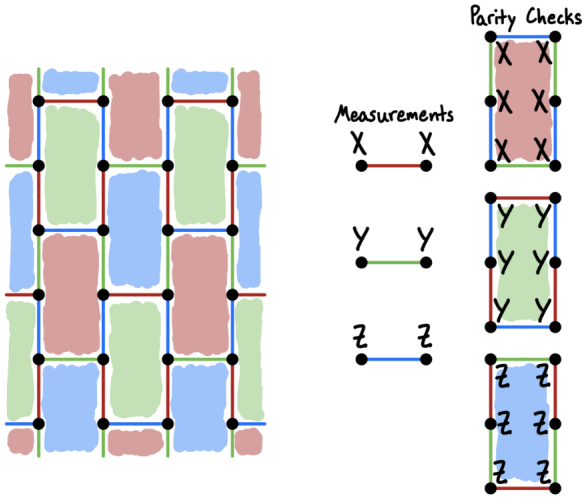
动态码——蜂巢码

林雨轩 2300011304

May 2025

1 宏观理解



动态码的结构如图所示，黑色的点代表数据比特，边代表真正要进行的测量（意味着需要在每条边上放一个辅助比特用于显示测量结果），其中红色而对边代表 X 测量，绿色的边代表 Y 测量，蓝色的边代表 Z 测量。

需要注意的是，我们真正用来进行纠错的探测器并不是以上所提到的那些边，而是由这些边组成的面稳定子。也就是说我们并不是通过直接测量得到右侧的面稳定子的测量结果，而是通过我们已经获得的边检测的测量结果算出面稳定子，进而用于纠错。这一点与 stim 库本身的思路非常像，因为 stim 库进行解码并非直接使用稳定子，而是探测器 (detector)，具体信息参见“stim 基本操作.ipynb”。

因此更确切地说，这些面稳定子实际上正是编写 stim 电路时其中的探测器，解码是基于探测器的探测结果进行的。于是乎在有关如何解码，如何识别错误模式的后续表述中，我们不再使用“稳定子”这一概念，而是使用“探测器”进行表述。

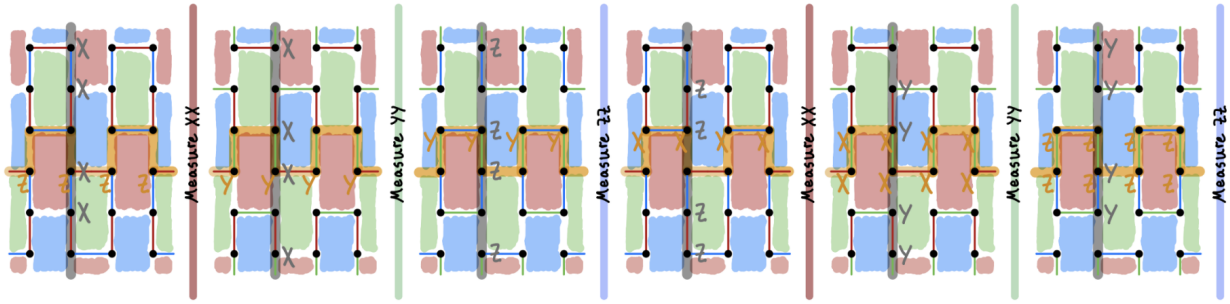
按理说以上编码结构应该放在环面上，但是作者自己表明还没有能够很好地解决这个问题。我觉得他指的可能是无法解决周期边界条件涉及非局域操作的意思，不过模拟是完全没问题的。

通常来说，我们理解一套编码总在关注稳定子是什么、逻辑算符的形式是什么；但是对于判断是否造成逻辑错误以及造成何种逻辑错误而言，我们实际上不必在意以上两点，只需关注：探测器给出的征

