Extension Implementation Plan

03 March 2025 15:12

The basic version of the algorithm is only for a single round, this can easily be extended to be multiple rounds by making it 3D. You can also then add measurement errors this way.

My current confusion is how to generate the initial erasure model (which is then updated using the syndromes).

I do currently have the probabilities of error for each of the syndromes (these can be found in the detector error model)

There are 3 parts of the extension that need to be implemented:

1. Preparing the input

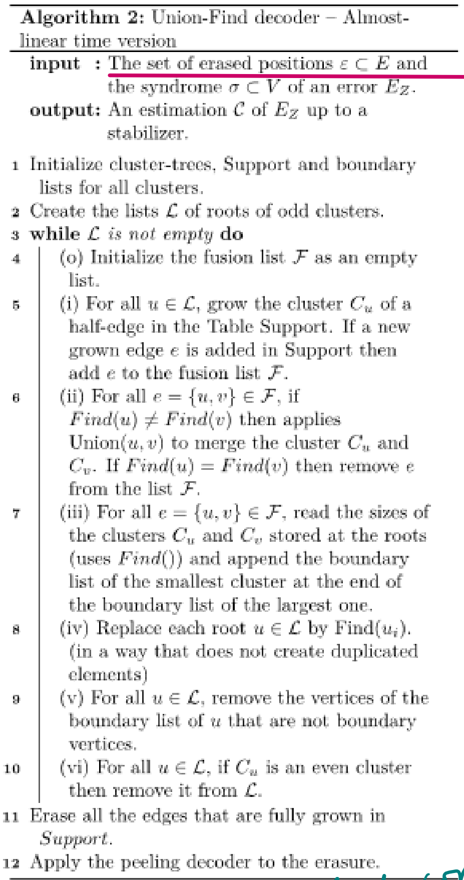
   From stim I can easily get:

   The probabilities of errors for each of the syndrome (as well as the conditional probabilities of the syndrome outputs)

   This potentially may be needed to make the initial erasure model, especially if using integer weights proportional to probability

   The detection events for each shot (eg each different experiment of the code) - large arrays containing TRUE and FALSE for each of the detectors

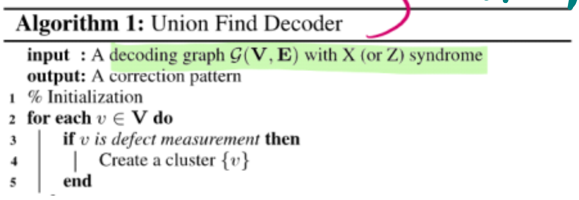
2. Implementing the erasure
   a. Syndrome Validation



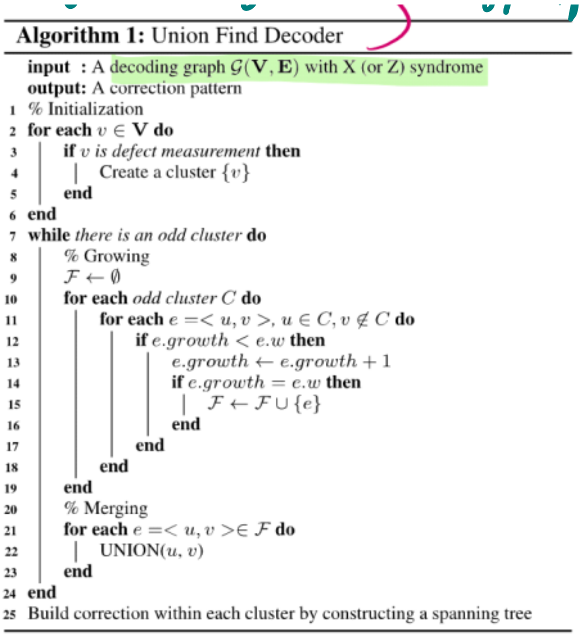
```
Algorithm 2: Union-Find decoder – Almost-linear time version
input  : The set of erased positions $\varepsilon \subset E$ and
         the syndrome $\sigma \subset V$ of an error $E_Z$.
output: An estimation $\mathcal{C}$ of $E_Z$ up to a
         stabilizer.
Initialize cluster-trees, Support and boundary
  lists for all clusters.
Create the lists $\mathcal{L}$ of roots of odd clusters.
while $\mathcal{L}$ is not empty do
    (o) Initialize the fusion list $\mathcal{F}$ as an empty
      list.
    (i) For all $u \in \mathcal{L}$, grow the cluster $C_u$ of a
      half-edge in the Table Support. If a new
      grown edge $e$ is added in Support then
      add $e$ to the fusion list $\mathcal{F}$.
    (ii) For all $e = \{u, v\} \in \mathcal{F}$, if
      $Find(u) \neq Find(v)$ then applies
      Union$(u, v)$ to merge the cluster $C_u$ and
      $C_v$. If $Find(u) = Find(v)$ then remove $e$
      from the list $\mathcal{F}$.
    (iii) For all $e = \{u, v\} \in \mathcal{F}$, read the sizes of
      the clusters $C_u$ and $C_v$ stored at the roots
      (uses $Find()$) and append the boundary
      list of the smallest cluster at the end of
      the boundary list of the largest one.
    (iv) Replace each root $u \in \mathcal{L}$ by Find$(u_i)$.
      (in a way that does not create duplicated
      elements)
    (v) For all $u \in \mathcal{L}$, remove the vertices of the
      boundary list of $u$ that are not boundary
      vertices.
    (vi) For all $u \in \mathcal{L}$, if $C_u$ is an even cluster
      then remove it from $\mathcal{L}$.
Erase all the edges that are fully grown in
  Support.
Apply the peeling decoder to the erasure.
```

