Quantum Natural Language Processing

D1.1

Overview of DisCo Algorithms & Methods for Testing and Evaluation

---

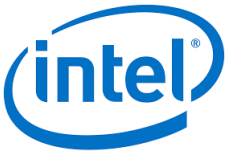
Authors

Lee J. O'Riordan, ICHEC

Myles Doyle, ICHEC

Venkatesh Kannan, ICHEC

April 15, 2019



Executive Summary

In Q2 2018, Intel Ireland and the Irish Centre for High-End Computing (ICHEC) identified the opportunity to leverage the Intel Quantum Simulator (qHiPSTER) to port and implement the quantum version of an existing compositional semantics NLP algorithm to analyse the meaning sentences in a given corpus. Funded by both Intel Ireland and Enterprise Ireland, this project has the following aims: (1) to leverage and evaluate the computing power that quantum devices can offer to computation intensive NLP algorithms such as in the distributional compositional semantics model; (2) to develop the ecosystem of proof-of-concept applications ported to the emerging quantum computing domain, particularly using a highly relevant application domain such as NLP; and (3) to pioneer a collaborative innovation environment in Ireland between industry and research organisations to develop expertise to program quantum computers.

In this deliverable, we report on the strategies to map an existing NLP model called DisCo (Distributional Compositional Semantics) and its component algorithms to quantum computing platforms, particularly Intel qHiPSTER. The strategies address computing and encoding the meaning spaces of a given sentence into quantum states, and computing the closeness of the meanings of two sentences. This report also presents the representative corpora and the methods that will be used to test and evaluate the implementations of the algorithms, along with an overview of the details of the Intel qHiPSTER installation on the Irish national supercomputer (Kay). We also present an overview of the open challenges that have been identified and the workarounds that are employed, along with a summary of the next steps in the project towards implementing the solution on Intel qHiPSTER.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Natural Language Processing and Quantum Advantage

Natural language processing (NLP) is often used to perform tasks such as machine translation, sentiment analysis, relationship extraction, word sense disambiguation and automatic summary generation. Most traditional NLP algorithms for these problems are defined to operate over strings of words, and are commonly referred to as the "bag of words" approach. The challenge, and thus limitation, of this approach is that the algorithms analyse sentences in a corpus based on meanings of the component words and lack information from the grammatical rules and nuances of the language. Consequently, the qualities of results of these traditional algorithms are often unsatisfactory when the complexity of the problem increases.

On the other hand, an alternate approach called "compositional semantics" incorporates the grammatical structure of sentences in a language into the analysis algorithms. Compositional semantics algorithms include the information flows between words in a sentence to determine the meaning of the whole sentence. One such model is "distributional compositional semantics" (DisCo) [1], which is based on tensor product composition to give a grammatically informed algorithm that computes the meaning of sentences and phrases. This algorithm has been noted to potentially offer improvements to the quality of results, particularly for more complex sentences, in terms of memory and computational requirements. However, the main challenge in its implementation is the need for large classical computational resources.

For instance, given a corpus whose word meaning space is based on 2000 most common words, Table 1 presents the comparison between classical and quantum storage requirements [1].

Classical vs. Quantum Storage Requirements		
	1 transitive verb	10K transitive verbs
Classical	1 GB	10 TB
Quantum	33 qubits	47 qubits

Table 1: Quantum Advantage for Storage

1.2 Objective

The distributional compositional semantics (DisCo) model was originally developed by its authors with direct inspiration from quantum theory, and present the quantum version of the DisCo model based on two algorithms [1]:

- **Closest vector problem**: An algorithm for the "closest vector problem" is used to determine the word/phrase out of a set of words/phrases that has the closest relation (for instance, meaning) to a given word/phrase. This finds application in many computational linguistic tasks such as text classification, word/phrase similarity, test classification and sentiment analysis.

- **CSC sentence similarity**: This algorithm is an adaptation of the "closest vector problem" quantum algorithm to perform sentence similarity calculations in the distributional compositional framework. This algorithm is based on tensor product composition that gives a grammatically informed algorithm to compute meaning of sentences/phrases and stores the meanings in quantum systems.

The objective of this project is to implement the two quantum algorithms ("closest vector problem" and "sentence similarity") of the DisCo model on the Intel Quantum Simulator (qHiPSTER). Given a corpus, the implemented solution aims to compute the meanings of two sentences (built from words in the corpus) and decide if their meanings match.

1.3 Structure of the Document

The rest of the document is structured as follows: Sections 2 and 3 present abstractions of the DisCo algorithms that maps them to the quantum Dirac notation and discusses the strategies for their implementation on the Intel qHiPSTER, respectively; Section 4 presents the representative corpora that are used during the implementation of the algorithms for their testing and evaluation; Section 5 summarises the methods that will be used to test and evaluate the implementation of the algorithms; Section 6 presents an overview of the Intel qHiPSTER installation on the Irish national super computer (Kay); Section 7 summarises and discusses the next steps towards implementing the algorithms.

2 Abstraction of DisCo Algorithms

To implement the DisCo model algorithms, we first consider the mappings between DisCo's graphical notation and quantum Dirac notation. Figure 1 represents the meaning space and mappings necessary to understand the sentence "MARY LIKES JOHN", and follows directly from the work of Coecke et al. [2]. Above the dotted line, the blue triangles indicate each component (word) of the sentence. The lines beneath the sentence entities indicate the space in which their meanings exist; **P** and $\mathbf{P}^*$ indicate the meaning space of nouns and their conjugates, **S** is the overall meaning of the sentence, and verbs exist in the composite meaning space of NOUN-SENTENCE-NOUN. Below the dotted line, connecting these wires allows us to perform computations on this abstract formalism, and is akin to contractions over tensor-network diagrams commonly used in condensed matter systems [3, 4]. By performing these "U"-shaped wirings, we contract these tensor indices, with the remaining free-index in the space **S** (the sentence meaning).

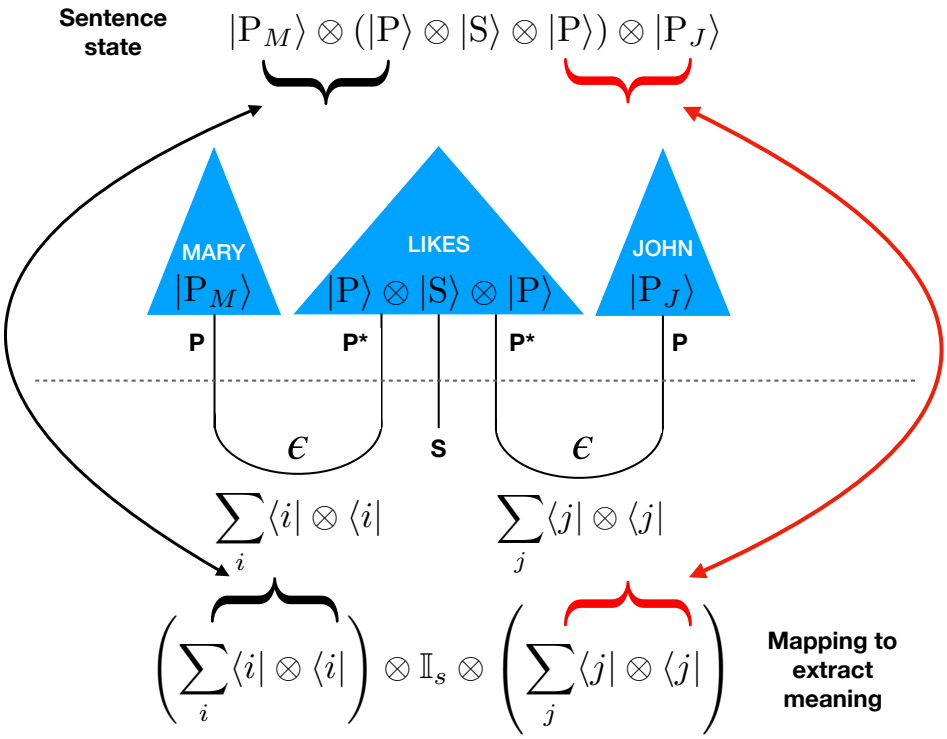


Figure 1: Structural mapping of DisCo graphical notation to quantum Dirac notation.