状是什么、逻辑算子的测量结果发生了什么变化。也就是说尽管逻辑算子的形式一直在变，但我们不把注意力放在它如何变化上，只关心当前的征状是否影响了逻辑算子的测量结果。

2 细节阐述

首先说明我们所进行的操作是不断地进行以下 3 步循环：测量所有红边、测量所有绿边、测量所有蓝边，然后通过相邻三次测量结果算出探测器的值，进而根据征状判断当前的逻辑测量值是否发生了变化。下图为具体过程：



当然这个过程之所以能成立是需要验证的，具体而言，我们可以把它想成不断地在改变系统的稳定子，这和在表面码中不断改变稳定子从而实现逻辑操作本质上是一样的，只不过这里我们不需要逻辑比特改变，而是维持逻辑比特不变。因此我们只需要检查改动稳定子的过程是否合理即可。

具体地，在第一幅图中，横向的黄色标识中的 Z 代表一个逻辑 Z，纵向的两个 X 连接中间跳过一个数据比特代表的是对应的逻辑 X，可以看出这两个算符是相互对易的。

第一幅图中，所有的稳定子包括所有的蓝边以及所有的格子，显然稳定子之间是相互对易的，因为蓝边与蓝格子对易，而蓝边与红、绿格子有两个交点。

同时，所有的稳定子也与两个逻辑算子对易。具体而言，两个逻辑算子都与所有的格子有两个交点，因此与所有的格子稳定子对易。下面检查两个逻辑算子与所有蓝边的对易性。

逻辑 Z 算子全部由 Z 组成，显然与所有的蓝边对易。逻辑 X 算子与蓝边要么无交点，要么两个交点，因此也对易。

于是乎我们声称的逻辑算子与稳定子的确满足条件。

接下来是最巧妙的部分，这里的两个逻辑算子不仅与所有的蓝边对易，并且与所有的红边也对易。

具体地，逻辑 X 全部由 X 组成，显然与所有的红边对易，逻辑 Z 恰好排布在红边上，因此与所有的红边要么无交点，要么两个交点，因此也是全部对易的。

因此我们在此时可以随意地进行所有的红边测量而不会改变逻辑比特，因为他们与此时的逻辑算子相对易。当我们进行了所有的红边测量之后，相当于把所有的稳定子改成了全部的格子以及所有的红边，并且这个过程是保持逻辑比特不变的。

自然地我们会想知道逻辑算子是否与所有的绿边对易，答案是否定的。比如从上往下数的第二个 X 右侧应该是一条绿边，他就与 luojiX 反对易。

我们想要让整个过程持续起来，就得在把所有红边当作稳定子以后让所有绿边变成稳定子，但是此时绿边与逻辑算子并不对易，此时应该如何处理呢?

事实上，当红边被测量从而成为稳定子之后，逻辑算子就可以进行改动，也就进入到第二幅图中。、

此时，可以检查两个逻辑算子与所有的红边以及绿边都对易，但是与蓝边并不对易。不过我们已经完成了使命，使得此时可以无忧无虑地测量所有绿边并声明其为稳定子却不影响逻辑比特。

由此我们可以继续以上的循环，可以发现循环 3 次并不能让逻辑算子回到原样，必须循环 6 次才能回到初始状态。

实际上我们仔细想想，这和表面码中改变稳定子但是不影响数据比特做的是完全一样的事情，只不过在表面码中我们只是修改某个小的局部，而在蜂窝码中我们则进行大规模有规律的修改稳定子。这里面需要想明白的核心就在于修改稳定子意味着进行新测量，我们并不要求新的测量与当前所有的稳定子都对易，只需要与想保留下来的稳定子对易即可，与那些被取代的稳定子之间的关系并不重要，同时新的稳定子得与逻辑算子对易才能做到不影响逻辑比特。

