Robustness of an AKLT State on a Honeycomb Lattice

Improvements to Simulation Algorithms

by

Philip Allen Mar

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE

in

The Faculty of Science

(Combined Honours in Physics and Mathematics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

May 2011

© Philip Allen Mar 2011
Abstract

Cluster states used in measurement based quantum computation do not exist as nondegenerate ground states of two-body Hamiltonians, making them difficult to realize physically [8]. Using Affleck-Kennedy-Lieb-Tasaki (AKLT) states on a honeycomb lattice, a universal resource can be obtained for performing MBQC, by transforming these states into cluster states via two transformations. The first transformation turns the AKLT states into a graph state by using a three-element positive operator value measures and analyzing the effects of these operators in yielded encoded qubits on groups of vertices (domains). The graph state is then transformed to a cluster state as described in [8]. The transformation to a cluster state requires that the graph state contain a connected path from the left side of the original lattice to the right side. Since POVMs are measurements, they may sometimes, in a physical implementation of MBQC, contain errors. Instead of a three-element POVM, we might have a nine-element POVM, with various effects on the groups of vertices they are applied to. This can cause entire groups of vertices to be rendered useless for quantum computation, in particular, they may prevent the existence of a connected path, so that the graph state cannot be transformed to a cluster state. We must make sure the AKLT state is robust, that is, even with certain errors in the POVM, it still can be transformed to a cluster state. We ran simulations to confirm that for certain errors in the POVM (dependent on a parameter $p_{\text{delete}}$), there always exists a connected path. The main result in this study was the formulation of an algorithm that helped drastically reduce the time to run these simulations. Since the AKLT states are large entangled systems, it is time consuming to produce ‘typical lattices, but the introduction of this algorithm should help in running further simulations for other POVM outcomes in the future.
# Table of Contents

Abstract .......................................................... ii

Table of Contents .................................................. iii

List of Tables ..................................................... vi

List of Figures ...................................................... vii

Acknowledgements ................................................... viii

Dedication ........................................................... ix

1 Introduction and Motivation .................................. 1
  1.1 Quantum Computation ........................................ 1
  1.2 Measurement Based Quantum Computation (MBQC) .......... 1
  1.3 Affleck Kennedy Lieb Tasaki (AKLT) states ................. 2
  1.4 Robustness of the AKLT state ................................ 2

2 Theory .............................................................. 4
  2.1 Circuit Model ................................................ 4
     2.1.1 Universality ............................................. 5
  2.2 Cluster State Quantum Computation .......................... 5
  2.3 Affleck-Kennedy-Lieb-Tasaki (AKLT) States ............... 7
     2.3.1 One Dimensional AKLT State ............................ 7
     2.3.2 Two Dimensional AKLT State ............................ 8
  2.4 Objectives ................................................... 10
     2.4.1 Positive Operator Value Measure (POVM) ............ 10
# Table of Contents

2.4.2 Transformation Rules ........................................ 11
2.4.3 Conditions for Universality ................................. 13
2.4.4 Calculation ..................................................... 13

3 Progress .......................................................... 17
  3.1 Algorithm Overview/Programs ................................. 17
  3.2 Previous Work .................................................. 23
  3.3 Testing Algorithms ............................................. 25
  3.4 Probability Weighting .......................................... 28
  3.5 Brute-Force Algorithm Summary ............................. 29
    3.5.1 Single Lattice ............................................ 30
  3.6 Algorithm Improvement ....................................... 31
    3.6.1 Domain Grouping .......................................... 31
  3.7 Time Improvement ............................................. 34
    3.7.1 Single Lattice Test ....................................... 34
    3.7.2 Full Simulation Test ..................................... 35
  3.8 Accuracy ....................................................... 36
    3.8.1 Average Size of Domains & Average Domain Degree . 38
    3.8.2 20×20 Lattices with Connected Paths at $p_{\text{delete}} = 0.35$ ................................................. 39

4 Conclusion ......................................................... 40
  4.1 Results Summary ............................................... 40
  4.2 Future Work .................................................... 41

Bibliography .......................................................... 43
List of Tables

3.1 Simulations Times for Different-Sized Lattices .......................... 25
3.2 Probability Weighting: Single Simulations Times for Different-
Sized Lattices ........................................................................ 35
3.3 Lattice Characteristics: Average Domain Size and Degree Using
Brute Force Algorithm .......................................................... 38
3.4 Lattice Characteristics: Average Domain Size and Degree Using
New Algorithm ...................................................................... 39
# List of Figures

2.1 Cluster State Diagram .............................................. 6  
2.2 One Dimensional AKLT State ................................. 7  
2.3 Two Dimensional AKLT State ................................. 9  
2.4 Step by step transformation ................................. 11  
2.5 Transformations ............................................... 11  
2.6 POVM Map .................................................. 12  
2.7 Transformation Rule 1 ....................................... 13  
2.8 Transformation Rule 2 ....................................... 14  
2.9 Conditions for Universality .............................. 15  
3.1 Generating a Hexagonal Lattice ......................... 18  
3.2 Transformation Rule 1/Union Find ....................... 19  
3.3 Transformation Rule 2/Graph Generation .............. 20  
3.4 Domain Deletion .......................................... 21  
3.5 Breadth First Search .................................. 22  
3.6 Probability Weighting .................................. 23  
3.7 Previous Work: Domain Deletion ....................... 24  
3.8 Previous Work: Edge Deletion ......................... 24  
3.9 No Probability Weighting: 40 by 40 lattice .......... 26  
3.10 No Probability Weighting: 60 by 60 lattice .......... 26  
3.11 No Probability Weighting: 80 by 80 lattice .......... 27  
3.12 No Probability Weighting: 100 by 100 lattice ....... 27  
3.13 No Probability Weighting: 40 by 40 lattice with Threshold .. 28  
3.14 Probability Weighting: 12 by 12 lattice .............. 29  
3.15 Time for Simulation versus Lattice Height ........ 30  
3.16 Time for Simulation versus Lattice Size ............ 31
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.17</td>
<td>Algorithm Improvement</td>
<td>32</td>
</tr>
<tr>
<td>3.18</td>
<td>12×12 Lattice with Probability Weighting</td>
<td>36</td>
</tr>
<tr>
<td>3.19</td>
<td>40×40 Lattice with Probability Weighting</td>
<td>37</td>
</tr>
<tr>
<td>3.20</td>
<td>60×60 Lattice with Probability Weighting</td>
<td>37</td>
</tr>
</tbody>
</table>
Acknowledgements

Thank you to Robert Raussendorf who was my supervisor for the 449 Thesis Project and to Tzu-Chieh Wei whose work was the basis for most of the project. Thank you to Robert Kiefl for his supervision of the 449 thesis class.
Dedication

Dedicated to my parents, who have loved me and raised me in patience, my friend from high school who shared a passion for physics (Eric), my friends in university who are going on to other things (Zion, Yakov), my friends at church, my physics and math teachers (in particular, Jon and John), Auntie Maribel, and to the real Jesus, to whom I wish I could dedicate this with all my heart, because He has given me everything.
Chapter 1

Introduction and Motivation

1.1 Quantum Computation

Quantum computation is interested in exploiting quantum effects in order to perform certain computational tasks using specialized algorithms that do take these effects into account. The most immediate application is in performing tasks that would otherwise take classical computation much longer, or in running simulations that are inherently quantum mechanical in nature [6][7]. It should be noted that the quantum computing resources needed for implementing Shor’s algorithm have not yet been achieved. Previous models of quantum computation were experimentally designed by controlling interactions acting as gates between various spin-1/2 particles acting as ‘bits’ or ‘quantum-bits’ (qubits).

