MWPM Decoder

Zicheng Yang,Yuheng Ma,Yiming Xiao,Zikang Lv

CKC College Zhejiang UniversityHangzhou 310058
Dated: June 9th, 2024

Abstract

In the field of quantum informatics, the Minimum Weight Perfect Matching (MWPM) decoder stands as a pivotal error-correction technique within surface code frameworks. This paper do a coarse research on the implementation and evaluation of the MWPM surface code decoder through two principal algorithms: the Blossom V algorithm and a localized Dijkstra algorithm. We rigorously test their error thresholds and time complexities, and further compare their performance against a contemporary MWPM decoder library, PyMatching 2.0. Furthermore, we extend our study to juxtapose the MWPM decoder's performance with other kinds of surface code decoders, specifically the cellular automaton decoder. Hopefully, this paper will provide an introduction to the MWPM decoder for future students who do the similar project . For more information, please visit our official GitHub repository at `https://github.com/Winfred666/SurfaceCode-MWPMDecoder.git`.

I. Introduction

With the popularity of quantum computing spreading in the investors and researchers,quantum error correcting has also been much under scrutiny.Though in the early time, people have the scepticism that weather quantum computing can keep and manipulate a quantum state error-free for a long time"Quantum computing: dream or nightmare?" ,the development of quantum error correction making it more optimistic.

Among so many prospective codes of quantum error correcting,the most promising classes is the surface code,witch is proposed by Alexei Kitaev in 1997.("Fault-tolerant quantum computation by anyons.")the topological concept of the surface code or toric code is to encode the qubit in a 2D lattice of physics qubits system which has some specific boundary condition("Topological quantum memory").In this case ,we can influence and correct a wide range of qubits by applying a given way on the torus.Although challenges of fabricating a large-scale quantum memory still exist ,the progresses on experimental advances in recent years give us hope to make it a reality.("Implementing a strand of a scalable fault-tolerant quantum computing fabric""Logic gates at the surface code threshold: Superconducting qubits poised for fault-tolerant quantum computing.")

Another significant period of a quantum error correcting task is to decoder.It is natural that we want to find an optimal solution to figure out the most probably error and correct it for every given measurement,witch is called the Maximums Like Algorithm.But it seems unpractical for its exponential operation time.("On the hardnesses of several quantum decoding problems")So how to balance between performance and running time of the algorithm becomes the priority .A lot of solution have been designed to solve such problems,such as the graph algorithm and neural network("Decoders for Topological Quantum Error Correction").

In this paper,we will take an insight of a kind of graph algorithm,the Blossom Algorithm. As well as we will compare it with other decoder algorithm from multiple dimensions.

II. Background on Quantum Error Correction

A standard error correction usually consists of four step"Decoders for Topological Quantum Error Correction":

1. **Encode** : encode the quantum state to another state state:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \rightarrow |\psi_E\rangle = \alpha|000\rangle + \beta|111\rangle$$

2. **Detect** : detect the quantum errors by measurement
3. **correct** : correct those errors and decode the qubits to get the corresponding logical state

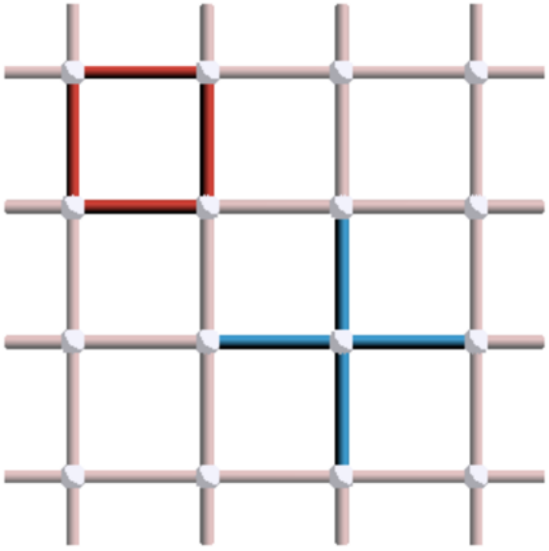


Figure 1: *An illustration of the lattice.The unit marked by the red line or the blue line shows the A_s operator and B_p operator respectively*

4. **compute** : compute logical operations on the state by redefining a universal gate set on the code.

