ZX-calculus and Surface Code Lattice Surgery

Internship Report

Yicheng Zhou

August 30, 2019

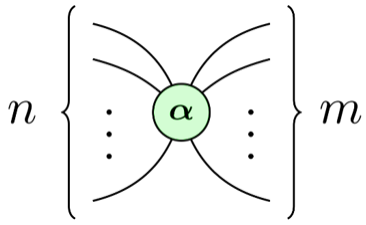
Contents

1 ZX-calculus

ZX-calculus is a graphical language for diagrammatic reasoning in quantum mechanics and quantum information theory. ZX-calculus consists of ZX-graphs and a set of axioms, where ZX-graphs represents linear maps and the axioms tell us which ZX-graphs represent the same linear maps. For its introduction (axiomatization, completeness), we refer to [JPV19].

1.1 ZX-graph

Roughly speaking, ZX-graphs are generated by (1) usual edges; (2) Hadamard gates; (3) constructive units of the following type representing linear map $|0\rangle^{\otimes m}\langle 0|^{\otimes n} + e^{i\alpha}|1\rangle^{\otimes m}\langle 1|^{\otimes n}$:



We denote such vertices as $sp_g(n, m, \alpha)$, meaning "green spider" with n input "legs" and m output "legs" and phase α. α is omitted if it is zero. By changing green to red, we simply change the base to $|\pm\rangle$ and sp_g to sp_r.

Sometimes, it is more convenient to represent an edge with one Hadamard gate by a blue edge as is done in PyZX, which is a free and open source software that allows users to efficiently rewrite ZX-diagrams using built-in simplification strategies [KvdW19].

For finitely many linear maps represented by ZX-graphs, doing the tensor product $\otimes$ is equivalent to putting ZX-graphs abreast vertically, the composition $\circ$ is equivalent to putting ZX-graph abreast horizontally in order and connecting the corresponding open edges.

Sometimes, we consider only the ZX-graphs with angle (or phase) α in some restricted set, for example, $\frac{\pi}{2}\mathbb{Z}$, $\frac{\pi}{4}\mathbb{Z}$, etc. With such restrictions, the sub-ZX-calculi are called, for example, the $\frac{\pi}{2}$-*fragment*, the $\frac{\pi}{4}$-*fragment* of the general ZX-calculus.

1.2 Axiomatization and completeness

Axioms can help us simplify a ZX-graph. Below in Figure 1 are some axioms frequently used in practice. We hope to have completeness of ZX-calculus requires that for any two equivalent ZX-graphs (i.e. representing the same map), we can transform one to the other by applying axioms for finitely many times. However, we may have to add other axioms and obtain the following results [JPV19]:

1. The axioms in Figure 1 is complete for the $\frac{\pi}{2}$-fragment;
2. With four additional axioms, the $\frac{\pi}{4}$-fragment is complete;
3. By adding only one additional axiom to the $\frac{\pi}{4}$-fragment, we obtain the completeness of the general ZX-calculus.

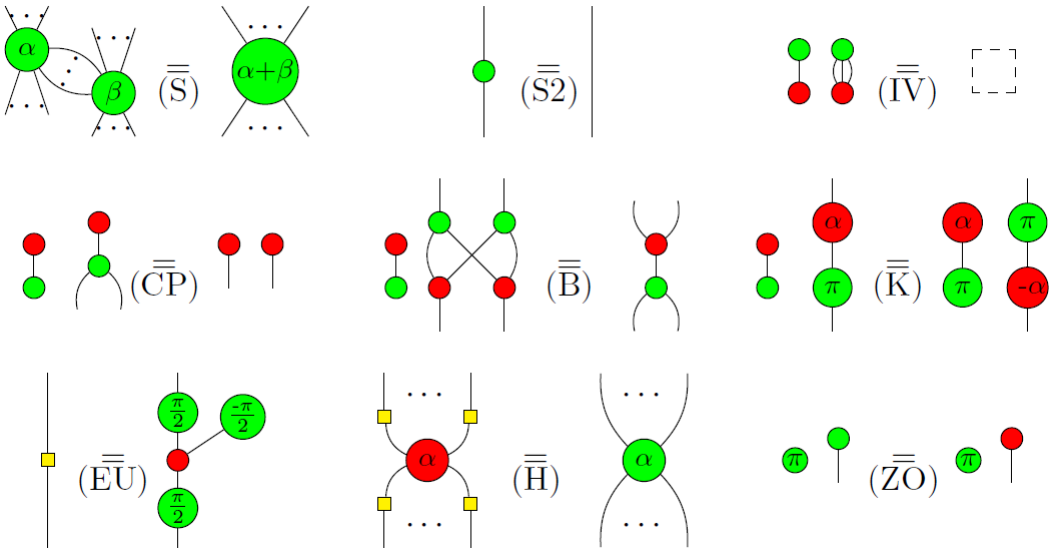


Figure 1: [JPV19] A set of axioms for ZX-calculus.

1.3 Useful identities

Here are some useful identities of ZX-calculus that one may use frequently:

$$\begin{aligned} sp_g(n,1)\circ sp_g(1,1,\alpha+\beta)\circ sp_g(1,m) &\equiv sp_g(n,1,\alpha)\circ sp_g(1,m,\beta) \equiv sp_g(n,m,\alpha+\beta) \\ (\text{Hadamard})^{\otimes n}\circ sp_g(n,m)\circ(\text{Hadamard})^{\otimes m} &\equiv sp_r(n,m) \\ (\text{red } \pi\text{-phase})^{\otimes n}\circ sp_g(n,m,\alpha)\circ(\text{red } \pi\text{-phase})^{\otimes m} &\equiv sp_g(n,m,-\alpha) \end{aligned}$$

where $\equiv$ means equality modulo constant factors and the composition $\circ$ of ZX-graphs is consistent with their relative positions.

2 Surface code lattice surgery

We consider here planar surface code, which is a stabilizer code whose stabilizers are represented by small units. So it uses many physical qubits to encode one logical qubit. It has a high estimated error correction threshold of 1%, a simple error correction procedure and allows a simple approach to magic state distillation [dBH17],[Lit18]. For basics of surface code and split merge operations, we refer to [HFDVM12]. There, we have an explicit example of performing CNOT using split-merge. We can represent it in a somewhat abstract way as follows, where green edges stand for Z-logical operators and red for X:

2.1 ZX-graph and lattice surgery

With ZX-calculus, we can simplify a quantum circuit in, for example, depth, T-counts, etc. However, the result of simplification is usually a ZX-graph of degree greater than 3. But ZX-graph of a CNOT has degree 3, so the physical realization of ZX-graph seems a problem. However, we can interpret the constructive unit $sp(n,m,\alpha)$ as a merge $sp(n,1)$ plus a one-qubit rotation $sp(1,1,\alpha)$ plus a split $sp(1,m)$ (section 1.3).