1.2 Measurement Based Quantum Computation (MBQC)

In 2000, Robert Raussendorf and Hans J. Briegel proposed a new model of quantum computation that contrasts with these types of experiments: measurement based quantum computation, called a ‘one-way quantum computer’. Rather than implementing interactions between spin-1/2 particles to perform computation; the computation itself is performed by making local measurements on a prepared highly-entangled state of spin-1/2 systems, called cluster states. It can be shown that cluster states are universal for quantum computation; that is, from a finite set of operations on this cluster state we can approximate any unitary operation on the qubits (i.e. we can...
1.3 Affleck Kennedy Lieb Tasaki (AKLT) states

One solution that was recently found was to use what are called AKLT states (for Affleck-Kennedy-Lieb-Tasaki states) on a honeycomb lattice arrangement. AKLT states exist as ground states of Hamiltonians written as summations of adjacent two-body interactions, so that they can be realized physically [7]. It was shown that an AKLT state can be transformed to a cluster state via two transformations (first to a graph state, and then second to a cluster state in a square lattice), and thus can be used as a universal resource for quantum computation.

1.4 Robustness of the AKLT state

The two aforementioned transformations require actual physical operations on the AKLT states, and thus are, in real life, subject to errors. The main objective is to investigate how tolerant the AKLT state is to errors, that is, given certain errors in the transformation of the AKLT state, how certain is the AKLT state going to be still universal for quantum computation? Specifically, if there are errors in the first transformation, does the resulting graph state still satisfy the conditions for the further transformation to the universal cluster state?

The investigation proceeded primarily through simulations: increasing the probability of an error in the first transformation and checking whether various lattices satisfied conditions to transform to a universal cluster state. A confirmation of such robustness of the AKLT states is an encouraging, albeit very small, step towards the realization of measurement based quantum
1.4. Robustness of the AKLT state

computing\textsuperscript{1}.

\textsuperscript{1}This just ensures universality, much more needs to be done in making sure the resulting computation is feasible and error-free as well.
Chapter 2

Theory

2.1 Circuit Model

The most common model for quantum computation is the circuit model, essential for understanding the basics of quantum computation. The fundamental object in quantum computation is the qubit, which can be described as a state vector in a two-dimensional Hilbert Space \(^2\), called the state space. Qubits can be expressed as linear combinations of computational basis states; so if \(\psi\rangle\) is the bra-ket describing the qubit, and we denote as \(|0\rangle\) and \(|1\rangle\), then the qubit can be expressed as \(\psi = \alpha|0\rangle + \beta|1\rangle\), with the normalization condition that \(|\alpha|^2 + |\beta|^2 = 1\). An essential aspect of quantum computation is the property of quantum systems to be entangled. To analyze these, we need a way to describe composite systems of several qubits. This is described by using tensor product notation. If each individual system is denoted by \(\psi_i\rangle\), then the composite system is the tensor product of these states: \(\otimes_{i=1}^n \psi_i\rangle\).

Quantum computation in the circuit model begins with the preparation of a simple state that initializes the computer, such as \(\otimes_{i=1}^n |0\rangle_i\). In order to perform a computation, operators on these states are defined, analogous to logic gates in classical computing. These operators in the circuit model are unitary\(^3\). Finally, we define a means of measurement on the qubits. Extending to operations on multiple qubits, there are controlled operations, where the unitary operation on one or more qubits (target qubits) is dependent on the state of other qubits (control qubits).

\(^2\)A vector space over some field together with an inner product that is complete \(^3\)Since unitary operators \((UU^\dagger = U^\dagger U = I)\) define deterministic operations on the qubits, contrasted to measurements which are probabilistic.
2.2. Cluster State Quantum Computation

2.1.1 Universality

The most important concept is that there exist finite set of gates that can be used to approximate any unitary operator to arbitrary accuracy \(^4\). The quantum circuit model represents qubits by quantum wires (moving left to right in time) that are processed by a set of unitary operators (quantum gates). From this model, we find that any quantum circuit can be replaced by one that contains only single-qubit gates and controlled-phase qubit gates. Controlled-phase qubit gates are two-qubit gates with the action of taking a composite state and switching a sign, for example,

\[ |x\rangle \otimes |y\rangle \rightarrow (-1)^{xy} |x\rangle \otimes |y\rangle \quad \text{for } x, y \in \{0, 1\} \tag{2.1} \]

Thus, the single qubit gates and controlled phase two qubit gates are universal for quantum computation \(^5\)\(^2\).

2.2 Cluster State Quantum Computation

The cluster state model uses measurements on a prepared system of many entangled qubits. Outputs are expressed in terms of the un-measured qubits of the system. An \(n\)-qubit cluster state can be represented by a graph \(G\) where, the vertices represent prepared qubits in the state \(|+\rangle := (|0\rangle + |1\rangle)/\sqrt{2}\) (where \(|0\rangle, |1\rangle\) is the computational basis state) and the edges represent controlled-phase two-qubit gates. Since controlled-phase gates commute, it does not matter the order in which they are applied to the qubits does not matter. When applying measurements to single qubits in the cluster state, we label them according to what sequence we apply them; sometimes having the same time-label indicating that the order in which they are applied does not matter. The time label also determines how qubits of a

\(^4\)In mathematical terms, this means that there are a finite number of operators in this model of computation, that, for any unitary operator, there exists a finite sequence of compositions of the operators in the universal set that approximate the unitary operator to arbitrary accuracy (strong convergence in norm)

\(^5\)Note that the ‘single qubit gates’ really mean a finite set of single qubit gates, these are the Hadamard and the \(\pi/8\) gates. More on this either on wikipedia.org on “Quantum gates”, or in Nielsen and Chuang’s textbook, [5]
later time label may depend on the measurement outcomes of qubits labelled with earlier time labels. Measurements must satisfy the conditions that they are single-qubit measurements; that they may depend on the measurement outcomes of previous measurements on other qubits and that measurement results from previous qubit measurements can be processed by a classical computer (to help determine what new measurements to be made) [4].

