Minimum-Weight Parity Factor Decoder for Quantum Error Correction

Yue Wu,[1] Binghong Li,[1] Kathleen (Katie) Chang,[2] Shruti Puri,[2] and Lin Zhong[1]

[1] *Yale University, Department of Computer Science, New Haven, CT 06511*
[2] *Yale University, Department of Applied Physics, New Haven, CT 06520*

ABSTRACT

Fast and accurate quantum error correction (QEC) decoding is crucial for scalable fault-tolerant quantum computation. Most-Likely-Error (MLE) decoding, while being near-optimal, is intractable on general quantum Low-Density Parity-Check (qLDPC) codes and typically relies on approximation and heuristics. We propose HyperBlossom, a unified framework that formulates MLE decoding as a Minimum-Weight Parity Factor (MWPF) problem and generalizes the blossom algorithm to hypergraphs via a similar primal-dual linear programming model with certifiable proximity bounds. HyperBlossom unifies all the existing graph-based decoders like (Hypergraph) Union-Find decoders and Minimum-Weight Perfect Matching (MWPM) decoder, thus bridging the gap between heuristic and certifying decoders.

We implement HyperBlossom in software, namely Hyperion. Hyperion achieves a 4.8x lower logical error rate compared to the MWPM decoder on the distance-11 surface code and 1.6x lower logical error rate compared to a fine-tuned BPOSD decoder on the $[[90, 8, 10]]$ bivariate bicycle code under code-capacity noise. It also achieves an almost-linear average runtime scaling on both the surface code and the color code, with numerical results up to sufficiently large code distances of 99 and 31 for code-capacity noise and circuit-level noise, respectively.

I. INTRODUCTION

Quantum error correction (QEC) is crucial for realizing scalable and fault-tolerant quantum computation. A fundamental challenge in QEC is decoding. Most-Likely-Error (MLE) decoding provides near-optimal accuracy by finding the error pattern with maximum probability. While MLE decoding for surface codes can be efficiently solved as a Minimum-Weight Perfect Matching (MWPM) [1] problem via the blossom algorithm [2–4], extending this approach to general quantum Low-Density Parity-Check (qLDPC) codes remains intractable given its NP-hardness on general hypergraphs [5].

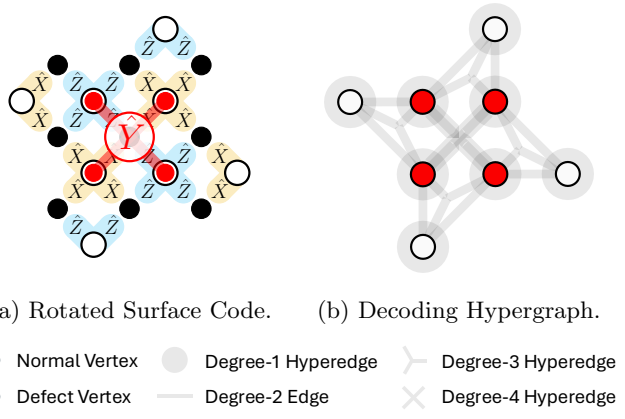
Existing qLDPC decoders, such as Union-Find (UF) [6, 7] and Belief Propagation with Ordered Statistics (BPOSD) [8, 9] decoders, offer fast heuristic solutions but lack rigorous guarantees and often require careful parameter tuning. Meanwhile, certifying decoders that provide rigorous optimality or proximity bounds have been largely limited to special cases, such as surface codes. This gap between heuristic efficiency and certifiable correctness has hindered the design of fast and accurate QEC decoders.

We address this challenge by introducing a unified mathematical framework, called HyperBlossom, that formulates the MLE decoding problem of qLDPC codes as a Minimum-Weight Parity Factor (MWPF) problem on the decoding hypergraph. This framework generalizes the principles of the blossom algorithm to hypergraphs through three key innovations:

- **Primal-dual linear programming (LP)** formulation that provides a certifiable proximity bound and guarantees optimality for certain hypergraphs, e.g., simple graphs and nullity$_{\leq 1}$ hypergraphs.
- **Relaxing** and modular relaxer-finding algorithms that decompose the complex LP optimization problem into tractable relaxer finders.
- **Clustering** technique that exploits the locality of MLE decoding to achieve an almost-linear average runtime while preserving global optimality under the **Certifiability Condition**.

We propose *SingleHair*, an efficient but suboptimal relaxer-finding algorithm for general hypergraphs. Our software implementation, Hyperion, realizes the HyperBlossom framework and *SingleHair* relaxer finder in practice. In our evaluation, Hyperion achieves a 4.8x lower logical error rate compared to the MWPM decoder on the distance-11 surface code and 1.6x lower logical error rate compared to a fine-tuned BPOSD decoder on the $[[90, 8, 10]]$ bivariate bicycle code under code-capacity noise. It also achieves an almost-linear average runtime scaling on both the surface code and the color code, with numerical results up to sufficiently large code distances of 99 and 31 for code-capacity noise and circuit-level noise, respectively.

More importantly, the HyperBlossom framework unifies UF, MWPM, and Hypergraph UF (HUF) [7] decoders within a single mathematical formulation. Within this framework, the MWPM decoder is realized through a *Blossom* relaxer finder, while the UF and HUF decoders correspond to a *UnionFind* relaxer finder. In particular, UF can be viewed as a special case of HUF on simple graphs, and both can be interpreted as simplified MWPF decoders with a trivial relaxer finder. Thus, the HyperBlossom framework not only generalizes these decoders but also provides a unified lens through which their designs and performance trade-offs can be understood in previously impossible ways.



(a) Rotated Surface Code. (b) Decoding Hypergraph.

FIG. 1: Example decoding hypergraph of rotated surface code with depolarizing noise.

II. BACKGROUND

A. Most Likely Error (MLE) Decoding

An MLE decoder aims to determine the error pattern that is most likely to happen given a noise model, while agreeing with the measured syndrome. Using protocols like Pauli twirling [10], a noise model can be decomposed into a set of independent physical error sources, each of which is represented by Pauli operators acting on qubits. If a physical error occurs, it flips the stabilizer measurements intended to monitor this error source. Since a stabilizer may monitor multiple error sources, it may be flipped multiple times. In QEC, a stabilizer measures a *defect* if it is flipped an odd number of times.

We represent the MLE decoding problem on a hypergraph $G = (V, E)$, where each stabilizer measurement corresponds to a vertex $v \in V$, and each independent physical error corresponds to a hyperedge $e \in E$ that connects the defect vertices it generates when it occurs alone. Fig. 1 shows an example decoding hypergraph. In our visualization, we fill the defect vertices D with solid red, and keep the non-defect vertices in white. For a degree-1 hyperedge $e = \{v\}$, we draw a circle centered at v. For other hyperedges, we use the Tanner graph visualization by connecting every vertex $v \in e$ to a center point (hidden for visual simplicity).

MLE decoding on general decoding hypergraphs is notoriously difficult given its NP-hardness [5].

A possible solution for the MLE decoding problem is a subset of physical errors, called the error pattern $\mathcal{E} \subseteq E$. It can be denoted by $\vec{x} = (x_e)_{e\in E} \in \mathbb{F}_2^{|E|}$ where $x_e = 1$ if e is part of the error pattern, i.e., $e \in \mathcal{E}$. The measured syndrome denoted by the defects $D \subseteq V$ puts constraints on the solutions. Let $\mathcal{D}(\mathcal{E})$ denote the defects of the error pattern $\mathcal{E}$, which is defined below.

Definition: Defects of Error Pattern. Given any error pattern $\mathcal{E} \subseteq E$, $\mathcal{D}(\mathcal{E})$ is the set of defects it generates.

$$\mathcal{D}(\mathcal{E}) = \{v \in V |\ \mathcal{E} \cap E(v) \text{ has an odd cardinality}\}$$

Definition: Parity Factor. Given syndrome D, a parity factor is an error pattern $\mathcal{E} \subseteq E$ such that $\mathcal{D}(\mathcal{E}) = D$.

This term was first introduced in [11] and then further discussed in [12, 13]. The general form, known as the parity (g, f)-factor [13], describes a subgraph $\mathcal{E} \subseteq E$ such that for every vertex $v \in V$, the degree satisfies $g(v) \le |\mathcal{E} \cap E(v)| \le f(v)$ and $|\mathcal{E} \cap E(v)| \equiv g(v) \mod 2$. In the decoding context, only the parity constraint is enforced, which corresponds to setting $f(v) = +\infty$ and $g(v) = 1$ for defect vertices and $g(v) = 0$ otherwise.

The solution for the MLE decoding problem must be a *Parity Factor*. Additionally, MLE decoding aims to identify the parity factor of the highest probability. Given the independence between the error sources, the probability of a parity factor $\mathcal{E}$ is:

$$P(\mathcal{E}) = \left(\prod_{e\in\mathcal{E}} p_e\right)\left(\prod_{e\in E\setminus\mathcal{E}} (1-p_e)\right) \propto \prod_{e\in\mathcal{E}} \frac{p_e}{1-p_e}$$

We define the weight of an edge so that the sum of the edge weights relates to the probability of the error pattern. Maximizing probability $P(\mathcal{E})$ is equivalent to minimizing the sum of edge weights in $\mathcal{E}$, i.e., $\min \sum_{e\in\mathcal{E}} w_e$.

Definition: Weight of Edge. For an error source of edge $e \in E$ with probability p_e, we define its edge weight as:

$$w_e = \log \frac{1-p_e}{p_e}$$

Definition: Weight of Error Pattern. For an error pattern $\mathcal{E} \subseteq E$, we define its weight as $W(\mathcal{E})$.

$$W(\mathcal{E}) = \sum_{e\in\mathcal{E}} w_e = \log \prod_{e\in\mathcal{E}} \frac{1-p_e}{p_e} = -\log P(\mathcal{E}) + C$$

Thus, the goal of MLE decoding problem is to find a parity factor $\mathcal{E} \subseteq E, \mathcal{D}(\mathcal{E}) = D$ with minimum $W(\mathcal{E})$. That is, to solve the MLE decoding problem is to find a *Minimum Weight Parity Factor* (MWPF) in the decoding hypergraph.

One caveat is that the edge weights w_e can be negative when its error probability $p_e > 50\%$. In this case, we can convert the problem to an equivalent one with $w_e \ge 0, \forall e \in E$ in polynomial time [3]. Specifically, if $p_e > 50\%$, we can treat e as an "always-occurring" error and redefine the probability to $p'_e = 1 - p_e \le 50\%$. We can solve the MLE $\mathcal{E}'$ on a modified graph of $w'_e = -w_e \ge 0$ and a modified syndrome $D' = D \oplus e$. The MLE of the original problem is then $\mathcal{E} = \mathcal{E}' \oplus \{e\}$. In the rest of the paper, we assume all edge weights are non-negative.

B. MWPF Problem Formulation

In the above, we showed that the MLE decoding problem is equivalent to the **Minimum Weight Parity Factor** (MWPF) problem on the decoding hypergraph. That is, maximizing the probability of a parity factor $\mathcal{E} \subseteq E, \mathcal{D}(\mathcal{E}) = D$ is equivalent to minimizing its weight $W(\mathcal{E})$. Inspired by the blossom algorithm, we formulate the MWPF problem by introducing a set of variables x_e, $\forall e \in E$, with which an error pattern $\mathcal{E}$ is represented: $x_e = 1,\ \forall e \in \mathcal{E}$ and $x_e = 0,\ \forall e \in E \setminus \mathcal{E}$. Given the set of defect vertices $D \subseteq V$, the non-defect vertices are $\overline{D} = V \setminus D$. Unlike the blossom algorithm, which solves the MWPM problem on the syndrome graph, we formulate the MWPF problem and solve it on the decoding hypergraph.

MWPF (Minimum-Weight Parity Factor)

$$\min \quad \sum_{e \in E} w_e x_e \tag{1}$$

$$\text{subject to} \quad x_e \in \{0,1\} \qquad \forall e \in E \tag{1a}$$

$$\sum_{e \in E(v)} x_e = 1 \mod 2 \qquad \forall v \in D \tag{1b}$$

$$\sum_{e \in E(v)} x_e = 0 \mod 2 \qquad \forall v \in \overline{D} \tag{1c}$$

The formulation above involves mixing $\mathbb{F}_2$ constraints (Eq. (1b) and Eq. (1c)) with an objective function defined in the real field $\mathbb{R}$. Note that this formulation is widely considered in prior work [14–18], albeit not in this particular form.

C. Minimum-Weight Perfect Matching (MWPM)

When the decoding hypergraph is a simple graph, i.e., $\deg(e) = 2, \forall e \in E$, the MLE decoding problem reduces to the minimum-weight perfect matching (MWPM) problem on a simple graph called syndrome graph [1, 19]. The syndrome graph $G^* = (D, E^*)$ is a complete graph over the defect vertices D, where each edge $(u, v) \in E^* = \{(u, v) | u, v \in D, u \neq v\}$ is weighted by the weight of the minimum-weight path between u and v in the decoding graph. We use a superscript $*$ to distinguish notations specific to the syndrome graph from that of the decoding graph. The MWPM decoder [20] solves this special case using the blossom algorithm with polynomial complexity [21, 22].

However, it is commonly believed that the same reduction does not work for hypergraphs, for two reasons. First, the reduced "syndrome hypergraph" would have exponentially many hyperedges because the trick of path compression is only applicable to simple graphs. Second, hypergraph perfect matching is a well-known NP-hard problem [23].

Parity Blossom [4] is one of the fastest MWPM decoder implementations, using the famous blossom algorithm [22]. It structures the decoder into two phases: the Primal phase and the Dual phase, working on the primal solution $\vec{x^*}$ and dual solution $\vec{y^*}$, respectively. By implementing the Dual phase on the decoding graph instead of the syndrome graph, it achieves an almost-linear average time complexity.

III. HYPERBLOSSOM FRAMEWORK

We introduce a mathematical framework for solving the MWPF problem for QEC decoding, called HyperBlossom. The framework exploits the sparsity of the edges and defect vertices in quantum error correction to accelerate the **common** case. It draws inspiration from the blossom algorithm family [2, 22], especially Parity Blossom [4]. We formally define the linear programming problems in §III B. We then discuss the clustering technique (§III F) and relaxing technique (§III E), both preserving optimality and simplifying the problem. The mathematical framework provides the following benefits that distinguish from other qLDPC decoders:

- Unlike heuristic decoders, it rigorously proves a proximity bound along with each solution (§III B).
- It interoperates with MWPM decoders (§C) for decoding heterogeneous QEC architectures [24].
- With clustering technique (§III F), it decomposes a large problem into small clusters when $p \ll 1$, with the potential to achieve an almost-linear average decoding time to $|D| \propto p|V|$.
- It reduces the complex linear programming problem into simpler relaxer finding algorithms (§III E). Using the *relaxing* technique, relaxer finding algorithms can compose and complement each other. We discuss some relaxer finding algorithms in §IV and we expect more to be found in the future.

A. Important Concepts

We first introduce important concepts and notations for the problem definition. Following convention, we use $E(V_S) = \{e \in E | e \cap V_S \neq \varnothing\}$ and $E[V_S] = \{e \in E | e \subseteq V_S\}$ to denote the set of edges incident to $V_S \subseteq V$ and the set of edges within it, respectively. We note that the above also uses e to stand for the subset of vertices incident to the edge e as is convention.

Definition: Invalid. For a decoding hypergraph $G(V, E)$ and syndrome $D \subseteq V$, we say it is *Invalid* if and only if there exists no parity factor within it.

$$\forall \mathcal{E} \subseteq E, \mathcal{D}(\mathcal{E}) \neq D$$

For a subgraph $S = (V_S, E_S)$, $V_S \subseteq V, E_S \subseteq E[V_S]$ of a decoding hypergraph $G(V, E)$, we say it is an *Invalid Subgraph* if (V_S, E_S) is *Invalid* for the partial syndrome $D \cap V_S$. $\mathcal{O}$ denotes the set of all *Invalid* subgraphs of G. The decoding hypergraph $G(V, E)$ is *Valid* by definition, thus $G \notin \mathcal{O}$.

Definition: Hair. The *Hair* of an *Invalid* subgraph S is the set of edges not in E_S but incident to at least one vertex in V_S.

$$\delta(S) = E(V_S) \setminus E_S$$

Note that here E_S could be any subset of $E[V_S]$. This is different from the blossom algorithm (§II C), where E_S is always $E[V_S]$ so that $\delta(S) = E(V_S) \setminus E[V_S] = \{(u, v) \in E | u \in V_S \wedge v \notin V_S\}$ as defined in the original blossom algorithm [22]. We introduce this difference for two reasons. First, MWPF on hypergraphs requires a more complicated polytope to describe the parity constraints, and we show that removing flexibility in E_S for certain hypergraphs leads to suboptimal polytope (§A 5). Second, determining the existence of a parity factor on a subgraph (V_S, E_S) (using parity matrix in §IV A 1) is much easier than finding the minimum-weighted one, while determining the existence of a perfect matching is almost as hard as finding the minimum-weighted one, even if the subgraph (V_S, E_S) is a simple graph.

B. Problem Definitions

We next introduce an Integer Linear Programming (ILP) problem that is equivalent to the MWPF problem described in §II B. The ILP problem is reminiscent of the LP formulation used by the blossom algorithm. However, our ILP problem is formulated on the decoding hypergraph while the LP formulation in the blossom algorithm is on the syndrome graph.

ILP (Integer Linear Programming)

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e x_e & & (2) \\ \text{subject to} \quad & x_e \in \{0, 1\} & \forall e \in E \quad & (2a) \\ & \sum_{e \in \delta(S)} x_e \geq 1 & \forall S \in \mathcal{O} \quad & (2b) \end{aligned}$$

The constraints Eq. (2b) simply say that for each *Invalid* subgraph $S \in \mathcal{O}$, at least one edge of its hair $\delta(S)$ must appear in a parity factor. This is quite intuitive: an *Invalid* subgraph S must resort to at least one external edge $\delta(S)$ to satisfy the parity constraints posed by V_S.

We further prove that MWPF to ILP is an equivalent transformation with the following theorem in §A 1.

Theorem: min ILP = min MWPF. The optimal objective value of the two problems are equal.

Note that not every feasible ILP solution is a feasible parity factor. For example, $x_e = 1, \forall e \in E$ is feasible in ILP but not necessarily a feasible parity factor.

Next, we relax the integer constraints from $x_e \in \{0, 1\}$ to $x_e \in \mathbb{R}_+$ to derive a Linear Programming (LP) problem.

LP (Linear Programming)

$$\begin{aligned} \min \quad & \sum_{e \in E} w_e x_e & & (3) \\ \text{subject to} \quad & x_e \geq 0 & \forall e \in E \quad & (3a) \\ & \sum_{e \in \delta(S)} x_e \geq 1 & \forall S \in \mathcal{O} \quad & (3b) \end{aligned}$$

As a relaxation, we have $\min \text{LP} \leq \min \text{ILP}$. For some specific classes of hypergraphs, like simple graphs (§C) and nullity$_{\leq 1}$ hypergraphs (defined in §IV D), we prove that $\min \text{LP} = \min \text{ILP}$. For more general hypergraphs, we do not have a proof of equality, nor have we identified a counterexample despite extensive efforts to construct one analytically and numerically. Thus, we will not assume the equality throughout the paper.

We then consider the dual LP problem [25, 26], again drawing inspiration from the blossom algorithm (§II C).

DLP (Dual Linear Programming)

$$\begin{aligned} \max \quad & \sum_{S \in \mathcal{O}} y_S & & (4) \\ \text{subject to} \quad & y_S \geq 0 & \forall S \in \mathcal{O} \quad & (4a) \\ & \sum_{S \in \mathcal{O} | e \in \delta(S)} y_S \leq w_e & \forall e \in E \quad & (4b) \end{aligned}$$

Similar to $\vec{x}$, we use $\vec{y}$ to denote the vector of dual variables $y_S, \forall S \in \mathcal{O}$. We can visualize the dual variables on the decoding hypergraph as shown in Fig. 2.

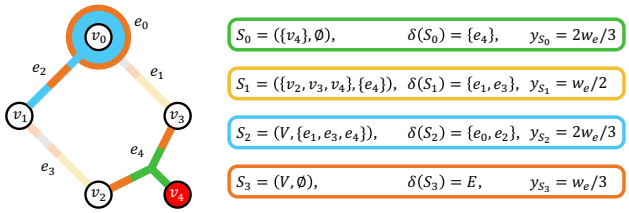
According to the Duality Theorem [27], $\max \text{DLP} = \min \text{LP}$. Thus, the optimal DLP solution has an objective value of $\sum_{S \in \mathcal{O}} y_S = \min \text{LP}$. Taken together, we have this chain of inequality:

$$\begin{aligned} & \text{(DLP)} & & & & \text{(MWPF)} \\ & \sum_{S \in \mathcal{O}} y_S \leq \min \text{LP} \leq \min \text{ILP} = \min \text{MWPF} \leq \sum_{e \in E} w_e x_e & & & & (5) \end{aligned}$$

Equality holds for the left and right $\leq$ when the DLP and MWPF solution are optimal, respectively.

C. More Important Concepts

Astute readers might notice that the cardinality $|\mathcal{O}|$ grows exponentially with $|V|$ and $|E|$. Fortunately, although the number of dual variables $|\mathcal{O}|$ is exponential,



(a) Visual of $\vec{y}$. (b) *Hyperblossoms* $\mathcal{B}$ and $y_S, \forall S \in \mathcal{B}$.

FIG. 2: Visualization of DLP solution $\vec{y}$ on the decoding hypergraph $G = (V, E)$ with a uniform edge weight of w_e. For each *Invalid* subgraph $S \in \mathcal{O}$, we visualize its dual variable y_S as colored segments occupying a y_S/w_e portion of each edge $e \in \delta(S)$. By definition, *Tight Edges* are those fully occupied, e.g., $T = \{e_0, e_2, e_4\}$ highlighted in solid color instead of the greyed colors.

there exists a trivial feasible DLP solution $y_S = 0, \forall S \in \mathcal{O}$. Thus, we only need to track those $y_S > 0$, as the blossom algorithm does.

Definition: Hyperblossom. A *Hyperblossom* is an *Invalid* subgraph S whose corresponding dual variable y_S is positive. We denote the set of all *Hyperblossoms* with $\mathcal{B} = \{S \in \mathcal{O} | y_S > 0\}$.

The notion of *Hyperblossom* is analogous to that of *blossom* in the blossom algorithm family, both representing positive dual variables (4a). We note that *Hyperblossoms* are defined on the decoding hypergraph, while *blossoms* are defined on the syndrome graph.

Definition: Tight Edge. A hyperedge $e \in E$ is *Tight* when its dual constraint (Eq. (4b)) takes the equal sign: $\sum_{S \in \mathcal{O} | e \in \delta(S)} y_S = w_e$. We use $T \subseteq E$ to represent the set of *Tight Edges*.

According to the Complementary Slackness Theorem [27], for any optimal MWPF $\vec{x}$ and DLP $\vec{y}$:

$$\sum_{S \in \mathcal{O} | e \in \delta(S)} y_S < w_e \implies x_e = 0, \qquad \forall e \in E \tag{C1}$$

That is, an optimal MWPF solution includes only *Tight Edges* of an optimal DLP solution.

Definition: Direction. A *Direction* $\Delta\vec{y}$ updates the DLP solution $\vec{y}$ by $\vec{y}' := \vec{y} + l\Delta\vec{y}$ given a length $l > 0$.

Definition: Feasible Direction. A *Direction* $\Delta\vec{y}$ is called *Feasible* when it can grow a small positive length without violating any DLP constraint. We denote a *Feasible Direction* with $\Delta\vec{y}[G, \mathcal{B}, T]$. When the context is clear, we elide G, $\mathcal{B}$, T and write $\Delta\vec{y}$ or $\Delta\vec{y}[T]$ for $\Delta\vec{y}[G, \mathcal{B}, T]$.

$$\forall S \in \mathcal{O} \setminus \mathcal{B}, \quad \Delta y_S \geq 0 \tag{6a}$$

$$\forall e \in T, \quad \sum_{S \in \mathcal{O} | e \in \delta(S)} \Delta y_S \leq 0 \tag{6b}$$

Eq. (6a) says that a *Feasible Direction* will not decrease dual variables that are already 0. Eq. (6b) says that it will not over-grow a *Tight Edge*.

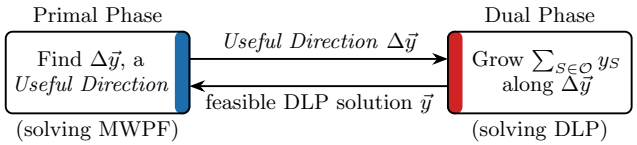


FIG. 3: Overview of the HyperBlossom algorithm. The Primal phase solves the MWPF problem, while the Dual phase solves the DLP problem. They exchange information through a narrow interface. This interaction is inspired by the blossom algorithm.

Definition: Useful Direction. A *Feasible Direction* $\Delta\vec{y}$ is *Useful* if $\sum \Delta\vec{y} = \sum_{S \in \mathcal{O}} \Delta y_S > 0$.

Applying a *Useful Direction* for a positive length will improve a suboptimal DLP solution.

Definition: Trivial Direction. A *Feasible Direction* $\Delta\vec{y}$ is *Trivial* if $\Delta\vec{y} = \{\Delta y_S : +1\}$ where $S \in \mathcal{O}$.

Applying a *Trivial Direction* will only grow one *Invalid* subgraph S while leaving others unchanged. We have $\delta(S) \cap T = \varnothing$ because $\Delta\vec{y}$ is *Feasible* and must satisfy Eq. (6b). By definition, a *Trivial Direction* is also a *Useful Direction*.

D. HyperBlossom Algorithm Overview

Built on top of the concepts and problem definitions introduced so far, the HyperBlossom algorithm features three levels of innovations. At the highest level, drawing inspiration from the blossom algorithm, HyperBlossom consists of two phases: the Primal phase works on the MWPF problem as formulated in §II B; the Dual phase works on the DLP problem formulated in §III B. The two phases exchange information using a narrow interface, as shown in Fig. 3.

The Primal phase maintains parity factors consisting of *Tight Edges* and informs the Dual phase new *Useful Directions* $\Delta\vec{y}$ to find better DLP solutions. The Dual phase tracks the *Hyperblossoms* and grows them along the *Useful Directions* $\Delta\vec{y}$ from the Primal phase, and informs the Primal phase the improved DLP solution $\vec{y}$.

According to Eq. (5), given a feasible DLP solution $\vec{y}$ and a feasible MWPF solution (parity factor) $\vec{x}$, their objective values are the lower and upper bounds, respectively, of that of the optimal MWPF solution. We name $\sum_{e \in E} w_e x_e - \sum_{S \in \mathcal{O}} y_S$ the *primal-dual gap*. The HyperBlossom algorithm seeks to reduce this gap and get closer to optimality. Especially, when the primal-dual gap is zero, the optimality is certified by the following theorem.

Theorem: Certifying Optimum. A feasible DLP $\vec{y}$ with the same objective value as the weight of a parity

TABLE I: Relationship between the Blossom Algorithm (§II C) and the HyperBlossom Algorithm

	Blossom [3, 4, 22]	HyperBlossom
Problem Solved	MWPM on syndrome graph	MWPF on decoding hypergraph
Primal Variables	x_e^* per syndrome graph edge $e \in E^*$	x_e per decoding hypergraph edge $e \in E$
Primal Constraints (Dual Variables)	$\mathcal{O}^* = \left\{ S^* \subseteq D \;\middle|\; \lvert S^* \rvert = 1 \mod 2 \right\}$	$\mathcal{O} = \{S = (V_S, E_S) \mid V_S \subseteq V, E_S \subseteq E[V_S], \forall \mathcal{E} \subseteq E_S, \mathcal{D}(\mathcal{E}) \neq D \cap V_S\}$
(Hyper)Blossoms	*Blossoms* $\mathcal{B}^* = \{S^* \in \mathcal{O}^* \mid y_{S^*}^* > 0\}$	*Hyperblossoms* $\mathcal{B} = \{S \in \mathcal{O} \mid y_S > 0\}$
Dual Updates	along *Direction* $\Delta\vec{y^*} \in \{0, +1, -1\}^{\lvert\mathcal{O}^*\rvert}$	along *Direction* $\Delta\vec{y} \in \mathbb{R}^{\lvert\mathcal{O}\rvert}$
Find more (Hyper)Blossoms	alternating tree [22]	*relaxing* (§III E) (§IV)

factor $\mathcal{E}$ certifies the optimality of $\mathcal{E}$ as an MWPF.

$$W(\mathcal{E}) = \sum_{S \in \mathcal{O}} y_S \quad \Longrightarrow \quad \mathcal{E} \text{ is an MWPF}$$

Note that in the blossom algorithm, the Primal and Dual phases also work on two related problems: ILP formulation of MWPM and the dual of a relaxed LP problem of the ILP problem (DLP), respectively [4]. However, because the ILP and DLP problems have the same objective value, the blossom algorithm always finds the optimal solution for the MWPM problem. In contrast, the HyperBlossom algorithm is subject to the inequality chain of Eq. (5): there might exist cases where $\min \text{LP} < \min \text{ILP}$. Only when the **Certifiability Condition** below holds, the HyperBlossom algorithm guarantees to certify the MWPF solution.

Certifiability Condition: $\min \text{LP} = \min \text{ILP}$

We prove that two classes of decoding hypergraphs satisfy the above condition: simple graphs and nullity$_{\leq 1}$ hypergraphs (whose incidence matrix has a nullity of 0 or 1). We prove the theorems below in §C 4 and §IV D, respectively.

Theorem: Simple Graphs $\min \text{LP} = \min \text{ILP}$.

Theorem: Nullity$_{\leq 1}$ Hypergraphs $\min \text{LP} = \min \text{ILP}$.

At the second level, HyperBlossom features two important optimizations. *Relaxing* (§III E) turns the search for an optimal DLP solution into a sequence of calls to a relaxer finding algorithm. *Clustering* (§III F) exploits spatial locality in defects to divide the decoding hypergraph into clusters and conquer them independently. The Primal phase implements *relaxing* because it is responsible for computing the *Direction*. Both phases are aware of the clusters, using cluster-oriented data structures.

At the third and lowest level, HyperBlossom features several *Relaxer*-finding algorithms (§IV). We show that the blossom algorithm (§II C) can be derived from one of those *Relaxer*-finding algorithms (§IV C), demonstrating the power of *relaxing*. Importantly, we propose a new, simple *Relaxer*-finding algorithm called the *SingleHair* (§IV A) for general hypergraphs. We analytically prove the optimality of the *SingleHair* algorithm in certain conditions (§B 5) and also show the limitations of it (§B 3 and §B 4). Empirically, we show that the *SingleHair* algorithm is an accurate and fast *Relaxer*-finding algorithm for a variety of qLDPC codes (§VI).

We compare the blossom algorithm and the HyperBlossom algorithm in Table I.

E. Relaxing

The Primal phase helps the Dual phase by finding a *Useful Direction* so that the latter can improve its current solution toward optimality, as shown in Fig. 3. The challenge is that the dimensionality of $\vec{y}$ or $\Delta\vec{y}$ is $|\mathcal{O}|$: there are too many dual variables. *Relaxing* converts the problem of finding a *Useful Direction* into finding *Relaxers*, as defined below.

Definition: Relaxer. Given a decoding hypergraph G, its *Tight Edges* T and *Hyperblossoms* $\mathcal{B}$, a *Relaxer* $R[G, \mathcal{B}, T]$ is a *Feasible Direction* $\Delta\vec{y}$ that satisfies the following conditions. (7a) Applying it relaxes a non-empty set of *Tight Edges* $\mathcal{R}(R) \subseteq T$. That is, these edges will no longer be tight after growing along $\Delta\vec{y}$. (7b) It does not reduce the DLP objective. When the context is clear, we elide $G, \mathcal{B}, T$ and write R or $R[T]$ for $R[G, \mathcal{B}, T]$.

$$\mathcal{R}(R) \neq \varnothing, \quad \forall e \in \mathcal{R}(R) \subseteq T, \sum_{S \in \mathcal{O} | e \in \delta(S)} \Delta y_S < 0 \tag{7a}$$

$$\sum_{S \in \mathcal{O}} \Delta y_S \geq 0 \tag{7b}$$

Applying a *Relaxer* reduces the number of *Tight Edges*. As a result, it reduces the search space of the MWPF problem. Importantly, it also helps find a *Useful Direction* with the following theorem proved in §A 2:

Theorem: Relaxing. Given a suboptimal DLP $\vec{y}$, there exists a *Relaxer* or a *Trivial Direction*, or both.

A simple algorithm to compute the *Trivial Direction* can iterate over $S \in \mathcal{O}$ and find one with $\delta(S) \cap T = \varnothing$. As noted earlier, $\Delta\vec{y} = \{\Delta y_S : +1\}$ is *Feasible* (and therefore *Trivial*). Later, we will present a much more efficient algorithm in Algorithm 8, line 7.

Applying the theorem, the Primal Phase will find a sequence of *Relaxers* $R_i[T_i]$, $i = 1, 2, ..., n$, before it could no longer find any *Relaxer*. $T_i = T \setminus (\cup_{j=1}^{i-1} \mathcal{R}_j)$ is the set of *Tight Edges* after applying the first $(i-1)$ *Relaxers*. At this point, according to the theorem, the Primal phase can find a *Trivial Direction* $\Delta\vec{y}[T_{n+1}]$ for the Dual phase. Since a *Trivial Direction* is *Useful* by definition, the theorem guarantees that the Dual phase makes progress via *relaxing*. Because each *Relaxer* reduces the number of *Tight Edges* by at least 1, we have $|T_{n+1}| < |T_n| < ... < |T_1| = |T|$. That is, n is upper bounded by $|T|$. In the end, the Primal phase will find a *Trivial Direction* for the Dual phase to make progress after at most $|T|$ *Relaxers*.

1. Dual Phase Optimization

Theorem: Relaxing, however, does not guarantee the Dual phase terminates. To tackle this, we introduce an optimization in the Dual phase. Instead of growing dual variables strictly along $\Delta\vec{y}$, the Dual phase infers from $\Delta\vec{y}$ what new *Hyperblossoms* will be formed if this *Direction* is applied. It then calls an LP solver to solve the DLP problem only using dual variables corresponding to *Hyperblossoms* ever observed as far, called *History*.

Definition: History. The *History* $\mathcal{B}^H$ includes all *Hyperblossoms* that have formed so far during the Dual phase operation.

When using an LP solver to solve the DLP problem, the Dual phase fixes the dual variables corresponding to $\mathcal{O} \setminus \mathcal{B}^H$ to 0. As the LP solver always finds a DLP solution that maximizes $\sum_{S \in \mathcal{B}^H} y_S$, further progress by the Dual phase according to **Theorem: Relaxing** must introduce new *Hyperblossoms*. That is, each time the Dual phase receives a *Useful Direction*, $|\mathcal{B}^H|$ increases while $|\mathcal{O} \setminus \mathcal{B}^H|$ decreases. Therefore, the Dual phase must terminate with at most $|\mathcal{O}|$ *Useful Directions* from the Primal phase.

2. Batch Relaxing

Applying a *Relaxer* $R = \Delta\vec{y}$ is expensive because (i) it will convert $S \in \mathcal{O}$ into a *Hyperblossom* if $\Delta y_S > 0$ and therefore increase $|\mathcal{B}^H|$; and (ii) the time complexity of the LP solver is cubic in the number of the *History* $|\mathcal{B}^H|$. On the other hand, **Theorem: Relaxing** requires multiple *Relaxers* to be applied before a *Trivial Direction* can be found. Fortunately, with *batch relaxing*, the Primal phase no longer needs to send each of $R_i[T_i]$, $i = 1, 2, ..., n$ and the *Trivial Direction* $\Delta\vec{y}[T_{n+1}]$ to the Dual phase to apply. Rather, it can compute a *Useful Direction* $\Delta'\vec{y}[T]$ from $R_i[T_i]$ and $\Delta\vec{y}[T_{n+1}]$ so that the Dual phase only needs to grow once, instead $n+1$ times, and will grow the *History* $|\mathcal{B}^H|$ only if necessary.

