表面码解码算法

林雨轩

April 2025

1 MWPM 算法

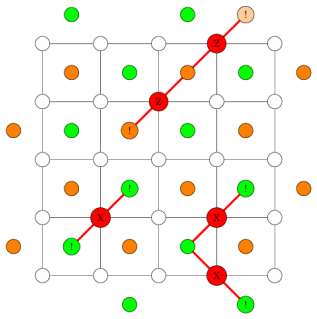
MWPM 的全称是 Minimum Weight Perfect Matching（最小权重完美匹配），其核心思想是将解码问题转化为图论中的最小权重完美匹配问题。我们首先给出这个图论问题的定义：

考虑点的集合 V，边的集合 E，完美匹配指的是选取 E 中的某些元素，使得所有 V 中的元素恰好都只被一条边关联。最小权重指的是使得所选取的边的权重之和最小。

现在建立解码问题与该图论问题之间的关联：将所有的稳定子视作点，所有的物理比特视作边。解码问题是给出一些报错的稳定子，期待得到最可能的犯错的物理比特。最可能指的是，存在多种错误使得稳定子报错状态相同，所有这些错误中错误算符数最少的概率最大。在表面码中，X 稳定子用以对抗 Z 错误链；Z 稳定子用以对抗 X 错误链（相互对易的算符不影响测量结果），因此我们总是可以将两者分开处理。

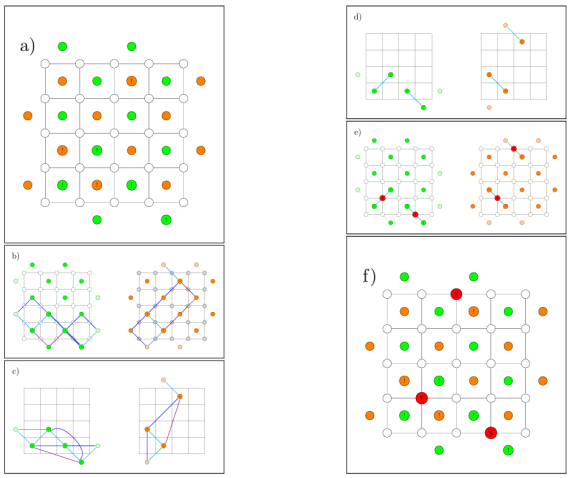
于是问题变成了给定报错的稳定子集合，视作点集 V，如何得到总长度最短的错误链集合，视作边集 E。至此已经完成了最小权重部分的对应，下面来看完美匹配。一个报错稳定子，必然有与之相连的奇数（2k+1）条错误链，其中的 2k 条总是可以由不经过该点权重为 2 的长边来取代，因此每个报错稳定子必须关联且仅关联 1 个错误链，这就对应了完美匹配。

最后我们需要小心地对待边界情况。因为按照上面的假设，每条错误链的端点总是报错的稳定子，然而实际上边界上可能并不存在这样的稳定子（如下图右上方浅橙色点所示），因此需要引入虚拟稳定子，以保证每条错误链的两端都是报错的稳定子。

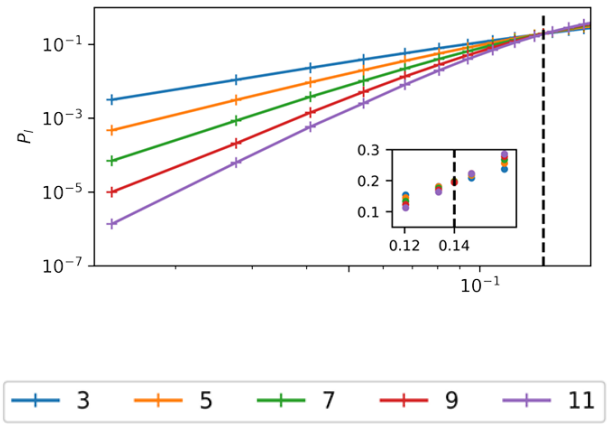


总结来说，我们进行如下操作将问题转化成 MWPM 问题：将所有报错稳定子纳入点集 V，并将

他们两两之间的边计算权重后纳入边集 E。然后将与主结构距离最近的几个虚拟节点及最近的边添加进入点集 V 和边集 E 中，后续的匹配过程能够使用边集 E 中的所有元素，但无需对点集 V 中的所有虚拟节点实现匹配。在完成了转移步骤之后，应用图论中的 MWPM 算法即可给出对应的错误链。一个具体的算法流程如下：



图论中的 MWPM 问题可以使用 Blossom 算法解决，节点数为 N 的情况下，原始算法复杂度为 $O(N^3 logN)$。现有软件包 Stim，pymatching 中使用的稀疏 Blossom 算符复杂度为 $O(N^{1.32})$，由此可以得到比较高的盈亏点和阈值，大概都在 10% 左右，（盈亏点指的是增大码距可以降低逻辑错误率的点，阈值指的是能够使得逻辑错误率低于物理错误率的点）具体如下所示：



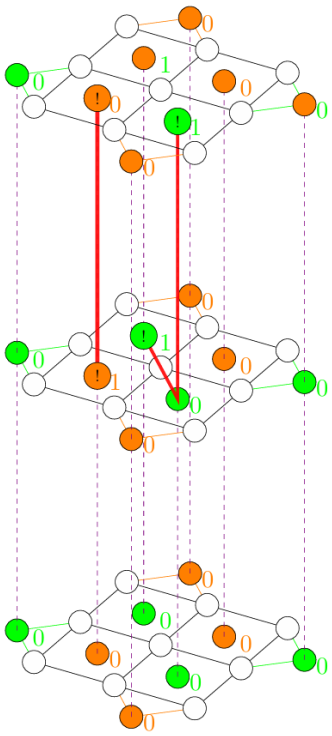
不过这张图的码距加的不大，并且也不知道是否故意没给出横坐标，导致难以观察出显著降低逻辑错误率时所需的码距和物理错误率具体是多少，或许应该实际操作一番看看性能究竟如何。

可能的问题在于两个点：其一是当噪声是偏置噪声，即发生 Z 错误的概率远高于 X 时阈值将会降低。不过降低的幅度也不大，看起来属于可接受范围，而且发生某种错误的概率提高无非是相当于把这个图像左右平移了一些：