The cluster state model can be shown to be universal for quantum computation. The proof can be found in [4], and involves verifying a circuit identity, then showing that the output of a quantum circuit of specific types of single qubit operators ($HZ_\theta$) has the same output (up to a Pauli matrix) as a cluster-state computation (by showing that this cluster state computation is in turn equivalent to a quantum circuit consisting of a sequence of the circuit identity). The proof concludes by stating that $HZ_\theta$ and the controlled-phase gate are a universal set for quantum computation. A helpful diagram is given in Fig. 2.1.

![Figure 2.1](image)

Figure 2.1: Measurement Based Quantum Computation on a Cluster State Scheme. Labels 1, 2, denote the order in which they are measured, and lines between circles represent controlled-PHASE operations between qubits, entangling them.
2.3. Affleck-Kennedy-Lieb-Tasaki (AKLT) States

An Affleck-Kennedy-Lieb-Tasaki (AKLT) state is “the ground state of a quantum antiferromagnet” [1]. For illustration of what an AKLT state is, we first consider a one-dimensional AKLT state [9].

2.3.1 One Dimensional AKLT State

Rigorously, the AKLT state is the “ground state of the Hamiltonian”

\[
H = J \sum_{b=1}^{N-1} [S_b S_{b+1} + \frac{1}{3}(S_b S_{b+1})^2]
\]  (2.2)

\[
P_{r-j} = \begin{cases} 
0 & \text{if } r - j \text{ is odd,} \\
r!(-1)^{(r-j)/2} & \text{if } r - j \text{ is even.}
\end{cases} \]  (2.3)

Note that here the Hamiltonian depends on pairwise interactions between spin-1/2 systems, and thus are easier to analyze than five-body Hamiltonians [9].

Suppose we are given \(N\) spin-1/2 systems. To construct the AKLT state, first entangle these systems in pairs in singlet (\(|\phi\rangle = |01\rangle - |10\rangle\)) states. Then, align these singlet states in a line, so that the singlet states are aligned end-to-end. Project these ends of singlet states in a spin-1 states (spin-1 states are triplet states, with values \(|-1\rangle, |0\rangle, |1\rangle\)). Fig. 2.2 illustrates this.

![Spin-1/2 Spin-1](image)

Figure 2.2: An AKLT state in one dimension; as an entangled state of singlet states, each end of each singlet pair projected to a spin-1 system.
In summary, the AKLT state takes the form:

\[
|\text{AKLT}\rangle = \bigotimes_{a=1}^{N} P_{S,a}\left(|\psi_L\rangle|s = 0\rangle_{b,b+1}|\psi_R\rangle\right)
\] (2.4)

Where we have that \(|\psi_L\rangle\) and \(|\psi_R\rangle\) are the wavefunctions for the left and right spin-1/2 systems \([9]\).

### 2.3.2 Two Dimensional AKLT State

In this case, the AKLT state takes the form of a hexagonal lattice, which we denote by the symbol \(\mathcal{L}\), and where \(\mathcal{L} = (V(\mathcal{L}), E(\mathcal{L}))\), expressing \(\mathcal{L}\) as a pair of sets \(V(\mathcal{L})\) of vertices, and \(E(\mathcal{L})\) of edges. This type of construction can be extended to two dimensions. The Hamiltonian in this case is given by

\[
H = J \sum_{e \in E(\mathcal{L})} P_{3,e}
\] (2.5)

Where \(P_{3,e}\) is the projection operator on the spin-3 subspace on each edge. Now the construction of the AKLT state is as follows.

Given the spin-1/2 systems in the previous section, again pair them together. For each vertex \(v \in V(\mathcal{L})\), place the ends of three pairs. Thus, on each edge \(e \in E(\mathcal{L})\), connecting two vertices, there exist six spin-1/2 systems. The projection \(P_{3,e}\) projects this space unto a spin-3 subspace. In the end, the AKLT state takes the form:

\[
|\text{AKLT}\rangle := \bigotimes_{v \in V(\mathcal{L})} P_{S,v} \bigotimes_{e \in E(\mathcal{L})} |\phi\rangle_e
\] (2.6)

The AKLT state is then made by projecting each vertex of 3 spin-1/2 systems into a spin-3/2 subspace (4 states). This is represented by the projector \(P_{S,v}\), where \(S\) denotes the spin-3/2 subspace and \(v\) denotes the vertex in question. If \(|0\rangle\) and \(|1\rangle\) are the two computational basis states for
2.3. Affleck-Kennedy-Lieb-Tasaki (AKLT) States

a spin-1/2 system, then \( P_{S,v} \) takes the form

\[
P_{S,v} := |000\rangle\langle 000 | + |111\rangle\langle 111 | + |W\rangle\langle W | + |\bar{W}\rangle\langle \bar{W} | \tag{2.7}
\]

where \(|W\rangle = \frac{1}{\sqrt{3}}(|001\rangle + |010\rangle + |100\rangle)\), \(|\bar{W}\rangle = \frac{1}{\sqrt{3}}(|110\rangle + |101\rangle + |011\rangle)\).

The \(|\phi\rangle_{e} \) denotes the edges that represent the singlet states, pairing one spin-1/2 system in one vertex, to another spin1/2 system in another vertex. Fig. 2.3 illustrates this.

Figure 2.3: An AKLT state in two dimensions; as an entangled state of singlet states, each end of each singlet pair projected with two other pair-ends into a spin-3/2 system.
2.4 Objectives

Having introduced the circuit model, cluster-state model and AKLT states, we can present the objectives of the thesis project. Recall that the two-dimensional cluster state on a square lattice is universal for quantum computation, that is, we can approximate any arbitrary unitary operator with finite sequences of a finite set of operations in cluster-state computation. However, a physical realization of two-dimensional cluster states directly is difficult. Thus, we aim to use AKLT states which can be transformed into cluster states in order to obtain a substrate that is universal for quantum computation [7] [8].

The overarching objective is to then transform the AKLT state to a cluster state. The immediate objective is to ensure that this transformation is robust, that is, the transformation to a cluster state is still possible even when errors are made in the method of transformation. In summary, the way the AKLT state is transformed to a cluster state is first to transform the AKLT state to a graph state $|G\rangle$. These involve so-called Transformation Rules, which are discussed later. These need to be simulated in the computer program, and constitute the bulk of the algorithm and time in calculation. Then, the graph state $|G\rangle$ is further transformed to the cluster state via the transformation described in Wei, Raussendorf and Affleck’s paper that supplements the primary one in [8]. This transformation requires that the graph state satisfy two conditions, which we will call conditions for universality, required in the proof that the graph state can indeed be transformed to a cluster state. Satisfying these conditions is the practical ‘definition’ of the robustness of the AKLT state. If resulting graph state satisfies these conditions despite having errors in the transformation from an AKLT state, then it is robust. Fig. 2.5 illustrates this [7] [8].