For a one-qubit patch, there are two Z-edges and X-edges. If we put n patches in a horizontal line, with horizontal X-edges and vertical Z-edges, then measure out the stabilizers between each pair of neighbouring

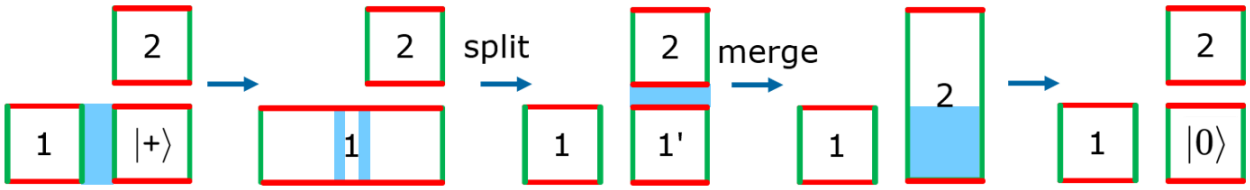


Figure 2: Lattice surgery implementation of CNOT (1 controls 2). The blue areas represent measurements of stabilizers in the first three schemata, Z-measurement in the fourth.

patches, then we are actually measuring the mutually commuting operators $Z_1Z_2, Z_2Z_3, \ldots, Z_{n-1}Z_n$. On the one hand, this is a merge of n patches along Z-edge. On the other hand, this operation is represented by $sp_g(n,1)$ except that there are perhaps error correction vertices π-phases on some $n-1$ input edges, which can be calculated from the measurement results of $Z_1Z_2, Z_2Z_3, \ldots, Z_{n-1}Z_n$. Similarly for split, and for split and merge along X-edge. So we have

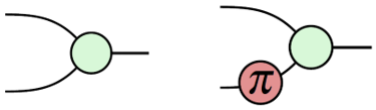
Lemma 2.1. *Modulo error correction π-phases, $sp_g(n,1), sp_g(1,m)$ represent respectively merge of n patches along Z-edge and split of a patch as m patches along Z-edge. For sp_r, just change Z to X.* □

3 Pauli-Fusion model

We have seen that for lattice surgery, we must do en error correction *before* we detect the error, so it is physically unrealistic in practice. However, Pauli-Fusion context allows us to tackle this problem. The following part is essentially the same as in [dBDHP19], but we give its modified version in the case where *Hadamard gates are represented by blue edges* and *all red vertices are changed to green by conjugation with Hadamard gates* (i.e. changing color of edges).

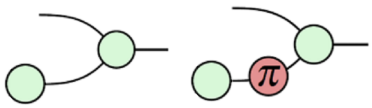
3.1 Definitions and properties

Lemma 3.1. [dBH17] *A smooth merge operation (i.e. merge along Z-edge) can be represented by*



where the error π-phase can be on either input edge. For a rough merge (i.e. along X-edge), just interchange the colors.

This can explain why we have to prepare a $|+\rangle$ patch for extending a given patch by a merge. This amounts to plug a green node to ZX-graphs above:



In ZX-calculus, for a green vertex with phase α totally surrounded by red π-phase, we can remove all these π-phase by negating α as $-\alpha$ (section 1.3). In particular, for a green node with phase 0, these red π-phases are simply absorbed. Therefore the above two ZX-graphs are equivalent, which means no error correction is needed in this case.

In a general merge represented as in Lemma 3.1, however, we cannot *propagate* the error correction node π-phase (by applying identities in section 1.3) to make two input edges "clean": at least one of the two input edges contains an error π-phase. Therefore, we must do an error correction *before* we detect the error, which is physically impossible.

Nevertheless, in a larger ZX-graph where there exist other nodes, it is possible to propagate π-phase to somewhere "later" than this merge. For example in the following ZX-graph representing CNOT (Figure 3), we can propagate the error π-phase for w to the edge below (by section 1.3), so that we can do split at v, then merge at w, finally correct the error π-phase later on the edge below if needed. For a CZ gate (just turn the red vertex green and turn the edge between two green vertices Hadamard), it is essentially the same since we have $H \circ X = Z \circ H$.

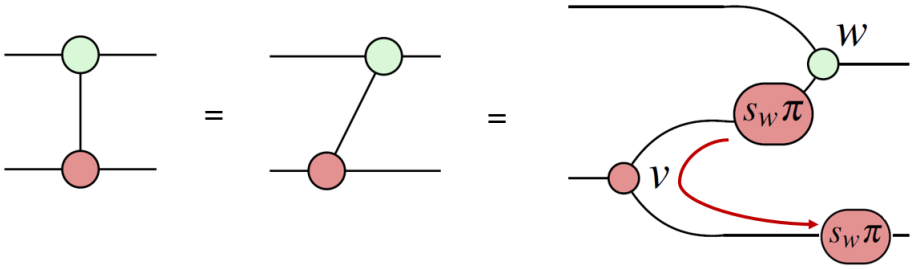


Figure 3: Error correction for CNOT implemented by lattice surgery: the red arrow shows the propagation of the error correction vertex produced at merge w to somewhere that can be "later" than w.

The Pauli-Fusion model is meant to make this idea explicit. We modify the definition of *graph-like ZX-graph* in [dBDHP19] as follows, while the other definitions, properties and theorems, such as definitions of corrector and PF-flow, the procedure of error correction, etc., stay the same as in [dBDHP19].

Definition 3.1 (Substitution for Definition 7 in [dBDHP19])**.** A (labelled) ZX-graph is in a *graph-like form* (or is *graph-like*) if it does not have red vertices, connections between spiders of the same color, parallel wires nor loops on any single vertex.

By turning all red vertices in green and changing the corresponding edge types, condensing all spiders, and removing all loops and (pairs of) parallel edges, it is easy to show that:

Lemma 3.2. [dBDHP19] *Any ZX-graph can be transformed into a graph-like ZX-graph.*

Remark 3.1. PF-Flow is not unique, even modulo order-preserving isomorphism. For example for a CNOT represented as above, each of green and red vertices can precede the other.

3.2 An efficient algorithm for finding PF-flows

Theorem 3.3. *Given a graph-like ZX-graph D, there is a polynomial time algorithm to decide whether it has a PF-flow and to construct a PF-flow if one exists.*

Proof. The algorithm PF-FLOW FINDING given below satisfies the conditions. The general idea is searching backwards vertices that can be corrected "later".