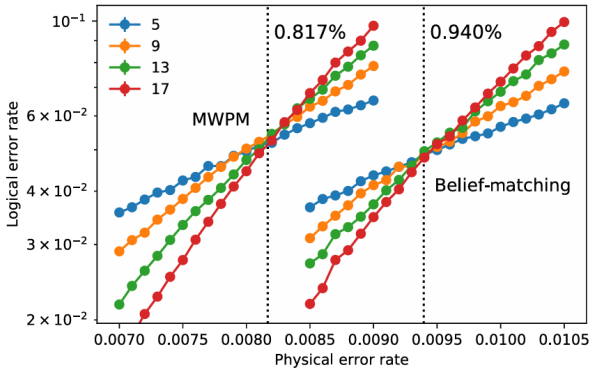
η	p_{th}
1/2	0.140
1	0.138
10	0.098
100	0.095
1000	0.088

更加严重的问题在于稳定子测量时产生的错误。如果说稳定子测量不是完美的，一般而言认为需要 d 轮次的结果综合判定，具体方式如下：

构建时空图，将不同轮次的解码图排列在不同的平面上，最底层为每个稳定子均不报错的辅助状态。在每一轮新的稳定子测量中，如果结果与上一次不同，则纳入点集 V 中，在构建边集 E 时，需要注意的是有两类型的边，水平面内的边对应物理比特上的错误，竖直平面内的边对应测量时犯的错误，计算权重时要对所有边进行计算，但是输出错误链时只考虑水平边。具体图示如下：



主流观点认为需要 d 轮次的测量才能继续保持码距特性，不过对此暂未找到具体说明。可以预见的是，随着轮数的提升，真正有用的水平面内的解码效果将会被大幅冲淡，目前看到的数据如下所示：



盈亏点直接掉到了 1%，更为夸张的是，这个纠错竟然越纠越错，图中的逻辑错误率比物理错误率还高，这似乎直接冲击了表面码的可行性，目前来看，或许这是一个非常需要解决的问题。如果硬要说作用的话，或许在物理错误率本身很低的情况下还是能够进行纠错的。

实际上对于稳定子测量过程中的错误在之前一直是缺乏考虑的，尽管编码电路是容错的，但是测量稳定子的线路看起来并不是容错的，这样一来或许真正导致错误的反而不是量子算法线路中的的错误，而是用以提取稳定子信息时产生的错误。而且这似乎是一个没法通过级联解决的问题，因为纠错依赖于我们能够获取一些有用的信息，即使我们能够完全充分地利用这些信息，但是对于真实情况预测的正确率仍然被提取出有效信息这一步的正确率所主导，看起来像是木桶原理那样，这是否意味着量子纠错本质上是不可行的呢?

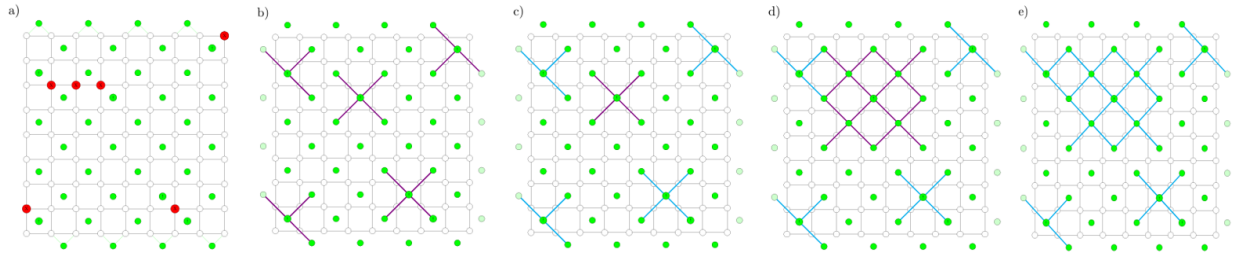
2 UF 算法

分析 MWPM 算法的复杂度，不难看出实际上有很多步骤是不必要的，因为一些错误相距非常远，低权重的错误必然分布在其周围，没有必要一一列举那些连接到这些遥远地带的边（错误链）。因此我们可以将错误分块处理，由每个报错的稳定子开始生长，当每个簇长到一定条件（簇的内部能找到错误链满足端点奇偶性条件）时，停止簇的生长，这样一来每个错误链将被局限在簇的内部，因而只需单独分析每个簇便能找到对应错误链。从原理上看，这种带有先验性猜测的做法将极大缩短程序时间，但并不能保证给出最可能得错误组合。

整个过程分为两步，我们首先介绍簇生长过程的条件。每个簇的奇偶性定义为簇内部报错稳定子数量的奇偶性，每一次生长向外探索一格，同时考虑所有的虚拟稳定子，当满足以下条件时簇停止生长：