unsure how to initialise

an alternative that consider's integer weights e.w that depend on probability

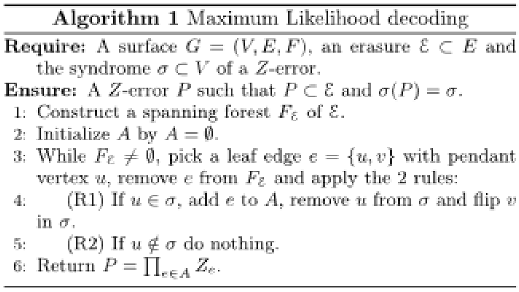


```
Algorithm 1: Union Find Decoder
input  : A decoding graph $\mathcal{G}(\mathbf{V}, \mathbf{E})$ with X (or Z) syndrome
output: A correction pattern
% Initialization
for each $v \in \mathbf{V}$ do
    if $v$ is defect measurement then
        Create a cluster $\{v\}$
    end
```



```
Algorithm 1: Union Find Decoder
input  : A decoding graph $\mathcal{G}(\mathbf{V},\mathbf{E})$ with X (or Z) syndrome
output: A correction pattern
% Initialization
for each $v \in \mathbf{V}$ do
    if $v$ is defect measurement then
        Create a cluster $\{v\}$
    end
end
while there is an odd cluster do
    % Growing
    $\mathcal{F} \leftarrow \emptyset$
    for each odd cluster $C$ do
        for each $e =< u, v >, u \in C, v \notin C$ do
            if $e.growth < e.w$ then
                $e.growth \leftarrow e.growth + 1$
                if $e.growth = e.w$ then
                    $\mathcal{F} \leftarrow \mathcal{F} \cup \{e\}$
                end
            end
        end
    end
    % Merging
    for each $e =< u, v > \in \mathcal{F}$ do
        UNION($u$, $v$)
    end
end
Build correction within each cluster by constructing a spanning tree
```

b. Erasure decoder - the peeling algorithm



```
Algorithm 1 Maximum Likelihood decoding
Require: A surface $G = (V, E, F)$, an erasure $\mathcal{E} \subset E$ and
    the syndrome $\sigma \subset V$ of a $Z$-error.
Ensure: A $Z$-error $P$ such that $P \subset \mathcal{E}$ and $\sigma(P) = \sigma$.
Construct a spanning forest $F_{\mathcal{E}}$ of $\mathcal{E}$.
Initialize $A$ by $A = \emptyset$.
While $F_{\mathcal{E}} \neq \emptyset$, pick a leaf edge $e = \{u, v\}$ with pendant
    vertex $u$, remove $e$ from $F_{\mathcal{E}}$ and apply the 2 rules:
    (R1) If $u \in \sigma$, add $e$ to $A$, remove $u$ from $\sigma$ and flip $v$
    in $\sigma$.
    (R2) If $u \notin \sigma$ do nothing.
Return $P = \prod_{e \in A} Z_e$.
```

I don't think that constructing the spanning tree is a necessary step as I think that the output of syndrome validation is spanning trees

3. Evaluating the decoder

From STIM can get 'observable_flips' - which I think returns the result of the logical observable (where the logical observables defined by the user).

Basically the output of the erasure decoder will be corrections that need to be done

So if there is a corresponding error and correction event, or no error and no correction event, then it is deemed a success, otherwise it is a failure

Could initially just implement for 1 round (meaning that the graph is only in 2D - even though I think it works exactly the same in 3D)

I've had a quick look into Graph libraries:

The key graph contains all the vertices that exist as well as the edges connecting them (the detector error model)

As far as I can tell, the implementation of the 'Syndrome Validation' algorithm doesn't need any graph functions directly, it stores vertices in union-find data structures, and edges in an array called 'Support'. Would need to be able to identify which edges from which vertices (to be able to grow the odd clusters). The initial clusters are all the vertices which were marked 'True' - would be useful to store this information with the node.

```
list(G.adj[1])# or list(G.neighbors(1))
```

Output of syndrome validation stage seems to be clusters of vertices, and the edges have been annotated with which cluster they are part of

A graph library could be useful for identifying which edges are the 'leaf edges'?

A common graph library is NetworkX

I could implement my own union-find?