Faster MPC Algorithms for Approximate Allocation and Matching in Uniformly Sparse Graphs

ANONYMOUS AUTHOR(S)*

We study the allocation problem in the Massively Parallel Computation (MPC) model. This problem is a special case of b-matching in which the input is a bipartite graph with capacities greater than 1 in only one of the bipartitions. We give a $(1 + \epsilon)$ approximate algorithm for the problem, which runs in $\tilde{O}(\sqrt{\log \lambda})$ MPC rounds, using sublinear space per machine and $\tilde{O}(\lambda n)$ total space, where λ is the arboricity of the input graph. Our result is obtained by providing a new analysis of a LOCAL algorithm by Agrawal, Zadimoghaddam, and Mirrokni [ICML 2018], which improves its round complexity from $O(\log n)$ to $O(\log \lambda)$. Prior to our work, no $o(\log n)$ round algorithm for constant-approximate allocation was known in either LOCAL or sublinear space MPC models for graphs with low arboricity.

As a side result, we obtain the first sublinear-space MPC algorithm for approximate matching which runs in $o(\log \log n)$ rounds for graphs with $\lambda = O(\text{poly} \log n)$ and does not require the knowledge of λ.

CCS Concepts: • **Computing methodologies → Distributed algorithms**; • **Mathematics of computing → Matchings and factors**.

Additional Key Words and Phrases: distributed algorithms, approximate maximum matching, Massively Parallel Computation

1 INTRODUCTION

Given a bipartite graph $G = (L \cup R, E)$ with integer capacities $C : R \rightarrow \mathbb{N}_{\geq 1}$, the allocation problem aims to find a set of edges $M \subseteq E$ such that a vertex in L is incident to at most 1 edge and a vertex $v \in R$ is incident to at most C_v edges in M. Particularly, the allocation problem is a generalization of the maximum matching problem in bipartite graphs. This problem has been extensively studied in theoretical computer science, especially in the context of online ads and server-client resource allocation. Moreover, the allocation problem is used as a subroutine in other problems. In particular, it was used to obtain the state-of-the-art algorithm for load balancing [Ahmadian et al. 2021].

Motivated by the increase in the size of real-world graphs over the last two decades, there has been a significant interest in designing highly scalable graph algorithms, with the Massively Parallel Computation (MPC) model emerging as a theoretical abstraction of popular frameworks for large-scale computation. The complexity of MPC algorithms is primarily measured by the number of synchronous rounds of communication it takes to solve the problem of interest; the fewer rounds, the better. A particularly active and rich line of research is in designing round-efficient MPC approaches for solving the approximate maximum matching problem [Assadi et al. 2019; Behnezhad et al. 2019a,b; Czumaj et al. 2018; Ghaffari et al. 2018, 2020, 2022; Ghaffari and Uitto 2019; Lattanzi et al. 2011]. We contribute to this line of research by providing a new MPC algorithm for the allocation problem, which is a generalization of the matching problem.

1.1 Our contributions

Our first contribution is a new analysis of the LOCAL fractional allocation algorithm of [Agrawal et al. 2018]. We show that the same algorithm, in fact, converges to a $2 + \epsilon$ approximate solution within $O_\epsilon(\log \lambda)$ rounds.

Theorem 1. *There is a deterministic* LOCAL *algorithm that computes a* $2+\epsilon$ *approximate fractional allocation in* $O_\epsilon(\log \lambda)$ *rounds.*

Leveraging Theorem 1, we give a new algorithm for the allocation problem in the Massively Parallel Computation (MPC) model. The MPC model is a theoretical abstraction of large-scale frameworks, such as MapReduce, Flume, Hadoop, and Spark, which was introduced in a series of papers [Dean and Ghemawat 2004, 2008; Goodrich et al. 2011; Karloff et al. 2010]. An instance of this model has N machines, each equipped with S words of space. The computation on these machines proceeds in synchronous rounds. Originally, the input data is partitioned across the machine arbitrarily. During a round, a machine performs computation on the data it has in its memory. At the end of a round, the machines simultaneously exchange messages. Each machine can send messages to any other machine as long as the total number of words sent and received by a machine in one round does not exceed S.

For solving a problem on n-vertex graphs, the known MPC algorithms focus on one of the following regimes: the superlinear in which $S = n^{1+\gamma}$; the near-linear for which $S = n \cdot \text{poly}\log n$; and, the sublinear for which $S = n^{\gamma}$, where $\gamma > 0$ is an arbitrary constant.

To obtain our MPC algorithm, we show how to sparsify the computation graph for the allocation problem, which enables us to simulate $\Omega(\sqrt{\log \lambda})$ LOCAL in $O(\log\log\lambda)$ MPC rounds. The algorithm we design computes a fractional allocation. In Section 5, we outline that by using standard techniques, this $(1+\epsilon)$-approximate fractional can be transformed into a $\Theta(1)$-approximate integral solution. Finally, applying the recent framework of [Ghaffari et al. 2022] for b-matching, we can improve our approximation factor to $1 + \epsilon$.

Interestingly, the analysis of Theorem 1 also enables us to check for termination in MPC without knowing λ, resulting in the following claim.

Theorem 2. *There is an $O_\epsilon(\sqrt{\log \lambda} \cdot \log\log\lambda)$-round MPC algorithm using n^{α} local memory (for any constant $\alpha \in (0,1)$) and $\tilde{O}(\lambda n)$ total memory that with high probability computes a $1+\epsilon$ approximate solution for allocation. Moreover, our algorithm does not require knowledge of λ.*

The best previously known algorithm for this problem follows by adapting a LOCAL algorithm by [Agrawal et al. 2018] to MPC. This adaptation yields an $O(\log n)$ MPC round approach for the $(1+\epsilon)$-approximate allocation problem.

The allocation problem is defined for bipartite graphs. Nevertheless, it can be applied to computing approximate maximum matching in general graphs by losing a factor of 2 in the approximation: given a general graph $G = (V, E)$, the vertices of V are uniformly at random partitioned between L and R; invoke the allocation algorithm on this bipartite graph. Via this reduction and after rounding, Theorem 2 yields a $\Theta(1)$-approximate maximum matching in general graphs. Finally, by known techniques [Fischer et al. 2022; McGregor 2005] for improving the approximation of a maximum matching, our result implies the following claim.

Corollary 3. *There is an $O_\epsilon(\sqrt{\log \lambda} \cdot \log\log\lambda)$-round MPC algorithm using n^{α} local memory (for any constant $\alpha \in (0,1)$) and $\tilde{O}(\lambda n)$ total memory that with high probability computes a $1+\epsilon$ approximate maximum matching. Moreover, our algorithm does not require knowledge of λ.*

Corollary 3 improves on the state-of-art ([Ghaffari et al. 2020], [Behnezhad et al. 2019a]) by an additional $\log\log n$ term, thereby removing the dependence on n entirely. We note that [Ghaffari et al. 2020] solve the problem of *maximal matching*, which is harder in the LOCAL model.

1.2 Background and State of the Art

The allocation and b-matching problem in MPC. [Agrawal et al. 2018] show that the fractional $(1+\epsilon)$-approximate allocation problem can be solved in $O\left(\frac{\log n}{\epsilon^2}\right)$ rounds of distributed computation. Since their approach sends only poly $\log n$-size messages over an edge in each round, the algorithm readily translates to MPC with the same round complexity. In related work, [Ahmadian et al. 2021] developed a framework for approximate load balancing that uses an algorithm for the allocation problem as a subroutine.

Several groups studied the complexity of computing b-matching in MPC, which is a generalization of the allocation problem. The most recent work shows how to find a $(1+\epsilon)$-approximate maximum b-matching in $O_\epsilon(\log\log n)$ MPC rounds [Ghaffari et al. 2022] in the near-linear memory regime. That work's approach yields $O(\log n)$ round complexity for the sublinear memory regime. In the blackboard distributed computation, [Liu et al. 2023] study $(1+\epsilon)$-approximate b-matching in bipartite graphs aiming to reduce the dependence on $1/\epsilon$. They obtain $O_\epsilon(\log n)$ round complexity. To the best of our knowledge, no algorithm is known for $\Theta(1)$-approximate maximum b-matching that runs in $o(\log n)$ MPC rounds in the sublinear memory regime, and it remains an open question whether such an algorithm exists. Our work on the allocation problem can be seen as the first step towards answering that question in the affirmative.

Matching in MPC. [Lattanzi et al. 2011] showed that a maximal matching can be found in $O(1)$ in the super-linear regime. In a more challenging, near-linear memory regime, a line of work [Assadi et al. 2019; Behnezhad et al. 2019b; Czumaj et al. 2018; Ghaffari et al. 2018, 2022] resulted in a $O(\log\log\bar{\Delta})$ MPC round complexity for computing maximal matching, where $\bar{\Delta}$ the average degree. In the sublinear memory regime, it is known how to compute a maximal matching in $\tilde{O}(\sqrt{\log\Delta})$ MPC rounds [Ghaffari and Uitto 2019].

Several authors focused on obtaining faster algorithms for graphs with low arboricity. In particular, [Behnezhad et al. 2019a] show how to compute a maximal matching in $O(\log^2\log n + \sqrt{\log\lambda}\log\log\lambda)$ many rounds. That result was improved by [Ghaffari et al. 2020] to $O(\log\log n + \sqrt{\log\lambda}\log\log\lambda)$ round complexity. We point out that the algorithm uses $O(\log\log n)$ many rounds, even on trees. If the value of λ is known, then the degree reduction designed in [Czygrinow et al. 2009] can be used to obtain a $\tilde{O}(\log\lambda)$ MPC round algorithm for constant-factor approximate matching.

MPC lower-bounds. [Czumaj et al. 2021] and [Ghaffari et al. 2019] showed conditional lower bounds for *component-stable* MPC algorithms that lift LOCAL lower bounds into the MPC setting for a large class of graph problems including $\Omega(1)$-approximate maximum matching. Using the LOCAL lower bound of [Kuhn et al. 2006], they show that every component stable MPC algorithm requires $\Omega(\log\log n)$ rounds, provided the connectivity conjecture holds. We note that our algorithms that succeed with high probability are **not** component stable.

1.3 Preliminaries and notation

Definition 4 (Arboricity). *The arboricity of a graph G is the minimum number of forests into which it can be decomposed into, and we denote it by $\lambda(G)$ or simply λ when G is clear in context.*

Definition 5 (Allocation). *Given a bipartite graph G with bipartition $L\dot{\bigcup}R$ and a weight function $C : R \longrightarrow \mathbb{N}$, the allocation problem asks for a subset of edges $M \subseteq E(G)$ of maximum size such that (i) every $u \in L$ has at most one incident edge in M and (ii) every $v \in R$ has at most C_v incident edges in M.*

We note that the Allocation problem is a special case of the b-matching problem for bipartite graphs wherein all vertices on one side of G have b-value as 1.

Notation. In this paper we deal with bipartite graphs G with the bipartition $V(G) = L \cup R$. We use $\mathsf{N}_{G,v}$ to refer to the neighborhood of v in the graph G. We shall drop G when it is clear in context.

