$(1-\epsilon)$-Approximate Maximum Weighted Matching in $\text{poly}(1/\epsilon, \log n)$ Time in the Distributed and Parallel Settings

Shang-En Huang
huangaul@bc.edu
Boston College
Chestnut Hill, MA, USA

Hsin-Hao Su
suhx@bc.edu
Boston College
Chestnut Hill, MA, USA

ABSTRACT

The maximum weighted matching (MWM) problem is one of the most well-studied combinatorial optimization problems in distributed graph algorithms. Despite a long development on the problem, and the recent progress of Fischer, Mitrovic, and Uitto [16] who gave a $\text{poly}(1/\epsilon, \log n)$-round algorithm for obtaining a $(1-\epsilon)$-approximate solution for unweighted maximum matching, it had been an open problem whether a $(1-\epsilon)$-approximate MWM can be obtained in $\text{poly}(1/\epsilon, \log n)$ rounds in the CONGEST model. Algorithms with such running times were only known for special graph classes such as bipartite graphs [1] and minor-free graphs [8]. For general graphs, the previously known algorithms require exponential in $(1/\epsilon)$ rounds for obtaining a $(1-\epsilon)$-approximate solution [13] or achieve an approximation factor of at most 2/3 [1]. In this work, we settle this open problem by giving a deterministic $\text{poly}(1/\epsilon, \log n)$-round algorithm for computing a $(1-\epsilon)$-approximate MWM for general graphs in the CONGEST model. Our proposed solution extends the algorithm of Fischer, Mitrovic, and Uitto [16], blends in the sequential algorithm from Duan and Pettie [11] and the work of Faour, Fuchs, and Kuhn [13]. Interestingly, this solution also implies a CREW PRAM algorithm with $\text{poly}(1/\epsilon, \log n)$ span using only $O(m)$ processors, and a $\text{poly}(1/\epsilon)$-passes algorithm in the semi-streaming model.

ACM Reference Format:
Shang-En Huang and Hsin-Hao Su. 2023. $(1-\epsilon)$-Approximate Maximum Weighted Matching in $\text{poly}(1/\epsilon, \log n)$ Time in the Distributed and Parallel Settings. In *ACM Symposium on Principles of Distributed Computing (PODC '23), June 19–23, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3583668.3594570

1 INTRODUCTION

Matching problems are central problems in the study of both sequential and distributed graph algorithms. A *matching* is a set of edges that do not share endpoints. Given a weighted graph $G = (V, E, w)$, where $w : E \rightarrow \{1, \ldots, W\}$, the maximum weight matching (MWM) problem is to compute a matching M with the maximum weight, where the weight of M is defined as $\sum_{e \in M} w(e)$. Given an unweighted graph $G = (V, E)$, the maximum cardinality matching (MCM) problem is to compute a matching M such that $|M|$ is maximized. Clearly, the MCM problem is a special case of the MWM problem. For $0 < \epsilon < 1$, a $(1-\epsilon)$-MWM (or $(1-\epsilon)$-MCM) is a $(1-\epsilon)$-approximate solution to the MWM (or MCM) problem. Throughout the paper, we let $n = |V|$ and $m = |E|$.

In distributed computing, the MCM and MWM problems have been studied extensively in the CONGEST model and the LOCAL model. In these models, nodes host processors and operate in synchronized rounds. In each round, each node sends a message to its neighbors, receives messages from its neighbors, and performs local computations. The time complexity of an algorithm is defined to be the number of rounds used. In the LOCAL model, there are no limits on the message size, while the CONGEST model is a more realistic model where the message size is limited by $O(\log n)$ bits per link per round.

Computing an exact MWM requires $\Omega(n)$ rounds in both the CONGEST model and the LOCAL model (e.g., consider the graph G to be a unit-weight even cycle.) Thus, the focus has been on developing efficient approximate algorithms. In fact, the approximate MWM problem is also one of the few classic combinatorial optimization problems where it is possible to bypass the notorious CONGEST model lower bound of $\tilde{\Omega}(D + \sqrt{n})$ by [36], where D denotes the diameter of the graph. For $(1-\epsilon)$-MWM in the CONGEST model, the lower bounds of [1, 29] imply that polynomial dependencies on $(\log n)$ and $(1/\epsilon)$ are needed. Whether matching upper bounds can be achieved is an intriguing and important problem, as also mentioned in [13]:

> "Obtaining a $(1-\epsilon)$-approximation (for MWM) in $\text{poly}(\log n/\epsilon)$ CONGEST rounds is one of the key open questions in understanding the distributed complexity of maximum matching."

A long line of studies has been pushing progress toward the goal. Below, we summarize the current fronts made by the existing results (also see Table 1).

- c-MWM algorithms for $c < 2/3$. Wattenhofer and Wattenhofer [38] were among the first to study the MWM problem in the CONGEST model. They gave an algorithm for computing a $(1/5)$-MWM that runs in $O(\log^2 n)$ rounds. Then Lotker, Patt-Shamir, and Rosén [31] developed an algorithm that computes a $(1/4 - \epsilon)$-MWM in $O((1/\epsilon)\log(1/\epsilon)\log n)$ rounds. Later, Lotker, Patt-Shamir, and Pettie [30] improved the approximation ratio and the number of rounds to $1/2 - \epsilon$ and $O(\log(1/\epsilon) \cdot \log n)$ respectively. [5] gave a $(1/2)$-MWM algorithm that runs in $O(T_{\text{MIS}}(n) \cdot \log W)$ rounds, where $T_{\text{MIS}}(n)$ is the time needed to compute a maximal independent set (MIS) in an n-node graph. Fischer [14] gave a deterministic algorithm that computes a $(1/2 - \epsilon)$-MWM in $O(\log^2 \Delta \cdot \log \epsilon^{-1} + \log^* n)$ rounds by using a rounding approach,

PODC '23, June 19–23, 2023, Orlando, FL, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0121-4/23/06.
https://doi.org/10.1145/3583668.3594570

Table 1: Previous results on MCM and MWM in the CONGEST model. Deterministic algorithms are marked with "Det." †Bipartite graphs. ‡Minor-free graphs.

Citation	Problem	Ratio	Running Time
[28]	MCM	$\frac{1}{2}$	$O(\log n)$
[4]	MCM	$\frac{1}{2}$	$O(\log n)$
[32]	MCM	$\frac{1}{2}$	$O(\log n)$
[23]	MCM	$\frac{1}{2}$	$O(\log^4 n)$ Det.
[38]	MWM	$\frac{1}{5}$	$O(\log^2 n)$
[31]	MWM	$\frac{1}{4} - \epsilon$	$O(\epsilon^{-1} \log \epsilon^{-1} \log n)$
[30]	MCM†	$1 - \epsilon$	$O(\log n/\epsilon^3)$
	MCM	$1 - \epsilon$	$2^{O(1/\epsilon)} \cdot O(\epsilon^{-4} \log \epsilon^{-1} \cdot \log n)$
	MWM	$\frac{1}{2} - \epsilon$	$O(\log(1/\epsilon) \cdot \log n)$
[5]	MWM	$\frac{1}{2}$	$O(T_{\text{MIS}}(n) \cdot \log W)$
	MWM	$\frac{1}{2}$	$O(\Delta + \log n)$ Det.
	MWM	$\frac{1}{2} - \epsilon$	$O(\log \Delta / \log \log \Delta)$
	MCM	$1 - \epsilon$	$2^{O(1/\epsilon)} \cdot O(\log \Delta / \log \log \Delta)$
[14]	MCM	$\frac{1}{2}$	$O(\log^2 \Delta \cdot \log n)$ Det.
	MWM	$\frac{1}{2} - \epsilon$	$O(\log^2 \Delta \cdot \log \epsilon^{-1} + \log^* n)$ Det.
[1]	MWM†	$1 - \epsilon$	$O(\frac{\log(\Delta W)}{\epsilon^2} + \frac{\log^2 \Delta + \log^* n}{\epsilon})$ Det.
	MWM	$\frac{2}{3} - \epsilon$	$O(\frac{\log(\Delta W)}{\epsilon^2} + \frac{\log^2 \Delta + \log^* n}{\epsilon})$ Det.
[13]	MWM	$1 - \epsilon$	$2^{O(1/\epsilon)} \cdot \text{polylog}(n)$ Det.
[16]	MCM	$1 - \epsilon$	$\text{poly}(\log n, 1/\epsilon)$ Det.
[8]	MWM‡	$1 - \epsilon$	$\text{poly}(\log n, 1/\epsilon)$
new	MWM	$1 - \epsilon$	$\text{poly}(\log n, 1/\epsilon)$ Det.

where Δ is the maximum degree. Then Ahmadi, Khun, and Oshman [1] gave another rounding approach for $(2/3 - \epsilon)$-MWM that runs in $O(\frac{\log(\Delta W)}{\epsilon^2} + \frac{\log^2 \Delta + \log^* n}{\epsilon})$ rounds deterministically. The rounding approaches of [14] and [1] inherently induce a 2/3 approximation ratio because the linear programs they consider have an integrality gap of 2/3 in general graphs.

