

# Distributed Fractional Packing and Maximum Weighted b-Matching via Tail-Recursive Duality

Christos Koufogiannakis and Neal E. Young\*

Department of Computer Science, University of California, Riverside  
`{ckou,neal}@cs.ucr.edu`

**Abstract.** We present efficient distributed  $\delta$ -approximation algorithms for FRACTIONAL PACKING and MAXIMUM WEIGHTED b-MATCHING in hypergraphs, where  $\delta$  is the maximum number of packing constraints in which a variable appears (for MAXIMUM WEIGHTED b-MATCHING  $\delta$  is the maximum edge degree — for graphs  $\delta = 2$ ). (a) For  $\delta = 2$  the algorithm runs in  $O(\log m)$  rounds in expectation and with high probability. (b) For general  $\delta$ , the algorithm runs in  $O(\log^2 m)$  rounds in expectation and with high probability.

## 1 Background and Results

Given a weight vector  $w \in \mathbb{R}_+^m$ , a coefficient matrix  $A \in \mathbb{R}_+^{n \times m}$  and a vector  $b \in \mathbb{R}_+^n$ , the FRACTIONAL PACKING problem is to compute a vector  $x \in \mathbb{R}_+^m$  to maximize  $\sum_{j=1}^m w_j x_j$  and at the same time meet all the constraints  $\sum_{j=1}^m A_{ij} x_j \leq b_i$  ( $\forall i = 1 \dots n$ ). We use  $\delta$  to denote the maximum number of packing constraints in which a variable appears, that is,  $\delta = \max_j |\{i \mid A_{ij} \neq 0\}|$ . In the centralized setting, FRACTIONAL PACKING can be solved optimally in polynomial time using linear programming. Alternatively, one can use a faster approximation algorithm (i.e. [11]).

MAXIMUM WEIGHTED b-MATCHING on a (hyper)graph is the variant where each  $A_{ij} \in \{0, 1\}$  and the solution  $x$  must take integer values (without loss of generality each vertex capacity is also integer). An instance is defined by a given hypergraph  $H(V, E)$  and  $b \in \mathbb{Z}_+^{|V|}$ ; a solution is given by a vector  $x \in \mathbb{Z}_+^{|E|}$  maximizing  $\sum_{e \in E} w_e x_e$  and meeting all the vertex capacity constraints  $\sum_{e \in E(u)} x_e \leq b_u$  ( $\forall u \in V$ ), where  $E(u)$  is the set of edges incident to vertex  $u$ . For this problem,  $n = |V|$ ,  $m = |E|$  and  $\delta$  is the maximum (hyper)edge degree (for graphs  $\delta = 2$ ).

MAXIMUM WEIGHTED b-MATCHING is a cornerstone optimization problem in graph theory and Computer Science. As a special case it includes the ordinary MAXIMUM WEIGHTED MATCHING problem ( $b_u = 1$  for all  $u \in V$ ). In the centralized setting, MAXIMUM WEIGHTED b-MATCHING on graphs belongs to the “well-solved class of integer linear programs” in the sense that it can be solved in polynomial time [5,6,19]. Moreover, getting a 2-approximate<sup>1</sup> solution for MAXIMUM

---

\* Partially supported by NSF awards CNS-0626912, CCF-0729071.

<sup>1</sup> Since it is a maximization problem it is also referred to as a 1/2-approximation.

WEIGHTED MATCHING is relatively easy, since the obvious greedy algorithm, which selects the heaviest edge that is not conflicting with already selected edges, gives a 2-approximation. For hypergraphs the problem is NP-hard, since it generalizes SET PACKING, one of Karp's 21 NP-complete problems [10].

**Our results.** In this work we present efficient distributed  $\delta$ -approximation algorithms for the above problems. If the input is a MAXIMUM WEIGHTED b-MATCHING instance, the algorithms produce integral solutions. The method we use is of particular interest in the distributed setting, where it is the first primal-dual extension of a non-standard local-ratio technique [13,2].