Algorithm 1 A LOCAL algorithm for finding a fractional matching

```
   Input: $G = (L \cup R, E)$, capacity constraints $\{\mathsf{C}_v\}_{v \in R}$, number of rounds $\tau \geq 1$, approximation
   parameter $\epsilon > 0$.
   Output: a fractional matching $\{\mathsf{x}'_{u,v}\}_{(u,v) \in E}$
   Initialization: Set $\beta_v \leftarrow 1$ for every $v \in R$
1: for rounds $r = 1, 2, \ldots \tau$ do
2:     for each vertex $u \in L$:
           Set $\mathsf{x}_{u,v} = \frac{\beta_v}{\sum_{v' \in \mathsf{N}_u} \beta_{v'}}$ for each $v \in \mathsf{N}_u$
3:     for each vertex $v \in R$:
           Set $\mathsf{alloc}_v \leftarrow \sum_{u \in \mathsf{N}_v} \mathsf{x}_{u,v}$
4:     for each vertex $v \in R$, update $\beta_v$ as follows,
           $$\beta_v \leftarrow \begin{cases} \beta_v(1+\epsilon) & \text{if } \mathsf{alloc}_v \leq \mathsf{C}_v/(1+\epsilon) \\ \beta_v/(1+\epsilon) & \text{if } \mathsf{alloc}_v \geq \mathsf{C}_v(1+\epsilon) \\ \beta_v & \text{otherwise} \end{cases}$$
5: for each vertex $v \in R$ do
6:     for all $u \in \mathsf{N}_v$, set $\mathsf{x}'_{u,v} \leftarrow \begin{cases} \frac{\mathsf{C}_v}{\mathsf{alloc}_v} \mathsf{x}_{u,v} & \text{if } \mathsf{alloc}_v > \mathsf{C}_v \\ \mathsf{x}_{u,v} & \text{otherwise} \end{cases}$
```

2 OVERVIEW OF TECHNIQUES

2.1 Review of the algorithm by [Agrawal et al. 2018]

We first briefly overview the main idea of the *proportional allocation algorithm* by [Agrawal et al. 2018], which is shown in Algorithm 1. **For computing approximate maximum matching, one sets $C_v = 1$.** The allocation problem is more general and works for arbitrary nonnegative values of C_v.

The input to Algorithm 1 is a bipartite graph with a known bipartition (L, R), capacity constraints $\{\mathsf{C}_v\}_{v \in R}$, and two parameters $\tau \geq 1$ and $\epsilon > 0$. The goal is to find a fractional matching satisfying all capacity constraints. For each vertex $v \in R$, the algorithm maintains a *priority value* β_v, initially set to 1. Each edge (u, v) is assigned a fractional value $x_{u,v}$. The algorithm performs τ rounds of computation to update the fractional values. (See lines 1-4.) In each round, each vertex $u \in L$ calculates the fractional value $\mathsf{x}_{u,v}$ assigned to each vertex $v \in \mathsf{N}_u$; the value is computed as the ratio between β_v and the sum of β values over all vertices in N_v. We remark that the fractional values computed in this step may not form a fractional matching because the sum of all fractions allocated to a vertex $v \in R$ may exceed C_v. After this step, each vertex $v \in R$ examines $\mathsf{alloc}_v = \sum_{u \in \mathsf{N}_v} \mathsf{x}_{u,v}$ to update its priority. (See line 4.) The update is based on the following rules: if v is *under-allocated* by a factor of $(1 + \epsilon)$, meaning $\mathsf{alloc}_v > \mathsf{C}_v/(1 + \epsilon)$, then β_v is increased by a factor of $(1 + \epsilon)$. Symmetrically, if v is *over-allocated* by a factor of $(1 + \epsilon)$, β_v is decreased. If neither of the above two cases holds, β_v stays unchanged this round. After τ rounds of computation, lines 5-6 transform $\{\mathsf{x}_{u,v}\}_{(u,v) \in E}$ to a fractional matching by scaling the values assigned to each vertex $v \in R$.

Agrawal et al. showed that the algorithm finds a $(1 + O(\epsilon))$-approximate matching if $\tau \geq O(\frac{\log(|R|/\epsilon)}{\epsilon^2})$. Their proof is based on two crucial observations. Consider the state after the end of the τ rounds. Partition R into $2\tau + 1$ *level sets* $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_{2\tau}$, where $\mathcal{L}_i = \{v \in R \mid \beta_v = (1+\epsilon)^{(i-\tau)}\}$.

The first observation is that each vertex $v \in R$ with $v \notin \mathcal{L}_{2\tau}$ has filled up a $(1 - O(\epsilon))$ fraction of its capacity; in addition, each vertex $v \in R$ with $v \notin \mathcal{L}_0$ is over-allocated by at most a factor of

$1 + O(\epsilon)$, which implies that lines 5-6 only scale $\mathsf{x}_{u,v}$ by a $\frac{1}{1+O(\epsilon)}$ factor. This observation provides a lower bound on the output fractional matching.

The second observation is that given any vertex subset $S = \bigcup_{j \le \ell} \mathcal{L}_j$ for some $\ell \le 2\tau$, the cardinality of the optimal fractional matching is at most the total capacity of S plus the size of the neighborhood of $R \setminus S$. Agrawal et al. proved the approximation factor by showing that the lower bound in the first observation almost matches one of the upper bounds in the second observation.

2.2 Our approach

To obtain our result, a key finding is that Algorithm 1 after $\log_{1+\epsilon}(4\lambda/\epsilon) + 1$ rounds outputs a $(2 + O(\epsilon))$-approximate matching. By observations discussed above, the size of the optimal solution is at most the total capacity of $\mathcal{L}_0, \mathcal{L}_1, \dots, \mathcal{L}_{2\tau-1}$ plus the number of neighbors of $\mathcal{L}_{2\tau}$. Furthermore, all vertices in $\mathcal{L}_0, \mathcal{L}_1, \dots, \mathcal{L}_{2\tau-1}$ almost fill up their capacity. Hence, to obtain a $(2 + O(\epsilon))$-approximation, it suffices to choose a large enough τ so that $|\mathsf{N}(\mathcal{L}_{2\tau})|$ is upper-bounded by the weight of the output matching. We observe that this holds if almost all fractional values from $\mathsf{N}(\mathcal{L}_{2\tau})$ are allocated to vertices not in $\mathcal{L}_0$, as these vertices have $1 + O(\epsilon)$ over-allocation. To obtain such a guarantee, we choose $\tau \approx \log_{1+\epsilon}(\lambda/\epsilon)$, so that each edge between $\mathsf{N}(\mathcal{L}_{2\tau})$ and $\mathcal{L}_0$ is assigned a fractional value of at most $\frac{\epsilon}{\lambda}$. Then, we use the property of bounded arboricity to upper bound the number of edges between $\mathsf{N}(\mathcal{L}_{2\tau})$ and $\mathcal{L}_0$, which yields our result.

The finding above provides a $O(\log \lambda)$-round LOCAL algorithm to compute a constant-approximate maximum fractional matching on a bipartite graph with a known bipartition (L, R) and known upper bound on the arboricity. Two technicalities need to be addressed to use this result for finding a constant-approximate integral matching on general graphs or even on bipartite graphs when L and R are not given on the input.

First, when the bipartition is unknown, or the input graph is not bipartite, we use a standard technique in which each vertex randomly decides whether it is in L or R (independently and with uniform probabilities). Then, all edges connecting two vertices in the same set are removed. In expectation, the resulting bipartite graph has a maximum matching size of at least half of the maximum matching size of the original graph. Thus, we can apply Algorithm 1 on the new graph to find a constant approximation of the maximum fractional matching of the original graph. This approach works for general graphs.

Second, the output of Algorithm 1 may not be an integral matching. In Section 5, we show that by using standard techniques it can be rounded to an integral one while losing only a constant factor in approximation.

2.2.1 Improved round complexity in MPC. The round complexity of Algorithm 1 can further be improved by using the graph sparsification technique [Ghaffari and Uitto 2019] in a nontrivial way.

The general idea is to split the execution of R rounds of a LOCAL algorithm into R/B phases of B rounds each and simulate each B rounds within a phase in $o(B)$ MPC rounds. Observe that in order to simulate B LOCAL rounds in MPC, for each vertex v, it suffices to collect v's B-hop neighborhood on a single MPC machine. Subsequently, the entire phase consisting of B LOCAL rounds can be simulated without additional communication. Collecting subgraphs can be done efficiently in $O(\log B)$ rounds using the graph exponentiation technique [Ghaffari and Uitto 2019; Lenzen and Wattenhofer 2010], i.e., by doubling the radius every round. This approach yields a $O(R/B \cdot \log B)$ round MPC algorithm, *assuming* that the B-hop neighborhood of each vertex fits into the memory of a single machine. However, since vertices can have large degrees, the size of a B-hop neighborhood may be larger than the available memory per machine (even for $B = 1$), which makes a direct implementation of this approach infeasible.

To circumvent this, we redesign our LOCAL algorithm such that no communication occurs along many edges during B rounds. These edges are then ignored when collecting the B-hop neighborhoods, enabling our algorithm to leverage the aforementioned graph exponentiation technique. Our redesign of Algorithm 1 stems from two observations:

(1) A value β_v changes by a factor of at most $1+\epsilon$ in a single round.
(2) Estimating β_v is enough for Algorithm 1 to be correct. In particular, a simple modification of the update rule by replacing $1+\epsilon$ to $1+k\epsilon$ for some bounded constant k would still yield a $O(1)$-approximation. We describe the modified algorithm and analyze it in Appendix A.

Based on Observations 1 and 2, we arrive at the following idea: instead of computing β_v by aggregating over the entire neighborhood of v in each round (as in Algorithm 1), sample a subset of the neighborhood and estimate the desired aggregations by extrapolating from the samples. If we have enough samples so that β is approximated to within a constant, then by Observation 2, our algorithm is correct.

We outline how these two observations are incorporated into our MPC algorithm. Suppose we wish to simulate rounds $r+1, \dots r+B$ of Algorithm 3. Assume that we have previously simulated the first r rounds, and so every vertex v knows its β_v after r rounds. For each vertex v, let $\mathcal{L}_v^s(i)$ denote the set of neighbors w of v such that $\beta_w \in [(1+\epsilon)^{i-1}, (1+\epsilon)^i)$ at the end of round s.

The update rules in Algorithm 1 require the sum of β values over the neighbors of v. To approximate these sums, motivated by Observation 1, our algorithm uniformly at random samples a subset of $\mathcal{L}_v^r(i)$ for every i and every v. Since within B rounds of the LOCAL algorithm, the β value of a vertex can change by $(1+\epsilon)^B$ at most, sampling $O((1+\epsilon)^{O(B)} \cdot \operatorname{poly}\log n)$ vertices from $\mathcal{L}_v^r(i)$ allows to approximate the total sum of the β values of the vertices in $\mathcal{L}_v^r(i)$ in each of the next B rounds. To simulate each round s (for $s \in [r, r+B]$), we use *different* independent samples from $\mathcal{L}_v^r(i)$. The poly log factor ensures that we have enough samples to use, as there are at most $B < \log n$ rounds. We emphasize here that our analysis is based on showing concentration for our estimates of the sum of β values within $\mathcal{L}_v^r(i)$, as opposed to within $\mathcal{L}_v^q(j)$ for $q = r+1 \dots r+B$, over the next B rounds.

We make this intuition formal in Lemma 10, by proving that $O((1+\epsilon)^{2B} \log n)$ samples from each set $\mathcal{L}_v^r(i)$ guarantees a $1+O(\epsilon)$ approximation, leading us to Algorithm 2.