- Exponential-in-$(1/\epsilon)$ algorithms. [30] showed that the random bipartition technique can be applied to get a randomized $2^{O(1/\epsilon)} \cdot O(\log n)$-round $(1 - \epsilon)$-MCM algorithm. Such a technique was later also applied by [13], who gave a deterministic $2^{O(1/\epsilon)} \cdot \text{poly}(\log n)$-round algorithm for $(1 - \epsilon)$-MWM.
- Bipartite graphs and other special graphs. For bipartite graphs, Lotker et al. [30] gave an algorithm for $(1 - \epsilon)$-MCM that runs in $O(\log n/\epsilon^3)$ rounds. Ahmadi et al. [1] showed that $(1 - \epsilon)$-MWM in bipartite graphs can be computed in $O(\log(\Delta W)/\epsilon^2 + (\log^2 \Delta + \log^* n)/\epsilon)$ rounds deterministically. Recently, Chang and Su [8] showed that a $(1 - \epsilon)$-MWM can be obtained in $\text{poly}(1/\epsilon, \log n)$ rounds in minor-free graphs with randomization by using expander decompositions.
- Algorithms using larger messages. It was shown in [15] that the $(1 - \epsilon)$-MWM problem can be reduced to hypergraph maximal matching problems, which are known to be solvable efficiently in the LOCAL model. A number of $\text{poly}(1/\epsilon, \log n)$-round algorithms are known for obtaining $(1 - \epsilon)$-MWM [20, 25, 34]. The current fastest algorithms are by [25], who gave a $O(\epsilon^{-3} \log(\Delta + \log \log n) + \epsilon^{-2} (\log \log n)^2)$-round randomized algorithm and a $O(\epsilon^{-4} \log^2 \Delta + \epsilon^{-1} \log^* n)$-round deterministic algorithm.

Recently, Fischer, Mitrović, and Uitto [16] made significant progress by giving a $\text{poly}(1/\epsilon, \log n)$-round algorithm for computing a $(1-\epsilon)$-MCM — the unweighted version of the problem. Despite the progress, the complexity of $(1 - \epsilon)$-MWM still remains unsettled. We close the gap by giving the first $\text{poly}(1/\epsilon, \log n)$ round algorithm for computing $(1 - \epsilon)$-MWM in the CONGEST model. The result is summarized as Theorem 1.1.

Theorem 1.1. *There exists a deterministic* CONGEST *algorithm that solves the* $(1 - \epsilon)$-MWM *problem in* $\text{poly}(1/\epsilon, \log n)$ *rounds.*

In the parallel setting, Hougardy and Vinkemeier [26] gave a CREW PRAM[1] algorithm that solves the $(1 - \epsilon)$-MWM problem in $O(\frac{1}{\epsilon} \log^5 n)$ span with $n^{O(1/\epsilon)}$ processors. However, it is still not clear whether a *work-efficient* algorithm with a $\text{poly}(1/\epsilon, \log n)$-span and $O(m)$ processors exists. Our CONGEST algorithm can be directly simulated in the CREW PRAM model, obtaining a $\text{poly}(1/\epsilon, \log n)$ span algorithm that uses only $O(m)$ processors. The total work matches the best known sequential algorithm of [11], up to $\text{poly}(1/\epsilon, \log n)$ factors.

Corollary 1.2. *There exists a deterministic CREW PRAM algorithm that solves the* $(1-\epsilon)$-MWM *problem with* $\text{poly}(1/\epsilon, \log n)$ *span and uses only* $O(m)$ *processors.*

1.1 Related Works

Sequential Model: For the sequential model, by the classical results of [7, 17, 33], it was known that the exact MCM and MWM problems can be solved in $\tilde{O}(m\sqrt{n})$ time. For approximate matching, it is well-known that a $\frac{1}{2}$-MWM can be computed in linear time by computing a maximal matching. Although near-linear time algorithms for $(1 - \epsilon)$-MCM were known in the 1980s [17, 33], it was a challenging task to obtain a near-linear time α-MWM algorithm for the approximate ratio $\alpha > \frac{1}{2}$. Several near-linear time algorithms were developed, such as $(\frac{2}{3} - \epsilon)$-MWM [9, 35] and $(\frac{3}{4} - \epsilon)$-MWM [10, 24]. Duan and Pettie [11] gave the first near-linear time algorithms for $(1 - \epsilon)$-MWM, which runs in $O(\epsilon^{-1} \log(1/\epsilon) \cdot m)$ time.

Semi-Streaming Model: In the semi-streaming model, the celebrated results of $(1 - \epsilon)$-MWM with $\text{poly}(1/\epsilon, \log n)$ passes were already known by Ahn and Guha [2, 3]. Thus, in the semi-streaming model, the focus has been on obtaining algorithms with $o(\log n)$ dependencies on n. The state of the art algorithms for $(1 - \epsilon)$-MWM still have exponential dependencies on $(1/\epsilon)$ (see [18]). Recently, Fischer, Mitrović, and Uitto [16] made a breakthrough in the semi-streaming model, obtaining a $\text{poly}(1/\epsilon)$ passes algorithm for the $(1 - \epsilon)$-MCM problem. Our CONGEST algorithm translates to an $\text{poly}(1/\epsilon) \cdot \log W$ passes algorithm in the semi-streaming model. Bernstein and Dudeja [6] pointed out that, with the reduction from Gupta and Peng [22], an input instance can be reduced into $O(\log_{1/\epsilon} W)$ instances of $(1 - O(\epsilon))$-MWM such that, the largest weight W' in each instance can be upper bounded by $W' = (1/\epsilon)^{O(1/\epsilon)}$. Our algorithm extends to the first $\text{poly}(1/\epsilon)$

[1]A parallel random access machine that allows concurrent reads but requires exclusive writes.

passes algorithm for the $(1-\epsilon)$-MWM problem, see the full version [27].

Theorem 1.3 ([27, Appendix D]). *There exists a deterministic algorithm that returns an $(1-\epsilon)$-approximate maximum weighted matching using $\text{poly}(1/\epsilon)$ passes in the semi-streaming model. The algorithm requires $O(n \cdot \log W \cdot \text{poly}(1/\epsilon))$ words of memory.*

We remark that the results of Ahn and Guha [2, 3] do not translate easily to a CONGEST algorithm within $\text{poly}(1/\epsilon, \log n)$ rounds. In particular, in [2] the algorithm reduces to solving several instances of minimum odd edge cut[2]. It seems hard to solve minimum odd edge cut in CONGEST, given the fact that approximate minimum edge cut has a lower bound $\tilde{\Omega}(D + \sqrt{n})$ [19], where D is the diameter of the graph. On the other hand, in [3] the runtime per pass could be as high as $n^{O(1/\epsilon)}$, so it would be inefficient in CONGEST.

1.2 Technique Overview

Our approach is to parallelize Duan and Pettie's [11] near-linear time algorithm, which involves combining the recent approaches of [8] and [16] as well as several new techniques. The algorithm of [11] is a primal-dual based algorithm that utilizes Edmonds' formulation [12]. Roughly speaking, the algorithm maintains a matching M, a set of active blossoms $\Omega \subseteq 2^V$, dual variables $y : V \to \mathbb{R}$ and $z : 2^V \to \mathbb{R}$ (see Section 2 for details of blossoms). It consists of $O(\log W)$ scales with exponentially decreasing step sizes. Each scale consists of multiple primal-dual iterations that operate on a contracted **unweighted** subgraph, G_{elig}/Ω, which they referred to as the *eligible graph*. For each iteration in scale i, it tries to make progress on both the primal variables (M, Ω) and the dual variables (y, z) by the step size of the scale.

Initially, $\Omega = \emptyset$ so no blossoms are contracted. The first step in adjusting the primal variable is to search for an (inclusion-wise) maximal set of augmenting paths in the eligible graph and augment along them. After the augmentation, their edges will disappear from the eligible graph. Although [11] showed that such a step can be performed in linear time in the sequential setting, it is unclear how it can be done efficiently in $\text{poly}(1/\epsilon, \log n)$ time in the CONGEST model or the PRAM model. Specifically, for example, it is impossible to find the augmenting paths of length $\Theta(n)$ in Figure 1a in such time in the CONGEST model.

Our first ingredient is an idea from [8], where they introduced the weight modifier Δw and dummy free vertices to effectively remove edges and free vertices from the eligible graph. They used this technique to integrate the expander decomposition procedure into the algorithm of [11] for minor-free graphs. As long as the total number of edges and free vertices removed is small, one can show that the final error can be bounded.

With this tool introduced, it becomes more plausible that a maximal set of augmenting paths can be found in $\text{poly}(1/\epsilon, \log n)$ time, as we may remove edges to cut the long ones. Indeed, in *bipartite graphs*, this can be done by partitioning matched edges into layers. An edge is in the i-th layer if the shortest alternating path from any free vertex that ends at it contains exactly i matched edges.

[2]The goal is to return a mincut $(X, V \setminus X)$ among all subsets $X \subseteq V$ with an odd cardinality and $|X| = O(1/\epsilon)$.

Let M_i be the set of matched edges of the i-th layer. It must be that the removal of M_i disconnects all augmenting paths that contain more than i matched edges. Let $i^* = \arg\min_{1 \le i \le 1/\epsilon} |M_i|$ and thus $|M_{i^*}| \le \epsilon |M|$. The removal of M_{i^*} would cause all the leftover augmenting paths to have lengths of $O(1/\epsilon)$.

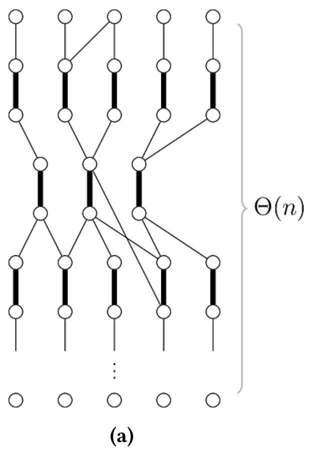
In general graphs, the above *path-cutting technique* no longer works. The removal of M_i would not necessarily disconnect augmenting paths that contain more than i matched edges. Consider the example in Figure 1b: for any matched edge e, the shortest alternating path from a free vertex that ends at e contains at most 2 matched edges. There is a (unique) augmenting path from α to β with 12 matched edges. However, the removal of M_5 (notice that $5 < 12$) would not disconnect this augmenting path, since $M_5 = \emptyset$. One of the technical challenges is to have an efficient procedure to *find a small fraction of edges whose removal cut all the remaining long augmenting paths in general graphs.*

Secondly, the second step of the primal-dual iterations of [11] is to find a maximal set of full blossoms reachable from free vertices and add them to Ω so they become contracted in the eligible graph. The problem here is that such a blossom can have a size as large as $\Theta(n)$ (See Figure 1c), so contracting it would take $\Theta(n)$ time in the CONGEST model. So the other technical challenge is *to ensure such blossoms will not be formed, possibly by removing a small fraction of edges and free vertices.* In general, these technical challenges are to remove a small fraction of edges and free vertices to achieve the so-called *primal blocking condition*, which we formally define in Property 3.2.