Data: Signature $(\mathscr{G}_D, \mathscr{I}, \mathscr{O}, \mathscr{P})$ of a graph-like ZX-graph D.
Result: Construct a PF-flow if one exists, else return "No solution!".
initialization:
  $M \coloneqq \mathscr{O}$, set of marked elements;
  $\delta M \coloneqq \mathscr{O}$, set of newly marked elements;
  $\preceq \coloneqq \{(u,u) : u \in V(\mathscr{G}_D)\}$, partial order relation on M;
  $f \coloneqq \varnothing$, function on the empty subset $\varnothing \subset V(\mathscr{G}_D)$;
  $\mathfrak{C} \coloneqq \varnothing$, empty set of corrector-sets.
repeat
  Reset $\delta M = \varnothing$;
  Set $R = \{u \in V(D)\backslash M : \exists$ a set $C_u \subset M \bigcup \mathscr{P}$ such that $\mathrm{Odd}(C_u)\backslash M = \{u\}\}$;
  $Treated \coloneqq \varnothing$, set of vertices treated in this repeat loop;
  for $v \in V(D)\backslash M$ **do**
    if $v \in Treated$ **then** skip to the next v.;
    Let $N_v \coloneqq N(v) \bigcup \{v\}$ if $N(v) \bigcap M = \varnothing$, otherwise $\coloneqq N(v)\backslash M$; vertices to test correctability;
    Let $F_v \coloneqq N_v \backslash R$, set of non-correctable vertices "nearby" to v;
    if $\#F_v \leq 1$ *and* $v \notin F_v$ **then**
      $\delta M \coloneqq \delta M \bigcup \{v\}$, add v to the set of newly marked elements;
      For each $u \in N_v \bigcap R$, let $C_{u,v} = C_u$ as constructed above, and update $\mathfrak{C} \coloneqq \mathfrak{C} \bigcup \{C_{u,v}\}$.
    end
    Update $f \coloneqq f \bigcup \{(v,w)\}$ if $F_v = \{w\}$, otherwise $\coloneqq f \bigcup \{(v,m)\}$ with m picked in M;
    Update $Treated \coloneqq Treated \bigcup N_v$;
  end
  Update the partial order $\preceq \coloneqq \preceq \bigcup (\delta M \times M)$;
  Update the set of marked elements $N \coloneqq M \bigcup \delta M$.
until $\delta M = \varnothing$;
return $(\preceq, f, \mathfrak{C})$ if $V(D) \subset M$, otherwise $(\varnothing, \varnothing, \varnothing)$.

□

Remark 3.2. There are two uncertainties: (1) the order in which we go through $V(D)\backslash M$ at line 13; (2) the m we picked from M at line -7. It is not clear whether a "good" choice can be made to optimize something.

4 Lattice surgery implementation of ZX-graph

From now on, we will consider only executable ZX-graphs in the sense of Pauli-Fusion context, but we must keep in mind that error corrections can always take place even though it is not our main topic in the following sections.

4.1 Problem description: split-merge only once

When we do several splits and then several merges just once, we are led to consider the decision problem defined in [HND17], which we shall reformulate and consider alternatively in order to generalize it to general ZX-graphs. We denote this problem by **LSHerr[Opt]**.

4.1.1 First description

We will reformulate **LSHerr[Opt]** as well as formulate **LSHerr[Exist]**, similar to the previous one but in addition having space cost in its input.

To be a little wordy: the problem is a non-physical description of executing multi-target CNOTs on surface lattice. Though we will not cover the terms like "multi-target" nor "CNOT", "qubit", keeping in mind the picture of split and merge of qubit patches is always good for understanding the problem.

Our objective is to minimize the surface area of a lattice which consists of individual patches of the same size and placed in a regular way.

Starting with data description: on a surface lattice, some patches are respectively assigned an integer $q_{i,j}$. Furthermore, multiple patches can have the same integer, but $\{q_{i,j}\}_j$ forms a set for any fixed i (i.e. $q_{i,j} \neq q_{i,k}$ for $j \neq k$ and for any i). A *horizontal (vertical) neighbour* of a patch is defined as the next nonempty patch (i.e. assigned with an integer) to the right or left (up or down),so it is in fact a reciprocal relation between two patches.

Definition 4.1. A configuration of $\{q_{i,j}\}$ on surface lattice is *valid* if it satisfies the following conditions:

($LS1$) All $q_{i,j}$ have to be placed;

($LS2$) For each i, $q_{i,j}$ form a chain of horizontal neighbours;

($LS3$) For each j, $q_{i,j}$ form a chain of vertical neighbours.

One can imagine that this visualizes the split and merge, where for each j, the patches $\{q_{i,j}\}_j$ together with their intermediate patches formed the big patch that encodes the qubit j before splitting, and for each i, the patches $q_{i,j}$ with the same number together with their intermediate patches will form the big patch that encodes the qubit corresponding to the value $q_{i,j}$ after the merge. See Figure 4 for example.

We want to minimize the area of the lattice surface which allows a *valid* configuration of $q_{i,j}$. Of course, we have a *theoretical* optimality of area M_{opt} given by the total number of $q_{i,j}$ (multiplicities counted), i.e.

$$M_{opt} = \#\{(i,j) \,|\, q_{i,j} \text{ is given}\}.$$

Now we can formulate two decision problems as follows:

LSHerr[Opt] Given data $\{q_{i,j}\}$, can we find a *valid* configuration of $q_{i,j}$ on *some* surface lattice that reaches the theoretical optimality M_{opt}?

LSHerr[Exist] Given data $\{q_{i,j}\}$ and a surface lattice of size $R \times K$, can we find a *valid* configuration of $q_{i,j}$ on it?

It seems that the second problem is more reasonable, because due to the structure of the high-level circuit that needs to be compiled, it is not always possible to reach the theoretical optimum M_{opt} [HND17]. Therefore we will focus more on **LSHerr[Exist]**: given a *fixed size* space of patches, can we implement the given multi-target CNOTs in parallel?

4.1.2 Other representations of data

We give other representations for **LSHerr** problems so that we can generalize it to the case of many layers.

Representation as tuple of sets Given data $\{q_{i,j}\}$, let $Q_i = \{q_{i,j}\}_j$, and $Q = \bigcup_{i\in I} Q_i$ where I is the index set for i. Then we have a obvious bijection:

$$\{q_{i,j}\} \quad \longleftrightarrow \quad \text{tuple of finite sets } I,\, Q,\, \{Q_i \subset Q\}_{i\in I}.$$

We will frequently mix the usage of these two representations.

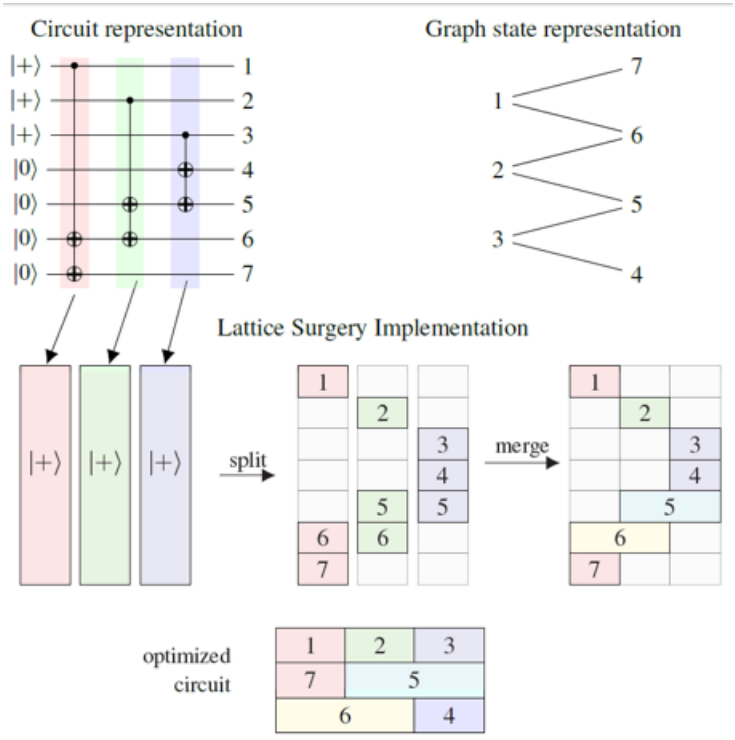


Figure 4: [HND17]. Here multi-target CNOTs are implemented using LS. During initialization three patches of surface code with N by 7N qubits are created, which are then split and merged to perform the computation. The faded boxes indicate ancillary qubits. An optimized version of this circuit is shown at the bottom where the placement of the patches ensures a minimal bounding box of the whole circuit area. This circuit can achieve the theoretical optimality.