现在我们来进行量子纠错码中最喜闻乐见的计数环节，即数物理比特，数据比特，逻辑比特的数量。

我们假设有 n 个格子，每个格子 6 个数据比特，但是每个逻辑比特被 3 个格子共享，因此有 2n 个物理比特。我们先忽略边界问题，让整个格子铺满无穷大空间。

每一幅图中的稳定子有两类，其中包括所有的面（共计 n 个），以及某一颜色（假定为红色）的所有边（共计 n 个）。

不过这些稳定子之间并不完全独立，其中有两个约束：

1. 所有的面稳定子的乘积为 1。

2. 所有的红边与所有的红格子乘积为 1。

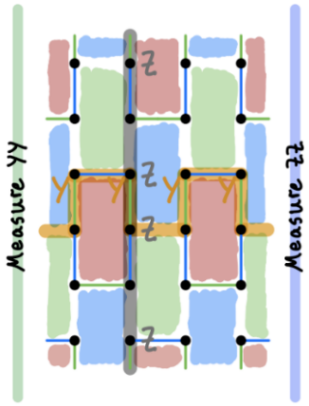
因此我们有 2n-2 个稳定子，理应获得 2 个逻辑比特，现在我们寻找另一个逻辑比特在哪儿。

实际上我们看第一幅图和第四幅图，这两幅图的稳定子完全相同，但是逻辑算子却不同。实际上我们可以验证，第一幅图与第四幅图中的逻辑算子应该是对易的。具体来说，1 黄与 4 黄对易，1 黄与 4 黑对易，1 黑与 4 黄对易，1 黑与 4 黑需要仔细检查一下。可以发现当上下能形成周期边界条件时，需要有偶数个交点，因此相互对易。由此可见，这确实是两个互相独立的逻辑比特。

所以在更改稳定子的过程中，实际上实现了两个逻辑比特之间信息的互换，我不确定这是否可以作为 swap 门的一种实现方式，但看起来确实合理。把图 1 和图 4 拎出来看，稳定子结构完全相同，但是原来那些储存第一逻辑比特的物理比特现在储存的是第二逻辑比特。

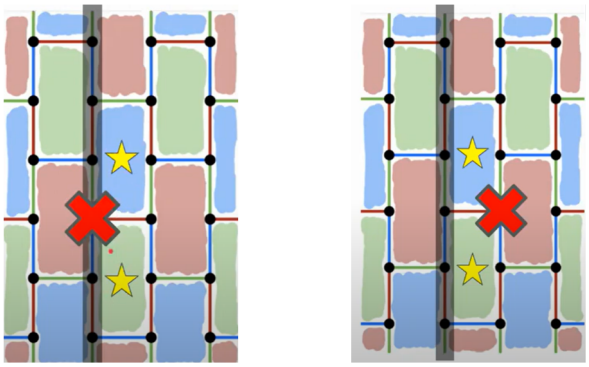
3 纠错过程

我们现在通过一个例子说明为什么不能通过直接测量面稳定子作为探测结果，而必须结合最近 3 轮的边测量从而得到所有的面稳定子。考虑现在稳定子的变化过程进行到了这一步：



此刻所有的稳定子由绿边以及所有面稳定子构成，我们即将将之替换成所有蓝边与面稳定子。

我们考虑如下两种单比特 X 错误：



在左图中，该错误会导致与之相邻的两个蓝色，绿色面稳定子报错，并同时引发逻辑错误，因为逻辑比特包含该位置的 Z 算符，与之反对易。在右图中，错误会导致同样位置的两个稳定子错误，但是却不引发逻辑错误。

由此，如果同时发生这两个错误，将会导致面稳定子均不报错，但是逻辑比特发生错误，因此这个码的码距上限为 2。

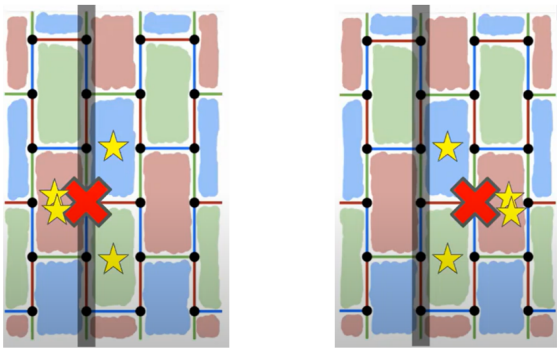
这就是直接测量面稳定子将导致的后果，但是如果利用所有最近 3 轮的边测量来不断更新面稳定子（我们后面将使用探测器来表述），这样的错误实际上是能够被识别出来的。

