Q = q_1, \ldots, q_n$, into the input register and with the auxiliary bit in the zero state, i.e.

$$|\psi_0\rangle = |q_1, \ldots, q_n\rangle |M\rangle |0\rangle . \tag{6.44}$$

The memory state is then prepared such that the difference between memory and query is easily retrievable. This is done through the same method as in memory construction, (6.4), i.e.

$$|\psi_1\rangle = \prod_{j=1}^{n} X_{m_j}\, \mathrm{CNOT}_{p_j m_j} |\psi_0\rangle . \tag{6.45}$$

The result of this will be, as before, to make the memory state bits $|m_j^k\rangle$ for a stored kth pattern become $|1\rangle$, if for the query state $|q_j\rangle = |m_j^k\rangle$, and $|0\rangle$ otherwise. Thus the memory register will now encode the “difference flags”

$$d_j^k = \begin{cases} 1 & |q_j\rangle = |m_j^k\rangle \\ 0 & \text{otherwise} \end{cases}, \tag{6.46}$$

for the various memories and query pattern, in terms of the bit values. Then, a superposition state in one of the auxiliary bit is made by a Hadamard gate

$$|\psi_2\rangle = H_{b_1} |\psi_1\rangle = \frac{1}{\sqrt{2p}} \sum_{k=1}^{p} |Q\rangle |d_1^k, \ldots, d_n^k\rangle |0\rangle + \frac{1}{\sqrt{2p}} \sum_{k=1}^{p} |Q\rangle |d_1^k, \ldots, d_n^k\rangle |1\rangle . \tag{6.47}$$

Once this stage is reached, the metric unitary $\mathcal{H}$ is applied, which is on the form

$$\mathcal{H} = \exp\left(i\frac{\pi}{2D_{max}}D_m \otimes Z_{b_1}\right), \tag{6.48}$$

where D_m is a metric operator applied on the memory register and D_{max} is its maximum eigenvalue. This is the central operation of the algorithm, which determines the desired amplitude distribution. In the proposed version of this structure, the Hamming distance measure $d_H(q,p)$ was assumed, i.e.

$$D_m = \sum_{k=1}^{n}\left(\frac{Z_k+1}{2}\right), \tag{6.49}$$

with $D_{max} = n$, such that acting on a memory state in its "difference form", it counts the number of ones in the state,

$$D_m\left|d_1^k,\dots,d_n^k\right\rangle = d_H(Q,p^k)\left|d_1^k,\dots,d_n^k\right\rangle = \left[\sum_{j=1}^{n}(1-d_j^k)\right]\left|d_1^k,\dots,d_n^k\right\rangle. \tag{6.50}$$

A decomposition of $\mathcal{H}$ would be needed for actual implementation, and for the Hamming distance metric, this is given by Trugenberger as

$$\mathcal{H} = \left(\prod_{i=1}^{n}\left(CU^{-2}\right)_{cm_i}\right)\left(\prod_{i=1}^{n}U_{m_i}\right), \text{ for } U = \begin{pmatrix}\exp(i\frac{\pi}{2n}) & 0\\ 0 & 1\end{pmatrix}. \tag{6.51}$$

By applying the unitary (6.48) to the obtained state, one gets

$$\begin{aligned}|\psi_3\rangle = \frac{1}{\sqrt{2p}}\sum_{k=1}^{p}\exp\left(i\frac{\pi}{2n}d_H(Q,p^k)\right)|Q\rangle\left|d_1^k,\dots,d_n^k\right\rangle|0\rangle \\ + \frac{1}{\sqrt{2p}}\sum_{k=1}^{p}\exp\left(-i\frac{\pi}{2n}d_H(Q,p^k)\right)|Q\rangle\left|d_1^k,\dots,d_n^k\right\rangle|1\rangle.\end{aligned} \tag{6.52}$$

Now applying the inverse operations of (6.45) and (6.47), the memory register is restored to the memory state, and a new superposition is made from the Hadamard operation on the auxiliary bit, namely

$$|\psi_5\rangle = \frac{1}{\sqrt{p}}\sum_{k=1}^{p}\cos\left(\frac{\pi}{2n}d_H(Q,p^k)\right)|Q\rangle\left|p^k\right\rangle|0\rangle + \frac{i}{\sqrt{p}}\sum_{k=1}^{p}\sin\left(\frac{\pi}{2n}d_H(Q,p^k)\right)|Q\rangle\left|p^k\right\rangle|1\rangle. \tag{6.53}$$

At this point it is clear that Trugenberger has managed to create a desirable state in one term of the superposition, the cosine term. This term exhibits a maximum for $d_H = 0$ and monotonically decreases as d_H increases, reaching a value of zero at $d_H = n$. It is possible to obtain this state by a postselection on the auxiliary bit, expecting the $|0\rangle$ state. However, obtaining the other term is undesirable as it has the exact opposite properties to the cosine state. The probabilities of obtaining either the positive or negative results are

$$P(|b_1\rangle = |0\rangle) = \frac{1}{p}\sum_{k=1}^{p}\cos^2\left(\frac{\pi}{2n}d_H(Q,p^k)\right), \tag{6.54}$$

$$P(|b_1\rangle = |1\rangle) = \frac{1}{p}\sum_{k=1}^{p}\sin^2\left(\frac{\pi}{2n}d_H(Q,p^k)\right), \tag{6.55}$$

which are promptly affected by the overall match between query and database. If there are many memory states which are relatively close to the query state by the defined metric, then the probability will be biased toward (6.54) and vice versa for (6.55).

In later contributions, Trugenberger and Diamantini[63] extend the procedure, by allowing an arbitrary number of auxiliary bits, B, and repeating processes (6.45) to (6.53) for every such bit. The resulting state is then obtained as

$$|\psi\rangle = \frac{1}{\sqrt{p}}\sum_{k=1}^{p}\sum_{\ell=0}^{B} i^{\ell}\cos^{b-\ell}\left(\frac{\pi}{2n}d_H(Q,p^k)\right)\sin^{\ell}\left(\frac{\pi}{2n}d_H(Q,p^k)\right)\sum_{\{J^\ell\}}|Q\rangle\left|p^k\right\rangle\left|J^\ell\right\rangle, \tag{6.56}$$

where $\{J^\ell\}$ stands for all possible permutations of the B auxiliary bits with exactly ℓ bits equal to one. As the optimal distribution resulting from measurement is still obtained for the result $|J^\ell\rangle = |0_1, \dots, 0_B\rangle$, the *recognition probability*, i.e. the postselection success probability, is only changed to

$$P_{rec} = \frac{1}{p}\sum_{k=1}^{p} \cos^{2B}\Big(\frac{\pi}{2n} d_H(Q, p^k)\Big), \tag{6.57}$$

which is observed to be strictly lower than (6.54) for $B > 1$ and decreasing with increasing B, for any memory non-identical to the query due to the narrowing of the distribution toward $d_H = 0$. However, the resulting distribution if postselection succeeds also changes in the same way,

$$|\psi_{post}\rangle = \frac{1}{pP_{rec}}\sum_{k=1}^{p} \cos^{2B}\Big(\frac{\pi}{2n} d_H(Q, p^k)\Big) |p^k\rangle, \tag{6.58}$$

becoming more narrowed toward existing states closer to the query with increasing B. As such, the number of auxiliary bits can be used as a parameter specifying a range for relevant state retrieval, in terms of the introduced metric.

6.2.1 Efficiency Considerations

On the complexity of usage, it is clear that the memory construction is $\mathcal{O}(Nn)$ in the usage of simple gates, as it is the same algorithm as used in section 6.1.1. Although in contrast to the Grover-based memories, the retrieval algorithm initially proposed[51] uses merely $\mathcal{O}(n)$ number of simple gates, all with fairly small elementary gate decomposition if used with the introduced Hamming distance metric. The extended version[63] would, however, increase in complexity to $\mathcal{O}(Bn)$ since the initial algorithm is repeated.

The bit usage is a maximum of $2n+\max(2, B)$ bits considering both memory construction and retrieval, although one then neglects any eventual garbage qubits used for efficient implementation of advanced gates. However, Trugenberger and Diamantini suggest that for the retrieval, the input register could be removed in favor of applying unitary operators designed from the query instead. In doing so, the step (6.45) would be replaced by the operator

$$V^k = \prod_{j=1}^{n} U_j^k, \text{ with } U_j = \sin\Big(\frac{\pi}{2}p_j^k\Big)I + \cos\Big(\frac{\pi}{2}p_j^k\Big)X, \tag{6.59}$$

reducing the bit usage of retrieval by n. Although not discussed by the authors, it is not unthinkable that a similar change could be applied to memory construction. As is explained in section 6.1.1, the input register is merely used for copying and transforming the memory state into its “difference form” in relation to the pattern being processed. The conditional gates making use of the input register could very well be replaced by designing the usage of operators such as those in (6.59). This would lower the overall bit usage to mainly $n + B$ bits, greatly increasing the length of the patterns allowed to be used in the structure. This should however be weighed against the additional preprocessing required in the circuit design.

While the efficiency in gate cost of retrieval is highly beneficial, the postselection could pose a problem for practical use. If the postselection fails and one wishes to complete the retrieval, one is forced to reconstruct the involved state from scratch, starting from the memory state. If this procedure is to replace or aid any kind of database retrieval, then a time complexity of $\mathcal{O}(N)$ is already on the verge of being inefficient without consideration of repeated attempts. Trugenberger therefore considers the application of quantum copying (not exact cloning), of which there are approximate[64] or probabilistic[65] methods which in theory could circumvent a total reconstruction of the memory. However, approximate cloning might not be desirable as the reduced fidelity could greatly affect the retrieval, and so far, probabilistic cloning techniques have yet to demonstrate a significant success rate, with recent studies reaching at most 5%[66].

Another approach, suggested by Trugenberger and Diamanti[63], is the use of amplitude amplification[61] in amplifying the probability of successful postselection. Although, this is only to be used as an alternative to simply repeating T times in hopes for success. The reason is that, if we denote RM as the operator for both memory construction and retrieval state preparation, amplitude amplification would require the application of

$$\mathcal{Q} = -RMS_{\overline{0}}(RM)^{-1}S_B, \tag{6.60}$$

on the finished state (6.56), where S_B would change sign on the desired $|b_1, \ldots, b_B\rangle = |0_1, \ldots, 0_B\rangle$ state and $S_{\bar{0}}$ on the zero state of all registers involved. The $\mathcal{Q}$ operator requires at least twice the number of quantum gates compared to the initial procedure RM and would be applied several times. But on average, $\mathcal{Q}$ would be applied $T_{amp} = 1/\sqrt{P_{rec}}$ times to make successful recognition highly probable, in comparison to without amplification, which would on average require $T = 1/P_{rec}$ attempts. This could make the amplitude amplification the preferred choice when a success is required, even if $\mathcal{Q}$ is more expensive. However, with P_{rec} being determined by an overall match between query and memory, P_{rec} is normally unknown, and so is the number of applications of $\mathcal{Q}$. The amplitude amplification is then reduced to a mere heuristic with no guarantee of success. At most, one could limit the iteration number by

$$P_{rec} \geq \frac{N-1}{N} \sin^{2B}\left(\frac{\pi}{2n}\right) \implies T_{amp} \leq \sqrt{\frac{N}{N-1}} \frac{1}{\sin^B\left(\frac{\pi}{2n}\right)}, \tag{6.61}$$

following the derivation in [63], although this grows quite quickly with increasing n and B.

But reevaluating why the postselection could fail, the reason for low success probability is that the query is too dissimilar to most states in the memory. If one were to prioritize, for example a Hamming distance of up to D, and choose B accordingly, then failing more than a set number of times could be a good indication that no state with $d_H \leq D$ exists in memory. In contrast to the more advanced Grover-based memories, the Trugenberger model gives the advantage that the properties of the postselected state are well known if the measurement result on the auxiliary bits are known. Furthermore, the resulting state (6.58) holds no spurious memories, given that the construction and measurements are done error-free, which is not as certain for the other described models that allow fuzzy matching.

With all things considered, the structure provides a relatively fail-safe, although probabilistic, method of acquiring an amplitude distribution suitable for retrieval, and which prioritizes similarity to a query. One downside is the form of this distribution, which is restricted to the shown form of cosines and sines because of the way interference is made during the retrieval. Although this distribution is restricted, the structure allows some degree of shaping through the number of auxiliary bits, B, capable of setting the priority for metric values. Increasing B would weigh the distribution toward smaller distance values, but as shown by (6.57) and the bound (6.61), increasing B generally makes the retrieval success less likely or harder to achieve for all but the most ideal cases. One is of course free to choose a different metric in (6.48), but the difficulty in doing so involves both finding a metric which would encapsulate the priority in similarity, give an adequate probability of recognition, as well as being easily decomposed.

7 Quantum Applications in Metagenomics

As has been explained in section 2.3, the two vital parts of analysis in a metagenomics project are genetic assembly and binning. There have already been suggestions of using quantum pattern matching in assembly methods[33] to improve efficiency, but binning methods such as Kraken[18] that rely on K-mers of low size in its analysis are worth exploring for their possible quantum improvements.

As first introduced in section 2.3, the framework of Kraken is used for taxonomic binning in the analysis of environment microbial content. It constructs a database of taxonomic IDs, with key entries being K-mers of the corresponding genetic string. As K-mer counting approaches uses K-mers of extremely small length in comparison to a full sequence, e.g. $K \leq 31$ in Kraken, it provides a good starting point for improvements by quantum means. As the procedure of Kraken is based on exact matching of a query K-mer in the database, the first thing to come to mind is if a Grover-based search could be designed for the purpose of replacing this database lookup. While technically possible to implement in a similar fashion to the methods proposed in section 4 for alignment, the disadvantage of such an approach is the problem of state preparation and collapse, as with many of the quantum algorithms that have been reviewed. Without any successful form of quantum copying, the construction of the database state would have to be repeated for every search query. Even if it were possible to use a form of quantum cloning, the search structure in Kraken would be hard to outperform, as it is ordered and extremely efficient. Quantum ordered search has been shown to require fewer comparisons than classical ordered search[67, 68], but only by a constant factor and shares the same bound of $\mathcal{O}(\log_2 N)$ as all search algorithms relying on comparisons. Therefore a quantum exact search might not achieve any extraordinary speedup in comparison to classical algorithms even if it is adequately implemented.