Representation as *simple* ZX-graph Given data $(I, Q, \{Q_i\}_i)$ (as defined above), we construct ZX-graph $G(I, Q, \{Q_i\}_i)$ as follows (see Figure 5 for example):

1. choose $\#I$ inputs respectively connected to $i \in I$ put in the first columns;
2. put $q \in Q$ in the second column and connect them respectively to $\#Q$ outputs by Hadamard edge;
3. join i and q by a Hadamard edge if $q \in Q_i$.

During this construction, we treat I and Q as "mutually disjoint sets": for example, if x appears both in I and in Q, there will be two vertices in ZX-graph of x, but with different superscripts (Figure 5). We call the ZX-graphs that can be constructed from such representation *simple* ZX-graphs. Obviously, we have a one-to-one map (modulo superscripts)

$$(I, Q, \{Q_i\}_i) \longleftrightarrow G(I, Q, \{Q_i\}_i).$$

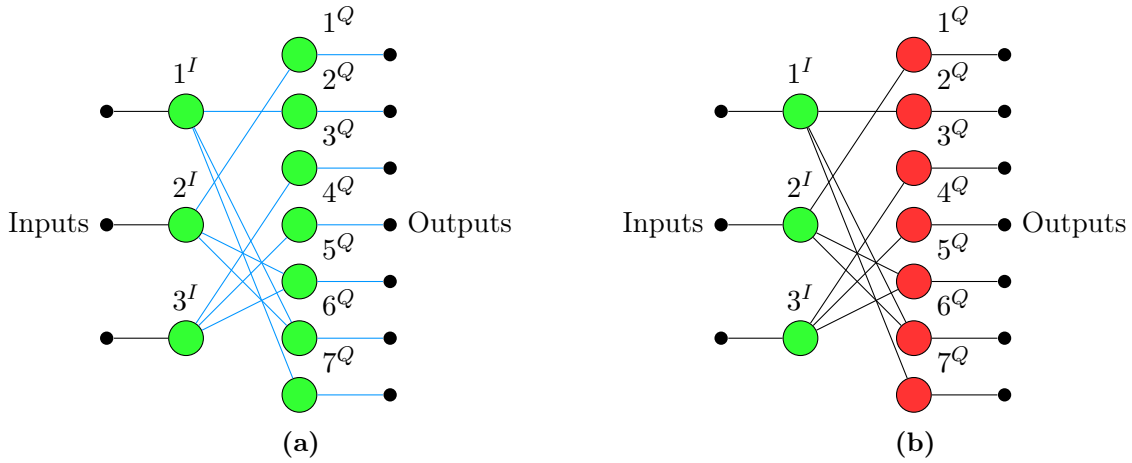


Figure 5: (a) Example of representation of data $Q_1 = \{2, 6, 7\}, Q_2 = \{1, 5, 6\}, Q_3 = \{3, 4, 5\}$ by ZX-graph. Since $I = \{1, 2, 3\}$ intersects $Q = \{1, \ldots, 7\}$, we use superscripts I, Q to differ them from each other. When we read the corresponding lattice surgery implementation (Figure 6), we can think it as (b) split along Z-operator then merge along X; or (a) split along Z-operator, perform Hadamard gates with "twist" (i.e. relabeling edges: $X \rightsquigarrow Z$ and $Z \rightsquigarrow X$ [LO18]), then merge along Z. The Hadamard gates of connected to outputs are *not* performed.

4.1.3 Re-interpretation

Given a *simple* ZX-graph, i.e. a ZX-graph of the form $G(I, Q, \{Q_i\}_i)$ that comes from data $(I, Q, \{Q_i\}_i)$ for **LSHerr**, how can we implement this ZX-graph using lattice surgery, minimizing the space cost while doing splits in parallel and the same for merges?

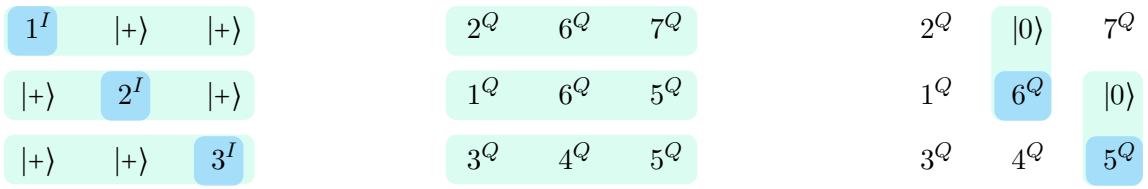
For the purpose of parallelizing splits, we have to put the involved patches in some sort of chains on the surface lattice. The simplest way is to put them all in the same, for example, row. Since a Hadamard gate has to be performed before a split patch is merged with others, we do it in software as described in [LO18] by exchanging Z-logic qubit and X-logical qubit. As a result, if now we do merges, they have to be done along the direction of columns and the patches to merge together form a vertical chain. The best situation is that, such chains do not intersect each other, or we can say such chains consists of "horizontal/vertical neighbours". This is exactly the content of $(LS2), (LS3)$.

Therefore, from now on, in our implementation, we fix one split direction (horizontal), and one merge direction (vertical), so that the splits and merges are realized for a chain of patches in the respective direction. Thus, on a surface lattice, we always split a patch to some patches in the same row, and the patches we merge are always in the same column.

Under this convention, for a *simple* ZX-graph $G(I, Q, \{Q_i\}_i)$ with $I \bigcap Q = \varnothing$, a valid configuration for $(I, Q, \{Q_i\}_i)$ gives naturally a split-merge procedure, illustrated in Figure 6, described as follows:

1. For $i \in I$, put the patch of input for i in the same row $\rho(i)$ as $\{q_{i,j}\}_j$;
2. Split i to patches of $\{q_{i,j}\}_j$ in the same row, putting $q_{i,j}$ at row $\rho(i)$, column $\kappa(q)$ if $q_{i,j} = q \in Q$;

The function ρ is defined on I, and κ is defined on Q. Therefore, no matter in which Q_i belongs q (perhaps not unique), the corresponding $q_{i,j} = q$ is always put in the column $\kappa(q)$.



(a) prepare I **(b)** split I to Q **(c)** merge Q

Figure 6: An example of so-called natural split-merge implementation of ZX-graph in Figure 5. First, we get a *valid* configuration of $Q_i, i \in I$ on the 3×3 surface lattice as in (b), from which we get the row-function $\rho : 1^I \mapsto 1, 2^I \mapsto 2, 3^I \mapsto 3$ and the column function $\kappa : 1^Q \mapsto 1, 2^Q \mapsto 1, 3^Q \mapsto 1, 4^Q \mapsto 2, 5^Q \mapsto 3, 6^Q \mapsto 2, 7^Q \mapsto 3$. Then we (a) prepare the patch for $i \in I$ according to ρ; (b) split i to Q_i, putting $q \in Q_i$ at row $\rho(i)$ and column $\kappa(q)$; (c) merge the patches with the same number while keeping them respectively in the same column as in (b).