Algorithm 1 Compose Relaxers

Input: $R'_i[T], i = 1, 2, ..., n$ and $\Delta\vec{y}[T \setminus (\cup_i \mathcal{R}'_i))]$
Output: $\Delta'\vec{y}[T]$ in **Theorem: Batch Relaxing**

1: **procedure** COMPOSE($\{R'_i[T]\}, \Delta\vec{y}$)
2: $\quad\Delta'\vec{y} \leftarrow \Delta\vec{y}$
3: $\quad$**for** $e \in \cup_i \mathcal{R}'_i \cap T$ **do**
4: $\quad\quad\alpha \leftarrow \sum_{S \in \mathcal{O} | e \in \delta(S)} \Delta' y_S$
5: $\quad\quad$**if** $\alpha > 0$ **then** ▷ if $\Delta\vec{y}$ violates (4b) of e
6: $\quad\quad\quad\Delta^e\vec{y} \leftarrow R'_k$ where $e \in \mathcal{R}'_k$, $1 \le k \le n$
7: $\quad\quad\quad\Delta'\vec{y} \leftarrow \Delta'\vec{y} - \frac{\alpha}{\sum_{S \in \mathcal{O} | e \in \delta(S)} \Delta^e y_S} \Delta^e\vec{y}$ ▷ fix violation
8: $\quad$**return** $\Delta'\vec{y}$

Algorithm 2 Batched Relaxing

Input: $G = (V, E)$ (decoding hypergraph), $\vec{y}$ (DLP solution)
Output: $\{R'[T]\}$ (a set of at most $|T|$ *Relaxers* that maximally relaxes *Tight Edges* T)

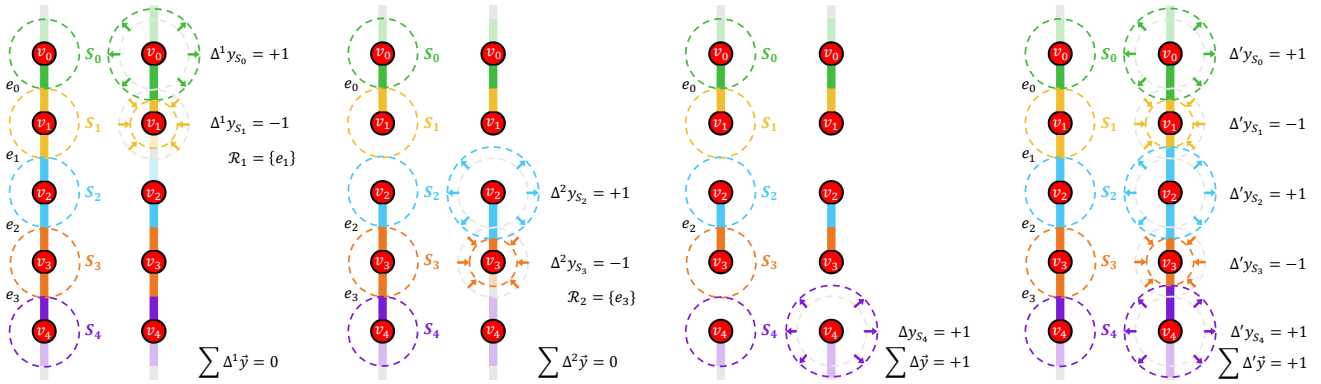
1: **procedure** BATCHEDRELAXING($G, \vec{y}$)
2: $\quad Rs' \leftarrow \varnothing$ ▷ set of *Relaxers*
3: $\quad T' \leftarrow T$ ▷ remaining *Tight Edges* T_i
4: $\quad$**while** $(R[T'] \leftarrow$ FINDRELAXER$(G, \mathcal{B}, T')) \neq$ NIL **do**
5: $\quad\quad R'[T] \leftarrow$ COMPOSE$(Rs', R[T'])$
6: $\quad\quad Rs' \leftarrow Rs' \cup \{R'\}$
7: $\quad\quad T' \leftarrow T' \setminus \mathcal{R}'$
8: $\quad$**return** Rs'

Theorem: Batch Relaxing. Given *Relaxers* $R'_i[T], i = 1, ..., n$, if there exists a *Feasible Direction* $\Delta\vec{y}[T \setminus (\cup_i \mathcal{R}'_i)]$, we can compose a *Feasible Direction* $\Delta'\vec{y}[T]$ such that $\sum \Delta'\vec{y} \ge \sum \Delta\vec{y}$. In the special case where $\Delta\vec{y}$ is a *Relaxer*, $\Delta'\vec{y}$ is also a *Relaxer* $R'[T]$ such that $\mathcal{R}' \supseteq \mathcal{R}$.

Algorithm 1 describes a method to compose the *Feasible Direction* $\Delta'\vec{y}[T]$ and therefore, serves as a constructive proof of the theorem.

We next describe how the Primal phase computes a *Useful Direction* $\Delta'\vec{y}$ from $R_i[T_i], i = 1, 2, ..., n$, and $\Delta\vec{y}$. First, it computes $R'_i[T], i = 1, 2, ..n$ from $R_i[T_i]$ such that $R'_i[T]$ relaxes all the edges that $R_i[T_i]$ relaxes. Because $T_1 = T$, the first *Relaxer* $R'_1[T]$ is simply $R_1[T_1]$. Assuming we have all the *Relaxers* $R'_j[T]$, $1 \le j < i$, we can compose $R'_i[T]$ that relaxes $\mathcal{R}'_i \supseteq \mathcal{R}_i$, according to **Theorem: Batch Relaxing**. Overall, we have $(\cup_i \mathcal{R}_i) \subseteq (\cup_i \mathcal{R}'_i)$. Algorithm 2 is a pseudo code of the above process, by repeatedly finding *Relaxers* $R_i[T_i]$ (line 4) and computing $R'_i[T]$ (line 5).

Second, the Primal phase computes a *Useful Direction* $\Delta'\vec{y}[T]$ from *Relaxers* $R'_i[T], i = 1, 2, ..., n$ and the *Trivial Direction* $\Delta\vec{y}[T \setminus (\cup_i \mathcal{R}_i)]$. Because $T \setminus (\cup_i \mathcal{R}'_i) \subseteq T \setminus$



(a) *Relaxer* $R_1[T_1] = \Delta^1\vec{y}$ (b) *Relaxer* $R_2[T_2] = \Delta^2\vec{y}$ (c) *Trivial Direction* $\Delta\vec{y}[T_3]$ (d) *Useful Direction* $\Delta'\vec{y}[T]$

FIG. 4: An example of *batch relaxing*. The radius of the circle centered at a vertex v_i represents the corresponding dual variable $y_S, S = (\{v\}, \varnothing)$. (a) The initial DLP solution has *Tight Edges* $T_1 = T = \{e_0, e_1, e_2, e_3\}$. We find a *Relaxer* $R_1[T_1]$ that increases y_{S_0}, decreases y_{S_1}, and leaves others unchanged. We have $\mathcal{R}_1 = \{e_1\}$. (b) Given the remaining *Tight Edges* $T_2 = T_1 \setminus \mathcal{R}_1 = \{e_0, e_2, e_3\}$, we find a *Relaxer* $R_2[T_2]$ with $\mathcal{R}_2 = \{e_3\}$. (c) For the remaining *Tight Edges* $T_3 = T_2 \setminus \mathcal{R}_2 = \{e_0, e_2\}$, we find a *Trivial Direction* $\Delta\vec{y}[T_3]$ to increase the dual sum: growing y_{S_4}. (d) We compose these *Directions* $R_1[T_1]$, $R_2[T_2]$ and $\Delta\vec{y}[T_3]$ into a single *Useful Direction* $\Delta'\vec{y}[T]$ using **Theorem: Batch Relaxing** (Algorithm 1).

$(\cup_i \mathcal{R}_i)$, $\Delta\vec{y}$ is also *Feasible* on $T \setminus (\cup_i \mathcal{R}'_i)$. With *Relaxers* $R'_i[T], i = 1, 2, ..., n$, the Primal phase can compose a *Feasible Direction* $\Delta'\vec{y}[T]$ such that $\sum \Delta'\vec{y} \geq \sum \Delta\vec{y} > 0$, according to **Theorem: Batch Relaxing** (Algorithm 1). That is, $\Delta'\vec{y}$ is a *Useful Direction*.

In order to keep the *History* $\mathcal{B}^H$ small, Algorithm 1 takes special care so that applying the *Useful Direction* $\Delta'\vec{y}[T]$ will not create unnecessary *Hyperblossoms* in the Dual phase. Initially (line 2), the *Trivial Direction* $\Delta'\vec{y}$ would introduce a single *Hyperblossom* S because $\Delta\vec{y} = \{\Delta y_S : +1\}$. However, $\Delta\vec{y}$ is *Feasible* on $[T \setminus (\cup_i \mathcal{R}_i)]$, not on T. Applying it on T may violate Eq. (4b) for some edges in $\cup_i \mathcal{R}_i \cap T$. The algorithm fixes this violation using a for-loop (line 3) over all edges in $\cup_i \mathcal{R}_i \cap T$ and include a *Relaxer* to relax each (lines 6 and 7). Instead of relaxing all edges $\cup_i \mathcal{R}_i \cap T$, Eq. (A2) only relax e if e will violate Eq. (4b) (line 5), and includes the *Hyperblossoms* of this *Relaxer* in $\Delta'\vec{y}$. As a result, the output $\Delta'\vec{y}[T]$ usually uses a small number of *Relaxers* from $\{R'_i\}$ and will only create new *Hyperblossoms* from them. We note that our implementation actually employs an even more sophisticated algorithm whose output creates even fewer *Hyperblossoms* and therefore achieves higher performance.

We show an example of *batch relaxing* in Fig. 4.

F. Clustering

When the physical error rate is sufficiently low $p \ll 1$, a defect vertex is likely to be caused by a single incident hyperedge. Many have exploited this property of locality to speed up decoding [3, 4, 6, 7, 20, 28, 29]. They group defect vertices into clusters and seek to find parity factors within the same cluster first. For example, in the blossom algorithm [22], alternating trees and matched pairs can be considered as clusters, while the (Hypergraph) Union-Find decoders [6, 7] explicitly define clusters. We formally define the concept of clusters and prove that the *clustering* technique preserves global optimality under certain conditions.

The HyperBlossom framework organizes vertices, *Tight Edges* and *Hyperblossoms* into *Clusters*. Both the Primal and Dual phases operate on the *Clusters* and the interface in Section III D becomes per-*Cluster*.

Definition: Cluster. A *Cluster* $C = (V_C, E_C)$ is a subgraph of the decoding hypergraph defined as:

- A defect vertex $v \in D$ not connected by any *Tight Edge* is a *Cluster* $C = (\{v\}, \varnothing)$.
- A maximally connected subgraph of *Tight Edges* $E_C \subseteq T$ is a *Cluster* $C = (\cup_{e \in E_C} e, E_C)$.
- A *Hyperblossom* $S \in \mathcal{B}$ spanning *Clusters* $\mathcal{C}_S = \{C \in \mathcal{C} | V_S \cap V_C \neq \varnothing\}$ merges them into a single *Cluster* $C' = (\cup_{C \in \mathcal{C}_S} V_C \cup V_S, \cup_{C \in \mathcal{C}_S} E_C)$.

The *Hyperblossoms* of a *Cluster* $C \in \mathcal{C}$ is the set of *Hyperblossoms* whose vertices overlap with that of the *Cluster*, represented by $\mathcal{B}_C = \{S \in \mathcal{B} | V_S \cap V_C \neq \varnothing\}$. According to the definition of *Clusters*, the *Hyperblossoms* of different *Clusters* do not overlap, i.e., $\forall C_1, C_2 \in \mathcal{C}, C_1 \neq C_2 \implies \mathcal{B}_{C_1} \cap \mathcal{B}_{C_2} = \varnothing$.

By definition, no two *Clusters* share any vertex, *Tight Edge* or *Hyperblossom*. A *Cluster* only includes *Tight Edges*, i.e., $E_C \subseteq T$. Fig. 5 illustrates a few examples of *Clusters*.

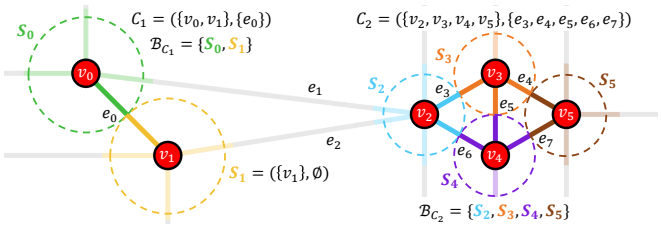


FIG. 5: An example of *clustering*. Two *Clusters* C_1 and C_2 are not connected by any *Tight Edge* (solid color), so they can be solved independently. The two *Clusters* merge into one if either e_1 or e_2 becomes tight.

Definition: Locally Optimal Cluster. A *Cluster* C is called *locally optimal* if there exists a parity factor $\mathcal{E}_C \subseteq E_C$ such that $\mathcal{D}(\mathcal{E}_C) = D \cap V_C$ and $W(\mathcal{E}_C) = \sum_{S \in \mathcal{B}_C} y_S$.

Theorem: Optimality Criteria of Clusters. When all the *Clusters* are *Locally Optimal Clusters*, the union of all the local MWPF and DLP solutions are the global optimal MWPF $\mathcal{E} = \cup_{C \in \mathcal{C}} \mathcal{E}_C$ and global optimal DLP solution $\vec{y} = \bigcup_{C \in \mathcal{C}} \{S : y_S | S \in \mathcal{B}_C\}$, respectively.

With the above criteria proved in §A 3, we can use Algorithm 3 to determine whether every *Cluster* is a *Locally Optimal Cluster*. If not, we can optimize each suboptimal *Cluster* individually by improving and merging the *Clusters* using Algorithm 4. Once all the *Clusters* are *Locally Optimal Clusters*, we then use Algorithm 5 to find the global optimal solution. In the special case where the **Certifiability Condition** is satisfied, HyperBlossom framework can find optimal MWPF and DLP solutions by operating on each *Cluster* individually (§III G).

With the *clustering* technique, we can design optimal decoders that adaptively consider larger *Clusters* only when necessary. This has the potential to achieve an average runtime almost-linear to the number of defects $|D| \propto p|V|$, like those fast MWPM decoders [3, 4, 20, 29]. As illustrated in Fig. 5, defect vertices remain isolated until a *Tight Edge* connects them, at which point they are merged into a single *Cluster*. The *clustering* technique does not improve the worst-case time complexity because a *Cluster* may span the entire decoding hypergraph

G. HyperBlossom Algorithm

The HyperBlossom algorithm combines *relaxing* (§III E) and *clustering* (§III F) as described by Algorithm 6. Specifically, it applies *batch relaxing* (Algorithm 2) to each *Cluster*. Using a stronger version of **Theorem: Relaxing**, it is able to find a *Trivial Direction* much more efficiently, as proved in §A 4.

Theorem: Relaxing with Clusters. Given a suboptimal DLP solution $\vec{y}$, there exists a *Relaxer* or an *Invalid Cluster*, or both.

Algorithm 3 Check Cluster Local Optimality

Input: C (*Cluster*)
Output: whether C is locally optimal
1: **procedure** IsLocallyOptimal(C)
2: **if** $C \in \mathcal{O}$ **then** ▷ *Invalid Cluster*
3: **return** False
4: $\mathcal{E}_C \leftarrow$ MWPF of subgraph $C = (V_C, E_C)$
5: **if** $W(\mathcal{E}_C) = \sum_{S \in \mathcal{B}_C} y_S$ **then**
6: **return** True
7: **return** False

Algorithm 4 Merge Clusters

Input: $\mathcal{C}$ (*Clusters*), $\mathcal{B}$ (*Hyperblossoms*), T (*Tight Edges*)
Output: $\mathcal{C}$ (merged *Clusters*)
1: **procedure** Merge($\mathcal{C}, \mathcal{B}, T$)
2: **while** $\exists C \in \mathcal{C}, \exists e \in T, e \cap V_C \neq \varnothing, e \nsubseteq V_C$ **do**
3: $V_C \leftarrow V_C \cup e$ ▷ merge vertices into *Clusters*
4: ▷ no need to merge $\mathcal{B}$ because a per-*Cluster Relaxer* finder only finds *Hyperblossom* $S = (V_S \subseteq V_C, E_S \subseteq E_C)$
5: **while** $\exists C_1, C_2 \in \mathcal{C}, V_{C_1} \cap V_{C_2} \neq \varnothing$ **do**
6: $\mathcal{C} \leftarrow \mathcal{C} \cup \{(V_{C_1} \cup V_{C_2}, E_{C_1} \cup E_{C_2})\} \setminus \{C_1, C_2\}$ ▷ merge two *Clusters*
7: **for** $C \in \mathcal{C}$ **do**
8: $E_C \leftarrow E[V_C] \cap T$
9: $\mathcal{B}_C \leftarrow \{S \in \mathcal{B} | V_S \cap V_C \neq \varnothing\}$
10: **return** $\mathcal{C}$

Algorithm 5 Optimal Solutions from Clusters

Input: $\mathcal{C}$ (*Clusters*)
Output: $\mathcal{E}$, $\vec{y}$ (optimal MWPF and DLP solutions when all the *Clusters* are locally optimal checked by Algorithm 3)
1: **procedure** OptimalSolutions($\mathcal{C}$)
2: $\mathcal{E} \leftarrow \varnothing$
3: $\vec{y} \leftarrow \varnothing$ ▷ default $y_S = 0$ for all $S \in \mathcal{O}$
4: **for** $C \in \mathcal{C}$ **do**
5: $\mathcal{E}_C \leftarrow$ MWPF of subgraph $C = (V_C, E_C)$
6: $\mathcal{E} \leftarrow \mathcal{E} \cup \mathcal{E}_C$
7: $\vec{y} \leftarrow \vec{y} \cup \{S : y_S | S \in \mathcal{B}_C\}$
8: **return** $\mathcal{E}, \vec{y}$

Using the above theorem, the Primal phase, described by Algorithm 8, can easily find a *Trivial Direction* by checking whether the *Cluster* is still *Valid* after relaxing a maximal set of edges in line 7. In the special case where the decoding hypergraph and all its subgraphs satisfy the **Certifiability Condition**, Algorithm 6 finds the optimal MWPF and DLP solutions, according to the following theorem proved in §A 4.

Theorem: HyperBlossom Algorithm Optimality. There exists a *Relaxer* finder so that the HyperBlossom algorithm equipped with it would find optimal MWPF and DLP solutions, if the decoding hypergraph and all its subgraphs satisfy $\mathbf{min\,LP = min\,ILP}$.

We note that when a property is true for a graph and all its subgraphs, the property is called *hereditary* [30]. For example, being "simple" is a hereditary property because any subgraph of a simple graph is also a simple graph. Given **Theorem: Simple Graphs min LP = min ILP**, **min LP = min ILP** is a hereditary property of any simple graph.

Nullity$_{\le 1}$ is also a hereditary property of hypergraphs, as proved in §B 5. We define a *nullity*$_{\le 1}$ *hypergraph* as a hypergraph whose incidence matrix M_I has a nullity of at most 1, i.e., its null space ($M_I\vec{x} = 0$) has a dimensionality of at most 1. Thus, the HyperBlossom algorithm is optimal for both simple graphs and nullity$_{\le 1}$ hypergraphs.

The worst-case time complexity of the algorithm depends on the *Relaxer*-finding algorithm and is exponential with regard to the size of the decoding graph. There are at most $|\mathcal{O}|$ calls of PrimalPhase (line 3), each incrementing the number of *History* $|\cup_{C\in\mathcal{C}} \mathcal{B}_C^H|$ by at least one. Within each call, we need to find at most $|T| \le |E|$ *Relaxers* (line 5). Suppose the *Relaxer*-finding algorithm runs in $O(F(|V|, |E|))$ time, then the time complexity of finding a *Useful Direction* is $O(|E|F(|V|, |E|))$. We then need to run the LP solver for each iteration, consuming $O(|\mathcal{O}|^{1.5}|V|^2)$ time [31] given $O(|\mathcal{O}|)$ variables and $O(|V|)$ constraints. Overall, the time complexity of Algorithm 6 is $O(|\mathcal{O}|^{2.5}|V|^2 + |\mathcal{O}||E|F(|V|, |E|))$.

IV. RELAXER-FINDING ALGORITHMS

With the *relaxing* (§III E) and *clustering* (§III F) techniques, we reduce the MWPF decoding problem to *Relaxer*-finding algorithms that find *Relaxers* for each *Cluster* $C \in \mathcal{C}$. We present four *Relaxer*-finding algorithms or *Relaxer* finders that make different tradeoffs in generality, speed, and accuracy, as illustrated by Fig. 6. Here "generality" refers to an algorithm's ability for handling different types of decoding hypergraphs, e.g., algorithms that only works on simple graphs are less general.

- *SingleHair* (§IV A) finds *Relaxers* on general hypergraphs, which is an efficient algorithm that runs in $O(\text{poly}(|V_C| + |E_C| + |\mathcal{B}_C|))$.
- *UnionFind* (§IV B) does not find any *Relaxer*, which is efficient but less accurate.
- *Blossom* (§IV C) is efficient and optimal but only works for simple graphs.
- *Nullity*$_{\le 1}$ (§IV D) is efficient and optimal on a class of decoding hypergraphs whose incidence matrix has a nullity of 0 or 1.

We say a *Relaxer*-finding algorithm is *optimal* if the HyperBlossom algorithm equipped with it finds an optimal DLP solution. To prove a *Relaxer* finder is optimal, it suffices to prove that the *Relaxer* finder always finds a *Relaxer* for a suboptimal DLP solution or returns Nil when there exists an *Invalid Cluster*, according to **Theorem: Relaxing with Clusters**. For the special classes of hypergraphs that they are intended to work, *Blossom* and *Nullity*$_{\le 1}$ are optimal. *SingleHair* and *UnionFind* work for general hypergraphs but are not optimal. In §B, we compare *SingleHair* with the less general but optimal ones, i.e., *Blossom* and *Nullity*$_{\le 1}$, to reveal its lim-

Algorithm 6 HyperBlossom Algorithm

Input: G (decoding hypergraph), D (defects)
Output: $\mathcal{E}, \vec{y}$ (a pair of optimal MWPF and DLP solutions if **min LP = min ILP**; otherwise, feasible solutions)

```
1: procedure HyperBlossom(G, D)
2:     C ← InitializeDualPhase(G, D)
3:     while (C, Δy⃗ ← PrimalPhase(C)) ≠ Nil do
4:         C ← DualPhase(C, Δy⃗)
5:     E, y⃗ ← OptimalSolutions(C)                ▷ Algorithm 5
6:     return E, y⃗
```

Algorithm 7 HyperBlossom Dual Phase

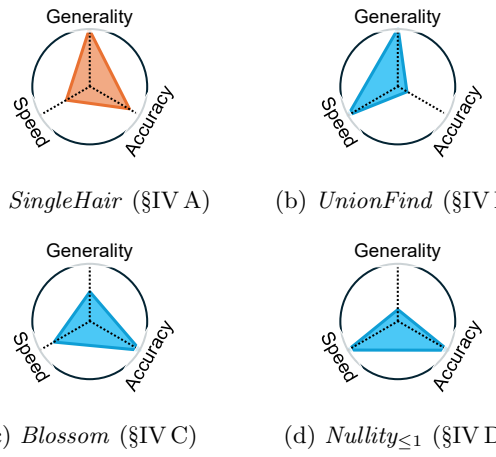
Input: G (decoding hypergraph), D (defects)
Output: $\mathcal{C}$ (*Clusters*)

```
1: procedure InitializeDualPhase(G, D)
2:     y⃗ ← 0                                     ▷ initial DLP solution
3:     C ← {({v}, ∅)|v ∈ D}
4:     ∀C ∈ C, B_C^H ← {(V_C, ∅)}                ▷ History
5:     T ← {e ∈ E|w_e = 0}                       ▷ initial Tight Edges
6:     C ← Merge(C, ∅, T)                        ▷ initial Clusters
7:     return C
8: procedure DualPhase(C, Δy⃗)                   ▷ update a Cluster
9:     B_C^H ← B_C^H ∪ {S ∈ O|Δy_S > 0}
10:    optimize partial DLP of {y_S|S ∈ B_C^H}
11:    B ← {S ∈ O|y_S > 0}                       ▷ Hyperblossoms
12:    T ← {e ∈ E| Σ_{S∈O|e∈δ(S)} y_S = w_e}     ▷ Tight Edges
13:    C ← Merge(C, B, T)                        ▷ Algorithm 4
14:    return C
```

Algorithm 8 HyperBlossom Primal Phase

Input: $\mathcal{C}$ (*Clusters* $C \in \mathcal{C}$ with their *Hyperblossoms* $\mathcal{B}_C$ and *Tight Edges* E_C)
Output: A locally suboptimal *Cluster* C and a *Useful Direction* $\Delta\vec{y}$; if not found, return Nil

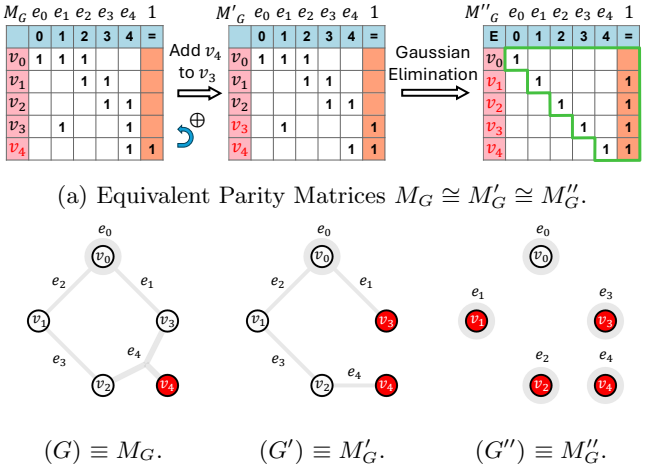
```
1: procedure PrimalPhase(C)
2:     for C ∈ C do
3:         if IsLocallyOptimal(C) then           ▷ (Algorithm 3)
4:             continue                          ▷ skip locally optimal Clusters
5:         Rs ← BatchedRelaxing(C, B_C, E_C)     ▷ (Algo. 2)
6:         E'_C ← E_C \ ∪_{R_i∈Rs} R_i            ▷ remaining Tight Edges
7:         if (V_C, E'_C) ∈ O then
8:             S ← any Invalid subgraph of (V_C, E'_C)
9:             Δy⃗ ← {Δy_S : +1}                  ▷ Trivial Direction
10:            return C, Compose(Rs, Δy⃗)          ▷ (Algorithm 1)
11:    return Nil
```



(a) *SingleHair* (§IV A) (b) *UnionFind* (§IV B)

(c) *Blossom* (§IV C) (d) *Nullity*$_{\leq 1}$ (§IV D)

FIG. 6: Tradeoffs of different *Relaxer* finders. Generality, speed and accuracy cannot be achieved simultaneously given the NP-hardness of MWPF [5].



(a) Equivalent Parity Matrices $M_G \cong M'_G \cong M''_G$.

$(G) \equiv M_G$. $(G') \equiv M'_G$. $(G'') \equiv M''_G$.

FIG. 7: Examples of equivalent parity matrices $M_G \cong M'_G \cong M''_G$ and their corresponding decoding hypergraphs $G \cong G' \cong G''$. We construct M'_G by adding row v_4 to row v_3 of M_G. The same transformation can be understood as adding the vertex v_4 to v_3 in the decoding hypergraph G while updating the adjacent hyperedges of v_3. Applying Gauss-Jordan elimination on M'_G results in an *Echelon Matrix* M''_G and a simplified decoding hypergraph G''.

itations and potential improvements, which may inspire future research into better *Relaxer*-finding algorithms.

Note that we describe the *Relaxer* finders using the language of the decoding hypergraph $(G, \mathcal{B}, T)$ for simplicity (See Algorithm 2 line 4). With the *clustering* technique, these *Relaxer* finders naturally apply to each *Cluster* $C \in \mathcal{C}$ individually, by passing in $(C, \mathcal{B}_C, E_C)$ as the input (See Algorithm 8 line 5).

Moreover, *relaxing* allows each time one or more *Relaxer* finders to be used, i.e., at line 4 of Algorithm 2. Therefore, the HyperBlossom framework can easily combine different *Relaxer* finders to achieve desirable tradeoffs. For example, the combination of the faster *Nullity*$_{\leq 1}$ and the more general *SingleHair* is useful to decode biased noise models.

A. *SingleHair* Relaxer Finder

We present a simple *Relaxer* finding algorithm for general hypergraphs, also described by Algorithm 9. When (V, T) is an *Invalid* subgraph, it returns Nil. Otherwise, it attempts to find *Relaxers* of the form $\Delta\vec{y} = \{\Delta y_S : -1, \Delta y_{S^+} : +1\}$: one *Hyperblossom* $S \in \mathcal{B}$ shrinks and another *Invalid* subgraph $S^+ \in \mathcal{O}$ grows. The algorithm examines each *Hyperblossom* $S \in \mathcal{B}$, constructs as many S^+ using its parity matrix (§IV A 1), and checks if $\Delta\vec{y} = \{\Delta y_S : -1, \Delta y_{S^+} : +1\}$ is a *Relaxer*.

1. Matrix Representation

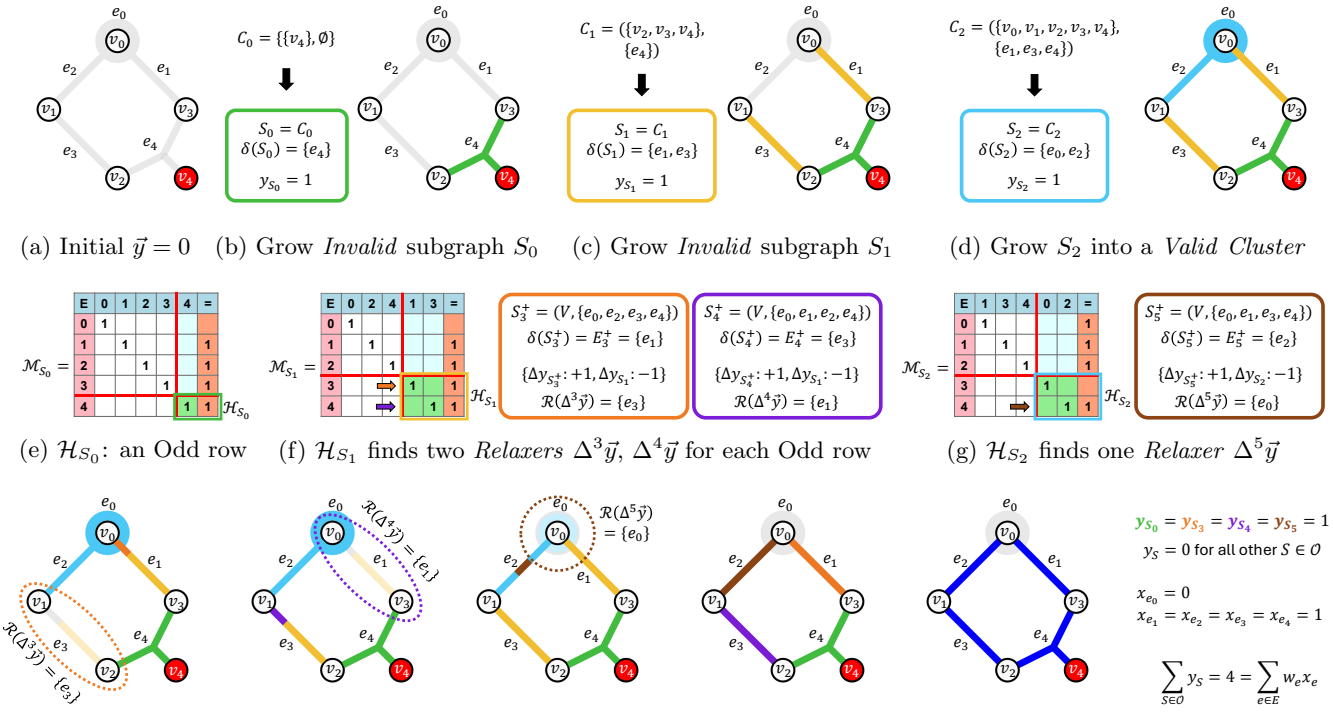
The *SingleHair Relaxer* finder uses a matrix representation of the parity constraints called *parity matrix*. A *parity matrix* M_G is an augmented matrix in linear algebra with $|V|$ rows and $|E|+1$ columns and defined on the modulo two field $\mathbb{F}_2$. The first $|E|$ columns correspond to the incidence matrix of the decoding hypergraph, while the last column is the syndrome D. The parity constraints set by D can be represented by $M_G(\vec{x}, 1)^T = 0$.

We show an example of a parity matrix in Fig. 7a.

There are two transformations of the parity matrix that preserve the solution space of $\vec{x}$. That is, $M_G(\vec{x}, 1)^T = 0$ if and only if $M'_G(\vec{x}, 1)^T = 0$ where M'_G is derived from M_G based on these transformations. In other words, a parity factor for G is also one for G' and vice versa. First, we can re-order the rows and columns, which correspond to renaming the vertices and edges, respectively.

Second, we can add one row to another row in $\mathbb{F}_2$, as demonstrated in Fig. 7a. Given the linearity of the $\mathbb{F}_2$ constraints, adding one constraint to another will not change the solution space [32].

With the *Row Addition*, we can use Gauss-Jordan elimination to obtain the *reduced row echelon form* [32] of the parity matrix, as shown in Fig. 7a. We call it *Echelon Matrix*. The columns that correspond to the row-leading 1s in the *Echelon Matrix* are called *pivot variables*, while the rest are *free variables*. The row containing the leading 1 of a pivot variable is called the *pivot row* of the variable. In the example of M''_G in Fig. 7a, all the variables are pivot and the solution space is singleton. However, general QEC codes usually have a large solution space with a lot of free variables due to quantum degeneracy [33]. That is, given a parity factor $\mathcal{E}$ and any logical operator $\mathcal{E}_L \subseteq E, \mathcal{D}(\mathcal{E}_L) = \varnothing$, $\mathcal{E}' = \mathcal{E} \oplus \mathcal{E}_L$ is also a parity factor because $\mathcal{D}(\mathcal{E}') = \mathcal{D}(\mathcal{E}) \oplus \mathcal{D}(\mathcal{E}_L) = D$. The time complexity of Gauss-Jordan elimination is $O(|V||E|^2)$, or $O(|V|^3)$



(a) Initial $\vec{y} = 0$ (b) Grow *Invalid* subgraph S_0 (c) Grow *Invalid* subgraph S_1 (d) Grow S_2 into a *Valid Cluster*

(e) $\mathcal{H}_{S_0}$: an Odd row (f) $\mathcal{H}_{S_1}$ finds two *Relaxers* $\Delta^3\vec{y}$, $\Delta^4\vec{y}$ for each Odd row (g) $\mathcal{H}_{S_2}$ finds one *Relaxer* $\Delta^5\vec{y}$

(h) *Relaxer* $\Delta^3\vec{y}$ (i) *Relaxer* $\Delta^4\vec{y}$ (j) *Relaxer* $\Delta^5\vec{y}$ (k) DLP solution $\vec{y}$ (l) parity factor $\mathcal{E}$ (m) MWPF Proof

FIG. 8: Example of the HyperBlossom algorithm with the *SingleHair Relaxer*-finding algorithm. The decoding hypergraph has a uniform weight $w_e = 1$ for all edges $e \in E$. (a-d) When the *Cluster* C is *Invalid*, we keep growing the dual variable y_C. The *Cluster* merges more vertices and *Tight Edges* until it becomes *Valid*. In this example, the *Valid Cluster* is $C = (V, E)$ and its *Hyperblossoms* are $\mathcal{B}_C = \{S_0, S_1, S_2\}$. (e-g) For each *Hyperblossom* $S \in \mathcal{B}$, the *SingleHair Relaxer*-finding algorithm constructs a *Hyperblossom Matrix* $\mathcal{M}_S$ and a *Hair Matrix* $\mathcal{H}_S$. Each Odd row in the *Hair Matrix* corresponds to a *Relaxer* $\Delta^i\vec{y}, \forall i \in \{3, 4, 5\}$. Each *Relaxer* grows a new dual variable $y_{S_i^+}$ and shrinks an existing dual variable y_{S_j}. (h-j) Growing along the *Direction* $\Delta^i\vec{y}$ for a small length of $1/4$ relaxes the edges $\mathcal{R}(\Delta^i\vec{y})$ in the dotted circle. (k) We find a better DLP solution $\vec{y}$ by solving the linear programming problem on the new *Hyperblossoms* $\mathcal{B}_C^H = \{S_0, S_1, S_2, S_3^+, S_4^+, S_5^+\}$. (l) We find a parity factor (blue edges) among the *Tight Edges*. (m) We prove the optimality of both the parity factor and $\vec{y}$ by **Theorem: Certifying Optimum**: the primal sum $\sum_{e \in E} w_e x_e$ is equal to the dual sum $\sum_{S \in \mathcal{O}} y_S$.

for sparse decoding hypergraphs from qLDPC codes. We denote equivalent parity matrices as $M_G \cong M'_G$ and and their decoding hypergraphs as $G \cong G'$.