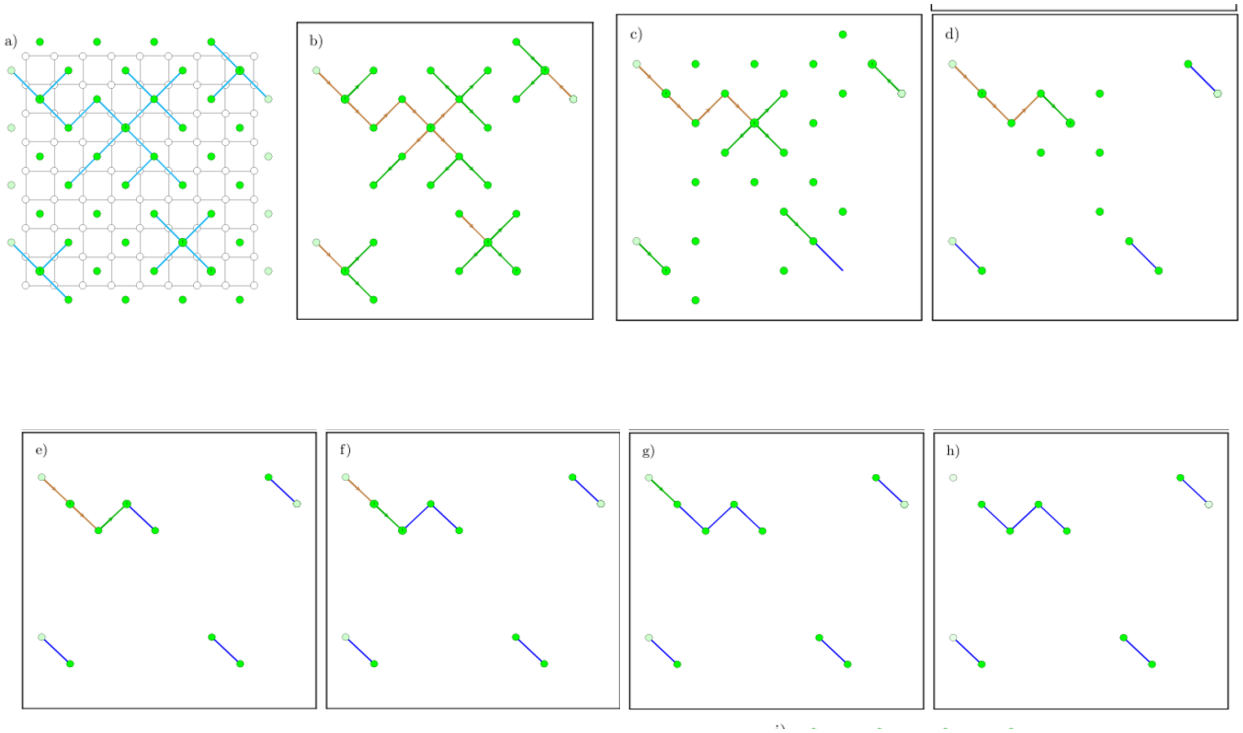
1. 簇的奇偶性变为偶。
2. 簇接触到虚拟稳定子。
3. 簇接触到接触了虚拟稳定子而被冻结的簇。

除此之外，簇将持续生长（碰到其他已被冻结的簇时合并为一个大簇，再依据以上条件判断是否冻结）。我们以如下情形展示簇的具体生长过程，注意由于稳定子码的特性，我们只需考虑仅由 X 错误引发的情况：



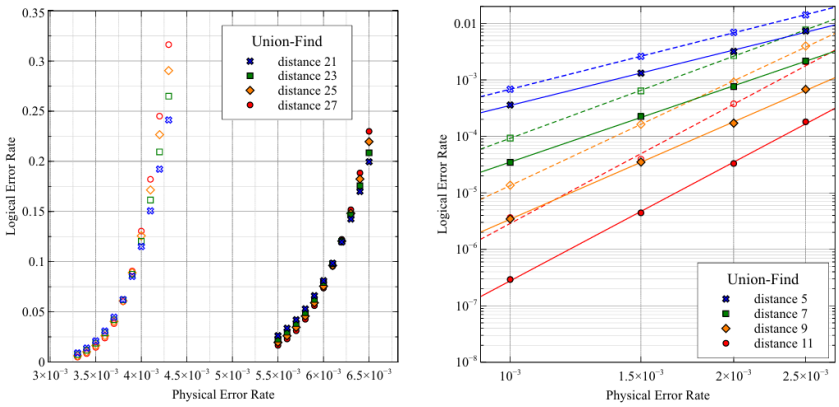
现在考虑从每个簇内找出对应的错误链。首先把每个簇变成生成树，即去掉其中的环路。然后选定叶子节点和根，叶子节点是那些度为 1 的节点，显然虚拟稳定子必然是叶子节点。如果该簇结构中存在虚拟稳定子，则任选其中之一作为根，其余作为叶子节点。如果不存在虚拟稳定子，则任选一个叶子节点作为根。

考虑从所有叶子节点向根的删除操作。报错稳定子节点记为 1，其他记为 0，如果说当前叶子节点为 0，则直接删除对应节点和边，并修改新的生成树；如果当前叶子节点为 1，则标记与之相连的边为错误链，删除该叶子节点，并翻转标记边所连接的另一节点的记号。按照以上规则一直进行删除直至根节点。注意到虚拟节点既可以报错也可以不报错，因此当最后剩余的根为虚拟稳定子时根据具体情况发挥作用。上图中生成的簇可以按照如下方式进行对应删除：



从算法流程上我们不难看出这是一个 $O(N)$ 量级的算法，不过它的正确率会较 MWPM 有所下降，在多轮次时空图上进行时这种效果将更明显，因而有相关研究提出了在时空图上带权重的算法。虽然解码成功率有所提升，但依然解决不了越纠越错的问题，仅在物理错误率本就很低的情况下有一定纠

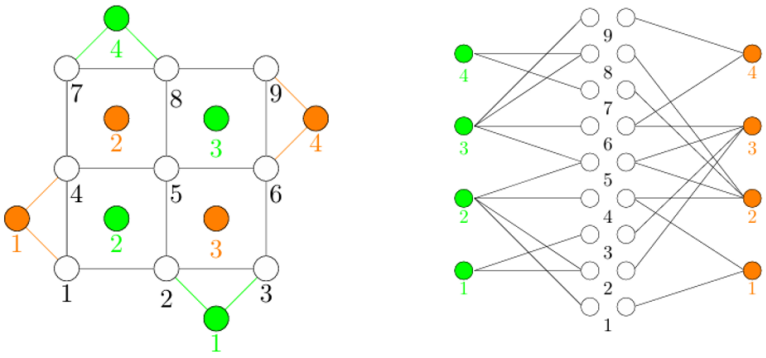
错效果：



3 BP 算法

BP 算法采用另一种思路，通过相邻的稳定子与物理比特之间交换信息进行迭代，根据每一轮迭代后的信息预测每一个物理比特是否发生错误。如果当前预测能够满足错误征状，则结束程序；若否，则持续迭代直至找到满足错误征状的错误或者达到最大迭代次数。

交换信息进行迭代的过程实际上与经典的 BP 算法完全一致，我们只需要将稳定子与错误征状进行相应解读即可。具体地，将物理比特与稳定子构造至如下 Tanner 图中，由于我们将 X 稳定子与 Z 稳定子分别进行处理，因而物理比特需要复制两遍，左右两半边可以进行并行处理，具体示例如下：