- For FRACTIONAL PACKING where each variable appears in at most two constraints ( $\delta = 2$ ), we show a distributed 2-approximation algorithm running in  $O(\log m)$  rounds in expectation and with high probability. This is the first 2-approximation algorithm requiring only  $O(\log m)$  rounds. This improves the approximation ratio over the previously best known algorithm [14]. (For a summary of known results see Figure 1.)
- For FRACTIONAL PACKING where each variable appears in at most  $\delta$  constraints, we give a distributed  $\delta$ -approximation algorithm running in  $O(\log^2 m)$  rounds in expectation and with high probability, where  $m$  is the number of variables. For small  $\delta$ , this improves over the best previously known constant factor approximation [14], but the running time is slower by a logarithmic-factor.
- For MAXIMUM WEIGHTED b-MATCHING on graphs we give a distributed 2-approximation algorithm running in  $O(\log n)$  rounds in expectation and with high probability. MAXIMUM WEIGHTED b-MATCHING generalizes the well studied MAXIMUM WEIGHTED MATCHING problem. For a 2-approximation, our algorithm is faster by at least a logarithmic factor than any previous algorithm. Specifically, in  $O(\log n)$  rounds, our algorithm gives the best known approximation ratio. The best previously known algorithms compute a  $(1 + \epsilon)$ -approximation in  $O(\epsilon^{-4} \log^2 n)$  rounds [17] or in  $O(\epsilon^{-2} + \epsilon^{-1} \log(\epsilon^{-1} n) \log n)$  rounds [20]. For a 2-approximation both these algorithms need  $O(\log^2 n)$  rounds.
- For MAXIMUM WEIGHTED b-MATCHING on hypergraphs with maximum hyper-edge degree  $\delta$  we give a distributed  $\delta$ -approximation algorithm running in  $O(\log^2 m)$  rounds in expectation and with high probability, where  $m$  is the number of hyperedges. Our result improves over the best previously known  $O(\delta)$ -approximation ratio by [14], but it is slower by a logarithmic factor.

**Related work for Maximum Weighted Matching.** There are several works considering distributed MAXIMUM WEIGHTED MATCHING on edge-weighted graphs. Uehara and Chen present a constant time  $O(\Delta)$ -approximation algorithm [22], where  $\Delta$  is the maximum vertex degree. Wattenhofer and Wattenhofer improve this result, showing a randomized 5-approximation algorithm taking  $O(\log^2 n)$  rounds [23]. Hoepman shows a deterministic 2-approximation algorithm taking  $O(m)$  rounds [7]. Lotker, Patt-Shamir and Rosén give a randomized  $(4 + \epsilon)$ -approximation algorithm running in  $O(\epsilon^{-1} \log \epsilon^{-1} \log n)$  rounds

problem	approx. ratio	running time	where when
max weighted matching on graphs	$O(\Delta)$	$O(1)$	[22] 2000
	5	$O(\log^2 n)$	[23] 2004
	2	$O(m)$	[7] 2004
	$O(1)(> 2)$	$O(\log n)$	[14] 2006
	$(4 + \varepsilon)$	$O(\varepsilon^{-1} \log \varepsilon^{-1} \log n)$	[18] 2007
	$(2 + \varepsilon)$	$O(\log \varepsilon^{-1} \log n)$	[17] 2008
	$(1 + \varepsilon)$	$O(\varepsilon^{-4} \log^2 n)$	[17] 2008
	$(1 + \varepsilon)$	$O(\varepsilon^{-2} + \varepsilon^{-1} \log(\varepsilon^{-1}n) \log n)$	[20] 2008
	2	$O(\log^2 n)$	[17,20] ( $\varepsilon = 1$ ) 2008
	2	$O(\log n)$	here 2009
fractional packing with $\delta = 2$	$O(1)(> 2)$	$O(\log m)$	[14] 2006
	2	$O(\log m)$	here 2009
max weighted matching on hypergraphs	$O(\delta) > \delta$	$O(\log m)$	[14] 2006
	$\delta$	$O(\log^2 m)$	here 2009
fractional packing with general $\delta$	$O(1) > 12$	$O(\log m)$	[14] 2006
	$\delta$	$O(\log^2 m)$	here 2009