2. *Important Concepts*

Definition: Hyperblossom Matrix. Given a *Hyperblossom* $S \in \mathcal{B}$, the *Hyperblossom Matrix* $\mathcal{M}_S$ is an *Echelon Matrix* (§IV A 1) of the subgraph (V, T) that places all the columns corresponding to tight *Hairs* of S, $(\delta(S) \cap T)$, on the rightmost.

As an example, we show *Hyperblossom Matrices* in Fig. 8d with three *Hyperblossoms* $\mathcal{B} = \{S_0, S_1, S_2\}$ (Fig. 8e to Fig. 8g).

Definition: Hair Matrix. Given a *Hyperblossom* $S \in \mathcal{B}$, the *Hair Matrix* $\mathcal{H}_S$ is a submatrix of $\mathcal{M}_S$, consisting of the rightmost $|\delta(S) \cap T| + 1$ columns and all the rows below the last pivot row of the variables $T \setminus \delta(S)$.

The *Hair Matrix* represents the solution space for the primal variables corresponding to the tight *Hairs* of S, i.e., $x_e, \forall e \in \delta(S) \cap T$. For example, Fig. 8e shows a *Hair Matrix* $\mathcal{H}_{S_0}$ that constrains the parity factor solution space to have $x_{e_4} = 1$.

Lemma: Hair Matrix Odd Row Existence. In a *Hair Matrix*, a row of which the right most value is 1 is called an *Odd* row. An Odd row always exists in a *Hair Matrix*.

For example, Fig. 8g shows an Odd row in the *Hair Matrix* $\mathcal{H}_{S_2}$. We prove the above lemma in §B 1.

3. *Relaxer Finding Algorithm*

The algorithm enumerates through all *Hyperblossoms*. For each *Hyperblossom* $S \in \mathcal{B}$, the algorithm constructs

its *Hair Matrix*. According to the Lemma above, the algorithm finds all the Odd rows. An Odd row poses an odd parity constraint on a subset of primal variables $E^+ \subseteq \delta(S) \cap T$. That is, E^+ consists of the edges whose corresponding columns in the Odd row have value 1. For example, $E^+ = \{e_2\}$ for the Odd row in $\mathcal{H}_{S_2}$ in Fig. 8g.

The algorithm then constructs $S^+ = (V, T \setminus E^+)$. Because edges in E^+ are essential to satisfy the odd parity constraint of the Odd row, S^+ must be an *Invalid* subgraph, i.e., $S^+ = (V, T \setminus E^+) \in \mathcal{O}$.

We show that $\Delta\vec{y} = \{\Delta y_S : -1, \Delta y_{S^+} : +1\}$ is a *Feasible Direction*. As S shrinks, it will relax the DLP constraint Eq. (4b) for edges in $\delta(S)$ and therefore those in $E^+ \subseteq \delta(S)$. As a result, when S^+ grows and tightens the constraint for $\delta(S^+) \cap T = E^+$, there will not be any violation of Eq. (4b).

Finally, because $\Delta\vec{y}$ relaxes edges $E^- = \delta(S) \cap T \setminus E^+$, it is a *Relaxer* if $E^- \neq \varnothing$. And the algorithm will continue to construct S^+ from remaining Odd rows and check if $\Delta\vec{y} = \{\Delta y_S : -1, \Delta y_{S^+} : +1\}$ is a *Relaxer*. If $E^- = \varnothing$, there will be no more Odd rows according the following lemma proved in §B 2.

Lemma: Unique Row if Not a Relaxer. When $E^- = \varnothing$, i.e., there is no *Relaxer* corresponding to an Odd row, then there is only one row.

4. Time Complexity and Limitations

When the algorithm has enumerated all *Hyperblossoms* and could find no more *Relaxers*, the *Hair Matrices* of all *Hyperblossoms* must have become a row of all 1s, as shown in Fig. 8e. When this happens, given a tight *Hair* e of S ($e \in \delta(S) \cap T$), there exists a feasible parity factor that includes e but not other tight *Hairs* of S, i.e., $\exists \mathcal{E} \subseteq T, \mathcal{D}(\mathcal{E}) = D, \mathcal{E} \cap \delta(S) = \{e\}$, according to the property of an Echelon Matrix (§IV A 1). We say the DLP solution is in a *Single-Hair* state. That is, the *SingleHair Relaxer* finder will not be able to find more *Relaxers* if the DLP solution is in a Single-Hair state.

SingleHair runs in polynomial time of $O(\text{poly}(|V_C| + |E_C| + |\mathcal{B}_C|))$. However, when using *SingleHair*, the number of iterations of the HyperBlossom algorithm (Algorithm 6, line 3) may not be polynomially bounded, given the NP-hardness of the general MWPF problem [5]. Also, the HyperBlossom algorithm equipped with *SingleHair* alone may not eventually find an optimal DLP solution.

Being a very simple *Relaxer* finder algorithm, *SingleHair* has some limitations, as explained in §B. Nonetheless, our empirical evaluation shows that it achieves the same accuracy as the MWPM decoder on simple graphs (Fig. 13a), where MWPM decoder is an optimal MLE decoder. Moreover, it achieves a higher accuracy than existing UF [6], MWPM [1] and HUF [7] decoders on hypergraphs (§VI). *SingleHair* does not assume or exploit any special property of the decoding hypergraph. We will discuss two *Relaxer* finders that exploit the special properties of the decoding hypergraph in §IV C and §IV D.

Algorithm 9 *SingleHair Relaxer* finder

Input: G (hypergraph), $\mathcal{B}$ (*Hyperblossoms*), T (*Tight Edges*)
Output: $R[G, \mathcal{B}, T]$ (*Relaxer*) or Nil (if not found)

```
1: procedure SingleHairRelaxerFinder(G, B, T)
2:     if (V, T) is an Invalid subgraph then
3:         return Nil
4:     M_G ← ParityMatrix(V, T)  ▷ parity matrix §IV A 1
5:     for S ∈ B do
6:         M_S ← HyperBlossomMatrix(M_S, G)
7:         H_S ← HairMatrix(M_G, S)  ▷ Hair Matrix
8:         if H_S is a single row of all 1s then
9:             continue
10:        r ← any Odd row of H_S
11:        E+ ← the columns of r that are 1
12:        S+ ← (V, T \ E+)
13:        return {Δy_S : −1, Δy_{S+} : +1}  ▷ Relaxer
14:    return Nil
15:
16: procedure HyperBlossomMatrix(M_G, S)
17:    M_S ← ( ∀e_j ∉ δ(S), | ∀e_j ∈ δ(S), | M_G(:, 1+|T∩δ(S)|) )
                ( M_G(:, j)  |  M_G(:, j)  |
       ▷ reorder the columns according to Hyperblossom Matrix
18:    return GaussJordanElimination(M_S)  ▷ §IV A 1
19:
20: procedure HairMatrix(M_G, S)
21:    i_0 ← 1 + (last pivot row among columns T \ δ(S))
22:    j_0 ← 1 + |T \ δ(S)|
23:    H_S ← M_S(i_0 : , j_0 :)  ▷ by definition of Hair Matrix
```

B. *UnionFind* Relaxer Finder

The HyperBlossom framework can implement the Union-Find decoders [6, 7] with a trivial *Relaxer* finder that does nothing but returns Nil, as shown in Algorithm 10. The *Clusters* in the HyperBlossom framework corresponds to the clusters in the UF decoders. This reduces the Primal phase (Algorithm 8) to a simple logic. It first calls the *Relaxer* finder via Algorithm 2 (line 4), which returns $Rs \leftarrow \varnothing$ (Algorithm 8, line 5). It then checks if the *Cluster* C is *Invalid* ($C \in \mathcal{O}$, line 7). If so, it grows the *Invalid Cluster* uniformly with a *Direction* $\{\Delta y_C : +1\}$ that expands on all incident non-tight edges $\delta(C) = E[V_C] \setminus E_C$. This behavior is identical to that of a weighted UF decoder for simple graphs [19, 34]. More importantly, the HyperBlossom framework with *UnionFind* generalizes the Hypergraph Union-Finder (HUF) decoder [7] for weighted decoding hypergraphs.

Algorithm 10 *UnionFind Relaxer* finder

Input: G (hypergraph), $\mathcal{B}$ (*Hyperblossoms*), T (*Tight Edges*)
Output: $R[G, \mathcal{B}, T]$ (*Relaxer*) or NIL (if not found)
1: **procedure** UNIONFINDRELAXERFINDER($G, \mathcal{B}, T$)
2: **return** NIL

C. *Blossom* Relaxer Finder

For simple graphs, there exists a *Relaxer* finder such that the HYPERBLOSSOM algorithm equipped with it terminates with optimal MWPF and DLP solutions, according to **Theorem: Simple Graphs min LP = min ILP** and **Theorem: HyperBlossom Algorithm Optimality**. We describe such a polynomial-time and optimal *Relaxer* finder called *Blossom* using ideas from the blossom algorithm. In practice, the HYPERBLOSSOM algorithm with *Blossom* will be less efficient than the highly optimized blossom algorithm. Our goal here is to show that the HYPERBLOSSOM framework can implement an MWPM decoder for simple decoding graphs.

To show that the HYPERBLOSSOM algorithm with *Blossom* indeed implements the MWPM decoder, we look at the progression of their DLP solutions. We show that the two sequences of DLP solutions are step-by-step equivalent. That is, there exists a bijective function f such that the *Blossom DLP solution* $\vec{y^*}$ and the the DLP solution $\vec{y}$ satisfy $\vec{y} = f(\vec{y^*})$ and $\sum \vec{y} = \sum \vec{y^*}$ throughout the whole algorithm.

1. *Important Concepts*

We first provide some background on how the blossom algorithm works. As mentioned in §III D, the blossom algorithm also works with two interactive phases as illustrated by Fig. 3: the Primal phase solves the MWPM problem while the Dual phase solves a related DLP problem. Like in the HYPERBLOSSOM algorithm, the Primal phase finds *Directions* for the Dual phase to adjust the dual variables and moves toward a better solution.

Unlike the HYPERBLOSSOM algorithm, the blossom algorithm uses a data structure called *alternating tree* to find *Directions*. Before introducing the alternating tree, we must first introduce *blossoms* and *nodes*, which are its vertices. A *blossom* represents a set of an odd number of defect vertices $S^* \subseteq D, |S^*| = 1 \mod 2$ [4]. The set of all possible blossoms is denoted as $\mathcal{O}^*$. Each $S^* \in \mathcal{O}^*$ corresponds to a dual variable $y^*_{S^*}$ in the blossom algorithm formulation, similar to the dual variables y_S in the DLP. Those $S^* \in \mathcal{O}^*$ with positive dual variables $y^*_{S^*} > 0$ are called *blossoms*, similar to the *Hyperblossoms*. We use S^* with a superscript $*$ to denote the blossoms $S^* \subseteq D$ to avoid confusion with the *Invalid* subgraphs $S = (V_S, E_S)$. In general, $*$ superscript indicates a notion in the blossom algorithm. The blossoms have a hierarchical structure: a parent blossom is constructed from a cycle of children blossoms. A blossom without any parent blossom is called a *node*.

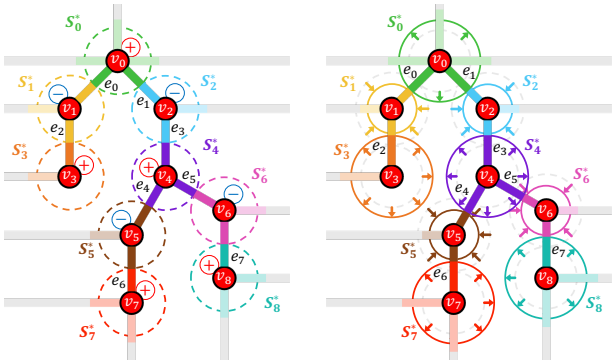
We define the blossom DLP solutions that are intermediate states of the blossom algorithm as below.

Definition: Blossom DLP solution. A feasible intermediate DLP solution $\vec{y^*}$ of the blossom algorithm.

Definition: Good Direction. Growing a *Blossom DLP solution* $\vec{y^*}$ along a *good direction* $\Delta\vec{y^*}$ results in a *Blossom DLP solution* solution. That is, there exists a positive length $l > 0$ such that any small growth $0 \le l' \le l$ results in a *Blossom DLP solution* $\vec{y^*} + l'\Delta\vec{y^*}$.

Definition: Blossom Useful Direction. A *Good Direction* $\Delta\vec{y^*}$ is *Useful* if $\sum \Delta\vec{y^*} = \sum_{S^* \in \mathcal{O}^*} \Delta y^*_{S^*} > 0$.

An *alternating tree* consists of an odd number of nodes, where each node S^* is marked by either "+" or "−" [4], as shown in Fig. 9a. A "+" node can have an arbitrary number of children and a "−" node has exactly one child. A parent and child nodes are connected by *Tight Edges* in the syndrome graph [4]. The root node and leaf nodes are "+" nodes. Any path from the root to a leaf has an alternating "+" and "−" pattern. Each alternating tree represents a *Blossom Useful Direction* $\Delta\vec{y^*}$ defined by $\Delta y^*_{S^*} = +1$ for "+" nodes and $\Delta y^*_{S^*} = -1$ for "−" nodes. Since there are more "+" nodes than "−" nodes in an alternating tree, $\Delta\vec{y^*}$ is *Useful* with $\sum_{S^* \in \mathcal{O}^*} \Delta y^*_{S^*} \ge 1$. We show an example of an alternating tree growing along $\Delta\vec{y^*}$ for a small length in Fig. 9b.



(a) Alternating Tree (b) *Direction* $\Delta\vec{y^*}$

FIG. 9: Alternating tree in the blossom algorithm finds a *Blossom Useful Direction* of alternating $\Delta y^*_{S^*} = +1$ and $\Delta y^*_{S^*} = -1$. Note that an alternating tree represents the relationship between nodes, which is visualized in the above example on top of the decoding graph, similar to that in [3].

The blossom algorithm grows along a sequence of such *Blossom Useful Directions* $\Delta\vec{y^*}$ from the alternating trees, until $\sum \vec{y^*} = W(\mathcal{E})$ where $\mathcal{E}$ is the parity factor that the MWPM decoder finds.

A *Blossom DLP solution* $\vec{y^*} \in \mathbb{R}^{|\mathcal{O}^*|}$ is related to a fea-

sible DLP solution $\vec{y} \in \mathbb{R}^{|\mathcal{O}|}$ given the following theorem:

Theorem: Blossom Dual Function. There exists a bijective function f from a *Blossom DLP solution* $\vec{y^*}$ to a DLP solution of the HyperBlossom algorithm $\vec{y} = f(\vec{y^*})$ while satisfying two conditions: their dual objectives are the same $\sum_{S \in \mathcal{O}} y_S = \sum_{S^* \in \mathcal{O}^*} y^*_{S^*}$, and, when a syndrome-graph edge (u, v) is tight [4] in $\vec{y^*}$, then the minimum-weight path between u and v in the DLP solution $\vec{y} = f(\vec{y^*})$ consists of all *Tight Edges*.

Note that the domain of the function $f(\cdot)$ is all *Blossom DLP solutions*, but not $\mathbb{R}^{|\mathcal{O}^*|}$ or all feasible dual solutions to the DLP problem of the blossom algorithm. Similarly, the codomain of $f(\cdot)$ (domain of $f^{-1}(\cdot)$) is not $\mathbb{R}^{|\mathcal{O}|}$ but only those $\vec{y} = f(\vec{y^*})$.

In §C 2, we provide such a function f (*Dual Mapping*) and use it as a constructive proof for the above theorem in §C 3. We then define the direction-specific gradient of f, which can map a *Good Direction* $\Delta\vec{y^*}$ to a *Feasible Direction* $\Delta\vec{y}$.

$$\Delta\vec{y} = (\Delta\vec{y^*} \circ \nabla) f(\vec{y^*}) \coloneqq \lim_{l \to 0^+} \frac{f(\vec{y^*} + l\Delta\vec{y^*}) - f(\vec{y^*})}{l}$$

In practice, we can use a sufficiently small l to compute the mapped *Feasible Direction* efficiently. We present the algorithm in Algorithm 11 line 11.

2. Relaxer Finding Algorithm

We now describe the *Blossom Relaxer*-finding algorithm based on the **Theorem: Blossom Dual Function**, as shown in Algorithm 11. The HyperBlossom algorithm equipped with *Blossom* always terminates with an optimal DLP solution. *Blossom* first maps the current DLP solution $\vec{y}$ to a *Blossom DLP solution* $\vec{y^*}$, reconstructs an alternating tree from $\vec{y^*}$, and then uses the alternating tree to find a *Good Direction* that maps to a *Relaxer* in certain cases.

Blossom starts with the decoding graph G and a DLP solution $\vec{y}$. It first computes the corresponding DLP solution for the blossom algorithm $\vec{y^*} = f^{-1}(\vec{y})$ (Algorithm 11, line 2).

From $\vec{y^*}$, the algorithm then constructs the alternating trees (Algorithm 11, line 3), using an algorithm included in a constructive proof of **Theorem: Alternating Tree Reconstruction** in §C 5.

If all the alternating trees consist of only one node, the algorithm returns Nil. Otherwise, it computes a *Good Direction* as below. As noted in §IV C 1, alternating trees represent a *Blossom Useful Direction* $\Delta\vec{y^*}$ for the blossom algorithm (line 6). Suppose the root node of this alternating tree is S^* and one of its children is S^*_c. The algorithm takes $\Delta\vec{y^*}$ and changes $\Delta y^*_{S^*} = +1$ to $\Delta y^*_{S^*} = 0$ (line 7). This results in a *Good Direction* $\Delta\vec{y'^*}$ with $\sum_{S^* \in \mathcal{O}^*} \Delta y'^*_{S^*} = \sum_{S^* \in \mathcal{O}^*} \Delta y^*_{S^*} - 1 \geq 0$.

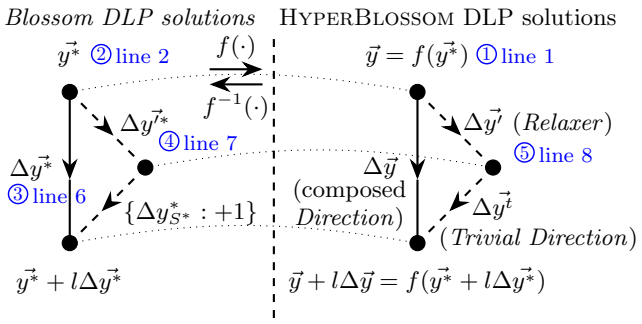


FIG. 10: The *Blossom DLP solutions* (dots on the left side) correspond to the DLP solutions (dots on the right side) with a bijective function f. The *Blossom Relaxer* finder finds a *Relaxer* $\Delta\vec{y'}$ following the path of $\vec{y} \to \vec{y^*} \to \Delta\vec{y^*} \to \Delta\vec{y'^*} \to \Delta\vec{y'}$. The HyperBlossom algorithm with *Blossom* progresses from $\vec{y} = f(\vec{y^*})$ to $\vec{y} + l\Delta\vec{y} = f(\vec{y^*} + l\Delta\vec{y^*})$, thus preserving the correspondence throughout the algorithm.

The algorithm then computes a *Feasible Direction* $\Delta\vec{y'} = (\Delta\vec{y'^*} \circ \nabla) f(\vec{y^*})$ from the *Good Direction* $\Delta\vec{y'^*}$.

Finally, we show that $\Delta\vec{y'}$ is a *Relaxer* by showing that it relaxes some *Tight Edges* between vertices in S^* and vertices in S^*_c. According to **Theorem: Blossom Dual Function**, there is a minimum-weight path consisting of all *Tight Edges* in $\vec{y} = f(\vec{y^*})$ between a pair of vertices in S^* and S^*_c, given that there exists a tight syndrome-graph edge between the nodes S^* and S^*_c in the alternating tree by definition. As $\Delta y'^*_{S^*} = 0$ and $\Delta y'^*_{S^*_c} = -1$, an edge on the minimum-weight path is relaxed. Thus, $\Delta\vec{y'}$ is a *Relaxer*.

We next show that the *Blossom Relaxer* finder is optimal. When there exists an alternating tree with more than one node, the HyperBlossom algorithm finds a *Trivial Direction* $\Delta\vec{y^t} = (\{\Delta y^*_{S^*} = +1\} \circ \nabla) f(\vec{y^*} + l\Delta\vec{y'^*})$ (line 7 of Algorithm 8) and composes a *Useful Direction* $\Delta\vec{y} = \Delta\vec{y^t} + \Delta\vec{y'}$ (Algorithm 1) that is identical to the *Useful Direction* mapped from the blossom algorithm $(\Delta\vec{y^*} \circ \nabla) f(\vec{y^*})$, as shown in Fig. 10. Otherwise, the HyperBlossom algorithm falls back to the Union-Find decoder and it still behaves the same as an MWPM decoder: odd *Clusters* (isolated nodes) grow uniformly while even *Clusters* (matched nodes) stay. In both cases, the HyperBlossom algorithm progresses from a DLP solution $\vec{y} = f(\vec{y^*})$ to a new DLP solution $\vec{y} + l\Delta\vec{y} = f(\vec{y^*} + l\Delta\vec{y^*})$, preserving the correspondence between the DLP solutions and the *Blossom DLP solutions* as shown in Fig. 10. Also given that the blossom algorithm terminates at a *Blossom DLP solution* with $\sum \vec{y^*} = W(\mathcal{E})$, the final DLP solution is $\vec{y} = f(\vec{y^*})$ with $\sum \vec{y} = \sum \vec{y^*} = W(\mathcal{E})$. Given **Theorem: Certifying Optimum**, $\vec{y}$ is an optimal DLP solution and thus the *Blossom Relaxer* finder is optimal.

Algorithm 11 *Blossom Relaxer* finder

Input: G (hypergraph), $\mathcal{B}$ (*Hyperblossoms*), T (*Tight Edges*)
Output: $R[G, \mathcal{B}, T]$ (*Relaxer*) or Nil (if not found)

procedure BlossomRelaxerFinder($G, \mathcal{B}, T$)
  $\vec{y^*} \leftarrow f^{-1}(\vec{y})$
  reconstruct alternating trees from $\vec{y^*}$
  if $\exists$ an alternating tree with multiple nodes **then**
    $S^* \leftarrow$ the root node of the alternating tree
    $\Delta\vec{y^*} \leftarrow$ *Useful Direction* of the alternating trees
    $\Delta\vec{y'^*} \leftarrow \Delta\vec{y^*} - \{\Delta y^*_{S^*} = +1\}$
    return MapFeasibleDirection($G, \vec{y}, \vec{y^*}, \Delta\vec{y'^*}$)
  return Nil

procedure MapFeasibleDirection($G, \vec{y}, \vec{y^*}, \Delta\vec{y^*}$)
  $L \leftarrow \{w_e - \sum_{S \in \mathcal{O} | e \in E} y_S | e \in E\} \cup \{y_S | S \in \mathcal{O}\}$
  $l \leftarrow \min(\{l \in L | l > 0\})/2$ ▷ sufficiently small length
  return $[f(\vec{y^*} + l\Delta\vec{y^*}) - f(\vec{y^*})]/l$

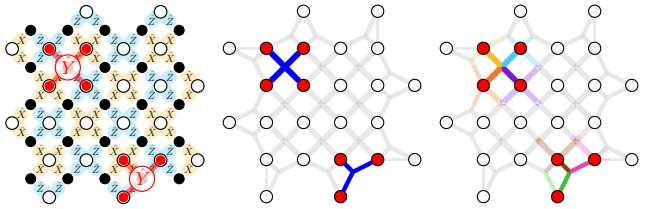
D. *Nullity*$_{\leq 1}$ Relaxer Finder

The nullity$_{\leq 1}$ hypergraphs are an important class of hypergraphs that satisfy **min LP = min ILP**, as proved with explicit constructions of optimal DLP solutions in §IV D 2. These hypergraphs have at most two parity factor solutions and, as a result, the MWPF problem is trivial for them. Nevertheless, the *Nullity*$_{\leq 1}$ *Relaxer* finder demonstrates the possibility of designing more efficient *Relaxer* finders exploiting special hypergraph properties. It is also practically important because some QEC codes with high noise bias tend to produce *Clusters* that predominantly has a nullity of no more than 1. An example of this type of decoding hypergraph comes from the rotated surface code with a biased Y noise model, as shown in Fig. 11a. The number of independent parity checks is d^2-1 while there are d^2 different Y error locations, leading to a nullity of 1. For example, given a valid syndrome in Fig. 11a, there are only two possible parity factors Fig. 11b and Fig. 12a. More generally, even if the decoding hypergraph is not a nullity$_{\leq 1}$ hypergraph, its small *Valid Clusters* tends to be nullity$_{\leq 1}$ hypergraphs because the number of *Tight Edges* is usually small. In such cases, *Nullity*$_{\leq 1}$ can complement other *Relaxer* finders to refine these *Clusters*.

1. Important Concepts

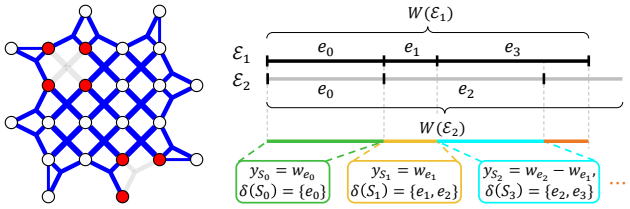
In the following section, we will construct an optimal DLP solution $\vec{y}$ for every nullity$_{\leq 1}$ hypergraph that has $\sum \vec{y} = W(\mathcal{E})$, which proves **Theorem: Nullity$_{\leq 1}$ Hypergraphs min LP = min ILP**.

Together with *Lemma: Nullity*$_{\leq 1}$ *is a hereditary property* (§B 5) and **Theorem: HyperBlossom Algorithm Optimality**, there exists an optimal *Relaxer* finder such that the HyperBlossom algorithm finds the optimal MWPF and DLP solutions. We will construct such an efficient and optimal *Relaxer* finder in the following section.



(a) Surface Code. (b) MWPF $\mathcal{E}_1$. (c) DLP $\vec{y}$.

FIG. 11: (a) The decoding hypergraph of a rotated surface code with biased Y noise is a nullity$_{\leq 1}$ hypergraph. (b,c) The optimal MWPF and DLP solutions.



(a) parity factor $\mathcal{E}_2$. (b) optimal DLP $\vec{y}$ with $|\mathcal{B}| \leq |E|$.

FIG. 12: *Nullity*$_{\leq 1}$ is efficient and optimal. (a) Given the other parity factor $\mathcal{E}_2$, (b) it finds an optimal DLP solution $\sum \vec{y} = W(\mathcal{E}_1)$ efficiently using no more than $|E|$ *Hyperblossoms*.

2. Relaxer Finding Algorithm

We present an efficient *Relaxer* finder for nullity$_{\leq 1}$ hypergraphs. The algorithm assumes that (V, T) is a *Valid* subgraph and that the hypergraph has a nullity 0 or 1. If not, it returns Nil. The basic idea of the algorithm is to find an optimal DLP solution $\vec{y^o}$ first and then construct a *Relaxer* for any suboptimal DLP solution $\vec{y}$. We will first explain how to find optimal DLP solutions for nullity$_{=0}$ and nullity$_{=1}$ hypergraphs, respectively.

Nullity$_{=0}$ Hypergraphs. When the nullity is 0, there exists only one parity factor $\mathcal{E}$. In this case, the optimal DLP solution has $\sum \vec{y^o} = W(\mathcal{E})$. We construct a trivial optimal DLP solution: $y^o_{S_e} = w_e, \forall e \in \mathcal{E}$ where $S_e = (V, E \setminus \{e\}) \in \mathcal{O}$. All the other dual variables are zero in $\vec{y^o}$.

Nullity$_{=1}$ Hypergraphs. When the nullity is 1, there exist two parity factors $\mathcal{E}_1$ and $\mathcal{E}_2$. Without losing generality, we assume $W(\mathcal{E}_1) \leq W(\mathcal{E}_2)$. In this case, the optimal DLP solution has $\sum \vec{y} = W(\mathcal{E}_1)$. Similar to the nullity$_{=0}$ case, we explicitly construct an optimal DLP solution $\vec{y^o}$. As shown in Fig. 12b, we visualize each edge in a parity factor with a length of its weight and arrange all edges in a straight line for $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively. We

Algorithm 12 *Nullity*$_{\leq 1}$ *Relaxer* finder

Input: G (hypergraph), $\mathcal{B}$ (*Hyperblossoms*), T (*Tight Edges*)
Output: $R[G, \mathcal{B}, T]$ (*Relaxer*) or Nil (if not found)

```
1: procedure NullityLE1RelaxerFinder(G, B, T)
2:     if (V, T) is an Invalid subgraph then
3:         return Nil
4:     if G is a nullity_0 hypergraph then
5:         E ← the parity factor
6:         if Σ y⃗ < W(E) then
7:             y⃗^o ← {y_(V,E\{e}) : w_e, ∀e ∈ E}
8:             return RelaxerFromOptimalDLP(y⃗, y⃗^o)
9:     if G is a nullity_1 hypergraph then
10:        E_1, E_2 ← the parity factors s.t. W(E_1) ≤ W(E_2)
11:        if Σ y⃗ < W(E_1) then
12:            y⃗^o ← 0
13:            arrange the edges of E_1 and E_2 like Fig. 12b
14:            for i-th interval with e_{1,i} ∈ E_1, e_{2,i} ∈ E_2 do
15:                y^o_(V,E\{e_{1,i},e_{2,i}}) ← w_i
16:            return RelaxerFromOptimalDLP(y⃗, y⃗^o)
17:    return Nil
18:
19: procedure RelaxerFromOptimalDLP(y⃗(B, T), y⃗^o)
20:    Δy⃗ ← y⃗^o − y⃗
21:    e ← pick one element from {e ∈ T | w_e > 0}
22:    S ← pick one element from {S ∈ B | e ∈ δ(S)}
23:    return Δy⃗ − {Δy_S : Σ_{S∈O} Δy_S}
```

place the shared edges $\mathcal{E}_1 \cap \mathcal{E}_2$ in the front of each line and align the fronts of the two lines. In each line, where two edges meet defines a point of joint. The points of joint in the two lines collectively define a series of *intervals*: an interval is defined by two nearest joints regardless which line they are from. Each interval with a length of w_i and a pair of edges $e_{1,i} \in \mathcal{E}_1, e_{2,i} \in \mathcal{E}_2$ corresponds to an *Invalid* subgraph $S_i = (V, E \backslash \{e_{1,i}, e_{2,i}\})$ with $\delta(S_i) = \{e_{1,i}, e_{2,i}\}$. $e_{1,i}$ and $e_{2,i}$ can be the same as for S_0 in Fig. 12b. By letting $y^o_{S_i} = w_i$ for every such overlapping interval and $y^o_S = 0$ for others, we have $\vec{y^o}$ as an optimal DLP solution because $\sum \vec{y^o} = \sum_i w_i = W(\mathcal{E}_1)$.

Once we have obtained the optimal DLP solution $\vec{y^o}$, we can then construct a *Relaxer* for every suboptimal DLP solution $\vec{y}$. By definition of suboptimality, $\sum_{S \in \mathcal{O}} y_S < \sum_{S \in \mathcal{O}} y^o_S$. In another word, $\Delta\vec{y} = \vec{y^o} - \vec{y}$ is a *Useful Direction*. There must exist at least one positive-weighted *Tight Edge* $e \in T, w_e > 0$ in the suboptimal solution. This is because (V, T) is a *Valid* subgraph: when no such edge exists, $(V, \{e \in E | w_e = 0\})$ is a *Valid* subgraph and the optimal MWPF $\mathcal{E}$ has $W(\mathcal{E}) = 0$. This contradicts with the assumption that $\vec{y}$ is suboptimal given that the initial $\vec{y} = \mathbf{0}$ is already optimal. There exists at least one *Hyperblossom* $S \in \mathcal{B}, y_S > 0$ that contributes to e to make it tight, i.e., $e \in \delta(S)$. Thus, reducing the value of y_S will relax e. We can then easily construct a *Relaxer* $\Delta\vec{y'} = \Delta\vec{y} - \{\Delta y_S : \sum_{S \in \mathcal{O}} \Delta y_S\}$.

V. IMPLEMENTATION

We present Hyperion, a software implementation of the HyperBlossom framework (§III) and two *Relaxer*-finding algorithms: *SingleHair* (§IV A) and *UnionFind* (§IV B). We implement Hyperion in Rust programming language (§V A) and expose a Python library (§V B) to support easy integration into other tools like stim [35], a widely used quantum error correction simulator.

A. System Implementation in Rust

We implement Hyperion in Rust for high performance and flexible generics at no runtime cost.

Hyperion features two data types of the edge weights and dual variables: rational and floating-point. The former provides infinite accuracy when solving the DLP problem, but usually runs slower than the latter. The two data types can be switched with a compile-time flag. The implementation are the same for the two data types, except for the linear-programming solver invoked by line 10 of Algorithm 7. We employ the HiGHS [36] solver for the floating-point data type and the SLP [37] solver for the rational data type.

To improve runtime efficiency, we split the decoding algorithm into two stages, the *search* stage and the *refine* stage. The major difference is that the *search* stage works on *Invalid Clusters* while the *refine* stage works on *Valid Clusters*. They use different data structures to update the DLP variables. In the *search* stage, we ensure that every *Cluster* finds a pair of feasible MWPF and DLP solutions. In the *refine* stage, we refine each *Cluster* individually.

In the *search* stage, all *Invalid Clusters* $C \in \mathcal{C} \cap \mathcal{O}$ grow simultaneously. This has the benefit of minimizing their average size ($|V_C| + |E_C|$), as explained in [3]. Hyperion use a priority queue data structure to update the DLP solution, inspired by [3]. The *search* stage terminates when all *Clusters* are *Valid*, i.e., we find a feasible parity factor $\mathcal{E} = \cup_{C \in \mathcal{C}} \mathcal{E}_C$ at the end of the *search* stage.

The subsequent *refine* stage improves the local MWPF $\mathcal{E}_C$ and the DLP solution for each *Cluster* individually. Hyperion iteratively call the *Relaxer* finders (§IV) which introduces more *Hyperblossoms* $\mathcal{B}_C$ to each *Cluster* $C \in \mathcal{C}$. The more *Hyperblossoms* in the *Cluster*, the slower the LP solver runs (line 10 of Algorithm 7). Thus, to improve decoding efficiency, we assign each *Cluster* $C \in \mathcal{C}$ a priority score Priority(C) empirically:

$$\text{Priority}(C) = \left(\sum_{S \in \mathcal{B}_C} y_S - \sum_{e \in E_C} w_e x_e \right) / \left(|E_C| + |\mathcal{B}_C| \right)^3$$

The score favors small *Clusters* with large primal-dual gaps between the MWPF and DLP objectives, enabling more efficient convergence per unit time.

Given the NP-hardness of the MWPF problem [5], the algorithm may not terminate in a polynomial time. To

mitigate this problem, we introduce a per-*Cluster* limit on the number of *Hyperblossoms* $c = \max|\mathcal{B}_C|$. Once a *Valid Cluster* hits the limit ($|\mathcal{B}_C| \geq c$), HYPERION stop invoking *Relaxer* finders for this *Cluster*. We denote an MWPF decoder with the per-*Cluster* limit c as "MWPF(c)". We note that "MWPF(c=0)" is equivalent to the Hypergraph Union-Find (HUF) decoder [7]. The per-*Cluster* limit prevents spending excessive amount of time on intractable *Clusters*. It improves the latency for real-time decoding at the cost of lower accuracy.

B. Python Library and Stim Integration

Through the Python library, users can use MWPF solvers with either rational or floating-point data type. Each solver includes an interactive visualization tool and an adaptor for `stim` [35].

The `mwpf` solver uses floating-point data type for fast decoding, while the alternative `mwpf-rational` solver uses rational data type for theoretical analysis. The solvers share an identical interface and are interchangeable in most use cases. They output a parity factor and an associated proximity bound as defined in Eq. (5).