Note on memory requirement. The size of the subgraph around each vertex is $2^{O_\epsilon(B^2)}$. Every vertex v needs to participate in the round executions; therefore, we require $n \cdot 2^{O_\epsilon(B^2)}$ total memory. The maximum possible value of B is thus $O_\epsilon(\alpha\sqrt{\log n})$, in which case the total required memory is $\tilde{O}(n^{1+\alpha} + m)$. Setting $B = O_\epsilon(\sqrt{\log \lambda})$, we get the total required memory of $\tilde{O}(\lambda n)$.

2.2.2 *Removing the assumption on known arboricity in MPC.*

Our algorithm depends on the knowledge of λ in two ways:

(1) The condition in Line 1 of Line 2 assumes the knowledge of τ.
(2) In the graph exponentiation approach, we need the value of $\sqrt{\log \lambda}$ to know how many rounds to simulate.

In order to remove these dependencies, we first observe that our algorithm can detect whether sufficiently many rounds have been run without knowing λ. Specifically, we show that in $O(\log \lambda)$ rounds, one of the following conditions must hold: either the size of $|\mathsf{N}(\mathcal{L}_{2\tau})|$ becomes smaller than $|\mathcal{L}_0|$, or a $\frac{1}{1+O(\epsilon)}$ fraction of $|\mathsf{N}(\mathcal{L}_{2\tau})|$ is allocated to vertices not in $\mathcal{L}_0$. Moreover, we prove that if one of the conditions holds, and we terminate the algorithm, the output is a $(2+O(\epsilon))$-approximation. We note that both conditions can be tested for in $O(1)$ rounds.

Thanks to the above observation, we can "guess" the value of λ. More precisely, we first run the algorithm assuming that λ is a small constant and check whether the termination condition holds after $O(\log \lambda)$ rounds. If yes, we know that we have obtained a $(2 + O(\epsilon))$-approximate solution. If not, we increase the value of λ and repeat. We choose the value of λ_i which we use in the ith trial, such that $\sqrt{\log \lambda_i} = 2^i$. This ensures that the running time is only a constant factor larger than what we would have spent if we knew λ upfront.

3 APPROXIMATING MAXIMUM FRACTIONAL MATCHING IN LOW ARBORICITY GRAPHS

The objective of this section is to show that Algorithm 1 has an approximation factor of $2 + 10\epsilon$ if $\tau \geq \log_{1+\epsilon}(4\lambda/\epsilon) + 1$. Let $\mathsf{MatchWeight} = \sum_{(u,v)\in E} \mathsf{x}'_{u,v}$ be the weight of the output fractional matching. Note that $\mathsf{MatchWeight} = \sum_{v\in R} \min(\mathsf{C}_v, \mathsf{alloc}_v)$.

Consider the β variables after the end of round τ. The minimum value the priority variables can take after τ rounds is $\beta_{min} = \frac{1}{(1+\epsilon)^\tau}$, and any $v \in R$ can take one of the following $2\tau + 1$ possible priority values:

$$\beta_v \in \{\beta_{min}, (1+\epsilon)\beta_{min}, \dots, (1+\epsilon)^{2\tau}\beta_{min}\}$$

For each $0 \leq j \leq 2\tau$, let $\mathcal{L}_j$ be the set of vertices in R with priority value $(1+\epsilon)^k \beta_{min}$, that is, $\mathcal{L}_j = \{v \mid v \in R \text{ and } \beta_v = (1+\epsilon)^j \beta_{min}\}$. We refer to $\mathcal{L}_j$ sets by *level sets*. We note that some of these sets may be empty. There are two main sources of possible suboptimality in the fractional matching that Algorithm 1 finds:

- Over-allocation: If alloc_v is greater than C_v, an $\mathsf{alloc}_v - \mathsf{C}_v$ amount of fraction allocated to v will not be counted towards the objective.
- Under-allocation: If alloc_v is less than C_v, an extra capacity of $\mathsf{C}_v - \mathsf{alloc}_v$ is left to be exploited for advertiser v.

The following lemma shows that for vertices not in $\mathcal{L}_{2\tau}$, the under-allocation loss is negligible, and for vertices not in $\mathcal{L}_0$, the over-allocation loss is negligible.

Lemma 6. *Assume that we run Algorithm 1 for $\tau \geq 1$ rounds. Then, after all rounds have completed, we have*

(1) for any $v \in \bigcup_{j=0}^{2\tau-1} \mathcal{L}_j$, $\mathsf{alloc}_v \geq \frac{1}{1+3\epsilon}\mathsf{C}_v$, and
(2) for any $v \in \bigcup_{j=1}^{2\tau} \mathcal{L}_j$, $\mathsf{alloc}_v \leq (1+3\epsilon)\mathsf{C}_v$.

Proof. Due to the symmetry of the two claims, we only prove the former. Since v is not in level set $\mathcal{L}_{2\tau}$, there was a time that we did not increase β_v. Let t be the last round where β_v is not increased. At the end of round t, $\mathsf{alloc}_v \geq \frac{1}{1+\epsilon}\mathsf{C}_v$. For $t = \tau$, this completes the proof. Otherwise, we consider the round $t+1 \leq \tau$. We first show that $\mathsf{alloc}_v \geq \frac{1}{1+3\epsilon}\mathsf{C}_v$ at the end of round $t+1$. Recall that $\mathsf{alloc}_v = \sum_{u\in \mathsf{N}_v} \frac{\beta_v}{\sum_{a'\in \mathsf{N}_u} \beta_{a'}}$. If β_v is unchanged at round t, the numerator of each term also remains unchanged. The denominator terms are increased at most by a factor of $(1+\epsilon)$. So in total, alloc_v is not decreased by more than a factor of $(1+\epsilon)$. Hence, $\mathsf{alloc}_v \geq \frac{1}{(1+\epsilon)^2}\mathsf{C}_v \geq \frac{1}{(1+3\epsilon)}\mathsf{C}_v$ at the end of round $t+1$. In the other case, β_v is decreased at round t, so the numerator of each term is also reduced by a factor of $(1+\epsilon)$. In total, alloc_v is decreased by a factor of at most $(1+\epsilon)^2$ at round $t+1$. Note that the reduction of β_v at round t means alloc_v was at least $(1+\epsilon)\mathsf{C}_v$, and therefore at least $\frac{1}{1+\epsilon}$ at round $t+1$. So independent of whether β_v was reduced or not, $\mathsf{alloc}_v \geq \frac{1}{1+3\epsilon}\mathsf{C}_v$ holds at round $t+1$.

By definition of t, β_v is increased in all rounds after t. With a similar argument, we know that alloc_v does not decrease at any of these rounds. So alloc_v remains at least $\frac{1}{1+3\epsilon}\mathsf{C}_v$ till the last round. □

We remark that Lemma 6 is an adaptation of [Agrawal et al. 2018, Lemma 1].

For ease of notation, for a subset S of vertices in R, we use $\mathsf{C}(S) = \sum_{v \in S} \mathsf{C}_v$ to denote the total capacity of S, and use $\mathsf{N}(S) = \bigcup_{v \in S} \mathsf{N}_v$ to denote the neighbor set of S. Lemma 6 implies the two lower bounds on MatchWeight. First, since each vertex in $\bigcup_{j<2\tau} \mathcal{L}_j$ almost fill up their capacities, we have

$$\mathsf{MatchWeight} \geq \tfrac{1}{1+3\epsilon} \textstyle\sum_{j=0}^{2\tau-1} \mathsf{C}(\mathcal{L}_j). \tag{1}$$

Second, since all vertices in $\bigcup_{j>0} \mathcal{L}_j$ has bounded over-allocation, when line 6 of Algorithm 1 was executed, the ratio of $\frac{\mathsf{C}_v}{\mathsf{alloc}_v}$ was at least $\frac{1}{1+3\epsilon}$. Hence, we have

$$\mathsf{MatchWeight} \geq \tfrac{1}{1+3\epsilon} \textstyle\sum_{j=1}^{2\tau} \sum_{v \in \mathcal{L}_j} \mathsf{alloc}_v. \tag{2}$$

Let OPT be the weight of an optimal fractional matching. The following lemma gives an upper bound on OPT.

Lemma 7. *Assume that we run Algorithm 1 for $\tau \geq 1$ rounds. Then, after all rounds have completed, we have*

$$OPT \leq (1+3\epsilon)\mathsf{MatchWeight} + |\mathsf{N}(\mathcal{L}_{2\tau})|. \tag{3}$$

Proof. In [Agrawal et al. 2018, Claim 1], it has been shown that

$$\text{for all } 0 \leq \ell \leq 2\tau, \text{ we have } OPT \leq \textstyle\sum_{j=0}^{\ell} \mathsf{C}(\mathcal{L}_j) + |\mathsf{N}(\bigcup_{j=\ell+1}^{2\tau} \mathcal{L}_j)|.$$

That is, the optimal fractional matching is at most the total capacity of $\bigcup_{j \leq \ell} \mathcal{L}_j$ plus the total number of neighbors of $\bigcup_{j>\ell} \mathcal{L}_j$. By setting $\ell = 2\tau - 1$, we obtain

$$OPT \leq \textstyle\sum_{j=0}^{2\tau-1} \mathsf{C}(\mathcal{L}_j) + |\mathsf{N}(\mathcal{L}_{2\tau})|.$$

By Eq. (1), the term $\sum_{j=0}^{2\tau-1} \mathsf{C}(\mathcal{L}_j)$ can be replaced by $(1+3\epsilon)\mathsf{MatchWeight}$, which yields the desired inequality. □

We are ready to prove the approximation factor of Algorithm 1.

Theorem 8. *Assume that we run Algorithm 1 for $\tau \geq \log_{1+\epsilon}(4\lambda/\epsilon) + 1$ rounds. Then, $OPT \leq (2+10\epsilon)\mathsf{MatchWeight}$.*

Proof. Let N' denote $\mathsf{N}(\mathcal{L}_{2\tau})$. By Eq. (3), if we can show that $|N'| \leq (1+7\epsilon)\mathsf{MatchWeight}$, then $OPT \leq (2+10\epsilon)\mathsf{MatchWeight}$ and thus Algorithm 1 is a $(2+10\epsilon)$-approximation algorithm. To prove that $|N'| \leq (1+7\epsilon)\mathsf{MatchWeight}$, we consider two cases:

Case 1: $|N'| \leq |\mathcal{L}_0|$. By Eq. (1), we have

$$\mathsf{MatchWeight} \geq \frac{1}{1+3\epsilon}\mathsf{C}(\mathcal{L}_0) \geq \frac{1}{1+3\epsilon}|\mathcal{L}_0| \geq \frac{1}{1+3\epsilon}|N'|.$$

Hence, $|N'| \leq (1+7\epsilon)\mathsf{MatchWeight}$ holds for this case.