This approach will require implementations of text-analysis, resource allocations, and mappings between the generated quantum states. We aim to develop a framework to ensure these mappings for sentence state-creation and meaning extraction are at high-level to enable ease of extensibility and development for more complex sentence structures. We propose the following algorithmic layout to overseeing the implementation of this work.

Algorithm 1 is used to tokenise and tag the text in a given corpus. Using this, we can determine in which respective space a token (word) exists, as shown by the wiring in Figure 1 and create the appropriate quantum state to represent its meaning.

Algorithm 1: Corpus to meaning-space tagging and encoding

Data: corpus strings
Result: data encodings, tagged words, and meaning-spaces
load corpus strings;
tokenise corpus strings;
while *not processed all corpus tokens* **do**
    read current token;
    if *token* $\in \{ x \mid x$ *is a noun* $\}$ **then**
        tag token as noun;
    else if *token* $\in \{ x \mid x$ *is a verb* $\}$ **then**
        tag token as verb;
    else
        ignore token;
    end
    generate unique bit-string encoding for token;
    add tagged token to the tagged set;
    token $\rightarrow$ next token;
end
write tagged and encoded tokens to external DB;

Following this step, we use Algorithm 2 to generate the necessary quantum gate operations to encode the corpus data. This generates the quantum state with the encoded meanings, again as presented by Figure 1. With this state, we can now examine the closest vectors and sentence similarities, through appropriately "wiring" test states to the resulting quantum state.

Algorithm 2: Corpus meaning-space quantum state creation

Data: tagged and encoded tokens,
Result: quantum state representation of meaning-space
load tagged and encoded data, D;
initialise data quantum register, $|D\rangle$ to $|00\cdots0\rangle$;
for $i \leftarrow D[1]$ **to** $D[n]$ **do**
    generate quantum gate sequence to realise $D[i]$ in quantum circuit;
    apply gate sequence to update $|D\rangle$ state with $D[i]$ data;
end

Following the decomposition of a corpus using Algorithm 1 and subsequent encoding using Algorithm 2, we use Algorithm 3 to evaluate the closest vector for a given encoded data set with their quantum states. Algorithm 3 presents a method for distance calculation between a test data vector (state) and the encoded data, and returns the representative vector (state) with closest meaning to that of the test as the output. By appropriately training the encoding data, we can affect the result, and as such different text corpora will likely produce different resulting values.

Algorithm 3: Closest vector problem using the DisCo framework.

Data: quantum state representation of meaning-space $|\Psi\rangle$, quantum state representation of test token $|s\rangle$

Result: quantum state representation of meaning-space $|\Psi\rangle$ with amplitudes proportional to the distance between $|s\rangle$ and each state in $|\Psi\rangle$

load $|\Psi\rangle$ using Algorithm 2;
similarly load $|s\rangle$ into separate register;
update amplitudes using distance calculation between $|s\rangle$ and each state in $|\Psi\rangle$;

As a natural extension to Algorithm 3, we may encode a variety of test and data states, and determine their similarity through an inner-product evaluation, defined by Algorithm 4.

Algorithm 4: Sentence similarity problem using the DisCo framework

Data: quantum state data tags and tokens, O_d

Data: quantum state test tags and tokens, O_t

Result: numerical similarity between sentences trained on corpus data

allocate quantum register, $|\Psi\rangle$ of length $2n + 2$;
foreach *token* $j \in \mathrm{O}_d$ **do**
    encode j into $|\Psi\rangle$ using Algorithm 2 for indices $1 \to n$
end
foreach *token* $k \in \mathrm{O}_t$ **do**
    encode k into $|\Psi\rangle$ using Algorithm 2 for indices $n + 1 \to 2n$
end
use Algorithm 3 to compare test with encoded data;
return inner-product value of comparison;

3 Mapping DisCo Algorithms on Intel Quantum Simulator

In this section, we discuss the details related to the implementation of the algorithms presented in Section 2. The encoding of classical data into a quantum system can be achieved through a variety of means, though they can be mapped to two different approaches: state (digital) encoding, or amplitude (analogue) encoding [5, 6]. For our work on the QNLP project the use of a state encoding method for corpus representation offered the best mapping to our problem. Though, much of the problem can be described and implemented using this digital approach, the use of amplitude-based methods are well suited when results are to be obtained. We aim to operate in a mixed-mode approach: encoding and querying our data using state encoding, and obtaining the best result through amplitude modification and measurement.

The nearest-neighbour and sentence similarity algorithms rely on the encoding of a large data-set into the quantum domain, and will follow the approaches discussed in Algorithms 1, 2, 3, and 4. Upon these layers, we will implement quantum state engineering and control methods as discussed in the work of Zeng and Coecke [1], and supplemented by Wiebe et al. [7] and Trugenburger [8, 9]. At a high level, these methods will allow the discovery of the closest encoded vector (state) to a given test vector (state), and additionally the similarity between meanings of an encoded sentence structure, and that of a test.

3.1 Design Considerations

The following discussion applies to the implementation of the methods proposed in Algorithms 1 and 2, as well as extensions to the closest vector and sentence similarity solutions presented in Algorithms 3 and 4. Figure 2 illustrates the control flow of the planned implementation.