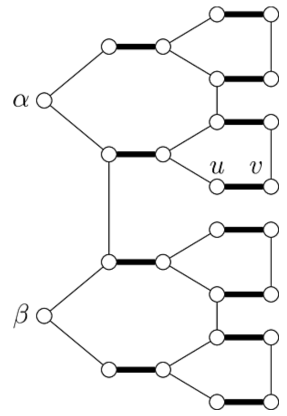
Note that the challenge may become more involved after the first iteration, where Ω is not necessarily empty. It may be the case that a blossom found in G_{elig}/Ω contains a very small number of vertices in the contracted graph G_{elig}/Ω but is very large in the original graph G. In this case, we cannot add it to Ω either, as it would take too much time to simulate algorithms on G_{elig}/Ω in the CONGEST model if Ω has a blossom containing too many vertices in G. Therefore, we also need to ensure such a blossom is never formed.

To overcome these challenges, our second ingredient is the parallel DFS algorithm of Fischer, Mitrovic, and Uitto [16]. In [16], they developed a procedure for finding an almost maximal set of k-augmenting paths in $\text{poly}(k)$ rounds, where a k-augmenting path is an augmenting path of length at most k. We show that the path-cutting technique for bipartite graphs can be combined seamlessly with a tweaked, *vertex-weighted version* of [16] to overcome these challenges for general graphs.

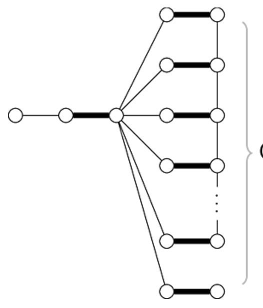
The central idea of [16] is parallel DFS [21]. A rough description of the approach of [16] is the following: Start a bounded-depth DFS from each free vertex where the depth is bounded by $O(k)$ and each search maintains a cluster of vertices. The clusters are always vertex-disjoint. In each step, each search tries to enlarge the cluster by adding the next edge from its active path. If there is no such edge, the search will back up one edge on its active path. If the search finds an augmenting path that goes from one cluster to the other, then the two clusters are removed from the graph. Note that this is a very high-level description for the purpose of understanding our usage, the actual algorithm of [16] is much more



(a)



(b)



(c)

Figure 1: Note that in these examples, we have $\Omega = \emptyset$ and so that $G/\Omega = G$.

involved. For example, it could be possible that the search from one cluster overtakes some portion of another cluster.

The key property shown in [16] is that at any point of the search all the remaining k-augmenting paths must pass one of the edges on the active paths, so removing the edges on active paths of the searches (in addition to the removal of clusters where augmenting paths are found) would cut all k-augmenting paths. Moreover, after searching for $\text{poly}(k)$ steps, it is shown at most $1/\text{poly}(k)$ fraction of searches remain active. Since each DFS will only search up to a depth of $O(k)$, the number of edges on the active paths is at most $O(k) \cdot 1/\text{poly}(k) = 1/\text{poly}(k)$ fraction of the searches. In addition, we note that the process has an extra benefit that, roughly speaking, if a blossom is ought to be contracted in the second step of [11], it will lie entirely within a cluster or it will be far away from any free vertices.

To better illustrate how we use [16] to overcome these challenges, we first describe our procedure for the first iteration of [11], where $\Omega = \emptyset$. In this case, we run several iterations [16] to find a collection of k-augmenting paths, where $k = \text{poly}(1/\epsilon, \log n)$, until the number of k-augmenting paths found is relatively small. Then remove (1) the clusters where augmenting paths have been found and (2) the active paths in the still active searches. By removing a structure, we meant using the weight modifier technique from [8] to remove the matched edges and free vertices inside the structure.

At this point, all the k-augmenting paths either overlap within the collection of k-augmenting paths or have been cut. The remaining augmenting paths must have lengths more than k. To cut them, we contract all the blossoms found within each cluster. As the search only runs for $\text{poly}(k)$ steps, each cluster has at most $\text{poly}(k)$ vertices so these blossoms can be contracted in each cluster on a vertex locally by aggregating the topology to the vertex in $\text{poly}(k)$ rounds. The key property we show is that after the contraction, if we assign each blossom a weight proportional to its size, the weighted $O(k)$-neighborhood of the free vertices becomes bipartite. The reason why this is correct is that the weighted distance is now an overestimate of the actual distance, and there are no full blossoms reachable within distance k from the free vertices in the graph now. Since the weighted $O(k)$-neighborhood from the free vertices are bipartite, we can run the aforementioned, but a weighted version, path-cutting technique on it to remove some edges augmenting paths of weighted length more than k. The weight assignment to the blossoms ensures that we will only remove a small fraction of the edges.

Starting from the second iteration of [11], the set of active blossoms Ω may not be empty anymore. We will need to be careful to not form any large nested blossoms after the Fischer-Mitrovic-Uitto parallel DFS algorithm (FMU-search), where the size of a blossom is measured by the number of vertices it contains in the original graph. To this end, when running the FMU-search, we run a weighted version of it, where each contracted vertex in G_{elig}/Ω is weighted proportional to the number of vertices it represents in the original graph. This way we can ensure the weight of each cluster is $\text{poly}(k)$ and so the largest blossom it can form will be $\text{poly}(k)$.

In order to generalize the properties guaranteed by FMU-search, one may have to open up the black-box and redo the whole sophisticated analysis of [16]. However, we show that the properties can be guaranteed by a blossom-to-path simulation analysis, where each weighted blossom is replaced by an unweighted path. The properties guaranteed by FMU-search from the transformed unweighted graph can then be carried back to the blossom-weighted graph.

Organization: In Section 2, we define the basic notations and give a brief overview of the scaling approach of [11] as well as the modification of [8]. In Section 3, we describe our modified scaling framework. In Section 4, we describe how [16] can be augmented to run in contracted graphs where vertices are weighted. In Section 5, we describe our Approx_Primal procedure for achieving the primal blocking conditions.

2 PRELIMINARIES AND ASSUMPTIONS

Throughout the paper, we denote $G = (V, E, \hat{w})$ to be the input weighted undirected graph, with an integer weight function $\hat{w} : E \to \{1, 2, \ldots, W\}$.

Matchings and Augmenting Paths: Given a matching M, a vertex is *free* if it is not incident to any edge in M. An *alternating path* is a path whose edges alternate between M and $E \setminus M$. An *augmenting* *path* P is an alternating path that begins and ends with free vertices. Given an augmenting path P, let $M \oplus P = (M \setminus P) \cup (P \setminus M)$ denote the resulting matching after we augment along P. Note that we must have $|M \oplus P| = |M| + 1$.

Linear Program for MWM: Edmonds [12] first formulated the matching polytope for general graphs. On top of the bipartite graph linear programs, there are additional exponentially many constraints over $\mathcal{V}_{odd}$ — all odd sized subsets of vertices. In this paper, we follow Edmonds' [12] linear program formulation for MWM for the graph (V, E, w):

$$\begin{array}{lll} \max & \sum_{e \in E} w(e)x(e) & \\ \text{st.} & \forall u \in V, \quad \sum_v x(uv) \le 1 & \\ & \forall B \in \mathcal{V}_{odd}, \quad \sum_{u,v \in B} x(uv) \le \frac{|B|-1}{2} & \textbf{(Primal)} \\ & x(e) \ge 0 \ \forall e \in E & \end{array}$$

Dual Variables: The variables $y(u)$ and $z(B)$ are called the *dual variables.* For convenience, given an edge $e = uv$, we define

$$yz(e) = y(u) + y(v) + \sum_{B \in \mathcal{V}_{odd} : e \in E(B)} z(B).$$

$$\begin{array}{lll} \min & \sum_{u \in V} y(u) + \sum_{B \in \mathcal{V}_{odd}} \frac{|B|-1}{2} z(B) & \\ \text{st.} & \forall uv \in E, & \\ & \quad y(u) + y(v) + \sum_{B \ni u,v} z(B) \ge w(uv) & \textbf{(Dual)} \\ & y(u) \ge 0, z(B) \ge 0 & \end{array}$$

Blossoms: A blossom is specified with a vertex set B and an edge set E_B. A trivial blossom is when $B = \{v\}$ for some $v \in V$ and $E_B = \emptyset$. A non-trivial blossom is defined recursively: If there are an odd number of blossoms $B_0 \dots B_\ell$ connected as an odd cycle by $e_i \in B_i \times B_{[(i+1) \mod (\ell+1)]}$ for $0 \le i \le \ell$, then $B = \bigcup_{i=0}^{\ell} B_i$ is a blossom with $E_B = \{e_0 \dots, e_\ell\} \cup \bigcup_{i=0}^{\ell} E_{B_i}$. It can be shown inductively that $|B|$ is odd and so $B \in \mathcal{V}_{odd}$. A blossom is *full* if $|M \cap E_B| = (|B| - 1)/2$. The only vertex that is not adjacent to the matched edges in a full blossom is called the *base* of B. Note that $E(B) = \{(u, v) \mid u, v \in B\}$ may contain edges not in E_B.