具体地，我们回到之前的 6 个循环过程中，我们发现识别不出来这两个错误会导致逻辑错误误判的情况仅在第 3 图与第 6 图处，即测完所有绿边，准备测量所有蓝边的时候，因此我们只需应对这种状况。

现在考虑刚测完所有的绿边，此时构成探测器测量结果的是当前的绿边测量，上一轮的红边测量，以及再上一轮的蓝边测量，这时如果发生单比特 X 错误的话，与之相邻的三个探测器都会报错，因为左侧的红探测器使用的蓝边测量是在还没发生此错误时发生的。

此后如果再进行蓝边测量，用新测的蓝边更新探测器结果，此时红探测器就不报错了，因此整体来

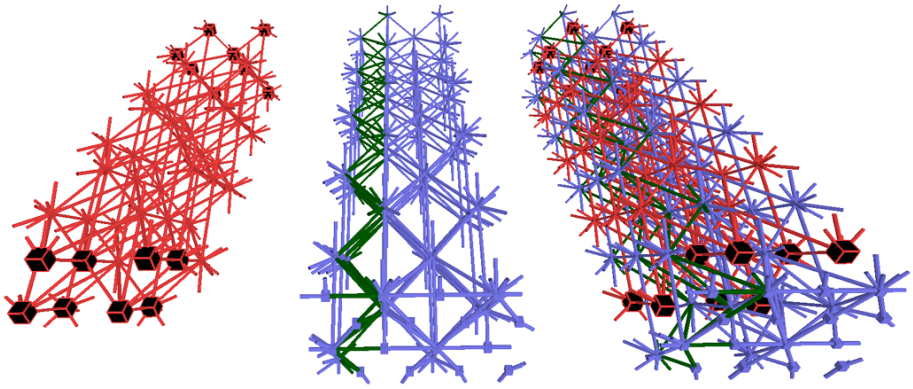
看，如果发生该错误的话，红探测器实际上会闪一下。同理，发生右边的错误的话，右边的红探测器也会闪一下，由此我们可以通过这种“闪一下”的征状判断出该错误的发生。可以参考如下图示：



由此，我们拥有一个基本的信念：使用探测器的探测结果能够有效地解决问题。现在我们开始为线路注入错误并声明探测器。

注意到在每一种颜色的边测量结束之后，所有的探测器都会进行更新，注意探测器的结果要乘上再上一轮的探测器结果，以保证错误是在两轮之间发生的。在开始边测量之前，我们先对要测量的比特施加一定概率的退极化错误，同时再对测量结果施加一定的错误，这样同时模拟了物理比特上的噪声以及测量噪声。

在声明了探测器并传入噪声以后，stim 将自动生成探测器噪声模型（dem），也就是把探测器的征状与错误机制之间建立了联系。stim 可以由此生成一个解码图，其中的点代表探测器，边代表某一种错误机制，解码图如下所示：



图中标黑的节点代表错误机制只引发单独的探测器报错。整个解码图可以分成两个非连通部分，左边代表那些与虽然会引发探测器报错但是却不造成逻辑错误的部分，中间图则表示会引发逻辑错误的联通部分，其中绿色的边代表直接引发逻辑错误的部分。右图是左侧两个图的合并。

最后我们在这个图上运行解码算法，找到探测器征状所对应的最可能错误机制，并根据该路径是否通过图中绿色部分判断是否引发逻辑错误。

从最后的结果上来看，这个蜂巢码似乎并没有比传统的表面码具有更多的优势，不过无论如何，确实是一种较为新颖的想法。