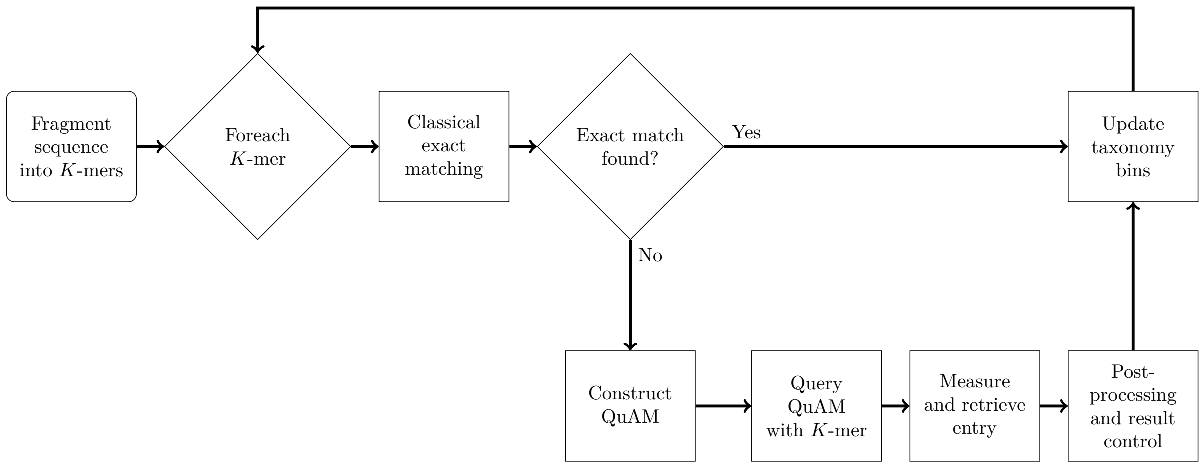


Figure 15: Suggested strategy at improving metagenomic binning techniques with quantum associative memories. Each K-mer is processed trough an exact classical search, but if no exact match is found, it is queried through a QuAM.

It is instead more realistic to propose an augmentation to the Kraken classification, and the most obvious suggestion would be the extension of the exact search to allow for fuzzy searching. For this purpose, the quantum associative memories discussed in section 6 provide good starting points for researching a possible fuzzy retrieval of database entries. Through entanglement, the memory entries could be coupled to states in a taxonomic register storing the various taxonomic tags, generalizing the memory as

$$|M\rangle = \sum_{m \in M} |m\rangle \otimes |T_m\rangle \,, \tag{7.1}$$

where $|T_m\rangle$ would be the tag state for pattern m. The fuzzy retrieval methods presented in sections 6.1 and 6.2 could then act on the memory register only, and measurement would retrieve both the pattern and associated tag. If this could be performed together with methods such as Kraken, then it could provide an improvement in the binning accuracy and add additional resilience to sequencing errors. But usage of these methods requires special considerations for efficiency as discussed in sections 6.1.4 and 6.2.1 and the construction of the database state is generally an expensive procedure. Therefore, it is proposed that one performs the quantum search only when the exact matching fails, as described in fig. 15. That way, one limits the amount of queries to the quantum device to cases where fuzzy matching is required. Additionally, they have to be adapted for storing and retrieving DNA sequences along with any eventual taxonomic tags, according to an appropriate metric.

In the following sections, suggestions for the adaption for DNA sequences are described and the efficiency will be tested to an extent. However, we will not consider the extension of tag states in the memory as it, apart from the additional bit usage, is theoretically not relevant for the efficiency of retrieval.

7.1 Grover-Based Quantum Associative Memory

7.1.1 Adaption for Metagenomics

The Grover-based pattern matching methods which support fuzzy matching are what we shall refer to as the Ezhov model in section 6.1.2 and the improved model in section 6.1.3. Using any of these pattern-matching structures, one must choose an encoding such that sequences can be uniquely translated to bit string patterns, which the structure supports. For the case of DNA, one has the four bases {A,T,C,G}, which by the use of two bits per character can be encoded as

$$E = \begin{cases} \text{A} & \to 00 \\ \text{T} & \to 01 \\ \text{C} & \to 10 \\ \text{G} & \to 11 \end{cases} . \tag{7.2}$$

One can note that this encoding for DNA encapsulates all possible permutations of the two bits, which is ideal. If there existed bit strings not corresponding to any valid patterns, such as for a similar encoding for the 20 amino acids, they would be difficult to exclude from the process when using an exclusive memory, unless at the cost of additional expensive processing, causing there to be an increasing amount of inevitably spurious states.

Then, one must tackle the change in metrics used for similarity in DNA sequences in contrast to simple bit strings. Although there are several metrics one could use, what is considered here is an extension of Hamming distance to a character difference metric acting on the base pairs as characters, i.e. with the use of the encoding (7.2), define the distance between the bit strings $A = a_1, \dots, a_n$ and $B = b_1, \dots, b_n$ as

$$d_c(A,B) = \sum_{i=1}^{n/2} \left(1 - \delta_{a_{2i},b_{2i}}\delta_{a_{2i-1},b_{2i-1}}\right) = \sum_{i=1}^{n/2} \begin{Bmatrix} 1 \text{ if } a_{2i-1}a_{2i} = b_{2i-1}b_{2i} \\ 0\text{otherwise} \end{Bmatrix}, \tag{7.3}$$

with each pair of bits representing a character contributing with a score of 1 only if their characters are not equal.

Then, based on this introduced metric, the shape of the query amplitude distribution should be designed such as to some extent allow for matches with dissimilarities to the query sequence. As we imagine the scenario of running the quantum routine when an exact match is not present, one can start by letting the amplitude for the query center (where $d_c = 0$) be zero. Our suggestion for the remaining states is to use the binomial distribution (6.21) that has been proposed and used with these structures, up to a point $d_c \leq D$. Binning methods in metagenomics could improve when counting entries with some deviation, but there would be a threshold in the metric distance for which an entry should not be included. The parameter D therefore introduces the choice of searching for differing sequences up to a limit. To summarize, the query state for a given query pattern q would be constructed as $|Q\rangle = \sum_{i=1}^{N} Q_q(p_i)\,|p_i\rangle$ with the amplitude distribution defined as

$$Q_q(p) = \left\{ \begin{array}{cc} 0 & p = q \\ C_{(1-\rho)}\sqrt{\gamma^{d_c(p,q)}(1-\gamma)^{n/2-d_c(p,q)}} & 0 < d_c(p,q) \leq D \\ C_\rho & \text{o.w.} \end{array} \right. , \tag{7.4}$$

for n-bit patterns, i.e. $n/2$ length sequences, where C_ρ and $C_{(1-\rho)}$ are normalization constants chosen such that unwanted states for which $d_c > D$ only receive a fixed probability ratio of ρ in the distribution. The reason for this is simply to avoid scenarios where the evolution fails because the query has no overlap with the exclusive memory state. Although more exactly, in adhering to the normalization condition of $\sum_p Q_q(p)^2 = 1$, the constants should be chosen as

$$C_\rho = \sqrt{\frac{\rho}{\sum_{k=D+1}^{n/2}(L-1)^k\binom{n/2}{k}}}, \quad C_{(1-\rho)} = \sqrt{\frac{1-\rho}{\sum_{k=1}^{D}\gamma^k(1-\gamma)^{n/2-k}(L-1)^k\binom{n/2}{k}}}, \tag{7.5}$$

where L stands for the number of letters in the alphabet, i.e. $L = 4$ for DNA. An example of this distribution is illustrated in fig. 16, where the zero amplitude at the center, the binomial shape and the change at the limit $d_c = D$ is shown.

7.1.2 Performance Tests

Although the design of the encoding and associated query is well motivated, as mentioned in sections 6.1.4 and 6.2.1, the probability of success and the different types of failures depend on the shape of the query and its correspondence to the memory, and there is also the question of how well the necessary approximations and heuristics work together with this design. In order to test this, the Ezhov and improved model are implemented in Python using Qutip[69] to simulate the quantum objects and operations (see section 10.1). Using these simulations, memory retrieval should be tested both for the different types of methods, sequence lengths, memory states and search criteria (e.g. D).

First, the ideal retrieval probability is tested, which will be calculated as the maximum probability obtained during a limited number of iterations, in order to first measure the efficiency of retrieval without relying on approximations and heuristics. As for this iteration limit, an upper limit of N, for N memory patterns, is chosen, but if a local maximum cannot be found within this interval, the upper limit is extended

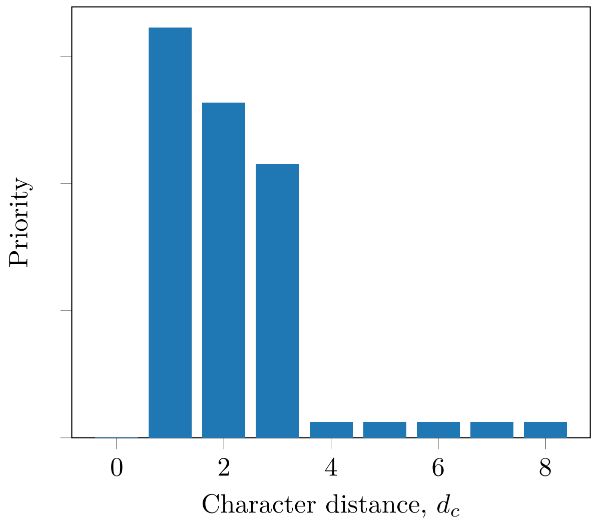


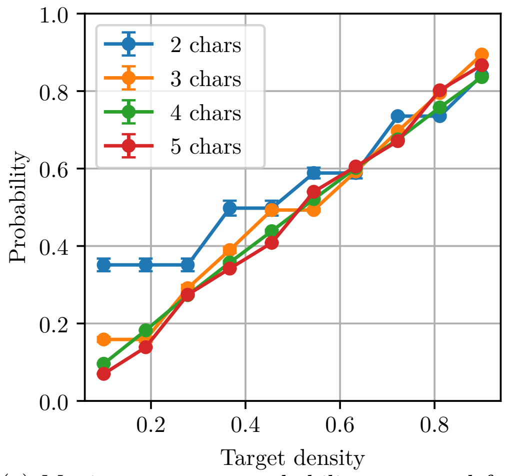
Figure 16: Illustration of proposed query distribution, using 8-character sequences, a binomial parameter of $\gamma = 0.4$, a probability ratio $\rho = 0.1$ and character search limit $D = 3$.

by N until one is found. This performance test is executed using the proposed query (7.4) with binomial coefficient $\gamma = 0.4$ and with a spurious ratio of $\rho = 0.1$, and will mainly test the success probability as a function of length of the sequences, the density of relevant solutions in memory and the search limit D. Success probabilities may also depend on the number of spurious memories, and exactly what states are in the memory could have some effect on the result. These quantities are therefore randomly varied during each test run, and averaged over, as to present the data in terms of the more relevant quantities. To show the dispersion of these values, each data point is plotted together with its standard error SE $= \sigma/\sqrt{N}$ with the standard deviation of the N values being averaged. However, due to time limitations in this project, it was not possible to sample a statistically significant amount of these random configurations and as such, the displayed SE is not a good estimate of any actual variance margin. The purpose of plotting the standard error here is to obtain a sense of the relative level of dispersion between the different methods.

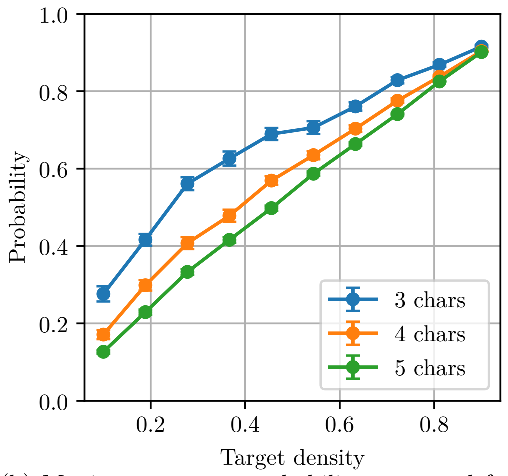
The success probabilities of the Ezhov model are shown in figs. 17a to 17c, each plot being for different values of the search limit D. Inspecting these plots, it can be seen that the retrieval probability is largely dependent on the density of desired states in the memory. One can note how the probability of retrieval seems to be directly proportional to the target density, especially in fig. 17a. However, this shows that the ideal retrieval probability is not much better than that of simply picking a state at random from memory. In the following tests, the same procedure as before is performed, except for the improved model. With this approach, plots in figs. 18a to 18c were given. Comparing plots in figs. 17a to 17c to those in figs. 18a to 18c it is clear that introducing the projection operator does improve success probabilities significantly, reaching average success probabilities of 90% for the larger character limit, and also being more resilient to the lack of desired states for most sequence lengths.

In the previous tests, the probability of unwanted retrieval, i.e. finding states with low priority in the query (e.g. states for which $d_c > D$) but that exist in memory, as well as spurious retrieval, i.e. obtaining a state not in memory, were both also measured to determine the behavior of these structures regarding failure. The results are shown for the $D = 3$ case. Figures 17e and 18e and figs. 17d and 18d show the probabilities of unwanted and spurious retrieval respectively. The first thing to note is the almost nonexistent values for unwanted retrieval. This is not unexpected, as the analytic expression (6.28) shows that a memory state with low query amplitude is expected to have low retrieval amplitude, and with the query of (7.4) unwanted states will per design have low probability. However, the cases of success and failure should span the space of possible scenarios, which is confirmed when comparing figs. 17c and 18c with figs. 17d and 18d, which are nearly each other's complements. Therefore, if the method fails to retrieve a desired state, it will almost definitely return a spurious state.

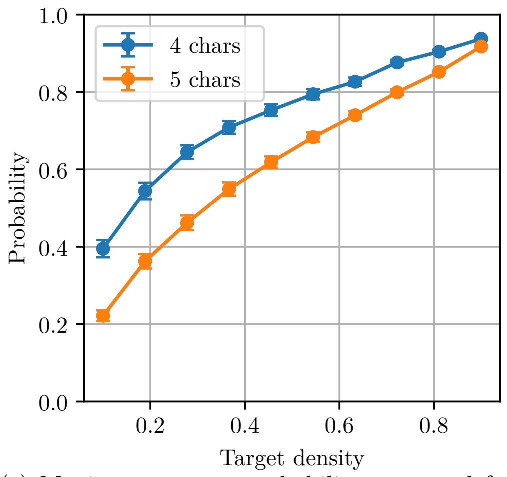
The next question is how these structures perform under a more realistic scenario, when using a precomputed number of iterations. For the Ezhov model, the number of iterations is calculated according to (6.32), however for the improved model there is the question of whether the heuristic (6.35) is effective



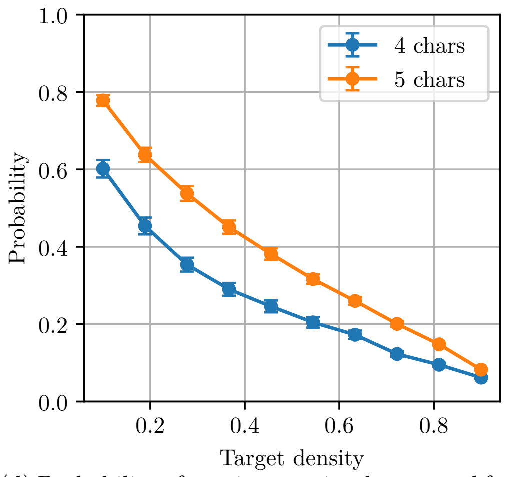
(a) Maximum success probability measured for a character distance limit of $D = 1$.