针对 X 与 Z 稳定子，可以列出其校验矩阵 H，对应的错误为 e，则错误征状为 $s = H \cdot e$，于是可以按照经典 BP 算法进行处理：

1. 每个物理比特的先验错误率以及第一次迭代是物理比特 v_i 向其关联稳定子 c_j 发送的信息为：

$$\mu^1_{v_i \to c_j} = lch(e_i) = log(\frac{p(e_i = 0)}{p(e_i = 1)})$$

2. 接下来的每一次迭代中物理比特 v_i 向其关联稳定子 c_j 发送的信息为：

$$\mu^t_{v_i\to c_j} = lch(e_i) + \sum_{k=1}^{\sigma-1} \mu^{t-1}_{c_k\to v_i}$$

求和少一次的原因在于不考虑当前发送目标 c_j 上一轮发给 v_i 的信息。

3. 每一次迭代中稳定子 c_j 向其关联物理比特 v_i 发送的信息为：

$$\mu^t_{c_j\to v_i} = (-1)^{s_j} \cdot 2tanh^{-1}(\prod_{k=1}^{\psi-1} tanh(\frac{\mu^t_{v_k\to c_j}}{2}))$$

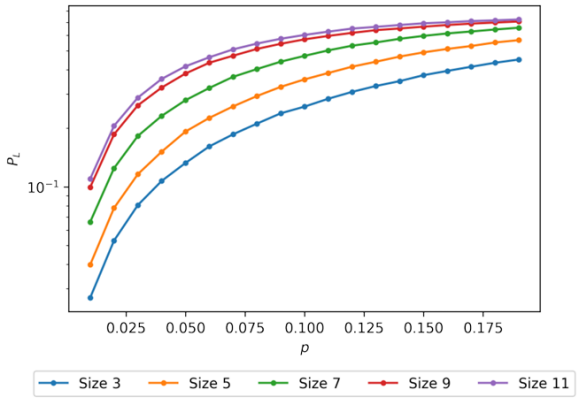
其中 s 表示当前稳定子的报错情况。这个表达式的具体解释暂且不得而知，后续可以深入了解。

发送信息完成后，开始对每个物理比特的错误情况进行估计，首先计算：

$$l^t_{ap}(e_i) = lch(e_i) + \sum_{k=1}^{\sigma} \mu^t_{c_k\to v_i}$$

如果 $l^t_{ap}(e_i) < 0$，则说明该比特犯错 $e_i = 1$，当前轮次预测出的错误为向量 $\hat{e}$，如果 $\hat{s} = H \cdot \hat{e} = s$，则找到满足错误征状的错误，停止迭代。

套用经典算法看似能够得到正确结果，但是实际上给出的解码效果非常糟糕，在不考虑测量错误的情况下不仅看不到盈亏点，而且越纠越错：



究其原因，在于量子纠错中的错误征状具有很高的简并性，只要乘上任意稳定子均给出相似征状，这就导致实际上最可能错误并不是某一个错误链最短的错误，而是与该错误链等价的集合中的最概然集合。BP 算法每一轮的预测依赖于对每个比特上错误的边缘分布的猜测（边缘分布有点像统计力学中考虑单粒子的速度分布一样，对其他所有概率分布积分，只研究关于某一参数的分布），然而表面码的错误简并性很强（稳定子权重远小于码距），因而这样的边缘分布实际上是平均而广泛的，于是乎采用经典的 BP 算法很难给出最可能结果。

这种现象称之为 Split-beliefs，为解决该问题，可以对 BP 算法进行相应后处理程序，称为 BP-OSD。

OSD 的全称为 Ordered Statistics Decoding(有序统计解码)，其核心思路如下：当 $s \neq H \cdot \hat{e}$ 时，我们仍可以通过 BP 算法给出的信息进行分析错误来源。具体地，按照 BP 算法所计算出的 $l_{ap}(e_i)$ 从小到大进行排序，该值越小意味着越有可能发生错误，我们认定错误只在前 rank(H) 个数据比特（挑选

出的 Λ 满秩时）中产生。选取排序后的前 rank(H) 个物理比特，并记录下各自索引位置，选中部分的错误向量可以简单地表述为 $\bar{e} = \Lambda^{-1} \cdot s$，其中 Λ 为选中部分重排后的校验矩阵，未选中部分默认不发生错误。以上就是 OSD-0 的具体过程，一个简单的图例如下：

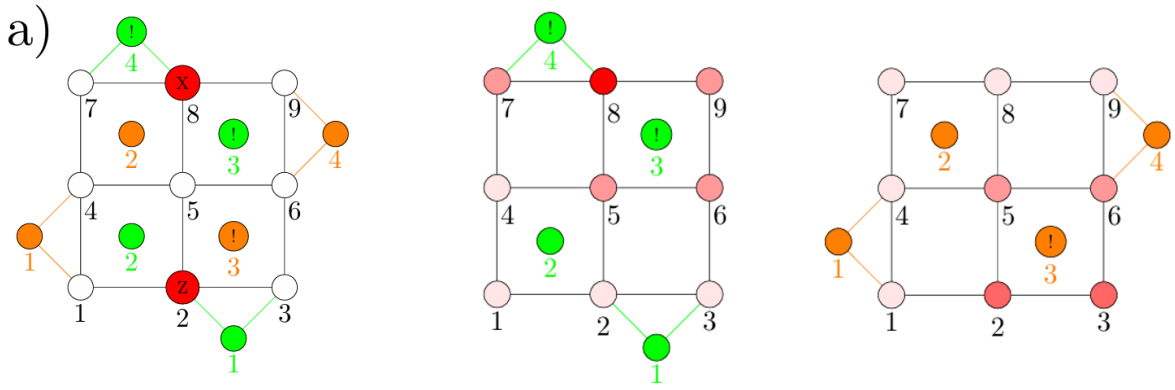
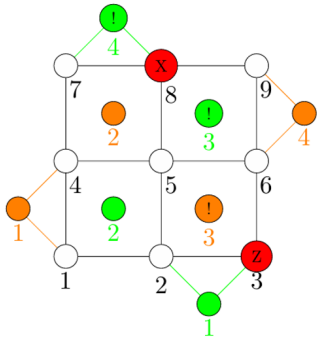