**Fig. 1.** Distributed algorithms for FRACTIONAL PACKING and MAXIMUM WEIGHTED MATCHING

[18]. Lotker, Patt-Shamir and Pettie improve this result to a randomized  $(2 + \varepsilon)$ -approximation algorithm taking  $O(\log \varepsilon^{-1} \log n)$  rounds [17]. Their algorithm uses as a black box any distributed constant-factor approximation algorithm for maximum weighted matching which takes  $O(\log n)$  rounds (i.e. [18]). Moreover, they mention (without details) that there is a distributed  $(1 + \varepsilon)$ -approximation algorithm taking  $O(\varepsilon^{-4} \log^2 n)$  rounds, based on the parallel algorithm by Hougardy and Vinkemeier [8]. Nieberg presents a  $(1 + \varepsilon)$ -approximation algorithm in  $O(\varepsilon^{-2} + \varepsilon^{-1} \log(\varepsilon^{-1}n) \log n)$  rounds [20]. The latter two results give randomized 2-approximation algorithms for MAXIMUM WEIGHTED MATCHING in  $O(\log^2 n)$  rounds.

**Related work for Fractional Packing.** Kuhn, Moscibroda and Wattenhofer show efficient distributed approximation algorithms for FRACTIONAL PACKING [14]. They first show a  $(1 + \varepsilon)$ -approximation algorithm for FRACTIONAL PACKING with logarithmic message size, but the running time depends on the input coefficients. For unbounded message size they show a constant-factor approximation algorithm for FRACTIONAL PACKING which takes  $O(\log m)$  rounds. If an integer solution is desired, then distributed randomized rounding ([15]) can be used. This gives an  $O(\delta)$ -approximation for MAXIMUM WEIGHTED b-MATCHING on (hyper)graphs with high probability in  $O(\log m)$  rounds, where  $\delta$  is the maximum hyperedge degree (for graphs  $\delta = 2$ ). (The hidden constant factor in the big-O notation of the approximation ratio can be relative large compared to a small  $\delta$ , say  $\delta = 2$ ).

**Lower bounds.** The best lower bounds known for distributed packing and matching are given by Kuhn, Moscibroda and Wattenhofer [14]. They prove that to achieve a constant or even a poly-logarithmic approximation ratio for fractional maximum matching, any algorithms requires at least  $\Omega(\sqrt{\log n / \log \log n})$  rounds and  $\Omega(\log \Delta / \log \log \Delta)$ , where  $\Delta$  is the maximum vertex degree.

**Other related work.** For UNWEIGHTED MAXIMUM MATCHING on graphs, Israeli and Itai give a randomized distributed 2-approximation algorithm running in  $O(\log n)$  rounds [9]. Lotker, Patt-Shamir and Pettie improve this result giving a randomized  $(1 + \varepsilon)$ -approximation algorithm taking  $O(\varepsilon^{-3} \log n)$  rounds [17]. Czygrinow, Hańćkowiak, and Szymańska show a deterministic 3/2-approximation algorithm which takes  $O(\log^4 n)$  rounds [4]. A  $(1 + \varepsilon)$ -approximation for MAXIMUM WEIGHTED MATCHING on graphs is in NC [8].

The rest of the paper is organized as follows. In Section 2 we describe a non-standard primal-dual technique to get a  $\delta$ -approximation algorithm for FRACTIONAL PACKING and MAXIMUM WEIGHTED b-MATCHING. In Section 3 we present the distributed implementation for  $\delta = 2$ . Then in Section 4 we show the distributed  $\delta$ -approximation algorithm for general  $\delta$ . We conclude in Section 5.

## 2 Covering and Packing

Koufogiannakis and Young show sequential and distributed  $\delta$ -approximation algorithms for general covering problems [13,12], where  $\delta$  is the maximum number of covering variables on which a covering constraint depends. As a special case their algorithms compute  $\delta$ -approximate solutions for FRACTIONAL COVERING problems of the form  $\min\{\sum_{i=1}^n b_i y_i : \sum_{i=1}^n A_{ij} y_i \geq w_j \ (\forall j = 1..m), y \in \mathbf{R}_+^n\}$ . The linear programming dual of such a problem is the following FRACTIONAL PACKING problem:  $\max\{\sum_{j=1}^m w_j x_j : \sum_{j=1}^m A_{ij} x_j \leq b_i \ (\forall i = 1..n), x \in \mathbf{R}_+^m\}$ . For packing,  $\delta$  is the maximum number of packing constraints in which a packing variable appears,  $\delta = \max_j |\{i \mid A_{ij} \neq 0\}|$ .