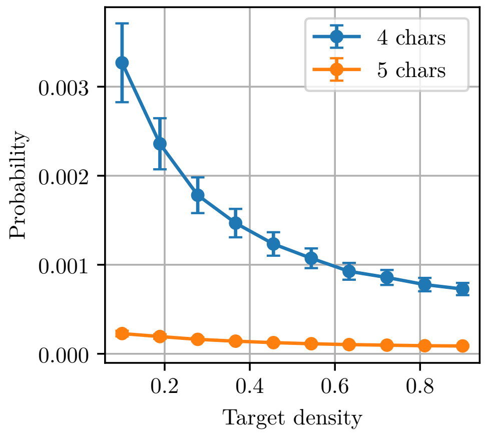
(b) Maximum success probability measured for a character distance limit of $D = 2$.



(c) Maximum success probability measured for a character distance limit of $D = 3$.

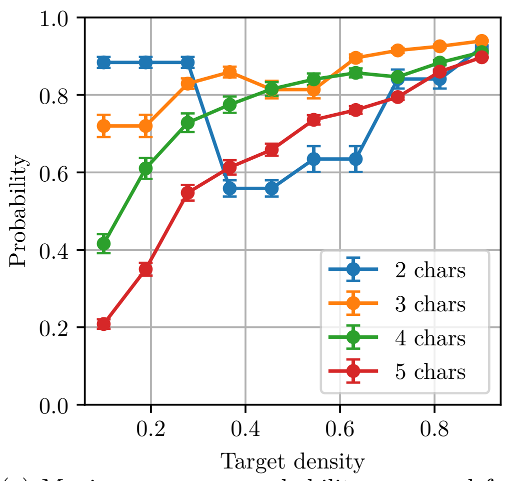


(d) Probability of spurious retrieval measured for a character distance limit of $D = 3$.

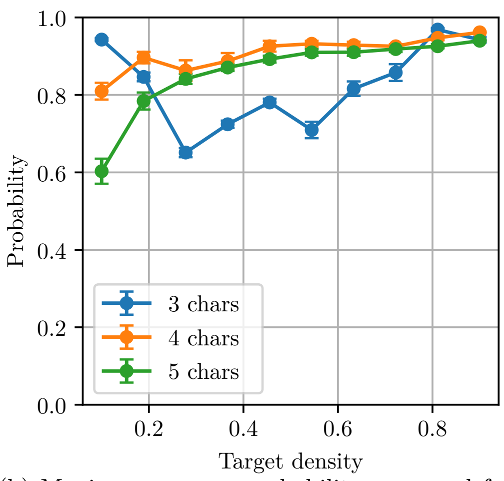


(e) Probability of unwanted retrieval measured for a character distance limit of $D = 3$.

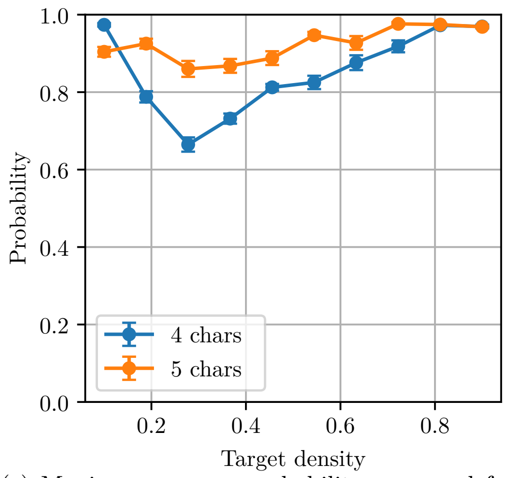
Figure 17: Probabilities measured at maximum success probability with standard error for the Ezhov model for varying density of target states in memory, sequence lengths and metric limits D, using the proposed query (7.4) with $\gamma = 0.4$ and $\rho = 0.1$.



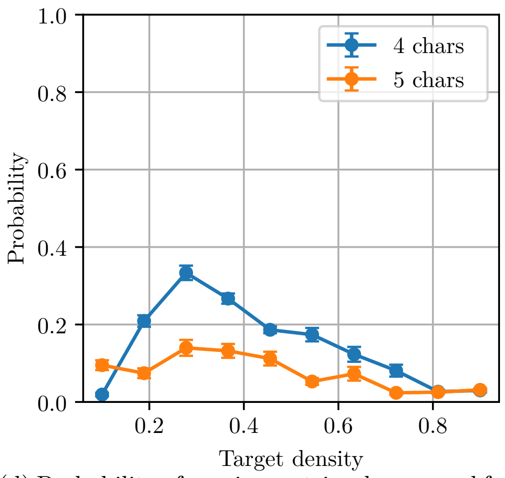
(a) Maximum success probability measured for a character distance limit of $D = 1$.



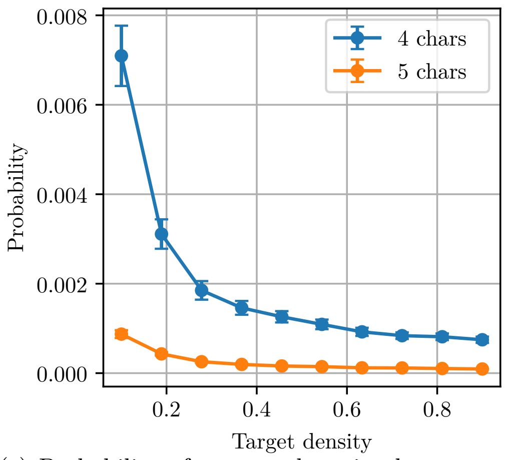
(b) Maximum success probability measured for a character distance limit of $D = 2$.



(c) Maximum success probability measured for a character distance limit of $D = 3$.



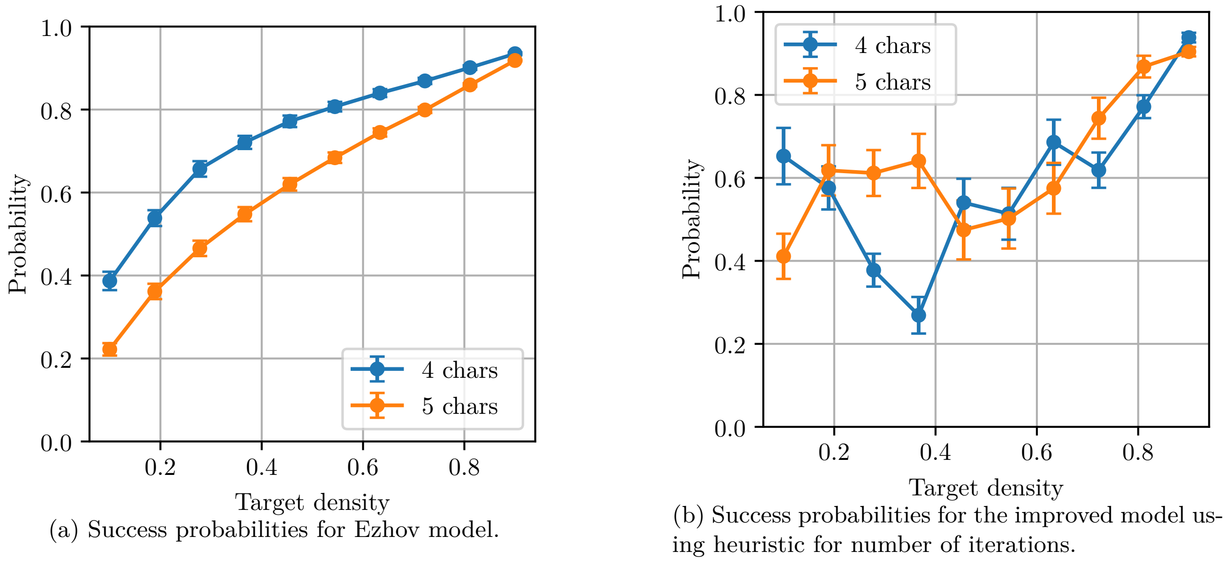
(d) Probability of spurious retrieval measured for a character distance limit of $D = 3$.



(e) Probability of unwanted retrieval measured for a character distance limit of $D = 3$.

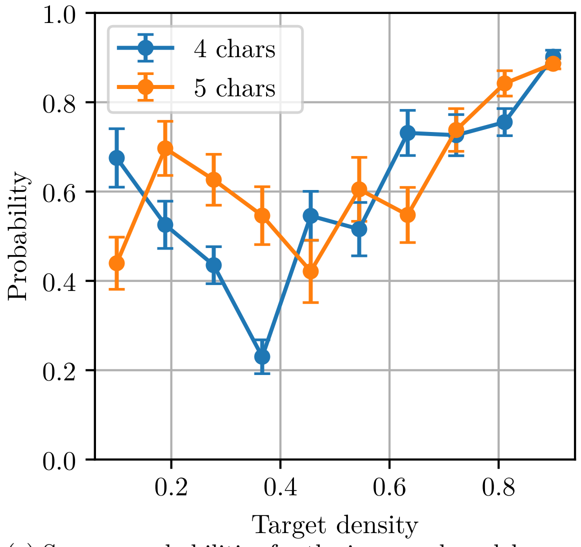
Figure 18: Probabilities measured at maximum success probability with standard error for the improved model for varying density of target states in memory, sequence lengths and metric limits D, using the proposed query (7.4) with $\gamma = 0.4$ and $\rho = 0.1$.

for the proposed query distribution. Therefore, at the first set of these tests, the oscillatory frequency is obtained exactly using the query and memory states in simulation and the respective iteration formulas are tested. However, two variants of the improved model results will be given, one using the exact formula (6.38) derived in this thesis, and the other using the heuristic (6.35). The second set of tests will do the same, but with the oscillation approximation (6.30). However, the approximation is not applicable when using the exact number of iterations for the improved model and therefore, only the heuristic is used. All of the iteration formulas use an arbitrary integer α to increase the number of iterations in search for a more optimal result, but here the integer is limited such that the number of iterations are less than N, the number of stored patterns, when possible.



(a) Success probabilities for Ezhov model.

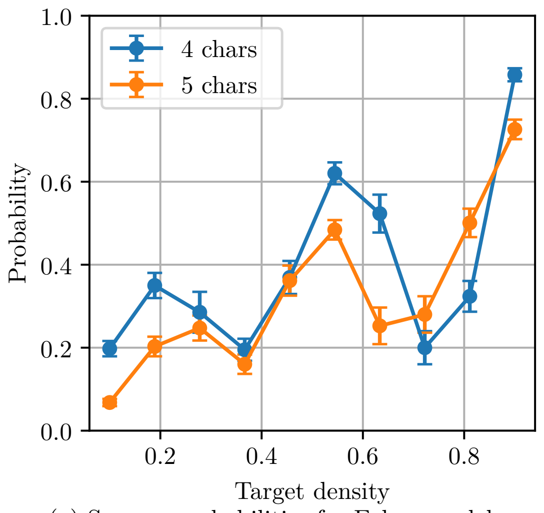
(b) Success probabilities for the improved model using heuristic for number of iterations.



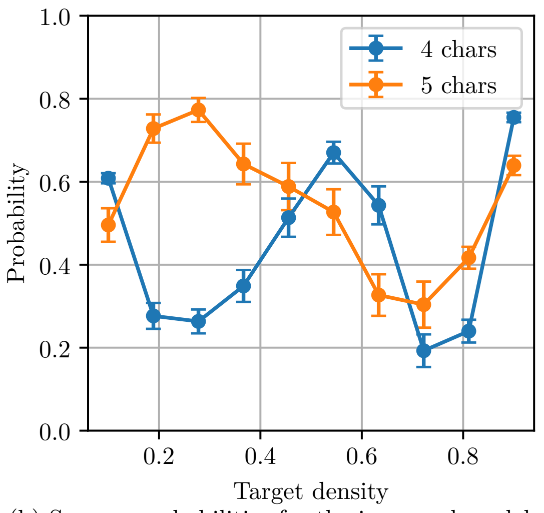
(c) Success probabilities for the improved model using analytic result for number of iterations.

Figure 19: Success probabilities with standard error for the relevant models using exact information on states, but with eventual heuristics needed in practice, shown for $D = 3$, using the proposed query (7.4) with $\gamma = 0.4$ and $\rho = 0.1$.

In figs. 19a and 19b the first set of tests are shown for a search limit of $D = 3$. The results in fig. 19a are more or less identical to the ideal probabilities found in the maximum probability test of fig. 17c, meanwhile the improved model case of fig. 19b using the heuristic approach achieves generally lower probabilities than



(a) Success probabilities for Ezhov model.



(b) Success probabilities for the improved model.

Figure 20: Success probabilities with standard error for the relevant models using all approximations and heuristics needed in practice, shown for $D = 3$, using the proposed query (7.4) with $\gamma = 0.4$ and $\rho = 0.1$.

in the ideal case. One can also comment on the results of fig. 19c that should be similar to fig. 18c, which is not the case. The obtained average probabilities are generally lower than that of fig. 18c and in fact similar to the heuristic result fig. 19b. The dispersion in the improved model is also larger than for any of the previously shown results, indicating that these methods of finding the number of iterations are more unreliable than for the Ezhov model. However, as the number of iterations were limited by a realistic bound dependent on the database size, the dispersion and the worse results could be explained by the algorithms choosing a suboptimal number of iterations because of discretization. This limitation may shadow the true capabilities of the associative memories, but these tests were designed to test realistic performances and that includes the limitations of the structures when restricted to realistic running times. Performing the second set of tests, with results shown in fig. 20, the trend shows a decrease in success probabilities for the Ezhov model in comparison to all previous results. The heuristic approach in the improved model appears more inconsistent as a function of the target density, as all previous tests have more or less indicated an increasing probability with increasing target density, which seems reasonable. What can be seen in fig. 20b is instead an oscillation, possibly caused by the heuristic "failing" at choosing the correct number of iterations, which the increased dispersion for low values appears to indicate. It does however manage to score higher than the Ezhov model for a larger part of the density values.