HYPERION also includes a 3D visualization tool that allows users to inspect each decoding step interactively. It visualizes decoding hypergraphs in a 3D space and visualizes the dual variables as colored segments on the hyperedges. In fact, we used it to generate the figures in this paper. For portability, HYPERION supports both an interactive widget for Jupyter notebooks and a self-contained HTML file [38].

HYPERION is compatible with the most popular QEC simulation library, `stim` [35], and integrates seamlessly with its standard `sinter` simulation pipeline. A `stim` user can incorporate HYPERION as the decoder with just a single line of code change. Additionally, users can supply a Clifford circuit to HYPERION so that it exploits circuit-level information beyond the Detector Error Model (DEM) of `stim`, such as heralded erasure error channels.

VI. EVALUATION

We evaluate HYPERION with the floating-point data type and the *SingleHair Relaxer* finder (§IV A), denoted as "MWPF(c)" given a tunable per-*Cluster* limit c (§V). Note that "MWPF(c=0)" is essentially the Hypergraph Union-Find (HUF) decoder. We answer the following questions in our evaluation:

- Accuracy: how does the logical error rate of HYPERION compare to existing decoders?
- Speed: how long does it take to decode a syndrome, and what is the distribution of decoding time?
- Accuracy-speed trade-off: how does the decoding time scale with the desired accuracy?

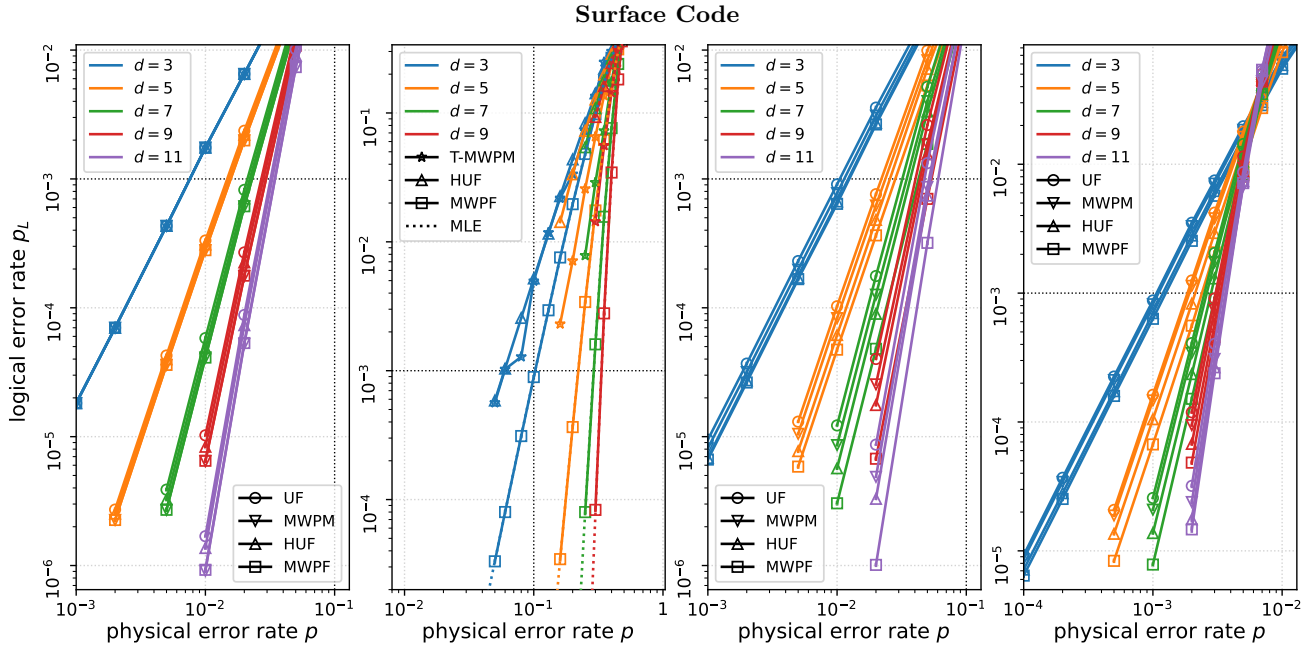
A. Simulation Setup

We use `stim` [35] to generate the syndrome given a circuit and noise model. We generate enough syndromes to ensure the error bar corresponding to 95% confidence interval is sufficiently small that the relative accuracies of different decoders are obvious. For each setting, we use the same sequence of syndromes across all decoders when applicable. The number of syndromes depends on the logical error rate, from 1000 for the highest to 4.6×10^9 for the lowest. We consider three types of qLDPC codes: surface code [1], color code [40] and Bivariate Bicycle (BB) code [41]. We employ code-capacity noise models including bit-flip noise ($p_X = p$, $p_Y = p_Z = 0$), depolarizing noise ($p_X = p_Y = p_Z = p/3$) and biased-Y noise ($p_Y = p$, $p_X = p_Z = 0$). We also consider circuit-level noise models defined in [35], [42] and [43] for the surface code, the color code and the the BB code, respectively. We compare HYPERION with existing MWPM decoder [3, 4] and BPOSD decoder [8, 9]. We also consider variants of the MWPM decoder optimized for the color code (denoted as "C-MWPM") as implemented by the authors of [42, 44] and for the tailored surface code with biased-Y noise [39] (denoted as "T-MWPM"), which we implement in [45]). For a fair comparison, we ensure that all decoders for the same setting are decoding the exact same sequence of syndromes, except for T-MWPM, which does not support `stim`. In this case, the results remain comparable within the 95% confidence interval marked by the error bars.

We choose Parity Blossom [4] for the baseline MWPM decoder for two reasons. First, it allows tuning between UF and MWPM decoder using a parameter similar to the per-cluster limit c in MWPF (§V A), denoted as "MWPM(c)". We note that MWPM(c=0) is equivalent to the UF decoder and MWPM(c=∞) is equivalent to the MWPM decoder. Second, it is implemented in Rust and has the same overhead adapting to the Python interface as HYPERION, for a fair comparison. For speed benchmarks, all decoders use a single thread on an Apple M4 Pro CPU with 24 GB of memory.

For a consistent comparison, we only use the *SingleHair Relaxer* finder (§IV A) in HYPERION, which is a simple *Relaxer* finder that does not exploit any special property of the decoding hypergraph. We use MWPF(c=200) as the HYPERION default and MWPF(c=0) as the HUF decoder.

We omit data near or above the threshold because (1) HYPERBLOSSOM relies on *clustering*, which becomes prohibitively slow at high error rates, and (2) sub-threshold scaling is more relevant for QEC, as systems cannot scale when physical error rates are too high. While this is a limitation of HYPERBLOSSOM, we believe focusing on lower error rates is a reasonable design choice.



(a) bit-flip noise (b) biased-Y noise (c) depolarizing noise (d) circuit-level noise

FIG. 13: Hyperion (MWPF) achieves the highest accuracy on surface code under a variety of noise models. (a) For the bit-flip noise, it achieves the same accuracy as the MWPM decoder, which is an optimal MLE decoder. (b) For the biased-Y noise, it achieves orders of magnitude higher accuracy than the state-of-the-art T-MWPM decoder [39], achieving the same accuracy as the optimal MLE decoder (dotted line). (c) For the depolarizing noise and (d) circuit-level noise, it empirically achieves higher accuracy than existing decoders.

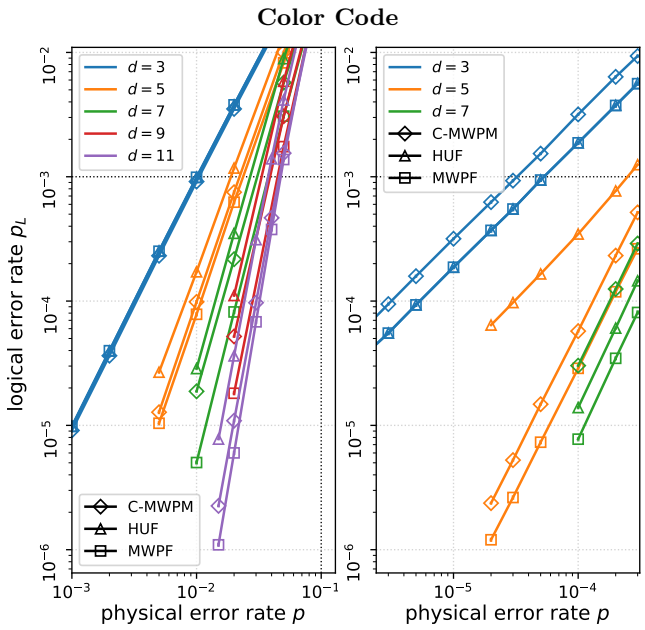
B. Decoding Accuracy

Hyperion achieves higher decoding accuracy compared to the MWPM decoder and its variants when decoding topological codes like surface code and color code. However, for non-topological codes like the BB code, its accuracy is lower than the BPOSD decoder at high error rates or large code distances.

For the **surface code**, we consider three types of code-capacity noise models (Fig. 13 (a-c)) and a circuit-level noise model (Fig. 13 (d)). Different noise models lead to different types of decoding hypergraphs, which impact the performance of decoders. This highlights the importance of evaluating decoders with the right noise model. For the bit-flip noise, the decoding hypergraph is a simple graph and as a result, the MWPM decoder is an optimal MLE decoder. In this case, Hyperion achieves the same accuracy as the MWPM decoder, despite that theoretically the *SingleHair Relaxer* finder may not find the optimal solution (§IV A). For the biased-Y noise, the decoding hypergraph is a nullity$_{\leq 1}$ hypergraph. Although the T-MWPM decoder [39] has a near-optimal threshold, it has been shown to be suboptimal in the logical error rate scaling [46]. MWPF(c=∞) is proven to be optimal for nullity$_{\leq 1}$ decoding hypergraphs (§B 5), and we show that MWPF(c=200) already achieves orders of magnitude higher accuracy than the T-MWPM decoder. For depolarizing noise and circuit-level noise, the decoding hypergraph is general hypergraph. We note that the (weighted) HUF decoder achieves a higher accuracy than the MWPM decoder, because it considers the hyperedges caused by the $\hat{Y}$ errors. Hyperion achieves even higher accuracy than the HUF decoder.

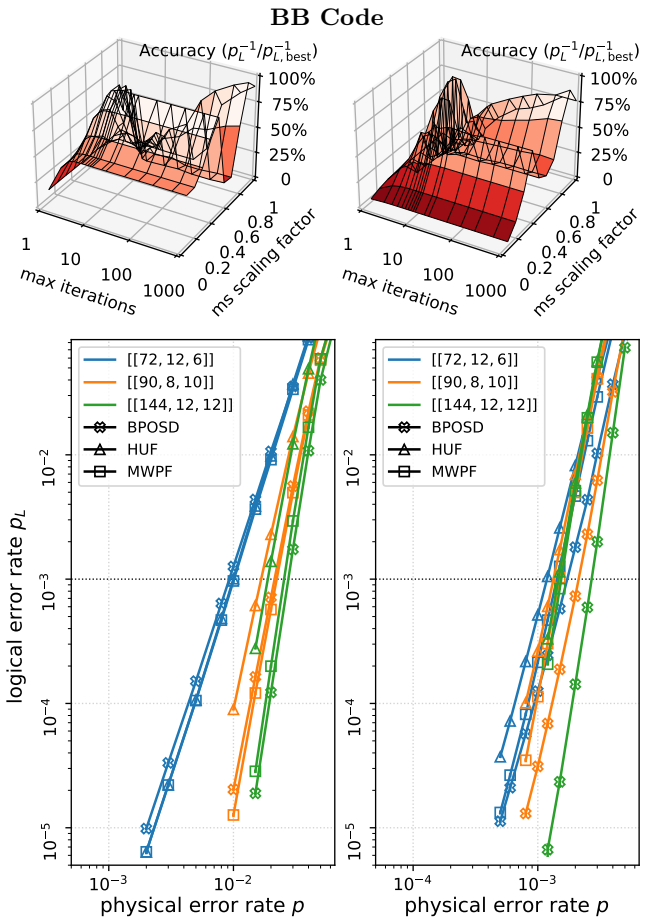
For **color code**, we consider the depolarizing noise and circuit-level noise in Fig. 14, whose decoding hypergraphs are both general hypergraphs. While the HUF decoder is generally not as accurate as the C-MWPM decoder [42], Hyperion achieves higher accuracy than the C-MWPM decoder. The circuit-level noise of the color code is suboptimal in terms of the effective code distance, as shown by the slope of the logical error rate curve in Fig. 14b. Nonetheless, Hyperion achieves the same effective code distance as the C-MWPM decoder. The HUF decoder, on the other hand, fails to achieve the same effective code distance in some cases, e.g., for $d = 5$.

For the **non-topological code**, we consider the the BB code in Fig. 15 as an example. As the baseline, we use the BPOSD decoder [8, 9], which is the state-of-the-art decoder for non-topological code. The BPOSD decoder first uses Belief Propagation (BP) to estimate a posterior distribution of the hyperedges. It then finds a parity factor by enumerating hyperedges by their weights, which indicates the error probabilities, henceforth Ordered Statistics Decoding (OSD). Since the BP process is suboptimal in decoding quantum codes, tuning the BP



(a) depolarizing noise (b) circuit-level noise

FIG. 14: Hyperion (MWPF) achieves higher accuracy than the C-MWPM decoder on color code under both code-capacity depolarizing noise and circuit-level noise.



(a) bit-flip noise (b) circuit-level noise

FIG. 15: Comparison of Hyperion (MWPF), HUF and BPOSD decoders on BB code under code-capacity and circuit-level noise. (top) exhaustive search for optimal BP parameters on the $[[90, 8, 10]]$ BB code at $p = 0.02$ code-capacity noise and $p = 0.1\%$ circuit-level noise. (bottom) using these optimal BP parameters, Hyperion is more accurate than the BPOSD decoder on small BB codes and at low physical error rates. However, at larger code sizes or higher error rates, BPOSD yields better accuracy.

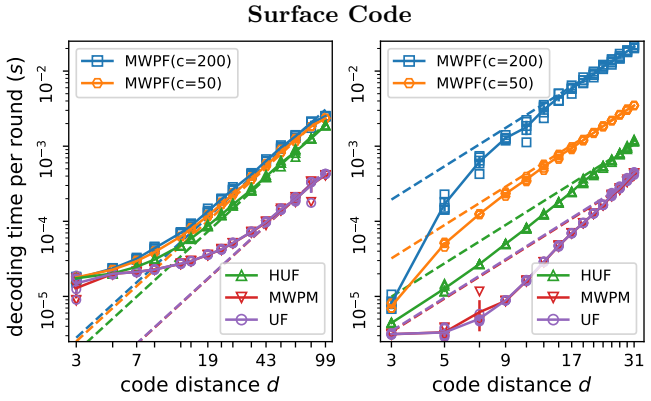
parameters is essential to achieve a high accuracy. As shown in the top figures of Fig. 15, we exhaustively search for the optimal BP parameters for the $[[90, 8, 10]]$ code. We use the physical error rate of $p = 0.02$ and $p = 0.1\%$ for the bit-flip noise model and the circuit-level noise, respectively. We find that a min-sum scaling factor of 0.9 and a large number of iterations (≥ 1000) achieves the highest accuracy. Note that this fine-tuned parameter is not necessarily optimal for other code distances and physical error rates. We compare Hyperion with this fine-tuned BPOSD decoder for BB codes. As shown in the bottom figures of Fig. 15, Hyperion achieves higher accuracy than the BPOSD decoder for small code distances, while the BPOSD decoder achieves higher accuracy than Hyperion for large code distances. Especially, for complicated circuit-level noise, the BPOSD decoder is orders of magnitude more accurate than Hyperion. It remains future work to explore better *Relaxer* finders for BB codes.

C. Decoding Speed

We evaluate the decoding speed of Hyperion on the surface code and the color code with both depolarizing noise and circuit-level noise.

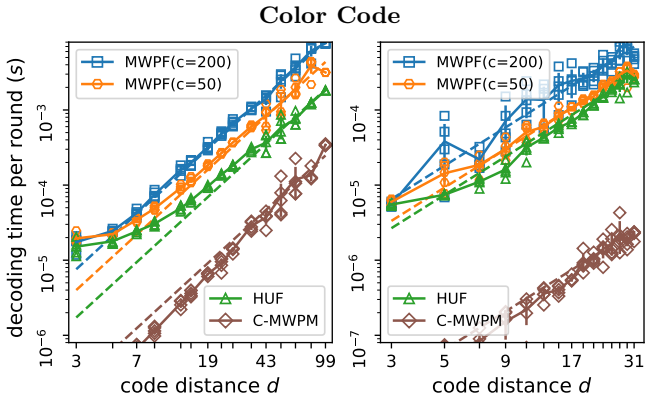
As shown in Fig. 16 and Fig. 17, Hyperion with a fixed per-*Cluster* limit of $c = 50$ and $c = 200$ achieves an almost-linear average decoding time as the code distance increases in all four code-noise combinations. This can be explained by the fact that larger *Clusters* are exponentially rare when the physical error rate is sufficiently low. For each d and p combination, we repeat the measurement for 5 times to show the variance of the average decoding time. For each data point, we sample at least 300k and 10k shots for the code-capacity noise and circuit-level noise, respectively.

We further look into the distribution of the decoding time. We plot the histogram of the decoding time in Fig. 18 for various c choices, each with 10^9 samples. We adopt the concept of k-cutoff decoding time in Micro Blossom [29] which defines the maximum decoding time when at most $(1 + k)p_L$ logical error rate can be tolerated. The cutoff decoding time is orders of mag-



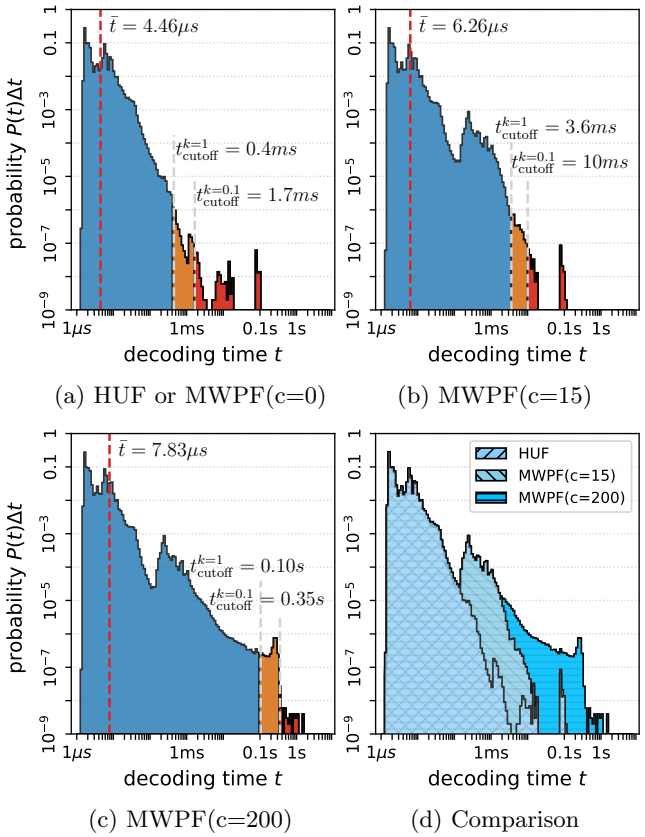
(a) depolarizing $p = 0.01$ (b) circuit-level $p = 10^{-3}$

FIG. 16: HYPERION (MWPF) achieves an almost-linear time scaling on surface code. Each dashed line fits the linear decoding time scaling with the number of physical qubits.



(a) depolarizing $p = 0.01$ (b) circuit-level $p = 10^{-4}$

FIG. 17: HYPERION (MWPF) achieves an almost-linear time scaling on color code.



(a) HUF or MWPF(c=0) (b) MWPF(c=15)

(c) MWPF(c=200) (d) Comparison

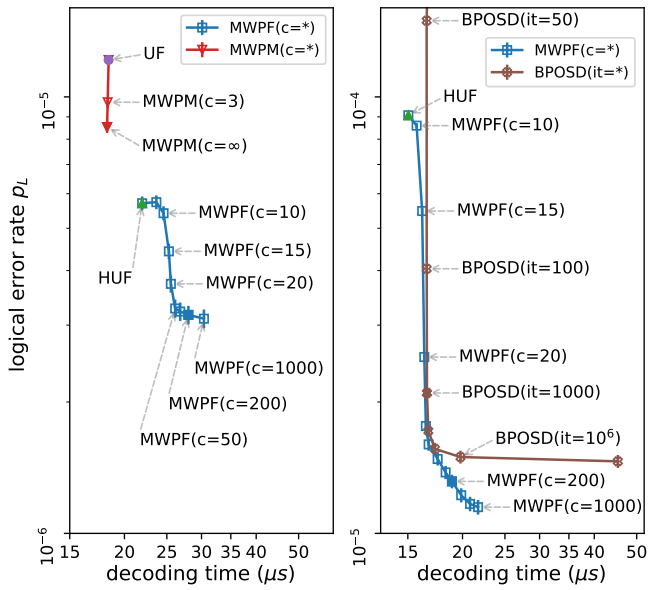
FIG. 18: Decoding runtime distribution ($d = 7$ surface code with $p = 0.01$ depolarizing noise). The MWPF decoder family (HYPERION) exhibits an exponential distribution of decoding time, with a noticeable tail lift at higher runtimes when the per-*Cluster* limit c is large. Note that the figure is plotted in log-log scale, so the area under the curve does not correspond to cumulative probability.

nitude larger than the average decoding time, resulting in milliseconds to seconds worst-case latency to achieve the desired accuracy. This is not favorable for real-time applications but we hope hardware acceleration like that in [29] can speed up MWPF decoders in future work. One interesting observation is that the larger-c increases the probability of higher decoding time while keeping low decoding time distribution largely unchanged in the log-scale plotting. This is because the HYPERBLOSSOM algorithm skips working on the *Locally Optimal Clusters* and focuses on those suboptimal *Clusters* that requires more refinement, i.e., a large primal-dual gap.

D. Accuracy-Speed Trade-off

With the tunable parameter of the per-*Cluster* limit c, HYPERION provides easy trade-off between decoding accuracy and decoding speed. We evaluate the trade-off for the surface code and the BB code, both with the code-capacity noise model.

For the surface code, we look at the MWPF(c=*) decoder family (HYPERION) and the MWPM(c=*) decoder family (Parity Blossom) (§VI A). We choose an intermediate code distance of $d = 7$ and a depolarizing error rate of $p = 0.01$. As shown in Fig. 19a, within the MWPF decoder family, the MWPF(c=200) decoder achieves 1.8x lower logical error rate at the cost of 1.3x longer decoding time compared to the MWPF(c=0) decoder. This demonstrates the usefulness of the *SingleHair Relaxer* finder beyond the trivial *UnionFind Relaxer* finder. Furthermore, we see an S-shaped curve for the MWPF decoder family. This is because for small c, HYPERION may not invoke the *SingleHair Relaxer* finder before a *Cluster* hits the limit. On the other hand, the logical error rate saturates at a large $c \geq 1000$ due to the limitation of the *SingleHair Relaxer* finder (§B 3). Comparing between the two decoder families, we see that the MWPF decoder family is more accurate but slower compared to



(a) $d = 7$ Surface Code (depolarizing noise $p = 0.01$)

(b) [[90, 8, 10]] BB Code (bit-flip noise $p = 0.01$)

FIG. 19: Trade-off between speed and decoding accuracy. (a) for surface code, MWPF decoders are more accurate but slower compared to MWPM decoders. (b) for BB code, MWPF decoders are both faster and more accurate than BPOSD decoders.

the MWPM decoder family.

For the BB code, we look at the MWPF(c=*) decoder family and the BPOSD(max_iter=*) decoder family. We use a min-sum scaling factor of 0.9 for the BPOSD decoders, which is optimal in an exhaustive search as shown in Fig. 15. We choose an intermediate code size of $[[90, 8, 10]]$ and a bit-flip error rate of $p = 0.01$, as shown in Fig. 19b. Within the MWPF decoder family, the MWPF(c=200) decoder achieves 6.9x lower logical error rate at the cost of 1.3x longer decoding time compared to the MWPF(c=0) decoder. We observe that the curve is not saturating at $c = 1000$, which suggests that larger c may further improve the accuracy. On the other hand, the BPOSD(max_iter) decoder family saturates at a maximum iteration of 10^6 yet still falls behind the MWPF(c=200) decoder in terms of the decoding accuracy. For this specific code and noise model, the MWPF decoder family is both faster and more accurate than the BPOSD decoder family. We note that larger BB codes beyond $[[90, 8, 10]]$ may observe that the BPOSD decoder family is better than the MWPF decoder family, as shown in Fig. 15.

VII. RELATED WORK

Graph-based Decoders model the decoding problem as a graph problem and use graph algorithms to solve it. MWPM (§II C) and Union-Find [6] are two popular graph-based decoders on simple graphs. For simple graphs, fast software [3, 4, 47] and hardware [29, 48–55] implementations have reduced the decoding time to sub-microseconds for real-time decoding on superconducting qubits with μs-level measurement round [56]. Their hypergraph generalization, MWPF (§III) and Hypergraph Union-Find (HUF) [7], are generally orders of magnitude slower than the simple graph decoders. However, they achieve higher decoding accuracy and are suitable for real-time decoding on trapped-ion [57–59] or neutral-atom [60, 61] qubits with ms-level measurement round.

Linear-Programming Decoders on classical LDPC codes [62] achieve a comparable decoding accuracy to the belief propagation (BP) decoder while providing stronger theoretical guarantees. Consequent work [63] accelerates the decoding process using decomposition methods. We use similar ideas to provide strong proximity bound (§III B) and uses clustering technique (§III F) to decompose a large problem adaptively. Generalizing LP decoders to quantum codes is non-trivial. Recent work [14] builds on a syndrome-oriented LP problem but does not work well for surface codes. In contrast, [64] and HyperBlossom build on an error-oriented LP problem.

ILP and MaxSAT Decoders use integer constraints to solve the MWPF problem but are generally slower. For example, [65–67] introduce extra integer variables to describe the MWPF problem as an integer linear programming problem. Alternatively, [17] uses MaxSAT solvers which natively support boolean variables. However, these decoders rely on external solvers which provide less control over the decoding process. Our HyperBlossom algorithm is a specialized MWPF solver that provides finer control with the *clustering* technique (§III F) and prioritized *Cluster* refinement (§V).

Heuristic Decoders provide fewer theoretical guarantees but are usually faster and can be tuned to high accuracy. Belief Propagation (BP) is near-optimal on some classical LDPC codes but is not suitable for QEC decoding alone, due to quantum degeneracy [68–71]. Thus, one typically uses BP decoder as a pre-processing step to generate a posterior distribution and uses another heuristic post-processing decoder to determine a correction [8, 16, 43, 69, 72–77]. Neural network decoders [78–82] are another promising approach for decoding quantum codes but they require prohibitively long training for large code distances. Search-based decoders [18, 83] also use heuristics to accelerate exploration of the parity factor solution space.

VIII. CONCLUSION AND FUTURE WORK

HyperBlossom is a new framework of certifying MLE decoding for quantum LDPC codes. It is a generalization of the blossom algorithm and existing graph-based decoders of UF, MWPM and HUF. We propose two im-

portant algorithmic designs: *relaxing* for simplifying the problem and *clustering* for faster decoding. By producing a certified proximity bound for each syndrome it decodes, HyperBlossom provides additional information that is helpful for theoretical analysis of QEC codes.

Several directions remain open for future work. First, developing specialized *Relaxer*-finding algorithms tailored to specific code families may further improve both accuracy and speed. Second, exploring parallelization strategies, including coarse-grained parallelization with fusion [4, 84] and fine-grained parallelization with hardware accelerator [29, 48, 52], may future boost the decoding speed. Finally, it is of theoretical interest to investigate whether $\mathbf{min\,LP} = \mathbf{min\,ILP}$ holds for all or some types of hypergraphs. This question is important because a proof that it holds would justify pursuing optimal relaxer-finding algorithms for general hypergraphs, whereas counterexamples would reveal a fundamental limitation of the HyperBlossom framework: it would not be able to certify certain MLE decoding problems.

IX. ACKNOWLEDGEMENT

This work was supported in part by Yale University and NSF MRI Award #2216030. Yue Wu is also grateful for the support from Unitary Foundation. The authors thank Shilin Huang, Arshpreet Singh Maan and Anqi Gong for insightful discussions on how to tune the BP decoder parameters.

[1] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, "Topological quantum memory," *Journal of Mathematical Physics*, 2002.

[2] J. Edmonds, "Paths, trees, and flowers," *Canadian Journal of mathematics*, 1965.

[3] O. Higgott and C. Gidney, "Sparse blossom: correcting a million errors per core second with minimum-weight matching," *Quantum*, 2025.

[4] Y. Wu and L. Zhong, "Fusion Blossom: Fast MWPM decoders for QEC," in *Proc. IEEE Int. Conf. Quantum Computing & Engineering (QCE)*, 2023.

[5] E. Berlekamp, R. McEliece, and H. Van Tilborg, "On the inherent intractability of certain coding problems," *IEEE Transactions on Information Theory*, 1978.

[6] N. Delfosse and N. H. Nickerson, "Almost-linear time decoding algorithm for topological codes," *Quantum*, 2021.

[7] N. Delfosse, V. Londe, and M. E. Beverland, "Toward a union-find decoder for quantum LDPC codes," *IEEE Transactions on Information Theory*, 2022.

[8] J. Roffe, D. R. White, S. Burton, and E. Campbell, "Decoding across the quantum low-density parity-check code landscape," *Physical Review Research*, 2020.

[9] J. Roffe, "LDPC: Python tools for low density parity check codes," 2022. [Online]. Available: https://pypi.org/project/ldpc/

[10] J. Emerson, M. Silva, O. Moussa, C. Ryan, M. Laforest, J. Baugh, D. G. Cory, and R. Laflamme, "Symmetrized characterization of noisy quantum processes," *Science*, 2007.

[11] L. Lovász, "The factorization of graphs. II," *Acta Mathematica Academiae Scientiarum Hungarica*, 1972.

[12] Q. R. Yu and G. Liu, *Graph factors and matching extensions*. Springer, 2010.

[13] J. Akiyama and M. Kano, *Factors and factorizations of graphs: Proof techniques in factor theory*. Springer, 2011.

[14] O. Fawzi, L. Grouès, and A. Leverrier, "Linear programming decoder for hypergraph product quantum codes," in *IEEE Information Theory Workshop (ITW)*, 2021.

[15] Y. Takada, Y. Takeuchi, and K. Fujii, "Highly accurate decoder for topological color codes with simulated annealing," *arXiv preprint arXiv:2303.01348*, 2023.

[16] T. Hillmann, L. Berent, A. O. Quintavalle, J. Eisert, R. Wille, and J. Roffe, "Localized statistics decoding: A parallel decoding algorithm for quantum low-density parity-check codes," *arXiv preprint arXiv:2406.18655*, 2024.

[17] L. Berent, L. Burgholzer, P.-J. H. Derks, J. Eisert, and R. Wille, "Decoding quantum color codes with maxsat," *Quantum*, 2024.

[18] K. R. Ott, B. Hetényi, and M. E. Beverland, "Decision-tree decoders for general quantum LDPC codes," *arXiv preprint arXiv:2502.16408*, 2025.

[19] Y. Wu, N. Liyanage, and L. Zhong, "An interpretation of union-find decoder on weighted graphs," *arXiv preprint arXiv:2211.03288*, 2022.

[20] A. G. Fowler, A. C. Whiteside, and L. C. Hollenberg, "Towards practical classical processing for the surface code," *Physical review letters*, 2012.

[21] J. Edmonds and E. L. Johnson, "Matching, Euler tours and the Chinese postman," *Mathematical programming*, 1973.

[22] V. Kolmogorov, "Blossom v: a new implementation of a minimum cost perfect matching algorithm," *Mathematical Programming Computation*, 2009.

[23] L. J. Stockmeyer and V. V. Vazirani, "Np-completeness of some generalizations of the maximum matching problem," *Information Processing Letters*, 1982.

[24] S. Stein, S. Xu, A. W. Cross, T. J. Yoder, A. Javadi-Abhari, C. Liu, K. Liu, Z. Zhou, C. Guinn, Y. Ding *et al.*, "HetEC: Architectures for heterogeneous quantum error correction codes," in *Proc. ACM Int. Conf. Architectural Support for Programming Languages & Operating Systems (ASPLOS)*, 2025.

[25] G. B. Dantzig, "Maximization of a linear function of variables subject to linear inequalities," *Activity analysis of production and allocation*, 1951.

[26] J. Matoušek and B. Gärtner, *Understanding and using linear programming*. Springer, 2007.

[27] W. L. Winston, *Operations research: applications and algorithm*. Thomson Learning, Inc., 2004.

[28] A. G. Fowler, "Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $O(1)$ parallel time," *arXiv preprint*

arXiv:1307.1740, 2013.

[29] Y. Wu, N. Liyanage, and L. Zhong, "Micro Blossom: Accelerated minimum-weight perfect matching decoding for quantum error correction," in *Proc. ACM Int. Conf. Architectural Support for Programming Languages & Operating Systems (ASPLOS)*, 2025.

[30] A. Farrugia, P. Mihók, R. B. Richter, and G. Semanivšin, "Factorizations and characterizations of induced-hereditary and compositive properties," *Journal of Graph Theory*, 2005.

[31] N. Karmarkar, "A new polynomial-time algorithm for linear programming," in *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, 1984.

[32] S. J. Leon, L. De Pillis, and L. G. De Pillis, *Linear algebra with applications*. Prentice Hall Upper Saddle River, NJ, 1998.

[33] A. deMarti iOlius, P. Fuentes, R. Orús, P. M. Crespo, and J. E. Martinez, "Decoding algorithms for surface codes," 2024.

[34] S. Huang, M. Newman, and K. R. Brown, "Fault-tolerant weighted union-find decoding on the toric code," *Physical Review A*, 2020.

[35] C. Gidney, "Stim: a fast stabilizer circuit simulator," *Quantum*, 2021.

[36] Q. Huangfu and J. J. Hall, "Parallelizing the dual revised simplex method," *Mathematical Programming Computation*, 2018.

[37] P. Kumar, "SLP," 2020. [Online]. Available: https://crates.io/crates/slp

[38] Y. Wu, "Examples of visualization in MWPF package," Apr. 2025. [Online]. Available: https://doi.org/10.5281/zenodo.15243982

[39] D. K. Tuckett, A. S. Darmawan, C. T. Chubb, S. Bravyi, S. D. Bartlett, and S. T. Flammia, "Tailoring surface codes for highly biased noise," *Physical Review X*, 2019.

[40] H. Bombin and M. A. Martin-Delgado, "Topological quantum distillation," *Physical review letters*, 2006.

[41] S. Bravyi, A. W. Cross, J. M. Gambetta, D. Maslov, P. Rall, and T. J. Yoder, "High-threshold and low-overhead fault-tolerant quantum memory," *Nature*, 2024.

[42] C. Gidney and C. Jones, "New circuits and an open source decoder for the color code," *arXiv preprint arXiv:2312.08813*, 2023.

[43] A. Gong, S. Cammerer, and J. M. Renes, "Toward low-latency iterative decoding of QLDPC codes under circuit-level noise," *arXiv preprint arXiv:2403.18901*, 2024.

[44] C. Gidney, "Chromobius: color code decoder," 2023. [Online]. Available: https://github.com/quantumlib/chromobius

[45] Y. Wu, "T-mwpm decoder implemented as tailoredmwpmdecoder in QEC Playground," 2023. [Online]. Available: https://github.com/yuewuo/QEC-Playground

[46] P.-K. Tsai, Y. Wu, and S. Puri, "Mitigating temporal fragility in the xy surface code," *Physical Review X*, 2024.

[47] O. Higgott, T. C. Bohdanowicz, A. Kubica, S. T. Flammia, and E. T. Campbell, "Improved decoding of circuit noise and fragile boundaries of tailored surface codes," *Physical Review X*, 2023.

[48] N. Liyanage, Y. Wu, A. Deters, and L. Zhong, "Scalable quantum error correction for surface codes using FPGA," in *Proc. IEEE Int. Conf. Quantum Computing & Engineering (QCE)*, 2023.