Here we extend the distributed approximation algorithm for FRACTIONAL COVERING by [12] to compute  $\delta$ -approximate solutions for FRACTIONAL PACKING using a non-standard primal-dual approach.

**Notation.** Let  $C_j$  denote the  $j$ -th covering constraint ( $\sum_{i=1}^n A_{ij} y_i \geq w_j$ ) and  $P_i$  denote the  $i$ -th packing constraint ( $\sum_{j=1}^m A_{ij} x_j \leq b_i$ ). Let  $\text{Vars}(S)$  denote the set of (covering or packing) variable indexes that appear in (covering or packing) constraint  $S$ . Let  $\text{Cons}(z)$  denote the set of (covering or packing) constraint indexes in which (covering or packing) variable  $z$  appears. Let  $N(x_s)$  denote the set of packing variables that appear in the packing constraints in which  $x_s$  appears, that is,  $N(x_s) = \{x_j \mid j \in \text{Vars}(P_i) \text{ for some } i \in \text{Cons}(x_s)\} = \text{Vars}(\text{Cons}(x_s))$ .

**Fractional Covering.** First we give a brief description of the  $\delta$ -approximation algorithm for fractional covering by [13,12]<sup>2</sup>. The algorithm performs *steps* to cover non-yet-satisfied covering constraints. Let  $y^t$  be the solution after the first  $t$  steps have been performed. (Initially  $y^0 = \mathbf{0}$ .) Given  $y^t$ , let  $w_j^t = w_j - \sum_{i=1}^n A_{ij} y_i^t$  be the slack of  $C_j$  after the first  $t$  steps. (Initially  $w^0 = w$ .) The algorithm is given by Alg. 1.

<sup>2</sup> The algorithm is equivalent to local-ratio when  $A \in \{0, 1\}^{n \times m}$  and  $y \in \{0, 1\}^n$  [1,2]. See [13] for a more general algorithm and a discussion on the relation between this algorithm and local ratio.

There may be covering constraints for which the algorithm never performs a step because they are covered by steps done for other constraints with which they share variables. Also note that increasing  $y_i$  for all  $i \in \text{Vars}(C_s)$ , decreases the slacks of all constraints which depend on  $y_i$ .

**Our general approach.** [13] shows that the above algorithm is a  $\delta$ -approximation for covering, but they don't show any result for matching or other packing problems. Our general approach is to recast their analysis as a primal-dual analysis, showing that the algorithm (Alg. 1) implicitly computes a solution to the dual packing problem of interest here. To do this we use the tail-recursive approach implicit in previous local-ratio analyses [3].

**greedy  $\delta$ -approximation algorithm for fractional covering [13,12]**      alg. 1

1. Initialize  $y^0 \leftarrow \mathbf{0}$ ,  $w^0 \leftarrow w$ ,  $t \leftarrow 0$ .
2. While there exist an unsatisfied covering constraint  $C_s$  do a step for  $C_s$ :
3.     Set  $t = t + 1$ .
4.     Let  $\beta_s \leftarrow w_s^{t-1} \cdot \min_{i \in \text{Vars}(C_s)} b_i / A_{is}$ .      ... *OPT cost to satisfy  $C_s$  given the current solution*
5.     For each  $i \in \text{Vars}(C_s)$ :
6.         Set  $y_i^t = y_i^{t-1} + \beta_s / b_i$ .      ... *increase  $y_i$  inversely proportional to its cost*
7.         For each  $j \in \text{Cons}(y_i)$  update  $w_j^t = w_j^{t-1} - A_{ij} \beta_s / b_i$ .      ... *new slacks*
8. Return  $y = y^t$ .

After the  $t$ -th step of the algorithm, define the *residual covering problem* to be  $\min\{\sum_{i=1}^n b_i y_i : \sum_{i=1}^n A_{ij} y_i \geq w_j^t \ (\forall j = 1..m), y \in \mathbb{R}_+^n\}$  and the *residual packing problem* to be its dual,  $\max\{\sum_{j=1}^m w_j^t x_j : \sum_{j=1}^m A_{ij} x_j \leq b_i \ (\forall i = 1..n), x \in \mathbb{R}_+^m\}$ . The algorithm will compute  $\delta$ -approximate primal and dual pairs  $(x^t, y^{T-t})$  for the residual problem for each  $t$ . As shown in what follows, the algorithm increments the covering solution  $x$  in a forward way, and the packing solution  $y$  in a “tail-recursive” manner.