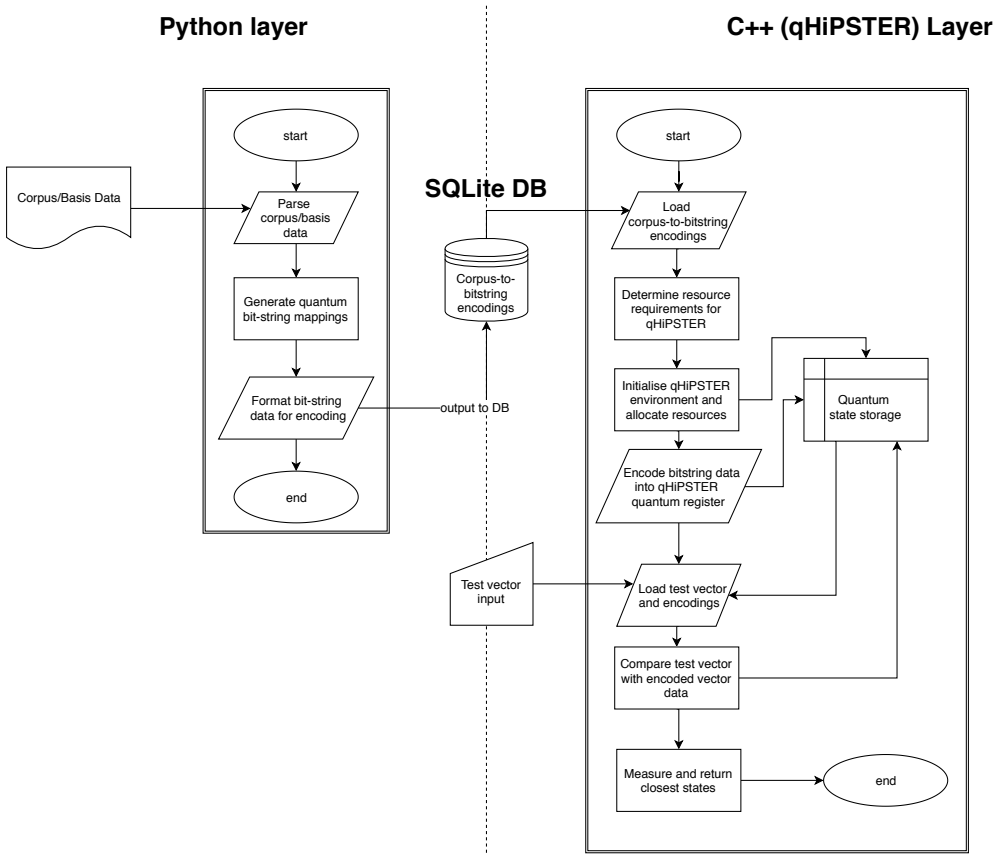


Figure 2: Control flow of proposed QNLP project implementation.

The design of this QNLP software solution is based on a two-level approach: (*i*) the corpus analysis and formatting layer, and (*ii*) the quantum processing layer. Layer (*i*) will be written in Python, to avail of ease-of-use in text processing and analysis. Since this layer will be called at the initial outset of the program run, the performance impact will be negligible compared to the DisCo methods. Layer (*ii*) will be written entirely in C++, and will directly leverage the Intel Quantum Simulator (qHiPSTER). The pre-processed data from layer (*i*) will be loaded, and encoded into qHiPSTER to generate the full meaning-space of the analysed corpus.

Pre-computation: To analyse and prepare the corpus data for encoding into the quantum state-space, we have determined that the use of well-defined classical routines for corpus tokenisation and tagging can be achieved using the NLTK suite [10]. Following the pre-compute step, all required data and meta-data for the qHiPSTER encoding and processing will be stored in an intermediary data format. This approach follows the methods discussed by Algorithm 1.

Alternative approach to QRAM methods: The methods proposed by Zeng and and Coecke [1] rely on the use of a quantum random access memory (QRAM) [11] model to achieve the data access bounds proposed for the closest vector and sentence similarity problems. However, the existence and use of realisable QRAM models in quantum computing algorithms is currently assumed, yet unimplemented in practice. This is in part due to the difficulties in realising the QRAM model as originally proposed (3-level quantum states required), or the large resource requirements necessary for a qubit-implementable model [12]. This renders it almost unusable for real-world problems in current physical or simulated quantum computing systems with low numbers of qubits. Due to the limited number of qubits which we aim to work with, we deemed it necessary to find an alternative approach to realising the state-creation and distance methods, and subsequently Algorithms 3 and 4. For the purpose of state creation, we aim to follow the approach of Trugenberger [8, 9], encoding the data by "carving-off" probability slices and encoding the required state information into these slices. Following this, we use an additional qubit register to store a test data vector to compare with. We make use of the Hamming distance metric between the encoded data and the test input to determine the closest encoded state.

This method allows us to avoid the use of a QRAM implementation, at the cost of an iterative state-preparation routine, as we feed in each bit-string to be encoded. The Hamming distance takes advantage of the inherent quantum parallelism offered by the register, and as such can be computed over the entire superposition state simultaneously.

At this stage, we have several options to determine the closest vector to a known data set: follow the approaches of Trugenberger [8, 9], Schuld et al. [13], Wiebe et al. [7], or follow alternative approaches defined by use of the the Durr-Hoyer (DH) algorithm [14] or phase shifting. These alternative approach are discussed in further detail in Appendices B and C.

It may be instructive to implement many of the above mentioned approaches to determine the optimal method for the closest vector problem, using the encoding strategy and similarity calculations from [8, 9], the amplitude-based approaches proposed separately by [13] and [7], as well as the modified Hamming approaches discussed above. Given any successful implementation of these methods for the closest vector problem, the extension to the sentence similarity problem is trivial.

We can now discuss the elements of these methods as we see them mapping to our problem.

3.2 Encoding of Classical Bit Strings to Superposition of States

As discussed above in Section 3.1, the sequential methods for encoding binary strings to a superposition of states proposed by Trugenberger [8, 9] will be used for encoding the binary representation of our corpus into a superposition of quantum states.

For a set of N binary patterns $p^i = \{p^i_1, \ldots, p^i_n\}$ for $i = 1, \ldots, N$ of length n being encoded, we can encode these patterns into a quantum superposition of states such that

$$|m\rangle = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |p^i\rangle. \tag{1}$$

Trugenberger developed a sequential method for creating the superposition $|m\rangle$ in [8], by first loading a pattern into an auxiliary register, which was then copied into $|m\rangle$, the memory qubit register. This process being repeated for each binary pattern to encode, "carving" off the newly encoded state term from the already encoded terms while maintaining the appropriate normalization factors in the amplitudes. This method required $2n+2$ qubits. It will be referred to as method 1.

Trugenberger generalizes this method in [9] which generates the superposition of states $|m\rangle$, by applying a unitary matrix to the state $|0_1 \dots 0_n\rangle$. The process is repeated for each input binary pattern using the aforementioned "carving" off method to store the new term in a state. This method requires only $n+2$ qubits, which is a significant reduction from method 1. This will be referred to as method 2.

The main idea of both methods is detailed below:

Method 1:

$$\begin{aligned}
|\psi_0\rangle &= |p_1^1, \dots, p_n^1\rangle |01\rangle |0_1, \dots, 0_n\rangle \\
|\psi_1\rangle &= \prod_{j=1}^{n} \mathrm{2XOR}_{p_j^i u_2 m_j} |\psi_0\rangle \\
|\psi_2\rangle &= \prod_{j=1}^{n} \mathrm{NOT}_{m_j} \mathrm{XOR}_{p_j^i m_j} |\psi_1\rangle \\
|\psi_3\rangle &= \mathrm{nXOR}_{m_1 \dots m_n u_1} |\psi_2\rangle \\
|\psi_4\rangle &= \mathrm{CS}_{u_1 u_2}^{p+1-i} |\psi_3\rangle \\
|\psi_5\rangle &= \mathrm{nXOR}_{m_1 \dots m_n u_1} |\psi_4\rangle \\
|\psi_6\rangle &= \prod_{j=n}^{1} \mathrm{XOR}_{p_j^i m_j} \mathrm{NOT}_{m_j} |\psi_5\rangle \\
|\psi_7\rangle &= \prod_{j=n}^{1} \mathrm{2XOR}_{p_j^i u_2 m_j} |\psi_6\rangle
\end{aligned}$$

Method 2: Let us define the single qubit unitary gate

$$U_j^i = \cos\left(\frac{\pi}{2} p_j^i\right) 1 + \sin\left(\frac{\pi}{2} p_j^i\right) \sigma_y.$$

For a pattern p^i, we can encode the pattern into a single state by applying U_j^i;

$$|p^i\rangle = \prod_{j=1}^{n} U_j^i |0_1 \dots 0_n\rangle.$$

Then to store it in a superposition state given initial state $|\psi_0\rangle$, we compute the following;

$$\begin{aligned}
|\psi_0\rangle &= |0_1 \dots 0_n\rangle |00\rangle \\
|\psi_1\rangle &= \prod_{j=1}^{n} \left[\left(\mathrm{CP}_{u_2}^i\right)^{-1} \mathrm{NOT}_{u_1} \mathrm{CS}_{u_1 u_2}^{p+1-i} \mathrm{XOR}_{u_2 u_1} \mathrm{CP}_{u_2}^i \right] \\
&\qquad \times \mathrm{NOT}_{u_2} |\psi_0\rangle.
\end{aligned}$$

Repeating either process appropriately for different inputted binary patterns allows us to build an encoded data superposition state in a reliable manner.