[49] A. B. Ziad, A. Zalawadiya, C. Topal, J. Camps, G. P. Gehér, M. P. Stafford, and M. L. Turner, "Local clustering decoder: a fast and adaptive hardware decoder for the surface code," *arXiv preprint arXiv:2411.10343*, 2024.

[50] N. Liyanage, Y. Wu, S. Tagare, and L. Zhong, "FPGA-based distributed union-find decoder for surface codes," *IEEE Transactions on Quantum Engineering*, 2024.

[51] B. Barber, K. M. Barnes, T. Bialas, O. Buğdaycı, E. T. Campbell, N. I. Gillespie, K. Johar, R. Rajan, A. W. Richardson, L. Skoric *et al.*, "A real-time, scalable, fast and resource-efficient decoder for a quantum computer," *Nature Electronics*, 2025.

[52] N. Liyanage, Y. Wu, E. Houghton, and L. Zhong, "Network-integrated decoding system for real-time quantum error correction with lattice surgery," in *Proc. IEEE Int. Conf. Quantum Computing & Engineering (QCE)*, 2025.

[53] P. Das, A. Locharla, and C. Jones, "LILLIPUT: a lightweight low-latency lookup-table decoder for near-term quantum error correction," in *Proc. ACM Int. Conf. Architectural Support for Programming Languages & Operating Systems (ASPLOS)*, 2022.

[54] R. W. Overwater, M. Babaie, and F. Sebastiano, "Neural-network decoders for quantum error correction using surface codes: A space exploration of the hardware cost-performance tradeoffs," *IEEE Transactions on Quantum Engineering*, 2022.

[55] D. Ristè, L. C. Govia, B. Donovan, S. D. Fallek, W. D. Kalfus, M. Brink, N. T. Bronn, and T. A. Ohki, "Real-time processing of stabilizer measurements in a bit-flip code," *npj Quantum Information*, 2020.

[56] "Exponential suppression of bit or phase errors with cyclic error correction," *Nature*, 2021.

[57] L. Egan, D. M. Debroy, C. Noel, A. Risinger, D. Zhu, D. Biswas, M. Newman, M. Li, K. R. Brown, M. Cetina *et al.*, "Fault-tolerant control of an error-corrected qubit," *Nature*, 2021.

[58] M. Webber, V. Elfving, S. Weidt, and W. K. Hensinger, "The impact of hardware specifications on reaching quantum advantage in the fault tolerant regime," *AVS Quantum Science*, 2022.

[59] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown *et al.*, "Realization of real-time fault-tolerant quantum error correction," *Physical Review X*, 2021.

[60] D. Bluvstein, S. J. Evered, A. A. Geim, S. H. Li, H. Zhou, T. Manovitz, S. Ebadi, M. Cain, M. Kalinowski, D. Hangleiter *et al.*, "Logical quantum processor based on reconfigurable atom arrays," *Nature*, 2024.

[61] L. Henriet, L. Beguin, A. Signoles, T. Lahaye, A. Browaeys, G.-O. Reymond, and C. Jurczak, "Quantum computing with neutral atoms," *Quantum*, 2020.

[62] J. Feldman, M. J. Wainwright, and D. R. Karger, "Using linear programming to decode binary linear codes," *IEEE Transactions on Information Theory*, 2005.

[63] S. Barman, X. Liu, S. C. Draper, and B. Recht, "Decomposition methods for large scale lp decoding," *IEEE Transactions on Information Theory*, 2013.

[64] J. X. Li and P. O. Vontobel, "Lp decoding of quantum stabilizer codes," in *IEEE International Symposium on Information Theory (ISIT)*, 2018.

[65] A. J. Landahl, J. T. Anderson, and P. R. Rice, "Fault-tolerant quantum computing with color codes," *arXiv preprint arXiv:1108.5738*, 2011.

[66] N. Lacroix, A. Bourassa, F. J. Heras, L. M. Zhang, J. Bausch, A. W. Senior, T. Edlich, N. Shutty, V. Sivak, A. Bengtsson *et al.*, "Scaling and logic in the color code on a superconducting quantum processor," *Nature*, 2025.
[67] M. Cain, C. Zhao, H. Zhou, N. Meister, J. P. B. Ataides, A. Jaffe, D. Bluvstein, and M. D. Lukin, "Correlated decoding of logical algorithms with transversal gates," *Physical Review Letters*, 2024.
[68] D. Poulin and Y. Chung, "On the iterative decoding of sparse quantum codes," *arXiv preprint arXiv:0801.1241*, 2008.
[69] P. Panteleev and G. Kalachev, "Degenerate quantum LDPC codes with good finite length performance," *Quantum*, 2021.
[70] N. Raveendran and B. Vasić, "Trapping sets of quantum LDPC codes," *Quantum*, 2021.
[71] K. D. Morris, T. Pllaha, and C. A. Kelley, "Absorbing sets in quantum LDPC codes," *arXiv preprint arXiv:2307.14532*, 2023.
[72] A. Grospellier, L. Grouès, A. Krishna, and A. Leverrier, "Combining hard and soft decoders for hypergraph product codes," *Quantum*, 2021.
[73] S. Wolanski and B. Barber, "Ambiguity clustering: an accurate and efficient decoder for QLDPC codes," *arXiv preprint arXiv:2406.14527*, 2024.
[74] A. d. iOlius, I. E. Martinez, J. Roffe, and J. E. Martinez, "An almost-linear time decoding algorithm for quantum ldpc codes under circuit-level noise," *arXiv preprint arXiv:2409.01440*, 2024.
[75] M. P. Fossorier, S. Lin, and J. Snyders, "Reliability-based syndrome decoding of linear block codes," *IEEE Transactions on Information Theory*, 1998.
[76] M. P. Fossorier and S. Lin, "Soft-decision decoding of linear block codes based on ordered statistics," *IEEE Transactions on information Theory*, 2002.
[77] A. S. Maan and A. Paler, "Machine learning message-passing for the scalable decoding of QLDPC codes," *npj Quantum Information*, 2025.
[78] S. Varsamopoulos, B. Criger, and K. Bertels, "Decoding small surface codes with feedforward neural networks," *Quantum Science and Technology*, 2017.
[79] A. Gong, S. Cammerer, and J. M. Renes, "Graph neural networks for enhanced decoding of quantum LDPC codes," in *IEEE International Symposium on Information Theory (ISIT)*, 2024.
[80] J. Bausch, A. W. Senior, F. J. Heras, T. Edlich, A. Davies, M. Newman, C. Jones, K. Satzinger, M. Y. Niu, S. Blackwell *et al.*, "Learning to decode the surface code with a recurrent, transformer-based neural network," *arXiv preprint arXiv:2310.05900*, 2023.
[81] M. Lange, P. Havström, B. Srivastava, I. Bengtsson, V. Bergentall, K. Hammar, O. Heuts, E. van Nieuwenburg, and M. Granath, "Data-driven decoding of quantum error correcting codes using graph neural networks," *Physical Review Research*, 2025.
[82] B. M. Varbanov, M. Serra-Peralta, D. Byfield, and B. M. Terhal, "Neural network decoder for near-term surface-code experiments," *Physical Review Research*, 2025.
[83] L. A. Beni, O. Higgott, and N. Shutty, "Tesseract: A search-based decoder for quantum error correction," *arXiv preprint arXiv:2503.10988*, 2025.
[84] L. Yang, Y. Wu, and L. Zhong, "Parallel minimum-weight parity factor decoding for quantum error correction," in *Proc. IEEE Int. Conf. Quantum Computing & Engineering (QCE)*, 2024.

Appendix A: HyperBlossom Framework Proofs

In this section, we prove the theorems presented in the HyperBlossom framework (§III). We first prove **Theorem: min ILP = min MWPF** (§III B) in §A 1. We then prove the optimality of the *relaxing* algorithm (§III E) in §A 2 and the optimality of the *clustering* algorithm (§III F) in §A 3. Putting together, we prove the optimality of the HyperBlossom algorithm (§III G) in §A 4. Finally, we show the necessity of E_S in the definition of *Invalid* subgraph $S = (V_S, E_S)$ in §A 5.

1. minILP = minMWPF

Theorem: min ILP = min MWPF.

Proof. To prove the equality of the optimal values of ILP and MWPF, we prove that $\min \text{ILP} \leq \min \text{MWPF}$ and $\min \text{MWPF} \leq \min \text{ILP}$.

Given *Lemma: Any parity factor is a feasible ILP solution*, ILP is a relaxation of MWPF, i.e., $\min \text{ILP} \leq \min \text{MWPF}$.

We then prove *Lemma: For every feasible ILP solution $\vec{x}$, there exists a parity factor $\mathcal{E}$ with no greater weight, i.e., $W(\mathcal{E}) \leq \sum_{e \in E} w_e x_e$.* In this way, the minimum value of MWPF is no larger than the minimum value of ILP, i.e., $\min \text{MWPF} \leq \min \text{ILP}$.

Together, we prove $\min \text{ILP} = \min \text{MWPF}$. □

Lemma: Any parity factor is a feasible ILP solution.

Proof. Given any parity factor $\vec{x} \in \mathbb{F}_2^{|E|}$, we prove the lemma by contradiction. Suppose $\vec{x}$ is not a feasible ILP solution, then there exists an *Invalid* subgraph $S \in \mathcal{O}$ where the solution $\vec{x}$ violates the corresponding ILP constraint Eq. (2b), i.e.,

$$\exists S = (V_S, E_S) \in \mathcal{O}, \sum_{e \in \delta(S)} x_e < 1$$

Since $x_e \in \mathbb{F}_2 = \{0, 1\}$ in ILP, we have

$$x_e = 0, \forall e \in \delta(S)$$

Given $\vec{x}$ is a parity factor, it must satisfy the parity constraints of V_S. That is, let $\mathcal{E}_S = \{e \in E(V_S) | x_e = 1\}$, we have

$$\mathcal{D}(\mathcal{E}_S) \cap V_S = D \cap V_S$$

We then transform the left hand side into a simpler form. By definition of *Hair* and *Invalid* subgraph, we have $\delta(S) = E(V_S) \setminus E_S$ and $E_S \subseteq E[V_S] \subseteq E(V_S)$. Together, we have $E(V_S) = E_S \cup \delta(S)$. Given $x_e = 0, \forall e \in \delta(S)$, we have $\mathcal{E}_S = \{e \in E(V_S) | x_e = 1\} = \{e \in E_S | x_e = 1\}$. Since $\mathcal{E}_S \subseteq E_S \subseteq E[V_S]$, it only generates defects within V_S, i.e., $\mathcal{D}(\mathcal{E}_S) \cap V_S = \mathcal{D}(\mathcal{E}_S)$. Thus, we have

$$\mathcal{D}(\mathcal{E}_S) = D \cap V_S$$

However, this violates the definition of *Invalid* subgraph: $\forall \mathcal{E} \subseteq E_S, \mathcal{D}(\mathcal{E}) \neq D \cap V_S$. Thus, the assumption that $\vec{x}$ violates an ILP constraint is wrong, i.e., $\vec{x}$ satisfies all ILP constraints and thus it is a feasible ILP solution. □

Lemma: For every feasible ILP solution $\vec{x}$, there exists a parity factor $\mathcal{E}$ with no greater weight, i.e., $W(\mathcal{E}) \leq \sum_{e \in E} w_e x_e$.

Proof. We first prove that for every feasible ILP solution $\vec{x}$, there exists $\mathcal{E} \subseteq E_S$ where $E_S = \{e \in E | x_e = 1\}$, such that $\mathcal{D}(\mathcal{E}) = D$. We prove this by contradiction. Suppose $\forall \mathcal{E} \subseteq E_S, \mathcal{D}(\mathcal{E}) \neq D$. Given $E_S \subseteq E = E[V]$, we have $S = (V, E_S)$ is an *Invalid* subgraph according to the definition, i.e., $S \in \mathcal{O}$. There is an ILP constraint Eq. (2b) for every *Invalid* subgraph $S \in \mathcal{O}$ that says $\sum_{e \in \delta(S)} x_e \geq 1$. However, according to the definition of *Hair*, $\delta(S) = E(V_S) \setminus E_S$, we have $\sum_{e \in \delta(S)} x_e = 0$. This contradicts the assumption that $\vec{x}$ satisfies all the ILP constraints Eq. (2b), including the one corresponding to S. Thus, there must exist $\mathcal{E} \subseteq E_S$ such that $\mathcal{D}(\mathcal{E}) = D$, i.e., $\mathcal{E}$ is a feasible MWPF solution.

We then prove that $W(\mathcal{E}) \leq \sum_{e \in E} w_e x_e$. This is trivial because the weights of the hypergraph are non-negative $w_e \geq 0, \forall e \in E$. Thus, given $\mathcal{E} \subseteq E_S$, we have $W(\mathcal{E}) = \sum_{e \in \mathcal{E}} w_e \leq \sum_{e \in E_S} w_e = \sum_{e \in E} w_e x_e$.

We shall note that not all feasible ILP solutions are parity factors, in contrast to *Lemma: Any parity factor is a feasible ILP solution*. For example, $x_e = 1, \forall e \in \mathcal{E}$ is clearly a feasible ILP solution, but $E = \{e \in E | x_e = 1\}$ is usually not a parity factor. One might instead expect that optimal ILP solutions are parity factors, but this is not true either. For example, given an optimal ILP solution $\vec{x}$, changing $x_e = 0$ to $x_e = 1$ for any zero-weighted hyperedge $e_0 \in E, w_{e_0} = 0$ leads to another optimal ILP solution $\vec{x'}$. This is because $\vec{x'}$ is clearly a feasible ILP solution, and also $\sum_{e \in E} w_e x'_e = \sum_{e \in E} w_e x_e + w_{e_0} = \sum_{e \in E} w_e x_e$ thus $\vec{x'}$ is optimal. However, $\mathcal{E}' = \{e \in E | x'_e = 1\}$ is not a parity factor, because $\mathcal{D}(\mathcal{E}') = \mathcal{D}(\{e \in E | x_e = 1\}) \oplus \{e_0\} = D \oplus \{e_0\} \neq D$. □

2. Optimality of Relaxing

Theorem: Relaxing. Given a suboptimal DLP $\vec{y}$, there exists a *Relaxer* or a *Trivial Direction*, or both.

Proof. Since $\vec{y}$ is suboptimal, there exists an optimal DLP solution $\vec{y^o}$ with $\sum \vec{y^o} > \sum \vec{y}$. We use $\vec{y^o}$ to construct either a *Relaxer* or a *Trivial Direction*.

Given the linearity of DLP, $\Delta\vec{y} \coloneqq \vec{y^o} - \vec{y}$ is a *Feasible Direction*, i.e., we can grow along $\Delta\vec{y}$ for a positive length $l > 0$. Since $\sum \Delta\vec{y} = \sum \vec{y^o} - \sum \vec{y} > 0$,

there exists at least one *Invalid* subgraph $S^+ \in \mathcal{O}$ with $\Delta y_{S^+} > 0$. We can then decompose the *Direction* $\Delta\vec{y}$ into two: $\Delta^a\vec{y}$ and $\Delta^b\vec{y}$. The *Direction* $\Delta^a\vec{y}$ consists of a single *Invalid* subgraph S^+ growing at a rate of $\Delta^a y_{S^+} \coloneqq \min(\Delta y_{S^+}, \sum \Delta\vec{y}) > 0$. The other *Direction* is defined as $\Delta^b\vec{y} \coloneqq \Delta\vec{y} - \Delta^a\vec{y}$. Depending on whether $\Delta^b\vec{y}$ relaxes any *Tight Edges*, we prove that either $\Delta^b\vec{y}$ is a *Relaxer*, or $\Delta^a\vec{y}$ is a *Trivial Direction* up to a scaling factor.

First, if $\Delta^b\vec{y}$ relaxes at least one *Tight Edge*, i.e., $\exists e \in T, \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta^b y_S < 0$ (Eq. (7a)), then we prove that $\Delta^b\vec{y}$ is a *Relaxer* by satisfying the other conditions Eq. (7b), (6a) and (6b). The first condition Eq. (7b) requires $\sum_{S\in\mathcal{O}} \Delta^b y_S \geq 0$.

$$\begin{aligned}
\Delta^b\vec{y} &= \Delta\vec{y} - \Delta^a\vec{y} \\
\sum_{S\in\mathcal{O}} \Delta^b y_S &= \left(\sum_{S\in\mathcal{O}} \Delta\vec{y}\right) - \Delta^a y_S \\
&= \left(\sum_{S\in\mathcal{O}} \Delta y_S\right) - \min\left(\Delta y_S, \sum_{S\in\mathcal{O}} \Delta y_S\right) \\
&\geq 0
\end{aligned}$$

We then prove Eq. (6a): $\forall S \in \mathcal{O}\setminus\mathcal{B}, \Delta^b y_S \geq 0$. We consider two cases when iterating every $S \in \mathcal{O}\setminus\mathcal{B}$. When $S = S^+$, we have $\Delta^b y_{S^+} = \Delta y_{S^+} - \Delta^a y_{S^+} = \Delta y_{S^+} - \min(\Delta y_{S^+}, \sum \Delta\vec{y}) \geq 0$. Otherwise, $S \in \mathcal{O}\setminus\mathcal{B}\setminus\{S^+\}$ thus $\Delta^a y_S = 0$ and $y_S = 0$. We have $\Delta^b y_S = \Delta y_S - \Delta^a y_S = \Delta y_S = y_S^o - y_S = y_S^o \geq 0$. Thus, in both cases, we have

$$\forall S \in \mathcal{O}\setminus\mathcal{B}, \Delta^b y_S \geq 0$$

Lastly Eq. (6b): $\forall e \in T, \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta^b y_S \leq 0$. Since $\Delta^b\vec{y} = \Delta\vec{y} - \Delta^a\vec{y}$ and $\forall S \in \mathcal{O}, \Delta^a y_S \geq 0$, we have $\forall S \in \mathcal{O}, \Delta^b y_S = \Delta y_S - \Delta^a y_S \leq \Delta y_S$.

$$\begin{aligned}
\forall e \in T, \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta^b y_S &= \sum_{S\in\mathcal{O}|e\in\delta(S)} (\Delta y_S - \Delta^a y_S) \\
&\leq \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta y_S \\
&\leq 0 \quad (\text{given } \Delta\vec{y} \text{ is } \textit{Feasible})
\end{aligned}$$

On the other hand, if $\Delta^b\vec{y}$ does not relax any *Tight Edge*, i.e., $\forall e \in T, \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta^b y_S \geq 0$, we prove that $\Delta^a\vec{y}$ is a *Trivial Direction* up to a scaling factor. Clearly $\Delta^a\vec{y} = \{\Delta y_S : \min(\Delta y_S, \sum \Delta\vec{y})\}$ trivially contains a single growing *Invalid* subgraph S. We then only need to prove that $\Delta^a\vec{y}$ is a *Feasible Direction*, i.e., it satisfies Eqs. (6a) and (6b).

$\Delta^a\vec{y}$ clearly satisfies Eq. (6a) because all the elements are non-negative: $\forall S \in \mathcal{O}, \Delta^a y_S \geq 0$.

We then prove $\Delta^a\vec{y}$ satisfies Eq. (6b): it must not grow on any *Tight Edge*, i.e., $\forall e \in T, \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta^a y_S \leq 0$. Given $\Delta\vec{y}$ is a *Feasible Direction*, it must satisfy Eq. (6b). Also, since $\Delta^b\vec{y}$ does not relax any *Tight Edge*, nor can it grow on any *Tight Edge*, we have $\forall e \in T, \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta^b y_S = 0$. Thus, we have

$$\begin{aligned}
\forall e \in T, \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta^a y_S &= \sum_{S\in\mathcal{O}|e\in\delta(S)} \left(\Delta y_S - \Delta^b y_S\right) \\
&\leq \sum_{S\in\mathcal{O}|e\in\delta(S)} -\Delta^b y_S \\
&= 0
\end{aligned}$$

Overall, we have proved that either $\Delta^b\vec{y}$ is a *Relaxer*, or $\Delta^a\vec{y}/\Delta^a y_{S^+}$ is a *Trivial Direction*.

□

Theorem: Batch Relaxing. Given *Relaxers* $R_i'[T], i = 1, ..., n$, if there exists a *Feasible Direction* $\Delta\vec{y}[T \setminus (\cup_i \mathcal{R}_i')]$, we can compose a *Feasible Direction* $\Delta'\vec{y}[T]$ such that $\sum \Delta'\vec{y} \geq \sum \Delta\vec{y}$. In the special case where $\Delta\vec{y}$ is a *Relaxer*, $\Delta'\vec{y}$ is also a *Relaxer* $R'[T]$ such that $\mathcal{R}' \supseteq \mathcal{R}$.

Proof. We explicitly construct the composed *Direction* $\Delta'\vec{y}$ and prove $\sum \Delta'\vec{y} \geq \sum \Delta\vec{y}$.

By definition of *Relaxer*, each edge $e \in \mathcal{R}_i'$ is relaxed when growing along $\Delta^i\vec{y} = R_i'$ for a positive length:

$$\sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta^i y_S < 0, \qquad \forall e \in \mathcal{R}(\Delta^i\vec{y}) \tag{A1}$$

For every edge $e \in \cup_i \mathcal{R}_i'$, there exists a relaxer $\Delta^e\vec{y}$ such that $e \in \mathcal{R}(\Delta^e\vec{y})$. Given Eq. (A1), we have $k_e = \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta^e y_S < 0$. That is, growing along the *Direction* $\Delta^e\vec{y}$ for a small positive length $l > 0$ will relax the edge e by $-k_e l > 0$.

For the *Feasible Direction* $\Delta\vec{y}[T \setminus \cup_i \mathcal{R}_i']$, we have

$$\sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta y_S \leq 0, \qquad \forall e \in T \setminus \cup_i \mathcal{R}_i'$$

However, $\Delta\vec{y}$ may violate the above inequality for *Tight Edges* $\cup_i \mathcal{R}_i' \cap T$. We define the amount of violation as $\alpha_e \coloneqq \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta y_S$, and use $\Delta^e\vec{y}$ to fix the violation of the edge e if the violation is positive $\alpha_e > 0$. We compose the new *Direction* $\Delta'\vec{y}$ as follows:

$$\Delta'\vec{y} = \Delta\vec{y} + \sum_{e\in\cup_i \mathcal{R}_i'\cap T|\alpha_e>0} \frac{\alpha_e}{-k_e} \Delta^e\vec{y} \tag{A2}$$

Before proving that $\Delta'\vec{y}$ satisfies the definition of a *Feasible Direction*, we first prove Eq. (A3) below.

$$\begin{aligned}
&\forall e \in T, \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta^i y_S \leq 0 \quad (\text{given } \{\Delta^i\vec{y}\} \text{ are } \textit{Relaxers}) \\
&\downarrow (\text{given } \forall e' \subseteq \cup_i \mathcal{R}_i', k_{e'} < 0) \\
&\implies \forall e \in T, \sum_{e'\in\cup_i \mathcal{R}_i'\cap T|\alpha_{e'}>0} \frac{\alpha_{e'}}{-k_{e'}} \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta^{e'} y_S \leq 0 \\
&\implies \forall e \in T, \sum_{S\in\mathcal{O}|e\in\delta(S)} \sum_{e'\in\cup_i \mathcal{R}_i'\cap T|\alpha_{e'}>0} \frac{\alpha_{e'}}{-k_{e'}} \Delta^{e'} y_S \leq 0
\end{aligned} \tag{A3}$$

We now prove that $\Delta'\vec{y}$ is a *Feasible Direction*, i.e., it satisfies Eqs. (6a) and (6b). We first prove Eq. (6a): $\forall S \in \mathcal{O} \setminus \mathcal{B}, \Delta' y_S \geq 0$. Since $\Delta\vec{y}$ and $\{\Delta^e\vec{y}\}$ are both *Feasible Directions*, we have $\forall S \in \mathcal{O}\setminus\mathcal{B}, \Delta y_S \geq 0, \Delta^e y_S \geq 0$ given Eq. (6a). Thus, we have

$$\begin{aligned}\forall S \in \mathcal{O} \setminus \mathcal{B}, \Delta' y_S &= \Delta y_S + \sum_{e \in \cup_i \mathcal{R}'_i \cap T | \alpha_e > 0} \frac{\alpha_e}{-k_e} \Delta^e y_S \\ &\geq 0 \quad \text{(every term is non-negative)}\end{aligned}$$

We then prove Eq. (6b): $\forall e \in T, \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta' y_S \leq 0$.

$$\begin{aligned}\forall e \in T, \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta' y_S &= \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta y_S + \\ &\sum_{S\in\mathcal{O}|e\in\delta(S)} \sum_{e' \in \cup_i \mathcal{R}'_i \cap T | \alpha_{e'} > 0} \frac{\alpha_{e'}}{-k_{e'}} \Delta^{e'} y_S \\ &\downarrow \text{(given Eq. (A3))} \\ &\leq \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta y_S \\ &\downarrow \text{(given } \Delta\vec{y} \text{ is a } \textit{Feasible Direction}) \\ &\leq 0\end{aligned}$$

Now we have proved that $\Delta'\vec{y}$ is a *Feasible Direction* with $\sum \Delta'\vec{y} \geq \sum \Delta\vec{y}$. The theorem also states that when $\Delta\vec{y}$ is a *Relaxer*, $\Delta'\vec{y}$ is also a *Relaxer* that relaxes a superset of *Tight Edges* $\mathcal{R}(\Delta'\vec{y}) \supseteq \mathcal{R}(\Delta\vec{y})$. We now prove $\Delta'\vec{y}$ is a *Relaxer*, i.e., it satisfies Eqs. (7a) and (7b). First, we prove Eq. (7a). Since $\mathcal{R}(\Delta\vec{y}) \neq \varnothing$ given $\Delta\vec{y}$ is a *Relaxer*, we then only need to prove $\forall e \in \mathcal{R}(\Delta\vec{y}), \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta' y_S < 0$.

$$\begin{aligned}\forall e \in \mathcal{R}(\Delta\vec{y}), \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta' y_S &= \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta y_S + \\ &\sum_{S\in\mathcal{O}|e\in\delta(S)} \sum_{e' \in \cup_i \mathcal{R}'_i \cap T | \alpha_{e'} > 0} \frac{\alpha_{e'}}{-k_{e'}} \Delta^{e'} y_S \\ &\downarrow \text{(given Eq. (A3))} \\ &\leq \sum_{S\in\mathcal{O}|e\in\delta(S)} \Delta y_S \\ &\downarrow \text{(given } \Delta\vec{y} \text{ is a } \textit{Relaxer} \text{ and } e \in \mathcal{R}(\Delta\vec{y})) \\ &< 0\end{aligned}$$

We next prove Eq. (7b): $\sum_{S\in\mathcal{O}} \Delta' y_S \geq 0$. Given that $\Delta\vec{y}$ and $\{\Delta^e\vec{y}\}$ are *Relaxers*, we have $\sum_{S\in\mathcal{O}} \Delta y_S \geq 0$ and $\sum_{S\in\mathcal{O}} \Delta^e y_S \geq 0$.

$$\begin{aligned}\sum_{S\in\mathcal{O}} \Delta' y_S &= \sum_{S\in\mathcal{O}} \Delta y_S + \sum_{e \in \cup_i \mathcal{R}'_i \cap T | \alpha_e > 0} \frac{\alpha_e}{-k_e} \sum_{S\in\mathcal{O}} \Delta^e y_S \\ &\downarrow \text{(given } \forall e \subseteq \cup_i \mathcal{R}'_i, k_e < 0) \\ &\geq \sum_{S\in\mathcal{O}} \Delta y_S \\ &\downarrow \text{(given } \Delta\vec{y} \text{ is a } \textit{Relaxer}) \\ &\geq 0\end{aligned}$$

Together, we prove that $\Delta'\vec{y}$ is a *Relaxer* $R'[T]$, relaxing a superset of the *Tight Edges* $\mathcal{R}(\Delta'\vec{y}) \supseteq \mathcal{R}(\Delta\vec{y})$. □

3. Optimality Criteria of Clusters

In this section, we prove the **Theorem: Optimality Criteria of Clusters** (§III F) and introduce some lemmas about the properties of *Clusters* that we use to prove the optimality of the HyperBlossom algorithm in §A 4.

Theorem: Optimality Criteria of Clusters. When all the *Clusters* are *Locally Optimal Clusters*, the union of all the local MWPF and DLP solutions are the global optimal MWPF $\mathcal{E} = \cup_{C\in\mathcal{C}}\mathcal{E}_C$ and global optimal DLP solution $\vec{y} = \bigcup_{C\in\mathcal{C}}\{S : y_S | S \in \mathcal{B}_C\}$, respectively.

Proof. To prove the theorem, we first show that the union of all local parity factors $\mathcal{E} = \bigcup_{C\in\mathcal{C}} \mathcal{E}_C$ is a global parity factor according to *Lemma: Cluster Parity Factor*. Also, the union of all local DLP solutions $\vec{y} = \bigcup_{C\in\mathcal{C}}\{S : y_S | \forall S \in \mathcal{B}_C\}$ is a feasible DLP solution according to *Lemma: Cluster DLP Solution*. Thus, we only need to prove that $W(\mathcal{E}) = \sum_{S\in\mathcal{O}} y_S$ and according to **Theorem: Certifying Optimum**, $\mathcal{E}$ and $\vec{y}$ are optimal MWPF and DLP solutions, respectively.

According to the definition of *Locally Optimal Cluster*, we have $W(\mathcal{E}_C) = \sum_{S\in\mathcal{B}_C} y_S$ for every *Cluster* $C \in \mathcal{C}$. Thus, we have

$$\begin{aligned}W(\mathcal{E}) &= W(\bigcup_{C\in\mathcal{C}} \mathcal{E}_C) \\ &\downarrow \text{(by } \textit{Lemma: Cluster Non-overlapping}) \\ &= \sum_{C\in\mathcal{C}} W(\mathcal{E}_C) \\ &= \sum_{C\in\mathcal{C}} \sum_{S\in\mathcal{B}_C} y_S \\ &\downarrow \text{(by } \textit{Lemma: Cluster Non-overlapping}) \\ &= \sum_{S\in\mathcal{B}} y_S \\ &= \sum_{S\in\mathcal{O}} y_S\end{aligned}$$

Together, we prove that $\mathcal{E}$ and $\vec{y}$ are optimal MWPF and DLP solutions, respectively.

□

Lemma: Cluster Non-overlapping. *Clusters* have exclusive vertices, edges or *Hyperblossoms*. $\forall C_1, C_2 \in \mathcal{C}, C_1 \neq C_2 \rightarrow V_{C_1} \cap V_{C_2} = \varnothing, E_{C_1} \cap E_{C_2} = \varnothing, \mathcal{B}_{C_1} \cap \mathcal{B}_{C_2} = \varnothing$.

Proof. By definition of the *Cluster*, every *Tight Edge* or *Hyperblossom* merges two *Clusters*, thus $E_{C_1} \cap E_{C_2} = \varnothing$ and $\mathcal{B}_{C_1} \cap \mathcal{B}_{C_2} = \varnothing$ when $C_1 \neq C_2$. A vertex is merged into a *Cluster* only when a *Tight Edge* or *Hyperblossom* touches it. Therefore, we have $V_{C_1} \cap V_{C_2} = \varnothing$. □

Lemma: Cluster of Defect Vertex. Each defect vertex $v \in D$ belongs to a unique *Cluster* $C = \mathcal{C}(v), v \in V_C$.

Proof. In the beginning of the algorithm, a defect vertex $v \in D$ belongs to a *Cluster* according to the definition of *Cluster*. Given *Lemma: Cluster Non-overlapping*, we have $V_{C_1} \cap V_{C_2} = \varnothing$ when $C_1 \neq C_2$. Thus, v belongs to a unique *Cluster*, denoted as $C = \mathcal{C}(v)$ where $v \in V_C$. □

Lemma: Cluster of Hyperblossom. Each *Hyperblossom* $S \in \mathcal{B}$ belongs to a unique *Cluster* $C = \mathcal{C}(S), S \in \mathcal{B}_C$.

Proof. According to the definition of *Cluster*, each *Hyperblossom* $S \in \mathcal{B}$ merges all the *Clusters* that it touches, i.e., $V_S \subseteq V_C$. It touches at least one *Cluster* because S is an *Invalid* subgraph with at least one defect vertex, i.e., $D \cap V_S \neq \varnothing$. Given *Lemma: Cluster of Defect Vertex*, such a defect vertex $v \in D \cap V_S$ belongs to a unique *Cluster* $C = \mathcal{C}(v)$. Thus, the *Hyperblossom* S belongs to a unique *Cluster*, denoted as $C = \mathcal{C}(S)$. Also, $S \in \mathcal{B}_C$ according to the definition of the *Hyperblossoms* of a *Cluster*. □

Lemma: Subgraph $\mathcal{O}' \subseteq \mathcal{O}$. Given a decoding hypergraph $G = (V, E)$ and a subgraph $G' = (V', E')$ with $V' \subseteq V$, $E' \subseteq E[V']$, the set of *Invalid* subgraphs $\mathcal{O}'$ of G' is a subset of the set of *Invalid* subgraphs $\mathcal{O}$ of G.

Proof. We prove $\mathcal{O}' \subseteq \mathcal{O}$ by proving $\forall S' \in \mathcal{O}', S' \in \mathcal{O}$.

According to the definition of *Invalid* subgraphs, every $S' = (V_{S'}, E_{S'}) \in \mathcal{O}'$ satisfies the following condition:

$$\forall \mathcal{E} \subseteq E_{S'}, \mathcal{D}(\mathcal{E}) \neq D \cap V_{S'}$$

Given $V_{S'} \subseteq V' \subseteq V$ and $E_{S'} \subseteq E'[V_{S'}] \subseteq E[V_{S'}]$, S' satisfies the definition of *Invalid* subgraphs for the original decoding hypergraph $G = (V, E)$, i.e., $S' \in \mathcal{O}$.

□

Lemma: Cluster Parity Factor. The union of all the local parity factors $\mathcal{E} = \bigcup_{C \in \mathcal{C}} \mathcal{E}_C$ is a global parity factor, i.e., $\mathcal{D}(\mathcal{E}) = D$.

Proof. According to the definition of *Locally Optimal Cluster*, $\mathcal{D}(\mathcal{E}_C) = D \cap V_C$ where $\mathcal{E}_C \subseteq E_C$. Also, given *Lemma: Cluster Non-overlapping*, $\mathcal{E}_C \subseteq E_C$ are disjoint, so we have $\bigcup_{C \in \mathcal{C}} \mathcal{E}_C = \bigoplus_{C \in \mathcal{C}} \mathcal{E}_C$ where $\bigoplus$ is the symmetric difference operator. Putting together, we have

$$\begin{aligned}
\mathcal{D}(\mathcal{E}) &= \mathcal{D}(\bigcup_{C \in \mathcal{C}} \mathcal{E}_C) \\
&\downarrow \text{(by } \textit{Lemma: Cluster Non-overlapping}\text{)} \\
&= \mathcal{D}(\bigoplus_{C \in \mathcal{C}} \mathcal{E}_C) \\
&= \bigoplus_{C \in \mathcal{C}} \mathcal{D}(\mathcal{E}_C) \\
&\downarrow \text{(by definition of } \textit{Locally Optimal Cluster}\text{)} \\
&= \bigoplus_{C \in \mathcal{C}} (D \cap V_C) \\
&\downarrow \text{(by } \textit{Lemma: Cluster Non-overlapping}\text{)} \\
&= \bigcup_{C \in \mathcal{C}} (D \cap V_C) \\
&= D \cap \Big(\bigcup_{C \in \mathcal{C}} V_C \Big) \\
&\downarrow \text{(by } \textit{Lemma: Cluster of Defect Vertex}\text{)} \\
&= D
\end{aligned}$$

□

Lemma: Cluster DLP Solution. The union of all the local DLP solutions $\vec{y} = \bigcup_{C \in \mathcal{C}} \{S : y_S | \forall S \in \mathcal{B}_C\}$ is a feasible DLP solution, i.e., $\vec{y}$ satisfies all the DLP constraints.

Proof. We prove that $\vec{y}$ satisfies Eq. (4a) and Eq. (4b).

For Eq. (4a), we need to prove that $y_S \geq 0, \forall S \in \mathcal{O}$. This is trivial because $y_S \geq 0, \forall S \in \mathcal{B}_C$ for every *Cluster* $C \in \mathcal{C}$, and the rest of the dual variables are $y_S = 0, \forall S \in \mathcal{O} \setminus \cup_{C \in \mathcal{C}} \mathcal{B}_C$.