图 a 为选定的错误与报错的稳定子，右侧两图为 BP 算法迭代一轮后的图像，颜色越深表示犯错可能性越高。

$$\Lambda = \begin{array}{c} \begin{matrix} 8 & 7 & 5 & 3 & 2 & 5 & 6 & 4 \end{matrix} \\ \begin{matrix} 4 \\ 3 \\ 2 \\ 1 \\ 4 \\ 3 \\ 2 \\ 1 \end{matrix} \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$



经过排序之后挑选出其中可能出错的物理比特，再进行相应的纠错，在本例中的确得到了正确的解码结果。

当然实际情况中最低权重的解不一定就是仅在挑选部分出错，其他位置处不出错。考虑如下优化，称为 OSD-w 算法，即在未挑选的部分中在前 w 个里出错的概率比较大，将总的错误分为两部分 $e = [e^{w=0}, e']$，则 $\bar{e} = [e^{w=0} + \Lambda_{w=0}^{-1}\Lambda' e', e']$ 满足 $\Lambda\bar{e} = s$。

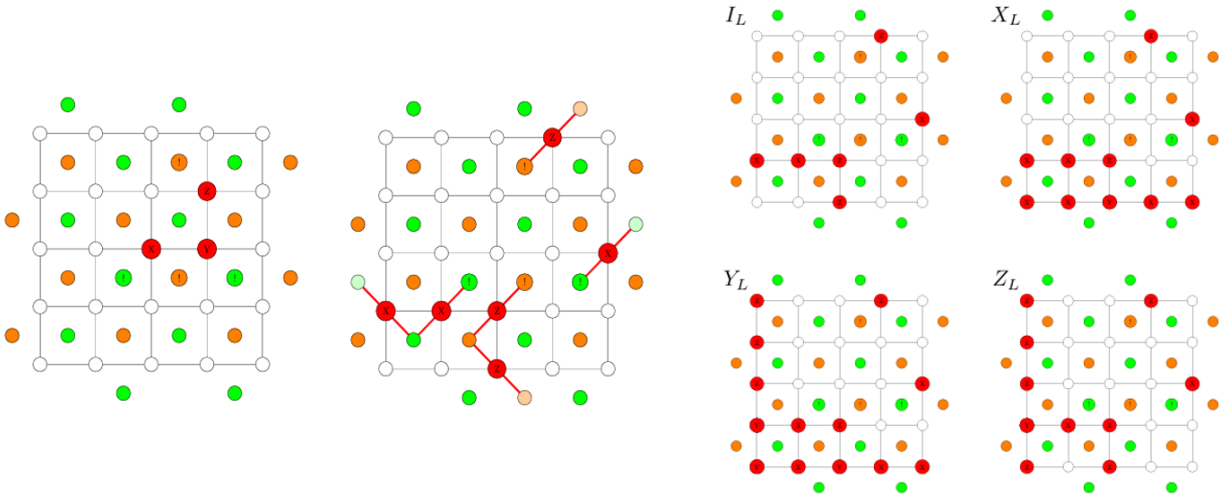
由于 n 通常远大于 rank(H)，因此穷举所有情形并不合理，我们只考虑在未挑选部分中权重为 1 以及前 w 个中权重为 2 的部分的情形，每次比较选择出总权重最低的解。不过实际上后续这些解似乎对结果影响不大，相较于经典 BP 算法的主要提升还是来源于 0 阶结果。

在使用了如上优化之后，阈值终于出现，不过由于其中需要进行矩阵求逆操作，复杂度也来到了 $O(n^3)$ 量级。在目前的使用中，BP 算法和优化并不会单独出现，其总是与前两种算法相结合，以优化前述算法中的权重生成部分。不过这似乎依然并不能绕过测量过程中的错误这一重大问题。

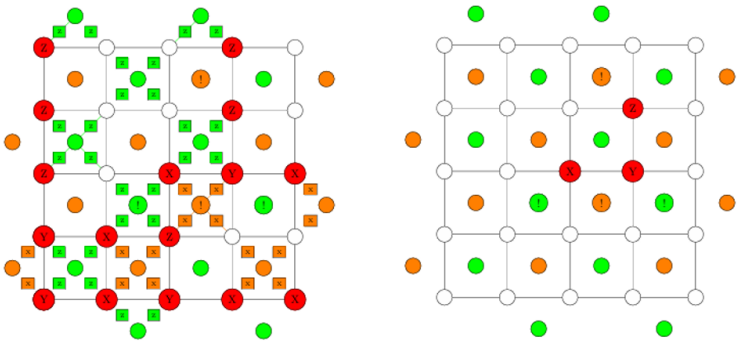
4 张量网络解码器

张量网络解码器的构建思路与之前大不相同，其核心思想在于之间计算各个简并集合的概率，而非某一种最概然的错误链，因而其天然针对的是 DQMLD 问题，可以预见这种解码算法的解码结果应该是最好的。

具体的，我们首先快速找到一组符合征状的错误链 E_{rec}，可以通过与最近的虚拟稳定子相连立即得到。实际上我们只需要判断错误链在 $\{I_L E_{rec}\}, \{X_L E_{rec}\}, \{Y_L E_{rec}\}, \{Z_L E_{rec}\}$ 中的哪一个即可。因而实际上只需要计算各个集合对应的概率，其中的最大值即为我们所寻找的结果。一个简单的例子如下所示：



第一幅图是预设的错误以及给出的报错稳定子，第二幅图是快速寻找的错误链 E_{rec}，通过比较四个集合的概率，最终可以判断出最概然集合为 $Y_L E_{rec}$:



可以看出预设的错误与 $Y_L E_{rec}$ 之间的确只差了左图中所标出的稳定子，说明确实在该集合当中。

于是整个过程的核心就变为了求解以上每个集合各自的概率，该过程可以利用张量网络的方法实现，具体过程如下所示：

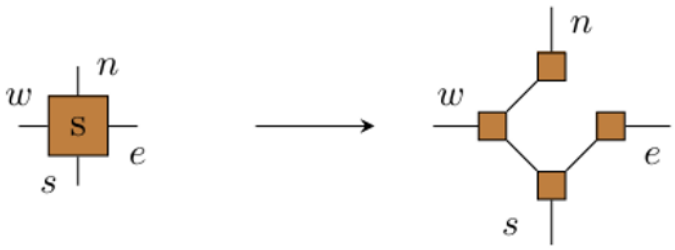
定义单比特算符概率 $\pi_1(X) = \pi_1(Y) = \pi_1(Z) = \frac{\epsilon}{3}, \pi_1(I) = 1 - \epsilon$,