We call such functions ρ and κ respectively *row-function* and *column-function*, and by a *natural implementation*, we mean such a split-merge implementation of a simple ZX-graph obtained from a valid configuration for $(I, Q, \{Q_i\}_i)$. We have therefore an one-to-one correspondence:

$$\text{valid configurations of } (I, Q, \{Q_i\}_i) \longleftrightarrow \text{ natural implementations of } G(I, Q, \{Q_i\}_i).$$

We consider the two following statements as equivalent (both will be called **LSHerr[Exist]**):

(a) $(I, Q, \{Q_i\}_i)$ admits a valid configuration on $R \times K$ surface lattice;

(b) $G(I, Q, \{Q_i\}_i)$ has a natural implementation on $R \times K$ surface lattice.

Note that a valid configuration and the corresponding natural implementation share the same ρ and κ.

When we read a lattice surgery implementation like Figure 6, we can think it as in Figure 5(b) split along Z-operator then merge along X; or as in Figure 5(a) split along Z-operator, perform Hadamard gates with "twist" (i.e. relabeling edges: $X \rightsquigarrow Z$ and $Z \rightsquigarrow X$ [LO18]), then merge along the new Z-edge. The Hadamard gates of connected to outputs are *not* performed.

4.2 Problem description: split-merge several times

Conventions:

(1) Here we only consider the ZX-graph whose vertices other than inputs and outputs are *green*, and are connected to each other only by *Hadamard* edges (otherwise we can merge together the two endpoints of such an edge by axiom (S) of ZX-calculus [JPV19]).

(2) We suppose the ZX-graph we are going to consider is equipped with a Pauli-Fusion flow whose time-ordering is given by the column number of vertices in the planar ZX-graph (increasing from left to right).

(3) We are *not* interested in the angle of Z-rotations in this section, even though such rotations may need additional space cost. The same for additional Hadamard gates that appear between two layers of a simple decomposition (defined below).

Given such a ZX-graph, we can break it down to several small components, each of them being a *simple* ZX-graph, i.e. in the form $G(I, Q, \{Q_i\}_i)$. We will first consider how to join different pieces, propose continuity conditions, then describe the exact procedure of this break-down or *simple decomposition*.

4.2.1 Continuity condition

If we do split-merge several times consecutively, we are not performing operations on all the patches at each time step. So, some patches among outputs of the previous split-merge, should be treated as occupied and inactive, so it seems that we have to put *more constraints* on surface lattice for the next split-merge:

(1) a chain of horizontal neighbours cannot pass through inactive patches;

(2) if an inactive patch is already labelled by $q \in Q$, then all patches corresponding to q split from I must be placed in the same column as the inactive patch;

(3) during the merge, merge all patches, including inactive patches.

Simply speaking, inactive patches "form an obstruction" on surface lattice. It seems natural to extend **LSHerr[Exist]** to such case, but we can do it in another way. In fact, we can treat inactive patches as "active", i.e. participating in the split-merge: denote by $Q^* \subset Q$ the subset consisting of labels that already exist on the surface lattice (these will participate in merges) and by D the set consisting of "dummy" labels (these will not participate in split nor merge, and will be treated as "split and then merge to themselves"). We modify the conventional data $(I, Q, \{Q_i\}_i)$ for **LSHerr[Exist]** as follows:

$$I \rightsquigarrow I \cup Q^* \cup D, \qquad Q \rightsquigarrow Q \cup D,$$

$$\{Q_i\}_{i\in I} \rightsquigarrow \{Q_i\}_{i\in I} \cup \{\{q^*\} : q^* \in Q^*\} \cup \{\{d\} : d \in D\}.$$

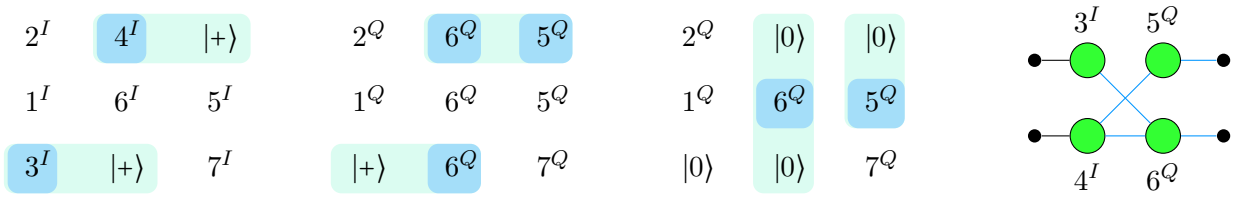
We will denote the new data by $\mu(I, Q, \{Q_i\}_i; Q^*, D)$ and call it the *modified form*.

Now it is time to "connect" two split-merge processes. However, one disadvantage of this modified form is that, we have lost information of the original configuration in exchange for flexibility of patch configuration. For example, continuing our split-merge in Figure 6, we do split-merge with data $I = \{3, 4\}, Q = \{5, 6\}, Q_3 = \{6\}, Q_4 = \{5, 6\}$. A natural implementation with the modified data with $Q^* = \{5, 6\}, D = \{1, 2, 7\}$ is shown in Figure 7, we see that it is impossible to modify the merge process of Figure 6 and the column arrangement of Figure 7(a) to make them coherent (ignoring the superscripts). In fact, one checks easily that starting from Figure 6(b), we can never make it on the same 3×3 surface lattice even by rearranging the split-merge in Figure 7.

To deal with this loss of information, we introduce a concept of "continuity condition": for two consecutive split-merge operations with data $(I^{(1)}, Q^{(1)}, \{Q_i^{(1)}\}_i), (I^{(2)}, Q^{(2)}, \{Q_i^{(2)}\}_i)$ and row-functions and column-functions $\rho^{(1)}, \kappa^{(1)}, \rho^{(2)}, \kappa^{(2)}$, the *continuity condition* means:

Continuity of sets: $(I^{(2)}, Q^{(2)}, \{Q_i^{(2)}\}_i)$ is in the modified form as described above, and $Q^{(1)} = I^{(2)}$.

Continuity of positions: it is feasible to do "merge $Q^{(1)}$" step by placing $q \in Q^{(1)}$ in row $\rho^{(2)}(q)$ *in a valid way*, i.e. without arousing any intersection of vertical chains with other $q \in Q^{(1)}$.