Active Blossoms: A blossom is *active* whenever $z(B) > 0$. We use Ω to denote the set of active blossoms throughout the execution of the algorithm. Throughout the execution, we maintain the property that only full blossoms will be contained in Ω. Moreover, the set of active blossoms Ω forms a laminar (nested) family, which can be represented by a set of rooted trees. The leaves of the trees are the trivial blossoms. If a blossom B is defined to be the cycle formed by $B_0, \dots, B_\ell$, then B is the parent of $B_0, \dots, B_\ell$. The blossoms that are represented by the roots are called the *root blossoms.*

Blossom-Contracted Graphs: Given Ω, let G/Ω denote the unweighted simple graph obtained by contracting all the root blossoms in Ω. A vertex in G/Ω is *free* if the vertices it represents in G contain a (unique) free vertex. The following lemma guarantees that the contraction of the blossoms does not tuck away all augmenting paths.

Lemma 2.1. *([11, Lemma 2.1]) Let Ω be a set of full blossoms with respect to a matching M.*

(1) If M is a matching in G, then M/Ω is a matching in G/Ω.

(2) Every augmenting path P' w.r.t. M/Ω in G/Ω extends to an augmenting path P w.r.t. M in G.

(3) Let P' and P be as in (2). Then Ω remains a valid set of full blossoms after the augmentation $M \leftarrow M \oplus P$.

Definition 2.2. Let v be a vertex in G/Ω, we use $\hat{v}$ to denote the set of vertices in G that contract to v. Also, given a set of vertices S, define $\hat{S} = \bigcup_{v \in S} \hat{v}$. For a free vertex f in G/Ω, we define $\dot{f}$ to be the unique free vertex in $\hat{f}$. Given a matched edge $e \in M/\Omega$, we use $\hat{e}$ to denote its corresponding matched edge in M.

Conversely, given a set of vertices $S \in \Omega$, let v_S^{Ω} be the vertex in G/Ω obtained by contracting S in G. Given a free vertex in G, let f^{Ω} denote the unique free vertex in G/Ω that contains f. Given a set of free vertices F of G, define $F^{\Omega} = \{f^{\Omega} \mid f \in F\}$. Similarly, given a matched edge $e \in M$, if both endpoints belong to different blossoms in Ω, then we define e^{Ω} to be the corresponding matched edge in M/Ω.

Definition 2.3. Let H be a subgraph of G with a matching M. We denote the set of free vertices in H by $F(H)$ and the set of matched edges in H by $M(H)$.

Definition 2.4 (Inner, outer, and reachable vertices). Let F be a set of free vertices in a graph H with matching M. Let $V_{in}^{H,M}(F)$ and $V_{out}^{H,M}(F)$ denote the set of vertices that are reachable from F with odd-length augmenting paths and even-length augmenting paths respectively. Define $R^{H,M}(F) = V_{in}^{H,M}(F) \cup V_{out}^{H,M}(F)$. When the reference to H and M are clear, we will omit the superscripts and write $R(F)$, $V_{in}(F)$, and $V_{out}(F)$ respectively.

Notice that using Definition 2.2, we have $\hat{V}_{in}(F) = \bigcup_{v \in V_{in}(F)} \hat{v}$ and $\hat{V}_{out}(F) = \bigcup_{v \in V_{out}(F)} \hat{v}$.

2.1 Assumptions on Edge Weights and Approximation Ratio

Since we are looking for a $(1 - \epsilon)$-approximation, we can always re-scale the edge weights to be $O(n/\epsilon)$ while introducing at most $(1-\Theta(\epsilon))$ error (see [11, Section 2]). Therefore, we can assume that $\epsilon > 1/n^2$ and so $W \le n^3$ and $O(\log W) = O(\log n)$; for otherwise we may aggregate the whole network at a node in $O(1/\epsilon) = O(n^2)$ rounds and have it compute a MWM locally. Let $\epsilon' = \Theta(\epsilon)$ be a parameter that we will choose later. We also assume without loss of generality that both W and ϵ' are powers of two.

2.2 Assumption of $O((1/\epsilon) \log^3 n)$ Weak Diameter

To begin, we process our input graph by applying a diameter reduction theorem developed by [13] to claim that we may assume that the graph we are considering has a broadcast tree of depth $O((1/\epsilon) \log^3 n)$ that can be used to aggregate and propagate information.

Theorem 2.5 ([13], Theorem 7). *Let $T_{SC}^{\alpha}(n, D)$ be the time required for computing an α-approximation for the* MWM *problem in the* SUPPORTED CONGEST *model with a communication graph of diameter D. Then, for every $\epsilon \in (0, 1]$, there is a* $\mathrm{poly}(\log n, 1/\epsilon) + O(\log n \cdot T_{SC}^{\alpha}(n, O((1/\epsilon) \log^3 n)))$*-round* CONGEST *algorithm to compute a $(1-\epsilon)\alpha$-approximation of* MWM *in the* CONGEST *model. If* *the given* SUPPORTED CONGEST *model algorithm is deterministic, then the resulting* CONGEST *model algorithm is also deterministic.*

The SUPPORTED CONGEST model is the same as the CONGEST model except that the input graph can be a subgraph of the communication graph. The above theorem implies that we can focus on solving the problem on G as if we were in the CONGEST model, except that we have access to a broadcast tree (potentially outside G) where an aggregation or a broadcast takes $O((1/\epsilon)\log^3 n)$ rounds. We slightly abuse the notation and say that G has a *weak diameter* of $O((1/\epsilon)\log^3 n)$.

With Theorem 2.5, we may broadcast W (the upper bound on edge weights) to every node in $O((1/\epsilon)\log^3 n)$ rounds. We remark that the assumption to the weak diameter is required not only in our algorithm but also in the $(1-\epsilon)$-MCM CONGEST algorithm described in [16][3].

2.3 Duan and Pettie's Scaling Framework

The scaling framework for solving MWM using the primal-dual approach was originally proposed by Gabow and Tarjan [17]. Let $L = \lfloor \log_2 W \rfloor$. A typical algorithm under this scaling framework consists of $L+1$ scales. In each scale i, such an algorithm puts its attention to the graph with *truncated weights* (whose definition varies in different algorithms). As i increases, these truncated weights typically move toward the actual input weight.

Duan and Pettie [11] introduced a scaling algorithm to solve the $(1-\epsilon)$-MWM problem. They proposed a new *relaxed complementary slackness criterion* (see Lemma 2.7). The criterion changes between iterations. At the end of the algorithm, the criterion can be used to certify the desired approximation guarantee of the maintained solution. Unlike Gabow and Tarjan's framework [17], Duan and Pettie's framework [11] allows the matching found in the previous scale to be carried over to the next scale without violating the feasibility, thereby improving the efficiency. In order to obtain this carry-over feature, Duan and Pettie also introduce the *type j edges* in their complementary slackness criterion.

Definition 2.6 (Type j Edges). A matched edge or a blossom edge is of *type j* if it was last made a matched edge or a blossom edge in scale j.

Let $\delta_0 = \epsilon' W$ and $\delta_i = \delta_0/2^i$ for all $i \in [0, L]$. At each scale i, the *truncated weight* of an edge e is defined as $w_i(e) = \delta_i \lfloor \hat{w}(e)/\delta_i \rfloor$. The relaxed complementary slackness criteria are based on the truncated weight at each scale.

Lemma 2.7 (Relaxed Feasibility and Complementary Slackness [11, Property 3.1]). *After each iteration $i = [0, L]$, the algorithm explicitly maintains the set of currently matched edges M, the dual variables $y(u)$ and $z(B)$, and the set of active blossoms $\Omega \subseteq \mathcal{V}_{odd}$. The following properties are guaranteed:*

1. **Granularity.** *For all $B \in \mathcal{V}_{odd}$, $z(B)$ is a nonnegative multiple of δ_i. For all $u \in V(G)$, $y(u)$ is a multiple of $\delta_i/2$.*
2. **Active Blossoms.** *Ω contains all B with $z(B) > 0$ and all root blossoms B have $z(B) > 0$.*
3. **Near Domination.** *For all $e \in E$, $yz(e) \geq w_i(e) - \delta_i$.*
4. **Near Tightness.** *If e is a type j edge, then $yz(e) \leq w_i(e) + 2(\delta_j - \delta_i)$.*
5. **Free Vertex Duals.** *If $u \in F(G)$ and $v \notin F(G)$ then $y(u) \leq y(v)$.*

Eligible Graph: To achieve Lemma 2.7 efficiently, at each scale an *eligible graph G_{elig}* is defined. An edge e is said to be *eligible*, if (1) $e \in E_B$ for some $B \in \Omega$, (2) $e \notin M$ and $yz(e) = w(e) - \delta_i$, or (3) $e \in M$ and $yz(e) - w_i(e)$ is a nonnegative integer multiple of δ_i. G_{elig} is the graph that consists of all edges that are currently eligible.

The algorithm initializes with an empty matching $M \leftarrow \emptyset$, an empty set of active blossoms $\Omega \leftarrow \emptyset$, and high vertex duals $y(u) \leftarrow W/2 - \delta_0/2$ for all $u \in V$. Then, in each scale $i = 0, 1, \ldots, L$ the algorithm repeatedly searches for a maximal set Ψ of vertex disjoint augmenting paths in G_{elig}, augments these paths, searches for new blossoms, adjust dual variables, and dissolves zero-valued inactive blossoms. These steps are iteratively applied for $O(1/\epsilon')$ times until the free vertex duals $y(v)$ reach $W/2^{i+2} - \delta_i/2$ whenever $i \in [0, L)$ or 0 whenever $i = L$. At the end of scale L, Lemma 2.7 guarantees a matching with the desired approximate ratio. We emphasize that the correctness of Lemma 2.7 relies on the fact that Ψ is maximal in G_{elig}, and the subroutine that searches for Ψ is a modified depth first search from Gabow and Tarjan [17] (see also [33, 37].) Unfortunately, some returned augmenting paths in Ψ could be very long. We do not immediately see an efficient parallel or distributed implementation of this subroutine.