稳定子集合即为 $\mathcal{G}$，对于某一错误链 f，所求即为

$$\pi(f\mathcal{G}) = \sum_{g\in\mathcal{G}} \prod_{e} \pi_1(f_e g_e)$$

其中的 e 指标代表的是每一个物理比特。

我们先考虑如下所示的平面表面码的张量网络构建，再针对我能所熟悉的如上形式的旋转表面码构建对应张量网络：



考虑码距为 d，则有 $d^2+(d-1)^2 = 2d^2-2d+1$ 个物理比特，两种类型的稳定子各有 $d(d-1)$ 个，因此维护了 1 个逻辑比特。

每一个稳定子集合中的元素 g 可以用两个长度为 $d(d-1)$ 的 01 串 α,β 来标记：

$$g(\alpha;\beta) = \prod_{u}(A_u)^{\alpha_u} \prod_{v}(B_v)^{\beta_v}$$

约定 $A_u^0 = B_v^0 = I$，于是每个稳定子在每个物理比特 e 上可以表述为：

$$g_e(\alpha;\beta) = g_e(\alpha_u,\alpha_v;\beta_p,\beta_q)$$

其中 u，v 指代的是该物理比特所关联的两个 X 型稳定子；p，q 指代的是该物理比特所指代的两个 Z 型稳定子。

于是可以利用 α,β 枚举所有稳定子集合中的元素：

$$\pi(f\mathcal{G}) = \sum_{\alpha}\sum_{\beta} T(\alpha;\beta)$$

其中的每一项可以写作：

$$T(\alpha;\beta) = \prod_{e} \pi_1(f_e g_e(\alpha_u,\alpha_v;\beta_p,\beta_q))$$

现在我们解释如何将之对应到右图所构造的张量网络上。在右图中，所有的稳定子我们认为是相同的，即为 s；所有的物理比特分为水平 H 与竖直 V 两类，其中 H 型物理比特的左右两侧是 X 稳定子，上下两侧是 Z 稳定子；V 型物理比特左右两侧是 Z 稳定子，上下两侧是 X 稳定子。

在张亮网络中，边所对应的是被缩并的指标，也就是求和指标。我们将张量网络中的边构型记为 γ，这是一个长度为张量网络中边的数目的 01 串，代表每条边选不选。在当前情形中，每条边选与不选是由 α, β，即各个稳定子选与不选决定的，即 $\gamma = \gamma(\alpha; \beta)$，于是有：

$$\pi(f\mathcal{G}) = \sum_{\alpha} \sum_{\beta} T(\alpha; \beta) = \sum_{valid} \prod_{\gamma} \prod_{e} T_e(\gamma)$$

注意这里 e 是对所有的物理比特求和，而 γ 并不包含所有可能情形的边构型。由于张量网络中边是求和指标，亦即所有边构型都应被遍历，于是我们希望将之变为对所有边构型求和的形式。

注意到只有当每个稳定子 s 周围的所有边都被使用或都不被使用时才是上述求和中所对应的有效边构型，于是可以引入 Kronecker 记号，每个稳定子 s 也计入相应贡献 $\delta_{n,s,w,e}$，其中 n,s,e,w 代表稳定子所连接的上下左右四条边。

综上，每个节点的贡献可以记为：

$$S^i_{n,s,w,e} = \delta_{n,s,w,e}$$

$$H^i_{n,s,w,e} = \pi_1(E^i_{rec} + Z^n + Z^s + X^w + X^s)$$

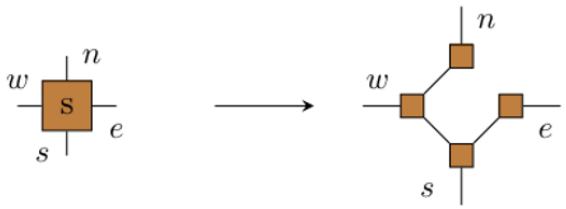
$$V^i_{n,s,w,e} = \pi_1(E^i_{rec} + X^n + X^s + Z^w + Z^s)$$

其中 n,s,w,e 指代某个节点上下左右对应边，对应的是 0 或 1，代表选与不选，$\pi_1(E^i_{rec} + Z^n + Z^s + X^w + X^s)$ 代表该物理比特被作用算子 $E^i_{rec}, Z^n, Z^s, X^w, X^s$ 的概率。

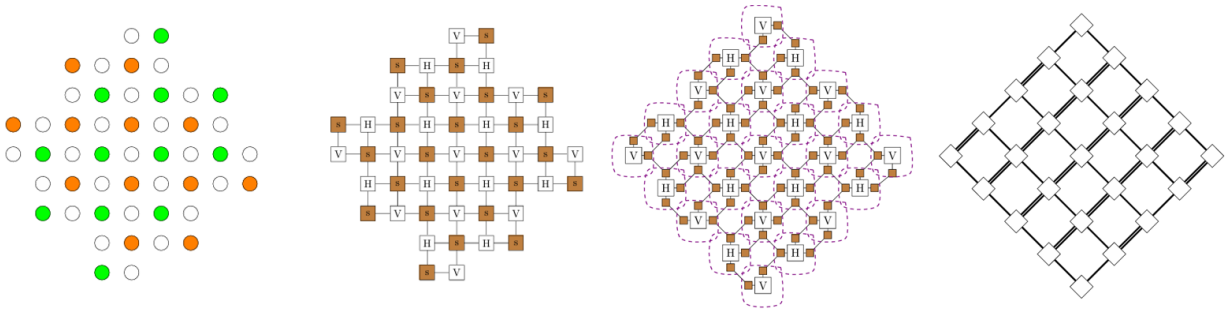
于是乎所求的某一类错误集合的概率被转换为了该张量网络的收缩值：

$$\pi(f\mathcal{G}) = \sum_{\gamma} \prod_{h} H \prod_{v} V \prod_{s} S$$

不过对于我们常用的旋转表面码而言，不能按照以上方式进行简单构造对应的二维张量网络。此时的构造略显复杂，核心想法是将稳定子对应的张量网络节点的连接方式拆分开为如下形式，再进行特定分块以得到排列整齐的二维网络：