Case 2: $|N'| > |\mathcal{L}_0|$. We first show that $\sum_{j=1}^{2\tau} \sum_{v \in \mathcal{L}_j} \mathsf{alloc}_v \geq (1 - \frac{\epsilon}{2})|N'|$. Consider a vertex $u \in L$ and a vertex $v \in \mathcal{L}_0$. Since N' is the neighbor set of $\mathcal{L}_{2\tau}$, there is a vertex $v' \in \mathcal{L}_{2\tau}$ that is a neighbor of u. In the end of round τ, $\beta_{v'}$ is larger than β_v by a factor of $(1+\epsilon)^{2\tau}$. Thus, in the beginning of round τ, $\beta_{v'}$ is larger than β_v by at least a factor of $(1+\epsilon)^{2\tau-2}$. Recall that $x_{u,v}$ and $x_{u,v'}$ are calculated proportional to the priority values of v and v' in the beginning of round τ. Thus, $\mathsf{x}_{u,v} \leq \mathsf{x}_{u,v'}/(1+\epsilon)^{2\tau-2}$. Since $\tau = \log_{1+\epsilon}(4\lambda/\epsilon) + 1$, we have $\mathsf{x}_{u,v} \leq \frac{\epsilon}{4\lambda} \cdot \mathsf{x}_{u,v} \leq \frac{\epsilon}{4\lambda}$. That is, every edge (u, v) with $u \in N'$ and $v \in \mathcal{L}_0$ allocates at most an amount of $\frac{\epsilon}{4\lambda}$ to $\mathcal{L}_0$. Consider the subgraph G' of G induced by $N' \cup \mathcal{L}_0$. Since the arboricity of G is bounded by λ and $|N'| > |\mathcal{L}_0|$, the number of edges in G' is at most

$\lambda \cdot (|N'| + |\mathcal{L}_0|) < 2\lambda|N'|$. Therefore, the total allocation from N' to $\mathcal{L}_0$, $\sum_{u\in N'} \sum_{v\in \mathcal{L}_0} \mathsf{x}_{u,v}$, is at most $2\lambda|N'| \cdot \frac{\epsilon}{4\lambda} = \frac{\epsilon}{2}|N'|$. That is, the total allocation from N' to advertisers not in $\mathcal{L}_0$ is at least $(1 - \frac{\epsilon}{2})|N'|$. This implies that $\sum_{j=1}^{2\tau} \sum_{v\in\mathcal{L}_j} \mathsf{alloc}_v \geq (1 - \frac{\epsilon}{2})|N'|$.
By Eq. (2), MatchWeight is at least $\frac{1}{1+3\epsilon} \sum_{j=1}^{2\tau} \sum_{v\in\mathcal{L}_j} \mathsf{alloc}_v$. Therefore, we obtain

$$\mathsf{MatchWeight} \geq \frac{1}{1+3\epsilon} \sum_{j=1}^{2\tau} \sum_{v\in\mathcal{L}_j} \mathsf{alloc}_v \geq \frac{1}{1+3\epsilon}(1-\frac{\epsilon}{2})|N'| \geq \frac{1}{1+7\epsilon}|N'|,$$

where the last inequality holds for $\epsilon \leq 1$. Hence, $|N'| \leq (1+7\epsilon)\mathsf{MatchWeight}$ also holds for this case.

Since we have $|N'| \leq (1+7\epsilon)\mathsf{MatchWeight}$ in both cases, the theorem holds. □

We conclude this section by the following remark. The proof of Theorem 8 also implies that after $\log_{1+\epsilon}(4\lambda/\epsilon) + 1$ rounds, at least one of the following conditions must hold: either $\mathsf{N}(\mathcal{L}_{2\tau}) \leq |\mathcal{L}_0|$, or $\sum_{j=1}^{2\tau} \sum_{v\in\mathcal{L}_j} \mathsf{alloc}_v \geq (1 - \frac{\epsilon}{2})|N'|$; moreover, if one of the conditions hold at the end of the last round, the output is a $(2 + 10\epsilon)$-approximation. This implies a variation of Algorithm 1 that does not assume the knowledge of λ: Instead of running the for-loop in lines 1-4 for $\log_{1+\epsilon}(4\lambda/\epsilon) + 1$ rounds, we terminate the loop when one of the above conditions hold. By the discussion above, the variation is a $(2 + 10\epsilon)$-approximation algorithm that performs $\tau \leq \log_{1+\epsilon}(4\lambda/\epsilon) + 1$ rounds of computation. The new termination condition for the loop can be checked in $O(1)$ MPC rounds. However, it is not clear how to check it in $O(1)$ LOCAL rounds.

4 MPC ALGORITHM

In this section we outline how Algorithm 1 can be efficiently implemented in sublinear MPC within $\tilde{O}(\sqrt{\log\lambda})$ rounds. Towards this goal, we present Algorithm 2, a LOCAL algorithm with small locality volume. In particular, for every vertex $v \in V(G)$ and $\Theta(\sqrt{\log\lambda})$ rounds of Algorithm 2, the output of v is influenced by at most $O(\lambda \cdot \mathrm{poly}(\log n))$ vertices. Mirroring the sparsification technique in [Ghaffari and Uitto 2019], and a reduction for randomized approximate matching onto bipartite graphs, yields us our result.

Theorem 9. *There is an $O_\epsilon(\sqrt{\log\lambda} \cdot \log\log\lambda)$-round MPC algorithm using n^α local memory (for any constant $\alpha \in (0,1)$) and $\tilde{O}(\lambda n)$ total memory that with high probability computes a $1+\epsilon$ approximate maximum matching of a graph with arboricity λ.*

In Algorithm 1, in each round a vertex v communicates with *all* of its neighbors. The locality volume for B rounds is thus, Δ^B in the worst case. The communication is done for exactly two purposes: to determine the value of either (i) $\sum\limits_{v\in\mathsf{N}_u} \beta_u$ for all $u \in L$ or (ii) $\sum\limits_{u\in\mathsf{N}_v} \beta_u^{-1}$ for all $v \in R$. Note that both are neighborhood aggregations and that $1 + O(\epsilon)$ approximations of β_w is sufficient to obtain approximations of these aggregates. Finally at the end of a single round, if these aggregations exceed a certain threshold, the β values are decreased by a factor of $1+\epsilon$. Similarly if they fall below a certain threshold, then the β values are increased by a factor of $1+\epsilon$.

Our main observation is that even if these thresholds are slightly "loose", the algorithm is still correct. We prove this observation in Appendix A. A consequence of this is that estimating the exact aggregations computed by Algorithm 1 in Lines 2 and 3 is sufficient.

To obtain approximations with an efficient number of samples, we make use of the fact that the β values cannot change drastically within a single round. In particular after B rounds the values change by at most $(1+\epsilon)^B$. The following lemma helps in this regard.

Algorithm 2 A fractional matching algorithm for bipartite graphs

```
    Input: $G = (L \cup R, E)$, capacity constraints $\{\mathsf{C}_v\}_{v \in R}$, number of rounds $\tau \geq 1$, approximation parameter $\epsilon > 0$.
    Output: A $1 + \epsilon$ approximate fractional matching of $G$
    Parameters: $B = B_{\epsilon/48}$ as per Eq. (4), and $t = (1+\epsilon)^{2B} \cdot \epsilon^{-5} \cdot \log n$
    Initialization: Set $\beta_v \leftarrow 1$ for every $v \in R$ and $\beta_u = \sum_{v \in \mathsf{N}_u} \beta_v$ for all $u \in L$
1:  for phases $p = 0, 1, \ldots \tau/B - 1$ do
2:      for each $w \in V(G)$:
            Partition $\mathsf{N}_w$ into $4\tau + 1$ groups $\mathcal{L}_{-2\tau}, \ldots \mathcal{L}_{2\tau}$ as follows,
              $$\mathcal{L}_x \leftarrow \left\{u \in \mathsf{N}_w \mid \beta_u \in ((1+\epsilon)^{x-1}, (1+\epsilon)^x]\right\} \text{ for each } x = -2\tau, \cdots, 2\tau$$
3:      for each vertex $w \in V(G)$, integer $r \in [0, B)$, integer $x \in [-2\tau, 2\tau]$:
            $E_{r,v,x} \leftarrow$ Sample $t$ edges from $\mathsf{N}_v \bigcap \mathcal{L}_x$ u.a.r.
        $H_r$ := Graph induced by edges in $\cup E_{r,v,x}$ over all $v, x$.
        $H_{r,v}$ := Ball of radius $B$ around $v$ in $H_r$.
        $\mathsf{N}_{r,v}$ := neighborhood of $v$ in $H_r$.
4:      for rounds $r = pB + 1, pB + 2, \ldots (p+1)B$ do
            for each $u \in L$:
5:              Set $\beta_u \leftarrow \frac{|\mathsf{N}_i|}{|\mathsf{N}_{r,i}|} \sum_{v \in \mathsf{N}_{r,i}} \beta_v$
            for each $v \in R$:
6:              Set $\overline{\mathsf{alloc}}_v \leftarrow \frac{|\mathsf{N}_v|}{|\mathsf{N}_{r,v}|} \sum_{u \in \mathsf{N}_{r,v}} \frac{\beta_v}{\beta_u}$ and update $\beta_v$ as follows,
7:
                $$\beta_v \leftarrow \begin{cases} \beta_v(1+\epsilon) & \text{if } \overline{\mathsf{alloc}}_v \leq \mathsf{C}_v/(1+\epsilon) \\ \beta_v/(1+\epsilon) & \text{if } \overline{\mathsf{alloc}}_v \geq \mathsf{C}_v(1+\epsilon) \\ \beta_v & \text{otherwise} \end{cases}$$
8:      Each $v \in R$ calculates $\mathsf{x}_{u,v} = \min(1, \frac{\mathsf{C}_v}{\mathsf{alloc}_v}) \cdot \sum_{u \in \mathsf{N}_v} \frac{\beta_v}{\beta_u}$, the desired fractional matching.
```

Lemma 10. *Consider any sequence X of n elements $x_1, x_2, \ldots x_n$ with $S_x = \sum_{i=1}^{n} x_i$ and such that each x_i lies in the interval $[V/t, V \cdot t]$ for some positive real V, and let $\epsilon \in (0, 1]$. For $s \geq 20 \cdot t^2 \log n/\epsilon^4$, let $y_1, y_2, \ldots y_s$ be a sequence of uniformly random independent samples from X. Denote the rescaled sum of the samples as $S_y = \frac{n}{s} \cdot \sum_{i=1}^{s} y_i$. Then,*

$$\Pr\left[|S_y - S_x| \leq 4\epsilon S_x\right] \geq 1 - n^{-10} \cdot \log_{1+\epsilon} t$$

Proof. We have $S_x \geq nV/t$. Let us split the interval $[V/t, V \cdot t]$ into groups $\mathcal{L}_i = ((1+\epsilon)^{r-1}, (1+\epsilon)^r]$ for i ranging from $-\lceil \log_{1+\epsilon} t \rceil$ to $\lceil \log_{1+\epsilon} t \rceil$. Let M_i be the set of elements j whose value x_j lies in $\mathcal{L}_i$.

We partition the contribution of the elements x_i into two parts, depending on the size of $|M_i|$. Let $S_x = S_{x,\mathsf{small}} + S_{x,\mathsf{large}}$ where, $S_{x,\mathsf{small}}$ is the sum of all x_j such that j lies in a group M_i of size at most $10n \log n/(s\epsilon^2)$. $S_{x,\mathsf{large}}$ is the sum of the remaining elements.