Here we will focus on the first three steps and outline our work by the following steps:

1. **Introduce errors**
2. **Plot syndromes**
3. **Match the syndromes**
4. **Get the result**

A. toric code

For a better understanding of the toric code,we assume a L×L lattice,where every single qubit lives on the stage,so we have L^2 qubits.The stabilizer group of toric code is generated by two types of operator A_s and B_p"Neural Network Decoders for Large-Distance 2D Toric Codes":

$$A_s = \bigotimes X_q, B_p = \bigotimes Z_q$$

The X and Z are the Pauli matrix the Hamiltonian of the system is the linear combination of the tow operator:

$$H = -A_s - B_p$$

We show the stabilizer and the lattice in Figure 1.

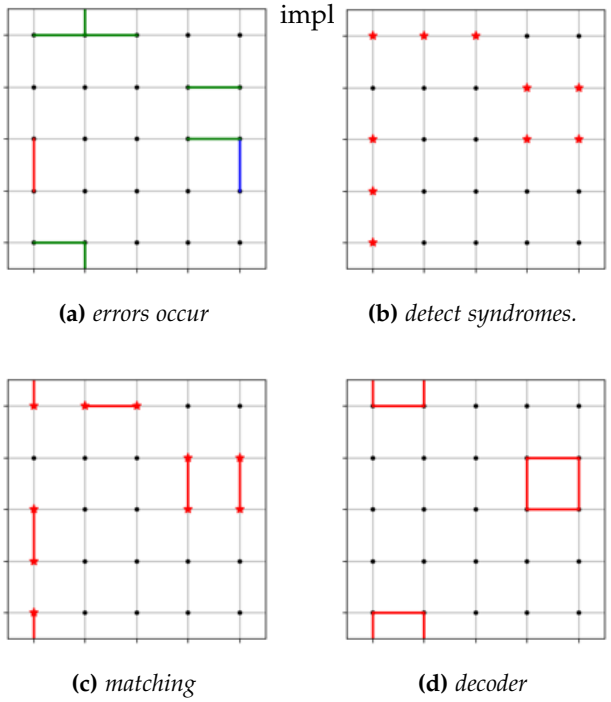
B.Quantum Error

Generally speaking,the noisy and environment disturbance is inevitable in real quantum system.It is impossible for us to clarify the procedure of the coupling between the system and the environment.But in fact,we can easy the challenge by simplifying all the quantum errors into two types of error:

$$\text{Bit-flips error:} |\psi\rangle \rightarrow \hat{X}|\psi\rangle$$

$$\text{Phase-slip error:} |\psi\rangle \rightarrow \hat{Z}|\psi\rangle$$

The X error and the Y error are the fundamental mode of quantum error and they can appear at the same time,which we called Y error.In this paper,we assume that each error has the same probability $p/3$



(a) *errors occur* **(b)** *detect syndromes.*

(c) *matching* **(d)** *decoder*

Figure 2: *the procedure of an error correction.(a)-(d) shows the different stage respectively*

Decoder

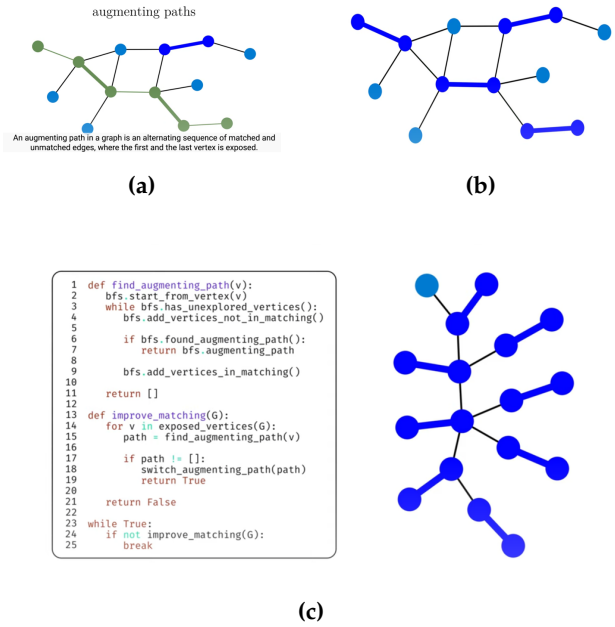
As show in Figure 2(a)-(d),when a series of errors happen and we detect this error by the measurement and collect all the syndromes.The task of decoder is to restore the most possible error.The green line ,red line and blue line in Figure 2(a) indicate the X error ,Z error and Y error.It is showed in the figure 2(b) that we detect these error with the measurement we mention above.The next step and the most challenging part is to match the syndromes and decoder them to get result.We definitely hope