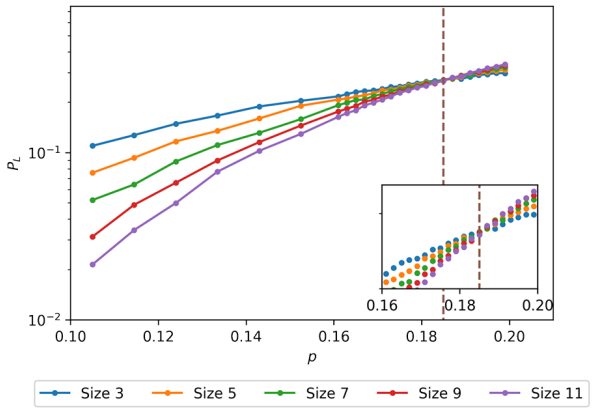


如下例子是一个码距为 5 的旋转表面码的张量网络构造，需要注意的是完成构造之后某些键已经进行了缩并：



如果能够准确求得各个收缩值，我们将得到非常完美的解码结果。可惜的是，随着收缩的进行，收缩键重数的增加是指数级的，因此通常需要设定键重数的截断参数 χ，以近似求得收缩值。该过程的时间复杂度为 $O(n\chi^3)$，可以看出为了得到较为不错的收缩值，χ 的增长造成的复杂度升高是非常迅猛的。

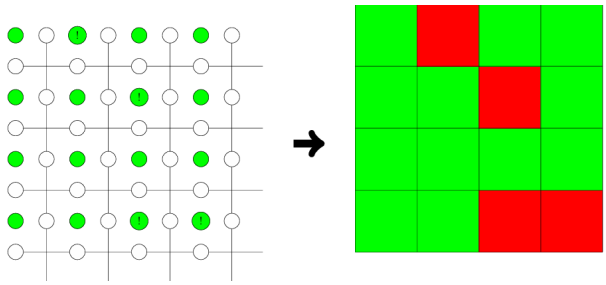
如下是 $\chi = 16$ 时的结果，可以看到无论是盈亏点还是阈值都得到了显著提升：



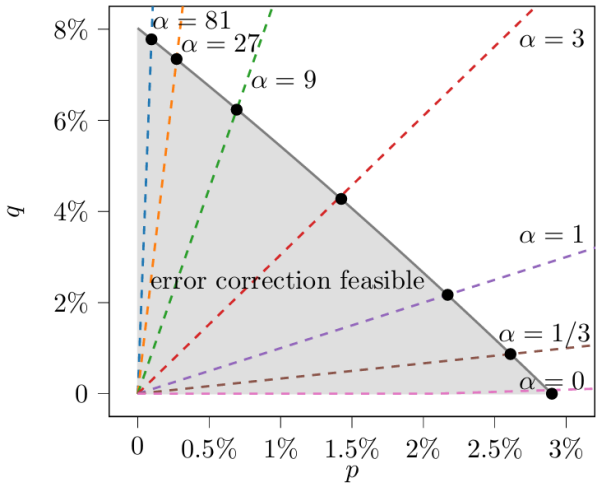
对于测量有错误的情况，仍旧需要综合分析讨论多轮结果。Google 的一篇文章中使用了 25 轮次的测量来得到码距为 5 的解码，这么看起来测量误差所带来的代价似乎显得更大，并且那篇文章看起来只是说了测量轮次达到 25 时解码正确率会变得更好，并没有给出是否得到阈值，之后可以进一步阅读考证一番。

5 元胞自动机解码器

元胞自动机可以理解为一种迭代更新的算法，将报错稳定子构造为活的元胞，通过某种特定的演化规则，可以使得活的元胞相互湮灭，而湮灭的路径则指示了错误链。一个简单的构造示例如下：



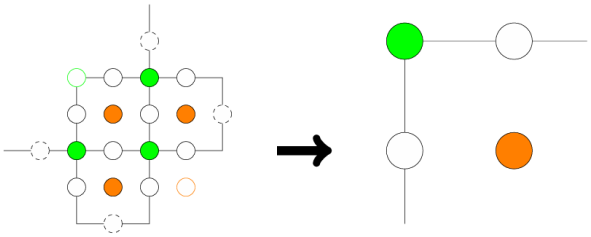
目前为止提出了多种迭代更新规则，比如 Toom 规则，Sweep 规则等。这个做法在我看来似乎与 UF 算法的思路差不多，也属于复杂度低但是解码不一定能得到最概然结果的例子，并且看起来也不能对测量错误有什么办法。不过有作者声称利用 Sweep 规则的元胞自动机可以有效应对测量错误，只需单次解码即可得到正确答案，他给出了如下的图：



纵坐标代表测量错误，横坐标代表相位翻转错误，他声称他的解码算法可以抵抗测量错误，并且对此给出的解释是：测量错误只有在距离相近时才能够导致逻辑错误，与相位翻转错误距离较远的测量错误并不会对结果的正确性造成影响。不过我没看懂他这个解释是什么意思，并且我严重怀疑他所谓的测量错误的定义方式是不是有所不同，我觉得从根源上讲所谓的单次解码就不靠谱。并且他的论文非常奇怪，他展示的结果都是在一些不那么常用的量子纠错码上，却并没有在最普通的二维表面码上展示他的结果，这种本末倒置的做法不免让人产生怀疑。或许之后可以仔细看一看这篇论文检验一下。如果真如作者所说能够有效解决测量错误问题那还是相当吸引人的。

6 重整化群解码器

重整化群解码的思路是将纠错码看作是近似级联码的形式，从外层向内层解码。每一层次的解码规模很小，直接穷举即可实现，因此这同样是一种快速的解码方式。不过该做法需要将纠错码近似看作级联码的形式，一个例子如下：