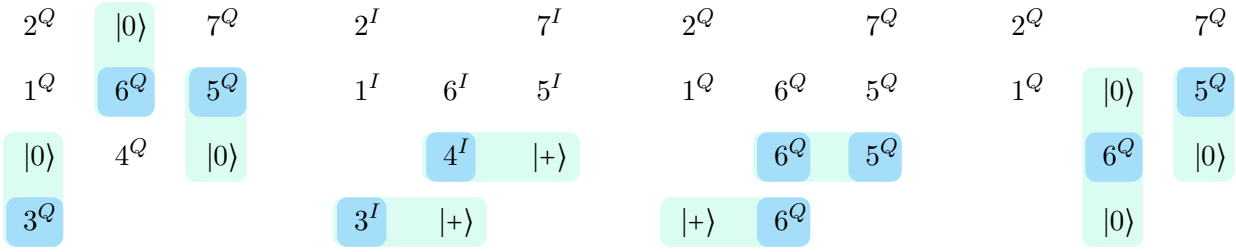


(a) prepare I **(b)** split I to Q **(c)** merge Q **(d)** ZX-graph

Figure 7: An example of natural split-merge implementation of ZX-graph (d) as continuation of that of Figure 6, with $I = \{3, 4\}, Q = \{5, 6\}, Q^* = \{5, 6\}$ and inactive patches $D = \{1, 2, 7\}$. From (a) we get the row-function $\rho : 1^I \mapsto 2, 2^I \mapsto 1, 3^I \mapsto 3, 4^I \mapsto 1, 5^I \mapsto 2, 6^I \mapsto 2, 7^I \mapsto 3$. From (b) or (c) we get the column function $\kappa : 1^Q \mapsto 1, 2^Q \mapsto 1, 5^Q \mapsto 3, 6^Q \mapsto 2, 7^Q \mapsto 3$.

More on continuity: remember that, in the "merge $Q^{(1)}$" step, only the columns are fixed by $\kappa^{(1)}$, leaving space for choice of rows. On the contrary, In the "prepare $I^{(2)}$" step, it is the rows that are fixed by $\rho^{(2)}$. If the continuity of positions holds, one will find that the configuration after "merge $Q^{(1)}$" is exactly the same as that after "prepare $I^{(2)}$", therefore two split-merges can be done continuously and coherently. In this case, we say the sequence of the latest configurations $C^{(1)}, C^{(2)}$ for two sets of data is *continuous*, or we can say it is *a continuous version* of the original sequence. See Figure 8 for a positive example.

However, we point out a shortness of this: while connecting two implementations, we need to perform at least the Hadamard gates connected to outputs in Figure 5(a), which rotate the patches and put them in the right split direction (i.e. put split horizontally) for the next implementation (see the caption of Figure 5), all of which demand additional space on the surface lattice. In our problem formulation below, we simply ignore this consideration of additional space cost.



(a) merge Q, cf. Figure 6(c) **(b)** prepare I **(c)** split I to Q **(d)** merge Q

Figure 8: An example of natural split-merge implementation of ZX-graph Figure 7(d) *on* 4×3 *surface lattice*, satisfying the *continuity condition* with Figure 6. More precisely, on 4×3 surface lattice, we can modify Figure 6(c) as (a) here, $\rho^{(1)}, \kappa^{(1)}$ staying the same as in Figure 6; for implementation of Figure 7(d), we do (b)-(d) here rather than in Figure 7(a)-(c), thus with $\rho^{(2)} : 1^I \mapsto 2, 2^I \mapsto 1, 3^I \mapsto 3, 4^I \mapsto 3, 5^I \mapsto 4, 6^I \mapsto 2, 7^I \mapsto 1$ and $\kappa^{(2)} : 1^Q \mapsto 1, 2^Q \mapsto 1, 5^Q \mapsto 3, 6^Q \mapsto 2, 7^Q \mapsto 3$. As a result (a) and (b) has the same configuration of numbers (regardless of superscripts), which means continuity condition holds for our choice of $\rho^{(1)}, \kappa^{(1)}, \rho^{(2)}, \kappa^{(2)}$ here.

4.2.2 Preparation and measurement

For further discussion, it is worth taking into consideration the case with qubit preparation and measurement. Suppose we have to do two split-merges $(I^{(t)}, Q^{(t)}, \{Q_i^{(t)}\}_i), t = 1, 2$ one by one. For each $t = 1, 2$, we have to prepare $I^{p(t)} \subset I^{(t)}$ before split-merge and measure out $Q^{m(t)} \subset Q^{(t)}$ after split-merge, respectively.

Therefore, we are just sandwiching the split-merge by preparation and measurement, so the definitions for row-function, column-function and $Q_i^{(t)}$ stay unchanged.

However, the continuity conditions are updated naturally and reasonably as follows:

Continuity of sets: $(I^{(2)}, Q^{(2)}, \{Q_i^{(2)}\}_i)$ is in the modified form, and $Q^{(1)} \backslash Q^{m(1)} = I^{(2)} \backslash I^{p(2)}$.

Continuity of positions: it is feasible do "merge $Q^{(1)}$" step by placing $q \in Q^{(1)}$ in row $\rho^{(2)}(q)$ *in a valid way* for $q \in Q^{(1)} \backslash Q^{m(1)}$, i.e. without arousing any intersection of vertical chains of other q.

More on continuity: f the continuity conditions hold, one will find the configuration after "merge $Q^{(1)}$" and measurement is exactly the same as that after "prepare $I^{(2)}$", therefore two split-merges can be done continuously and coherently. In this case, we say the sequence of the latest configurations $C^{(1)}, C^{(2)}$ for two sets of data is *continuous*, or we can say it is *a continuous version* of the original sequence.

For a possibly longer sequence of configurations of simple ZX-graphs, we can define the corresponding continuity conditions respectively as the conjunction of those of sub-sequences of two consecutive configurations.

4.2.3 Simple decomposition (or break-down) of ZX-graph

We first give a general description, then give an example of two *layers* (to be defined below). Our basic idea of break-down is to keep those patches resulting from splits but not yet merged, even if they might not participate in the next merge. These are patches with which we do nothing for the moment, i.e. *inactive* or "dummy" patches as we called them in previous paragraphs.

Notation

1. V : set of vertices of the ZX-graph;
2. E : set of edges;
3. $t(\cdot)$: time-ordering given by columns in ZX-graph, suppose $\text{Image}(t) = \{1, \ldots, T+1\}$;
4. $\text{pred}(v) = \{u \in V : uv \in E,\, t(u) < t(v)\}$: function that gives predecessors of a vertex v;
5. $\text{post}(v) = \{u \in V : uv \in E,\, t(u) > t(v)\}$: function that gives successors of a vertex v;
6. $V_t = \{v \in V : t(v) = t\}$: set of vertices at time t;
7. $V_t^{prepare} = \{v \in V_t : \text{pred}(v) = \varnothing\}$: set of vertices to prepare at time t;
8. $V_t^{measure} = \{v \in V_t : \text{post}(v) = \varnothing\}$: set of vertices to measure at time t;
9. $\text{Post}_t = \bigcup_{v \in V_t} \text{post}(v)$: resulting patches of splits at time t; we define this because sometimes we have a vertex splitting to another vertex much later in time.