For Eq. (4b), we need to prove that $\sum_{S \in \mathcal{O}} y_S \leq w_e$ for every hyperedge $e \in E$. For each hyperedge $e \in E$, we consider two cases: whether it belongs to E_C of some *Cluster* or not.

If $\exists C \in \mathcal{C}, e \in E_C$, then according to *Lemma: Cluster Non-overlapping*, none of the *Hyperblossoms* from other *Clusters* contribute to e. This is because $\forall C' \in \mathcal{C} \setminus \{C\}, \forall S \in \mathcal{B}_{C'}, V_S \cap e = \varnothing$ and thus $e \notin E(V_S) \supseteq E(V_S) \setminus E_S = \delta(S)$. Given that the local DLP solution is feasible and no other dual variables contribute to e, we have $\sum_{S \in \mathcal{O}} y_S \leq w_e$.

Otherwise, if e does not belong to any *Cluster*, then it must not be a *Tight Edge*, according to the definition of *Cluster*. In other words, $\sum_{S \in \mathcal{O}} y_S < w_e$.

In both cases, $\vec{y}$ satisfies Eq. (4b) for every hyperedge $e \in E$.

□

4. HyperBlossom Algorithm Optimality

Theorem: Relaxing with Clusters. Given a suboptimal DLP solution $\vec{y}$, there exists a *Relaxer* or an *Invalid Cluster*, or both.

Proof. This theorem is a minor extension of **Theorem: Relaxing**. The only additional step required in the proof is to show that if there exists a *Trivial Direction* $\Delta\vec{y} = \{\Delta y_S : +1\}$ for some *Invalid* subgraph $S \in \mathcal{O}$, then there must also exist an *Invalid Cluster* $\mathcal{C} \cap \mathcal{O} \neq \varnothing$. To do so, we first construct an *Invalid* subgraph S_3 that is the union of some *Clusters* $\mathcal{C}' \subseteq \mathcal{C}$, and then prove that at least one of the *Clusters* in $\mathcal{C}'$ must be *Invalid*.

Given that the *Trivial Direction* is a *Feasible Direction*, S must not grow on any *Tight Edge*, i.e., $\delta(S) \cap T = \varnothing$. We can construct another *Invalid* subgraph $S_1 = (V_S, E_S \cap T)$ where $\Delta\vec{y'} = \{\Delta y_{S_1} : +1\}$ is also a *Trivial Direction*. To see why S_1 is an *Invalid* subgraph, we use the definition of *Invalid* subgraph:

$$\begin{aligned}&\forall \mathcal{E} \subseteq E_S, \mathcal{D}(\mathcal{E}) \neq V_S \cap D\\ \Longrightarrow &\forall \mathcal{E} \subseteq E_S \cap T \subseteq E_S, \mathcal{D}(\mathcal{E}) \neq V_S \cap D\end{aligned}$$

To see why $\Delta\vec{y'}$ is a *Feasible Direction*, we show that it does not grow on any *Tight Edge*:

$$\begin{aligned}\delta(S_1) \cap T &= \Big(E(V_{S_1}) \setminus E_{S_1}\Big) \cap T\\ &= \Big(E(V_S) \setminus (E_S \cap T)\Big) \cap T\\ &\downarrow \text{(given } A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C))\\ &= \Big((E(V_S) \setminus E_S) \cup (E(V_S) \setminus T)\Big) \cap T\\ &= \Big((E(V_S) \setminus E_S) \cap T\Big) \cup \Big((E(V_S) \setminus T) \cap T\Big)\\ &\downarrow \text{(given } \delta(S) \cap T = (E(V_S) \setminus E_S) \cap T = \varnothing)\\ &= (E(V_S) \setminus T) \cap T\\ &= \varnothing\end{aligned}$$

We can then remove the isolated non-defect vertices from V_{S_1} to get another *Invalid* subgraph

$$S_2 = (\{v \in V_S | v \in D \vee v \notin \cup_{e \in E_{S_1}} e\}, E_{S_1})$$

To see why S_2 is an *Invalid* subgraph, we use the definition of *Invalid* subgraph:

$$\begin{aligned}&\forall \mathcal{E} \subseteq E_{S_1}, \mathcal{D}(\mathcal{E}) \neq V_S \cap D\\ &\downarrow \text{(given } V_S \setminus V_{S_2} \subseteq \overline{D} \text{ is not incident to any } e \in E_{S_1})\\ \Longrightarrow &\forall \mathcal{E} \subseteq E_{S_1}, \mathcal{D}(\mathcal{E}) \neq V_{S_2} \cap D\end{aligned}$$

We further merge S_2 with any *Hyperblossom* $S \in \mathcal{B}$ that touches S_2. In other words, when $V_S \cap V_{S_2} \neq \varnothing$, we merge S_2 and the *Cluster* of the *Hyperblossom* $\mathcal{C}(S)$ (*Lemma: Cluster of Hyperblossom*). This results in a new *Invalid* subgraph S_3.

Now we have constructed an *Invalid* subgraph S_3 that satisfy the properties of the *Cluster*: E_{S_3} consists of *Tight Edges* and V_{S_3} includes defect vertices and non-defect vertices connected by an edge in E_{S_3}. Note that this doesn't mean S_3 is a *Cluster*, rather, it can be the union of multiple *Clusters* $\mathcal{C}' \subseteq \mathcal{C}$. That is,

$$\begin{aligned}V_{S_3} &= \cup_{C \in \mathcal{C}'} V_C\\ E_{S_3} &= \cup_{C \in \mathcal{C}'} E_C\end{aligned}$$

Now we prove that there must exists an *Invalid Cluster* within $\mathcal{C}'$, i.e., $\mathcal{C}' \cap \mathcal{O} \neq \varnothing$. We prove it by contradiction. Suppose all the *Clusters* in $\mathcal{C}'$ are *Valid*, we have $\forall C \in \mathcal{C}', \exists \mathcal{E}_C \subseteq E_C, \mathcal{D}(\mathcal{E}_C) = D \cap V_C$. According to *Lemma: Cluster Parity Factor*, the union of these local parity factors is a global parity factor, i.e., $\mathcal{D}(\bigcup_{C \in \mathcal{C}'} \mathcal{E}_C) = D \cap V_{S_3}$. This contradicts with the fact that S_3 is an *Invalid* subgraph, because there exists a parity factor $\mathcal{E} = \bigcup_{C \in \mathcal{C}'} \mathcal{E}_C \subseteq E_{S_3}, \mathcal{D}(\mathcal{E}) = D \cap V_{S_3}$. Thus, there must exists an *Invalid Cluster* with $\mathcal{C}'$.

Together, we prove that there exists an *Invalid Cluster* $C \in \mathcal{C}' \cap \mathcal{O}$ when there exists a *Trivial Direction*.

□

Theorem: HyperBlossom Algorithm Optimality. There exists a *Relaxer* finder so that the HyperBlossom algorithm equipped with it would find optimal MWPF and DLP solutions, if the decoding hypergraph and all its subgraphs satisfy $\mathbf{min\,LP = min\,ILP}$.

Proof. According to the **Theorem: Optimality Criteria of Clusters**, once all *Clusters* are *Locally Optimal Clusters*, the HyperBlossom algorithm finds the optimal MWPF and DLP solutions. Thus, we only need to prove that the HyperBlossom algorithm terminates only when all *Clusters* are *Locally Optimal Clusters*.

We first prove that when the *Cluster* is *Invalid* or suboptimal, i.e., $\sum_{S \in \mathcal{B}_C} y_S < W(\mathcal{E}_C)$ where $\mathcal{E}_C$ is the MWPF of (V_C, E_C), the HyperBlossom algorithm finds a *Useful Direction* such that the local dual objective $\sum_{S \in \mathcal{B}_C} y_S$ grows by a positive length $l > 0$. We then prove that the HyperBlossom algorithm terminates within a finite number of iterations.

When the *Cluster* is *Invalid*, we prove that the HyperBlossom algorithm finds a *Useful Direction*. At line 7 of Algorithm 8, it checks whether $(V_C, E'_C) \in \mathcal{O}$. Given that $E'_C \subseteq E_C$ and that C is an *Invalid* subgraph, we have

$$\begin{aligned}&\forall \mathcal{E} \subseteq E_C, \mathcal{D}(\mathcal{E}) \neq D \cap V_C\\ \Longrightarrow &\forall \mathcal{E} \subseteq E'_C \subseteq E_C, \mathcal{D}(\mathcal{E}) \neq D \cap V_C\end{aligned}$$

Thus, the Primal phase finds a *Trivial Direction* $\Delta\vec{y}[E'_C]$ under the assumption of *Tight Edges* E'_C (line 9). A *Trivial Direction*, by definition, have $\sum \Delta\vec{y} > 0$. According to **Theorem: Batch Relaxing**, the composed *Direction* at line 10 must be a *Feasible Direction* with $\sum \Delta'\vec{y} \geq \sum \Delta\vec{y} > 0$. Thus, the HyperBlossom algorithm finds a *Useful Direction* $\Delta'\vec{y}$.

When the *Cluster* is *Valid* but suboptimal, we prove that the HyperBlossom algorithm also finds a *Useful Direction*. Given the inequality chain Eq. (5) and $\mathbf{min\,LP = min\,ILP}$ for any subgraph of G, there exists an optimal DLP solution $\vec{y^o}$ of the hypergraph $C = (V_C, E_C)$ that has $\sum_{S \in \mathcal{O}_C} y^o_S = W(\mathcal{E}_C)$ where $\mathcal{O}_C$ is the set of *Invalid* subgraphs of the subgraph C. Note that the DLP constraints are defined on $\mathcal{O}_C$ of the subgraph $C = (V_C, E_C)$ and thus $\vec{y^o}$ is not necessarily a feasible DLP solution for the original graph G. Nonetheless,

given *Lemma: Subgraph* $\mathcal{O}' \subseteq \mathcal{O}$, the dual variables still exist in the original problem $\mathcal{O}_C \subseteq \mathcal{O}$ and $\vec{y^o}$ is a (*Feasible* or not) DLP solution of the original graph G. To prove that $\Delta\vec{y} = \vec{y^o} - \vec{y}$ is a *Useful Direction*, we only need to prove $\sum \Delta\vec{y} > 0$ and that it satisfies the conditions of *Feasible Direction* Eqs. (6a) and (6b).

First, to prove $\sum \Delta\vec{y} > 0$, we have

$$\sum \Delta\vec{y} = \sum \vec{y^o} - \sum \vec{y} = W(\mathcal{E}_C) - \sum \vec{y} > 0$$

Second, we prove that $\Delta\vec{y}$ satisfies Eq. (6a): $\forall S \in \mathcal{O} \setminus \mathcal{B}, \Delta y_S \geq 0$. Both $\vec{y^o}$ and $\vec{y}$ only have non-zero values within $\mathcal{O}_C$. Thus, the *Hyperblossoms* $\mathcal{B} \subseteq \mathcal{O}_C$. For those $S \in \mathcal{O} \setminus \mathcal{O}_C$, we have $\Delta y_S = y_S^o - y_S = 0$. For those $S \in \mathcal{O}_C \setminus \mathcal{B}$, we have $y_S = 0$ and thus $\Delta y_S = y_S^o - y_S = y_S^o \geq 0$ given $\vec{y^o}$ is a feasible DLP solution of the subgraph $C = (V_C, E_C)$. Thus, $\Delta\vec{y}$ satisfies Eq. (6a).

Third, we prove that $\Delta\vec{y}$ satisfies Eq. (6b): $\forall e \in T, \sum_{S \in \mathcal{O} | e \in \delta(S)} \Delta y_S \leq 0$. For any *Tight Edge* $e \in E_C$, by definition we have $\sum_{S \in \mathcal{O} | e \in \delta(S)} y_S = w_e$. Given that $\vec{y^o}$ is a feasible DLP solution of the subgraph $C = (V_C, E_C)$, $\sum_{S \in \mathcal{O} | e \in \delta(S)} y_S \leq w_e$. Together, we have

$$\sum_{S \in \mathcal{O} | e \in \delta(S)} \Delta y_S = \sum_{S \in \mathcal{O} | e \in \delta(S)} y_S^o - w_e \leq 0$$

For those *Tight Edges* $e \in T \setminus E_C$, neither $\vec{y^o}$ nor $\vec{y}$ contribute to the DLP constraints of e because $\forall S \in \mathcal{O}_C, \delta(S) \cap T \subseteq E(V_C) \cap T = E_C$. In this case, we have $\sum_{S \in \mathcal{O} | e \in \delta(S)} \Delta y_S = 0$. Thus, in both cases, we have $\Delta\vec{y}$ satisfies Eq. (6b).

We now have finished the proof that when the *Cluster* is *Invalid* or suboptimal, the HyperBlossom algorithm finds a *Useful Direction*.

To prove the termination of the HyperBlossom algorithm, we prove that $\mathcal{B}_C^H$ is monotonically increasing for each iteration and $|\mathcal{B}_C^H| \leq |\mathcal{O}|$. Each *Cluster* maintains all the previously known *Hyperblossoms* as $\mathcal{B}_C^H$ (line 9 of Algorithm 7). Given that the HyperBlossom algorithm optimizes the partial DLP solution (line 10 of Algorithm 7), any further progression of the dual objective must introduce new *Hyperblossom* $S \in \mathcal{O} \setminus \mathcal{B}_C^H$. Thus, $|\mathcal{B}_C^H|$ is monotonically increasing for each iteration. Given that $|\mathcal{B}_C^H| \leq |\mathcal{O}|$, the number of iterations is at most $|\mathcal{O}|$, finite but exponentially large.

We have proved that the HyperBlossom algorithm terminates within a finite number of iterations, and it will not terminate until all the *Clusters* become *Locally Optimal Clusters*. Thus, the HyperBlossom algorithm is optimal under the condition that the decoding hypergraph and all its subgraphs satisfy $\mathbf{min\,LP = min\,ILP}$. □

5. Why E_S is Necessary in the Definition of Invalid Subgraphs $S = (V_S, E_S)$

In the *Dual Mapping* for simple graphs (§C), we fix $E_S = E[V_S]$ for all the mapped *Invalid* subgraphs, yet still achieves $\mathbf{min\,LP = min\,ILP}$. It raises the question of why not eliminating E_S from the definition of *Invalid* subgraphs and only use V_S to define *Invalid* subgraphs while fixing $E_S = E[V_S]$? Although this works for simple graphs, we show that E_S is necessary to ensure $\mathbf{min\,LP = min\,ILP}$ for even the simplest classes of nullity$_{=0}$ hypergraphs. For general hypergraphs, adding E_S helps reducing the gap between the minimum LP and ILP objective values.

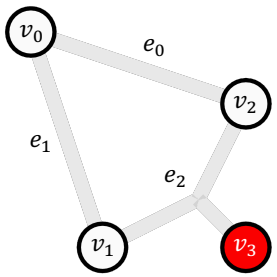


FIG. 20: A nullity$_{=0}$ hypergraph. All the edge weights are 1. The syndrome is $D = \{v_3\}$.

For the hypergraph in Fig. 20, which is a nullity$_{=0}$ hypergraph, we can prove that if we do not use E_S when we define *Invalid* subgraphs, i.e., fixing $E_S = E[V_S]$, then $\min$ LP $<$ $\min$ ILP.

For this small hypergraph, we can manually list all the *Invalid* subgraphs. An *Invalid* subgraph must contain at least one defect vertex, so every *Invalid* subgraph must contain v_3. Thus, we have 7 *Invalid* subgraphs assuming $E_{S_i} = E[V_{S_i}]$:

- $V_{S_1} = \{v_3\}, \delta(S_1) = \{e_2\}$
- $V_{S_2} = \{v_3, v_0\}, \delta(S_2) = \{e_0, e_1, e_2\}$
- $V_{S_3} = \{v_3, v_1\}, \delta(S_3) = \{e_1, e_2\}$
- $V_{S_4} = \{v_3, v_2\}, \delta(S_4) = \{e_0, e_2\}$
- $V_{S_5} = \{v_3, v_0, v_1\}, \delta(S_5) = \{e_0, e_2\}$
- $V_{S_6} = \{v_3, v_0, v_2\}, \delta(S_6) = \{e_1, e_2\}$
- $V_{S_7} = \{v_3, v_1, v_2\}, \delta(S_7) = \{e_0, e_1\}$

Since the only parity factor solution is $\mathcal{E} = \{e_0, e_1, e_2\}$ with $W(\mathcal{E}) = 3$, one can easily verify that there is no feasible modified-DLP solution so that $\sum_{S \in \mathcal{O}} y_S = 3$. Indeed, the optimal modified-DLP solution is $y_{S_1} = y_{S_7} = 1$ with $\sum_{S \in \mathcal{O}} y_S = 2$. Thus, removing E_S from the definition of *Invalid* subgraphs leads to a gap between the minimum LP and ILP objective values.

We can easily see that adding E_S back will remove the gap between the minimum LP and ILP objective values, given an optimal DLP solution:

- $S_1 = (V, E \setminus \{e_0\}), \delta(S_1) = \{e_0\}, y_{S_1} = 1$

- $S_2 = (V, E \setminus \{e_1\}), \delta(S_2) = \{e_1\}, y_{S_2} = 1$

- $S_3 = (V, E \setminus \{e_2\}), \delta(S_3) = \{e_2\}, y_{S_3} = 1$

Appendix B: Theoretical Analysis of *SingleHair*

We theoretically analyze the *SingleHair Relaxer*-finding algorithm (§IV A) with two parts. The first part proves two lemmas about why the *SingleHair* works (§B 1) and when it stops finding *Relaxers* (§B 2). The second part analyzes the limitations of the *SingleHair* on hypergraphs (§B 3) and even simple graphs (§B 4). We also prove its optimality on nullity$_{\leq 1}$ hypergraphs (§B 5).

1. Hair Matrix Odd Row Existence

Lemma: Hair Matrix Odd Row Existence. In a *Hair Matrix*, a row of which the right most value is 1 is called an *Odd* row. An Odd row always exists in a *Hair Matrix*.

Proof. We prove the lemma in two cases: the *Hyperblossom Matrix* $\mathcal{M}_S$ has a feasible solution or not.

First, if $\mathcal{M}_S$ does not have any feasible solution, the last row of the *Hyperblossom Matrix* $\mathcal{M}_S$ must have 1 on the last column and 0 on all other columns because $\mathcal{M}_S$ is in the reduced row echelon form. Since this row does not have a pivot, it must be part of the *Hair Matrix* $\mathcal{H}_S$. Thus, the last row of the *Hair Matrix* must be an Odd row given that the last column of this row is 1.

Otherwise, $\mathcal{M}_S$ has at least one feasible parity factor solution. We prove the lemma by contradiction. That is, if all the constraints in the *Hair Matrix* $\mathcal{H}_S$ are even parity constraints, then we prove that it contradicts with the fact that S is an *Invalid* subgraph. Since all the constraints in $\mathcal{H}_S$ are even parity constraints, $x_e = 0, \forall e \in E_C \cap \delta(S)$ corresponds to one feasible solution of $\mathcal{M}_S$. That is, there exists a feasible solution $\mathcal{E} \subseteq E_C \setminus \delta(S)$ that satisfies $\mathcal{D}(\mathcal{E}) = D \cap V_C$ but not using any of the edges in $\delta(S)$, i.e., $\mathcal{E} \cap \delta(S) = \varnothing$. We now prove that there exists $\mathcal{E}_S \subseteq E_S$ so that $\mathcal{D}(\mathcal{E}_S) = D \cap V_S$, i.e., it violates the assumption that S is an *Invalid* subgraph.

We explicitly construct $\mathcal{E}_S = \mathcal{E} \cap E_S$. We then use the definition of *Cluster* and *Invalid* subgraph to prove that $\mathcal{D}(\mathcal{E}_S) = D \cap V_S$.

$$
\begin{aligned}
\mathcal{D}(\mathcal{E}_S) &= \mathcal{D}(\mathcal{E} \cap E_S) \\
&\quad \downarrow \text{(by definition of } \textit{Defects of Error Pattern}) \\
&= \left\{ v \in V \,\middle|\, |\{e \in \mathcal{E} \cap E_S | v \in e\}| = 1 \bmod 2 \right\} \\
&\quad \downarrow \text{(given } \forall e \in E_S \subseteq E[V_S], e \subseteq V_S) \\
&= \left\{ v \in V_S \,\middle|\, |\{e \in \mathcal{E} \cap E_S | v \in e\}| = 1 \bmod 2 \right\} \\
&\quad \downarrow \text{(given } E(V_S) = E_S \cup \delta(S) \text{ and } \mathcal{E} \cap \delta(S) = \varnothing) \\
&= \left\{ v \in V_S \,\middle|\, |\{e \in \mathcal{E} \cap E(V_S) | v \in e\}| = 1 \bmod 2 \right\} \\
&\quad \downarrow \text{(given } E(V_S) \text{ includes all edges incident to } V_S) \\
&= \left\{ v \in V_S \,\middle|\, |\{e \in \mathcal{E} | v \in e\}| = 1 \bmod 2 \right\} \\
&= \mathcal{D}(\mathcal{E}) \cap V_S \\
&= (D \cap V_C) \cap V_S \\
&\quad \downarrow \text{(given } V_S \subseteq V_C \text{ according to } \textit{Cluster}) \\
&= D \cap V_S
\end{aligned}
$$

Clearly, we have $\mathcal{E}_S = \mathcal{E} \cap E_S \subseteq E_S$. Thus, there exists $\mathcal{E}_S \subseteq E_S$ with $\mathcal{D}(\mathcal{E}_S) = D \cap V_S$, which contradicts with the fact that S is an *Invalid* subgraph. Therefore, $\mathcal{H}_S$ must contains at least one Odd row.

In both cases, we have proved that the *Hair Matrix* $\mathcal{H}_S$ must contains at least one Odd row. □

2. Hair Matrix Stops at Unique Row

From the *SingleHair Relaxer*-finding algorithm (§IV A), we show that whenever $E^- \neq \varnothing$ for any *Hyperblossom* $S \in \mathcal{B}$, *SingleHair* will find a *Relaxer*. We prove the following lemma that shows the *SingleHair Relaxer* finder stops finding a *Relaxer* only when all the *Hair Matrices* consists of a single row. Together with *Lemma: Hair Matrix Odd Row Existence*, the only row must be an Odd row. Furthermore, the Odd row must consists of all 1s because an element of 0 means the corresponding hyperedge is totally unconstrained, which is impossible since a hyperedge connects to at least one vertex.

Lemma: Unique Row if Not a Relaxer. When $E^- = \varnothing$, i.e., there is no *Relaxer* corresponding to an Odd row, then there is only one row.

Proof. We use the property of the reduced row echelon form to prove the lemma.

First of all, *SingleHair* only constructs *Hair Matrices* when the *Cluster* is *Valid*. Thus, there is no Odd row that has a 1 on the last column but 0 on all other columns. Such a column causes the *Cluster* to be *Invalid* because no parity factor solution will ever satisfy the odd parity constraint. Since a row of 0s on all the columns is not considered a row in a reduced row echelon form, all the

rows in the *Hair Matrix* contains at least one 1 for the variable columns.

We then prove that the *Hair Matrix* must have a single row. We use contradiction to prove the lemma, i.e., assuming there are multiple rows and then prove $E^- \neq \varnothing$, which contradicts with the assumption that $E^- = \varnothing$.

In the reduced row echelon form, each variable column must be either a pivot column or a free column. Each pivot column corresponds to a unique row and each row corresponds to a unique pivot column. If there exists more than one row in the *Hair Matrix*, then there must be more than one pivot column. Suppose the pivot columns are $E_p \subseteq T \cap \delta(S)$ where $|E_p| \geq 2$.

Given *Lemma: Hair Matrix Odd Row Existence*, there exists an Odd row in the *Hair Matrix* $\mathcal{H}_S$. Suppose the pivot column of this Odd row is $e_p \in E_p$.

We then prove that $E^- \neq \varnothing$. A pivot column must be 1 on its corresponding row and 0 on all other rows. Thus, $\forall e \in E_p \setminus \{e_p\}$, the column corresponding to e must be 0 on the Odd row. According to the definition of *Hair Matrix*, the columns of $\mathcal{H}_S$ are $\delta(S) \cap T$. Given the definition of $E^- = \delta(S) \cap T \setminus E^+$ where E^+ are the columns whose value on the Odd row is 1, we can see that E^- are the columns of the *Hair Matrix* $\mathcal{H}_S$ whose value on the Odd row is 0. Thus, we have $E_p \setminus \{e_p\} \subseteq E^-$. Given $|E_p| \geq 2$, we have $|E^-| \geq |E_p| - 1 \geq 1$, i.e.,$E^- \neq \varnothing$.

This contradicts with the assumption that $E^- = \varnothing$. Therefore, a *Hair Matrix* must consist of a single row when $E^- = \varnothing$.

□

3. *SingleHair* is Suboptimal on Hypergraphs

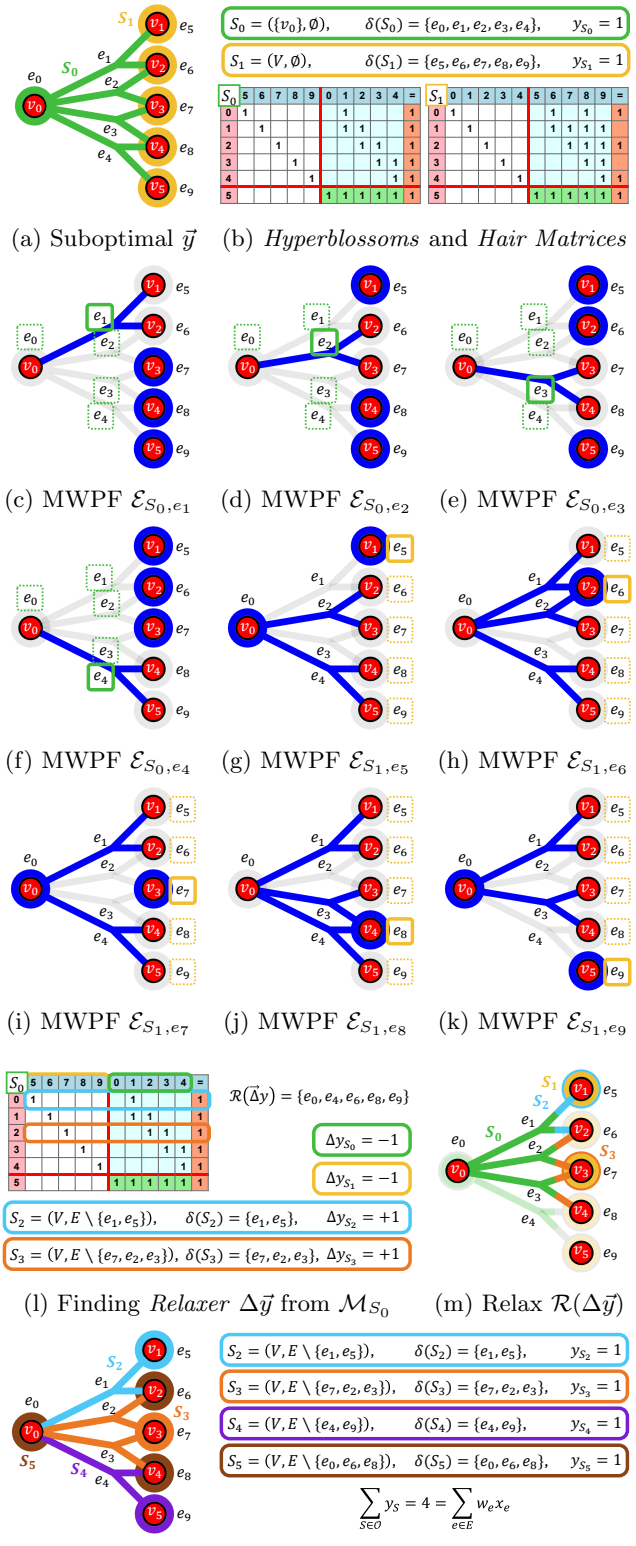
We construct a failing case of the *SingleHair Relaxer*-finding algorithm by exploiting its limitation: the inability to consider dependency between *Hyperblossoms*. As shown in Fig. 21, the *SingleHair Relaxer* finder constructs the *Hair Matrices* $\mathcal{H}_{S_0}$ and $\mathcal{H}_{S_1}$ in Fig. 21b, both consisting of a single Odd row. Thus, *SingleHair* returns NIL with this suboptimal DLP solution.

However, the MWPF solutions only satisfy the following complementary slackness theorem for one of the two *Hyperblossoms* at a time, but never both.

$$y_S > 0 \implies \sum_{e \in \delta(S)} x_e = 1, \qquad \forall S \in \mathcal{O} \qquad \text{(C2)}$$

In particular, Figs. 21c to 21f satisfy Eq. (C2) for S_0 and Figs. 21g to 21k satisfy Eq. (C2) for S_1.

Although the *SingleHair Relaxer* finder cannot find more *Relaxers*, we can manually find a *Relaxer* to make progress in the HYPERBLOSSOM algorithm. As shown in Fig. 21l, $\Delta\vec{y}$ is a *Feasible Direction* that involves four *Hyperblossoms* (two shrinking and two growing). It is a *Relaxer* because $\Delta\vec{y}$ relaxes edges $\mathcal{R}(\Delta\vec{y}) = \{e_0, e_4, e_6, e_8, e_9\}$, as shown in Fig. 21m. After the relaxation, the *Cluster* becomes *Invalid* $C = (V, E \setminus \mathcal{R}(\Delta\vec{y})) \in$



(a) Suboptimal $\vec{y}$ (b) *Hyperblossoms* and *Hair Matrices*

(c) MWPF $\mathcal{E}_{S_0,e_1}$ (d) MWPF $\mathcal{E}_{S_0,e_2}$ (e) MWPF $\mathcal{E}_{S_0,e_3}$

(f) MWPF $\mathcal{E}_{S_0,e_4}$ (g) MWPF $\mathcal{E}_{S_1,e_5}$ (h) MWPF $\mathcal{E}_{S_1,e_6}$

(i) MWPF $\mathcal{E}_{S_1,e_7}$ (j) MWPF $\mathcal{E}_{S_1,e_8}$ (k) MWPF $\mathcal{E}_{S_1,e_9}$

(l) Finding *Relaxer* $\Delta\vec{y}$ from $\mathcal{M}_{S_0}$ (m) Relax $\mathcal{R}(\Delta\vec{y})$

(n) Optimal DLP $\vec{y}$ (o) MWPF Optimality Certificate

FIG. 21: An example of the *SingleHair Relaxer*-finding algorithm fails to find any *Relaxer* given a suboptimal DLP solution. The decoding hypergraph has a uniform weight of $w_e = 1$, with two *Hyperblossoms* S_0 and S_1 of $y_{S_0} = y_{S_1} = 1$.

$\mathcal{O}$, so that the HyperBlossom algorithm can compose a *Useful Direction* to grow. The final DLP solution $\vec{y}$ has a weight of 4, equal to the weight of all the MWPF solutions Figs. 21c to 21k. In fact, all these parity factors satisfy Eq. (C2) for the *Hyperblossoms* $\mathcal{B} = \{S_2, S_3, S_4, S_5\}$ at the same time. Thus, we prove the optimality of both the DLP solution $\vec{y}$ and the MWPF solutions.

This example shows that in order to achieve optimality, a *Relaxer* finder may need to explore *Relaxers* that involve more than a pair of *Invalid* subgraphs. Indeed, optimal *Relaxer* finders like *Blossom* (§IV C) and *Nullity*$_{\leq 1}$ (§IV D) find these more complicated *Relaxers* to reach optimality.

4. *SingleHair* is Suboptimal on Simple Graphs

One might expect that the artificially constructed failing case in §B 3 only applies to hypergraphs, but we show in this section that it also applies to simple graphs.

We use numerical simulation to find a suboptimal case of the *SingleHair Relaxer*-finding algorithm on simple graphs, as shown in Fig. 22. Given the initial DLP solution with 10 *Hyperblossoms* (Fig. 22a), the *SingleHair Relaxer*-finding algorithm cannot find a *Relaxer* any more because all *Hair Matrices* consist of a single Odd row, as shown in Figs. 22e to 22n. However, any MWPF solution violates the complementary slackness theorem Eq. (C2) for at least one *Hyperblossom*. For example, $\mathcal{E}$ in Fig. 22b violates Eq. (C2) for S_9 while $\mathcal{E}'$ in Fig. 22c violates Eq. (C2) for S_4. This is because the DLP solution is suboptimal. An optimal solution $\vec{y'}$ in Fig. 22d satisfies Eq. (C2) for all the *Hyperblossoms*.

5. *SingleHair* is Optimal on Nullity$_{\leq 1}$ Hypergraphs

We prove that the *SingleHair Relaxer*-finding algorithm is optimal for the nullity$_{\leq 1}$ hypergraphs, although it is not as efficient as the *Nullity*$_{\leq 1}$ *Relaxer* finder because the latter exploits the special property of nullity$_{\leq 1}$. Before looking into the *SingleHair Relaxer* finder, we first prove the following lemma.

Lemma: Nullity$_{\leq 1}$ *is a hereditary property.* Any subgraph of a nullity$_{\leq 1}$ hypergraph is also nullity$_{\leq 1}$

Proof. Any subgraph is constructed by removing some edges and then removing some vertices that are not incident to any remaining edge.

Removing an edge corresponds to removing a column of the incidence matrix. According to the rank-nullity theorem, the number of columns is equal to the sum of the rank and the nullity. The rank of the incidence matrix decreases by at most 1 when removing a column. Thus, the nullity does not increase.

When removing a vertex that is not incident to any remaining edge, the row corresponding to the vertex is all 0s. Thus, the rank does not change by removing an all-0 row. Also, given that the number of columns does not change, the nullity does not change.

Overall, the nullity of any subgraph is no larger than the nullity of the original hypergraph. Therefore, nullity$_{\leq 1}$ is a hereditary property of hypergraphs.

□

Lemma: SingleHair Optimality on Nullity$_{\leq 1}$ *Hypergraphs.* For a decoding hypergraph whose incidence matrix has a nullity of 0 or 1, the *SingleHair Relaxer*-finding algorithm is optimal.

Proof. We prove the lemma by explicitly proving the optimality of the final parity factor solution using **Theorem: Certifying Optimum**.

Given *Lemma: Unique Row if Not a Relaxer* (§B 2), when the HyperBlossom algorithm with the *SingleHair Relaxer* finder terminates, all the *Hair Matrices* consist of a single Odd row of all 1s.

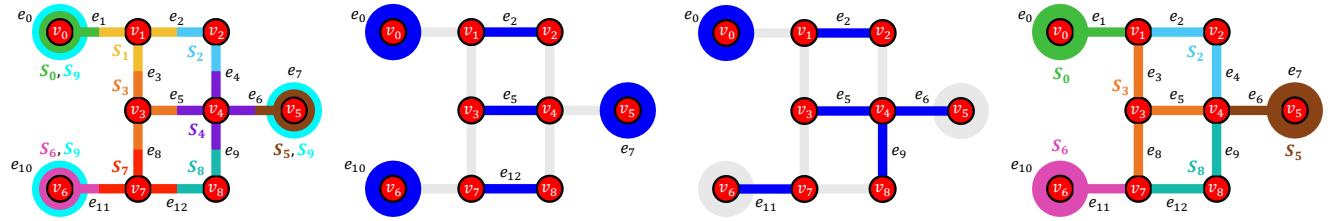
Given that the decoding hypergraph is nullity$_{\leq 1}$ and *Lemma: Nullity*$_{\leq 1}$ *is a hereditary property*, any *Hyperblossom Matrix* $\mathcal{M}_S$ has a nullity of at most 1, i.e., $\mathcal{M}_S$ has at most two parity factors. In the following, we consider two cases: nullity$_{=0}$ and nullity$_{=1}$.