**Standard Primal-Dual approach does not work.** For even simple instances, generating a  $\delta$ -approximate primal-dual pair for the above greedy algorithm requires a non-standard approach. For example, consider  $\min\{y_1 + y_2 + y_3 : y_1 + y_2 \geq 1, y_1 + y_3 \geq 5, y_1, y_2 \geq 0\}$ . If the greedy algorithm (Alg. 1) does the constraints in *either* order and chooses  $\beta$  maximally, it gives a solution of cost 10. In the dual  $\max\{x_{12} + 5x_{13} : x_{12} + x_{13} \leq 1, x_{12}, x_{13} \geq 0\}$ , the only way to generate a solution of cost 5 is to set  $x_{13} = 1$  and  $x_{12} = 0$ . A standard primal-dual approach would raise the dual variable for each covering constraint when that constraint is processed (essentially allowing a dual solution to be generated in an *online* fashion, constraint by constraint). That can't work here. For example, if the constraint  $y_1 + y_2 \geq 1$  is covered first by setting  $y_1 = y_2 = 1$ , then the dual variable  $x_{12}$  would be increased, thus preventing  $x_{13}$  from reaching 1.

Instead, assuming the step to cover  $y_1 + y_2 \geq 1$  is done first, the algorithm should not increase any packing variable until a solution to the residual dual problem is computed. After this step the residual primal problem is  $\min\{y'_1 + y'_2 + y'_3 : y'_1 + y'_2 \geq -1, y'_1 + y'_3 \geq 4, y'_1, y'_2 \geq 0\}$ , and the residual dual problem

is  $\max\{-x'_{12} + 4x'_{13} : x'_{12} + x'_{13} \leq 1, x'_{12}, x'_{13} \geq 0\}$ . Once a solution  $x'$  to the residual dual problem is computed (either recursively or as shown later in this section) then the dual variable  $x'_{12}$  for the current covering constraint should be raised maximally, giving the dual solution  $x$  for the current problem. In detail, the residual dual solution  $x'$  is  $x'_{12} = 0$  and  $x'_{13} = 1$  and the cost of the residual dual solution is 4. Then the variable  $x'_{12}$  is raised maximally to give  $x_{12}$ . However, since  $x'_{13} = 1$ ,  $x'_{12}$  cannot be increased, thus  $x = x'$ . Although neither dual coordinate is increased at this step, the dual cost is increased from 4 to 5, because the weight of  $x_{13}$  is increased from  $w'_{13} = 4$  to  $w_{13} = 5$ . (See Figure 2 in the appendix.) In what follows we present this formally.

**Fractional Packing.** We show that the greedy algorithm for covering creates an ordering of the covering constraints for which it performs steps, which we can then use to raise the corresponding packing variables. Let  $t_j$  denote the time<sup>3</sup> at which a step to cover  $C_j$  was performed. Let  $t_j = 0$  if no step was performed for  $C_j$ . We define the relation “ $C_{j'} \prec C_j$ ” on two covering constraints  $C_{j'}$  and  $C_j$  which share a variable and for which the algorithm performed steps to indicate that constraint  $C_{j'}$  was done first by the algorithm.

**Definition 1.** Let  $C_{j'} \prec C_j$  if  $\text{Vars}(C_{j'}) \cap \text{Vars}(C_j) \neq \emptyset$  and  $0 < t_{j'} < t_j$ .

Note that the relation is not defined for covering constraints for which a step was never performed by the algorithm. Then let  $\mathcal{D}$  be the partially ordered set (poset) of all covering constraints for which the algorithm performed a step, ordered according to “ $\prec$ ”.  $\mathcal{D}$  is partially ordered because “ $\prec$ ” is not defined for covering constraints that do not share a variable. In addition, since for each covering constraint  $C_j$  we have a corresponding dual packing variable  $x_j$ , abusing notation we write  $x_{j'} \prec x_j$  if  $C_{j'} \prec C_j$ . Therefore,  $\mathcal{D}$  is also a poset of packing variables.