We decompose a ZX-graph as a sequence of *simple* ZX-graphs. Each of them is called a *layer*. There are in total T layers, parametrized by $\{(t, t+1)\}_t$ or simply by t. For the layer $(t, t+1)$, the data in the modified form (i.e. concerning all patches present on the surface lattice) $\left(I^{(t)}, Q^{(t)}, \{Q_i^{(t)}\}_{i \in I^{(t)}}\right)$ and the

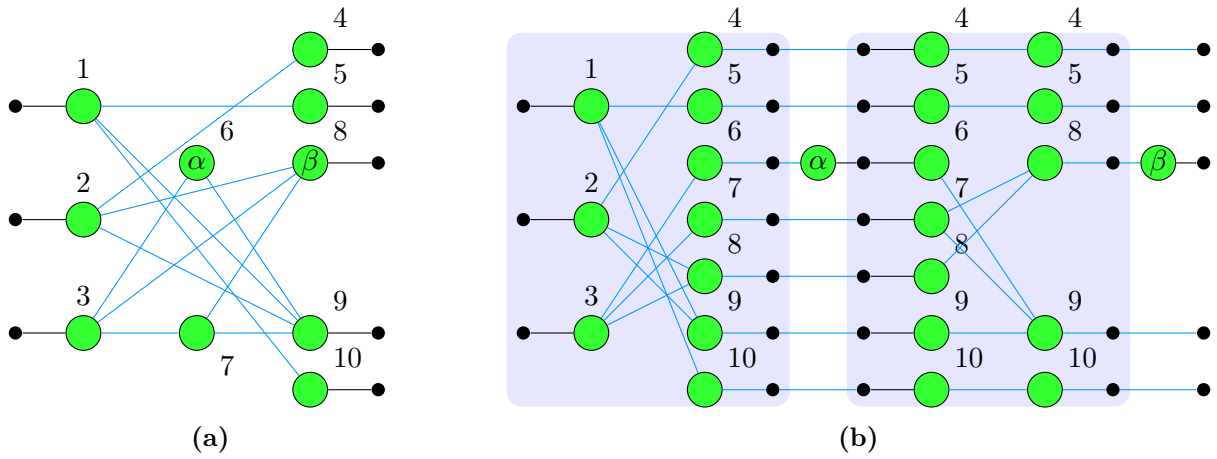


Figure 9: An example of simple decomposition of ZX-graph. We have decomposed (a) as (b), where shadowed boxes represent the simple ZX-graphs. There are phase gates between them but no intermediate preparation or measurement.

extra data $I^{p(t)}, Q^{m(t)}$ are given recursively by:

$$\begin{aligned}
I^{(t)} &= \left(Q^{(t-1)} \backslash V_t^{measure}\right) \bigcup V_t^{prepare};\\
I^{p(t)} &= V_t^{prepare};\\
Q^{(t)} &= \left(I^{(t)} \backslash V_t\right) \bigcup \left(V_{t+1} \backslash V_{t+1}^{prepare}\right) \bigcup \mathrm{Post}_t;\\
Q^{m(t)} &= V_{t+1}^{measure};\\
Q_i^{(t)} &= \begin{cases} \mathrm{post}(v) & v \in V_t \\ \{i\} & \text{otherwise.} \end{cases}
\end{aligned}$$

They are defined to be $\varnothing$ for $t \ll 0$. One verifies easily that the continuity of sets (in the sense of 4.2.2) are satisfied.

Definition 4.2. The above decomposition into simple ZX-graphs is called the *simple decomposition* of this ZX-graph.

Now we can implementing a ZX-graph by a sequence of implementations of simple ZX-graph of each layer, except that while connecting these implementations, we need to perform one-qubit gates, including rotations that put the square patches in the right split direction (i.e. put split horizontally) for the next implementation, all of which demand additional space on the surface lattice. In our problem formulation below, we simply ignore this consideration of additional space cost. Nevertheless, we show these one-qubit gates in the example Figure 9.

4.2.4 Problem description

Definition 4.3. For a ZX-graph, a *simple split-merge implementation* is a *continuous* sequence of configurations $\{C^{(t)}\}_t$ on some surface lattice, where $C^{(t)}$ is a valid configuration of layer t of the simple decomposition of this ZX-graph.

It is heuristically natural and reasonable to define the theoretical optimum M_{opt} of a ZX-graph as the maximum of $M_{opt}^{(t)}$, theoretical optimum for layer t in the simple decomposition.

We have the following problems:

LSSimple[Opt] Given a ZX-graph, can we find a *simple split-merge implementation* that reaches its theoretical optimality M_{opt}?

LSSimple[Exist] Given a ZX-graph and a surface lattice of size $R \times K$, can we find a *simple split-merge implementation* on it?

4.3 Complexity

In [HND17], the authors proved the NP-completeness by a clever reduction to the NP-complete **3-partition** problem [GJ90]:

> Given a set of non-negative integers $A = \{a_i\}_{1 \le i \le 3s}$ and another non-negative integer L such that $\frac{L}{4} < a_i < \frac{L}{2}$ for all i and $\sum_i a_i = sL$, decide whether A can be partitioned into s subsets $A_1, \ldots, A_s$ such that $\sum_{i \in A_j} a_i = L$ for all j.

4.3.1 Reduction from 3-partition to LSHerr[Opt]

Given data A as above for **3-partition**, we have to define data $f(A)$ for **LSHerr[Opt]** (of course, $f(A)$ is also part of data for **LSHerr[Exist]**).

First, take sL consecutive integers $s+2+L+1, \ldots, s+2+L+sL$, which are different from $1, \ldots, s+2+L$; then divide them into $3s$ groups $\{q_{i,j}\}_j$, $1 \le j \le 3s$, whose cardinals are given by A, i.e. we have $\#\{q_{i,j}\}_j = a_i$ for all i. The choice of division is arbitrary, but we just cut them in sequence just to make it simple. Now we are ready to define $f(A)$ as

$$\{1, s+2, s+3, \ldots, s+2+L\},$$

$$\{2, s+2\},\ \{3, s+2\},\ \ldots,\ \{s+1, s+2\},$$

$$\{q_{i,j}\}_j \text{ for all } i \text{ as defined above.}$$

Lemma 4.1. [HND17] *f gives a polynomial-time reduction from* **3-partition** *to* **LSHerr[Opt]** .

Sketch of proof. By the construction above, f is obviously a polynomial-time operation.

For the "reduction" part, we first calculate the theoretical optimum

$$M_{opt} = (2+L) + s \cdot 2 + \sum_i \#\{q_{i,j}\}_j = (s+1)(L+2).$$

On the other hand, due to the repetitive occurrences of the number $s+2$ and the length of $\{1, s+2, s+3, \ldots, s+2+L\}$, the surface lattice that supports a theoretically optimal configuration must have at least $s+1$ rows and $L+2$ columns. Therefore, an optimal surface lattice must have the size $(s+1) \times (L+2)$.