It is observed that method 2 is more efficient than method 1 in terms of the number of qubits required. Method 2 also requires less operations than method 1, hence is less computationally expensive. Therefore, method 2 appears to be the best approach out of the two to use as an encoding strategy. For the sake of completeness and for comparison, both methods will be implemented.

Both of the aforementioned methods are realizable using the Intel Quantum Simulator since they use well defined quantum gate operations including the NOT, XOR, 2XOR (Toffoli) and CU (controlled unitary) gates. We have also developed some necessary extensions of the standard routines available in Intel's Quantum Simulator, including the n-qubit controlled unitary gate (nCU), which is required in the form of an n-qubit controlled not (nCX) gate for method 1. Hence, both methods prove to be reliable approaches to proceed with. To implement either

method, the following unitary matrix must be first constructed for $i = 1, \ldots, N$.

$$S^i = \begin{bmatrix} \sqrt{\frac{i-1}{i}} & \frac{1}{\sqrt{i}} \\ -\frac{1}{\sqrt{i}} & \sqrt{\frac{i-1}{i}} \end{bmatrix},$$

S^i is applied using the controlled unitary operator with S^i being the corresponding unitary matrix. The quantum operations required to implement both methods are displayed above. Each operation used is directly implementable using the Intel Quantum Simulator, apart from nXOR which required an extension of the available gate operations.

3.3 Hamming Distance for Sentence Similarity

Two approaches to calculate the Hamming distance are proposed. One, being Trugenberger's, the other a variation of Trugenberger's Hamming distance subroutine developed by Schuld et al. [8, 13]. For a known test word represented as a binary state, we can calculate the Hamming distance between this word's representation and every other basis word in the superposition. The approach pulls the Hamming distance into the eigenvalue of the eigenstate corresponding to each individual term in the superposition. Thus, the Hamming distance can be stored in the amplitude of its corresponding state. This enables us to quantify the similarity in meaning between the test word and the set of basis words, resulting in the test word's meaning.

The algorithm is detailed in Dirac notation as defined by Trugenberger [8] as follows. Given the superposition of encoded binary patterns $|m\rangle$ as defined in Eq. (1) which will be denoted as register m, we define two extra registers d of length n and c of length 1. Thus a total of $2n + 1$ qubits are required for this stage of the calculation. Note, that the ancilla register of the previous encoding strategy can be reused here, reducing the overhead of the number of qubits used. We define our initial state to be the tensor product of these three registers, where c is in the state $|0\rangle$ and the register d is the binary test pattern $x = x_1, \ldots, x_n$. Thus,

$$\begin{aligned} |\psi_0\rangle &= \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |x\rangle |p^i\rangle |0\rangle \\ &= \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |x_1 \ldots x_n\rangle |p_1^i \ldots p_n^i\rangle |0\rangle \end{aligned}$$

The next step is to apply a Hadamard operation to the ancillary qubit;

$$|\psi_1\rangle = \frac{1}{\sqrt{2N}} \sum_{i=1}^{N} |x_1 \ldots x_n\rangle |p_1^i \ldots p_n^i\rangle |0\rangle + \frac{1}{\sqrt{2N}} \sum_{i=1}^{N} |x_1 \ldots x_n\rangle |p_1^i \ldots p_n^i\rangle |1\rangle$$

The Hamming distance between x and p^i for $j = 1, \ldots, n$ is then calculated in binary format and stored in the d register overwriting the test pattern stored there. This is done by flipping all qubits in register d, then applying a controlled-NOT gate on each qubit in register d with the corresponding qubit in m acting as control;

$$\begin{aligned} |\psi_2\rangle &= \prod_{j=1}^{n} \mathrm{X}_{x_j} \mathrm{CNOT}_{x_j p_j^i} |\psi_1\rangle \\ &= \frac{1}{\sqrt{2N}} \sum_{i=1}^{N} |d_1^i \ldots d_n^i\rangle |p_1^i \ldots p_n^i\rangle |0\rangle + \frac{1}{\sqrt{2N}} \sum_{i=1}^{N} |d_1^i \ldots d_n^i\rangle |p_1^i \ldots p_n^i\rangle |1\rangle \end{aligned}$$

where d_j^i is the distance between the test pattern and the i^{th} training pattern for their j^{th} qubit. Due to a technique used later in the algorithm, it is desired that similar patterns will have a large Hamming distance, hence we say

$$d_j^i = \begin{cases} 1 & \text{if } |v_j^i\rangle = |p_j^i\rangle, \\ 0 & \text{otherwise} \end{cases}$$

which is why the NOT gate was applied to the m register.

The next step is to pull the Hamming distance into the amplitude of its corresponding state. This is done by applying the Hamiltonian

$$\mathcal{H} = \sum_{j=1}^{n} \left(\frac{\sigma_z + 1}{2} \right)_{d_j} \otimes 1 \otimes (\sigma_z),$$

through the unitary operator

$$U = e^{-\imath \frac{\pi}{2n} \mathcal{H}}.$$

Thus we obtain the state

$$\begin{aligned} |\psi_3\rangle &= e^{-\imath \frac{\pi}{2n} \mathcal{H}} |\psi_2\rangle \\ &= \frac{1}{\sqrt{2N}} \sum_{i=1}^{N} e^{\imath \frac{\pi}{2n} d_{\mathcal{H}}\left(\vec{x}, \vec{p^i}\right)} |d_1^i \dots d_n^i\rangle |p_1^i \dots p_n^i\rangle |0\rangle + \frac{1}{\sqrt{2N}} \sum_{i=1}^{N} e^{-\imath \frac{\pi}{2n} d_{\mathcal{H}}\left(\vec{x}, \vec{p^i}\right)} |d_1^i \dots d_n^i\rangle |p_1^i \dots p_n^i\rangle |1\rangle, \end{aligned}$$

where $d_{\mathcal{H}}\left(\vec{x}, \vec{p^i}\right)$ is the Hamming distance between $\vec{x}$ and $\vec{p^i}$ with the closer the value is to n, the more similar the patterns are to each other.

Finally, applying a Hadamard to the control register c, the new state reduces to

$$\begin{aligned} |\psi_4\rangle = &\frac{1}{\sqrt{2N}} \sum_{i=1}^{N} \cos\left[\frac{\pi}{n} d_{\mathcal{H}}\left(\vec{x}, \vec{p^i}\right) \right] |d_1^i \dots d_n^i\rangle |p_1^i \dots p_n^i\rangle |0\rangle \\ &+ \frac{1}{\sqrt{2N}} \sum_{i=1}^{N} \sin\left[\frac{\pi}{n} d_{\mathcal{H}}\left(\vec{x}, \vec{p^i}\right) \right] |d_1^i \dots d_n^i\rangle |p_1^i \dots p_n^i\rangle |1\rangle. \end{aligned}$$

Due to the nature of measurement of a quantum system it is not possible to obtain the amplitude of each state directly. Instead, we can obtain the probability of each quantum state (training pattern) occurring $\mathcal{P}\left(p^i\right)$. This probability is influenced by the Hamming distance in such a way that higher probabilities correspond to a higher Hamming distance, indicating that the test pattern is close to that particular training pattern.