**Definition 2.** A reverse order of poset  $\mathcal{D}$  is an order  $C_{j_1}, C_{j_2}, \dots, C_{j_k}$  (or equivalently  $x_{j_1}, x_{j_2}, \dots, x_{j_k}$ ) such that for  $l > i$  either we have  $C_{j_l} \prec C_{j_i}$  or the relation “ $\prec$ ” is not defined for constraints  $C_{j_i}$  and  $C_{j_l}$  (because they do not share a variable).

Then the following figure (Alg. 2) shows the sequential  $\delta$ -approximation algorithm for FRACTIONAL PACKING.

The algorithm simply considers the packing variables corresponding to covering constraints that Alg. 1 did steps for, and raises each variable maximally without violating the packing constraints. The order in which the variables are considered matters: *the variables should be considered in the reverse of the order in which steps were done for the corresponding constraints, or an order which is “equivalent” (see Lemma 1).* (This flexibility is necessary for the distributed setting.)

---

<sup>3</sup> In general by “time” we mean some reasonable way to distinguish in which order steps were performed to satisfy covering constraints. For now, the time at which a step was performed can be thought as the step number (line 3 at Alg. 1). It will be slightly different in the distributed setting.

**greedy  $\delta$ -approximation algorithm for fractional packing**

alg. 2

1. Run Alg. 1, recording the poset  $\mathcal{D}$ .
2. Let  $T$  be the number of steps performed by Alg. 1.
3. Initialize  $x^T \leftarrow \mathbf{0}$ ,  $t \leftarrow T$ . ... note that  $t$  will be decreasing from  $T$  to 0
4. Let  $\Pi$  be some reverse order of  $\mathcal{D}$ . ... any reverse order of  $\mathcal{D}$  works, see Lemma 1
5. For each variable  $x_s \in \mathcal{D}$  in the order given by  $\Pi$  do:
  6. Set  $x^{t-1} = x^t$ .
  7. Raise  $x_s^{t-1}$  until a packing constraint that depends on  $x_s^{t-1}$  is tight, that is, set  $x_s^{t-1} = \max_{i \in \text{Cons}(x_j)} (b_i - \sum_{j=1}^m A_{ij} x_j^{t-1})$ .
  8. Set  $t = t - 1$ .
9. Return  $x = x^0$ .

The solution  $x$  is feasible at all times since a packing variable is increased only until a packing constraint gets tight.

**Lemma 1.** *Alg. 2 returns the same solution  $x$  using (at line 4) any reverse order of  $\mathcal{D}$ .*

*Proof.* Let  $\Pi = x_{j_1}, x_{j_2}, \dots, x_{j_k}$  and  $\Pi' = x_{j'_1}, x_{j'_2}, \dots, x_{j'_k}$  be two different reverse orders of  $\mathcal{D}$ . Let  $x^{\Pi, 1 \dots m}$  be the solution computed so far by Alg. 2 after raising the first  $m$  packing variables of order  $\Pi$ . We prove that  $x^{\Pi, 1 \dots k} = x^{\Pi', 1 \dots k}$ .

Assume that  $\Pi$  and  $\Pi'$  have the same order for their first  $q$  variables, that is  $j_i = j'_i$  for all  $i \leq q$ . Then,  $x^{\Pi, 1 \dots q} = x^{\Pi', 1 \dots q}$ . The first variable in which the two orders disagree is the  $(q + 1)$ -th one, that is,  $j_{q+1} \neq j'_{q+1}$ . Let  $s = j_{q+1}$ . Then  $x_s$  should appear in some position  $l$  in  $\Pi'$  such that  $q + 1 < l \leq k$ . The value of  $x_s$  depends only on the values of variables in  $N(x_s)$  at the time when  $x_s$  is set. We prove that for each  $x_j \in N(x_s)$  we have  $x_j^{\Pi, 1 \dots q} = x_j^{\Pi', 1 \dots l}$ , thus  $x_s^{\Pi, 1 \dots q} = x_s^{\Pi', 1 \dots l}$ . Moreover since the algorithm considers each packing variable only once this implies  $x_s^{\Pi, 1 \dots k} = x_s^{\Pi', 1 \dots q} = x_s^{\Pi', 1 \dots l} = x_s^{\Pi', 1 \dots k}$ .