2.4.1 Positive Operator Value Measure (POVM)

A Positive Operator Value Measure (POVM) is informally a mapping between sets of outcomes of a quantum state to the set of operators of the Hilbert state space of that quantum state. It gives a useful method to
2.4. Objectives

Figure 2.4: Step by step transformation

Figure 2.5: Transforming an AKLT state to a graph state to a cluster state.

project the state space into a subspace depending on some probabilistic outcome. A special case of a POVM are projective measurements. For each set of outcomes of the state, we obtain a projection operator \( P_i \) that we apply to the original state that projects it to a subspace of the original state space. In this case, we do not only have projections, but, more generally, positive operators \(^6\) [Conway/wikipedia] As such, POVMs are thought of as generalized measurements. Either the mapping itself or the range of the mapping in the set of operators can be thought of as the POVM; so the set \( \{F_i\}_{i \in I} \) for an index set \( I \) can be called the POVM \(^{11}\).

The POVM is a valuable tool in the transformation to a graph state. We apply the POVM to every vertex in the AKLT lattice in order to obtain one of three outcome operators: \( F_x \), \( F_y \) and \( F_z \), each projecting the vertex to an encoded spin-1/2 system. Fig. 2.6 illustrates the POVM mapping.

2.4.2 Transformation Rules

To transform the AKLT state to a graph state, we go through each vertex in the original lattice, then apply the POVM to each vertex. This means each vertex is projected unto a subspace of either \( x \), \( y \) or \( z \). We now introduce the two transformation rules to change this lattice with POVM outcomes into a graph state \( |G\rangle \) \(^7\) \(^8\).

\(^{6}\) that is, if \( A \) is a positive operator, then for all \( |h\rangle \in \mathcal{H} \), the state space, then \( \langle Ah|h\rangle \geq 0 \).
2.4. Objectives

Figure 2.6: POVM maps the set of outcomes to one of many positive operators on the Hilbert Space.

1. Group all vertices adjacent to one another with the same outcomes to be one vertex in the new graph $G$. These groups of vertices with the same outcomes are called *domains* to distinguish them from the vertices of the original lattice. Fig. 2.7 illustrates this.

2. For the resulting graph $G$, all edges of even multiplicity are deleted, and all edges of odd multiplicity are reduced to only one between the domains. Fig. 2.8 illustrates this.

Rules 1 and 2 are derived by the properties of the AKLT state, and are proved to be valid transformations in [7] [8]. In short, when the vertices have the same POVM outcome, due to the way the spin-1/2 systems are entangled, they act as a single encoded spin-1/2 system, and so each domain acts as a single spin-1/2 system. This also means that edges of even multiplicity are deleted because their effects essentially cancel each other out for the domains they connect [7] [8].
2.4. Objectives

Figure 2.7: Grouping Similar Outcomes. One color for each of $x, y$ and $z$.

2.4.3 Conditions for Universality

The proof for conversion of the graph state $|G\rangle$ to a cluster state requires that the graph $G$ satisfy two properties, which we call the Conditions for Universality. The proof is lengthy and given in detail in [8].

1. The size of the domains in the honeycomb lattice are microscopic for large lattices $\mathcal{L}$, that is, the size of the individual domains are independent of the size of the lattice, $|\mathcal{L}|$.

2. The probability of the existence of a connected path (also called “spanning cluster”) traversing $G$ from left to right approaches 1 in the limit of large lattice size $|\mathcal{L}|$. Fig. 2.9 illustrates the connected path from left to right.

2.4.4 Calculation

The primary error under investigation are the POVMs. The ideal case is that the POVMs always project the spin-$3/2$ system of vertices to a spin-$1/2$ system. However, this is akin to a projective measurement, and so is prone to errors because it is a physical operator. For example, the POVMs
2.4. Objectives

Figure 2.8: Deleting even multiplicity edges and retaining odd multiplicity edges.

might not project the system to a spin-1/2 system, but instead to one of the computational basis states; effectively destroying the ability for the vertex to be used in quantum computation, since a qubit must be a spin-1/2 system [7] [8].

An essential property of the POVM is that its elements satisfy the completeness relation, \( \sum_{i=x,y,z} (F_i)(F_i)\dagger = I \). Using this completeness relation, we can define the ideal POVM, a three-element POVM, to be applied to each vertex:

\[
F_{v,z} = \sqrt{\frac{2}{3}}(|000\rangle\langle 000| + |111\rangle\langle 111|) \tag{2.8}
\]

\[
F_{v,x} = \sqrt{\frac{2}{3}}(|++\rangle\langle ++| + |--\rangle\langle --|) \tag{2.9}
\]

\[
F_{v,y} = \sqrt{\frac{2}{3}}(|i,i,i\rangle\langle i,i,i| + |{-i,-i,-i}\rangle\langle {-i,-i,-i}|) \tag{2.10}
\]

Where \( |\pm\rangle := \frac{|0\rangle \pm |1\rangle}{\sqrt{2}} \), \( |\pm i\rangle := \frac{|0\rangle \pm i|1\rangle}{\sqrt{2}} \).

An erroneous POVM would introduce new POVM elements that include the projection operator to the encoded computational basis states (that is, the eigenstate that describes the encoded \( |\bar{0}\rangle \) and \( |\bar{1}\rangle \) basis states, encoded
over several qubits). For each of the elements $x, y$ and $z$, there are 3 POVM elements, instead of 1, for a total of 9 POVM elements:

$$\{\sqrt{1-p}F_{v,x}, \sqrt{\frac{2p}{3}}|++\rangle\langle++|, \sqrt{\frac{2p}{3}}|---\rangle\langle---|\}$$  \hspace{1cm} (2.11)

The value $p$ is an arbitrary parameter that is associated to a 'probability' that we make such an error in our calculation. The extra POVMs can be physically thought of as making an essentially perfect POVM operation followed by measuring a specific state destroying the spin-1/2 state. The more important effect is that the other vertices with the same POVM outcome (i.e. $x, y$ or $z$) that are adjacent to this vertex that was 'deleted', are also deleted, so the entire domain is deleted, not just a vertex, since the domain in total acts like one encoded spin-1/2 system.

While this type of vertex/domain-deleting POVM is the primary focus of this study, other types of erroneous POVMs exist, like measuring
2.4. Objectives

\((\langle 000 | + |111 \rangle)(\langle 000 | + |111 |)/\sqrt{2}\), which may cause other, more complex effects. Another effect perviously studied was edge-deletion [7] [8].
Chapter 3

Progress

In order to simulate erroneous POVMs, we focus on vertex-deletion, and change the parameter \( p \), increasing it to signify ‘increasing’ error in our POVM application. The objective is to determine the value of \( p \) at which the AKLT state fails to satisfy the conditions for universality, thus, \( p \) gives a quantitative measure for how robust the AKLT state is with respect to making this type of error. To test the robustness of the AKLT state, fix the value of \( p \), and generate many large lattices. For each of these large lattices, assign POVM values to each vertex, generate a graph state, and check that it satisfies both conditions for universality. This thesis focused on the existence of a spanning cluster. The process of assigning POVM outcomes to each vertex is the most time consuming part of the simulation; the subtitle for this thesis, “improvements to simulation algorithms” refers to the way the algorithm was improved for greater efficiency, allowing for accurate simulations to be run in a realistic amount of time.