To conduct this measurement, we must first collapse the ancillary cubit in register c. If it collapses to $|0\rangle$ with a high probability, then the test pattern is very different from all of the training patterns. Otherwise, if it collapses to $|0\rangle$ with a high probability, then the test pattern is close to the training patterns. More formally,

$$\begin{aligned} \mathcal{P}\left(|c\rangle = |0\rangle\right) &= \frac{1}{N} \sum_{i=1}^{N} \cos^2\left[\frac{\pi}{2n} d_{\mathcal{H}}\left(\vec{x}, \vec{p^i}\right) \right], \\ \mathcal{P}\left(|c\rangle = |1\rangle\right) &= \frac{1}{N} \sum_{i=1}^{N} \sin^2\left[\frac{\pi}{2n} d_{\mathcal{H}}\left(\vec{x}, \vec{p^i}\right) \right]. \end{aligned}$$

The probability of each individual pattern occurring can be obtained by repeating the experiment for a large number of times, measuring the register m corresponding to the training pattern, and recording the observed state of this register. Thus a distribution of probabilities of each state occurring can be obtained. Furthermore, we know that

$$\mathcal{P}\left(|m\rangle = |p^i\rangle\right) = \frac{1}{N\mathcal{P}\left(|c\rangle = |0\rangle\right)} \cos^2\left[\frac{\pi}{2n} d_{\mathcal{H}}\left(\vec{x}, \vec{p^i}\right)\right],$$

and so the Hamming distance can be obtained. Note that the slightly different method used by Schuld et al. [13] associates each training pattern with a class which is also encoded into the superposition. Measurement is then conducted on the class qubits rather than the training pattern qubits. These methods will also be considered as trivial extensions to Trugenberger's method.

It is also worth noting that Trugenberger proposed a version of the Hamming distance memory retrieval algorithm which uses only a unitary operator to fuse the standard quantum gate operations in a similar way as discussed in method 1 of Section 3.2. This method, or a slight modification of it, will also be examined to determine on which performs best.

Hamming distance similarity example: The method introduced here demonstrates the use of the Hamming distance as discussed earlier in Section 3.3, to outline a sample closest vector (state) problem. We define a set of orthogonal words (i.e. words with no overlapping meaning) through which we can define the meaning of other words. For this test case, we choose a simple vocabulary. Following the work of Coecke et al., we opt for the simplified NOUN-VERB-NOUN sentence structure, and define sets of words within each of these spaces, through which we can construct our full meaning space. For nouns, we have two sets, $n_s = \{\text{ADULT}, \text{CHILD}, \text{SMITH}, \text{SURGEON}\}$ and $n_o = \{\text{OUTSIDE}, \text{INSIDE}\}$, and for verbs, $v = \{\text{STAND}, \text{SIT}, \text{MOVE}, \text{SLEEP}\}$. With this NOUN-VERB-NOUN meaning space sentence structure, we can calculate this as

$$\begin{pmatrix} \text{ADULT} \\ \text{CHILD} \\ \text{SMITH} \\ \text{SURGEON} \end{pmatrix} \otimes \begin{pmatrix} \text{STAND} \\ \text{SIT} \\ \text{MOVE} \\ \text{SLEEP} \end{pmatrix} \otimes \begin{pmatrix} \text{OUTSIDE} \\ \text{INSIDE} \end{pmatrix}$$

For the above meaning-space, we can begin to define entities that exist in this space, and choose to encode them into quantum states as follows:

- JOHN IS AN ADULT, AND A SMITH. The state is defined as $|\text{JOHN}\rangle = 1/\sqrt{2}\left(|\text{ADULT}\rangle + |\text{SMITH}\rangle\right)$, which is a superposition of the number of matched entities from the basis set.
- MARY IS A CHILD, AND A SURGEON. The state is defined as $|\text{MARY}\rangle = 1/\sqrt{2}\left(|\text{CHILD}\rangle + |\text{SURGEON}\rangle\right)$, again, following the same procedure as above.

With these definitions, we can look at some sentences with JOHN and MARY. An example corpus with meaning that can exist in our meaning-space is: JOHN RESTS OUTSIDE, AND MARY WALKS INSIDE.

In this instance, we also require meanings for RESTS and WALKS. If we examine synonyms for RESTS and cross-compare with our chosen vocabulary, we can find SIT and SLEEP. Similarly,

for WALKS we can have STAND and MOVE. We can define the states of these words as $|\text{REST}\rangle = 1/\sqrt{2}\,(|\text{SIT}\rangle + |\text{SLEEP}\rangle)$ and $|\text{WALK}\rangle = 1/\sqrt{2}\,(|\text{STAND}\rangle + |\text{MOVE}\rangle)$. Now that we have a means to define the states in terms of our vocabulary, we can begin constructing quantum states to encode the data: firstly, however, we must decide on how to encode the states.

We can give each entity in the sets of nouns and verbs a unique binary index, where the word can be mapped to the binary string (gives the encoding value), and inversely mapped back (decoding value). Classically, we can use a key-value pair for this task, wherein the key (word) gives the value (index), and vice versa. As we intend to encode this data into quantum states, we opt for a sufficient number of qubits to define our states: namely, we require $\lceil \log_2(\text{num elements}) \rceil$ qubits to represent our data from each respective set.

For this example, we opt for the following mappings:

Dataset	**Word**	**Bin. Index**
n_s	ADULT	00
n_s	CHILD	01
n_s	SMITH	10
n_s	SURGEON	11
v	STAND	00
v	MOVE	01
v	SIT	10
v	SLEEP	11
n_o	INSIDE	0
n_o	OUTSIDE	1

It should be noted that here we have chosen a naïve mapping for our key-value pairs; the use of a Gray code [15], or alternative encoding strategies, may provide more instructive meanings to the meaning-space queries later. However, we leave this to future work for implementation and testing, and use the proposed encoding method for this example. To encode the meaning of the statement JOHN RESTS OUTSIDE, AND MARY WALKS INSIDE, we define the following quantum states:

Dataset	**Word**	**State**
n_s	JOHN	$(\|00\rangle + \|10\rangle)/\sqrt{2}$
n_s	MARY	$(\|01\rangle + \|11\rangle)/\sqrt{2}$
v	WALK	$(\|00\rangle + \|01\rangle)/\sqrt{2}$
v	REST	$(\|10\rangle + \|11\rangle)/\sqrt{2}$
n_o	INSIDE	$\|0\rangle$
n_o	OUTSIDE	$\|1\rangle$

We construct our quantum state by tensoring these entities, following the NOUN-VERB-NOUN DisCo-like formalism. If we consider the JOHN and MARY sentences separately for the moment, they are respectively given by the states $(1/2) * (|00\rangle + |10\rangle) \otimes (|10\rangle + |11\rangle) \otimes |0\rangle$ for John, and $(1/2) * (|01\rangle + |11\rangle) \otimes (|00\rangle + |01\rangle) \otimes |1\rangle$ for Mary. Tidying these up and multiplying out gives

$$\text{JOHN RESTS OUTSIDE} \quad \rightarrow \quad \tfrac{1}{2}(|00100\rangle + |00110\rangle + |10100\rangle + |10110\rangle) \rightarrow |J\rangle,$$

$$\text{MARY WALKS INSIDE} \quad \rightarrow \quad \tfrac{1}{2}(|01001\rangle + |01011\rangle + |11001\rangle + |11011\rangle) \rightarrow |M\rangle,$$

where the full meaning is given by $|S\rangle = \frac{|J\rangle + |M\rangle}{\sqrt{2}}$, which is a superposition of the 8 unique encodings defined by our meaning-space and sentence.