(a) For each  $x_j \in N(x_s)$  with  $x_s \prec x_j$ , the variable  $x_j$  should have already been set in the first  $q$  steps, otherwise  $\Pi$  would not be a valid reverse order of  $\mathcal{D}$ . Moreover each packing variable can be increased only once, so once it is set it maintains the same value till the end. Thus, for each  $x_j$  such that  $x_s \prec x_e$ , we have  $x_j^{\Pi, 1 \dots q} = x_j^{\Pi', 1 \dots q} = x_j^{\Pi', 1 \dots l}$ .

(b) For each  $x_j \in N(x_s)$  with  $x_j \prec x_s$ ,  $j$  cannot be in the interval  $[j'_{q+1}, \dots, j'_{l-1}]$  of  $\Pi'$ , otherwise  $\Pi'$  would not be a valid reverse order of  $\mathcal{D}$ . Thus, for each  $x_j$  such that  $x_j \prec x_s$ , we have  $x_j^{\Pi, 1 \dots q} = x_j^{\Pi', 1 \dots q} = x_j^{\Pi', 1 \dots l} = 0$ .

So in any case, for each  $x_j \in N(x_s)$ , we have  $x_j^{\Pi, 1 \dots q} = x_j^{\Pi', 1 \dots l}$  and thus  $x_s^{\Pi, 1 \dots q} = x_s^{\Pi', 1 \dots l}$ .

The lemma follows by induction on the number of edges. □

The following lemma and weak duality prove that the solution  $x$  returned by Alg. 2 is  $\delta$ -approximate.

**Lemma 2.** *For the solutions  $y$  and  $x$  returned by Alg. 1 and Alg. 2 respectively,  $\sum_{j=1}^m w_j x_j \geq 1/\delta \sum_{i=1}^n b_i y_i$ .*

*Proof.* Lemma 1 shows that any reverse order of  $\mathcal{D}$  produces the same solution, so w.l.o.g. here we assume that the reverse order  $\Pi$  used by Alg. 2 is the reverse of the order in which steps to satisfy covering constraints were performed by Alg. 1.

When Alg. 1 does a step to satisfy the covering constraint  $C_s$  (by increasing  $y_i$  by  $\beta_s/b_i$  for all  $i \in \text{Vars}(C_s)$ ), the cost of the covering solution  $\sum_i b_i y_i$  increases by at most  $\delta\beta_s$ , since  $C_s$  depends on at most  $\delta$  variables ( $|\text{Vars}(C_s)| \leq \delta$ ). Thus the final cost of the cover  $y$  is at most  $\sum_{s \in \mathcal{D}} \delta\beta_s$ .

Define  $\Psi^t = \sum_j w_j^t x_j^t$  to be the cost of the packing  $x^t$ . Recall that  $x^T = \mathbf{0}$  so  $\Psi^T = 0$ , and that the final packing solution is given by vector  $x^0$ , so the the cost of the final packing solution is  $\Psi^0$ . To prove the theorem we have to show that  $\Psi^0 \geq \sum_{s \in \mathcal{D}} \beta_s$ . We have that  $\Psi^0 = \Psi^0 - \Psi^T = \sum_{t=1}^T \Psi^{t-1} - \Psi^t$  so it is enough to show that  $\Psi^{t-1} - \Psi^t \geq \beta_s$  where  $C_s$  is the covering constraint done at the  $t$ -th step of Alg. 1.

Then,  $\Psi^{t-1} - \Psi^t$  is

$$\sum_j w_j^{t-1} x_j^{t-1} - w_j^t x_j^t \tag{1}$$

$$= w_s^{t-1} x_s^{t-1} + \sum_{j \neq s} (w_j^{t-1} - w_j^t) x_j^{t-1} \tag{2}$$

$$= w_s^{t-1} x_s^{t-1} + \sum_{i \in \text{Cons}(x_s)} \sum_{j \in \{\text{Vars}(P_j) - s\}} A_{ij} \frac{\beta_s}{b_i} x_j^{t-1} \tag{3}$$

$$= \beta_s x_s^{t-1} \max_{i \in \text{Cons}(x_s)} \frac{A_{is}}{b_i} + \sum_{i \in \text{Cons}(x_s)} \sum_{j \in \{\text{Vars}(P_j) - s\}} A_{ij} \frac{\beta_s}{b_i} x_j^{t-1} \tag{4}$$

$$\geq \beta_s \frac{1}{b_i} \sum_{j=1}^m A_{ij} x_j^{t-1} \quad (\text{for } i \text{ s.t. constraint } P_i \text{ becomes tight after raising } x_s) \tag{5}$$