To summarize, the results show promise in the improved model for the proposed goals in metagenomics, meanwhile the Ezhov model did not appear to offer much higher probability than random selection, particularly not when using required approximations. However, the method is only proven to be useful in ideal conditions. In other scenarios it appears unpredictable and with considerably lower probabilities of successful retrieval, especially when the similarity of query state and memory is initially unknown. If it is to be used in practice, more sophisticated approximation techniques or heuristics would need to be developed. Alternatively it could be applied in a scenario where this information is partially known or can be estimated in advance.

7.2 Trugenberger Quantum Associative Memory

The Trugenberger version of associative memories was described in section 6.2 and determined to be relatively safe from providing false matches (in the sense of not retrieving spurious states). However, it was also determined that it does not offer as much flexibility in its priority distribution in comparison to the models with distributed queries. What will be discussed in this section is mainly how to choose parameters and distance metric to adapt this method for the proposed metagenomics scenario.

The same DNA encoding as (7.2) is assumed for simplicity, however one can note that this model would not be hindered by encodings that do not necessarily span all possible bit patterns. Even if there would exist spurious states by the nature of the chosen encoding, spurious states are non-retrievable in this structure, and thus would not matter for its usage. With this choice of encoding, the same metric proposed in the previous section, (7.3), shall also be used. What then has to be investigated is an efficient decomposition of the metric unitary operator (6.48), how this affects the resulting distribution, and how the auxiliary bit parameter would be selected for some distance limit D.

For the first requirement, one should note the similarity in the current distance measure and the Hamming distance. Since the Hamming distance operator only needed a diagonal single-qubit gate in its decomposition, giving an appropriate phase change for the $|0\rangle$ state (indicating equality in the query and pattern bit), then this type of decomposition unitary can also be provided for our character distance measure, although as a two-qubit gate. One can readily observe that the metric can be obtained using the same kind of decomposition as (6.51),

$$\mathcal{H} = \left(\prod_{i=1}^{n/2} \left(CU^{-2}\right)_{c,m_{2i-1}m_{2i}}\right)\left(\prod_{i=1}^{n/2} U_{m_{2i-1}m_{2i}}\right), \tag{7.6}$$

except for using the two-qubit gate $U = \operatorname{diag}\left(\exp\left(i\frac{\pi}{n}\right), 1, 1, 1\right)$. This unitary can in turn, because of its diagonality, be decomposed into five simple gates[70], with the circuit in fig. 21.

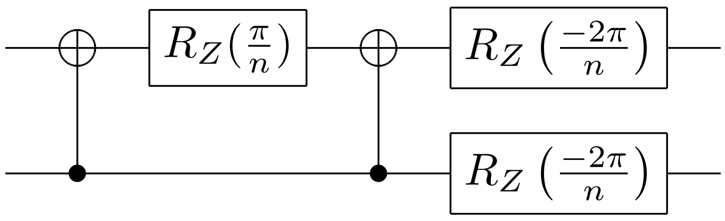

$R_Z(\frac{\pi}{n})$
$R_Z\left(\frac{-2\pi}{n}\right)$
$R_Z\left(\frac{-2\pi}{n}\right)$

Figure 21: Decomposition of diagonal gate for DNA Hamming distance operator.

With the decomposition provided, it is clear from the modifications in (7.6) that the resulting state will be the same as (6.56) except for that the change of metric $d_H \to 2d_c$. This would produce a retrieval probability distribution as a function of the character distance d_c given by

$$P(p_i) = \frac{1}{NP_{rec}} \cos^{2B}\left(\frac{\pi}{n} d_c(Q, p_i)\right), \tag{7.7}$$

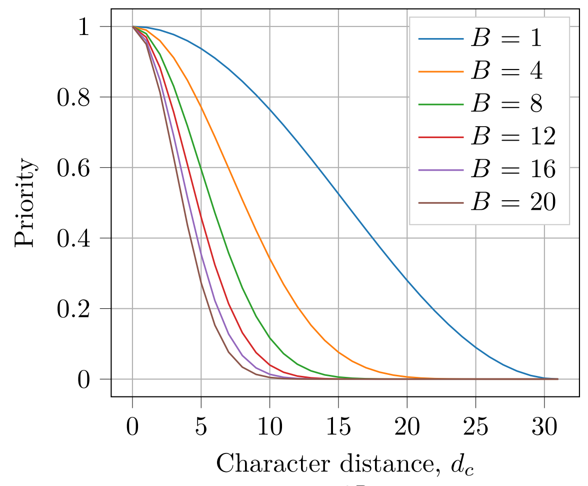
with the appropriate normalization of P_{rec} after the postselection measurement. The shape of this distribution as a function of the metric d_c is shown in fig. 22a for different choices of B. It is apparent that if one wishes to have a more strict priority with a low distance limit, the number of auxiliary bits required does not increase linearly. In fact, given a threshold δ for which to define the maximum priority past the character distance $d_c = D$, the number of auxiliary bits required to achieve this is found to be

$$\cos^{2B}\left(\frac{D\pi}{n}\right) \leq \delta \implies B \geq \frac{\log(\delta)}{2\log\left(\cos\left(\frac{D\pi}{n}\right)\right)}. \tag{7.8}$$

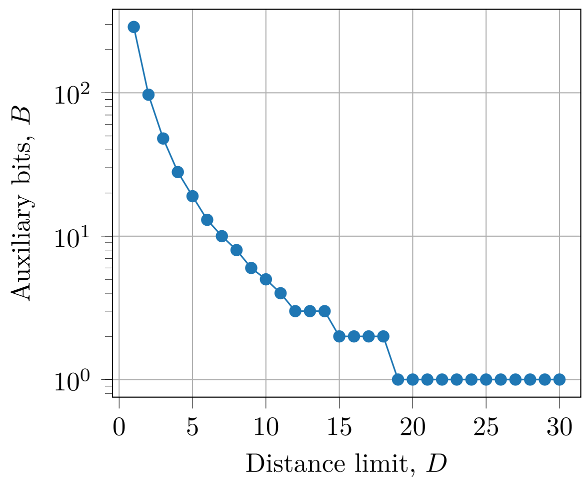
Of course, the actual probability is normalized by an initially unknown constant P_{rec} dependent on the overall match between query and memory, so knowing an appropriate δ as it is defined in (7.8) is difficult. One should instead reformulate our conditions purely in terms of the distribution, i.e. demand that when counting all possible patterns p, the sum of the priority for all states below the distance limit D should be at least a ratio of δ in comparison to the rest of the states. Writing

$$\frac{\sum\limits_{d_c(Q,p)\leq D} P(p)}{\sum_p P(p)} \geq \delta, \tag{7.9}$$

One can choose δ such that the distribution is mostly centered on the distances for which one prioritizes. Choosing $\delta = 0.9$ as an example, and working with patterns of length $n = 62$, i.e. sequences of length $K = 31$, the number of bits required can be found numerically as shown in fig. 22b. As is seen, the number of qubits required increases exponentially with a lower distance limit. However, the bit usage is



(a) The priority distribution $\cos^{2B}\left(\frac{\pi}{n}d_c\right)$ for different B and using $n = 62$, i.e. 31 sequence characters.



(b) Number of auxiliary bits required to put a priority limit at D for $\delta = 0.9$. Note the logarithmic y-scale.

not unreasonable for $D \geq 5$ at this threshold, considering a reasonable resource limit of a hundred qubits for future quantum computers.

We have thus seen how to adapt this technique to DNA sequences and for metagenomics purposes. The Trugenberger model offers more certainty in performance than the Grover-based models, since the properties of the retrieval state before measurement are known. The only problem is the bit usage and probability of obtaining said state, i.e. getting past the recognition phase. With the recognition probability possibly reaching extremely low levels when using many auxiliary bits, testing the methods mentioned in section 6.2.1 to circumvent this would be a good next step in researching this structure. But this is left for further research.

8 Summary

In section 4 the most recent methods to have been developed for bioinformatics purposes or for similar problems were presented. The methods demonstrate interesting ways of using both Grover search and the quantum Fourier transform to perform local alignment that could act as fundamental techniques in methods to be developed. However, among the current methods, few stand out in comparison to the commonly used classical algorithms, especially at the scale of most bioinformatic problems. The pattern matching algorithm for local sequence alignment by Prousalis and Konofaos[37] would be the sole exception if one could get past its more practical difficulties. The limitations of the methods mostly concern inefficient implementation of involved operators, being a black-box unitary or some oracle operator, the small amount of qubits at ones disposal and efficient transference of classical information into the quantum device, i.e. state preparation. These challenges will need to be faced for a quantum device to replace classical methods for the alignment problem. Although it was not explored if these techniques are applicable in a more niche setting where smaller sequences occur, or as a subroutine to other classical algorithms.

For the purpose of augmenting K-mer based binning methods in metagenomics with fuzzy matching capabilities, quantum associative networks were in section 7 adapted to the matching of DNA sequences, in the scenario that the quantum algorithms follows a failed exact matching. Grover-based memories had their success probabilities and spurious retrieval investigated for the different cases of ideal retrieval conditions, retrieval with full knowledge of similarity in query and memory, and a realistic scenario assuming that no relation between query and memory is known. What we called the improved model, proposed by [54], showed excellent potential for successful retrieval during ideal conditions, but displayed lower and more unpredictable values for the success probabilities when using necessary approximations. Although efficient methods of memory state and operator construction still needs to be studied in order for these structures to be efficient in practice, if one could tackle those issues and find improved approximations for the necessary information regarding query and memory, then Grover-based associative memories could prove a useful tool in improving metagenomic binning methods. A DNA sequence adaption was provided for the associative memory proposed by Trugenberger[51] and the effect of the choice of parameters was studied. This model has the advantage of being simple and efficient to implement, incapable of spurious retrieval in a noise-free environment and having reasonable bit usage for the considered sequence lengths. While the issues regarding postselection failure would need to be researched further, especially for when the fuzzy search is to be limited to a small distance range, the structure would make a near ideal retrieval method for the purpose of metagenomics if these issues could be resolved or improved upon.

In section 5 an Ising spin formulation of the multiple sequence alignment problem was presented, exhibiting a spin usage which is linear in the number of sequences and quadratic in their lengths. It was shown to be correctly constructed and capable of solving simple alignment problems both using a simulated annealer and a D-Wave quantum annealer. Although there were limitations to what configurations it could solve on real quantum hardware, its correctness and demonstration will hopefully be a good first step for further developments. Its issues are possible to have been caused by an inefficient chimera graph embedding or insufficient annealing time. The investigation of this and eventual improvements to this solving technique is left for future endeavors.

9 Further Research

It is difficult to cover all areas and discoveries in quantum computation that could be applied to bioinformatics. It was attempted to do so within the limited time frame of this thesis, but inevitably there are areas that could be of interest which were not possible to thoroughly study. Here, proposals for areas to be researched further are listed.

9.1 Quantum Pattern-Matching

There could be ways of improving the quantum associative memories we have seen in this thesis, to either increase the probability of matching or extending the capabilities of fuzzy matching and completion. It has been investigated if one could make the wildcard positions in associative memory completion unknown, and still find a match[71]. By using another kind of phase change in the Grover search operators one has been able to achieve an almost certain success rate for pattern recognition[72] although limited to when certain conditions for the memory are fulfilled. A non-linear retrieval method as an alternative to Grover-based retrieval has been proposed[73], and such methods have also been shown to make quantum computations capable of solving more difficult problems efficiently[74], but it is unclear whether such operations could be implemented in practice, especially in our current, circuit based, computational model.

Other forms of quantum associative memories, or general pattern matching algorithms have also been studied to some extent. It has been suggested to implement a quantum associative memory through adiabatic computation[75] and basis states other than the computational basis has been proposed for memory state representation[76]. As for other forms of pattern matching, the problem has been formulated in terms of quantum template matching[77, 78], and a type of quantum neural network, the quantum Hopfield network, has also been investigated for its classification capabilities using RNA data[79].

9.2 Alignment Methods

While most methods developed for any kind of genetic sequence alignment have been summarized in section 4, there are a plethora of methods that make up modern routines for (multiple) sequence alignments, that potentially could be adapted to bioinformatic purposes if thoroughly studied. Implementation of quantum dynamic programming has been studied with speedup for some problem configurations[80] and could possibly be applied to improve parts of popular alignment algorithms such as the Needleman-Wunsch and Smith-Waterman methods, heavily reliant on dynamic programming techniques. As has been described previously, some modern methods of multiple sequence alignment rely on iterative techniques, such as evolutionary and genetic algorithms, to find approximate solutions. There have been successful quantum inspired algorithms that make use of quantum mechanisms to improve the search for iterative solutions, although designed for classical computations[81]. However, this shows the potential of quantum mechanics for such purposes if the iterative approaches could be implemented using true quantum computations, which has been suggested and reviewed[82, 83, 84]. Alignment problems have also been treated classically using hidden Markov models[85], and the development of quantum Markov models[86] could be of use in making those approaches more memory efficient.

References

[1] Z. D. Stephens et al. “Big Data: Astronomical or Genomical?” In: *PLOS Biology* 13.7 (July 2015), pp. 1–11.

[2] *IBMQ*. 2019. URL: `https://www.research.ibm.com/ibm-q/`.

[3] *Rigetti Computing*. 2019. URL: `https://www.rigetti.com/`.

[4] *D-Wave Systems*. 2019. URL: `https://www.dwavesys.com/`.

[5] J. Preskill. “Quantum Computing in the NISQ era and beyond”. In: *Quantum* 2 (2018), p. 79.

[6] *Entropica labs*. 2019. URL: `https://entropicalabs.com/about`.

[7] Madprime. *DNA Chemical Structure*. 2007. URL: `https://commons.wikimedia.org/wiki/File:DNA_chemical_structure.svg`.

[8] L. Wang and T. Jiang. “On the Complexity of Multiple Sequence Alignment”. In: *Journal of Computational Biology* 1.4 (1994), pp. 337–348.

[9] D. F. Feng and R. F. Doolittle. “Progressive sequence alignment as a prerequisite to correct phylogenetic trees.” In: *Journal of molecular evolution* 25.4 (1987), pp. 351–60.

[10] J. D. Thompson, D. G. Higgins, and T. J. Gibson. “CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice”. In: *Nucleic Acids Research* 22.22 (1994), pp. 4673–4680.

[11] C. Notredame, D. G. Higgins, and J. Heringa. “T-coffee: a novel method for fast and accurate multiple sequence alignment, Edited by J. Thornton”. In: *Journal of Molecular Biology* 302.1 (2000), pp. 205–217.

[12] T. Thomas, J. Gilbert, and F. Meyer. “Metagenomics - a guide from sampling to data analysis”. In: *Microbial Informatics and Experimentation* 2.1 (2012), p. 3.

[13] V. H. Pham and J. Kim. “Cultivation of unculturable soil bacteria”. In: *Trends in Biotechnology* 30.9 (2012), pp. 475–484.