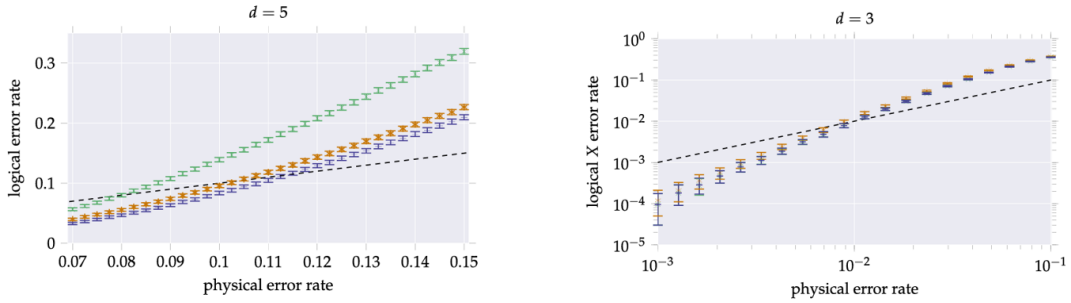


不过我没太看懂这个近似是什么意思，并且最为常用的表面码好像没有相应构造。同时这种解码方式的性能表现也并不是很好，对于测量错误也没什么办法，因此看起来不像是一个有前途的方法。之后有时间可能会仔细看一看，不过优先级应该不高。

7 神经网络解码器

神经网络解码器的核心思路其实与张量网络解码器差不多，我们都是将错误分解为 $E = S \cdot T \cdot L$ 三个部分，分别表示稳定子，一个简单错误，逻辑错误，我们只需要指认对于特定的错误征状，它属于逻辑算子 I, X, Y, Z 所对应的哪一个集合即可。张量网络方法采用直接计算这四个集合对应概率的方式，而神经网络方法通过机器学习完成这一步指认。

不过二者有一个共同的问题，就是对于高码率的编码，即同时编码到多个逻辑比特时，逻辑错误数量显著增多，将会导致时间复杂度急剧上升。同时，该机器学习方法生成训练成本随着码距的升高指数级上升，因此目前看到的结果仅限于非常小的码距：



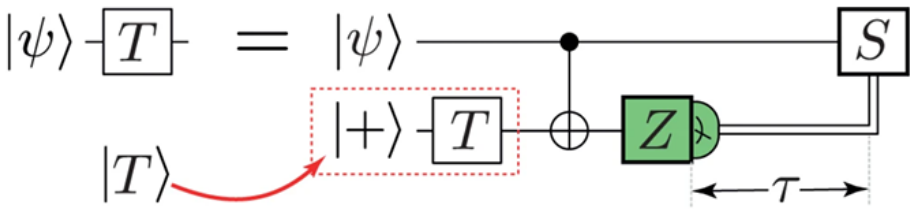
左图是不考虑测量错误的情形，可以看到神经网络解码器对应的蓝色线与 MWPM 对应的橙色线性能差不多，并且都在 0.11 附近出现了阈值。右图是考虑了测量错误的情形，同样可以看到二者性能接近，不过我个人感觉非常小码距的测量错误影响应该不大，测量错误的麻烦之处在于对于较大码距依然难以处理，阈值也是直接下降到了 0.01 以下，不过尽管如此，好像还是和之前的结果不太匹配，毕竟之前的结果经常出现越纠越错看不到阈值的情形。

8 总结与讨论

目前来看，有以下三个问题需要解决：

1. 在具体的物理应用中，某些噪声是偏置的，譬如说超导情形中 Z 噪声的概率更高。如果说此时再像先前的假设那样将 X 与 Z 错误同等对待，显然有所浪费。不过这个问题目前来看是最好解决的，只需要进行一些设计上的调整。比如说用矩形的表面码，让 Z 错误链与 X 错误链有不同的码距等，总之只需要在构造表面码时进行一些调整即可。

2. 解码的速度。对于只用到 Clifford 操作的线路，我们甚至不需要实时解码，因为这样的错误也只会把 Clifford 操作中的一个变为另一个，因此我们可以在最后解码进行错误恢复，并不消耗线路运行时间。但是数学上已经证明仅靠 Clifford 操作无法完成通用量子计算，目前最好的解决方式是采用魔术态蒸馏预先制备一些魔术态，然后将之应用到线路中变成一个逻辑 T 门。利用魔术态引入逻辑 T 门的方式如下：



可以看到其中有一个经典控制的 S 门，这意味着前序的错误恢复必须提前完成，否则经过该 S 门之后错误将被传播得乱七八糟，不再可以通过最后一并处理的方式完成。目前已有相关研究讨论了解码算法的实时性对于线路的影响。同时现成的解码包已经越做越好，许多已经能够在不降低解码性能的情况下接近线性复杂度，因此个人感觉想要在此基础上再有所优化比较困难。

3. 测量错误。在考虑测量错误时盈亏点陡然下降（许多文献中所说的阈值看起来并不是逻辑错误率低于物理错误率的点，而是提高码距能够降低逻辑错误率的点），许多文献中甚至看不到阈值现象，这不禁让人产生疑问。关于现有的采用多轮稳定子测量根据时空图解码的有效性我觉得有必要具体考证。许多文章在面对测量错误问题时通常模糊表述，让人很难搞清楚究竟实现了怎样的功能，要不然就是在一些奇怪的不常用的码上实现。同时，我觉得这里存在一个理论性的问题，我们似乎没办法把获得稳定子测量结果这一步变成容错的，这样一来试图用来纠错的信息总是有错的，那么是否会像木桶原理那样真实的错误实际上由这个测量错误所主导，同时这将导致逻辑错误率始终高于这个测量错误率，因而观察不到阈值现象。不过这不是致命的，因为量子信息是无法复制的，但是稳定子测量却是可以重复进行的，目前的多轮测量稳定子的方式看起来就是使用重复若干遍来降低错误率的方式，不过这看起来并不是一个十分聪明高效的办法。如何有效解决测量错误，或者说测量错误能否被有效解决，这是我觉得比较关键的问题。因为目前来看，按照不计测量错误所得到的接近 10% 的阈值与盈亏点，在现有实验条件下做出实验应该是可能的，所以如果能解决该问题，应该能保证实验上验证纠错。不过我个人现在甚至开始怀疑测量错误是否能够被有效或者高效解决，因为从感觉上来说，如果完全没办法对一个含噪声的系统提取一个完美信息的话，似乎最有效的方式就只能是多次重复了。