3.1 Algorithm Overview/Programs

All these algorithms were implemented in MATLAB, and are presented here starting with the name of the program, followed by a short description listing its intended function (as an algorithm), and its inputs and outputs.

1. HexPOVMTwo.m  **Input:** \( N \), vertical size of lattice.  **Output:** \( A, N \times (N + 1) \times 2 \) array. A function whose input, a value \( N \), denotes the number of ‘rows’ of a hexagonal lattice. The way a square-lattice stores information for a hexagonal-lattice is by considering the hexagonal lattice in a brick-layout. A separate program assigns the connections.
between each vertex. The first dimension of the array \((N \times (N+1) \times 1)\) contains all the POVM outcomes, that are randomly assigned either 1, 2 or 3, denoting \(x, y\) and \(z\). These are assigned with a probability \(\frac{1}{3}\) to each, but since the vertices are entangled, we require that there be a probability weighting that shifts these outcomes later. The second dimension of the array \((N \times (N+1) \times 2)\) contains all the cluster-labels of the vertices, in this case, we number them from 1 to \(N \times (N + 1)\), since each vertex is, at first, in separate clusters. There is a separate algorithm that groups them together.

![Figure 3.1: Horizontal rectangles denote how vertices are arranged in rows, resp. vertical to columns.](image)

2. **HexAdjacentTwo.m**

**Input:** \(N\), the vertical size of the lattice, \(i, j\), the row,column coordinate of the vertex in question. A function whose input gives the vertex in question and outputs the vertices that are adjacent to the vertex in question in a hexagonal lattice. This is com-
3.1. Algorithm Overview/Programs

complicated because for the vertices on the outermost rows and columns must still have tri-valency (i.e. have three edges coming out of them), and so they are sometimes connected more than once to the same vertex to ensure this (this ensures it is an AKLT state). For example, vertex (1,1) has two edges to vertex (1,2).

3. **UnionFindThree.m** **Input**: A, the output of HexPOVMTwo.m. **Output**: A, the lattice with updated cluster-labels. The UnionFind algorithm here is also known as the Hoshen-Kopelman algorithm\(^7\). No computer science enhancements were made to this brute-force implementation of it. In fact, it mostly uses ideas from the UnionFind algorithm rather than being a formal UnionFind algorithm. It goes through each vertex, checks adjacent vertices for whether they share the same POVM outcome; for vertices that do share the same POVM outcome, the algorithm ensures they are labelled with the same cluster label, that is, changing the value on the second dimension of the array A. Fig. 3.2 illustrates.

![Figure 3.2](image)

Figure 3.2: Grouping Similar Outcomes. One color for each of \(x, y\) and \(z\).

4. **GraphGeneratorTwo.m** **Input**: A, the output of UnionFindThree.m.

\(^7\)The latter is a special case of the former.
3.1. Algorithm Overview/Programs

**Output:** $C$, an adjacency matrix of the graph state $|G\rangle$ associated with the lattice given by $A$. A function whose input is the array with updated cluster-labels. This algorithm generates the graph by assigning a vertex to each domain, and listing the edges between the domains, taking into account the deleting the even multiplicity edges and reducing odd multiplicity edges to multiplicity 1. Fig. 3.3 illustrates.

![Graph Illustration](image)

Figure 3.3: Deleting even multiplicity edges and retaining odd multiplicity edges.

5. **DomainDeleteTwo.m**

**Input:** $C$, adjacency matrix $p_{\text{delete}}$, probability of domain deletion

**Output:** $C$, updated adjacency matrix. A function that goes through each domain and, with probability $p_{\text{delete}}$, deletes the domains, destroying all the edges connected to it. This is associated with the erroneous POVM mentioned earlier. Fig. 3.4 illustrates.

6. **BreadthFirstSearch.m**

**Input:** $C$, adjacency matrix and $A$, array.

**Output:** 0 or 1, corresponding to 'no' and 'yes'. A function which checks domains that have vertices that are on the left side of the hexagonal lattice, and domains those that have vertices on the right side of the hexagonal lattice, and determines, by a breadth first search on the adjacency matrix $C$, whether there is a spanning cluster. The breadth
3.1. Algorithm Overview/Programs

Figure 3.4: Deleting domains with probability $p_{\text{delete}}$

first search works by checking the neighbors of all the domains with 'left-side' vertices, checking whether any of these neighbors are the domains with 'right-side' vertices, and if none of the neighbors are, checking the neighbors of the neighbors for domains with 'right-side' vertices, and so on. Fig. 3.5 illustrates this. Domain 1 has a left-side vertex, so check its neighbor, domain 5. But domain 5 does not have a right-side vertex, so check the neighbors of 5. Domain 6 does not have a right-side vertex, so check the neighbors of 6; and domain 4 has a right-side vertex. Note that this graph was generated using the 'part-lattice' in Fig. 2.7.

7. ProbabilityWeightingBrute.m This is the over-arching simulation program. It performs the previous algorithms for many different lattices of a fixed size. This way, we have a sampling of many lattices. For each set of about 50 lattices (each of size either $40 \times 40$ up to $100 \times 100$), the $p_{\text{delete}}$ is varied from 0.3 to 0.55 (since we suspect that the 'percolation' threshold is around 0.33). This only works since it was found that this was a percolation problem. This program also

---

8 Explained in more detail later, but in short, there exists a threshold of $p_{\text{delete}}$, for
Figure 3.5: Domain 1 has a left-side vertex, domain 4 has a right-side vertex. This graph has a spanning cluster.

implements the proper probability weighting for each of the vertices, by using the sub-program `BruteProbabilityShift.m`.

8. **BruteProbabilityShift.m**
   **Input:** A input from `HexPOVMTwo.m`
   **Output:** C, A, adjacency matrix and updated array. This program uses both `UnionFindThree.m` and `GraphGeneratorTwo.m`. It implements the probability weighting necessary to take into account the entanglement between the vertices. This is done first by choosing a random vertex. Determine the number of domains $V$ and the number of edges $E$ for this vertex, flip, with equal probability, into one of the other POVM outcomes. Obtain the number of new domains $V'$ and new edges $E'$ due to this change. Calculate the probability of accepting this change in POVM outcome based on the change of the number of domains and number of edges: $p_{\text{flip}} = \max\{1, 2|V'| - |E'| - |V| + |E|\}$. This means using `UnionFindThree.m` and `GraphGeneratorTwo.m` for each vertex. This program is the one that takes the longest and scales the worse, because which below there is guaranteed to have a spanning cluster, and above which there is no spanning cluster.
3.2 Previous Work

The previous work by Tzu-Chieh Wei and Robert Raussendorf [7] [8] showed that this simulation of increasing \( p_{\text{delete}} \) and checking for a connected path yielded a percolation problem. That is, below a threshold probability, a connected path always existed, but above the threshold probability, a connected path did not exist (for any of the randomly generated lattices). Fig. 3.7 illustrates the problem of domain deletion. Notice that for larger lattices, the threshold between a connected path existing and not becomes more defined. For domain deletion, the percolation threshold was found to be about \( p_{\text{delete}} = 0.33 \). Fig. 3.8 illustrates the problem of edge deletion. In this case, the threshold was about \( p_{\text{delete}} = 0.43 \) [7] [8].