2.4 Chang and Su's Scaling Framework

Chang and Su [8] noticed that it is possible to relax Duan and Pettie's framework further, by introducing the *weight modifiers* $\Delta w(e)$ that satisfy the following new invariants (appended to Lemma 2.7 with some changes to the other properties) after each iteration:

6. **Bounded Weight Change.** *The sum of $|\Delta w(e)|$ is at most $\epsilon' \cdot \hat{w}(M^*)$, where M^* is a maximum weight matching with respect to $\hat{w}$.*

Chang and Su showed that it is possible to efficiently obtain a maximal set Ψ of augmenting paths from G_{elig} in an expander decomposed H-minor-free graph. By carefully tweaking the definition of the eligibility of an edge, their modified Duan-Pettie framework fits well into the expander decomposition in the CONGEST model. Notice that Chang and Su's scaling algorithm depends on a "center process" in each decomposed subgraph. The center process in each subgraph can obtain the entire subgraph topology within polylog(n) rounds, with the assumption that the underlying graph is H-minor-free. Thus, a maximal set of augmenting paths in each subgraph can then be computed sequentially in each center process. This explains two non-trivial difficulties: First, it is not clear if the same framework can be generalized to general graphs. Furthermore, the sequential subroutine searching for augmenting paths may return long augmenting paths. It is not clear how to efficiently implement this subroutine in the PRAM model or the semi-streaming model.

[3]The application of Theorem 2.5 can also tie up loose ends left in [16], where they presented a semi-streaming algorithm first and then described the adaption to other models. One of the primitives, Storage in item (v) in Section 6 assumed a memory of $\Omega(n \text{ poly } 1/\epsilon)$ is available to all nodes. This may be needed in some of their procedures, e.g. counting $|M_H|$ in Algorithm 7. The running time was not analyzed, but it may take $O(\textit{diameter})$ rounds to implement in the CONGEST model.

Our scaling framework for general graphs is modified from both Duan-Pettie [11] and Chang-Su [8]. In Section 3 we present our modified scaling framework. With the adaption of [16], we believe our framework is simpler compared with Chang and Su [8]. Benefiting from [16] searching for short augmenting paths, the new framework can now be efficiently implemented in the PRAM model and the semi-streaming model.

3 OUR MODIFIED SCALING FRAMEWORK

Our modified scaling framework maintains the following variables during the execution of the algorithm:

M:	The set of matched edges.
$y(u)$:	The dual variable defined on each vertex $u \in V$.
$z(B)$:	The dual variable defined on each $B \in \mathcal{V}_{odd}$.
Ω:	$\Omega \subseteq \mathcal{V}_{odd}$ is the set of *active blossoms*.
$\Delta w(e)$:	The weight modifier defined on each edge $e \in E$.

Our algorithm runs for $L+1 = \lfloor \log_2 W \rfloor + 1$ scales. In each scale i, the same granularity value $\delta_i = \delta_0/2^i$ is used, where $\delta_0 := \epsilon' W$. Moreover, the truncated weight $w_i(e)$ is now derived from the *effective weight* $w(e) := \hat{w}(e) + \Delta w(e)$, namely $w_i(e) = \delta_i \lfloor w(e)/\delta_i \rfloor$. There will be $O(1/\epsilon')$ iterations within each scale. Within each iteration, the algorithm subsequently performs augmentations, updates to weight modifiers, dual adjustments, and updates to active blossoms (the details are described later in this section and in Fig. 2).

Similar to Chang and Su's framework [8] but opposed to Duan and Pettie's framework [11], the y-values of free vertices are no longer the same during the execution. To make sure that there are still $O(1/\epsilon')$ iterations in each scale, a special quantity τ is introduced. Within each scale i, the quantity τ will be decreased from $W/2^{i+1} - \delta_{i+1}/2$ to a specified target value $W/2^{i+2} - \delta_i/2$ (or 0 if $i = L$). Intuitively, τ is the desired free vertex dual which gets decreased by $\delta_i/2$ after every dual adjustment as in [11] (hence $O(1/\epsilon')$ iterations per scale). However, in both [8] and our framework, some free vertices will be isolated from the eligible graph. This isolation is achieved by increasing the y-value of the free vertex by δ_i. Therefore, τ can be seen as a lower bound to all free vertex duals. Our modified relaxed complementary slackness (Property 3.1) guarantees that the sum of such gaps will be small.

Eligible Graph: The eligible graph G_{elig} is an unweighted subgraph of G defined dynamically throughout the algorithm execution. The edges in G_{elig} are *eligible edges*. Conceptually, eligible edges are "tight", which are either blossom edges or the ones that nearly violate the complementary slackness condition. The precise definition of such eligible edges is given in Definition 3.1.

Definition 3.1. At scale i, an edge is *eligible* if at least one of the following holds.

(1) $e \in E_B$ for some $B \in \Omega$.
(2) $e \notin M$ and $yz(e) = w_i(e) - \delta_i$.
(3) $e \in M$ is a type j edge and $yz(e) = w_i(e) + 2(\delta_j - \delta_i)$.

We remark that (3) is more constrained compared to the Duan-Pettie framework [11]. With the new definition of (3), an eligible edge can be made ineligible by adjusting its weight modifier $\Delta w(e)$. Now we describe the relaxed complementary slackness properties:

Property 3.1. *(Relaxed Complementary Slackness) After scale i:*

(1) **Granularity.** *$z(B)$ and $w_i(e)$ are non-negative multiples of δ_i for all $B \in \mathcal{V}_{odd}$, $e \in E$ and $y(u)$ is a non-negative multiple of $\delta_i/2$ for all $u \in V$.*
(2) **Active Blossoms.** *All blossoms in Ω are full. If $B \in \Omega$ is a root blossom then $z(B) > 0$; if $B \notin \Omega$ then $z(B) = 0$. Non-root active blossoms may have zero z-values.*
(3) **Near Domination.** *For all edges $e \in E$, $yz(e) \geq w_i(e) - \delta_i$.*
(4) **Near Tightness.** *If e is a type j edge, then $yz(e) \leq w_i(e) + 2(\delta_j - \delta_i)$.*
(5) **Accumulated Free Vertex Duals.** *The y-values of all free vertices have the same parity as multiples of $\delta_i/2$. Moreover, $\sum_{v \in F(G)} y(v) \leq \tau \cdot |F(G)| + \epsilon' \cdot \hat{w}(M^*)$, where M^* is a maximum weight matching w.r.t. $\hat{w}$ and τ is a variable where $y(v) \geq \tau$ for every v. Also, τ decreases to 0 when the algorithm ends.*
(6) **Bounded Weight Change.** *The weight modifiers are non-negative. Moreover, the sum of $\Delta w(e)$ is at most $\epsilon' \cdot \hat{w}(M^*)$.*

With the modified relaxed complementary slackness properties, the following Lemma 3.2 guarantees the desired approximate ratio of the matching at the end of the algorithm. As the proof technique is similar to [11] and [8], we defer the proof of Lemma 3.2 to the full version [27].

Lemma 3.2. *Suppose that y, z, M, Ω, and Δw satisfy Property 3.1 at the end of scale L. Then $\hat{w}(M) \geq (1-\epsilon) \cdot \hat{w}(M^*)$.*

Iterations in each Scale: There will be $O(1/\epsilon')$ iterations within each scale. In each scale i, the ultimate goal of the algorithm is to make progress on primal (M) and dual (y, z) solutions such that they meet the complementary slackness properties (Property 3.1). This can be achieved by iteratively seeking for a set of augmenting paths Ψ, updating the matching $M \leftarrow M \oplus \cup_{P \in \Psi} P$, and then performing dual adjustments on y and z variables. However, in order to ensure that dual variables are adjusted properly, we enforce the following *primal blocking conditions* for Ψ:

Property 3.2 (*Primal Blocking Conditions*).

(1) No augmenting paths exist in G_{elig}/Ω.
(2) No full blossoms can be reached from any free vertices in G_{elig}/Ω via alternating paths.

Here we briefly explain why Property 3.2 leads to satisfactory dual adjustments. In a dual adjustment step, the algorithm decreases the y-values of inner vertices in $\hat{V}_{in}(F(G_{elig}/\Omega))$ by $\delta_i/2$ and increases the y-values of outer vertices in $\hat{V}_{out}(F(G_{elig}/\Omega))$ by $\delta_i/2$. Property 3.2 ensures that $\hat{V}_{in}(F(G_{elig}/\Omega)) \cap \hat{V}_{out}(F(G_{elig}/\Omega)) = \emptyset$ and so the duals can be adjusted without ambiguity.

As mentioned in Section 1.1, it is difficult to achieve the primal blocking conditions efficiently in CONGEST and PRAM due to long augmenting paths and large blossoms. Fortunately, with the weight modifiers Δw introduced from [8], now we are allowed to remove some matched edges and free vertices from G_{elig}/Ω, which enables a trick of *retrospective eligibility modification*: after *some* set of augmenting paths Ψ is returned, we modify G_{elig} such that Ψ satisfies Property 3.2.