Nullity$_{=0}$. If the parity matrix contains a single parity factor solution $\mathcal{E}$, then we prove the parity factor and DLP solution satisfies $\sum_{S\in\mathcal{O}} y_S = W(\mathcal{E})$. When the nullity is 0, there is no free variable. That is, all the *Hair Matrices* consist of a single (pivot) column $e \in T$ with a fixed choice of $x_e = 1$ constrained by the only Odd row, i.e., $e \in \mathcal{E}$. Thus, every *Hyperblossom* $S \in \mathcal{B}$ contributes to exactly one edge of $\mathcal{E}$ because $\delta(S) \cap T = \{e\}$, the only column of the *Hair Matrix*. We have

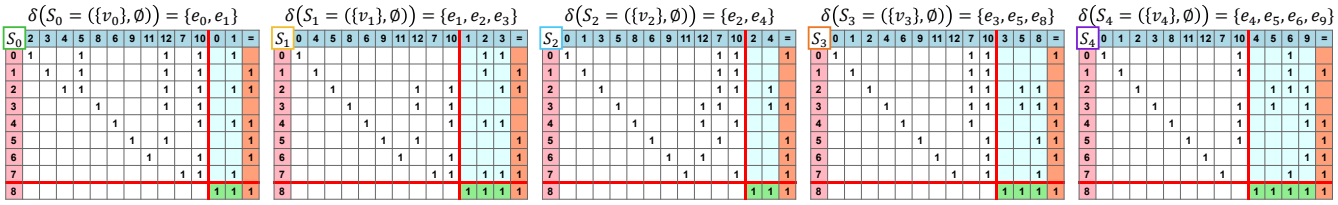
$$
\begin{aligned}
1.\ W(\mathcal{E}) &= \sum_{e\in\mathcal{E}} w_e \\
&\downarrow \text{(given } \mathcal{E} \subseteq T\text{)} \\
&= \sum_{e\in\mathcal{E}} \sum_{S\in\mathcal{B}|e\in\delta(S)} y_S \\
&\downarrow \text{(a } \textit{Hyperblossom} \text{ contributes at most once)} \\
&\leq \sum_{S\in\mathcal{B}} y_S = \sum_{S\in\mathcal{O}} y_S, \\
2.\ W(\mathcal{E}) &\geq \sum_{S\in\mathcal{O}} y_S, \quad \text{(given Eq. (5))} \\
\implies W(\mathcal{E}) &= \sum_{S\in\mathcal{O}} y_S
\end{aligned}
$$

Nullity$_{=1}$. If the parity matrix contains two parity factor solutions $\mathcal{E}_1 \neq \mathcal{E}_2$, then we prove that they are both optimal, i.e., $\sum_{S\in\mathcal{O}} y_S = W(\mathcal{E}_1) = W(\mathcal{E}_2)$.

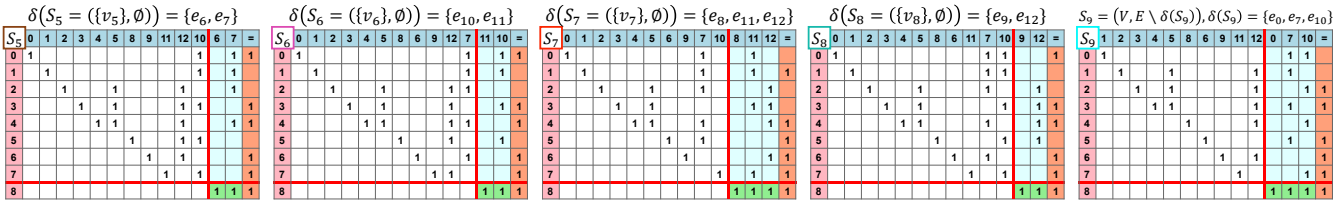
We define *common edges* as the edges that are shared by both parity factors, i.e., $E_{1,2} \coloneqq \mathcal{E}_1 \cap \mathcal{E}_2$. Similarly, the edges exclusive to $\mathcal{E}_1$ and $\mathcal{E}_2$ are $E_1 \coloneqq \mathcal{E}_1 \setminus \mathcal{E}_2$ and $E_2 \coloneqq \mathcal{E}_2 \setminus \mathcal{E}_1$, respectively. We prove that each *Hair Matrix* $\mathcal{H}_S$ contains either a single variable of common edge $e \in E_{1,2}$, or two variables of exclusive edges $e_1 \in E_1, e_2 \in E_2$.



(a) Suboptimal $\sum \vec{y} = 5$. (b) MWPF $W(\mathcal{E}) = 6$. (c) Another MWPF $\mathcal{E}'$. (d) Optimal dual $\sum \vec{y}' = 6$.



(e) *Hyperblossom* S_0. (f) *Hyperblossom* S_1. (g) *Hyperblossom* S_2. (h) *Hyperblossom* S_3. (i) *Hyperblossom* S_4.



(j) *Hyperblossom* S_5. (k) *Hyperblossom* S_6. (l) *Hyperblossom* S_7. (m) *Hyperblossom* S_8. (n) *Hyperblossom* S_9.

FIG. 22: The *SingleHair Relaxer*-finding algorithm fails to find an optimal DLP solution. (a) The decoding graph has a uniform weight of 1. The ten *Hyperblossoms* $S_0, S_1, \ldots, S_9$ all have DLP variable of $y_{S_i} = 1/2$. (b,c) Two MWPF solutions $\mathcal{E}$ and $\mathcal{E}'$ have higher weights than the DLP solution, indicating that the DLP solution is suboptimal. (e-n) The *SingleHair Relaxer*-finding algorithm cannot find any *Relaxers* because the *Hair Matrices* $\mathcal{H}_{S_i}$ all consist of a single Odd row. Thus, the HyperBlossom algorithm with the *SingleHair Relaxer* finder terminates with this suboptimal DLP solution. (d) The optimal DLP solution $\vec{y}'$ has a weight of 6.

When a *Hair Matrix* $\mathcal{H}_S$ of a *Hyperblossom* $S \in \mathcal{B}$ contains a single variable $e \in T$, it is a pivot variable of an Odd row with no free variables, i.e., $x_e = 1$. Thus, e belongs to both parity factors $\mathcal{E}_1$ and $\mathcal{E}_2$, i.e., it is a common edge $e \in E_{1,2}$.

When a *Hair Matrix* $\mathcal{H}_S$ contains two variables $e_1, e_2 \in T$, none of them can be common edges, because otherwise there exists a feasible solution with $x_{e_i} = 0$ violating the definition of a common edge. Also, e_1 and e_2 cannot be exclusive edges of the same feasible parity factor solution, otherwise there exists a third feasible solution $\mathcal{E}_3$ that includes e_1 but not e_2. Thus, e_1 and e_2 must be exclusive edges of different parity factors $e_1 \in \mathcal{E}_1$ and $e_2 \in \mathcal{E}_2$.

A *Hair Matrix* cannot have more than two variables because it has only one single row and the nullity (the number of free variables) is no more than 1.

Thus, each *Hyperblossom* either contributes uniquely to a common edge, or contributes to two exclusive edges simultaneously.

$$
\begin{aligned}
1.\ W(\mathcal{E}_i) &= \sum_{e \in \mathcal{E}_i} w_e \\
&\downarrow \text{(given } \mathcal{E}_i \subseteq T\text{)} \\
&= \sum_{e \in \mathcal{E}_i} \sum_{S \in \mathcal{B} | e \in \delta(S)} y_S \\
&\downarrow \text{(given } \mathcal{E}_i = E_{1,2} \cup E_i\text{)} \\
&= \sum_{e \in E_{1,2} \cup E_i} \sum_{S \in \mathcal{B} | e \in \delta(S)} y_S \\
&\downarrow \text{(a } \textit{Hyperblossom} \text{ contributes at most once)} \\
&\le \sum_{S \in \mathcal{B}} y_S = \sum_{S \in \mathcal{O}} y_S \quad \forall i \in \{1, 2\}, \\
2.\ W(\mathcal{E}_i) &\ge \sum_{S \in \mathcal{O}} y_S \quad \forall i \in \{1, 2\}, \quad \text{(given Eq. (5))} \\
\implies W(\mathcal{E}_i) &= \sum_{S \in \mathcal{O}} y_S \quad \forall i \in \{1, 2\}
\end{aligned}
$$

□

Appendix C: Interoperability with MWPM Decoder

Given the similarity between the math behind MWPF (HyperBlossom framework in §III B) and MWPM (blossom algorithm in §II C), one would wonder whether they can interoperate with each other. This is particularly useful for heterogeneous QEC architectures [24] where we could use a faster MWPM decoder on some portion of the decoding hypergraph and transfer the optimal *Blossom DLP solution* to the HyperBlossom algorithm.

In this section, we first extend the definitions of the blossom algorithm in [4] so that we can use them to define the mapping (§C 1). We then define the bijective function f that maps a *Blossom DLP solution* $\vec{y^*} \in \mathbb{R}^{|\mathcal{O}^*|}$ to a DLP solution $\vec{y} \in \mathbb{R}^{|\mathcal{O}|}$ (§C 2). We prove some important properties of the mapping function in §C 3. Using the properties of f, we prove that all simple graphs satisfy $\mathbf{min\,LP = min\,ILP}$ in §C 4. We then reconstruct the blossom algorithm data structures from a *Blossom DLP solution* $\vec{y^*}$ (§C 5). Finally, we explain how to decode heterogeneous QEC codes (§C 6) more efficiently, using a technique from Fusion Blossom [4].

1. Blossom Algorithm Definitions

We use the definitions and theorems in [4] extensively to assist the proofs. We carefully identify the conflicts of the notations in [4] and this work, and minimize the confusion with the following conventions.

- We use a superscript $*$ for the concepts in the blossom algorithm, e.g. y^*, S^* and $\mathcal{O}^*$, to distinguish between similar concepts in [4] and this work.
- "$\mathcal{D}(S^*)$" refers to the *Progeny* of a blossom $S^* \in \mathcal{O}^*$ in [4]. Since we use $\mathcal{D}(\mathcal{E})$ to refer to *Defects of Error Pattern* in this work, we will use "$\mathcal{P}(S^*)$" instead for the *Progeny* of a blossom.
- $C(v)$ refers to the geometric concept of *Circle* in [4] while $C \in \mathcal{C}$ refers to the *Cluster* in this paper. We will not use the *Circle* $C(v)$ in this section to avoid confusions.

Fusion Blossom [4] relates the variables of *Blossom DLP solution* $y^*_{S^*}, \forall S^* \in \mathcal{O}^*$ with geometric *Covers* on the decoding graph. The geometric concepts not only help to visualize the dual variables, but also help prove the correctness using their finite-overlapping properties, like **Theorem: Node Cover Finite Overlap** [4]. The theorem only describes a *node*, which is a special kind of blossom who has no parent, i.e., not part of a larger blossom. This is fine in the blossom algorithm because it only cares about *nodes* in the alternating trees. However, in this work, we need to consider the internal structure of a *Cover*. Namely, each blossom has its own *Ring*, which is part of the *Cover* that it solely contributes to. The *Rings* of different blossoms are finite-overlapping, and the union of all the *Rings* of a node's *Progeny* constitute its *Cover*. We extend the definitions in [4] with the *Descendants*, *Rings* and the *DesCover Distance* that is useful to determine whether a point belongs to a *Ring*.

Definition: Descendants. Given a blossom $S^* \in \mathcal{O}^*$, its *Descendants* $\mathcal{J}(S^*)$ is the set of all blossoms with S^* as their ancestor: $\mathcal{J}(S^*) = \{C^* \in \mathcal{O}^* | y^*_{C^*} > 0, C^* \subsetneq S^*\}$.

A related concept in [4] is the *Progeny* $\mathcal{P}(S^*)$ of a blossom $S^* \in \mathcal{O}^*$, which includes its *Descendants* and itself, i.e., $\mathcal{P}(S^*) = \mathcal{J}(S^*) \cup \{S^*\}$. In this work, we need to clearly distinguish between the *Progeny* $\mathcal{P}(S^*)$ and the *Descendants* $\mathcal{J}(S^*)$ of a blossom.

Before defining the *Descendants Cover* and *Ring* for this work, we review the definition of *Cover* in [4].

Definition: Cover. The *Cover* of a blossom $S^* \in \mathcal{O}^*$ includes the points $p \in G$ (vertices V and all the points of the edges E in the decoding graph G) within certain distance from the defect vertices S^*.

$$\mathrm{Cover}(S^*) = \bigcup_{v \in S^*} \left\{ p \in G \middle| \mathrm{Dist}(p, v) \le \sum_{D^* \in \mathcal{P}(S^*) | v \in D^*} y^*_{D^*} \right\}$$

Definition: Descendants Cover. For a blossom $S^* \in \mathcal{O}^*$, $\mathrm{DesCover}(S^*)$ is the union of its *Descendants*' *Covers* and its defect vertices:

$$\mathrm{DesCover}(S^*) = \bigcup_{C^* \in \mathcal{J}(S^*)} \mathrm{Cover}(C^*) \cup S^*$$

Definition: Ring. Given a blossom $S^* \in \mathcal{O}^*$, $\mathrm{Ring}(S^*)$ is the exclusive part of its *Cover* by excluding its *Descendants Cover*:

$$\mathrm{Ring}(S^*) = \mathrm{Cover}(S^*) \setminus \mathrm{DesCover}(S^*)$$

Lemma: Ring Finite Overlap. Given blossoms $S^*_1, S^*_2 \in \mathcal{O}^*, S^*_1 \neq S^*_2$, $\mathrm{Ring}(S^*_1) \cap \mathrm{Ring}(S^*_2)$ is a finite set of points.

Proof. The proof is intuitive from the visualization in Fig. 23. Here we prove it rigorously.

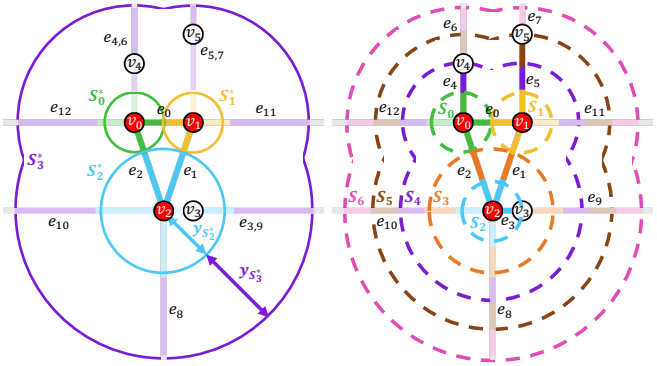
If S^*_1 and S^*_2 have different root node, $\mathrm{Root}(S^*_1) \neq \mathrm{Root}(S^*_2)$, their *Covers* have a finite overlap given **Theorem: Node Cover Finite Overlap** in [4]. Given the definition of *Rings*, $\mathrm{Ring}(S^*) \subseteq \mathrm{Cover}(S^*)$, we have

$$\begin{aligned} &|\mathrm{Cover}(S^*_1) \cap \mathrm{Cover}(S^*_2)| \text{ is finite,} \\ &\mathrm{Ring}(S^*_1) \subseteq \mathrm{Cover}(S^*_1), \mathrm{Ring}(S^*_2) \subseteq \mathrm{Cover}(S^*_2) \\ &\quad \Longrightarrow |\mathrm{Ring}(S^*_1) \cap \mathrm{Ring}(S^*_2)| \text{ is finite} \end{aligned} \tag{C1}$$

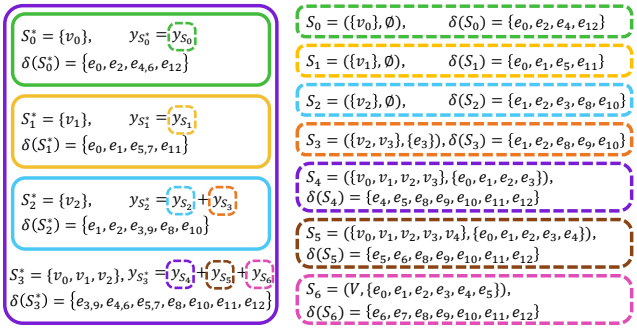
If S^*_1 and S^*_2 belong to the same node, we only need to prove that their *Rings* has a finite overlap at the moment they merge into the same node, since their *Rings* do not change afterwards according to *Lemma: Frozen Ring*. Before they merge into the same node, their *Rings* have a finite overlap due to the same reason in Eq. (C1).

In both cases, the overlap of the *Rings* of S^*_1 and S^*_2 remains finite.

□



(a) Blossom algorithm. (b) MWPF.



(c) *Blossom DLP solution* $\vec{y}^*$. (d) DLP solution $\vec{y}$.

FIG. 23: An example of mapping a *Blossom DLP solution* $\vec{y}^*$ to a DLP solution $\vec{y} = f(\vec{y}^*)$. (a,b) Each dual variable $y^*_{S^*}$ (y_S) is visualized by a *Ring* with a radius of $y^*_{S^*}$ (y_S). (c) The blossom algorithm works on the syndrome graph (§II C), which only includes the defect vertices $D = \{v_0, v_1, v_2, \cdots\}$. For reference, we show the non-defect vertices $\{v_3, v_4, v_5\}$ that are on the syndrome graph edges incident to $v_i \in D$, but note that these non-defect vertices do not exist in the syndrome graph. Each blossom $S^* \in \mathcal{B}^*$ maps to multiple *Hyperblossoms* $\mathcal{O}(S^*) \cap \mathcal{B}$. For example, S^*_2 maps to two *Hyperblossoms* $\mathcal{O}(S^*) \cap \mathcal{B} = \{S_2, S_3\}$. (d) To understand the mapping, imagine whenever the *Ring* of a blossom $S^* \in \mathcal{O}^*$ encounters a non-defect vertex $v \in \overline{D}$ when it grows, a new *Hyperblossom* $S = (V_S, E[V_S]) \in \mathcal{O}$ is created to include v, i.e., $v \in V_S$.

Lemma: Frozen Ring. The *Ring* of a blossom S^* does not change once it merges into a parent blossom S^*_p.

Proof. We use some properties of the blossom algorithm to prove this lemma. Once S^* merges into another blossom S^*_p where $S^* \subsetneq S^*_p$, according to the blossom algorithm procedure [4], the dual variables $y^*_{S^*}$ of these *Descendants* $S^* \in \mathcal{J}(S^*_p)$ do not change, unless S^* becomes a node again (under an event of blossom expansion).

It then suffices to prove that the *Ring* of S^* does not change when it is among the *Descendants* of S^*_p. By definition of *Cover*, it only depends on the dual variables of its *Progeny* $\mathcal{P}(S^*)$. The *Progeny* of S^* is among the *Descendants* of S^*_p, i.e., $\mathcal{P}(S^*) = \mathcal{J}(S^*) \cup \{S^*\} \subseteq \mathcal{J}(S^*_p)$. Thus, the *Cover* of S^* is frozen, i.e., $\text{Cover}(S^*)$ does not change. Similarly, the *Descendants Cover* of S^* is frozen, i.e., $\text{DesCover}(S^*)$ does not change. Thus, according to the definition, the *Ring* of S^* is frozen, i.e., $\text{Ring}(S^*) = \text{Cover}(S^*) \cap \text{DesCover}(S^*)$ does not change. □

Next, we define concepts that are useful to determine whether a vertex belongs to a *Ring* or not.

Definition: DesCover Distance. Given a vertex $v \in V$ and a blossom $S^* \in \mathcal{B}^*$, the *DesCover Distance*, denoted by $\text{DCDist}(S^*, v)$, is the minimum distance from v to the points in the *Descendants Cover* $\text{DesCover}(S^*)$.

$$\text{DCDist}(S^*, v) = \min_{p \in \text{DesCover}(S^*)} \text{Dist}(p, v)$$

Lemma: DesCover Distance Criteria. Given a blossom $S^* \in \mathcal{B}^*$ and a vertex $v \in V$, we have the criteria:

$$\text{DCDist}(S^*, v) = 0 \iff v \in \text{DesCover}(S^*) \quad \text{(C2)}$$
$$0 < \text{DCDist}(S^*, v) \le y^*_{S^*} \iff v \in \text{Ring}(S^*) \quad \text{(C3)}$$
$$\text{DCDist}(S^*, v) > y^*_{S^*} \iff v \notin \text{Cover}(S^*) \quad \text{(C4)}$$

Proof. The proofs of these criteria are intuitive from the visualization in Fig. 23, where the *Descendants Cover* $\text{DesCover}(S^*_3) = \text{Cover}(S^*_1) \cup \text{Cover}(S^*_2) \cup \text{Cover}(S^*_3)$ unions the region of the three circles of S^*_0, S^*_1 and S^*_2. Any point that has zero distance to this region falls in $\text{DesCover}(S^*_3)$ (Eq. (C2)). Any point that has a distance larger than $y_{S^*_3}$, marked by the radius of the *Ring* in Fig. 23c, is outside of the $\text{Cover}(S^*_3)$ (Eq. (C2)). The rest of the points are within $\text{Ring}(S^*_3)$. We prove them rigorously below. To prove Eq. (C2),

$$\begin{aligned} &\text{DCDist}(S^*, v) = 0 \\ &\quad \downarrow \text{(by definition of } \textit{DesCover Distance}) \\ &\iff \exists p \in \text{DesCover}(S^*), \text{Dist}(p, v) = 0 \\ &\iff v \in \text{DesCover}(S^*) \end{aligned} \quad \text{(C5)}$$

To prove Eq. (C4), we use the definition of *Descendants Cover* and *Cover*. We first prove Eq. (C4) for the case $\mathcal{J}(S^*) = \varnothing$ so that we only need to focus on $\mathcal{J}(S^*) \neq \varnothing$ later. When S^* does not have any *Descendants* $\mathcal{J}(S^*) = \varnothing$, it consists of a single defect vertex $S^* = \{u\}$. Its *Descendants Cover* is $\text{DesCover}(S^*) = S^* = \{u\}$ and thus $\text{DCDist}(S^*, v) = \text{Dist}(u, v)$. The *Cover* of S^* is $\text{Cover}(S^*) = \{p \in G | \text{Dist}(p, u) \le y^*_{S^*}\}$ and thus

$$v \notin \text{Cover}(S^*) \iff \text{DCDist}(S^*, v) = \text{Dist}(u, v) > y^*_{S^*}$$

We now prove Eq. (C4) for the case of $\mathcal{J}(S^*) \neq \varnothing$. In this case, every defect vertex $u \in S^*$ must belong to one of its *Descendants*' *Covers*, i.e., $\exists C^* \in \mathcal{J}(S^*), u \in \text{Cover}(C^*)$. we can then simplify the *Descendants Cover*

as $\text{DesCover}(S^*) = \bigcup_{C^* \in \mathcal{J}(S^*)} \text{Cover}(C^*)$. We have

$$
\begin{aligned}
&\text{DCDist}(S^*, v) > y^*_{S^*} \\
&\iff \forall p \in \text{DesCover}(S^*), \text{Dist}(p, v) > y^*_{S^*} \\
&\quad \downarrow \text{(given the above simplification of DesCover}(S^*)) \\
&\iff \forall p \in \bigcup_{C^* \in \mathcal{J}(S^*)} \text{Cover}(C^*), \text{Dist}(p, v) > y^*_{S^*} \\
&\iff \forall C^* \in \mathcal{J}(S^*), \forall p \in \text{Cover}(C^*), \text{Dist}(p, v) > y^*_{S^*} \\
&\quad \downarrow \text{(by definition of } \textit{Cover}) \\
&\iff \forall C^* \in \mathcal{J}(S^*), \forall u \in C^*, \forall p \in G, \\
&\qquad \Big(\text{Dist}(p, u) \leq \sum_{D^* \in \mathcal{P}(C^*) | u \in D^*} y^*_{D^*}\Big) \rightarrow \Big(\text{Dist}(p, v) > y^*_{S^*}\Big) \\
&\iff \forall C^* \in \mathcal{J}(S^*), \forall u \in C^*, \text{Dist}(u, v) > y^*_{S^*} + \sum_{D^* \in \mathcal{P}(C^*) | u \in D^*} y^*_{D^*} \\
&\quad \downarrow \text{(given } \textit{Lemma: Blossom Hierarchy Dual Sum}) \\
&\iff \forall u \in S^*, \text{Dist}(u, v) > y^*_{S^*} + \sum_{D^* \in \mathcal{J}(S^*) | u \in D^*} y^*_{D^*} \\
&\quad \downarrow \text{(given } \mathcal{P}(S^*) = \mathcal{J}(S^*) \cup \{S^*\} \text{ and } S^* \notin \mathcal{J}(S^*)) \\
&\iff \forall u \in S^*, \text{Dist}(u, v) > \sum_{D^* \in \mathcal{P}(S^*) | u \in D^*} y^*_{D^*} \\
&\quad \downarrow \text{(by definition of } \textit{Cover}) \\
&\iff v \notin \text{Cover}(S^*) \qquad \text{(C6)}
\end{aligned}
$$

To prove Eq. (C3), we simply use Eqs. (C5) and (C6):

$$
\begin{aligned}
&0 < \text{DCDist}(S^*, v) \leq y^*_{S^*} \\
&\quad \downarrow \text{(given Eq. (C5) and Eq. (C6))} \\
&\iff v \notin \text{DesCover}(S^*), v \in \text{Cover}(S^*) \\
&\quad \downarrow \text{(by definition of } \textit{Ring}) \\
&\iff v \in \text{Ring}(S^*)
\end{aligned}
$$

□

Lemma: Blossom Hierarchy Dual Sum. Given a blossom $S^* \in \mathcal{B}^*$ with $\mathcal{J}(S^*) \neq \varnothing$ and any its vertex $u \in S^*$,

$$\sum_{D^* \in \mathcal{J}(S^*) | u \in D^*} y^*_{D^*} = \max_{C^* \in \mathcal{J}(S^*)} \sum_{D^* \in \mathcal{P}(C^*) | u \in D^*} y^*_{D^*}$$

Proof. We prove the equation by first proving there exists $C^* \in \mathcal{J}(S^*)$ such that the dual sums above are equal, and then proving that for all $C^* \in \mathcal{J}(S^*)$, the right-hand side is no larger than the left-hand side.

Given the hierarchy of blossoms, there exists an immediate child $C^*_u \in \mathcal{J}(S^*)$ of S^* such that $u \in C^*_u$. All the blossoms that contains u must be either *Descendants* of C^*_u or it has to be S^* or S^*'s parent blossoms. Since S^* and its parent blossoms are not in $\mathcal{J}(S^*)$, we have $\{D^* \in \mathcal{J}(S^*) | u \in D^*\} = \{D^* \in \mathcal{P}(C^*_u) | u \in D^*\}$. Thus, there exists $C^* = C^*_u \in \mathcal{J}(S^*)$ such that

$$\sum_{D^* \in \mathcal{J}(S^*) | u \in D^*} y^*_{D^*} = \sum_{D^* \in \mathcal{P}(C^*) | u \in D^*} y^*_{D^*}$$

We next prove that for all $C^* \in \mathcal{J}(S^*)$, the right-hand side is no larger than the left-hand side. We consider two cases: $u \in C^*$ and $u \notin C^*$. In the first case, C^* must be one of the *Descendants* of C^*_u, i.e., $C^* \in \mathcal{P}(C^*_u)$. Given the hierarchy, the *Descendants* of C^* must be a subset of the *Descendants* of C^*_u, i.e., $\mathcal{P}(C^*) \subseteq \mathcal{P}(C^*_u)$. Thus,

$$\sum_{D^* \in \mathcal{P}(C^*) | u \in D^*} y^*_{D^*} \leq \sum_{D^* \in \mathcal{P}(C^*_u) | u \in D^*} y^*_{D^*} = \sum_{D^* \in \mathcal{J}(S^*) | u \in D^*} y^*_{D^*}$$

In the second case when $u \notin C^*$, u must not be in any of the *Descendants* of C^*, i.e., $\forall D^* \in \mathcal{P}(C^*), u \notin D^*$. Thus

$$\sum_{D^* \in \mathcal{P}(C^*) | u \in D^*} y^*_{D^*} = 0 \leq \sum_{D^* \in \mathcal{J}(S^*) | u \in D^*} y^*_{D^*}$$

In both cases, the right-hand side is no larger than the left-hand side. Thus, we have proved the lemma.

□

2. Dual Mapping Definition

Definition: Dual Mapping. Given a *Blossom DLP solution* $\vec{y^*}$, we map each blossom dual variable $y^*_{S^*}, \forall S^* \in \mathcal{B}^*$ to a set of DLP variables given the following rules. Let the non-defect vertices in the *Ring* be $\overline{D}(S^*) = \overline{D} \cap \text{Ring}(S^*)$, we sort these non-defect vertices based on their *DesCover Distance* to S^*. That is,

$$
\begin{aligned}
&\forall v_i, v_j \in \overline{D}(S^*), 1 \leq i, j \leq |\overline{D}(S^*)|, \\
&i \leq j \Longrightarrow \text{DCDist}(S^*, v_i) \leq \text{DCDist}(S^*, v_j)
\end{aligned}
$$

Let all the vertices in the *Descendants Cover* be $V(S^*) = V \cap \text{DesCover}(S^*)$. Then S^* maps to a set of *Invalid* subgraphs $\mathcal{O}(S^*) \subseteq \mathcal{O}$ defined below.

$$
\begin{aligned}
&\forall k, 0 \leq k \leq |\overline{D}(S^*)|, \\
&\qquad V_{S_k} \coloneqq V(S^*) \cup \{v_i \in \overline{D}(S^*) | \forall i, 1 \leq i \leq k\} \\
&\qquad \mathcal{O}(S^*) = \{S_k = (V_{S_k}, E[V_{S_k}]) | \forall k, 0 \leq k \leq |\overline{D}(S^*)|\}
\end{aligned}
$$

Note that here we use the edge set induced by the subgraph $E[V'] = \{e = (u, v) \in E | u \in V' \wedge v \in V'\}$ where both u and v must be vertices in the set $V' \subseteq V$. This is different from the set of edges incident to any vertex $E(V') = \{e = (u, v) \in E | u \in V' \vee v \in V'\}$.

We then construct the dual variables $y_{S_k}, \forall S_k \in \mathcal{O}(S^*)$. When there is no non-defect vertex in the *Ring*, we have $S_0 = (V(S^*), E[V(S^*)])$ and $y_{S_0} = y^*_{S^*}$. In general, they are defined below:

$$
\begin{aligned}
&\forall k, 0 \leq k < |\overline{D}(S^*)|, \\
&\qquad y_{S_k} \coloneqq \text{DCDist}(S^*, v_{k+1}) - \text{DCDist}(S^*, v_k) \\
&\text{if } k = |\overline{D}(S^*)|, \\
&\qquad y_{S_k} \coloneqq \begin{cases} y^*_{S^*}, & \text{if } \overline{D}(S^*) = \varnothing \\ y^*_{S^*} - \text{DCDist}(S^*, v_k), & \text{otherwise} \end{cases}
\end{aligned}
$$

We can simplify the above equations by defining v_0 as a point on the boundary of $\text{DesCover}(S^*)$ and $v_{|\overline{D}(S^*)|+1}$ as a point on the boundary of $\text{Cover}(S^*)$. By definition, we have $\text{DCDist}(S^*, v_0) = 0$ and $\text{DCDist}(S^*, v_{|\overline{D}(S^*)|+1}) = y^*_{S^*}$. In this way, the equations are simplified to:

$$\begin{aligned}&\forall k, 0 \le k \le |\overline{D}(S^*)|,\\ &\quad y_{S_k} \coloneqq \text{DCDist}(S^*, v_{k+1}) - \text{DCDist}(S^*, v_k)\end{aligned}$$

Definition: Inverse Dual Mapping. Given any DLP solution $\vec{y}$, we define the inverse of *Dual Mapping* f^{-1}:

$$\vec{y^*} = f^{-1}(\vec{y}): \quad y^*_{S^*} = \sum_{S \in \mathcal{O} | V_S \cap D = S^*} y_S, \quad \forall S^* \in \mathcal{O}^*$$

3. Dual Mapping Properties

We prove the following theorem about the *Dual Mapping*, with a couple lemmas and their proofs following.

Theorem: Blossom Dual Function. There exists a bijective function f from a *Blossom DLP solution* $\vec{y^*}$ to a DLP solution of the HyperBlossom algorithm $\vec{y} = f(\vec{y^*})$ while satisfying two conditions: their dual objectives are the same $\sum_{S \in \mathcal{O}} y_S = \sum_{S^* \in \mathcal{O}^*} y^*_{S^*}$, and, when a syndrome-graph edge (u, v) is tight [4] in $\vec{y^*}$, then the minimum-weight path between u and v in the DLP solution $\vec{y} = f(\vec{y^*})$ consists of all *Tight Edges*.

Proof. We prove that the *Dual Mapping* defined in §C 2 satisfies all the properties of f above, using several lemmas proved later in this section.

- f is bijective because we explicitly construct the *Inverse Dual Mapping* f^{-1} and prove that $f^{-1}(f(\vec{y^*})) = \vec{y^*}$ for any *Blossom DLP solution* $\vec{y^*}$ in *Lemma: Blossom Inverse Mapping*.
- any feasible *Blossom DLP solution* $\vec{y^*}$ maps to a feasible DLP solution $\vec{y} = f(\vec{y^*})$ given *Lemma: Dual Mapping Feasibility*.
- the dual objectives are the same, i.e., $\sum_{S \in \mathcal{O}} y_S = \sum_{S^* \in \mathcal{O}^*} y^*_{S^*}$, given *Lemma: Dual Mapping Sum*.
- the existence of tight edge (syndrome graph) implies the existence of minimum-weight tight path (decoding graph) and vice versa, given *Lemma: Dual Mapping Tight Edge and Path*. Note that the lemma requires the two vertices belong to different nodes. This requirement can be removed because according to *Lemma: Frozen Dual Mapping*, once they are merged into the same node, the *Tight Edges* remain tight.

□

Lemma: Blossom Inverse Mapping. The *Inverse Dual Mapping* f^{-1} satisfies $f^{-1}(f(\vec{y^*})) = \vec{y^*}$ for any *Blossom DLP solution* $\vec{y^*}$.

Proof. According to *Lemma: Hyperblossom Unique Mapping*, every *Hyperblossom* $S = (V_S, E_S)$ is mapped uniquely from a blossom $S^* = V_S \cap D$.

We prove the that dual variables are equal for every $S^* \in \mathcal{O}^*$. If S^* is a blossom, i.e., $y^*_{S^*} > 0$, then the inverse mapped y'_{S^*} is equal to the original $y^*_{S^*}$:

$$\begin{aligned}y'_{S^*} &= \sum_{S \in \mathcal{O} | V_S \cap D = S^*} y_S\\ &= \sum_{S \in \mathcal{O}(S^*)} y_S\\ &\downarrow \text{(given } \textit{Lemma: Blossom Dual Mapping Sum}\text{)}\\ &= y^*_{S^*}\end{aligned}$$

On the other hand, if S^* is not a blossom, i.e., $y^*_{S^*} = 0$, then it does not map to any *Invalid* subgraphs and thus any *Invalid* subgraph must have $y_S = 0$ if $V_S \cap D = S^*$. The inverse mapped $y'_{S^*} = 0 = y^*_{S^*}$ which is also equal to the original dual variable. Since the inverse mapped dual variable is equal to the original dual variable for every $S^* \in \mathcal{O}^*$, we have $f^{-1}(f(\vec{y^*})) = \vec{y^*}$

□

Lemma: Hyperblossom Unique Mapping. Given a *Blossom DLP solution* $\vec{y^*}$ and its corresponding *Dual Mapping* $\vec{y} = f(\vec{y^*})$, a *Hyperblossom* $S = (V_S, E_S)$ of $\vec{y}$ is mapped from a unique blossom $S^* = V_S \cap D$.

Proof. Given S is a *Hyperblossom* in $\vec{y} = f(\vec{y^*})$, it must be constructed as the k-th *Invalid* subgraph $S = S_k = (V_{S_k}, E[V_{S_k}])$ of some blossom $S^* \in \mathcal{O}^*$ according to the definition of *Dual Mapping*. We then prove that $V_{S_k} \cap D = S^*$ and thus S is mapped uniquely from S^*:

$$\begin{aligned}V_{S_k} \cap D &= \Big(V(S^*) \cup \{v_i \in \overline{D}(S^*) | \forall k, 1 \le i \le k\}\Big) \cap D\\ &= \Big(V(S^*) \cap D\Big) \cup\\ &\qquad \Big(\{v_i \in \overline{D}(S^*) | \forall k, 1 \le i \le k\} \cap D\Big)\\ &\downarrow \text{(given } \overline{D}(S^*) \cap D = \overline{D} \cap D \cap \text{Ring}(S^*) = \varnothing\text{)}\\ &= V(S^*) \cap D\\ &= V \cap \text{DesCover}(S^*) \cap D\\ &\downarrow \text{(given } D \subseteq V\text{)}\\ &= \text{DesCover}(S^*) \cap D\\ &\downarrow \text{(by definition of } \textit{Descendants Cover}\text{)}\\ &= \Big(\cup_{C^* \in \mathcal{J}(S^*)} \text{Cover}(C^*) \cup S^*\Big) \cap D\\ &\downarrow \text{(given } S^* \subseteq D\text{)}\\ &= \Big(D \cap \Big(\cup_{C^* \in \mathcal{J}(S^*)} \text{Cover}(C^*)\Big)\Big) \cup S^*\end{aligned}$$

We then prove that the term in the big parenthesis of the last step is a subset of S^*. It suffices to prove that the *Covers* of the *Descendants* of S^* do not

contain any other defect vertices apart from S^*, i.e., $\forall C^* \in \mathcal{J}(S^*), \text{Cover}(C^*) \cap D \subseteq S^*$.