In this case, the probability weighting was done through the algorithm
3.2. Previous Work

Figure 3.7: Probability of Spanning Cluster for Domain Deletion

Figure 3.8: Probability of Spanning Cluster for Edge Deletion
3.3 Testing Algorithms

<table>
<thead>
<tr>
<th>Lattice Size</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$40 \times 40$</td>
<td>129 s</td>
</tr>
<tr>
<td>$60 \times 60$</td>
<td>438 s</td>
</tr>
<tr>
<td>$80 \times 80$</td>
<td>879 s</td>
</tr>
<tr>
<td>$100 \times 100$</td>
<td>1470 s</td>
</tr>
</tbody>
</table>

Table 3.1: Simulation Times for Different-Sized Lattices.

described above, that is, for each vertex, we flip the POVM outcome and re-group the vertices to get a change in the number of domains and edges from which we can determine the flipping probability.

3.3 Testing Algorithms

In order to get a clear idea of how well the algorithm from Section 3.1 works, we first tried to reproduce the results of the previous work done. Instead of implementing the exact same program (which was written in C), we used MATLAB to produce results and see whether the results would still match even without reference to the original programs.

As can be intuitively seen, the algorithm that implements the probability weighting takes the most time. In order to realistically check if the algorithms give a reasonable answer, we apply the algorithm without the probability weighting first. By checking that we do have a percolation-type threshold, we can proceed to then apply the probability weighting. Here, we test the algorithm for 25 lattices for each of the different $p_{\text{delete}}$. The values of $p_{\text{delete}}$ increment from 0 to 0.7 in steps of 0.05. We do this because we anticipate that the threshold will be somewhere in between.

Fig. 3.9, 3.10, 3.11 and 3.12 are the graphs showing the number of lattices (out of 25) that contained connected paths for different values of $p_{\text{delete}}$, and for different lattice sizes ($40 \times 40$ to $100 \times 100$). To get an idea of how long each algorithm takes, consult Table 3.1.

We find that the percolation threshold is approximately 0.4 just by in-

\[9\text{Actually, the smallest multiple of 4 that is larger than or equal to these numbers}\]
3.3. Testing Algorithms

Figure 3.9: Percolation Threshold for $40 \times 40$ lattices without the Probability Weighting.

Figure 3.10: Percolation Threshold for $60 \times 60$ lattices without the Probability Weighting.
3.3. Testing Algorithms

Figure 3.11: Percolation Threshold for 80 × 80 lattices without the Probability Weighting.

Figure 3.12: Percolation Threshold for 100 × 100 lattices without the Probability Weighting.
3.4 Probability Weighting

In order to get a better idea of where the percolation threshold is; we can increase the resolution around 0.4 (0.35 to 0.45, increment $p_{\text{delete}}$ by 0.005 instead of 0.05. Fig. 3.13 illustrates this, with 40 by 40 lattices. This confirms that the threshold is approximately around 0.4.

Notice that this value of $p_{\text{delete}} = 0.4$ is higher than that predicted in previous work. However, since we did not implement the probability weighting, it can be expected for this discrepancy to occur.

3.4 Probability Weighting

Now we implement the probability weighting algorithm as outlined above. This implementation takes much longer, and so, to simply demonstrate the algorithm, we apply this algorithm only for much smaller lattices ($12 \times 12$) and for only very few of them (10 per point). The fact that we used very small and very few lattices is evidenced in Fig. 3.14: we see the downward trend, with 0.4 about half way through the decaying 'slope', but it does
not look at all like the percolation problem earlier. This simulation took 1343 seconds, almost the time for the non-probability weighted 100 × 100 lattices (each point also had 25, instead of 10 per point). If we want to get a better resolution around the threshold, we need both more lattices, and larger lattices, which qualitatively will take much longer, so it is not really feasible to implement the probability weighting in this way (in MATLAB at least).

### 3.5 Brute-Force Algorithm Summary

Table 3.1 shows the times for different lattices; but we want to see how the time scales with increasing sizes of lattices (for the unweighted lattices simulation). For this, we examine how long it takes for lattices of size 20×20, 30×30, 50×50 and 70×70 as well.

From Fig. 3.15, scaling looks as if it were either polynomial or exponential; it could be polynomial because we are looking only at the scaling with
3.5. Brute-Force Algorithm Summary

![Graph showing time vs lattice height](image.png)

**Figure 3.15**: Time for Simulation versus Lattice Height

respect to height; by graphing against the actual lattice size, there may be a better idea of the simulation time scaling.

From Fig. 3.16, the scaling looks roughly linear for increased lattice size, and so the algorithm is good qualitatively in this regard because the time scales linearly with lattice size.

### 3.5.1 Single Lattice

Applying the probability weighting algorithm, we first test how long it takes for the algorithm to work on a single 40×40 lattice. We found that it took 1300 seconds for the algorithm to perform the probability weighting on the lattice for a single 40×40 lattice which is the same as having the algorithm run for all the 100×100 lattices without the probability weighting! If we were to apply the same algorithm of the 40×40 lattices and change for different values of probability weighting, then the total time for 25 lattices for each of the 13 different values of \( p_{\text{delete}} \) would take nearly five days to complete, and it would scale much more for 100×100 lattices.
3.6 Algorithm Improvement

To make the simulation of probability weighting feasible, we have to find a way to streamline the probability weighting algorithm. Currently, the algorithm depends on the size of the lattice in two ways: first, for each vertex, we must apply the algorithm of grouping and counting domains and edges on the entire lattice, so for each individual vertex, we depend on the lattice size; second, the number of vertices on which we check increases with increasing lattice size. The main improvement to the algorithm is that we can remove the first dependence, so that, for each vertex, we can make the algorithm independent of the lattice size (first dependence). This will consequently make the second dependence also much faster because each vertex will take much less time.

3.6.1 Domain Grouping

The idea is that the Union-Find Algorithm does not need to be applied to the entire lattice when a vertex is changed, since the vertex POVM outcome
flipping is a 'local', rather than 'global effect'. That is, only a specific number of domains will be affected; and so the change in the number of domains and edges will be determined only by the change to these specific number of 'local' domains.