[14] P. J. Turnbaugh et al. “The Human Microbiome Project”. In: *Nature* 449.7164 (2007), pp. 804–810.

[15] J. Banerjee, N. Mishra, and Y. Dhas. “Metagenomics: A new horizon in cancer research”. In: *Meta Gene* 5 (2015), pp. 84–89.

[16] S. Anderson. “Shotgun DNA sequencing using cloned DNase I-generated fragments”. In: *Nucleic Acids Research* 9.13 (1981), pp. 3015–3027.

[17] Illumina. *Illumina Expands World’s Most Comprehensive Next-Generation Sequencing Portfolio*. Illumina. Jan. 12, 2015. URL: `https://www.illumina.com/company/news-center/press-releases/press-release-details.html?newsid=2006979`.

[18] D. E. Wood and S. L. Salzberg. “Kraken: ultrafast metagenomic sequence classification using exact alignments”. In: *Genome Biology* 15.3 (2014), R46.

[19] W. Wootters and W. H. Zurek. “A single quantum cannot be cloned”. In: *Nature* 299.September (1982), pp. 802–803.

[20] M. A. Nielsen and I. L. Chuang. “Quantum Circuits”. In: *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press, 2010. Chap. 4, pp. 171–216.

[21] D. Biron et al. “Generalized Grover Search Algorithm for Arbitrary Initial Amplitude Distribution”. In: *ArXiv eprints* (1998). arXiv: `quant-ph/9801066`.

[22] M. Boyer et al. “Tight Bounds on Quantum Searching”. In: *Fortschritte der Physik* 46.4-5 (1998), pp. 493–505.

[23] P. Li and K. Song. “Adaptive Phase Matching in Grover’s Algorithm”. In: *Journal of Quantum Information Science* 01.02 (2011), pp. 43–49.

[24] E. Farhi et al. “Quantum Computation by Adiabatic Evolution”. In: *ArXiv eprints* (2000). arXiv: `quant-ph/0001106`.

[25] *D-Wave System Documentations*. May 30, 2019. URL: `https://docs.dwavesys.com/docs/latest/index.html`.

[26] M. A. Nielsen and I. L. Chuang. “Quantum Fourier Transform and Its Applications”. In: *Quantum Computation and Quantum Information*. Cambridge: Cambridge University Press, 2010. Chap. 5, pp. 216–242.

[27] L. C. L. Hollenberg. “Fast Quantum Search Algorithms in Protein Sequence Comparison - Quantum Biocomputing”. In: *Physical Review E* 62.5 (2000), p. 5. arXiv: `quant-ph/0002076v1`.

[28] W. P. Baritompa, D. W. Bulger, and G. R. Wood. “Grover’s Quantum Algorithm Applied to Global Optimization”. In: *SIAM Journal on Optimization* 15.4 (2005), pp. 1170–1184.

[29] A. Ahuja and S. Kapoor. “A Quantum Algorithm for finding the Maximum”. In: *ArXiv eprints* (1999), pp. 1–5. arXiv: `quant-ph/9911082`.

[30] S. Hakak et al. “Exact String Matching Algorithms: Survey, Issues, and Future Research Directions”. In: *IEEE Access* PP.c (2019), pp. 1–1.

[31] P. Mateus and Y. Omar. “Quantum Pattern Matching”. In: *ArXiv eprints* (2005). arXiv: `quant-ph/0508237`.

[32] P. Mateus and Y. Omar. “A Quantum Algorithm for Closest Pattern Matching”. In: *Quantum Information Processing: From Theory to Experiment* (2006), pp. 180–183.

[33] A. Sarkar. “Quantum Algorithms for pattern-matching in genomic sequences”. Master Thesis. University of Delft, 2018, pp. 1–116.

[34] H. Ramesh and V. Vinay. “String matching in $\tilde{O}(\sqrt{n} + \sqrt{m})$ quantum time”. In: *Journal of Discrete Algorithms* 1.1 (2003), pp. 103–110.

[35] M. Boroujeni et al. “Approximating Edit Distance in Truly Subquadratic Time: Quantum and MapReduce”. In: *ArXiv eprints* (2018), pp. 1–38. arXiv: `1804.04178 [quant-ph]`.

[36] R. Schützhold. “Pattern recognition on a quantum computer”. In: *Physical Review A - Atomic, Molecular, and Optical Physics* 67.6 (2003), p. 6.

[37] K. Prousalis and N. Konofaos. “Quantum Pattern Recognition for Local Sequence Alignment”. In: *2017 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2017, pp. 1–5.

[38] S. Iriyama and M. Ohya. “On Quantum Algorithm for Multiple Alignment of Amino Acid Sequences”. In: *Quantum Bio-Informatics II* 2 (2009), pp. 92–104.

[39] A. Peruzzo et al. “A variational eigenvalue solver on a photonic quantum processor”. In: *Nature Communications* 5.1 (2014), p. 4213. arXiv: `1304.3061v1 [quant-ph]`.

[40] E. Farhi, J. Goldstone, and S. Gutmann. “A Quantum Approximate Optimization Algorithm”. In: *ArXiv eprints* (2014), pp. 1–16. arXiv: `1411.4028 [quant-ph]`.

[41] N. Moll et al. “Quantum optimization using variational algorithms on near-term quantum devices”. In: *Quantum Science and Technology* 3.3 (2018), pp. 1–30. arXiv: `1710.01022v2 [quant-ph]`.

[42] A. Lucas. “Ising formulations of many NP problems”. In: *Frontiers in Physics* 2 (2014), pp. 1–27. arXiv: `1302.5843 [quant-ph]`.

[43] D-Wave Systems. *D-Wave Initiates Open Quantum Software Environment*. D-Wave Systems. Jan. 1, 2017. URL: `https://www.dwavesys.com/press-releases/d-wave-initiates-open-quantum-software-environment` (visited on 05/30/2019).

[44] D. Mapleson et al. *msa2qubo*. `https://github.com/maplesond/msa2qubo`. 2016.

[45] J. Kececioglu. “The maximum weight trace problem in multiple sequence alignment”. In: *Combinatorial Pattern Matching* February (2005), pp. 106–119.

[46] H. Lenhof, B. Morgenstern, and K. Reinert. “An exact solution for the segment-to-segment multiple sequence alignment problem”. In: *Bioinformatics* 15.3 (1999), pp. 203–210.

[47] *D-Wave Ocean SDK*. `https://github.com/dwavesystems/dwave-ocean-sdk`. 2019.

[48] I. Shapoval and P. Calafiura. “Quantum Associative Memory in HEP Track Pattern Recognition”. In: *ArXiv eprints* (2019). arXiv: `1902.00498 [quant-ph]`.

[49] D. Ventura and T. Martinez. “Quantum associative memory”. In: *Information Sciences* 124.1-4 (2000), pp. 273–296.

[50] D. Ventura and T. Martinez. "Initializing the Amplitude Distribution of a Quantum State". In: *ArXiv eprints* 12.6 (1998), pp. 547–559. arXiv: `quant-ph/9807054`.

[51] C. A. Trugenberger. "Probabilistic Quantum Memories". In: *Physical Review Letters* 87.6 (2001), pp. 67901–1–67901–4.

[52] A. Ezhov, A. Nifanova, and D. Ventura. "Quantum associative memory with distributed queries". In: *Information Sciences* 128.3-4 (2000), pp. 271–293.

[53] D. Ventura. "Pattern Classification Using a Quantum System". In: *Proceedings of the 6th Joint Conference on Information Sciences.* 2002, pp. 537–540.

[54] J. P. Tchapet Njafa, S. G. Nana Engo, and P. Woafo. "Quantum Associative Memory with Improved Distributed Queries". In: *International Journal of Theoretical Physics* 52.6 (2013), pp. 1787–1801.

[55] Y. Liu and G. L. Long. "Deleting a marked basis-state from an even superposition of all basis-states within a single query". In: *International Journal of Quantum Information* 07.02 (2009), pp. 567–572.

[56] Y. Liu and X. Ouyang. "A quantum algorithm that deletes marked states from an arbitrary database". In: *Chinese Science Bulletin* 58.19 (2013), pp. 2329–2333.

[57] Y. Liu. "Deleting a marked state in quantum database in a duality computing mode". In: *Chinese Science Bulletin* 58.24 (2013), pp. 2927–2931.

[58] P. Li and S. Li. "Phase matching in quantum searching algorithm with weighted targets". In: *Jisuan Wuli/Chinese Journal of Computational Physics* 25.5 (2008), pp. 363–369.

[59] S. Lloyd, M. Mohseni, and P. Rebentrost. "Quantum principal component analysis". In: *Nature Physics* 10.9 (2014), pp. 631–633.

[60] H. Buhrman et al. "Quantum Fingerprinting". In: *Physical Review Letters* 87.16 (2001), p. 167902. arXiv: `quant-ph/0102001`.

[61] G. Brassard et al. "Quantum Amplitude Amplification and Estimation". In: *ArXiv eprints* (2000). arXiv: `quant-ph/0005055`.

[62] J. P. Tchapet Njafa and S. G. Nana Engo. "Quantum associative memory with linear and non-linear algorithms for the diagnosis of some tropical diseases". In: *Neural Networks* 97 (2018), pp. 1–10.

[63] M. C. Diamantini and C. A. Trugenberger. "High-Capacity Quantum Associative Memories". In: *Journal of Applied Mathematics and Physics* 04.11 (2016), pp. 2079–2112.

[64] V. Bužek and M. Hillery. "Quantum copying: Beyond the no-cloning theorem". In: *Physical Review A - Atomic, Molecular, and Optical Physics* 54.3 (1996), pp. 1844–1852. arXiv: `quant-ph/9607018v1`.

[65] L.-M. Duan and G.-C. Guo. "Probabilistic Cloning and Identification of Linearly Independent Quantum States". In: *Physical Review Letters* 80.22 (1998), pp. 4999–5002.

[66] J. Y. Haw et al. "Surpassing the no-cloning limit with a heralded hybrid linear amplifier". In: *2017 Conference on Lasers and Electro-Optics Pacific Rim (CLEO-PR).* Vol. 2017-Janua. IEEE, 2017, pp. 1–1.

[67] A. M. Childs, A. J. Landahl, and P. A. Parrilo. "Quantum algorithms for the ordered search problem via semidefinite programming". In: *Physical Review A* 75.3 (2007), p. 032335. arXiv: `quant-ph/0608161v1`.

[68] P. Høyer, J. Neerbek, and Y. Shi. "Quantum Complexities of Ordered Searching, Sorting, and Element Distinctness". In: 2001, pp. 346–357. arXiv: `quant-ph/0102078`.

[69] J. Johansson, P. Nation, and F. Nori. "QuTiP 2: A Python framework for the dynamics of open quantum systems". In: *Computer Physics Communications* 184.4 (2013), pp. 1234–1240.

[70] S. S. Bullock and I. L. Markov. "Arbitrary two-qubit computation in 23 elementary gates". In: *Physical Review A - Atomic, Molecular, and Optical Physics* 68.1 (2003), p. 7. arXiv: `quant-ph/0211002v3`.

[71] R. G. Zhou et al. "Quantum Pattern Search with Closed Match". In: *International Journal of Theoretical Physics* 52.11 (2013), pp. 3970–3980.

[72] R. Zhou and Q. Ding. "Quantum pattern recognition with probability of 100%". In: *International Journal of Theoretical Physics* 47.5 (2008), pp. 1278–1285.

[73] J.-P. Tchapet Njafa and S. Nana Engo. “Concise quantum associative memories with nonlinear search algorithm”. In: *Fortschritte der Physik* 64.2-3 (2016), pp. 250–268.

[74] D. S. Abrams and S. Lloyd. “Nonlinear Quantum Mechanics Implies Polynomial-Time Solution for NP-Complete and P Problems”. In: *Physical Review Letters* 81.18 (1998), pp. 3992–3995.

[75] Y. OSAKABE et al. “Quantum Associative Memory with Quantum Neural Network via Adiabatic Hamiltonian Evolution”. In: *IEICE Transactions on Information and Systems* E100.D.11 (2017), pp. 2683–2689.

[76] M. P. Singh and B. S. Rajput. “New Maximally Entangled States for Pattern-Association Through Evolutionary Processes in a Two-Qubit System”. In: *International Journal of Theoretical Physics* 56.4 (2017), pp. 1274–1285.

[77] M. Sasaki, A. Carlini, and R. Jozsa. “Quantum template matching”. In: *Physical Review A* 64.2 (2001), p. 022317. arXiv: `quant-ph/0102020`.

[78] D. Curtis and D. A. Meyer. “Towards quantum template matching”. In: *Quantum Communications and Quantum Imaging* 5161.February 2004 (2004), p. 134.

[79] P. Rebentrost et al. “Quantum Hopfield neural network”. In: *Physical Review A* 98.4 (2018), pp. 1–13. arXiv: `1710.03599v3 [quant-ph]`.

[80] B. Furrow. “A Panoply of Quantum Algorithms”. In: *ArXiv eprints* (2006), pp. 1–32. arXiv: `quant-ph/0606127`.

[81] K. H. Tsutomu, T. Yokoyama, and T. Shimizu. “Multiple Sequence Alignment by Genetic Algorithm”. In: *Genome Informatics* 11 (2000), pp. 317–318.

[82] D. A. Sofge. “Prospective Algorithms for Quantum Evolutionary Computation”. In: *ArXiv eprints* (2008). arXiv: `0804.1133 [quant-ph]`.

[83] D. Johannsen, P. P. Kurur, and J. Lengler. “Evolutionary algorithms for quantum computers”. In: *Algorithmica* 68.1 (2014), pp. 152–189.

[84] R. Lahoz-Beltra. “Quantum Genetic Algorithms for Computer Scientists”. In: *Computers* 5.4 (2016), p. 24.

[85] S. R. Eddy. “Multiple alignment using hidden Markov models Seminar Hot Topics in Bioinformatics”. In: *Int Conf Intell Syst Mol Biol.* Vol. 3. 1995, pp. 114–120.

[86] S. Srinivasan, G. Gordon, and B. Boots. “Learning Hidden Quantum Markov Models”. In: *ArXiv eprints* (2017), pp. 1–19. arXiv: `1710.09016 [quant-ph]`.

10 Appendix

10.1 Quantum Associative Memories

This section includes the code used to simulate and test the Grover-based quantum associative memories that have been investigated. It consists of three files: the code containing the QuAM class used to simulate the memories, the test file for calculating maximum retrieval probabilities, and the code for calculating probabilities under the more realistic conditions (using approximations and heuristics).