From here, we can now introduce an additional register of equal size to $|S\rangle$, wherein we will encode a test vector for comparison against the encoded meanings, denoted as $|T\rangle$. This test vector will be overwritten with the Hamming distance of it against the other states, with the resulting value giving the intended meaning, or closest possible meaning to that of the test. Continuing with our example, we use "Adults stand inside", which is encoded as $|T\rangle = |00000\rangle$. Constructing our query state, $|Q\rangle = |T\rangle \otimes |S\rangle$, we get the following fully expanded state

$$|Q\rangle = \frac{1}{2/\sqrt{2}}(|00000\rangle|00100\rangle + |00000\rangle|00110\rangle + |00000\rangle|10100\rangle + |00000\rangle|10110\rangle \\ +|00000\rangle|01001\rangle + |00000\rangle|01011\rangle + |00000\rangle|11001\rangle + |00000\rangle|11011\rangle).$$

Marking the Hamming distance between both registers, and overwriting the test register modifies the above state to

$$|Q'\rangle = \frac{1}{2/\sqrt{2}}(|00100\rangle|00100\rangle + |00110\rangle|00110\rangle + |10100\rangle|10100\rangle + |10110\rangle|10110\rangle \\ +|01001\rangle|01001\rangle + |01011\rangle|01011\rangle + |11001\rangle|11001\rangle + |11011\rangle|11011\rangle).$$

This example is simple as any state differing from $|00\cdots\rangle$ flips the test when a $|1\rangle$ is seen in the training. The data in register $|T\rangle$ now encodes the number of different flips between both test and training data; thus, it can be seen as a similarity measure, and the state with the fewest flips will give the closest meaning to the posed test. The above example has only a single data set with one flip, that of $|00100\rangle$. By decoding the result using the mapping provided earlier, we can see that this corresponds with a meaning of ADULT(S) SIT INSIDE. We can subsequently see that this data will give a non-zero overlap with $|J\rangle$.

The above approach can be mapped directly onto a qubit register, with indices used for partitioning into sub-registers. As such, the above methods can be directly implemented on qHiPSTER.

4 Representative Corpora

For the purposes of our implementation, we have opted for some simple corpora examples. Given the ambiguity and complexity of many natural languages, we have decided to maintain a simple data set for use during our development stages.

To follow the DisCo model as proposed, we opt for the commonly discussed sentence structure of NOUN-VERB-NOUN [1, 2] for the majority of our development and testing. We choose a small sample of public domain and custom phrases to analyse for this type of sentence structure in increasing levels of difficulty:

- User-defined simple NOUN-VERB-NOUN statements (e.g. "Mary likes fish, John likes cake"). Follows the example presented in Section 3.3.
- Jack and Jill (nursery rhyme). Largely follows simple structure, though more complex structure than previous example.
- Peter Pan (Page 1, paragraph 1). Most complex example. Large departure from NOUN-VERB-NOUN style.

For the early stages of implementation and development the above listed text corpora should suffice to allow us methods to evaluate the QNLP functionality and performance. While we expect longer corpora to be used later in the project, we currently think it best to choose those at a later date based on some realistic test cases involving the implemented methods. While a general purpose solution would be ideal, it may be required to adapt the longer corpora based on criteria we later determine from the implementations themselves.

5 Testing and Evaluation Methodologies

5.1 Proposed Methods for Testing

5.1.1 Unit

For each individual software component (class, method, etc.) developed in the project, an associated unit-test will be deployed to verify its correctness. The Catch2 testing framework [16] appears to be an ideal means to allow this, with header-only requirements on the C++ layer, and is available under the Boost Software License.

5.1.2 Integration

Like unit testing, we can again rely on the Catch2 framework to implement integration tests using a well-defined set of examples for each component of the Intel-QNLP suite. The use of continuous integration (CI) framework can be used to routinely examine these tests.

5.1.3 Regression

As with Section 5.1.2, we can use CI frameworks to perform periodic tests of those outlined for the previous levels of granularity in Unit and Integration tests, but also consider additional metrics such as:

- Vectorisation and threading
- Strong and weak scaling
- Memory occupancy
- Compute and memory intensity using roofline models
- MPI load-balancing
- Additional compilation options (inter-procedural/link-time optimisation, compiler optimisation levels, fast-math, etc.)

5.2 Proposed Methods for Evaluation

5.2.1 Acceptance

This final level of testing and evaluation will be used to determine overall correctness of the routines with a set of well-known results for the suite. We can use a manually-defined model, and compare the results to the computed data in an automated manner, making use of the CI testing methods as proposed by the Section 5.1.2. Additionally, we may use classical NLP methods to compare results with the implemented DisCo model, such as discussed below.

spaCy: spaCy [17] provides an industry-backed framework for natural language processing, which includes semantic comparison, and diagrammatic dependency visualisation. For the diagrammatic structure proposed by Zeng et al. [1, Example 5.3], the authors derive a graph for computing the meaning of a sentence using the proposed DisCo framework. While subtle differences exist, a similar graphical style can be obtained through use of the spaCy *displaCy* module. Figure 3 shows a comparison between both formats, wherein (a) is derived manually using the model proposed by Zeng et al., and (b) can be automatically derived using spaCy, both defined for the sentence "*John saw Mary read a book.*". This provides us with a promising candidate for comparative analysis of structure for our proposed model and implementation. Additionally, spaCy provides semantic similarity comparison based on use of word-vectors, provides us with a direct means to evaluate sentence similarity using classical NLP methods.

SEMILAR: SEMILAR [18] is a Java tool-kit for performing semantic sentence similarity analysis on text corpora. We have not currently investigated its use for this task, but leave it as a potential means to generate comparative data as a future task.

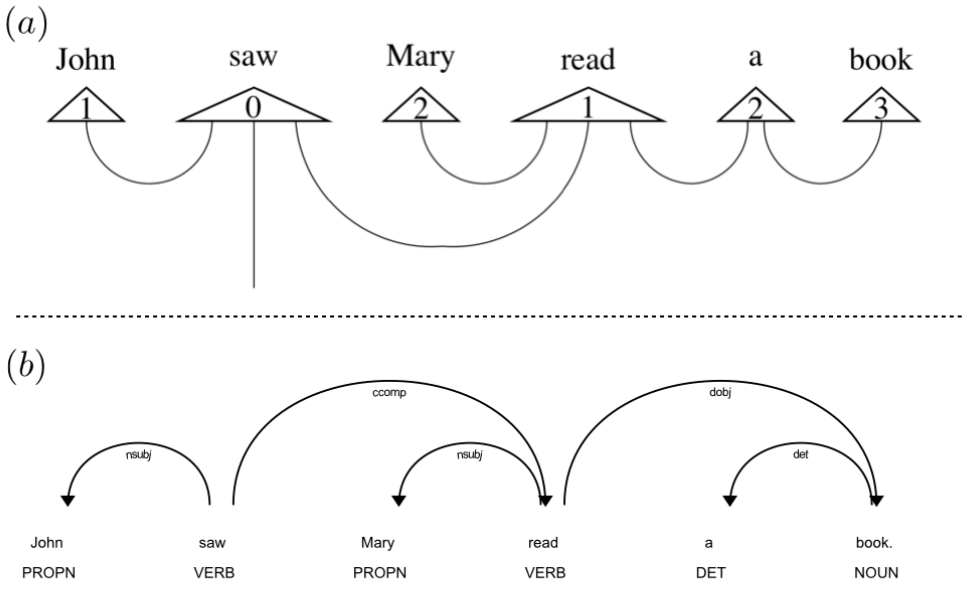


Figure 3: Sentence parsing structure for (a) the DisCo model, and (b) that from the classical NLP package spaCy.