To remove a matched edge e from G_{elig}, we simply add δ_i to $\Delta w(e)$ and so e becomes ineligible. To remove a free vertex f, we

```
**Initialize** $M \leftarrow \emptyset, \Omega \leftarrow \emptyset, \delta_0 \leftarrow \epsilon' W, \tau = W/2 - \delta_0/2$,
$y(u) \leftarrow \tau$ for all $u \in V$, $z(B) \leftarrow 0$ for all $B \in \mathcal{V}_{odd}$, and $\Delta w(e) \leftarrow 0$ for $e \in E$.

**Execute** scales $i = 0, 1, \ldots, L = \log_2 W$ and return the matching $M$.

Scale $i$:
    – Repeat the following until $\tau = W/2^{i+2} - \delta_i/2$ if $i \in [0, L)$, or until it reaches 0 if $i = L$.
      (1) **Augmentation and Blossom Shrinking:**
          – Let $\lambda = \epsilon'/(12(L+1))$. Invoke Approx_Primal($G_{elig}, M, \Omega, \lambda$) to obtain:
              (a) The set of edge-disjoint augmenting path $\Psi$.
              (b) The set of free vertices $F'$ and the set of matched edges $M'$ that are to be removed.
              (c) The set of new blossoms $\Omega'$.
          – Set $M \leftarrow M \oplus \bigcup_{P \in \Psi} P$.
          – Set $\Omega \leftarrow \Omega \cup \Omega'$
          – For each $e \in M'$, set $\Delta w(e) + \delta_i$.
          – For each $f \in F'$, set $y(u) \leftarrow y(u) + \delta_i$ for $u \in \hat{f}^{\Omega}$ and $z(B) \leftarrow z(B) - 2\delta_i$ for the root blossom $B \ni f$ if it exists. (Note that $z(B)$ can be
            as small as $-\delta_i$ after this step, but it will become non-negative after the dual adjustment).
      (2) **Dual Adjustment:**
          – $\tau \leftarrow \tau - \delta_i/2$
          – $y(u) \leftarrow y(u) - \delta_i/2$, if $u \in \hat{V}_{out}(F(G_{elig}/\Omega))$
          – $y(u) \leftarrow y(u) + \delta_i/2$, if $u \in \hat{V}_{in}(F(G_{elig}/\Omega))$
          – $z(B) \leftarrow z(B) + \delta_i$, if $B \in \Omega$ is a root blossom with $B \subseteq \hat{V}_{out}(F(G_{elig}/\Omega))$
          – $z(B) \leftarrow z(B) - \delta_i$, if $B \in \Omega$ is a root blossom with $B \subseteq \hat{V}_{in}(F(G_{elig}/\Omega))$
      (3) **Blossom Dissolution:**
          – Some root blossoms may have zero $z$-values after the dual adjustment step. Dissolve them by removing them from $\Omega$ until there are no such
            root blossoms. Update $G_{elig}$ with the new $\Omega$.
    – If $i \in [0, L)$, set $\delta_{i+1} \leftarrow \delta_i/2$, $\tau \leftarrow \tau + \delta_{i+1}$ and $y(u) \leftarrow y(u) + \delta_{i+1}$ for every $u \in V$.
```

Figure 2: The modified scaling framework.

add δ_i to the y-values of vertices in $\hat{f}^{\Omega}$ and decrease $z(B)$ by $2\delta_i$ where B is the root blossom containing f. By doing so, the vertex f^{Ω} is isolated from all the other vertices in G_{elig}/Ω since all the edges incident to f^{Ω} become ineligible (note that all these edges must be unmatched). Additionally, all the internal edges inside $\hat{f}^{\Omega}$ will have their yz-values unchanged. Note that the reason that we increase the y-values by δ_i instead of $\delta_i/2$ is that we need to synchronize the parity of the y-values (as a multiple of $\delta_i/2$) as a technicality required for the analysis.

We present the details of the entire scaling algorithm in Figure 2. The **augmentation and blossom shrinking step** is the step that adjusts the primal variables M (and also Ω) and removes some matched edges and free vertices to achieve the primal blocking conditions. It uses procedure Approx_Primal, which we describe in Section 5, that runs in poly$(1/\epsilon, \log n)$ rounds and returns a set of matched edges M' and free vertices F' of sizes $O((\epsilon/\log n) \cdot |M|)$ as well as a set of augmenting paths Ψ and a set of blossoms Ω' such that the primal blocking conditions hold in $(G_{elig} - F' - M' - \Psi)/(\Omega \cup \Omega')$. Assuming such a procedure exists, in the full version [27, Appendix B] we prove that the algorithm runs in poly$(1/\epsilon, \log n)$ rounds and outputs a $(1-\epsilon)$-mwm.

Implementation in CONGEST model: In the CONGEST model, all quantities $M, y(u), y(B), \Omega$, and $\Delta w(e)$ shall be stored and accessed locally. We present one straightforward implementation in [27, Appendix A]. We remark that there is no need to store τ as a variable since the number of iterations per scale can be precomputed at the beginning of the algorithm.

Implementation in PRAM and semi-streaming models: We can simulate the CONGEST implementation mentioned above in the CREW PRAM as well as in the semi-streaming model. A detailed discussion is in the full version [27].

4 THE PARALLEL DEPTH-FIRST SEARCH

Fischer, Mitrovic, and Uitto [16] give a deterministic algorithm for $(1-\epsilon)$-approximate mcm in the semi-streaming model as well as in other models such as CONGEST and the massive parallel computation (MPC) model. The core of their algorithm is the procedure Alg-Phase, which searches for an almost maximal set of (short) augmenting paths. In particular, Alg-Phase runs a parallel DFS from every free vertex and returns a set of augmenting paths Ψ and two small-sized sets of vertices V' and V_A such that there exists no augmenting paths of length $O(1/\epsilon)$ on $G - \Psi - V' - V_A$. The DFS originating from a free vertex α defines a search *structure*, denoted as S_α. For the efficiency purpose, the algorithm imposes several restrictions to the DFS on these structures which are parametrized by a *pass limit* $\tau_{\max}$, a *size limit* LIMIT, and a *depth limit* $\ell_{\max}$. We provide an overview of the ALG-PHASE algorithm of [16] in [27, Appendix B].

Unfortunately, directly running ALG-PHASE on G for an almost maximal set of augmenting paths without considering the blossoms would break the scaling framework. For example, the framework does not allow the search to return two disjoint augmenting paths that pass through the same blossom. We now describe the modified ALG-PHASE that works for the contracted graph G/Ω.

VERTEX-WEIGHTED-ALG-PHASE **on the Contracted Graph** G/Ω: Our goal of the modified ALG-PHASE is clear: all we need to do is to come up with an almost maximal set Ψ^Ω of short augmenting paths on G/Ω. After Ψ^Ω is found, the algorithm recovers Ψ, the actual corresponding augmenting paths on G. Moreover, the algorithm returns two small sets of vertices V' and V_A so that no short augmenting path can be found in $(G-\Psi-V'-V_A)/\Omega$.

Observe that the lengths of the paths in Ψ on G could be much longer than the paths in Ψ^Ω on G/Ω, due to the size of blossoms in Ω. This observation motivates us to consider a *vertex-weighted* version of ALG-PHASE. When computing lengths to an augmenting path on G/Ω, each contracted root blossom now has a weight corresponding to the number of matched edges inside the blossom. Now we define the *matching length* and *matching distance* in a contracted graph.

Definition 4.1. Given $u \in G/\Omega$, define $\|u\| = |\hat{u}|$ to be the number of vertices represented by u in the original graph G. Given a set of vertices S, define $\|S\| = \sum_{u\in S}\|u\|$.

Definition 4.2. Let M be a matching on G and Ω be a set of full blossoms with respect to M on G. Define $\tilde{M} := M/\Omega$ to be the set of corresponding matched edges of M on G/Ω. Given an alternating path $P = (u_1, \ldots, u_k)$ in G/Ω, define the *matching length* of P, $\|P\|_M = |P \cap \tilde{M}| + \sum_{i=1}^{k}(\|u_i\|-1)/2$. For any matched edge $e = uv \in \tilde{M}$ we define $\|e\|_M = (\|u\|+\|v\|)/2$ which corresponds to the total number of matched edges in the blossoms $\hat{u}$ and $\hat{v}$ as well as the edge e itself.

In the DFS algorithm searching for augmenting paths, a search process may visit a matched edge e in both directions. We distinguish these two situations by giving an orientation to e, denoting them as *matched arcs* $\vec{e}$ and $\overleftarrow{e}$. Definition 4.3 defines the matching distances in G/Ω:

Definition 4.3. Given a subgraph $H \subseteq G/\Omega$, a set of free vertices F, a matching M, and a matched arc $\vec{e}$, the matching distance to $\vec{e}$, $d_{H,M}(F,\vec{e})$, is defined to be the shortest matching length of all alternating paths in H that start from a free vertex F and end at $\vec{e}$. When the first parameter is omitted, $d_{H,M}(\vec{e})$ is the shortest matching length among all alternating paths in H that start from any free vertex in H and end at $\vec{e}$.

Throughout this paper, if an alternating path starts with a free vertex u_0 but ends at a non-free vertex, we conveniently denote this alternating path by $(u_0, \vec{e}_1, \vec{e}_2, \ldots, \vec{e}_t)$, where u_0 is the starting free vertex and $e_1, e_2, \ldots, e_t$ is the sequence of the matched edges along the path. Also for convenience we define $\|\vec{e}\|_M = \|e\|_M$ for each matched arc $\vec{e}$.