10.1.1 The QuAM Class

The class file for the QuAM object that is used to prepare and execute a QuAM retrieval simulation.

```python
import numpy as np
import qutip as qu
import matplotlib.pyplot as plt
import types

def bit_arr_to_int(pattern):
    return np.sum(pattern*np.power(2, np.arange(len(pattern)-1, -1, -1)))

def str_to_bit_arr(bit_str):
    return np.array([x == "1" for x in bit_str])

def find_closest_int(func, iter_limit=100, scheme="ezhov"):
    """
    Finds closest integer for function func within given limit
    """
    start = func(0)
    point = func(1)
    increase = point - start
    limit = int(np.floor((iter_limit - start)/increase))
    while func(limit) < 1:
        limit = 1
    nums = np.arange(0, limit+1)
    vals = func(nums)
    vals = np.abs(np.mod(vals, 1) - 0.5)
    alph = np.argmax(vals)
    M = int(np.round(func(nums[alph])))
    if scheme != "ezhov" and M < 2:
        M = 2
    return M

def hamm_bin(center, *params):
    """
    built-in function for binomial distribution on the hamming distance
    """
    a = params[0]
    def hamm_dist(pattern, ref):
        return np.sum(np.logical_not(np.logical_xor(pattern, ref)))
    N = len(center)
    perm_numbers = np.arange(2**N-1, -1, -1)
    perms = [format(s, "0" + str(N) + "b") for s in range(len(perm_numbers))]
    perms = np.array([np.array([x == "1" for x in s]) for s in perms])
    weights = np.zeros(len(perms))
    for i in range(len(perms)):
        dist = hamm_dist(perms[i], center)
        weights[i] = np.sqrt((a**dist)*(1-a)**(N-dist))
    for i in range(len(perms)):
        if i == 0:
            state = weights[i]*qu.basis(len(perm_numbers), perm_numbers[i])
        else:
            state += weights[i]*qu.basis(len(perm_numbers), perm_numbers[i])
    return state.unit()

def excl_mem(pattern, state):
    """
    exclusive memory creation
    """
    N = 2**len(pattern)
    if state is None:
        state = sum([qu.basis(N, i) for i in range(N)])
        state = state.unit()
    pattern_state = qu.basis(N, bit_arr_to_int(pattern))
    state -= pattern_state.overlap(state)*pattern_state
    return state

def incl_mem(pattern, state):
    """
    inclusive memory creation
    """
    N = 2**len(pattern)
    if state is None:
        state = qu.Qobj()
    state += qu.basis(N, bit_arr_to_int(pattern))
    return state

class QuAM:

    def __init__(self):
        self.diffusion = None
        self.oracle = None
        self.mem_op = None
        self.query = None
        self.patterns = None
        self.memory = None
        self.memories = None
        self.state = None

    def set_query(self, patterns, weights=None, scheme=None, bin_param=0.25):
        """Sets the pattern of the search

        Unless weights or different scheme is given, uses a Hamming distance
        binomial distribution for the distributed pattern query

        patterns - list of binary array for the patterns sought after : list of bool array
        weights - (relative) probability weights for the patterns in the query : numpy array of floats
        scheme - Function for making query distribution : function handle with 1 argument, the pattern
        """
        if weights is None:
            if scheme is None:
                scheme = lambda pattern: hamm_bin(pattern, bin_param)
            state = qu.Qobj()
            for pattern in patterns:
                state += scheme(str_to_bit_arr(pattern))
        else:
            state = qu.Qobj()
            max_ID = 2**len(patterns[0])
            for (pattern, weight) in zip(patterns, weights):
                state_id = bit_arr_to_int(str_to_bit_arr(pattern))
                state += weight*qu.basis(max_ID, state_id)
        state = state.unit()
        self.query = state
        self.patterns = patterns

    def set_mem(self, memories, scheme=None):
        """Sets the memory of the search

        Takes a list of states to remember, and a scheme of how to input them
        into the memory (standards: 'inclusion', 'exclusion', 'vary_phase')

        memories - list of binary arrays to memorize : list of bool numpy arrays
        scheme - the memorization scheme (standard: exclusion) : string or
        function handle taking a pattern and current memory state
        """
        if not scheme:
            scheme = "exclusion"
        if isinstance(scheme, str):
            if scheme == "inclusion":
                scheme = incl_mem
            elif scheme == "exclusion":
                scheme = excl_mem
            elif scheme == "vary_phase":
                # TODO
                pass
        elif not isinstance(scheme, types.FunctionType):
            raise Exception("Memorization scheme is not valid")

        state = None
        mem_op = qu.Qobj()
        N = 2**len(memories[0])
        for mem in memories:
            state = scheme(str_to_bit_arr(mem), state)

            state_ID = bit_arr_to_int(str_to_bit_arr(mem))
            mem_op += qu.ket2dm(qu.basis(N, state_ID))
        self.mem_op = qu.identity(N) - 2*mem_op
        if state.norm() > 0:
            self.memory = state.unit()
        else:
            self.memory = state
        self.memories = memories

    def get_iter_num(self, match_type, scheme="approx", mem=None, quer=None):
        """
        Calculates the number of iteration given a strategy for calculation and
        the current scenario of using approximations or the exact information
        """
        if mem is None:
            mem = self.memory
        if quer is None:
            quer = self.query

        if match_type == "ezhov":
            """
            Ezhov model
            """
            if scheme == "approx":
                B = np.sum(quer.data)/np.sqrt(quer.shape[0])
            else:
                B = quer.overlap(mem)

            B = np.abs(B)
            # print("B", B)
            w = 2*np.arcsin(B)
            # print("w", w)
            T = 2*np.pi/w
            # print("T", T)
            iter_limit = len(self.memories)
            M = find_closest_int(lambda x : T*(1/4 + x), iter_limit=iter_limit, scheme=match_type)
        elif match_type == "C1" and scheme == "approx":
            """
            Improved Ezhov model with approximation, but heuristic approach
            """
            B = np.sum(quer.data)/np.sqrt(quer.shape[0])

            B = np.abs(B)
            # print("B", B)
            w = 2*np.arcsin(B)
            # print("w", w)
            T = 2*np.pi/w
            # print("T", T)
            iter_limit = len(self.memories)
            M = find_closest_int(lambda x : T*(1/4 + x), iter_limit=iter_limit, scheme=match_type)
        elif match_type == "C1" and scheme == "semi_approx":
            """
            Improved Ezhov model with exact info, but heuristic approach
            """
            B = quer.overlap(mem)

            B = np.abs(B)
            # print("B", B)
            w = 2*np.arcsin(B)
            # print("w", w)
            T = 2*np.pi/w
            # print("T", T)
            iter_limit = len(self.memories)
            M = find_closest_int(lambda x : T*(1/4 + x), iter_limit=iter_limit, scheme=match_type)
        elif match_type == "C1" and scheme == "exact":
            """
            Improved Ezhov model with exact info, analytic approach
            """
            B = quer.overlap(mem)

            B = np.abs(B)
            w = 2*np.arcsin(B)
            N = 2**len(self.memories[0])
            mem_projector = (qu.identity(N) - self.mem_op) / 2

            init_state = self.memory + 2*B*(2*mem_projector - 1)*self.query
            alpha = np.real(init_state.overlap(self.memory))
            beta = np.real(init_state.overlap(self.query))
            phi = np.arctan((1 - np.cos(w) - 2*B*alpha/beta)/np.sin(w))

            iter_limit = len(self.memories)
            func = lambda n : (n*np.pi - phi)/w + 1/2
            M = find_closest_int(func, iter_limit=iter_limit, scheme=match_type) + 1
        else:
            raise Exception("Invalid iteration identifiers")
        return M


    def match_ezhov(self, iteration="approx"):
        """
        Match using Ezhov QuAM
        """
        if isinstance(iteration, str):
            M = self.get_iter_num("ezhov", iteration)
        elif isinstance(iteration, int):
            M = iteration
        else:
            raise Exception("Invalid iteration parameter")

        state = self.memory.copy()
        state_hist = np.zeros(M+1, dtype=qu.Qobj)
        state_hist[0] = state
        for i in range(M):
            state = self.oracle*state
            state = self.diffusion*state
            state_hist[i+1] = state
        return state, state_hist

    def match_C1(self, iteration="approx"):
        """
        Match using improved QuAM
        """
        if type(iteration) is int:
            M = iteration
        else:
            M = self.get_iter_num("C1", iteration)

        if M == 0:
            input("this fails here")

        state_hist = np.zeros(M+2, dtype=qu.Qobj)
        state = self.memory.copy()
        state_hist[0] = state
        state = self.oracle*state
        state = self.diffusion*state
        state_hist[1] = state
        state = self.mem_op*state
        state = self.diffusion*state
        state_hist[2] = state
        for i in range(M-1):
            state = self.oracle*state
            state = self.diffusion*state
            state_hist[i+3] = state
        return state, state_hist

    def set_oracle(self, oracle=None):
        """
        Instantiates the oracle operator
        """
        N = self.query.shape[0]
        if oracle is None:
            self.oracle = qu.identity(N) - 2*qu.ket2dm(self.query)
        elif isinstance(oracle, qu.Qobj) and oracle.type == "ket":
            self.oracle = qu.identity(N) - 2*qu.ket2dm(oracle)
        else:
            self.oracle = oracle

    def set_diffusion(self, diffusion=None):
        """
        Instantiates the diffusion operator
        """
        if not diffusion:
            N = self.memory.shape[0]
            self.diffusion = 2*qu.ket2dm(self.memory) - qu.identity(N)
        else:
            self.diffusion = diffusion

    def set_state(self, state):
        self.state = state
```