the result we get is non-trivial,but sometimes the logical error is inevitable due to the limitation of decoder algorithm"Logical error rate scaling of the toric code".In the next section,we will introduce the Blossom Algorithm and test its logical error and compare it with other decoder algorithm.

III. Blossom Algorithm

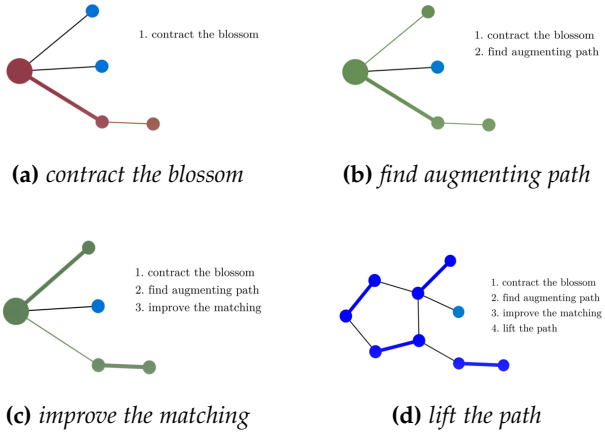
Here,we implement a decoder algorithm called Blossom Algorithm,which is first proposed by Edmonds in 1965.Edmond's idea requires $O(n^2m)$ times,where n is the number of nodes in the graph and m is the number of edges.The Blossom Algorithm has been developed with the time and different improvement has been conducted.It is impossible for us to cover all the creative ideas.Before we discuss the kind of minimum weight matching Blossom Algorithm,we first get an insight of the oldest Blossom Algorithm.

1. **Augmenting Paths**:And then,an augmenting path is required.Like branches on a tree,an augmenting path is a path that stars and ends at unmatched vertices and alternate between unmatched and matched edges,as show in Figure 3(b).It can improve the size of the current matching by switching the matched and unmatched edges,as show in Figure 3(c).And the process of matching can repeatedly until no free vertices are left,at which point,we know the matching is maximum,as show in figure 3(d).
2. **Blossoms**:But things don't always go well.When there is a blossom,a cycle containing an odd number of pseudo nodes, where a pseudo node is either a vertex or another blossom,the algorithm fails,as show in Figure 3(e) and 3(f).We can see that since the augmenting path is longer than the shortest path,so is not found.
3. **Blossom Contraction and Expansion**:In this case,When an odd-length cycle (blossom) is detected, we should contract it into a single vertex, simplifying the problem and after finding a matching, expand the blossoms back to update the matching accordingly.The Figure 5(a)-(d) show the solution when a blossom is detected step by step.
4. **Iterative Process**:Finally,repeat the search for




```
def find_augmenting_path(v):
    bfs.start_from_vertex(v)
    while bfs.has_unexplored_vertices():
        bfs.add_vertices_not_in_matching()

        if bfs.found_augmenting_path():
            return bfs.augmenting_path

        bfs.add_vertices_in_matching()

    return []

def improve_matching(G):
    for v in exposed_vertices(G):
        path = find_augmenting_path(v)

        if path != []:
            switch_augmenting_path(path)
            return True

    return False

while True:
    if not improve_matching(G):
        break
```