6 Intel Quantum Simulator on Kay

6.1 Installation Setup

The Irish national supercomputer (Kay) was installed in August 2018, is operated by ICHEC and is comprised of

- a 336 node cluster.
- 13,440 CPU cores, 63 TB distributed memory.
- Dual-socket 20-core Intel Xeon Gold (Skylake) 6148 at 2.4 GHz with 192 GB memory.
- 400 GB local SSD scratch.
- 100 GB Intel OmniPath network.
- Additional partitions
  - Intel Xeon Phi (Knights Landing architecture).
  - High-memory 1.5 TB RAM with 1TB local SSD scratch.
  - Dual NVIDIA Tesla V100.

The *master* branch of the Intel Quantum Simulator from the intel/Intel-QS GitHub repository was installed on Kay in the path `/ichec/work/ichec001/`. The simulator was built with Intel Parallel Studio XE 2018 update 4 and vectorisation was enabled using AVX512 instead of the default SSE2.

6.2 Performance and Scaling

Testing was conducted using the *tests/qft_test.cpp* application supplied with the simulator which conducted a quantum Fourier transform using a specified number of qubits. Experiments were conducted for this application to demonstrate its weak and strong scaling performance. Due to the large memory footprint of the number of states, which doubles as the number of qubits increases by one, and the overhead of the corresponding access of that memory, the application was expected to be memory bound, and thus have relatively good weak scaling performance. As the number of quantum gate operations is increased in a quantum circuit the number of classical gates required to conduct the simulation can increase very quickly since these quantum gates are applied to an increasing number of qubits. Therefore, as the number of qubits are increased, the number of classical operations would increase significantly. Thus applications on the Intel Quantum Simulator are also subject to being compute bound.

Table 2 details the configuration of each experiment for both strong and weak scaling using double and also single precision numbers to represent the states. Figure 4 illustrates the results of the strong scaling performance and Figure 5 that of the weak scaling experiments. Both Figure 4 and Figure 5 appear to scale approximately in a linear fashion. This implies that the application scaled relatively well for both weak and strong scaling. However, it would be desirable to conduct more experiments for larger problem sizes to further observe the scalability of the application. This was not possible due to the previously mentioned limit of the MPI message size.

Scaling			Strong		Weak	
Nodes	p	NumProcs 2^p	Local States 2^{n-p}	Qubits n	Local States $2^{n-p} = 2^{27}$	Qubits n
1	1	2	2^{28-1}	28	2^{28-1}	28
2	2	4	2^{28-2}	28	2^{29-2}	29
4	3	8	2^{28-3}	28	2^{30-3}	30
8	4	16	2^{28-4}	28	2^{31-4}	31
16	5	32	2^{28-5}	28	2^{32-5}	32
32	6	64	2^{28-6}	28	2^{33-6}	33
64	7	128	2^{28-7}	28	2^{34-7}	34

Table 2: Details of number of total qubits, and local number of states per process for the corresponding experiment for both strong and weak scaling.

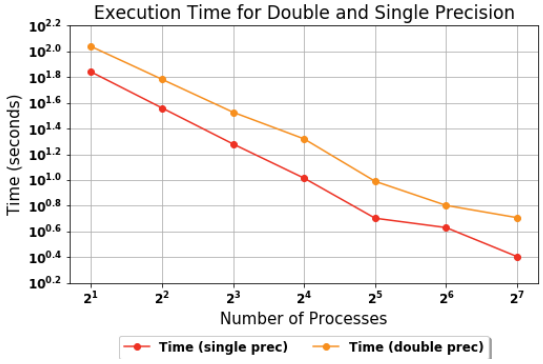


Figure 4: Strong scaling of execution time.

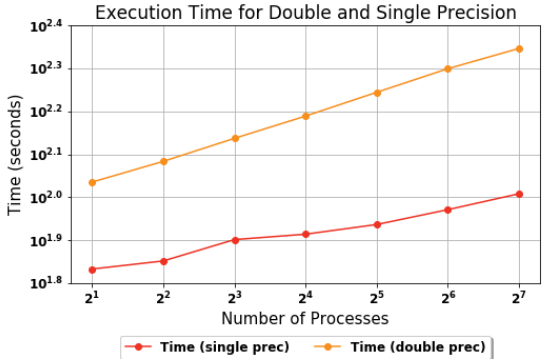


Figure 5: Weak scaling of execution time.

6.3 Scalability Limitations

The limit on the maximum MPI message size prevented larger problem sizes being tested for the Intel Quantum Simulator. This limited the number of double precision states per process to be approximately 2^{27}. Thus the simulator could not be tested up to the maximum available memory on a node.

To address this issue, the simulator was rebuilt using BigMPI instead of the standard MPI. However, the same problem persisted and BigMPI was later deprecated from being used with the simulator, reverting to standard MPI. As an alternative approach, Intel proposed running experiments on larger allocations on bigger European clusters, which would allow experiments using larger numbers of qubits, despite the memory on each node still being severely under-utilised. To pursue this line of action, ICHEC plans on applying for a PRACE project to get the access to a larger HPC facility.

It is also noted that Intel advised that experiments with greater than two processes per node do not perform as well as for one or two processes. Thus, using many processes to fill a node's memory without exceeding the maximum MPI message size was deemed not to be a valid workaround.

7 Discussions and Summary

Beginning with an outline of our proposed algorithmic approach in Section 2, we discussed the steps required to realise the DisCo models for closest vector and sentence similarity from an implementation independent approach. The use of pre-computation allows us to effectively prepare our data into a format that may be represented in a quantum state. The pre-computed data may then be encoded into a quantum state using a viable method. Following this, we may directly use the closest vector, or sentence similarity algorithms as discussed by Zeng and Coecke [1].

We proceed to mention issues encountered with the state creation approach as proposed in the original work, where the use of QRAM models in quantum algorithm design presents challenges to small-scale count quantum computer systems and simulators with limited number of qubits. Workarounds for this were introduced, following the encoding approach of Trugenburger [8], with the use of Hamming distance methods being effective to determine quantum state distance measures.

An outline of the software architecture is proposed, wherein we make use of a two-tiered approach to perform pre-computation (implemented using Python) and subsequent encoding and evaluation (implemented using C++). Intel qHiPSTER is used within this latter layer to realise the quantum computing aspects of the described algorithms.

Our proposed testing and evaluation methodology is presented, and forms an outline for our design decisions. We aim for a granular procedure, wherein testing and evaluation follows a *unit-integration-regression-acceptance* layout.

The system requirements and software dependencies for the Intel Quantum Simulator are detailed. The simulator was observed to scale well for both weak and strong scaling. However, the limitations of the scaling for larger problem sizes are noted. A limited proposed workaround and the expected drawbacks are described; using a large node allocation on a bigger HPC cluster.

The next steps of this project involve implementing the algorithms and methods described in this report. After these algorithms have been implemented and their results are validated, we must integrate the different algorithms into a single software solution – firstly, combining components of the Python and C++ implementations into modules, and following this integrating the Python and C++ modules to form one seamless application as illustrated in Figure 2. The results must be validated for the different corpora, which may require further refinement of the implementation. These steps will form the majority of the remaining work.