10.1.2 The Maximum Probability Test

The test file used for calculating the ideal retrieval probabilities under the metagenomic scenario suggested in section 7.1.

```
import sys

import Ventura_QuAM

import numpy as np
import matplotlib.pyplot as plt

"""
Test for measuring the maximal probability of retrieval and failure of QuAMs
"""

def int_to_bitstr(integer, bits=None):
    if bits is None:
        return format(integer, "b")
    else:
        return format(integer, "0" + str(bits) + "b")

def hamm_dist(state1, state2, n):
    """
    character hamming distance
    """
    s_state1 = format(state1, "0" + str(n) + "b")
    s_state2 = format(state2, "0" + str(n) + "b")

    return str_dist(s_state1, s_state2)
```

```python
def str_dist(s1, s2):
    dist = 0
    for i in range(int(len(s1)/2)):
        c1 = s1[2*i:2*(i+1)]
        c2 = s2[2*i:2*(i+1)]
        if c1 != c2:
            dist += 1
    return dist

def vicinity_ideal_test(data_file=None, method="ezhov"):
    """Vicinity test

    Match with a distributed query of one target, iterating over number of
    qubits, filling of database, as well as averaging over random choices of
    marked state. Include states in memory, which are at least a hamming
    distance 3 away from mark and also a certain percentage of states within a
    specified distance from target.
    """
    max_bits = 10 # maximum number of qubits
    n_bits = np.arange(4, max_bits+1, 2) # step of 2 to only have DNA sequences
    n_nums = len(n_bits)
    number_of_marks = 10 # number of random marks to test for
    max_dist = 3
    distances = np.arange(1, max_dist+1) # the distance at which to put the target memory state
    ratio_vals = 4 # number of ratio parameters to test for
    mark_ratio_vals = 10
    ratio_range = np.linspace(0.1, 0.9, ratio_vals)
    mark_ratios = np.linspace(0.1, 0.9, mark_ratio_vals)
    peak_factor = 0.4 # 0 is a delta peak at mark
    amp_ratio = 0.1

    data_struct = (n_nums, number_of_marks, len(mark_ratios), ratio_vals, len(distances))
    ventura_probs = np.zeros(data_struct)
    ventura_prob_inds = np.zeros(data_struct)
    target_failure_probs = np.zeros(data_struct)
    memory_failure_probs = np.zeros(data_struct)

    for i1 in range(len(n_bits)): # iterate number of bits
        n = n_bits[i1]
        N = 2**n
        mark_num = min(N, number_of_marks) # number of marks to test
        for i2 in range(number_of_marks): # iterate number of marks
            mark = np.random.randint(0, N)
            for dist_ind in range(len(distances)):
                target_dist = distances[dist_ind]
                max_dist = target_dist
                if target_dist >= int(n/2):
                    continue
                possible_states = [i for i in range(N)]
                dist_states = set()
                mark_projector = qu.Qobj()
                i = 0
                while i < len(possible_states):
                    if possible_states[i] == mark: # remove mark from memory
                        possible_states.pop(i)
                    elif hamm_dist(possible_states[i], mark, n) <= target_dist:
                        dist_states.add(possible_states.pop(i))
                    elif hamm_dist(possible_states[i], mark, n) <= max_dist:
                        possible_states.pop(i)
                    else:
                        i += 1

                mem_states = set()
                for mark_ratio_ind in range(len(mark_ratios)):
                    # prepare target states and projector
                    mark_ratio = mark_ratios[mark_ratio_ind]
                    target_states = set()
                    possible_targets = dist_states.copy()
                    target_num = int(len(possible_targets)*mark_ratio)
                    if target_num < 1:
```

```
        target_num = 1
    mark_projector = qu.Qobj()
    for i in range(target_num):
        state = possible_targets.pop()
        target_states.add(state)
        mark_projector += qu.projection(N, state, state)

    for i3 in range(len(ratio_range)): # iterate ratios
        ratio = ratio_range[i3]

        # create memory list
        mem_states = target_states.copy()

        # number of memory states
        available_states = possible_states.copy()
        N_mem = int(np.ceil(ratio*len(available_states)))

        for j in range(N_mem): # add random memories
            num_ind = np.random.randint(0, len(available_states))
            add_state = available_states.pop(num_ind)
            mem_states.add(add_state)

        # create memory projector
        mem_projector = qu.Qobj()
        for mem in mem_states:
            mem_projector += qu.projection(N, mem, mem)
        # Create complement (spurious) projector to memory
        non_mem_projector = qu.identity(N) - mem_projector
        non_target_mem_projector = mem_projector - mark_projector

        # make query and memory list
        query = [int_to_bitstr(i, n) for i in range(N)]
        weights = np.zeros(N)
        for i in range(N):
            dist = hamm_dist(i, mark, n)
            if dist <= target_dist:
                weights[i] = np.sqrt((peak_factor)**dist * (1-peak_factor)**(n-dist))
        weights[mark] = 0
        n_zero = len(weights[weights == 0])-1
        if n_zero > 0:
            weights[weights == 0] = np.sqrt(np.sum(weights**2)\
            *amp_ratio/(1-amp_ratio))/n_zero
        weights[mark] = 0

        memory = [int_to_bitstr(mem, n) for mem in mem_states]

        # initialize QuAM

        ventura = Ventura_QuAM.QuAM()
        ventura.set_mem(memory)
        ventura.set_query(query, weights=weights)
        ventura.set_oracle()
        ventura.set_diffusion()

        # perform matching
        repeat_case = True
        iteration_cap = int(np.ceil(np.sqrt(len(mem_states))))
        iteration_limit = iteration_cap
        while repeat_case:
            repeat_case = False
            if method == "ezhov":
                _, ventura_states = ventura.match_ezhov(iteration=iteration_limit)
            if method == "improved":
                _, ventura_states = ventura.match_C1(iteration=iteration_limit)

            # gather probabilites
            success_probs = [(mark_projector*state).norm()**2\
            for state in ventura_states]
            max_prob_ind = np.argmax(np.array(success_probs))
            max_prob = success_probs[max_prob_ind]
```

```
                            good_failure_prob = (non_target_mem_projector*ventura_states[max_prob_ind])\
                            .norm()**2
                            bad_failure_prob = (non_mem_projector*ventura_states[max_prob_ind])\
                            .norm()**2
                            print("n ratio mark_ratio dist prob fail_prob bad_fail ind")
                            print(n, format(ratio, "0.3f"),\
                            format(mark_ratio, "0.3f"), target_dist, format(max_prob, "0.4f"),\
                            format(good_failure_prob, "0.4f"), format(bad_failure_prob, "0.4f"),\
                            max_prob_ind)

                            ventura_probs[i1,i2,mark_ratio_ind,i3, dist_ind] = max_prob
                            target_failure_probs[i1,i2,mark_ratio_ind,i3,dist_ind] = good_failure_prob
                            memory_failure_probs[i1,i2,mark_ratio_ind,i3,dist_ind] = bad_failure_prob
                            ventura_prob_inds[i1,i2,mark_ratio_ind,i3, dist_ind] = max_prob_ind

                            if max_prob_ind >= iteration_limit or max_prob_ind <= 0:
                                iteration_limit += iteration_cap
                                repeat_case = True

    # save results
    if data_file is None:
        data_file = "data"

    np.savez(data_file, bits=n_bits, ratios=ratio_range, mark_ratios=mark_ratios,\
    peak_factor=np.array([peak_factor]),
    amp_ratio=np.array([amp_ratio]), distances=distances,\
    values=ventura_probs,
    good_failure=target_failure_probs,\
    bad_failure=memory_failure_probs, inds=ventura_prob_inds)

def plot_vicinity_data(data_file):
    data = np.load(data_file)

    bits = data["bits"]
    ratios = data["ratios"]
    mark_ratios = data["mark_ratios"]
    param = data["peak_factor"]
    vals = data["values"]
    inds = data["inds"]
    distances = data["distances"]
    failure_vals = data["bad_failure"]
    good_failure_vals = data["good_failure"]

    vals[vals == 0] = np.nan

    # project out the ratio parameter and random iterations
    mean_vals = np.nanmean(vals, axis=(1,3))
    sum_vals = np.nansum(vals, axis=(1,3))
    N_vals = sum_vals/mean_vals
    error_vals = np.nanstd(vals, axis=(1,3))/np.sqrt(N_vals)

    failure_vals[failure_vals == 0] = np.nan
    good_failure_vals[good_failure_vals == 0] = np.nan

    failure_vals_means = np.nanmean(failure_vals, axis=(1,3))
    N_vals = np.nansum(failure_vals, axis=(1,3))/failure_vals_means
    failure_vals_errors = np.nanstd(failure_vals, axis=(1,3))/np.sqrt(N_vals)

    good_failure_vals_means = np.nanmean(good_failure_vals, axis=(1,3))
    N_vals = np.nansum(good_failure_vals, axis=(1,3))/good_failure_vals_means
    good_failure_vals_errors = np.nanstd(good_failure_vals, axis=(1,3))/np.sqrt(N_vals)

    for i in range(len(distances)):
        fig = plt.figure()
        ax = fig.gca()
        X,Y = mark_ratios, bits
        Z = mean_vals[:,:,i]


        for y_ind in range(len(Y)):
            if Y[y_ind]/2 > distances[i]:
                ax.errorbar(X, Z[y_ind, :], yerr=error_vals[y_ind,:,i],\
                label=str(int(Y[y_ind]/2)) + " chars", marker='o',\
                markersize=5, capsize=5)

        ax.legend()
        plt.xlabel("Target density")
        plt.ylabel("Probability")
        plt.grid(True)
        plt.ylim([0, 1])

        print("Showing "\
        + "success probability of items up to distance {} from target".format(distances[i]))
        plt.show()

    for i in range(len(distances)):
        fig = plt.figure()
        ax = fig.gca()
        X,Y = mark_ratios, bits
        Z = failure_vals_means[:,:,i]

        for y_ind in range(len(Y)):
            if Y[y_ind]/2 > distances[i]:
                ax.errorbar(X, Z[y_ind, :], yerr=failure_vals_errors[y_ind,:,i],\
                label=str(int(Y[y_ind]/2)) + " chars", marker='o',\
                markersize=5, capsize=5)

        ax.legend()
        plt.xlabel("Target density")
        plt.ylabel("Probability")
        plt.grid(True)
        plt.ylim([0, 1])

        print("Showing "\
        + "false positives probability of items up to distance {} from target".format(distances[i]))
        plt.show()

    for i in range(len(distances)):
        fig = plt.figure()
        ax = fig.gca()
        X,Y = mark_ratios, bits
        Z = good_failure_vals_means[:,:,i]

        for y_ind in range(len(Y)):
            if Y[y_ind]/2 > distances[i]:
                ax.errorbar(X, Z[y_ind, :], yerr=good_failure_vals_errors[y_ind,:,i],\
                label=str(int(Y[y_ind]/2)) + " chars", marker='o',\
                markersize=5, capsize=5)

        ax.legend()
        plt.xlabel("Target density")
        plt.ylabel("Probability")
        plt.grid(True)

        print("Showing "\
        + "false negatives probability of items up to distance {} from target".format(distances[i]))
        plt.show()

name = "filename" # choose filename for saving and loading data

# run the test with a chosen QuAM
vicinity_ideal_test(name, method="improved")

# plot data
plot_vicinity_data(name + ".npz")
```

10.1.3 The Realistic Probability Tests

The code for simulating the QuAM retrieval under realistic conditions, using approximations and heuristics. The exact scenario is determined by the input to the QuAM class object.

```python
import sys

import Ventura_QuAM

import numpy as np
import qutip as qu
import matplotlib.pyplot as plt

"""
Test for measuring the realistic probability of retrieval and failure of QuAMs
"""

def int_to_bitstr(integer, bits=None):
    if bits is None:
        return format(integer, "b")
    else:
        return format(integer, "0" + str(bits) + "b")

def hamm_dist(state1, state2, n):
    """
    character hamming distance
    """
    s_state1 = format(state1, "0" + str(n) + "b")
    s_state2 = format(state2, "0" + str(n) + "b")

    return str_dist(s_state1, s_state2)

def str_dist(s1, s2):
    dist = 0
    for i in range(int(len(s1)/2)):
        c1 = s1[2*i:2*(i+1)]
        c2 = s2[2*i:2*(i+1)]
        if c1 != c2:
            dist += 1
    return dist

def vicinity_ideal_test(data_file=None, method="ezhov", approx_scheme="approx"):
    """Vicinity test

    Match with a distributed query of one target, iterating over number of
    qubits, filling of database, as well as averaging over random choices of
    marked state. Include states in memory, which are at least a hamming
    distance 3 away from mark and also a certain percentage of states within a
    specified distance from target.
    """
    max_bits = 10 # maximum number of qubits
    n_bits = np.arange(4, max_bits+1, 2) # step of 2 to only have DNA sequences
    n_nums = len(n_bits)
    number_of_marks = 10 # number of random marks to test for
    max_dist = 3
    distances = np.arange(1, max_dist+1) # the distance at which to put the target memory state
    ratio_vals = 4 # number of ratio parameters to test for
    mark_ratio_vals = 10
    ratio_range = np.linspace(0.1, 0.9, ratio_vals)
    mark_ratios = np.linspace(0.1, 0.9, mark_ratio_vals)
    peak_factor = 0.4 # 0 is a delta peak at mark
    amp_ratio = 0.1

    data_struct = (n_nums, number_of_marks, len(mark_ratios), ratio_vals, len(distances))
    ventura_probs = np.zeros(data_struct)
    ventura_prob_inds = np.zeros(data_struct)
    target_failure_probs = np.zeros(data_struct)
    memory_failure_probs = np.zeros(data_struct)

    for i1 in range(len(n_bits)): # iterate number of bits
```

```python
        n = n_bits[i1]
        N = 2**n
        mark_num = min(N, number_of_marks) # number of marks to test
        for i2 in range(number_of_marks): # iterate number of marks
            mark = np.random.randint(0, N)
            for dist_ind in range(len(distances)):
                target_dist = distances[dist_ind]
                max_dist = target_dist
                if target_dist >= int(n/2):
                    continue
                possible_states = [i for i in range(N)]
                dist_states = set()
                mark_projector = qu.Qobj()
                i = 0
                while i < len(possible_states):
                    if possible_states[i] == mark: # remove mark from memory
                        possible_states.pop(i)
                    elif hamm_dist(possible_states[i], mark, n) <= target_dist:
                        dist_states.add(possible_states.pop(i))
                    elif hamm_dist(possible_states[i], mark, n) <= max_dist:
                        possible_states.pop(i)
                    else:
                        i += 1

                mem_states = set()
                for mark_ratio_ind in range(len(mark_ratios)):
                    # prepare target states and projector
                    mark_ratio = mark_ratios[mark_ratio_ind]
                    target_states = set()
                    possible_targets = dist_states.copy()
                    target_num = int(len(possible_targets)*mark_ratio)
                    if target_num < 1:
                        target_num = 1
                    mark_projector = qu.Qobj()
                    for i in range(target_num):
                        state = possible_targets.pop()
                        target_states.add(state)
                        mark_projector += qu.projection(N, state, state)

                    for i3 in range(len(ratio_range)): # iterate ratios
                        ratio = ratio_range[i3]

                        # create memory list
                        mem_states = target_states.copy()

                        # number of memory states
                        available_states = possible_states.copy()
                        N_mem = int(np.ceil(ratio*len(available_states)))

                        for j in range(N_mem): # add random memories
                            num_ind = np.random.randint(0, len(available_states))
                            add_state = available_states.pop(num_ind)
                            mem_states.add(add_state)

                        # create memory projector
                        mem_projector = qu.Qobj()
                        for mem in mem_states:
                            mem_projector += qu.projection(N, mem, mem)
                        # Create complement (spurious) projector to memory
                        non_mem_projector = qu.identity(N) - mem_projector
                        non_target_mem_projector = mem_projector - mark_projector

                        # make query and memory list
                        query = [int_to_bitstr(i, n) for i in range(N)]
                        weights = np.zeros(N)
                        for i in range(N):
                            dist = hamm_dist(i, mark, n)
                            if dist <= max_dist:
                                weights[i] = np.sqrt((peak_factor)**dist * (1-peak_factor)**(n-dist))
                        weights[mark] = 0
                        n_zero = len(weights[weights == 0])-1
                        if n_zero > 0:
                            weights[weights == 0] = np.sqrt(np.sum(weights**2)*amp_ratio\
                            /(1-amp_ratio))/n_zero
                        weights[mark] = 0

                        memory = [int_to_bitstr(mem, n) for mem in mem_states]

                        # initialize QuAM

                        ventura = Ventura_QuAM.QuAM()
                        ventura.set_mem(memory)
                        ventura.set_query(query, weights=weights)
                        ventura.set_oracle()
                        ventura.set_diffusion()

                        # perform matching
                        repeat_case = True
                        iteration_cap = int(np.ceil(np.sqrt(N)))
                        iteration_limit = iteration_cap
                        if method == "ezhov":
                            _, ventura_states = ventura.match_ezhov(iteration=approx_scheme)
                        if method == "improved":
                            _, ventura_states = ventura.match_C1(iteration=approx_scheme)

                        # gather probabilites
                        success_prob = (mark_projector*ventura_states[-1]).norm()**2
                        max_prob_ind = len(ventura_states)
                        good_failure_prob = (non_target_mem_projector*ventura_states[-1]).norm()**2
                        bad_failure_prob = (non_mem_projector*ventura_states[-1]).norm()**2
                        print("n ratio mark_ratio dist prob fail_prob bad_fail ind")
                        print(n, format(ratio, "0.3f"), format(mark_ratio, "0.3f"), target_dist,\
                        format(success_prob, "0.4f"), format(good_failure_prob, "0.4f"),\
                        format(bad_failure_prob, "0.4f"), max_prob_ind)

                        ventura_probs[i1,i2,mark_ratio_ind,i3, dist_ind] = success_prob
                        target_failure_probs[i1,i2,mark_ratio_ind,i3,dist_ind] = good_failure_prob
                        memory_failure_probs[i1,i2,mark_ratio_ind,i3,dist_ind] = bad_failure_prob
                        ventura_prob_inds[i1,i2,mark_ratio_ind,i3, dist_ind] = max_prob_ind

    # save results
    if data_file is None:
        data_file = "data"

    np.savez(data_file, bits=n_bits, ratios=ratio_range, mark_ratios=mark_ratios,\
    peak_factor=np.array([peak_factor]), amp_ratio=np.array([amp_ratio]), distances=distances,\
    values=ventura_probs, good_failure=target_failure_probs, bad_failure=memory_failure_probs,\
    inds=ventura_prob_inds)

def plot_vicinity_data(data_file):
    data = np.load(data_file)

    bits = data["bits"]
    ratios = data["ratios"]
    mark_ratios = data["mark_ratios"]
    param = data["peak_factor"]
    vals = data["values"]
    inds = data["inds"]
    distances = data["distances"]
    failure_vals = data["bad_failure"]
    good_failure_vals = data["good_failure"]

    vals[vals == 0] = np.nan
    failure_vals[failure_vals == 0] = np.nan
    good_failure_vals[good_failure_vals == 0] = np.nan

    # project out the ratio parameter and random iterations
    mean_vals = np.nanmean(vals, axis=(1,3))
    sum_vals = np.nansum(vals, axis=(1,3))
    N_vals = sum_vals/mean_vals
    error_vals = np.nanstd(vals, axis=(1,3))/np.sqrt(N_vals)

    failure_vals[failure_vals == 0] = np.nan
    good_failure_vals[good_failure_vals == 0] = np.nan

    failure_vals_means = np.nanmean(failure_vals, axis=(1,3))
    N_vals = np.nansum(failure_vals, axis=(1,3))/failure_vals_means
    failure_vals_errors = np.nanstd(failure_vals, axis=(1,3))/np.sqrt(N_vals)

    good_failure_vals_means = np.nanmean(good_failure_vals, axis=(1,3))
    N_vals = np.nansum(good_failure_vals, axis=(1,3))/good_failure_vals_means
    good_failure_vals_errors = np.nanstd(good_failure_vals, axis=(1,3))/np.sqrt(N_vals)

    print("vals shape", mean_vals.shape)

    for i in range(len(distances)):
        fig = plt.figure()
        ax = fig.gca()
        X,Y = mark_ratios, bits
        Z = mean_vals[:,:,i]

        for y_ind in range(len(Y)):
            if Y[y_ind]/2 > distances[i]:
                ax.errorbar(X, Z[y_ind, :], yerr=error_vals[y_ind,:,i],\
                label=str(int(Y[y_ind]/2)) + " chars", marker='o',\
                markersize=5, capsize=5)

        ax.legend()
        plt.xlabel("Target density")
        plt.ylabel("Probability")
        plt.grid(True)
        plt.ylim([0, 1])

        print("Showing "\
        + "success probability of items up to distance {} from target".format(distances[i]))
        plt.show()

    for i in range(len(distances)):
        fig = plt.figure()
        ax = fig.gca()
        X,Y = mark_ratios, bits
        Z = failure_vals_means[:,:,i]

        for y_ind in range(len(Y)):
            if Y[y_ind]/2 > distances[i]:
                ax.errorbar(X, Z[y_ind, :], yerr=failure_vals_errors[y_ind,:,i],\
                label=str(int(Y[y_ind]/2)) + " chars", marker='o',\
                markersize=5, capsize=5)

        ax.legend()
        plt.xlabel("Target density")
        plt.ylabel("Probability")
        plt.grid(True)
        plt.ylim([0, 1])
        print("Showing "\
        + "false positives probability of items up to distance {} from target".format(distances[i]))
        plt.show()

    for i in range(len(distances)):
        fig = plt.figure()
        ax = fig.gca()
        X,Y = mark_ratios, bits
        Z = good_failure_vals_means[:,:,i]

        for y_ind in range(len(Y)):
            if Y[y_ind]/2 > distances[i]:
                ax.errorbar(X, Z[y_ind, :], yerr=good_failure_vals_errors[y_ind,:,i],\
                label=str(int(Y[y_ind]/2)) + " chars", marker='o',\
                markersize=5, capsize=5)
```

```python
        ax.legend()
        plt.xlabel("Target density")
        plt.ylabel("Probability")
        plt.grid(True)
        print("Showing "\
        + "false negatives probability of items up to distance {} from target".format(distances[i]))
        plt.show()

name = "filename" # choose filename for saving and loading data

# run the test with a chosen QuAM and iteration strategy
vicinity_ideal_test(name, method="improved/ezhov", approx_scheme="exact/approx/semi_approx")

# plot data
plot_vicinity_data(name + ".npz")
```