We use contradiction to prove the above. Suppose there exists $C^* \in \mathcal{J}(S^*)$ and a defect vertex $v \in D \setminus S^*$ so that $v \in \text{Cover}(C^*)$. By definition of *DesCover Distance*, we have $\text{DCDist}(S^*, v) = 0$. On the other hand, given **Theorem: Node Cover Finite Overlap** in [4], such an external defect vertex $v \notin S^*$ must not be an internal point of $\text{Cover}(S^*)$. According to *Lemma: DesCover Distance Criteria*, we have $\text{DCDist}(S^*, v) \geq y^*_{S^*}$. To conclude a contradiction, we simply need to prove $y^*_{S^*} > 0$ because the *DesCover Distance* $\text{DCDist}(S^*, v)$ cannot be both 0 and no less than some positive value $y^*_{S^*}$. According to *Lemma: Blossom Dual Mapping Sum*, we have $y^*_{S^*} = \sum_{S_i \in \mathcal{O}(S^*)} y_{S_i} \geq y_{S_k}$. Given S_k is a *Hyperblossom*, we have $y^*_{S^*} \geq y_{S_k} > 0$. Thus, we have found a contradiction.

Applying the conclusion $\forall C^* \in \mathcal{J}(S^*), \text{Cover}(C^*) \cap D \subseteq S^*$ to the previous equation, we have

$$\begin{aligned} V_{S_k} \cap D &= \left(D \cap \left(\cup_{C^* \in \mathcal{J}(S^*)} \text{Cover}(C^*) \right) \right) \cup S^* \\ &= S^* \end{aligned}$$

□

Lemma: Dual Mapping Disjoint Sets. Given two blossoms $S^*_1, S^*_2 \in \mathcal{O}^*, S^*_1 \neq S^*_2$ of a *Blossom DLP solution*, their *Dual Mappings* involve two disjoint sets of *Invalid* subgraphs, i.e., $\mathcal{O}(S^*_1) \cap \mathcal{O}(S^*_2) = \varnothing$.

Proof. According to *Lemma: Hyperblossom Unique Mapping*, all the mapped *Invalid* subgraphs $\mathcal{O}(S^*_i)$ must contain exactly the defect vertices S^*_i, i.e., $\forall S \in \mathcal{O}(S^*_i), V_S \cap D = S^*_i$. Thus,

$$\begin{aligned} \mathcal{O}(S^*_1) \cap \mathcal{O}(S^*_2) &\subseteq \{S \in \mathcal{O} | V_S \cap D = S^*_1 \wedge V_S \cap D = S^*_2\} \\ &\downarrow (\text{given } S^*_1 \neq S^*_2) \\ &= \varnothing \end{aligned}$$

□

Lemma: DesCover Distance Order. Given a blossom $S^* \in \mathcal{O}^*$, a vertex $v \in V$ and the k-th non-defect vertex of the *Dual Mapping* $v_k \in \text{Ring}(S^*)$,

$$\begin{aligned} v \in V_{S_k} &\Longrightarrow \text{DCDist}(S^*, v) \leq \text{DCDist}(S^*, v_k) \\ v \notin V_{S_k} &\Longrightarrow \text{DCDist}(S^*, v) \geq \text{DCDist}(S^*, v_{k+1}) \end{aligned}$$

Proof. According to *Dual Mapping*, if $v \in V_{S_k} = V(S^*) \cup \{v_i \in \overline{D}(S^*) | \forall i, 1 \leq i \leq k\}$, then v is either in $V(S^*)$ or it belongs to $\overline{D}(S^*)$ no later than the position of v_k. In the first case, $\text{DCDist}(S^*, v) = 0 \leq \text{DCDist}(S^*, v_k)$ given *Lemma: DesCover Distance Criteria* (Eq. (C2)). In the second case, $\text{DCDist}(S^*, v) \leq \text{DCDist}(S^*, v_k)$ according to the definition of the sorting in *Dual Mapping*. Thus,

$$v \in V_{S_k} \Longrightarrow \text{DCDist}(S^*, v) \leq \text{DCDist}(S^*, v_k)$$

Similarly, if $v \notin V_{S_k}$, then v must not be in $V(S^*)$ and must be behind v_k in the sorted list of $\overline{D}(S^*)$ if $v \in \overline{D}(S^*)$. We consider two cases: $v \notin \text{Cover}(S^*)$ and $v \in \text{Cover}(S^*)$. If $v \notin \text{Cover}(S^*)$, then $\text{DCDist}(S^*, v) > y^*_{S^*} \geq \text{DCDist}(S^*, v_{k+1})$ given *Lemma: DesCover Distance Criteria* (Eq. (C3) and Eq. (C4)). If $v \in \text{Cover}(S^*)$, we further split into two cases. First, if $v \in D$, then this external defect vertex $v \notin S^*$ must appear on the boundary of $\text{Cover}(S)$ because otherwise it will violate the **Theorem: Node Cover Finite Overlap** in [4]. In this case, $\text{DCDist}(S^*, v) = y^*_{S^*} \geq \text{DCDist}(S^*, v_{k+1})$. Second, if $v \notin D$, then v must be behind v_k in the sorted $\overline{D}(S^*)$, i.e., $\text{DCDist}(S^*, v) \geq \text{DCDist}(S^*, v_{k+1})$. Together, we have

$$v \notin V_{S_k} \Longrightarrow \text{DCDist}(S^*, v) \geq \text{DCDist}(S^*, v_{k+1})$$

□

Lemma: Blossom Dual Mapping Sum. Given a blossom $S^* \in \mathcal{O}^*$ from the blossom algorithm, its *Dual Mapping* has the same dual sum.

$$\forall S^* \in \mathcal{O}^*, \sum_{S \in \mathcal{O}(S^*)} y_S = y^*_{S^*}$$

Proof. According to the definition of *Dual Mapping*,

$$\begin{aligned} \sum_{S \in \mathcal{O}(S^*)} y_S &= \sum_{0 \leq k \leq |\overline{D}(S^*)|} y_{S_k} \\ &= \text{DCDist}(S^*, v_{|\overline{D}(S^*)|+1}) - \text{DCDist}(S^*, v_0) \\ &= y^*_{S^*} - 0 = y^*_{S^*} \end{aligned}$$

□

Lemma: Dual Mapping Sum. Given a feasible *Blossom DLP solution* $\vec{y^*}$, its *Dual Mapping* $\vec{y} = f(\vec{y^*})$ has the same dual sum.

$$\sum_{S \in \mathcal{O}} y_S = \sum_{S^* \in \mathcal{O}^*} y^*_{S^*}$$

Proof. According to *Lemma: Dual Mapping Disjoint Sets*, each blossom $S^* \in \mathcal{O}^*$ maps to a disjoint subset of dual variables $\mathcal{O}(S^*) \in \mathcal{O}$. The other dual variables default to 0, so we have

$$\begin{aligned} \sum_{S \in \mathcal{O}} y_S &= \sum_{S^* \in \mathcal{O}^*} \sum_{S \in \mathcal{O}(S^*)} y_S \\ &\downarrow (\textit{Lemma: Blossom Dual Mapping Sum}) \\ &= \sum_{S^* \in \mathcal{O}^*} y^*_{S^*} \end{aligned}$$

□

Lemma: Hair DesCover Distance Order. Given a blossom S^* and a *Hyperblossom* $S_k \in \mathcal{O}(S^*) \cap \mathcal{B}$ from its *Dual Mapping*, a *Hair* edge $e = (u, v) \in \delta(S_k)$ satisfies:

$$\begin{aligned}\text{DCDist}(S^*, u) &\le \text{DCDist}(S^*, v_k) \\ &< \text{DCDist}(S^*, v_{k+1}) \le \text{DCDist}(S^*, v)\end{aligned}$$

Proof. We first prove that $u \in V_{S_k}$ and $v \notin V_{S_k}$, and then use *Lemma: DesCover Distance Order* to prove the inequality.

According to the definition of *Dual Mapping*, $S_k = (V_{S_k}, E[V_{S_k}])$. Its edges $E_{S_k} = E[V_{S_k}]$ includes all the edges between vertices V_{S_k}. Given $\delta(S_k) = E(V_{S_k}) \setminus E[V_{S_k}]$, one of the incident vertex of a *Hair* $e = (u, v) \in \delta(S_k)$ must be in V_{S_k} and the other must not be. Without losing generality, we assume $u \in V_{S_k}$, $v \notin V_{S_k}$.

According to the definition of *Dual Mapping*, the *Invalid* subgraph S_k corresponds to two points v_k and v_{k+1}:

$$\begin{aligned}&y_{S_k} = \text{DCDist}(S^*, v_{k+1}) - \text{DCDist}(S^*, v_k) > 0 \\ &\implies \text{DCDist}(S^*, v_k) < \text{DCDist}(S^*, v_{k+1})\end{aligned}$$

According to *Lemma: DesCover Distance Order* and $u \in V_{S_k}$, we have $\text{DCDist}(S^*, u) \le \text{DCDist}(S^*, v_k)$. Similarly, given $v \notin V_{S_k}$, we have $\text{DCDist}(S^*, v) \ge \text{DCDist}(S^*, v_{k+1})$. Putting together, we have

$$\begin{aligned}\text{DCDist}(S^*, u) &\le \text{DCDist}(S^*, v_k) \\ &< \text{DCDist}(S^*, v_{k+1}) \le \text{DCDist}(S^*, v)\end{aligned}$$

□

Lemma: Ring Contributes to Edge. If a blossom $S^* \in \mathcal{B}^*$ contributes to the edge constraint Eq. (4b) of $e \in E$, then its *Ring* overlaps with e for a non-zero length:

$$\sum_{S \in \mathcal{O}(S^*) | e \in \delta(S)} y_S > 0 \implies \text{Length}\Big(e \cap \text{Ring}(S^*)\Big) > 0$$

Proof. We prove that there exists a non-zero overlapping $e \cap \text{Ring}(S^*)$ when $\sum_{S \in \mathcal{O}(S^*) | e \in \delta(S)} y_S > 0$. It suffices to prove that there exists an infinite number of points of e that belongs to the *Ring* of S^*.

Given $\sum_{S \in \mathcal{O}(S^*) | e \in \delta(S)} y_S > 0$, there exists at least one $S_k \in \mathcal{O}(S^*)$ such that $e = (u, v) \in \delta(S_k)$ and $y_{S_k} > 0$. According to *Lemma: Hair DesCover Distance Order*,

$$\begin{aligned}\text{DCDist}(S^*, u) &\le \text{DCDist}(S^*, v_k) \\ &< \text{DCDist}(S^*, v_{k+1}) \le \text{DCDist}(S^*, v)\end{aligned}$$

Since the function $\text{DCDict}(S^*, p)$ is a continuous function given the points on the edge $e = (u, v)$, there exists an infinite number of points $p \in e$ such that

$$\text{DCDict}(S^*, v_k) < \text{DCDict}(S^*, p) < \text{DCDist}(S^*, v_{k+1})$$

According to *Lemma: DesCover Distance Criteria* and $\text{DCDict}(S^*, v_k) \ge 0$, $\text{DCDist}(S^*, v_{k+1}) \le y^*_{S^*}$, all such points belong to the *Ring* $p \in \text{Ring}(S^*)$. Thus, we have $\text{Length}\Big(e \cap \text{Ring}(S^*)\Big) > 0$.

□

Lemma: Max Ring Contribution. The contribution of a blossom $S^* \in \mathcal{B}^*$ on the constraint Eq. (4b) of an edge $e \in E$ is upper bounded by the length of the overlapping segment of e with the *Ring* of S^*.

$$\sum_{S_k \in \mathcal{O}(S^*) | e \in \delta(S_k)} y_{S_k} \le \text{Length}(e \cap \text{Ring}(S^*))$$

Proof. We consider two cases: $\text{Length}(e \cap \text{Ring}(S^*)) = 0$ or $\text{Length}(e \cap \text{Ring}(S^*)) > 0$.

If $\text{Length}(e \cap \text{Ring}(S^*)) = 0$, then according to *Lemma: Ring Contributes to Edge*, none of its mapped dual variables y_{S_k} contributes to the edge constraint Eq. (4b) of e, i.e., $\sum_{S_k \in \mathcal{O}(S^*) | e \in \delta(S_k)} y_{S_k} = 0$. In this case, both sides of the equation are zero and they satisfy the inequality.

If $\text{Length}(e \cap \text{Ring}(S^*)) > 0$, there might be multiple $S_k \in \mathcal{O}(S^*)$ that contribute to the edge, i.e., $e \in \delta(S_k)$ and $y_{S_k} > 0$. We first prove that each contributing S_k corresponds to a disjoint segment of $e \cap \text{Ring}(S_k)$ whose length is at least y_{S_k} if $e \in \delta(S_k)$. We then prove that the union of all such segments is $e \cap \text{Ring}(S^*)$ up to a finite number of points. With these, we can conclude the inequality.

We define the corresponding segment of S_k as all the points $p \in e$ whose *DesCover Distance* is in between $\text{DCDict}(S^*, v_k)$ and $\text{DCDict}(S^*, v_{k+1})$. That is,

$$\begin{aligned}E_k = \{p \in e | &\text{DCDict}(S^*, v_k) < \text{DCDict}(S^*, p) \\ &< \text{DCDist}(S^*, v_{k+1})\}\end{aligned}$$

We prove that the segments are disjoint, i.e., $E_i \cap E_j = \varnothing$ for all $i \ne j$. Without loss of generality, we assume $i < j$. According to the definition of *Dual Mapping*, we have $\text{DCDict}(S^*, v_{i+1}) \le \text{DCDict}(S^*, v_j)$ given $i + 1 \le j$. That is, the open upper bound of *DesCover Distance* in E_i is no larger than the open lower bound of that in E_j. Thus, the segments are disjoint.

We then prove that the union of the segments is $e \cap \text{Ring}(S^*)$ up to a finite number of points. Given *Lemma: DesCover Distance Criteria* (Eq. (C3)), $e \cap \text{Ring}(S^*) = \{p \in e | 0 < \text{DCDict}(S^*, p) \le y^*_{S^*}\}$. According to the definition of *Dual Mapping*, we have $\text{DCDict}(S^*, v_0) = 0$ and $\text{DCDict}(S^*, v_{|\overline{D}(S^*)|+1}) = y^*_{S^*}$. Thus, we have

$$\begin{aligned}\cup_{0 \le k \le |\overline{D}(S^*)|} E_k = &\bigcup_{0 \le k \le |\overline{D}(S^*)|} \Big\{p \in e \Big| \text{DCDict}(S^*, v_k) < \\ &\text{DCDict}(S^*, p) < \text{DCDist}(S^*, v_{k+1})\Big\}\end{aligned}$$

($\approx$ up to a finite number of points)

$$\begin{aligned}&\approx \Big\{p \in e \Big| \text{DCDict}(S^*, v_0) < \text{DCDict}(S^*, p) \\ &\qquad \le \text{DCDist}(S^*, v_{|\overline{D}(S^*)|+1})\Big\} \\ &= \{p \in e | 0 < \text{DCDict}(S^*, p) \le y^*_{S^*}\} \\ &= e \cap \text{Ring}(S^*)\end{aligned}$$

Finally, we prove that when $e = (u, v) \in \delta(S_k)$, the length of the corresponding segment E_k is at least y_{S_k},

i.e., $\text{Length}(E_k) \geq y_{S_k}$. According to *Lemma: Hair DesCover Distance Order*,

$$\begin{aligned}\text{DCDist}(S^*, u) &\leq \text{DCDist}(S^*, v_k) \\ &< \text{DCDist}(S^*, v_{k+1}) \leq \text{DCDist}(S^*, v)\end{aligned}$$

The function $\text{DCDict}(S^*, p)$ is a continuous function for the points on the edge $e = (u, v)$. Thus, for every value l between $\text{DCDist}(S^*, v_k)$ and $\text{DCDist}(S^*, v_{k+1})$, there exists a point $p \in e$ whose *DesCover Distance* is l:

$$\begin{aligned}&\forall l, \text{DCDict}(S^*, v_k) \leq l \leq \text{DCDict}(S^*, v_{k+1}), \\ &\quad \exists p \in e, \text{DCDict}(S^*, p) = l\end{aligned}$$

Thus, the length of E_k satisfies

$$\begin{aligned}\text{Length}(E_k) &= \text{Length}\Big(\Big\{p \in e \Big| \text{DCDict}(S^*, v_k) \\ &\qquad < \text{DCDict}(S^*, p) < \text{DCDist}(S^*, v_{k+1})\Big\}\Big) \\ &\geq \text{DCDist}(S^*, v_{k+1}) - \text{DCDist}(S^*, v_k) \\ &= y_{S_k}\end{aligned}$$

□

Lemma: Dual Mapping Feasibility. Given a *Blossom DLP solution* $\vec{y^*}$, its *Dual Mapping* $\vec{y} = f(\vec{y^*})$ is a feasible DLP solution.

Proof. We prove the feasibility of the dual variables $y_S, \forall S \in \mathcal{O}$ by proving that they satisfy both constraints Eq. (4a) and Eq. (4b) in DLP.

First, the dual variables default to $y_S = 0$ satisfying Eq. (4a). If a variable y_{S_k} is mapped from a blossom $S^* \in \mathcal{O}^*$, then $y_{S_k} = \text{DCDict}(S^*, v_{k+1}) - \text{DCDict}(S^*, v_k) \geq 0$ also satisfies Eq. (4a).

Second, we prove that for each edge $e \in E$, the constraint Eq. (4b) is also satisfied:

$$\begin{aligned}&\sum_{S \in \mathcal{O} | e \in \delta(S)} y_S \\ &\downarrow \text{(given } \textit{Lemma: Dual Mapping Disjoint Sets}\text{)} \\ &= \sum_{S^* \in \mathcal{O}^*} \sum_{S_k \in \mathcal{O}(S^*) | e \in \delta(S_k)} y_{S_k} \\ &\downarrow \text{(given } \textit{Lemma: Max Ring Contribution}\text{)} \\ &\leq \sum_{S^* \in \mathcal{O}^*} \text{Length}(e \cap \text{Ring}(S^*)) \\ &\downarrow \text{(given } \textit{Lemma: Ring Finite Overlap}\text{)} \\ &\leq w_e\end{aligned}$$

□

Lemma: Dual Mapping Tight Edge and Path. Given a *Blossom DLP solution* $\vec{y^*}$ and *Dual Mapping* f, if a syndrome-graph edge (u, v) is tight in $\vec{y^*}$ between two nodes $S_1^*, S_2^* \in \mathcal{O}^*$ with $u \in S_1^*, v \in S_2^*$, then the minimum-weight path between u and v consists of all *Tight Edges* in the DLP solution $\vec{y} = f(\vec{y^*})$.

Proof. We first prove that all the edges on the minimum-weight path must be fully covered, and then use *Lemma: Length Cover Equals Contrib* to show the equality.

We first prove that $e \in \text{minPath}(u, v)$ must be fully covered. This is because every point $p \in \text{minPath}(u, v)$ has $\text{Dist}(p, u) + \text{Dist}(p, v) = w_{(u,v)}$ by definition of the minimum-weight path. By definition of a tight edge in the syndrome graph, we have

$$\begin{aligned}\sum_{S^* \in \mathcal{O}^* | (u,v) \in \delta(S^*)} y^*_{S^*} &= \sum_{S^* \in \mathcal{O}^* | u \in S^*} y^*_{S^*} + \sum_{S^* \in \mathcal{O}^* | v \in S^*} y^*_{S^*} \\ &\downarrow (\text{Root}(u) = S_1^*, \text{Root}(v) = S_2^*) \\ &= \sum_{D^* \in \mathcal{P}(S_1^*) | u \in D^*} y^*_{D^*} + \sum_{D^* \in \mathcal{P}(S_2^*) | v \in D^*} y^*_{D^*} \\ &= w_{(u,v)} \\ &= \text{Dist}(p, u) + \text{Dist}(p, v)\end{aligned}$$

Thus, we have either $\text{Dist}(p, u) \leq \sum_{D^* \in \mathcal{P}(S_1^*) | u \in D^*} y^*_{D^*}$ or $\text{Dist}(p, v) \leq \sum_{D^* \in \mathcal{P}(S_2^*) | v \in D^*} y^*_{D^*}$, or both. According to the definition of *Cover*, p either belongs to $\text{Cover}(S_1^*)$ or $\text{Cover}(S_2^*)$, or both. Thus, we have $\text{Length}((\text{Cover}(S_1^*) \cup \text{Cover}(S_2^*)) \cap e) = w_e$.

We then prove $\text{Contrib}(S_1^*) + \text{Contrib}(S_2^*) = w_e$:

$$\begin{aligned}&\text{Contrib}(S_1^*) + \text{Contrib}(S_2^*) \\ &\downarrow \text{(by } \textit{Lemma: Length Cover Equals Contrib}\text{)} \\ &= \text{Length}(\text{Cover}(S_1^*) \cap e) + \text{Length}(\text{Cover}(S_2^*) \cap e) \\ &\downarrow \text{(by } \textbf{Theorem: Node Cover Finite Overlap} \text{ [4])} \\ &= \text{Length}((\text{Cover}(S_1^*) \cup \text{Cover}(S_2^*)) \cap e) \\ &= w_e\end{aligned}$$

Given *Lemma: Dual Mapping Disjoint Sets*, we have $\mathcal{O}(D_1^*) \cap \mathcal{O}(D_2^*) = \varnothing$ for any two blossoms $D_1^*, D_2^* \in \mathcal{B}^*, D_1^* \neq D_2^*$, and thus

$$\begin{aligned}\sum_{S \in \mathcal{O} | e \in \delta(S)} y_S &= \sum_{S^* \in \mathcal{B}^*} \sum_{S \in \mathcal{O}(S^*) | e \in \delta(S)} y_S \\ &\geq \sum_{S^* \in \mathcal{P}(S_1^*) \cup \mathcal{P}(S_2^*)} \sum_{S \in \mathcal{O}(S^*) | e \in \delta(S)} y_S \\ &\downarrow (\mathcal{P}(S_1^*) \cap \mathcal{P}(S_2^*) = \varnothing \text{ for nodes } S_1^* \neq S_2^*) \\ &= \sum_{S^* \in \mathcal{P}(S_1^*)} \sum_{S \in \mathcal{O}(S^*) | e \in \delta(S)} y_S + \\ &\qquad \sum_{S^* \in \mathcal{P}(S_2^*)} \sum_{S \in \mathcal{O}(S^*) | e \in \delta(S)} y_S \\ &= \text{Contrib}(S_1^*) + \text{Contrib}(S_2^*) \\ &= w_e\end{aligned}$$

Also, we have $\sum_{S \in \mathcal{O} | e \in \delta(S)} y_S \leq w_e$ because $\vec{y} = f(\vec{y^*})$ is a feasible DLP solution satisfying Eq. (4b), according to *Lemma: Dual Mapping Feasibility*. Together, we have $\sum_{S \in \mathcal{O} | e \in \delta(S)} y_S = w_e$, i.e., e is a *Tight Edge*. This concludes the proof that every edge on the minimum-weight path between u and v is a *Tight Edge*. □

Lemma: Length Cover Equals Contrib. Given a node $S^* \in \mathcal{B}^*$ and any edge $e \in \text{minPath}(u, v)$ between an internal defect $u \in S^*$ and an external defect $v \in D \setminus S^*$,

$$\text{Length}(\text{Cover}(S^*) \cap e) = \text{Contrib}(S^*),$$
$$\text{where Contrib}(S^*) = \sum_{D^* \in \mathcal{P}(S^*)} \sum_{S \in \mathcal{O}(D^*) | e \in \delta(S)} y_S$$

Proof. We prove it using mathematical induction. At the beginning of the algorithm, they are equal because $\text{Length}(\text{Cover}(S^*) \cap e) = \text{Contrib}(S^*) = 0$. When the covered segment is starting to increase because of the growth of the node S^*, we prove that by growing for a small length l, both $\text{Length}(\text{Cover}(S^*) \cap e)$ and $\text{Contrib}(S^*)$ increase by the same amount l.

First, given that the edge e is on the minimum-weight path between u and v, the distance $\text{Dist}(u, p)$ is monotonically increasing for the points on the edge e. Thus, according to the definition of *Cover*, the covered segment $\text{Cover}(S^*) \cap e$ is also increasing for the same length l.

Second, according to the blossom algorithm, the only changing dual variable among $\mathcal{P}(S^*)$ is the node S^* itself. Thus, $y^*_{S^*}$ increases by l. According to the definition of *Dual Mapping*, only the outmost $S_k \in \mathcal{O}(S^*)$ changes for the same amount of l. Together, we have

$$\Delta\text{Contrib}(S^*) = \Delta \sum_{S \in \mathcal{O}(S^*) | e \in \delta(S)} y_S = \begin{cases} \Delta y_{S_k} & \text{if } e \in \delta(S_k) \\ 0 & \text{otherwise,} \end{cases}$$

Thus, we only need to prove $e \in \delta(S_k)$ in order to prove that $\text{Contrib}(S^*)$ also increases by l.

Without losing generality, we assume $e = (a, b)$ where $a \in \text{Cover}(S^*)$ and $b \notin \text{Cover}(S^*)$. Given a cannot be on the boundary of $\text{Cover}(S^*)$ after it grows for a small length l, we have $a \in \overline{D}$ according to **Theorem: Node Cover Finite Overlap** [4]. The vertices of S_k includes all the vertices in the *Descendants Cover* of S^* as well as those non-defect vertices within a *DesCover Distance* of $y^*_{S^*}$. Thus, $a \in V_{S_k}$ and as a result, $e \in E(V_{S_k})$. Similarly, $b \notin V_{S_k}$ and as a result, $e \notin E[V_{S_k}]$. Given $E_{S_k} \subseteq E[V_{S_k}]$, we have $e \notin E_{S_k}$. Thus, by the definition of *Hair*, we have $e \in E(V_{S_k}) \setminus E_{S_k} = \delta(S_k)$.

Together, $\text{Length}(\text{Cover}(S^*) \cap e) = \text{Contrib}(S^*)$ for any edge $e \in \text{minPath}(u, v)$. □

Lemma: Frozen Dual Mapping. The *Dual Mapping* of a blossom S^*, including $\mathcal{O}(S^*)$ and their corresponding dual variables $\{y_{S_k}, S_k \in \mathcal{O}(S^*)\}$, does not change once S^* merges into a parent blossom S^*_p.

Proof. We use a similar proof to that of *Lemma: Frozen Ring*, with the properties of the blossom algorithm. Once S^* merges into another blossom S^*_p where $S^* \subsetneq S^*_p$, according to the blossom algorithm [4], the dual variables $y^*_{S^*}$ of these *Descendants* $S^* \in \mathcal{J}(S^*_p)$ do not change, unless S^* becomes a node again (under an event of blossom expansion).

It then suffices to prove that the $\mathcal{O}(S^*)$ and their dual variables do not change when it is among the *Descendants* of S^*_p. According to the definition of *Dual Mapping*, $\mathcal{O}(S^*)$ and their dual variables only depends on the *Descendants Cover* of S^* and the value of $y^*_{S^*}$. The value of $y^*_{S^*}$ does not change because $S^* \in \mathcal{J}(S^*_p)$. Thus, we only need to prove that the *Descendants Cover* of S^* does not change.

By definition of *Cover*, it only depends on the dual variables of its *Progeny* $\mathcal{P}(S^*)$. The *Progeny* of S^* is among the *Descendants* of S^*_p, i.e., $\mathcal{P}(S^*) = \mathcal{J}(S^*) \cup \{S^*\} \subseteq \mathcal{J}(S^*_p)$. Thus, the *Covers* of all *Descendants* of S^* does not change. According to the definition of *Descendants Cover*, $\text{DesCover}(S^*)$ does not change.

Together, the *Dual Mapping* of S^* is frozen, i.e., $\mathcal{O}(S^*)$ and their corresponding dual variables $\{y_{S_k}, S_k \in \mathcal{O}(S^*)\}$ do not change.

□

4. minLP = minILP for Simple Graphs

Theorem: Simple Graphs min LP = min ILP.

Proof. We prove **min LP = min ILP** using the inequality chain Eq. (5) by finding a pair of feasible parity factor $\vec{x}$ and feasible DLP solution $\vec{y}$ with the same weight $\sum_{e \in E} w_e x_e = \sum_{S \in \mathcal{O}} y_S$ for any simple graph $G = (V, E)$.

An MWPM decoder finds a pair of feasible parity factor $\vec{x}$ and *Blossom DLP solution* $\vec{y^*}$ with the same weight $\sum_{e \in E} w_e x_e = \sum_{S^* \in \mathcal{O}^*} y^*_{S^*}$ [1, 19]. We then use *Dual Mapping* to map $\vec{y^*}$ to a DLP solution $\vec{y} = f(\vec{y^*})$. Given *Lemma: Dual Mapping Sum*,

$$\sum_{S \in \mathcal{O}} y_S = \sum_{S^* \in \mathcal{O}^*} y^*_{S^*} = \sum_{e \in E} w_e x_e$$

Also, given *Lemma: Dual Mapping Feasibility*, $\vec{y}$ is a feasible DLP solution. □

5. Alternating Tree Reconstruction

We prove that an alternating tree data structure can be reconstructed from a *Blossom DLP solution* $\vec{y^*}$ alone, without any other information. In this way, we have access to the blossom algorithm data structures even though the *Relaxer* finder only has $\vec{y} = f(\vec{y^*})$ as input.

Theorem: Alternating Tree Reconstruction. It is possible to reconstruct alternating trees from a *Blossom DLP solution* $\vec{y^*}$.

Proof. We use a constructive proof by explicitly describing such an algorithm. We first reconstruct the hierarchy of blossoms from the *Blossom DLP solution* $\vec{y^*}$, and then use the tight syndrome-graph edges between the reconstructed nodes to rebuild alternating trees and matched pairs, as shown in Algorithm 13.

Algorithm 13 Alternating Tree Reconstruction

Input: G, D (defects), $\vec{y^*}$ (*Blossom DLP solution*)
Output: nodes in alternating trees and matched pairs
1: **procedure** AlternatingTreeReconstruct($G, D, \vec{y^*}$)
2: $\mathcal{B}^* \leftarrow \{y^*_{S^*} > 0 | S^* \in \mathcal{O}^*\}$
3: Nodes $\leftarrow \{\{v\} | v \in D\}$
4: **while** $\mathcal{B}^* \neq \varnothing$ **do**
5: $S^* \leftarrow \arg\min_{S^* \in \mathcal{B}^*} |S^*|$ ▷ the smallest blossom
6: Des $\leftarrow \{S^*_c \in \text{Nodes} | S^*_c \subseteq S^*\}$ ▷ children
7: assert $S^* \equiv \cup_{S^*_c \in \text{Des}} S^*_c$ ▷ form a blossom
8: Nodes $\leftarrow$ (Nodes $\setminus$ Des) $\cup \{S^*\}$
9: $\mathcal{B}^* \leftarrow \mathcal{B}^* \setminus \{S^*\}$
10: $T^* \leftarrow \{e \in E^* | \sum_{S^* \in \mathcal{O}^* | e \in \delta(S^*)} y^*_{S^*} = w_e\}$ ▷ compute tight syndrome-graph edges
11: $\Delta y_{S^*} \leftarrow +1, \forall S^* \in$ Nodes ▷ nodes grow by default
12: **for** $e \in T^*$ **do** ▷ resolve *Conflicts* [29] between nodes
13: PrimalPhaseResolve(Nodes, Conflict(e))
14: **return** Nodes

Given all the blossoms $\mathcal{B}^* = \{y^*_{S^*} > 0 | S^* \in \mathcal{O}^*\}$ of $\vec{y^*}$, we can reconstruct the hierarchy of blossoms. As shown in Algorithm 13 line 5, we do a bottom-up reconstruction by iterating through $\mathcal{B}^*$ from small blossoms to large blossoms. In this way, a child blossom is always visited before its parent blossom. Initially, each defect vertex corresponds to a node $\{v\}, \forall v \in D$ (line 3). Once a blossom S^* is visited, its children blossoms are removed from Nodes and their parent blossom S^* is added to Nodes (line 8).

With the nodes, we reconstruct the alternating trees and matched pairs by iterating over the tight syndrome-graph edges. To do that, we first let all the nodes grow $\Delta y^*_{S^*} = +1, \forall S^* \in$ Nodes (line 11). We then call the Primal Phase of the Parity Blossom algorithm [4] to address the *Conflicts* [29] between the nodes (line 13), each corresponding to a tight syndrome-graph edge. During this process, the nodes are organized into alternating trees and matched pairs.

□

6. More Efficient Heterogeneous QEC Decoding

Although the HyperBlossom algorithm with the *Blossom Relaxer* finder (§IV C) is step-by-step equivalent to the MWPM decoder, it is not as efficient as running the MWPM decoder directly. Here we describe a more efficient way of decoding a heterogeneous QEC code [24], where the decoding graph is partially simple. The idea is to directly run the MWPM decoder for the matchable subgraph, and then map the optimal *Blossom DLP solution* solution to an initial feasible DLP solution, so that the HyperBlossom algorithm can make progress on top of the initial solution.

A *matchable subgraph* $G' = (V', E[V'])$ is an induced subgraph of the decoding hypergraph $G = (V, E)$ where $|e| \leq 2, \forall e \in E[V']$. We treat the vertices $V_b = \{v \in V' | E(v) \not\subseteq E[V']\}$ as temporary boundaries [4], since they could be incident to some external hyperedges. There must exists a feasible *Blossom DLP solution* for G' because we only relax the parity constraints on the boundary vertices V_b and that G is a *Valid* graph. In this case, we can use an MWPM decoder to find an optimal *Blossom DLP solution* for G'.

Once the MWPM decoder finishes, we then map the optimal *Blossom DLP solution* $\vec{y^*}$ to a feasible DLP solution of G' using *Dual Mapping*. In this case, a feasible DLP solution $\vec{y} = f(\vec{y^*})$ of the subgraph G' is also a DLP solution of G, according to *Lemma: Subgraph* $\mathcal{O}' \subseteq \mathcal{O}$.

To see why $\vec{y}$ is a feasible DLP solution on G, we use the property of the temporary boundaries V_b: any boundary vertex $v \in V_b$ is not strictly inside any *Cover*, i.e., either v does not belong to any *Cover* or v is on the boundary of a *Cover*. Since the *Covers* never grow beyond any boundary vertex $v \in V_b$, all the external hyperedges $E \setminus E[V']$ are not propagated by any blossom, i.e., $\forall e \in E \setminus E[V'], \forall S^* \in \mathcal{B}^*, \text{Length}(e \cap \text{Ring}(S^*)) = 0$. According to *Lemma: Max Ring Contribution*, we have $\sum_{S_k \in \mathcal{O}(S^*) | e \in \delta(S_k)} y_{S_k} = 0$. That is, the DLP solution $\vec{y}$ satisfies Eq. (4b) for every hyperedge, i.e., $\forall e \in E \setminus E[V'], \sum_{S \in \mathcal{O} | e \in \delta(S)} y_S = 0$. The DLP solution $\vec{y}$ also satisfies Eq. (4b) for all edges in $E[V']$ because it is a feasible DLP solution of G'. Thus, $\vec{y}$ satisfies Eq. (4b) for all hyperedges $e \in E = (E \setminus E[V']) \cup E[V']$. Also, $\vec{y}$ satisfies Eq. (4a) for all $S \in \mathcal{O}$ because $y_S = 0, \forall S \in \mathcal{O} \setminus \mathcal{O}'$. Together, $\vec{y}$ is a feasible DLP solution on G.

The HyperBlossom algorithm then takes the feasible DLP solution $\vec{y} = f(\vec{y^*})$ as the initial solution, and makes progress on top of it. This is more efficient than running the *Blossom Relaxer* finder, bypassing all the complication of the HyperBlossom algorithm for this matchable subgraph.