Figure 3: *The procedure of an error correction. (a)-(c) show the different stages respectively.*



(a) *contract the blossom* **(b)** *find augmenting path*

(c) *improve the matching* **(d)** *lift the path*

Figure 4: *the procedure of an error correction.(a)-(d) shows the different stage respectively*

```python
def find_augmenting_path(v):
    bfs.start_from_vertex(v)
    while bfs.has_unexplored_vertices():
        bfs.add_vertices_not_in_matching()

        if bfs.found_blossom():
            bfs.contract_blossom()
            find_augmenting_path(v)
            bfs.lift_blossom()

        if bfs.found_augmenting_path():
            return bfs.augmenting_path

        bfs.add_vertices_in_matching()


    return []

def improve_matching(G):
    for v in exposed_vertices(G):
        path = find_augmenting_path(v)

        if path != []:
            switch_augmenting_path(path)
            return True

    return False

while True:
    if not improve_matching(G):
        break
```

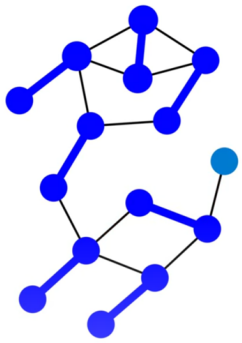


Figure 5: *the whole process and the code*

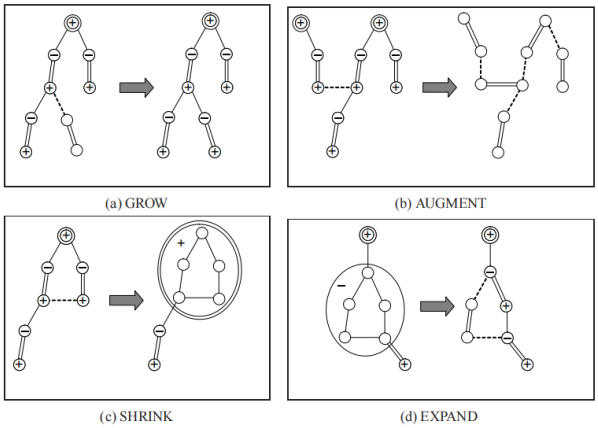


Figure 6: *An illustration of the minimum weight perfect matching*

augmenting paths, contraction, and expansion until no more augmenting paths can be found.The whole process and code is presented in Figure 6.

The minimum weight perfect matching algorithm is originated from the initial idea,but it has some improvement to handle with the practical problems. The implementation of the algorithm is the following:for every vertex i,it has a weight $l(i)$ and every edge is a coupling of the vertices i, j,it meets the condition:$w(i,j) \leq l(i) + l(j)$.Every perfect minimum weight matching,the sum of the weight edge is:

$$\begin{aligned} \text{val}(M) &= \sum_{(u,v)\in M} w(u,v) \\ &\leq \sum_{(u,v)\in M} (l(u) + l(v)) \\ &\leq \sum_{i=1}^{n} l(i) \end{aligned}$$

Defining z_u as the vertex labeling of u,and we also define $e(u,v)$ an equality edge if $(z_u + z_v = weight(e))$,and at this time the edge labeling of edge is called z_e.It requires $z_e = z_u + z_v - weight(e) = 0$. The augmenting path composed of "equality edges" is continuously expanded, and since all the edges used for expansion are "equality edges", the final maximum weight perfect matching obtained is still all "equality edges". In Figure 7,we demonstrate the whole process of this algorithm in detail.It consists of four step:

1. **Grow**:If edge(u,v) is tight,$l(u) = +$and $l(v) = \emptyset$ then the tree to which u belongs can be "grown" by acquiring node u and the corresponding matched node,as shown in the Figure 7(a)
2. **Augment**:If edge $(u,v)$is tight,$l(u) = l(v) = +$and u, v belong to different trees then the cardinality of matching x can be increased by "flipping" variable x_e for edges e along the path connecting the roots of the two trees,as shown in Figure 7(b).All nodes in the trees become error-free.
3. **Shrink**:If edge $(u,v)$is tight,$l(u) = l(v) = +$and u, v belong to the same tree then there is cycle of odd length that can be shrunk to a blossom,as shown in Figure 7(c).The dual variable for this new blossom is set to 0.
4. **Expand**:If node v is a blossom with $y_e = 0$ and$l(v) = -$ then it can be expanded.

the u^- represents odd vertex in an alternating tree,the u^+ represents the even vertex in an alternating tree and $u^\emptyset$ represents an vertex not in any alternating tree.