Let λ be a parameter[4]. Similar to the ALG-PHASE algorithm, our VERTEX-WEIGHTED-ALG-PHASE returns a collection of disjoint augmenting paths $\mathcal{P}$ where each augmenting path has a matching length at most $O(\text{poly}(1/\lambda))$; two sets of vertices to be removed V' and V_A with their total weight bounded by $\|V'\| = \text{poly}(1/\lambda)|\mathcal{P}| + \text{poly}(\lambda)|M|$ and $\|V_A\| = O(\lambda|M|)$; and the collection of search structures $\mathcal{S}$ where each search structure $S_\alpha \in \mathcal{S}$ has weight $\|S_\alpha\| = O(\text{poly}(1/\lambda))$. We summarize the vertex-weighted FMU algorithm below:

LEMMA 4.4. *Let λ be a parameter. Let G be the network with weak diameter $\text{poly}(1/\lambda)$ and M be the current matching. Let Ω be a laminar family of vertex subsets (e.g., the current collection of blossoms) such that each set $B \in \Omega$ contains at most $C_{\max} = O(1/\lambda^7)$ vertices. Define the DFS parameters $\tau_{\max} := 1/\lambda^4$, LIMIT $:= 1/\lambda^2$, and $\ell_{\max} := 1/\lambda$. Then, there exists a* CONGEST *algorithm VERTEX-WEIGHTED-ALG-PHASE such that in $\text{poly}(1/\lambda)$ rounds, returns $(\mathcal{P}, V', V_A, \mathcal{S})$ that satisfies the following:*

1. *$\|S_\alpha\| \le C_{\max}$ for each structure $S_\alpha \in \mathcal{S}$, where $C_{\max} := \tau_{\max} \cdot (\ell_{\max}+1) \cdot \text{LIMIT}$.*
2. *$\|V'\| \le C_{\max} \cdot (\ell_{\max}+1) \cdot (2|\mathcal{P}| + \lambda^{32}\tau_{\max}|M|)$.*
3. *$\|V_A\| \le h(\lambda) \cdot (2\ell_{\max}) \cdot |M|$, where $h(\lambda) := \frac{4+2/\lambda}{\lambda\cdot\tau_{\max}} + \frac{2}{\text{LIMIT}}$.*
4. *No augmenting path P with $\|P\|_M \le \ell_{\max}$ exists in $(G/\Omega) \setminus (V' \cup V_A)$.*
5. *For each matched arc $\vec{e}$, if $d_{(G/\Omega)\setminus(V'\cup V_A),M}(\vec{e}) \le \ell_{\max}$, then there exists a $S_\alpha \in \mathcal{S}$ such that $\vec{e}$ belongs to S_α.*

Furthermore, VERTEX-WEIGHTED-ALG-PHASE can be simulated in the CREW PRAM *model in $\text{poly}(1/\lambda)$ time with $O(m)$ processors.*

We defer the proof of Lemma 4.4 to the full version [27]. Specifically, we show that it is possible to apply a VERTEX-WEIGHTED-ALG-PHASE algorithm on G/Ω that can be simulated on the underlying network G with an additional $O(C_{\max}^2)$ factor in the round complexity. Interestingly, the VERTEX-WEIGHTED-ALG-PHASE itself is implemented via a black-box reduction back to the unweighted ALG-PHASE procedure of [16].

5 AUGMENTATION AND BLOSSOM SHRINKING

The main goal of this section is to prove the following theorem:

THEOREM 5.1. *Let λ be a parameter. Given a graph G, a matching M, a collection of active blossoms Ω where each active blossom $B \in \Omega$ has size at most $C_{\max} = O(1/\lambda^7)$ vertices. There exists a $\text{poly}(1/\lambda, \log n)$-time algorithm in the* CONGEST *model that identifies the following:*

1. *A set Ψ of vertex-disjoint augmenting paths with matching lengths at most $\text{poly}(1/\lambda)$.*
2. *A set of new blossoms Ω' in G/Ω, where the size of each blossom is at most $C_{\max}$.*
3. *A set of $O(\lambda \cdot |M|)$ matched edges M' and free vertices F'.*

[4]For the purpose of fitting this subroutine into the scaling framework shown in Figure 2 and not to be confused with the already-defined parameter ϵ, we introduce the parameter λ for the error ratio.

Let $\tilde{G}$ be the contracted graph $\tilde{G} := (G - \Psi - M' - F')/(\Omega \cup \Omega')$ after new blossoms are found and $\tilde{F} := F(\tilde{G})$ be the remaining free vertices. Then, the algorithm also obtains:

(4) The sets $\hat{V}_{in}(\tilde{F})$ and $\hat{V}_{out}(\tilde{F})$.

These objects are marked locally in the network (see [27, Appendix A]). Moreover, $V_{out}(\tilde{F}) \cap V_{in}(\tilde{F}) = \emptyset$.

Notice that the last statement of Theorem 5.1 implies that no blossoms can be detected in $\tilde{G}$ from a free vertex, and thus there is no augmenting path from $\tilde{F}$ on $\tilde{G}$. I.e., $(G_{elig} - M' - F')/(\Omega \cup \Omega')$ meets the primal blocking conditions (Property 3.2) after augmenting all the paths in Ψ.

Algorithm 1 Approx_Primal(G, M, Ω, λ)

```
Require: Unweighted graph $G$ with weak diameter $O(\frac{\log^3 n}{\epsilon})$, a matching $M$, a collection of blossoms $\Omega$, and $\lambda$ (recall from Figure 2 we set $\lambda = \Theta(\frac{\epsilon'}{\log W})$).
Ensure: Collection of augmenting paths $\Psi$, a set of blossoms $\Omega'$, a set of matched edges $M'$ and a set of free vertices $F'$. These objects as well as $\hat{V}_{in}$ and $\hat{V}_{out}$ are represented locally.
 1: Compute $|M|$.
    ▼ Step 1: Repeatedly search for augmenting paths.
 2: repeat
 3:     $(\mathcal{P}, V', V_A, \mathcal{S}) \leftarrow$ Vertex-Weighted-Alg-Phase$(G/\Omega, M, \lambda)$
 4:     $\Psi \leftarrow \mathcal{P} \cup \Psi$.
 5:     $G \leftarrow G \setminus \mathcal{P}$.
 6: until $|\mathcal{P}| \le \lambda \cdot |M|/(C_{\max}(\ell_{\max}+1))$
    ▼ Step 2: Remove matched edges and free vertices returned by the last call to Alg-Phase.
 7: Set $M' \leftarrow M(V') \cup M(V_A)$ and $F' \leftarrow F(V') \cup F(V_A)$.
    ▼ Step 3: Detect new blossoms.
 8: for each $S_\alpha \in \mathcal{S}$ do
 9:     Detect all the (possibly nested) blossoms in $S_\alpha$ and add them to $\Omega'$.
    ▼ Step 4: Remove some matched edges and free vertices such that all long augmenting paths disappear.
10: Define $\tilde{G} := (G - \Psi - M' - F')/(\Omega \cup \Omega')$, $\tilde{F} = F(\tilde{G})$, and $\tilde{M} := M(\tilde{G})$.
11: Use a Bellman-Ford style procedure to compute distance labels $\ell(\vec{e}) \in \{d_{\tilde{G},\tilde{M}}(\tilde{F}, \vec{e}), \infty\}$ for each matched arc $\vec{e} \in \tilde{M}$.
12: Let $E_i, F_i \leftarrow \emptyset$ for all $i \in 1, 2, \ldots, \lfloor \ell_{\max}/2 \rfloor$.
13: for each matched arc $\vec{e} \in \tilde{M}$ such that $\ell(\vec{e}) \neq \infty$ do
14:     Add the corresponding matched edge $\hat{e} \in M$ to $E_i$ for all $i \in [\ell(\vec{e}) - \|e\|_M, \ell(\vec{e})] \cap [1, \ell_{\max}/2]$.
15: for each free vertex $f \in \tilde{F}$ and for all $i \in [1, (\|f\| - 1)/2]$ do
16:     Add the corresponding free vertex $\dot{f} \in F$ to $F_i$.
17: Let $i^* = \arg\min_{1 \le i \le \ell_{\max}/2}\{|E_i| + |F_i|\}$.
18: Add all matched edges in $E_{i^*}$ to $M'$ and all free vertices in $F_{i^*}$ to $F'$.
19: return $\Psi, \Omega', M'$, and $F'$.
```

To prove Theorem 5.1, we propose the main algorithm Algorithm 1. The algorithm consists of four steps. In the first step (Line 2 to Line 6), the algorithm repeatedly invokes the Vertex-Weighted-Alg-Phase procedure, obtains a collection $\mathcal{P}$ of (short) augmenting paths, and temporarily removes the paths from the graph. The loop ends once the number of newly found augmenting paths becomes no more than $\lambda \cdot |M|/C_{\max}$. The algorithm then utilizes the output $(\mathcal{P}, V', V_A, \mathcal{S})$ from the last execution of Vertex-Weighted-Alg-Phase in the subsequent steps.

In step two (Line 7) the algorithm removes edges in $M(V') \cup M(V_A)$ and free vertices in $F(V') \cup F(V_A)$. This ensures that no short augmenting paths can be found from the remaining free vertices.

In the third step (Line 8 to Line 9), the algorithm detects and contracts Ω' — the set of all blossoms within any part of $\mathcal{S}$. Notice that there could still be an edge e connecting two outer vertices in $V_{out}(\tilde{G})$ after the contraction of the blossoms from Ω', leading to an undetected augmenting path. However, in this case, at least one endpoint of e must be far enough from any remaining free vertex, so after the fourth step, such an *outer-outer edge* no longer belongs to any augmenting path.

The fourth step (Line 11 to Line 18) of the algorithm assembles the collection of matched edges $\{E_i\}_{i=1,2,\ldots,\lfloor \ell_{\max}/2 \rfloor}$ and free vertices $\{F_i\}_{i=1,2,\ldots,\lfloor \ell_{\max}/2 \rfloor}$. Each pair of sets (E_i, F_i) has the property that after removing all matched edges in E_i and free vertices in F_i, there will be no more far-away outer-outer edges (and thus no augmenting path). Therefore, the algorithm chooses an index i^* with the smallest $|E_{i^*}| + |F_{i^*}|$ and then removes all matched edges in E_{i^*} and free vertices in F_{i^*}. Intuitively, any long enough alternating paths starting from a free vertex will be intercepted at the matching distance i^* by E_{i^*} and F_{i^*}.