Case 1: ($S_{x,\mathsf{small}} \leq S_x\epsilon$). In this case, the total contribution of such groups can be bound as follows,

$$S_{x,\mathsf{small}} \leq \frac{20nVt\log n}{s\epsilon^3} = \frac{nV}{t} \cdot \frac{20t^2\log n}{s\epsilon^3} \leq S_x \cdot \epsilon$$

For the leftmost inequality, we assumed that every group has the maximum size $10n\log n/(s\epsilon^2)$ and maximum value. The maximum value is bounded as follows, the value of the largest group is at most Vt. Therefore the total contribution over all groups is at most the infinite geometric sum with first term as $\frac{10nVt\log n}{s\epsilon^2}$ and common ratio $1+\epsilon$, which is bounded by $\frac{20nVt\log n}{s\epsilon^3}$. For the last inequality, we used the assumption that $s \geq 20t^2\log n/\epsilon^4$.

Case 2: ($|S_y - S_{x,\mathsf{large}}| \leq 3\epsilon S_{x,\mathsf{large}}$). In this case, for each x_j contributing to the sum, at least $10n\log n/(s\epsilon^2)$ values close to it exist (i.e., they lie in the same set M_i for some i). For each such group M_i, the expected number of samples y_k chosen from M_i is $10\log n/\epsilon^2$. By Chernoff bounds, the number of samples is off by a factor of $1+2\epsilon$ of the expected value by a probability of at most $2e^{-40\log n/3} \leq n^{-10}$.
The sum of the values of these samples incurs at most another $1+\epsilon$ factor, since the gap between the largest and smallest value in a group is $1+\epsilon$. The overall approximation ratio is $1+3\epsilon$ for this case.

Assuming we lose all the elements in Case 1, we still have at least $(1-\epsilon)S_x$ left in Case 2. In Case 2, we do not deviate from the actual value by more than $3\epsilon S_x$. Overall, S_y deviates from S_x by at most $4\epsilon S_x$. Using the union bound over all intervals in Case 2 gives us the desired probability. □

Guided by Lemma 10, if we want to compress B rounds of Algorithm 1 to MPC (being content with $1+O(\epsilon)$ approximations), then we can plugin $t = (1+\epsilon)^B$ in Lemma 10 and obtain s, the number of samples needed per group. The total degree per vertex is at most $d = 20(1+\epsilon)^{2B} \cdot \log^2 n \cdot \epsilon^{-5}$.

Now the B-radius neighborhood of the relevant vertices around each vertex has volume at most d^B. We choose B so that (i) the neighborhood can be stored within a single machine and (ii) the size is at most λ. (i) ensures that we can simulate B rounds for v in a single machine and (ii) ensures that the total memory is $\tilde{O}(\lambda n + m)$. We require:

$$2B^2\log(1+\epsilon) + 5B\log(\epsilon^{-1}) + 2B\log\log n + 5 \leq \min\{\log\lambda, \alpha\cdot\log n\}$$

Since we are concerned only with asymptotics, we obtain a simple candidate for B by bounding each term on the left by $\min(\log\lambda, \alpha\log n)/4$. Assuming ϵ is a constant, and n is large enough, the first term is dominant. Thus, we can set $B = B_\epsilon$ to guarantee that aggregations are computed within a $1+4\epsilon$ factor.

$$B_\epsilon = \min\left\{\frac{\sqrt{\alpha\log n}}{\sqrt{8\epsilon}}, \frac{\sqrt{\log\lambda}}{\sqrt{8\epsilon}}\right\} \tag{4}$$

For ease of analysis in the correctness proof, we set $B = B_{\epsilon/48}$, giving us the following lemma.

Lemma 11. *Consider Algorithm 2. We define the following parameters,*

$$\overline{\beta_u} = \sum_{u\in\mathsf{N}_u}\beta_u, \quad \mathsf{alloc}_v = \sum_{u\in\mathsf{N}_v}\overline{\beta_u}$$

We have $|\overline{\beta_u} - \beta_u| \leq \epsilon/12\cdot\overline{\beta_u}$ *and* $|\mathsf{alloc}_v - \overline{\mathsf{alloc}_v}| \leq \epsilon/4\cdot\mathsf{alloc}_v$ *with probability at least* $1-n^{-5}$ *over all rounds and vertices.*

Proof. For $\overline{\beta_u}$ we use Lemma 10 with $t = (1+\epsilon)^B$. Observe that B value is set so that the error term in Lemma 10 is $\epsilon/48$. The approximation ratio is $1+\epsilon/12$. Since the choice of B in Eq. (4) is $o(n)$ even when $\epsilon < 1/n$, the probability guarantee is at least $1-n^{-9}$.

For the second expression, we have the same situation. Applying union bound over all vertices and rounds we get a conservative failure probability of n^{-5} overall. For the approximation ratio, the $1 + \epsilon/12$ is cascaded twice, leading to overall $(1 + \epsilon/12)^2 \leq 1 + \epsilon/4$. □

From Lemma 11, we can infer that the B round simulation is identical to a run of the algorithm described in Appendix A with the value of k chosen appropriately. Our round complexity is $\tau \log B/B$ where τ is given in Algorithm 1 and B is as per Eq. (4), which completes the proof of Theorem 9.

We did not cover MPC simulations of certain lines in Algorithm 2 such as the graph exponentiation to compute the subgraphs $H_{r,v}$ for each v and the random sampling from the groups. These can be implemented from standard primitives such as graph exponentiation and sorting, which are by now standard in the MPC literature.

Finally, our algorithms as stated assume knowledge of λ. To remove this assumption, we can make use of the fact that the proof of correctness of Algorithm 1 provides a termination condition. In particular, we require to test if a subset of vertices, say $S \subseteq V(G)$ contains a large total incident matching. Again using known primitives in MPC, we can perform this test in $O(1)$ rounds. We therefore can repeat the algorithm by guessing the value of $\sqrt{\log \lambda}$ and doubling the estimate when the termination test fails.

5 ROUNDING FROM FRACTIONAL TO INTEGRAL

We describe in this section a simple rounding procedure that, in expectation, obtains an $\Theta(1)$-approximate integral allocation from a $4 + \epsilon$-approximate fractional allocation. In the MPC model, we can further obtain high probability of success by running $O(\log n)$ copies of this algorithm in parallel, and choosing the copy that corresponds to the largest rounded matching.

Let M_f be the given fractional allocation. Denote by $\mathrm{wt}(M_f)$ the total sum of the fractional values over all edges in M_f. The algorithm is as follows. First, sample edge e independently of the others with probability $M_f(e)/6$. Next, if a vertex v has the number of edges incident to it greater than its capacity, drop all the sampled edges incident to v. We call such vertex v *heavy*.

Let X_e be a 0/1 random variable equal to 1 if $e = \{u, v\}$ is sampled, and Y_e be a 0/1 random variable equal to 1 if e remains among the sampled edges. Then, we have

$$\mathbb{E}[Y_e] = \Pr[X_e = 1] \cdot \Pr[\text{neither } u \text{ nor } v \text{ is heavy} \mid X_e = 1]. \tag{5}$$

Note that $\Pr[X_e = 1] = \frac{M_f(e)}{6}$. We proceed to analyze $\Pr[u \text{ is heavy} \mid X_e = 1]$. Let E' be the set of edges $e' \neq e$ that are incident to u. Then,

$$\Pr[u \text{ is heavy} \mid X_e = 1] = \Pr\left[\sum_{e' \in E'} X_{e'} > \mathrm{C}_u - 1\right].$$

Consider two cases, where in both cases we show that $\Pr\left[\sum_{e' \in E'} X_{e'} > \mathrm{C}_u - 1\right] \leq \frac{1}{3}$:

(1) $C_u > 1$. In this case, $C_u - 1 \geq \frac{\mathrm{C}_u}{2}$. Therefore,

$$\Pr\left[\sum_{e' \in E'} X_{e'} > \mathrm{C}_u - 1\right] \leq \Pr\left[\sum_{e' \in E'} X_{e'} > \frac{\mathrm{C}_u}{2}\right]$$

By our algorithm, $\mathbb{E}[\sum_{e' \in E'} X_{e'}] \leq \frac{\mathrm{C}_v}{6}$. Hence, by Markov's inequality, it implies

$$\Pr\left[\sum_{e' \in E'} X_{e'} > \mathrm{C}_u - 1\right] \leq \frac{1}{3}.$$

(2) $C_u = 1$. In this case, we have

$$\Pr\left[\sum_{e' \in E'} X_{e'} > C_u - 1\right] = \Pr\left[\text{at least one edge in } E' \text{ is chosen}\right] \le \frac{1}{6},$$

where the last inequality is obtained by applying the union bound.

By the above analysis, we know that $\Pr\left[u \text{ is heavy} \mid X_e = 1\right] \le \frac{1}{3}$. By symmetry, we also have $\Pr\left[v \text{ is heavy} \mid X_e = 1\right] \le \frac{1}{3}$. Plugging this into Eq. (5) yields

$$\mathbb{E}\left[Y_e\right] \ge \frac{M_f(e)}{3} \cdot \left(1 - \frac{1}{3} - \frac{1}{3}\right) = \frac{M_f(e)}{9}.$$

Let M be the matching kept after the rounding procedure. By the linearity of expectation, we have

$$\mathbb{E}\left[|M|\right] = \sum_{e \in E} \mathbb{E}\left[Y_e\right] \ge \mathrm{wt}(M_f)/9.$$

Let $M^\star$ be the maximum allocation matching. Since M_f is a $4+\epsilon$ approximation, we have $\mathrm{wt}(M_f) \ge |M^\star|/5$. Therefore,

$$\mathbb{E}\left[|M|\right] \ge \frac{|M^\star|}{45}.$$

Let α be the probability that $|M| \le |M^\star|/450$. Then, by using that $|M| \le |M^\star|$ deterministically, we have

$$\alpha \frac{|M^\star|}{450} + (1-\alpha)|M^\star| \ge \mathbb{E}\left[|M|\right] \ge \frac{|M^\star|}{45}.$$

A simple calculation leads to $\alpha \le 440/449$. Therefore, with a constant positive probability, $|M|$ is a $\Theta(1)$ approximation.

REFERENCES

Shipra Agrawal, Morteza Zadimoghaddam, and Vahab S. Mirrokni. 2018. Proportional Allocation: Simple, Distributed, and Diverse Matching with High Entropy. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 99–108. http://proceedings.mlr.press/v80/agrawal18b.html

Sara Ahmadian, Allen Liu, Binghui Peng, and Morteza Zadimoghaddam. 2021. Distributed load balancing: A new framework and improved guarantees. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Sepehr Assadi, MohammadHossein Bateni, Aaron Bernstein, Vahab Mirrokni, and Cliff Stein. 2019. Coresets meet EDCS: algorithms for matching and vertex cover on massive graphs. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1616–1635.

Soheil Behnezhad, Sebastian Brandt, Mahsa Derakhshan, Manuela Fischer, MohammadTaghi Hajiaghayi, Richard M. Karp, and Jara Uitto. 2019a. Massively Parallel Computation of Matching and MIS in Sparse Graphs. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (Toronto ON, Canada) *(PODC '19)*. Association for Computing Machinery, New York, NY, USA, 481–490. https://doi.org/10.1145/3293611.3331609

Soheil Behnezhad, Mohammad Taghi Hajiaghayi, and David G Harris. 2019b. Exponentially faster massively parallel maximal matching. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 1637–1649.