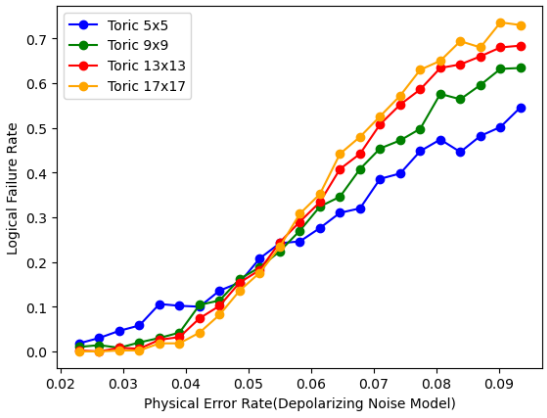
IV. Cellular Automation Decoder

Besides the Blossom Algorithm,we also get an insight of another decoder,the Cellular Automation Decoder.

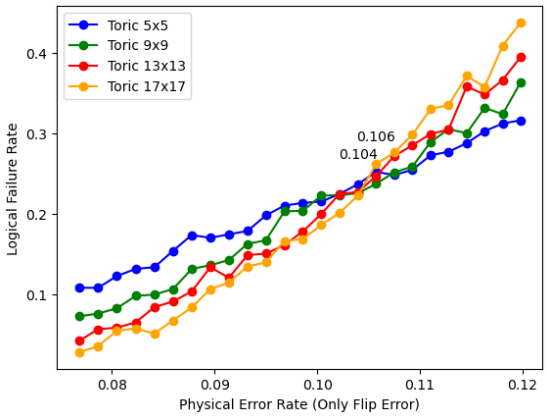
V. Test and result

There is two critical parameter to evaluate the performance of a decoder:the error threshold and the time complexity.

However, One pitfall about test set of surface code is the usage of noisy model. Though depolarizing noise model is the best one to simulate the real condition, most of the test program just generate only bit-flip errors to test the decoder. So in 7b we translate osur physical error rate to the unit of commonly used one.



(a)



(b)

Figure 7

In this part,we will show our results and compare different decoders with their error threshold and time complexity .You can see more details in our github website,here,we just give a brief introduction. The error threshold is a critical parameter in the design and implementation of fault-tolerant quantum computation.It is related to the threshold theorem,which states that arbitrarily long quantum computations are reliable below a specific error rate that varies with the quantum decoders we use.Above the specific error rate,the reliability of the quantum computation rapidly deteriorates,making it impossible to conduct a fault-tolerant quantum computation. Figure 8 (a)-(b) shows the logical error rate varying with the phase-flip and bit-flip error rate under the noise model respectively.For four different scale quantum memories, we set up 25 different groups of error rates,each group of error rates for 500 simulation tests, and calculate the logic error rate.We can calculate the error threshold by recording the intersection of four lines.And it shows the error threshold is about 0.104.As the time complexity ,shown in the Figure 9 (a)-(b) and table 1,the minimum weight perfect matching has the $O(n^3)$ time complexity.And the other,the Union Code represented by the previous group,has the $O(n)$ time complexity.Though,the minimum weight perfect matching is inferior in time complexity to previous work,it has an advantage over it by its higher error threshold.

Table 1: *Time complexity of different decoders (N is physical qubit number)*

Decoder	Time Complexity
MWPM(original)	$O(n^3 logn)$
MWPM(M-local-dijkstra)	$O(n^2 Mlogn)$
MWPM(sparse blossom)	$O(n)$
Cellular Automaton	$O(n^2)$
Union Find	$O(n)$

VI. Conclusion