Let $\tilde{G}$ be the current contracted graph $(G - \Psi - M' - F')/(\Omega \cup \Omega')$ after removing a set of augmenting paths Ψ, a set of matched edges M' and a set of free vertices F'. To form the collection of matched edges $\{E_i\}$ and free vertices $\{F_i\}$, the algorithm runs a Bellman-Ford style procedure that computes distance labels to each matched arc $\ell(\vec{e})$. The goal is to obtain $\ell(\vec{e}) = d_{\tilde{G},\tilde{M}}(\tilde{F}, \vec{e})$ whenever this matching distance is no more than $\ell_{\max} + \|e\|_M$, and $\ell(\vec{e}) = \infty$ otherwise. We note that the labels can be computed efficiently because the (weighted) vicinity of the free vertices, after the blossom contractions, is now bipartite.

For each matched arc $\vec{e}$ with a computed matching length $\ell(\vec{e}) = d_{\tilde{G},\tilde{M}}(\tilde{F}, \vec{e}) \le \ell_{\max} + \|e\|_M$, we add the corresponding matched edge $\hat{e} \in M$ to E_i for all integers $i \in [\ell(\vec{e}) - \|e\|_M, \ell(\vec{e})] \cap [1, \ell_{\max}/2]$. In addition, for each free vertex $f \in \tilde{F}$, its corresponding free vertex $\dot{f} \in F$ is added to F_i for all $i \in [1, (\|f\| - 1)/2]$. Finally, $i^* = \arg\min_i\{|E_i| + |F_i|\}$ can be computed and then E_{i^*} and F_{i^*} are removed from G.

The correctness proof and the analysis to Algorithm 1 are presented in the full version [27].

ACKNOWLEDGMENTS

The authors are supported by NSF CCF-2008422. The authors would like to thank Aaron Bernstein and Aditi Dudeja for their invaluable discussions on solving $(1-\epsilon)$-MWM in the semi-streaming model, and, in particular, for pointing out the reduction of [22] to us. The authors also thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] Mohamad Ahmadi, Fabian Kuhn, and Rotem Oshman. 2018. Distributed Approximate Maximum Matching in the CONGEST Model. In *Proc. 32nd International Symposium on Distributed Computing (DISC)*. 6:1–6:17.

[2] Kook Jin Ahn and Sudipto Guha. 2011. Laminar Families and Metric Embeddings: Non-bipartite Maximum Matching Problem in the Semi-Streaming Model. *CoRR* abs/1104.4058 (2011). arXiv:1104.4058 http://arxiv.org/abs/1104.4058

[3] Kook Jin Ahn and Sudipto Guha. 2013. Linear programming in the semi-streaming model with application to the maximum matching problem. *Inf. Comput.* 222

(2013), 59–79. https://doi.org/10.1016/j.ic.2012.10.006

[4] N. Alon, L. Babai, and A. Itai. 1986. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *J. Algor.* 7 (1986), 567–583.

[5] Reuven Bar-Yehuda, Keren Censor-Hillel, Mohsen Ghaffari, and Gregory Schwartzman. 2017. Distributed Approximation of Maximum Independent Set and Maximum Matching. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC).* 165–174.

[6] Aaron Bernstein and Aditi Dudeja. 2023. Private Communication.

[7] N. Blum. 1990. A New Approach to Maximum Matching in General Graphs. In *Proceedings 17th Int'l Colloq. on Automata, Languages, and Programming (ICALP).* 586–597.

[8] Yi-Jun Chang and Hsin-Hao Su. 2022. Narrowing the LOCAL-CONGEST Gaps in Sparse Networks via Expander Decompositions. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC).* to appear.

[9] D. Drake and S. Hougardy. 2003. A Simple Approximation Algorithm for the Weighted Matching Problem. *Info. Proc. Lett.* 85 (2003), 211–213.

[10] R. Duan and S. Pettie. 2010. Connectivity Oracles for Failure Prone Graphs. In *Proceedings 42nd ACM Symposium on Theory of Computing.* 465–474.

[11] R. Duan and S. Pettie. 2014. Linear-Time Approximation for Maximum Weight Matching. *J. ACM* 61, 1 (2014), 1.

[12] J. Edmonds. 1965. Maximum matching and a polyhedron with 0, 1-vertices. *J. Res. Nat. Bur. Standards Sect. B* 69B (1965), 125–130.

[13] Salwa Faour, Marc Fuchs, and Fabian Kuhn. 2021. Distributed CONGEST Approximation of Weighted Vertex Covers and Matchings. In *25th International Conference on Principles of Distributed Systems (OPODIS) (LIPIcs, Vol. 217).* 17:1–17:20.

[14] Manuela Fischer. 2018. Improved deterministic distributed matching via rounding. *Distributed Computing* (2018), 1–13.

[15] Manuela Fischer, Mohsen Ghaffari, and Fabian Kuhn. 2017. Deterministic Distributed Edge-Coloring via Hypergraph Maximal Matching. In *Proc. 58th IEEE Symposium on Foundations of Computer Science (FOCS).* 180–191.

[16] Manuela Fischer, Slobodan Mitrovic, and Jara Uitto. 2022. Deterministic $(1+\epsilon)$-approximate maximum matching with poly$(1/\epsilon)$ passes in the semi-streaming model and beyond. In *54th Annual ACM Symposium on Theory of Computing (STOC).* 248–260. Arxiv full version: *CoRR*, abs/2106.04179v5, 2022.

[17] H. N. Gabow and R. E. Tarjan. 1991. Faster scaling algorithms for general graph-matching problems. *J. ACM* 38, 4 (1991), 815–853.

[18] Buddhima Gamlath, Sagar Kale, Slobodan Mitrovic, and Ola Svensson. 2019. Weighted Matchings via Unweighted Augmentations. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)*, Peter Robinson and Faith Ellen (Eds.). ACM, 491–500.

[19] Mohsen Ghaffari and Fabian Kuhn. 2013. Distributed Minimum Cut Approximation. In *Distributed Computing - 27th International Symposium, DISC 2013, Jerusalem, Israel, October 14-18, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8205)*, Yehuda Afek (Ed.). Springer, 1–15. https://doi.org/10.1007/978-3-642-41527-2_1

[20] Mohsen Ghaffari, Fabian Kuhn, Yannic Maus, and Jara Uitto. 2018. Deterministic Distributed Edge-coloring with Fewer Colors. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing* (Los Angeles, CA, USA) *(STOC 2018).* ACM, New York, NY, USA, 418–430. https://doi.org/10.1145/3188745.3188906

[21] Andrew V. Goldberg, Serge A. Plotkin, and Pravin M. Vaidya. 1993. Sublinear-Time Parallel Algorithms for Matching and Related Problems. *J. Algor.* 14, 2 (1993), 180–213.

[22] Manoj Gupta and Richard Peng. 2013. Fully Dynamic $(1+\epsilon)$-Approximate Matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA.* IEEE Computer Society, 548–557. https://doi.org/10.1109/FOCS.2013.65

[23] Michał Hańćkowiak, Michał Karoński, and Alessandro Panconesi. 2001. On the distributed complexity of computing maximal matchings. *SIAM Journal on Discrete Mathematics* 15, 1 (2001), 41–57.

[24] S. Hanke and S. Hougardy. 2010. *New Approximation Algorithms for the Weighted Matching Problem.* Research Report No. 101010. Research Institute for Discrete Mathematics, University of Bonn.

[25] David G. Harris. 2019. Distributed approximation algorithms for maximum matching in graphs and hypergraphs. In *Proc. 60th IEEE Symposium on Foundations of Computer Science (FOCS).* 700–724.

[26] Stefan Hougardy and Doratha E. Drake Vinkemeier. 2006. Approximating weighted matchings in parallel. *Inf. Process. Lett.* 99, 3 (2006), 119–123. https://doi.org/10.1016/j.ipl.2006.03.005

[27] Shang-En Huang and Hsin-Hao Su. 2022. $(1-\epsilon)$-Approximate Maximum Weighted Matching in poly$(1/\epsilon$, log n) Time in the Distributed and Parallel Settings. *CoRR* abs/2212.14425 (2022). https://doi.org/10.48550/arXiv.2212.14425 arXiv:2212.14425

[28] A. Israeli and A. Itai. 1986. A fast and simple randomized parallel algorithm for maximal matching. *Info. Proc. Lett.* 22, 2 (1986), 77–80.

[29] F. Kuhn, T. Moscibroda, and R. Wattenhofer. 2016. Local Computation: Lower and Upper Bounds. *J. ACM* 63, 2 (2016), 17:1–17:44.

[30] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. 2015. Improved Distributed Approximate Matching. *J. ACM* 62, 5, Article 38 (2015), 17 pages.

[31] Zvi Lotker, Boaz Patt-Shamir, and Adi Rosén. 2009. Distributed Approximate Matching. *SIAM J. Comput.* 39, 2 (2009), 445–460.

[32] M. Luby. 1986. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.* 15, 4 (1986), 1036–1053.

[33] S. Micali and V. V. Vazirani. 1980. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *Proceedings 21st IEEE Symposium on Foundations of Computer Science (FOCS).* 17–27.

[34] Tim Nieberg. 2008. Local, Distributed Weighted Matching on General and Wireless Topologies. In *Proc. 5th Int'l Workshop on Foundations of Mobile Computing (DIALM-POMC) (DIALM-POMC '08).* 87–92.

[35] S. Pettie and P. Sanders. 2004. A simpler linear time $2/3 - \epsilon$ Approximation to Maximum Weight Matching. *Info. Proc. Lett.* 91, 6 (2004), 271–276.

[36] Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal Pandurangan, David Peleg, and Roger Wattenhofer. 2012. Distributed Verification and Hardness of Distributed Approximation. *SIAM J. Comput.* 41, 5 (2012), 1235–1265.

[37] V. V. Vazirani. 1994. A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{V}E)$ General Graph Maximum Matching Algorithm. *Combinatorica* 14, 1 (1994), 71–109.

[38] Mirjam Wattenhofer and Roger Wattenhofer. 2004. Distributed Weighted Matching. In *Proc. 18th International Symposium on Distributed Computing (DISC).* 335–348.