Furthermore, if time permits, different approaches to some of the above algorithms will be explored in greater detail. These algorithms involve an alternative to the DisCo model outlined in Appendix A, a Durr-Hoyer optimization of the Hamming distance approach in Appendix B, and finally a state-rotation method for use with measurement-based state determination in Appendix C.

Appendices

The following appendices represent some alternative approaches that can be taken to realise the closest vector and sentence similarity methods using quantum-enabled systems. They are listed here as stretch-goals or workarounds to consider following the implementation of the methods discussed in the bulk of this document.

Appendix A Alternative approach to DisCo

While the DisCo formalism is central to our body of work, we may consider alternative methods for data relationship encodings; instead of defining word meanings from their proximity to nearby words, we use can opt for a look-up model. This removes the distributional aspect of DisCo, but still enables use of the compositional approach via tensoring the meanings. As such, we propose calling this the "LoCo" (**Lo**ok-up **Co**mpositional) model. We make use of a basis set of terms, wherein the words follow an "orthogonal"-like set of meanings, with each word having little to no associated meaning with another in the language. For this, we may consider the approach of Ogden [19]. The author defines approximately 850 words which may be used to construct all other words in a language. We can treat these as the basis set of words, and define a mapping of our corpus to the basis by examining a synonym-based look-up (e.g. using NTLK [10], or spaCy [17]).

- For each word in Ogden's data set, tag and type each word.
- Give each type a set of unique bit-patterns to represent meaning in this space.
- Examine, tokenise and tag corpus data.
- Check basis set for appearance of word: if word exists, return given basis bit-pattern; if not, recursively examine synonyms and related words for matching terms in basis.
- The returned bit-patterns for the matched words enable the creation of an equal superposition to represent the corpus word meaning-space.
- If no matching term found, create new unique bit-pattern.

The use of this approach would be easily implementable in the pre-processing stage as described within Section 3.1. We describe the method here as a comparison may be beneficial for real-world data at a later stage in the project.

Appendix B Hamming distance and Durr-Hoyer optimization

The DH algorithm defines a method to determine a minimum value in a quantum state, wherein a threshold value is used to define relative from the calculated Hamming distance values. This method makes use of individual pieces proposed by Trugenberger [8, 9] and Wiebe et al. [7], and is modified as follows:

- Begin by flipping all possible Hamming distance qubit values, turning the smallest into the largest values.
- Define a threshold value for the DH algorithm randomly in the range of min to max of what the register may represent (the middle of the range is acceptable).
- Using quantum register arithmetic, subtract the flipped Hamming value from the threshold. Should the value underflow, the outermost qubit will be flipped from $|0\rangle \rightarrow |1\rangle$.

- Applying a Pauli-Z gate on the outermost qubit will perform a shift of the register from $|1\rangle \rightarrow -|1\rangle$.
- Using the diffusion operator from Grover's search algorithm, all marked states increase in amplitude relative to unmarked states.
- A new threshold can be set and iterated upon to the required convergence condition has been reached.
- A measurement of the state will now return the closest matching state with high probability, Multiple shot-based measurements will allow a statistical collection of the hierarchical state probability matches.

Given the implementation of methods as discussed in Section 3.1, this would be a viable extension to the model, and may potentially offer an advantage in terms of implementability.

Appendix C Controlled R_y rotation for measurement-based state determination

In addition to the use of a Hamming distance approach, as proposed in Section 3.3, we may use the Hamming distance to control the angle of rotation from $|0\rangle \rightarrow |1\rangle$ for an additional ancillary qubit, $|a\rangle$. Assuming we have use of a controlled rotation gate around the y-axis of the Bloch sphere, hence labelled $\mathrm{CR}_y(\theta, c, t)$, where θ is the angle of rotation, c is the control qubit, and t is the target, we may operate as follows:

- Encode the Hamming distance as described earlier, into a register $|H\rangle = |0\rangle \otimes \ldots |i-1\rangle$ of size i qubits.
- Apply Pauli-x to all qubits in $|H\rangle$ to flip the resulting Hamming distance.
- Iterate over each qubit in $|H\rangle$, applying $\mathrm{CR}_y(\pi/i, |i\rangle, |a\rangle)$.
- Perform projective measurement on the the ancillary qubit with $|1\rangle\langle 1|$.

Following these steps the system, after a measurement, will most likely be that with the lowest Hamming distance, and hence be the closest to the given test data. By repeating this with multiple shots, a statistical distribution can be obtained over the closest matching states. While this approach has the downside of multiple shots being required to obtain the result, it offers an experimentally realizable approach, assuming a low-overhead up-front state creation.

References

[1] William Zeng and Bob Coecke. Quantum algorithms for compositional natural language processing. *Proceedings of SLPCS*, 221:67–75, 2016.

[2] Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical foundations for a compositional distributional model of meaning. *arXiv:1003.4394 [cs, math]*, Mar 2010. arXiv: 1003.4394.

[3] Bob Coecke, Eric Oliver Paquette, and Dusko Pavlovic. Classical and quantum structuralism. *arXiv:0904.1997 [quant-ph]*, Apr 2009. arXiv: 0904.1997.

[4] Jacob D. Biamonte, Stephen R. Clark, and Dieter Jaksch. Categorical tensor network states. *AIP Advances*, 1(4):042172, Dec 2011. arXiv: 1012.0531.

[5] Maria Schuld. *Quantum machine learning for supervised pattern recognition.* PhD thesis, 2017.

[6] Kosuke Mitarai, Masahiro Kitagawa, and Keisuke Fujii. Quantum analog-digital conversion. *Physical Review A*, 99(1), Jan 2019. arXiv: 1805.11250.

[7] Nathan Wiebe, Ashish Kapoor, and Krysta Svore. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning. *arXiv:1401.2142 [quant-ph]*, Jan 2014. arXiv: 1401.2142.

[8] C. A. Trugenberger. Probabilistic quantum memories. *Physical Review Letters*, 87(6):067901, Jul 2001.

[9] Carlo A. Trugenberger. Quantum pattern recognition. *Quantum Information Processing*, 1(6):471–493, Dec 2002.

[10] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python.* O'Reilly Media, 2009.

[11] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Physical Review Letters*, 100(16), Apr 2008. arXiv: 0708.1879.

[12] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. On the robustness of bucket brigade quantum ram. *New Journal of Physics*, 17(12):123010, Dec 2015. arXiv: 1502.03450.

[13] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. Quantum computing for pattern classification. In Duc-Nghia Pham and Seong-BaeEditors Park, editors, *PRICAI 2014: Trends in Artificial Intelligence*, Lecture Notes in Computer Science, page 208–220. Springer International Publishing, 2014.

[14] Christoph Durr and Peter Hoyer. A quantum algorithm for finding the minimum. *arXiv:quant-ph/9607014*, Jul 1996. arXiv: quant-ph/9607014.

[15] Paul E. Black. Gray code. `https://www.nist.gov/dads/HTML/graycode.html`.

[16] Phil Nash et al. Catch2. `https://github.com/catchorg/Catch2`.

[17] Matthew Honnibal and Ines Montani. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 2017.

[18] V. Rus, M. Lintean, R. Banjade, N. Niraula, and D. Stefanescu. SEMILAR: The semantic similarity toolkit. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (August 4-9, 2013, Sofia, Bulgaria)*, 2013.

[19] C. K. (Charles Kay) Ogden. *Basic English : a general introduction with rules and grammar*. Psyche miniatures : General series ; no. 29. Kegan Paul, London, 8th ed. edition, 1940.