Fig. 3.17 illustrates this. For each vertex, there are at most three other domains that are connected to it, as in a hexagonal lattice, each vertex only has three neighbors. Since there are only three POVM outcomes, for the vertex and the neighbors, there are at most three domains that are either adjacent or contain the vertex in question (illustrated as the multi-colored vertex in Fig. 3.17). For simplicity, call the domains that contain all the vertices adjacent to the vertex in question plus the domain containing the vertex in question to be a **superdomain**. The claim is that the change in the
number of domains and edges depends only on the change of the number of
domains and edges within the superdomain, this depends on the fact that
the number of edges is counted before the mod2 operation. So consider
the three domains in the superdomain. No matter how the domains are
rearranged within the superdomain, all the edges to domains outside the
superdomain still remain, because this is before the mod2 operation. If we
change the POVM outcome of the vertex in question, only the domains that
are in the superdomain could possibly include or exclude this vertex, and so
the domains and edges within the superdomain are affected.

To implement this we have to modify the Union-Find algorithm and the
Graph Generator algorithm to work not on the entire lattice, but on a super-
domain that may be of variable shape. This means the localUnionFind.m
and localEdgeCount.m must work depending on the POVM outcomes of
the vertices and their domain labels, not on the fixed shape of the lattice
(which the ‘global’ versions did). To implement the probability weighting,
we included a few new programs:

1. ProbabilityShift.m Input: $A$, the matrix from HexPOVMTwo.m Output:
   $C$, the adjacency matrix and $A$, the modified hexagonal lattice. This
   is an overarching program that goes through each vertex, finding the
current POVM outcome, flipping it and deciding whether the flip re-
mains. This then generates the final probability-weighted adjacency
   matrix for the resultant graph $C$, and for the modified hexagonal lat-
tice $A$. This effectively encapsulates the probability weighting in a
   single program.

2. adjacentDomain.m Input: $A$, the hexagonal lattice, $(i,j)$, the coordi-
nate of the hexagonal lattice Output: The coordinates of the vertices
   that are in the domains adjacent to vertex $(i,j)$, the domain labels
   for each of these vertices and a vector of the different domain labels
   (listing them once). This is used to get all the relevant information
   for where to apply the local Union Find algorithm, as well as a way to
   count the number of domains after we apply the Union Find algorithm.
3.7 Time Improvement

3.7.1 Single Lattice Test

Recall that, for a single 40×40 lattice, it took about 1300 seconds to implement the probability weighting using the brute algorithm. With this local Union Find algorithm, the single 40×40 lattice takes only 3.65 seconds to implement the probability weighting.

Another useful measure of the speed of the new algorithm is how it scales with lattice size. Due to the prohibitive time it takes for the old algorithm to run, we only try lattices up to size 40×40, and see how the time scales with both the old algorithm and new algorithm, as shown in Table 3.2, which are ‘fresh’ runs, so the values may differ slightly from those given before. Both these graphs yield roughly quadratic increases in time, so while the new algorithm does not reduce the polynomial dependence on the height of
3.7. Time Improvement

<table>
<thead>
<tr>
<th>Lattice Height</th>
<th>Old Algorithm (s)</th>
<th>New Algorithm (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>34</td>
<td>0.51</td>
</tr>
<tr>
<td>20</td>
<td>81</td>
<td>0.83</td>
</tr>
<tr>
<td>24</td>
<td>117</td>
<td>1.26</td>
</tr>
<tr>
<td>28</td>
<td>320</td>
<td>1.70</td>
</tr>
<tr>
<td>32</td>
<td>562</td>
<td>2.35</td>
</tr>
<tr>
<td>36</td>
<td>893</td>
<td>2.95</td>
</tr>
<tr>
<td>40</td>
<td>1303</td>
<td>3.72</td>
</tr>
</tbody>
</table>

Table 3.2: Probability Weighting: Single Simulation Times for Different-Sized Lattices.

the lattice, it does make the increase much slower for increasing lattice size.\footnote{Although the new algorithm might look linear, by checking the value for a 100×100 lattice, the quadratic nature is shown, albeit a slow one.} This means we can apply this simulation feasibly for larger lattices (like 100×100, which, for this algorithm, takes 56 seconds, so a total of about 5 hours for a full simulation).

3.7.2 Full Simulation Test

Furthermore, recall that for the full simulation on 12×12 lattices took about 1343 seconds (with 10 lattices per \(p_{\text{delete}}\) value). With this new algorithm, the same simulation only takes 34 seconds. Fig. 3.18 gives the total graph of the 12×12 lattices with probability weighting, using this new algorithm. This only took about 124 seconds, even when more values of \(p_{\text{delete}}\) were used. Since these lattices are very small, the threshold is poorly defined, but it gives an idea of how much faster the simulation can run using the new algorithm.

Now we apply the full simulation on 60×60 lattices, 25 points per value of \(p_{\text{delete}}\). Fig. 3.20 gives the result, and it took about an hour to complete (3617 s). We note that the percolation threshold seems to be less than that of the unweighted lattices, around 0.35-0.37, and yet this is still above the estimated value done in previous work with the brute-force algorithm which yielded a value of 0.33 for domain deletion. This is an error related to the
3.8. Accuracy

accuracy of the implementation of the algorithm, and must be investigated further. What is (qualitatively) encouraging is that the algorithm at least reduces the percolation threshold from the overestimate of 0.4 without the weighting.

As seen in both Fig. 3.19[11] and 3.20, the percolation threshold is not as well defined as in the case of unweighted lattices. Thus, for future work, we might require that we go to more lattices (50-100) and larger lattices (100-200 height).

3.8 Accuracy

What other indicators of the accuracy of this algorithm exist other than only comparing it with the percolation threshold of previously-run simulations? Qualitatively, we could perhaps compare the features of ‘average’ probability-weighted lattice using the brute algorithm with the one generated by the new algorithm, such as (1) the ratios of POVM outcomes that result, (2) the average size of the domains and (3) the number of edges that result. Perhaps another good measure would be to compare the number

---

[11] This took 1037 seconds to run on MATLAB.
3.8. Accuracy

Figure 3.19: 40×40 Lattices with Probability Weighting using New Algorithm

Figure 3.20: 60×60 Lattices with Probability Weighting using New Algorithm
3.8. Accuracy

<table>
<thead>
<tr>
<th>Average Domain Size</th>
<th>Average Domain Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3369</td>
<td>2.2460</td>
</tr>
<tr>
<td>3.3589</td>
<td>2.0096</td>
</tr>
<tr>
<td>3.2018</td>
<td>1.8834</td>
</tr>
<tr>
<td>3.3211</td>
<td>1.9266</td>
</tr>
<tr>
<td>3.2617</td>
<td>1.9626</td>
</tr>
<tr>
<td>3.2294</td>
<td>1.9266</td>
</tr>
<tr>
<td>3.1927</td>
<td>1.9266</td>
</tr>
<tr>
<td>3.4112</td>
<td>2.1320</td>
</tr>
<tr>
<td>3.2430</td>
<td>1.9626</td>
</tr>
<tr>
<td>3.1963</td>
<td>1.9178</td>
</tr>
</tbody>
</table>

Table 3.3: Typical Lattice Characteristics: Average Domain Size and Degree, for 20×20 Lattices; Brute Force Algorithm

of lattices that do result with a connected path (4) for a specific value of $p_{\text{delete}}$. If we can show that at least for one value of $p_{\text{delete}}$ we have similar outcomes, we can have an idea whether the algorithm (and its implementation) need modifying. We want to check whether both algorithms produce the same ‘typical’ graphs with the probability weighting, for example, the average size of the domains and the average domain degree.