10.2 Trugenberger Model Calculations

This section includes a small script used for the numerical solving in fig. 22b.

```python
import numpy as np
import matplotlib.pyplot as plt

def ratio(n,x,D,B):
    """
    Calculates the priority ratio of states below the limit D, and those above
    """
    denom = np.cos(np.pi/n*x)**(2*B);
    nom = np.cos(np.pi/n*x)**(2*B);
    denom = denom[x<=D];
    denom = np.sum(denom);
    nom = np.sum(nom);
    r = denom/nom;
    return r

# parameters to test for, assuming using 62-bit patterns, i.e. 31 character sequences
n = 62;
D = np.arange(1,int(n/2));
d_vals = np.arange(1,int(n/2));
B_vals = np.arange(1,int(n/2));
threshold = 0.9;

# iterates to find the B value where ratio criteria is fulfilled
for i in range(len(d_vals)):
    f = lambda B : ratio(n, d_vals, D[i], B) - threshold;
    for B in range(10000):
      if f(B) >= 0:
            B_vals[i] = B
            break
B_vals[B_vals == 0] = 1

plt.semilogy(d_vals, B_vals, marker="o", markersize=5)
plt.grid(True)
plt.xlabel("Distance limit, $D$")
plt.ylabel("Auxiliary bits, $B$")
plt.show()
```

10.3 Quantum and Simulated Annealing Code

Here, the code used for encoding and running the multiple alignment problem on a quantum annealer is included.

10.3.1 Hamiltonian Construction

This code defines functions for constructing the Hamiltonian used for solving the MSA problem. It takes the sequences, parameters for the cost function and encoding parameters such as the number of columns, as input, and returns the QUBO form of the problem.

```python
import numpy as np

from collections import OrderedDict

def get_MSA_qubitmat(sizes, weights, gap_pen=0, extra_inserts=0, allow_delete=False, coeffs=None):
    """Generate Hamiltonian for Multiple Sequence Alignment (MSA) column formulation,
    as minimization problem.

    Args:
        sizes - size of each sequence : 1D np.array
        weights - weight/cost matrix between elements in sequences (s1,n1,s2,n2) : 4D np.array
        gap_pen - gap penaly : float
        extra_inserts - number of extra inserts to allow : int
        allow_delete - Flag for allowing deletion of elements in problem : Bool

    Returns:
        spin interaction (QUBO) matrix,
        energy shift for the hamiltonian
        and index scheme to map spin indices to indices for sequence,
        base element and position and vice versa.

    Goals:
        1 Minimize cost function
        2 Place every element somewhere
        3 Respect element order
    """

    L = len(sizes) # number of sequences
    n_tot = np.sum(sizes)
    N = np.max(sizes) # max number of elements in sequences
    num_pos = N + extra_inserts + 1*allow_delete # number of positions

    """Number of spins x_{s,n,i}
    s in [1, L]
    n in [1, sizes(s)]
    i in [1, N + extra + {1 if deletion}]
    number of spins = n_tot*positions
    """
    num_spins = n_tot*num_pos
    pauli_list = []
    shift = 0

    spin_mat = np.zeros((num_spins, num_spins))

    if coeffs is None:
        match_max = np.max(np.max(np.max(np.max(weights))))
        match_max = max(match_max, gap_pen)
        """
        maximum penalty given by (maximum row size)*sum_0_to_n(i*(i-1))*max_cost
        sum0_to_n(i*(i-1)) = [n(n+1)(2n+1)/6] - [n(n+1)/2]\
        = [n(n+1)/2][(2n+1)/3 - 1] = n(n+1)(n-1)/3 = n(n^2-1)/3
        """
        sum_0_to_n = lambda x: x*(x**2-1)/3
        max_penalty = (np.max(sizes) + extra_inserts)*sum_0_to_n(len(sizes))*match_max
        coeffs = [1, max_penalty]
        print("max_penalty 1=", max_penalty)

        """
        More tight penalty, given by math this time
        P = min(-N*max(2g - w) - L, L + N*min(w - 2g))
        """
        shifted_weights = 2*gap_pen - weights
        max_val = np.amax(shifted_weights, axis=(0,1,2,3))
        min_val = np.amin(-shifted_weights, axis=(0,1,2,3))
        max_penalty = np.abs(min([-n_tot*max_val - gap_pen*L, gap_pen*L + n_tot*min_val]))
        print("max_penalty 2=", max_penalty)
        coeffs = [1, max_penalty]

    A = coeffs[0]     # cost function coefficient
    B = coeffs[1]     # placement coefficient
    C = coeffs[1]     # order coefficient

    def pos2ind(s, n, i):
        """Return spin index from sequence, element and position indices
        Index scheme: first N*L spins are 'removal' spins
                      the following L*N*num_pos spins are location spins
        """
        return int((np.sum(sizes)*L)*allow_delete \
                + (np.sum(sizes[:s]) + n)*num_pos + i)

    rev_ind_scheme = np.empty(num_spins, dtype=tuple)
    for s in range(L):
        for n in range(sizes[s]):
            for i in range(num_pos):
                rev_ind_scheme[pos2ind(s,n,i)] = (s, n, i)

    def to_bool_arr(arr):
        length = int(np.ceil(np.log2(np.max(arr))))
        bool_arr = [0]*len(arr)
        for i in range(len(arr)):
            bin_string = format(arr[i], "0" + str(length) + "b")
            bool_arr[i] = np.array([x == '1' for x in bin_string], dtype=np.bool)
        return bool_arr

    def add_pauli_bool(coeff, *inds):
        nonlocal shift
        nonlocal spin_mat
        inds_arr = np.zeros(len(inds), dtype=np.int32)
        if len(inds) > 2 or len(inds) <= 0:
            raise ValueError("Invalid number of indices, must be on QUBO form")

        for i in range(len(inds)):
            s,n,j = inds[i]
            inds_arr[i] = pos2ind(s, n, j)

        if len(inds_arr) == 1:
            spin_mat[inds_arr[0], inds_arr[0]] += coeff
        elif len(inds_arr) >= 2:
            inds_arr = np.sort(inds_arr)
            spin_mat[tuple(inds_arr)] += coeff

    """Cost function (Goal 1)
    Matching at same position
    H_matching = A*sum_{s1,s2} sum_{n1,n2} sum_i w_{s1,s2,n1,n2} * x_{s1,n1,i}*x_{s2,n2,i}
    """
    for s1 in range(L):
        for s2 in range(s1+1,L):
            for n1 in range(sizes[s1]):
                for n2 in range(sizes[s2]):
                    for i in range(num_pos):
                        # matching cost
                        w = weights[s1,n1,s2,n2]
                        add_pauli_bool(A*w, (s1,n1,i), (s2,n2,i))

    """Penalties version 2 (pair of sum penalties)
    Pairing with gaps
    H_gap = A*sum_{s1,n1}sum_{s2}sum_i g*x_{s1,n1,i}(1 - sum_n2 x_{s2,n2,i})
    Represents pairing of (s1,n1) at i to nothing in s2
    """
    if gap_pen != 0:
        for s1 in range(L):
            for n1 in range(sizes[s1]):
                for s2 in range(L):
                    for i in range(num_pos):
                        w = A*gap_pen

                        add_pauli_bool(w, (s1, n1, i))
                        for n2 in range(sizes[s2]):
                            add_pauli_bool(-w, (s1, n1, i), (s2, n2, i))


    """Placement terms
    Place at only one position
    H_placement = B*sum_{s,n} (1-sum_i x_{s,n,i})^2
    = B*n_tot - 2*B*sum_{s,n}sum_i x_{s,n,i} \
    + sum_{s,n} sum_{i,j} x_{s,n,i}x_{s,n,j}
    = B*n_tot - B*sum_{s,n}sum_i x_{s,n,i} + B*sum_{s,n}sum_{i!=j} x_{s,n,i}x_{s,n,j}
    """
    shift += B*n_tot
    for s in range(L):
        for n in range(sizes[s]):
            for i in range(num_pos):
                for j in range(num_pos):
                    if i != j:
                        w = B
                        add_pauli_bool(w, (s,n,i), (s,n,j))
                    else:
                        w = -B
                        add_pauli_bool(w,(s,n,i))


    """Order terms
    no deletions
    H_order_no_del = C*sum_{s,n} sum_{i<=j} x_{s,n,j}x_{s,n+1,i}
    with deletions
    H_order = C*sum_s sum_{n1<n2}sum_{i<=j} x_{s,n1,j}x_{s,n2,i}
    """
    for s in range(L):
        if allow_delete:
            for n1 in range(sizes[s]):
                for n2 in range(n1):
                    for i in range(num_pos):
                        for j in range(i+1):
                            w = C
                            add_pauli_bool(w,(s,n1,i),(s,n2,j))
        else:
            for n in range(sizes[s]-1):
                for i in range(num_pos):
                    for j in range(i+1):
                        w = C
                        add_pauli_bool(w, (s,n,i), (s,n+1,j))

    return spin_mat, shift, rev_ind_scheme

def get_alignment_string(sequences, gaps, positions):
    string_size = max([len(s) for s in sequences]) + gaps
    align_strings = [["-" for i in range(string_size)] for i in range(len(sequences))]
    for (key, value) in positions.items():
        align_strings[key[0]][value] = sequences[key[0]][key[1]]
    return align_strings
```

10.3.2 Annealing Tests

This section includes two code files, one for running the given sequences on an annealer and one for analyzing and plotting the result.

```python
import MSA_column
from data_formats import SeqQuery
import numpy as np
import dimod
import neal
import pickle

from dwave.system.samplers import DWaveSampler
from dwave.system.composites import EmbeddingComposite

samples = 1000 # number of samples to run
save_file = "filename"

sequences = ["AT", "T", "T", "A"] # write your sequences here
simulation = False # flag determining to run simulated or quantum annealer

# rewards (negative) and costs (positive)
match_cost = -1
mismatch_cost = 1

# quantum spin column version
sizes = [len(sequences[i]) for i in range(len(sequences))]
# calculate weights for matching
matchings = np.zeros((len(sequences), max(sizes), len(sequences), max(sizes)))
for s1 in range(len(sequences)):
    for s2 in range(len(sequences)):
        for n1 in range(sizes[s1]):
            for n2 in range(sizes[s2]):
                if sequences[s1][n1] == sequences[s2][n2]:
                    matchings[s1,n1,s2,n2] = match_cost
                else:
                    matchings[s1,n1,s2,n2] = mismatch_cost

inserts = 0 # the number of extra gaps in addition the longest sequence length
gap_penalty = 0 # the gap penalty
params = {"gap_pen": gap_penalty, "extra_inserts": inserts}
mat, shift, rev_inds = MSA_column.get_MSA_qubitmat(sizes, matchings,\
gap_pen=gap_penalty, extra_inserts=inserts)

def mat_to_dimod_format(matrix):
    """
    interprets the QUBO matrix of interactions into separate format for linear and interaction terms
    """
    n = matrix.shape[0]
    linear = {}
    interaction = {}
    for i in range(n):
        linear[i] = matrix[i,i]
        for j in range(i+1,n):
            interaction[(i,j)] = matrix[i,j]
    return linear, interaction

h, J = mat_to_dimod_format(mat)
if not simulation:
    cont = input("Continue? y/n ")
    if cont != "y":
        exit()

bqm = dimod.BinaryQuadraticModel(h,J, shift, dimod.BINARY)
if simulation:
    solver = neal.SimulatedAnnealingSampler()
else:
    # WARNING! requires setup of D-wave certificate file beforehand
    solver = EmbeddingComposite(DWaveSampler())
response = solver.sample(bqm, num_reads=samples)

data_query = SeqQuery()
data_query.sequences = sequences
data_query.params = params
data_query.costs = [match_cost, mismatch_cost, gap_penalty]
data_query.spin_mat = mat
data_query.spin_shift = shift
data_query.rev_inds = rev_inds
data_query.h = h
data_query.J = J
data_query.response = response

# pickle results
file = open(save_file, "wb")
pickle.dump(data_query, file, pickle.HIGHEST_PROTOCOL)
file.close()
```

TRITA -SCI-GRU 2019:219