Furthermore, on such a surface lattice (Figure 10), the set $\{1, s+2, s+3, \ldots, s+2+L\}$ must occupy one entire row, and $s+2$ must fill out one entire column. By putting the set $\{1, s+2, s+3, \ldots, s+2+L\}$ on the top row and the sets $\{2, s+2\}$, $\{3, s+2\}$, …, $\{s+1, s+2\}$ along the left side (it is always better to do so in order to fit the other sets into the rest space), one gets an "almost" bijection between all possible 3-partitions of A and configurations of $\{q_{i,j}\}_j$ into the rest space (modulo permutation of rows and permutation of patches in each row). This means that f gives indeed a reduction. □

Lemma 4.2. *If $f(A) \in$ **LSHerr[Opt]** and C is a valid configuration that reaches the theorectical optimality, then we can obtain a 3-partition of A in polynomial time (polynomial in terms of A, $f(A)$ or the size of surface lattice for configuration C).* □

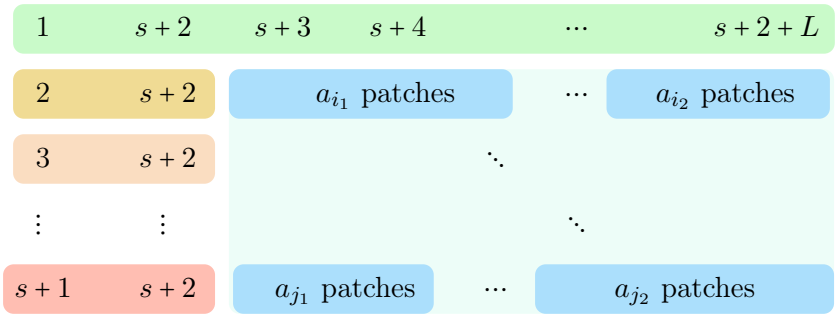


Figure 10: Reduction from **3-partition** to **LSHerr[Opt]**. The set $\{1, s+2, s+3, \ldots, s+2+L\}$ occupies the first row while the sets $\{2, s+2\}$, $\{3, s+2\}$, …, $\{s+1, s+2\}$ are put along the left side. It is always better to do so in order to fit the other sets into the rest space, i.e. in order to find a solution for **LSHerr[Opt]** with data $f(A)$.

4.3.2 Reduction from 3-partition to LSHerr[Exist]

Now we turn to the reduction from **3-partition** to **LSHerr[Exist]**, a problem in which we are more interested. Given data A as above for **3-partition**, we use the data $f(A)$ defined as in the reduction above. We define the desired reduction to be

$$f^{alt} : A \longmapsto (f(A),\, Q = s+1,\, K = L+2)$$

(remark on notation: *alt* for "altered").

Lemma 4.3. *f^{alt} gives a polynomial-time reduction from* **3-partition** *to* **LSHerr[Exist]**.

Proof. By construction, f^{alt} is obviously a polynomial-time operation.

Observing that we have in fact proved in lemma 4.1 that $f(A)$ admits a valid configuration on a $R \times K = (s+1) \times (L+2)$ surface lattice if and only if $f(A)$ admits a valid configuration with the theoretical optimal space cost $(s+1)(L+2)$ (if and only if, still by lemma 4.1, A admits a 3-partition). So indeed, f^{alt} gives a reduction. □

Lemma 4.4. *If $f^{alt}(A) \in$* **LSHerr[Exist]** *and C is a valid configuration that reaches the theoretical optimality, then we can obtain a 3-partition of A in polynomial time (polynomial in terms of A, $f^{alt}(A)$).* □

4.3.3 Reduction from LSHerr to LSSimple

From our definition of **LSSimple** and the re-interpretation of **LSHerr** in terms of simple ZX-graph, we find a natural embedding from **LSHerr** to **LSSimple** which is the identity map. This is of course a polynomial reduction from the first to the latter.

4.3.4 NP-completeness

Proposition 4.5. LSHerr[Opt,Exist] *and* **LSSimple[Opt,Exist]** *are NP-complete.*

Proof. They are NP-hard because all of them admit a polynomial reduction to the NP-complete problem **3-partition**. They are NP because in fact we can take valid (or not in the negative case) configurations as polynomial size certificates, or we can give a non-deterministic polynomial algorithm. □

4.3.5 An upper bound

What if we add additional constraints? For example, we wonder whether there exist two "good" functions $R(G)$ and $K(G)$ of ZX-graph G such that the following problem is "easier":

LSSimple[$R(\cdot)K(\cdot)$] Given a ZX-graph G, can we find a *simple split-merge implementation* of it on the surface lattice of size $R(G) \times K(G)$?

I have no idea about this question, but at least, we have a more or less trivial upper bound for $R(G)$ and $K(G)$ such that **LSSimple[$R(\cdot)K(\cdot)$]** is always true:

$$R(G) = \max_t \#I^{(t)}, \quad K(G) = \max_t \#Q^{(t)}.$$

In this case, we can choose row-functions $\rho^{(t)} : I^{(t)} \longrightarrow \{1, \ldots, R\}$ and column-functions $\kappa^{(t)} : Q^{(t)} \longrightarrow \{1, \ldots, K\}$ to be *injective*, so that a continuous sequence of valid configurations of each layer is always possible.

5 Summary

Given a ZX-graph, we can:

1. simplify it using PyZX;
2. find a Pauli-Fusion flow in polynomial time if one exists; otherwise decompose (or transform) the ZX-graph such that Pauli-Fusion flow exists for all parts of it (or for the entire graph);
3. do error correction as in Figure 2 in [dBDHP19];
4. implement it by lattice surgery, or more precisely, simple split-merge implementation (section 4.2.4);
5. find possibilities of optimization.

Problems still exist:

1. Pauli-Fusion flow may not exist after simplification, even for a circuit-like ZX-graph; so usually we have to do *local* simplification;
2. Uncertainty exists in PF-FLOW FINDING algorithm, and we do not know which choice is better;
3. It is hard to determine the optimal space cost for a simple split-merge implementation.
4. We have ignored the extra Hadamard gates between layers and the extra space demanded for one-qubit phase gates shown in Figure 9(b).

Index

References

[CDD+19] Alexander Cowtan, Silas Dilkes, Ross Duncan, Will Simmons, and Seyon Sivarajah. Phase gadget synthesis for shallow circuits. *arXiv preprint arXiv:1906.01734*, 2019.

[dBDHP19] Niel de Beaudrap, Ross Duncan, Dominic Horsman, and Simon Perdrix. Pauli fusion: a computational model to realise quantum transformations from zx terms. *arXiv preprint arXiv:1904.12817*, 2019.

[dBH17] Niel de Beaudrap and Dominic Horsman. The zx calculus is a language for surface code lattice surgery. *arXiv preprint arXiv:1704.08670*, 2017.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[HFDVM12] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[HND17] Daniel Herr, Franco Nori, and Simon J Devitt. Optimization of lattice surgery is np-hard. *npj Quantum Information*, 3(1):35, 2017.

[JPV19] Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Completeness of the zx-calculus. *arXiv preprint arXiv:1903.06035*, 2019.

[KvdW19] Aleks Kissinger and John van de Wetering. Pyzx: Large scale automated diagrammatic reasoning. *arXiv preprint arXiv:1904.04735*, 2019.

[Lit18] Daniel Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *arXiv preprint arXiv:1808.02892*, 2018.

[LO18] Daniel Litinski and Felix von Oppen. Lattice surgery with a twist: Simplifying clifford gates of surface codes. *Quantum*, 2, 2018.