3.8.1 Average Size of Domains & Average Domain Degree

We can easily calculate the average size of the domains because we already know the total size of all the domains together, we need only to divide it by the total number of domains. Another good indicator is the average degree of the domains; that is, to how many other domains is an average domain connected to? Tables 3.3 and 3.4 show the average domain size and average domain degree for ten different trials for the brute-force and the new algorithm.

For the brute-force algorithm, the average domain size was 3.2753, and the average domain degree was 1.9894. For the new algorithm, the average domain size was 3.2600, and the average domain degree was 2.0253. This is encouraging as the average domain size and domain degree must be integers.
3.8. Accuracy

<table>
<thead>
<tr>
<th>Average Domain Size</th>
<th>Average Domain Degree</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3611</td>
<td>1.9444</td>
</tr>
<tr>
<td>3.3299</td>
<td>2.1649</td>
</tr>
<tr>
<td>3.2511</td>
<td>1.8502</td>
</tr>
<tr>
<td>3.2574</td>
<td>2.0792</td>
</tr>
<tr>
<td>3.1728</td>
<td>2.1990</td>
</tr>
<tr>
<td>3.2056</td>
<td>1.9626</td>
</tr>
<tr>
<td>3.3073</td>
<td>2.0488</td>
</tr>
<tr>
<td>3.2709</td>
<td>2.0690</td>
</tr>
<tr>
<td>3.1754</td>
<td>1.9905</td>
</tr>
<tr>
<td>3.2685</td>
<td>1.9444</td>
</tr>
</tbody>
</table>

Table 3.4: Typical Lattice Characteristics: Average Domain Size and Degree, for $20 \times 20$ Lattices; New Algorithm

It might be helpful to repeat the same experiment for much larger lattices, except we only do it for the new algorithm, as the brute-force one would take too long. Previous work was done using the brute force algorithm, so we have a ‘literature’ value to compare with. For $100 \times 100$ lattices (10 of them) the average domain size was 3.4687, and the average domain degree was 2.0167 (average domain degree was 3.52 in previous work).

3.8.2 20×20 Lattices with Connected Paths at $p_{\text{delete}} = 0.35$

In order to measure how well the two algorithms agree, we want to test how many lattices do contain connected paths out of 100 (instead of 25) for very small lattices ($20 \times 20$) for a fixed probability of deletion (0.35). This is only feasible to calculate for small lattices because the old algorithm is too slow for larger lattices. The primary problem with this type of comparison is that $20 \times 20$ lattices give relatively unstable results compared with larger lattices. For $p_{\text{delete}} = 0.35$, 83 of the lattices did contain a connected path according to the brute-force algorithm (7839 s), while there were 74 of these lattices with a connected path according to the newer algorithm (84.05 s).

\[\text{which may ‘average out’ the errors that occur in smaller lattices}\]
Chapter 4

Conclusion

Measurement based quantum computation is one potential physical implementation of quantum computers. In particular, using AKLT states and transforming them into cluster states makes this feasible in a physical setting, provided we can prove that such a transformation is robust. The robustness of an AKLT state’s transformation critically depends on the two necessary properties for the graph state to be transformed to a cluster state: that the domains are microscopic compared with the size of the lattice, and that there exists a connected path from the left to right of the lattice. The focus of the study was on the effects of erroneous POVMs on the robustness of the AKLT state, simulated by an increasing value of $p_{\text{delete}}$ that corresponded to erroneous POVMs whose effect was to delete domains. Previous work had shown that this yields a percolation problem in which, for values of $p_{\text{delete}} \leq 0.33$, there is always a connected path, but beyond that, there is no connected path.

4.1 Results Summary

The original motivation of the thesis was to explore other effects of different types of erroneous POVMs on the robustness of the AKLT state. However, significant problems arose due to the time-consuming nature of implementing the probability weighting algorithm on lattices that simulate the entanglement of the vertices in the lattice with one another. Thus, the new motivation was to try to speed up this part of the algorithm. A new algorithm was proposed in which the preparation of probability-weighted lattices could be produced quickly by performing the vertex POVM outcome flipping only on local ‘superdomains’, rather than on the entire lattice. This
was achieved by writing up a ‘local’ Union Find algorithm that re-labelled the relevant domains for the new POVM outcome.

We checked the speed increase of this new algorithm. Conceptually, it should reduce one ‘layer’ of dependence on the lattice size. Checking for different lattice sizes, it was found that the new algorithm appeared to still have the same polynomial power dependence on the lattice size as the old algorithm. However, the increase in time to complete the task for a given increase in lattice size was much smaller than with the older algorithm. It was found that for a single 40×40 lattice, the new algorithm took 3.65 seconds while the older algorithm took 1300 seconds.

Next, we checked the accuracy in two ways. First by an overview method of actually varying \( p_{\text{delete}} \) and finding how many lattices (out of 25) still had a connected path. This resulted in a graph that showed a decreased percolation threshold (compared with the lattices without probability weighting), of about 0.37, but still not the 0.33 value that was obtained before. Second, we checked the properties of individual lattices by checking the average domain size and average domain degree. This yielded encouraging results because both the brute force algorithm (which we know is correct, but may have had wrong implementation) and the new algorithm yielded similar values for both.

This algorithm for probability weighting is useful in setting up the lattice even before the POVM effects are considered; thus, by speeding up the simulation, this algorithm allows us to more efficiently explore different POVM effects in the future.

4.2 Future Work

Overall, a general theoretical understanding of the underlying mathematics would be very good to help examine how accurate the algorithms are, and give a theoretical idea of how quickly the algorithms should run. Percolation problems are a problem in probability theory, and so it may be possible to examine other effects theoretically other than domain deletion. Using big-O notation to provide a framework to analyze the efficiency of the algorithms
4.2. Future Work

would help us understand whether the new scheme does provide any sort of advantage for producing probability-weighted lattices.

Another important work that must be done is checking the accuracy of the algorithm. Here we presented preliminary checks of accuracy. Clearly if we were to use many more lattices and much larger lattices, we could obtain more information on how accurate the new algorithm is compared with the (correct) brute-force algorithm. Another way to check for the accuracy is to use forced outcomes (rather than probabilistic ones) and see the resultant lattice.

The Union-Find algorithm is well known in computer science and many implementations are far more efficient than the one used here, so in the future, we could use these to increase the efficiency even further, and ensure that it is implemented correctly.

Finally, it would be hoped that more POVM effects could be analyzed both theoretically and computationally in the future. It is unfortunate that the algorithms took so long to implement, but it is hoped that this improvement to the efficiency to the preparation of weighted lattices will be helpful in future simulations.
Bibliography