Artur Czumaj, Peter Davies, and Merav Parter. 2021. Component Stability in Low-Space Massively Parallel Computation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing* (Virtual Event, Italy) *(PODC'21)*. Association for Computing Machinery, New York, NY, USA, 481–491. https://doi.org/10.1145/3465084.3467903

Artur Czumaj, Jakub Łącki, Aleksander Mądry, Slobodan Mitrović, Krzysztof Onak, and Piotr Sankowski. 2018. Round compression for parallel matching algorithms. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. 471–484.

Andrzej Czygrinow, Michal Hanckowiak, and Edyta Szymanska. 2009. Fast Distributed Approximation Algorithm for the Maximum Matching Problem in Bounded Arboricity Graphs. In *Algorithms and Computation, 20th International Symposium, ISAAC 2009, Honolulu, Hawaii, USA, December 16-18, 2009. Proceedings (Lecture Notes in Computer Science,* *Vol. 5878)*, Yingfei Dong, Ding-Zhu Du, and Oscar H. Ibarra (Eds.). Springer, 668–678. https://doi.org/10.1007/978-3-642-10631-6_68

Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. (2004).

Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

Manuela Fischer, Slobodan Mitrović, and Jara Uitto. 2022. Deterministic $(1+\epsilon)$-approximate maximum matching with poly $(1/\epsilon)$ passes in the semi-streaming model and beyond. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*. 248–260.

Mohsen Ghaffari, Themis Gouleakis, Christian Konrad, Slobodan Mitrović, and Ronitt Rubinfeld. 2018. Improved Massively Parallel Computation Algorithms for MIS, Matching, and Vertex Cover. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing* (Egham, United Kingdom) *(PODC '18)*. Association for Computing Machinery, New York, NY, USA, 129–138. https://doi.org/10.1145/3212734.3212743

Mohsen Ghaffari, Christoph Grunau, and Ce Jin. 2020. Improved MPC Algorithms for MIS, Matching, and Coloring on Trees and Beyond. In *34th International Symposium on Distributed Computing (DISC 2020) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 179)*, Hagit Attiya (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 34:1–34:18. https://doi.org/10.4230/LIPIcs.DISC.2020.34

Mohsen Ghaffari, Christoph Grunau, and Slobodan Mitrović. 2022. Massively Parallel Algorithms for b-Matching. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*. 35–44.

Mohsen Ghaffari, Fabian Kuhn, and Jara Uitto. 2019. Conditional Hardness Results for Massively Parallel Computation from Distributed Lower Bounds. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, David Zuckerman (Ed.). IEEE Computer Society, 1650–1663. https://doi.org/10.1109/FOCS.2019.00097

Mohsen Ghaffari and Jara Uitto. 2019. Sparsifying Distributed Algorithms with Ramifications in Massively Parallel Computation and Centralized Local Computation. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, Timothy M. Chan (Ed.). SIAM, 1636–1653. https://doi.org/10.1137/1.9781611975482.99

Michael T Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, searching, and simulation in the mapreduce framework. In *International Symposium on Algorithms and Computation*. Springer, 374–383.

Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A model of computation for mapreduce. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM, 938–948.

Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. 2006. The price of being near-sighted. In *SODA 2006: 17th ACM-SIAM Symposium on Discrete Algorithms, Miami, Florida, USA*.

Silvio Lattanzi, Benjamin Moseley, Siddharth Suri, and Sergei Vassilvitskii. 2011. Filtering: a method for solving graph problems in mapreduce. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*. 85–94.

Christoph Lenzen and Roger Wattenhofer. 2010. Brief announcement: Exponential speed-up of local algorithms using non-local communication. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*. 295–296.

Quanquan C Liu, Yiduo Ke, and Samir Khuller. 2023. Scalable Auction Algorithms for Bipartite Maximum Matching Problems. *arXiv preprint arXiv:2307.08979* (2023).

Andrew McGregor. 2005. Finding Graph Matchings in Data Streams. In *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3624)*, Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan (Eds.). Springer, 170–181. https://doi.org/10.1007/11538462_15

A CORRECTNESS OF THE MPC ALGORITHM

Algorithm 3 An adapted version of Algorithm 1.

```
    Input: $G = (L \cup R, E)$, capacity constraints $\{C_v\}_{v \in R}$, number of rounds $\tau \geq 1$, approximation parameter $\epsilon \in (0, \frac{1}{4}]$.
    Parameters: positive real numbers $\{k_{v,r}\}_{v \in R, 1 \leq r \leq \tau}$.
    Output: a fractional matching $\{x'_{u,v}\}_{(u,v) \in E}$
    Initialization: Set $\beta_v \leftarrow 1$ for every $v \in R$
1:  for rounds $r = 1, 2, \ldots \tau$ do
2:      for each vertex $u \in L$:
            Set $x_{u,v} = \frac{\beta_v}{\sum_{v' \in N_u} \beta_{v'}}$ for each $v \in N_u$
3:      for each vertex $v \in R$:
            Set $\text{alloc}_v \leftarrow \sum_{u \in N_v} x_{u,v}$
4:      for each vertex $v \in R$, update $\beta_v$ as follows,
            $$\beta_v \leftarrow \begin{cases} \beta_v(1+\epsilon) & \text{if alloc}_v \leq C_v/(1+k_{v,r}\epsilon) \\ \beta_v/(1+\epsilon) & \text{if alloc}_v \geq C_v(1+k_{v,r}\epsilon) \\ \beta_v & \text{otherwise} \end{cases}$$
5:  for each vertex $v \in R$ do
6:      For all $u \in N_v$, set $x'_{u,v} \leftarrow \begin{cases} \frac{C_v}{\text{alloc}_v} x_{u,v} & \text{if alloc}_v > C_v \\ x_{u,v} & \text{otherwise} \end{cases}$
```

To prove the correctness of the MPC algorithm (Algorithm 2), we consider an adaptation of our LOCAL algorithm (Algorithm 1). The adaptation is given in Algorithm 3. Algorithms 1 and 3 only have two differences: First, Algorithm 3 requires an additional set of parameters $\{k_{v,r}\}_{v \in R, 1 \leq r \leq \tau}$. Second, Algorithm 3 applies a different update rule in line 4, in which the thresholds for updating β_v are replaced with $C_v/(1 + k_{v,r}\epsilon)$ and $C_v(1 + k_{v,r}\epsilon)$. This adapted update rule is more flexible because the thresholds can be controled by adjusting the parameters. Note that Algorithm 1 is a special case where all parameters $k_{v,r}$ are fixed as 1.

Consider an input instance $(G, \{C_v\}_{v \in R}, \tau, \epsilon)$ of Algorithm 2. The correctness proof for Algorithm 2 consists of three parts.

- In the first part, we argue that Algorithm 2 is equivalent to Algorithm 3 with a specific set of parameters. More precisely, we show that given a fixed execution of Algorithm 2 (on the input instance $(G, \{C_v\}_{v \in R}, \tau, \epsilon)$), with high probability, there exists a set of parameters $\{k_{v,r}\}_{v \in R, 1 \leq r \leq \tau}$ such that the outputs of Algorithms 2 and 3 are the same; in addition, the constructed parameters satisfy $\frac{1}{4} \leq k_{v,r} \leq 4$ for all $v \in R$ and $1 \leq r \leq \tau$.
- In the second part, we show that Algorithm 3 has an approximation factor of $(2 + (2k + 8)\epsilon)$ if the input and parameters satisfy $\frac{1}{k} \leq k_{v,r} \leq k$ for some small number k, $\epsilon \leq \frac{1}{k}$, and $\tau \geq \log_{1+\epsilon}(4\lambda/\epsilon) + 1$. By combining the results of the first and second parts, we know that Algorithm 2 has an approximation factor of $2 + 16\epsilon$ if $\tau \geq \log_{1+\epsilon}(4\lambda/\epsilon) + 1$ and $\epsilon \leq \frac{1}{4}$.
- In the third part, we show that Algorithm 3 has an approximation factor of $1 + (k + 14)\epsilon$ if the input and parameters satisfy $\frac{1}{k} \leq k_{v,r} \leq k$ for some small number k, $\epsilon \leq \frac{1}{k}$, and $\tau \geq \frac{\log(|R|/\epsilon)}{\epsilon^2}$. By combining the results of the first and third parts, we know that Algorithm 2 has an approximation factor of $1 + 18\epsilon$ for $\tau = O(\frac{\log(|R|/\epsilon)}{\epsilon^2})$ and $\epsilon \leq \frac{1}{4}$.

The three parts are presented in Appendices A.1 to A.3, respectively.

A.1 Equivalence of Algorithms 2 and 3

In the following, we argue that Algorithm 2 is equivalent to Algorithm 3 with a specific set of parameters. Consider an execution of Algorithm 2. For ease of presentation, for a fixed round, we use $\{\overline{\beta}_v\}_{v\in R}$ to denote the priority values maintained by Algorithm 2, and use $\{\beta_v\}_{v\in R}$ to denote the priority values maintained by Algorithm 3. Let $\overline{\mathsf{x}}_{u,v} = \frac{\overline{\beta}_v}{\sum_{v'\in \mathsf{N}_u} \overline{\beta}_{u'}}$ denote the corresponding fractional values in Algorithm 2.

Lemma 12. *Let $(G, \{\mathsf{C}_v\}_{v\in A}, \epsilon, R)$ be an input instance for Algorithms 2 and 3. Consider an execution of Algorithm 2 on the instance. With high probability, there exists a set of parameters $\{k_{v,r}\}_{v\in R, 1\le r\le \tau}$ for Algorithm 3, all in the range $[\frac{1}{4}, 4]$, such that the output of the execution is the same as the output of Algorithm 3.*

Proof. We prove this lemma by constructing a set of parameters $\{k_{v,r}\}_{v\in R, 1\le r\le \tau}$ such that $\beta_v = \overline{\beta}_v$ holds for all $v \in R$ across all rounds. Initially, both algorithms set the priority values as 1. Hence, in beginning of the the first round, $\beta_v = \overline{\beta}_v$ holds for all $v \in R$.

Consider a fixed round r and assume that $\beta_v = \overline{\beta}_v$ holds in the beginning of round r. Let $v \in R$ be a vertex. In the following, we show how to determine the parameter $k_{v,r}$ such that the two algorithms update the priority values in the same way. Since $\beta_v = \overline{\beta}_v$ holds for all $v \in R$, we know that $\mathsf{x}_{u,v} = \overline{\mathsf{x}}_{u,v}$ holds for each edge (u, v) in round r. Therefore, $\mathsf{alloc}_v = \sum_{u\in \mathsf{N}_v} \mathsf{x}_{u,v}$ is equal to $\sum_{u\in \mathsf{N}_v} \overline{\mathsf{x}}_v$. By Lemma 11, the execution of Algorithm 2 satisfies

$$|\overline{\mathsf{alloc}}_v - \sum_{u\in \mathsf{N}_v} \overline{\mathsf{x}}_v| = |\overline{\mathsf{alloc}}_v - \mathsf{alloc}_v| \le \frac{\epsilon}{4}\mathsf{alloc}_v \tag{6}$$

with high probability. Assume that this inequality holds for the given execution. Consider two cases:

- **Case 1:** In round r, $\mathsf{alloc}_v \le 2\mathsf{C}_v$. By Eq. (6), we have
$$|\mathsf{alloc}_v - \overline{\mathsf{alloc}}_v| \le \frac{\epsilon}{4}\mathsf{alloc}_v \le \frac{\epsilon}{2}\epsilon\mathsf{C}_v. \tag{7}$$
There are three subcases:
  - **Case 1.1:** In the execution, $\overline{\beta}_v$ is increased in round r. That is, $\overline{\mathsf{alloc}}_v \le \mathsf{C}_v/(1+\epsilon)$. Our goal is to choose $k_{v,r}$ such that Algorithm 3 also increases β_v. By Eq. (7), we have $\mathsf{alloc}_v \le \overline{\mathsf{alloc}}_v + \frac{\epsilon}{2}\mathsf{C}_v \le (\frac{1}{1+\epsilon} + \frac{\epsilon}{2})\mathsf{C}_v$. Thus, we should choose $k_{v,r}$ such that $\mathsf{alloc}_v \le \frac{1}{(1+k_{v,r}\epsilon)}\mathsf{C}_v$. Since $(\frac{1}{1+\epsilon} + \frac{\epsilon}{2}) \le \frac{1}{1+\frac{1}{4}\epsilon}$ for $\epsilon \le \frac{1}{4}$, we may choose $k_{v,r} = \frac{1}{4}$.
  - **Case 1.2:** In the execution, $\overline{\beta}_v$ is decreased in round r. That is, $\overline{\mathsf{alloc}}_v \ge \mathsf{C}_v(1+\epsilon)$. Our goal is to choose $k_{v,r}$ such that Algorithm 3 also decreases β_v. By Eq. (7), we have $\mathsf{alloc}_v \ge \overline{\mathsf{alloc}}_v - \frac{\epsilon}{2}\mathsf{C}_v \ge (1+\epsilon-\frac{\epsilon}{2})\mathsf{C}_v$. By choosing $k_{v,r} = \frac{1}{2}$, we ensure that $\mathsf{alloc}_v \ge (1+k_{v,r}\epsilon)\mathsf{C}_v$ and thus Algorithm 3 will decrease β_v.
  - **Case 1.3:** In the execution, $\overline{\beta}_v$ is unchanged in round r. That is, $\mathsf{C}_v/(1+\epsilon) < \overline{\mathsf{alloc}}_v < \mathsf{C}_v(1+\epsilon)$. By Eq. (7), we have $(\frac{1}{1+\epsilon} - \frac{\epsilon}{2})\mathsf{C}_v < \mathsf{alloc}_v < (1+\frac{3\epsilon}{2})\mathsf{C}_v$. Using a similar argument, we know that Algorithm 3 will not change β_v if $\frac{1}{1+k_{v,r}\epsilon} \le \frac{1}{1+\epsilon} - \frac{\epsilon}{2}$ and $1 + k_{v,r}\epsilon \ge 1 + \frac{3\epsilon}{2}$. It is not hard to verify that $k_{v,r} = 3$ satisfies these conditions.
- **Case 2:** In round r, $\mathsf{alloc}_v > 2\mathsf{C}_v$. In this case, we choose $k_{v,r} = 1$ so that Algorithm 3 will decrease β_v. By Eq. (7), we have
$$\overline{\mathsf{alloc}}_v \ge (1-\epsilon)\mathsf{alloc}_v > 2(1-\epsilon)\mathsf{C}_v > (1+\epsilon)\mathsf{C}_v,$$

where the last inequality holds because $\epsilon \leq \frac{1}{4}$. Therefore, Algorithm 2 must also have decreased $\overline{\beta}_v$ in round r.

By repeating the above argument for $r = 0, 1, \ldots, R$, we obtain a set of parameters $\{k_{v,r}\}$ such that, in each round, the two algorithms update the priority value of each advertiser in exactly the same way. Hence, the lemma holds. □

A.2 Performance guarantee on low arboricity bipartite graphs

In the following, we show that Algorithm 3 has an approximation factor of $2 + (2k + 8)\epsilon$ if the parameters satisfy $\tau \geq \log_{1+\epsilon}(4\lambda/\epsilon) + 1$, $\frac{1}{k} \leq k_{v,r} \leq k$ holds for all $k_{v,r}$ and for some number k, and $\epsilon \leq \frac{1}{k}$. By Lemma 12, this result implies that Algorithm 2 has an approximation factor of $2 + 16\epsilon$ if $\tau \geq \log_{1+\epsilon}(4\lambda/\epsilon) + 1$ and $\epsilon \leq \frac{1}{4}$. The result is proven by modifying the proof in Section 3.

We first present the approximation factor of Algorithm 3. Let k be a number such that $\frac{1}{k} \leq k_{v,r} \leq k$ holds for all $v \in R$ and $1 \leq r \leq \tau$. As in Section 3, we consider the β variables after the end of round τ and partition R into level sets $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_{2\tau}$, where $\mathcal{L}_i = \{v \mid v \in R \text{ and } \beta_v = (1+\epsilon)^i\}$. We generalize Lemma 6 as follows.

Lemma 13. *Assume that we run Algorithm 3 for $\tau \geq 1$ rounds. Then, after all rounds have completed, we have*

(1) *for any $v \in \bigcup_{j=0}^{2\tau-1} \mathcal{L}_j$, $\mathsf{alloc}_v \geq \frac{1}{1+(k+2)\epsilon}\mathsf{C}_v$, and*
(2) *for any $v \in \bigcup_{j=1}^{2\tau} \mathcal{L}_j$, $\mathsf{alloc}_v \leq (1 + (k+2)\epsilon)\mathsf{C}_v$.*

Proof. Due to the symmetry of the two claims, we only prove the former. Since v is not in level set $\mathcal{L}_{2\tau}$, there was a time that we did not increase β_v. Let t be the last round where β_v is not increased. At the end of round t, $\mathsf{alloc}_v \geq \frac{1}{1+k\epsilon}\mathsf{C}_v$. For $t = \tau$, this completes the proof. Otherwise, we consider the round $t + 1 \leq \tau$. We first show that $\mathsf{alloc}_v \geq \frac{1}{1+(k+2)\epsilon}\mathsf{C}_v$ at the end of round $t + 1$. Recall that $\mathsf{alloc}_v = \sum_{u\in \mathsf{N}_v} \frac{\beta_v}{\sum_{a'\in \mathsf{N}_u} \beta_{a'}}$. If β_v is unchanged at round t, the numerator of each term also remains unchanged. The denominator terms are increased at most by a factor of $(1 + \epsilon)$. So in total, alloc_v is not decreased by more than a factor of $(1 + \epsilon)$. Hence, at the end of round $t + 1$, we have $\mathsf{alloc}_v \geq \frac{1}{(1+k\epsilon)(1+\epsilon)}\mathsf{C}_v \geq \frac{1}{(1+(k+2)\epsilon)}\mathsf{C}_v$, where the last inequality holds because $k\epsilon \leq 1$. In the other case, β_v is decreased at round t, so the numerator of each term is also reduced by a factor of $(1 + \epsilon)$. In total, alloc_v is decreased by a factor of at most $(1 + \epsilon)^2$ at round $t + 1$. Note that the reduction of β_v at round t means alloc_v was at least $(1 + \frac{1}{k}\epsilon)\mathsf{C}_v \geq \mathsf{C}_v$, and therefore at least $\frac{1}{(1+\epsilon)^2} \leq \frac{1}{(1+(k+2)\epsilon)}$ at round $t + 1$. So independent of whether β_v was reduced or not, $\mathsf{alloc}_v \geq \frac{1}{1+(k+2)\epsilon}\mathsf{C}_v$ holds at round $t + 1$.

By definition of t, β_v is increased in all rounds after t. With a similar argument, we know that alloc_v does not decrease at any of these rounds. So alloc_v remains at least $\frac{1}{1+(k+2)\epsilon}\mathsf{C}_v$ till the last round. □

Lemma 13 implies the two lower bounds on MatchWeight. First, since each vertex in $\bigcup_{j<2\tau} \mathcal{L}_j$ almost fill up their capacities, we have

$$\mathsf{MatchWeight} \geq \frac{1}{1+(k+2)\epsilon} \sum_{j=0}^{2\tau-1} \mathsf{C}(\mathcal{L}_j). \tag{8}$$

Second, since all vertices in $\bigcup_{j>0} \mathcal{L}_j$ has bounded over-allocation, when line 6 of Algorithm 3 was executed, the ratio of $\frac{\mathsf{C}_v}{\mathsf{alloc}_v}$ was at least $\frac{1}{1+(k+2)\epsilon}$. Hence, we have

$$\mathsf{MatchWeight} \geq \frac{1}{1+(k+2)\epsilon} \sum_{j=1}^{2\tau} \sum_{v\in \mathcal{L}_j} \mathsf{alloc}_v. \tag{9}$$

In the proof of Lemma 7, we have mentioned the following upper bounds on OPT, which is proven by [Agrawal et al. 2018].

$$OPT \leq \sum_{j=0}^{\ell} \mathsf{C}(\mathcal{L}_j) + |\mathsf{N}(\bigcup_{j=\ell+1}^{2\tau} \mathcal{L}_j)| \text{ for all } 0 \leq \ell \leq 2\tau. \tag{10}$$

We now present a generalization of Lemma 7.

Lemma 14. *Assume that we run Algorithm 3 for $\tau \geq 1$ rounds. Then, after all rounds have completed, we have*

$$OPT \leq (1 + (k+2)\epsilon)\mathsf{MatchWeight} + |\mathsf{N}(\mathcal{L}_{2\tau})|. \tag{11}$$

Proof. By setting $\ell = 2\tau - 1$ in Eq. (10), we obtain $OPT \leq \sum_{j=0}^{2\tau-1} \mathsf{C}(\mathcal{L}_j) + |\mathsf{N}(\mathcal{L}_{2\tau})|$. By Eq. (8), the term $\sum_{j=0}^{2\tau-1} \mathsf{C}(\mathcal{L}_j)$ can be replaced by $(1 + (k+2)\epsilon)\mathsf{MatchWeight}$, which yields the desired inequality. □

We are ready to prove the approximation factor of Algorithm 3.

Theorem 15. *Assume that we run Algorithm 3 for $\tau \geq \log_{1+\epsilon}(4\lambda/\epsilon) + 1$ rounds. Then, $OPT \leq (2 + (2k+8)\epsilon)\mathsf{MatchWeight}$.*

Proof. Let N' denote $\mathsf{N}(\mathcal{L}_{2\tau})$. By Eq. (11), if we can show that $|N'| \leq (1+(k+6)\epsilon)\mathsf{MatchWeight}$, then $OPT \leq (2 + (2k+8)\epsilon)\mathsf{MatchWeight}$. To prove that $|N'| \leq (1 + (k+6)\epsilon)\mathsf{MatchWeight}$, we consider two cases:

Case 1: $|N'| \leq |\mathcal{L}_0|$. By Eq. (8), we have

$$\mathsf{MatchWeight} \geq \frac{1}{1+(k+2)\epsilon}\mathsf{C}(\mathcal{L}_0) \geq \frac{1}{1+(k+2)\epsilon}|\mathcal{L}_0| \geq \frac{1}{1+(k+2)\epsilon}|N'|.$$

Hence, $|N'| \leq (1 + (k+6)\epsilon)\mathsf{MatchWeight}$ holds for this case.

Case 2: $|N'| > |\mathcal{L}_0|$. We first show that $\sum_{j=1}^{2\tau}\sum_{v\in\mathcal{L}_j} \mathsf{alloc}_v \geq (1 - \frac{\epsilon}{2})|N'|$. Consider a vertex $u \in L$ and a vertex $v \in \mathcal{L}_0$. Since N' is the neighbor set of $\mathcal{L}_{2\tau}$, there is a vertex $v' \in \mathcal{L}_{2\tau}$ that is a neighbor of u. In the end of round τ, $\beta_{v'}$ is larger than β_v by a factor of $(1+\epsilon)^{2\tau}$. Thus, in the beginning of round τ, $\beta_{v'}$ is larger than β_v by at least a factor of $(1+\epsilon)^{2\tau-2}$. Recall that $x_{u,v}$ and $x_{u,v'}$ are calculated proportional to the priority values of v and v' in the beginning of round τ. Thus, $\mathsf{x}_{u,v} \leq \mathsf{x}_{u,v'}/(1+\epsilon)^{2\tau-2}$. Since $\tau = \log_{1+\epsilon}(4\lambda/\epsilon) + 1$, we have $\mathsf{x}_{u,v} \leq \frac{\epsilon}{4\lambda} \cdot \mathsf{x}_{u,v} \leq \frac{\epsilon}{4\lambda}$. That is, every edge (u, v) with $u \in N'$ and $v \in \mathcal{L}_0$ allocates at most an amount of $\frac{\epsilon}{4\lambda}$ to $\mathcal{L}_0$. Consider the subgraph G' of G induced by $N' \cup \mathcal{L}_0$. Since the arboricity of G is bounded by λ and $|N'| > |\mathcal{L}_0|$, the number of edges in G' is at most $\lambda \cdot (|N'| + |\mathcal{L}_0|) < 2\lambda|N'|$. Therefore, the total allocation from N' to $\mathcal{L}_0$, $\sum_{u\in N'}\sum_{v\in\mathcal{L}_0} \mathsf{x}_{u,v}$, is at most $2\lambda|N'| \cdot \frac{\epsilon}{4\lambda} = \frac{\epsilon}{2}|N'|$. That is, the total allocation from N' to advertisers not in $\mathcal{L}_0$ is at least $(1 - \frac{\epsilon}{2})|N'|$. This implies that $\sum_{j=1}^{2\tau}\sum_{v\in\mathcal{L}_j} \mathsf{alloc}_v \geq (1 - \frac{\epsilon}{2})|N'|$.

By Eq. (9), $\mathsf{MatchWeight}$ is at least

$$\frac{1}{1+(k+2)\epsilon}\sum_{j=1}^{2\tau}\sum_{v\in\mathcal{L}_j} \mathsf{alloc}_v \geq \frac{1}{1+(k+2)\epsilon}(1 - \frac{\epsilon}{2})|N'| \geq \frac{1}{(1+(k+2)\epsilon)(1+\epsilon)}|N'|.$$

Since $k\epsilon \leq 1$, the last term is at least $\frac{1}{1+(k+6)\epsilon}$. Hence, $|N'| \leq (1 + (k+6)\epsilon)\mathsf{MatchWeight}$ also holds for this case.

Since we have $|N'| \leq (1 + (k+6)\epsilon)\mathsf{MatchWeight}$ in both cases, the theorem holds. □

Lemma 12 and Theorem 15 imply the following.

Theorem 16. *Assume that we execute Algorithm 2 with $\epsilon \leq \frac{1}{4}$ and $\tau \geq \log_{1+\epsilon}(4\lambda/\epsilon) + 1$. Then, with high probability, the algorithm finds a $(2 + 16\epsilon)$-approximate fractional matching.*

A.3 Performance guarantee on bipartite graphs

In the following, we show that Algorithm 3 has an approximation factor of $1+(k+14)\epsilon$ if $\frac{1}{k} \leq k_{v,r} \leq k$ holds for all $k_{v,r}$ and for some number k, $\epsilon \leq \frac{1}{k}$, and $\tau \geq \frac{2\log(2|R|/\epsilon)}{\epsilon^2} + \frac{1}{\epsilon}$. By Lemma 12, this result implies that Algorithm 2 has an approximation factor of $1 + 18\epsilon$ if $\tau \geq \frac{2\log(2|R|/\epsilon)}{\epsilon^2} + \frac{1}{\epsilon}$ and $\epsilon \leq \frac{1}{4}$. The result is proven by modifying a theorem in [Agrawal et al. 2018], which shows that Algorithm 1 has an approximation factor of $(1 + O(\epsilon))$.

We first present lower bounds for MatchWeight. Let g denote $\frac{2\log(2|R|/\epsilon)}{\epsilon} + 1$.

Lemma 17. *For any two indices $1 \leq \ell$ and $\ell + g \leq \ell' \leq 2\tau$, we have*

$$\mathsf{MatchWeight} \geq \left(\frac{1}{1+(k+6)\epsilon}\right)\left(\sum_{j=0}^{\ell} \mathsf{C}(\mathcal{L}_j) + \left|\mathsf{N}\left(\bigcup_{j'=\ell'+1}^{2\tau} \mathcal{L}_{j'}\right)\right|\right) \tag{12}$$

Proof. By Lemma 13, the vertices $\mathcal{L}_0, \mathcal{L}_1, \ldots, \mathcal{L}_\ell$ almost fill up their capacities. Therefore, they contribute at least $\frac{1}{1+(k+2)\epsilon}\sum_{j=0}^{\ell} \mathsf{C}(\mathcal{L}_j)$ which is larger than the first term of the lower bound.

The second term represents all neighbors of vertices in $\mathcal{L}_{\ell'+1}, \ldots, \mathcal{L}_{2\tau}$. We first show that for each vertex $u \in L$ having a neighbor in $\bigcup_{j=\ell'+1}^{2\tau} \mathcal{L}_j$, we have:

$$\sum_{j=\ell+1}^{2\tau} \sum_{v \in \mathcal{L}_j} \mathsf{x}_{u,v} \geq (1 - \frac{\epsilon}{2}), \tag{13}$$

That is, a large fraction of u is assigned to vertices in $\mathcal{L}_{\ell+1}, \ldots, \mathcal{L}_{2\tau}$. Consider vertex $u \in L$ that is a neighbor of $v' \in \mathcal{L}_{j'}$ for some $j' \geq \ell' + 1$. Because ℓ' is at least $\ell + \frac{2\log(2|R|/\epsilon)}{\epsilon} + 1$, we have $\beta_{v'} \geq \frac{2|R|}{\epsilon}\beta_v$ for any $v \in \mathcal{L}_j$ with $j \leq l$. Therefore $\mathsf{x}_{u,v}$ is at most $\frac{\epsilon}{2|R|}$ times $\mathsf{x}_{u,v'}$. Since there are at most $|R|$ vertices like v, we have

$$\sum_{j=0}^{\ell} \sum_{v \in \mathcal{L}_j} \mathsf{x}_{u,v} \leq \sum_{j=0}^{\ell} \sum_{v \in \mathcal{L}_j} \frac{\epsilon}{2|R|} \cdot \mathsf{x}_{i,a'} \leq \frac{\epsilon}{2}\mathsf{x}_{u,v'} \leq \frac{\epsilon}{2}.$$

Hence, Eq. (13) holds, which indicates that each vertex in $\mathsf{N}(\bigcup_{j'=\ell'+1}^{2\tau} \mathcal{L}_{j'})$ contributes at least $1 - \frac{\epsilon}{2}$ to $\sum_{j=\ell+1}^{2\tau} \mathsf{alloc}_v$. Using Lemma 13, at least $\frac{1}{1+(k+2)\epsilon}$ fraction of every such vertex will be counted toward MatchWeight. So in total, we get at least $\frac{1}{1+(k+2)\epsilon} \cdot (1 - \frac{\epsilon}{2}) \geq \frac{1}{1+6\epsilon}$ fraction for each vertex in $\mathsf{N}(\bigcup_{j'=\ell'+1}^{2\tau} \mathcal{L}_{j'})$. This concludes the proof. □

We prove the approximation factor of Algorithm 3 by combining the lower and upper bounds.

Lemma 18. *Algorithm 3 has an approximation factor of $1 + (k + 14)\epsilon$ if the input instance and parameters satisfy all of the following:*

- *for all $v \in R$ and $0 \leq r \leq \tau$, $\frac{1}{k} \leq k_{a,r} \leq k$ for some number k,*
- *$\epsilon \leq \frac{1}{k}$, and*
- *$\tau \geq \frac{2\log(2|R|/\epsilon)}{\epsilon} + 1$.*

Proof. There are two main gaps between the lower bound of Eq. (12) and the upper bound of Eq. (10): the $\frac{1}{1+(k+6)\epsilon}$ factor and the sum $\sum_{j=\ell+1}^{\ell'} \sum_{v \in \mathcal{L}_j} \mathsf{C}_v$. We show that the latter gap is small for some value of ℓ and $\ell' = \ell + g$. Recall that $g = \frac{2\log(2|R|/\epsilon)}{\epsilon} + 1$. Summing the gap over different values of ℓ yields

$$\sum_{\ell=0}^{2\tau-g-1} \sum_{j=\ell+1}^{\ell+g} \mathrm{C}(\mathcal{L}_j) \leq g \sum_{j=1}^{2\tau-1} \mathrm{C}(\mathcal{L}_j).$$

Therefore, there exists an ℓ with $0 \leq \ell \leq 2\tau - g - 1$ such that its associated gap $\sum_{j=\ell+1}^{\ell+g} \mathrm{C}(\mathcal{L}_j)$ is at most $\frac{g}{2\tau-g} \sum_{j=1}^{2\tau-1} \mathrm{C}(\mathcal{L}_j) \leq \epsilon \sum_{j=1}^{2\tau-1} \mathrm{C}(\mathcal{L}_j)$, where the last inequality holds when τ is at least $g/\epsilon = \frac{2\log(2|R|/\epsilon)}{\epsilon^2} + \frac{1}{\epsilon}$ and $\epsilon \leq 1$.

By Eq. (8), MatchWeight is at least $\frac{1}{1+(k+2)\epsilon} \sum_{j=1}^{2\tau} \mathrm{C}(\mathcal{L}_j)$. This means the gap associated for some ℓ is at most $\epsilon \cdot (1 + (k+2)\epsilon)$MatchWeight. Using Lemma 17 and Eq. (10), we have

$$\begin{aligned} \text{MatchWeight} &\geq \frac{1}{1+(k+6)\epsilon}(OPT - \epsilon(1+(k+2)\epsilon)\text{MatchWeight}) \\ &= \frac{1}{1+(k+6)\epsilon} OPT - \epsilon \cdot \frac{1+(k+2)\epsilon}{1+(k+6)\epsilon} \cdot \text{MatchWeight} \qquad (14) \\ &\geq \frac{1}{1+(k+6)\epsilon} OPT - \epsilon\text{MatchWeight}, \end{aligned}$$

and hence MatchWeight $\geq \frac{1}{(1+(k+6)\epsilon)(1+\epsilon)} OPT$, yielding a final approximation factor of at least $1 + (k+14)\epsilon$. □

Lemmas 12 and 18 imply the following.

THEOREM 19. *Assume that we execute Algorithm 2 with $\epsilon \leq \frac{1}{4}$ and $\tau \geq \frac{2\log(2|R|/\epsilon)}{\epsilon^2} + \frac{1}{\epsilon}$. Then, with high probability, the algorithm finds a $(1 + 18\epsilon)$-approximate fractional matching.*

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009