$$= \beta_s \tag{6}$$

In equation (2) we use the fact that  $x_s^t = 0$  and  $x_j^{t-1} = x_j^t$  for all  $j \neq s$ . For equation (3), we use the fact that the residual weights of packing variables in  $N(x_s)$  are increased. If  $x_j > 0$  for  $j \neq s$ , then  $x_j$  was increased before  $x_s$  ( $x_s < x_j$ ) so at the current step  $w_j^{t-1} > w_j^t > 0$ , and  $w_j^{t-1} - w_j^t = \sum_{i \in \text{Cons}(x_s)} A_{ij} \frac{\beta_s}{b_i}$ . For equation (4), by the definition of  $\beta_s$  we have  $w_s^{t-1} = \beta_s \max_{i \in \text{Cons}(x_s)} \frac{A_{is}}{b_i}$ . In inequality (5) we keep only the terms that appear in the constraint  $P_i$  that gets tight by raising  $x_s$ . The last equality holds because  $P_i$  is tight, that is,  $\sum_{j=1}^m A_{ij} x_j = b_i$ .  $\square$

The following lemma shows that Alg. 2 returns integral solutions if the coefficients  $A_{ij}$  are 0/1 and the  $b_i$ 's are integers, thus giving a  $\delta$ -approximation algorithm for MAXIMUM WEIGHTED b-MATCHING.



**Lemma 3.** *If  $A \in \{0, 1\}^{n \times m}$  and  $b \in \mathbb{Z}_+^n$  then the returned packing solution  $x$  is integral, that is,  $x \in \mathbb{Z}_+^m$ .*

*Proof.* Since all non-zero coefficients are 1, the packing constraints are of the form  $\sum_{j \in \text{Vars}(P_i)} x_j \leq b_i$  ( $\forall i$ ). We prove by induction that  $x \in \mathbb{Z}_+^m$ . The base case is trivial since the algorithm starts with a zero solution. Assume that at some point we have  $x^t \in \mathbb{Z}_+^m$ . Let  $x_s \in \mathcal{D}$ , be the next packing variable to be raised by the algorithm. We show that  $x_s^{t-1} \in \mathbb{Z}_+$  and thus the resulting solution remains integral. The algorithm sets  $x_s^{t-1} = \min_{i \in \text{Cons}(x_s)} \{b_i - \sum_{j=1}^m x_j^{t-1}\} = \min_{i \in \text{Cons}(x_s)} \{b_i - \sum_{j=1}^m x_j^t\} \geq 0$ . By the induction hypothesis, each  $x_j^t \in \mathbb{Z}_+$ , and since  $b \in \mathbb{Z}_+^n$ , then  $x_s^{t-1}$  is also a non-negative integer.  $\square$

### 3 Distributed Fractional Packing with $\delta = 2$

#### 3.1 Distributed model for $\delta = 2$

We assume the network in which the distributed computation takes place has vertices for covering variables (packing constraints) and edges for covering constraints (packing variables). So, the network has a node  $u_i$  for every covering variable  $y_i$ . An edge  $e_j$  connects vertices  $u_i$  and  $u_{i'}$  if  $y_i$  and  $y_{i'}$  belong to the same covering constraint  $C_j$ , that is, there exists a constraint  $A_{ij}y_i + A_{i'j}y_{i'} \geq w_j$  ( $\delta = 2$  so there can be at most 2 variables in each covering constraint). We assume the standard synchronous communication model, where in each round, nodes can exchange messages with neighbors, and perform some local computation [21]. We also assume no restriction on message size and local computation. (Note that a synchronous model algorithm can be transformed into an asynchronous algorithm with the same time complexity [21].)

#### 3.2 Distributed Algorithm for $\delta = 2$

Koufogiannakis and Young show a distributed implementation of Alg. 1, for (fractional) covering with  $\delta = 2$  that runs in  $O(\log n)$  rounds in expectation and with high probability [12]. In this section we augment their algorithm to distributively compute 2-approximate solutions to the dual fractional packing problem without increasing the time complexity. The high level idea is similar to that in the previous section: run the distributed algorithm for covering to get a partial order of the covering constraints for which steps were performed, then consider the corresponding dual packing variables in “some reverse” order raising them maximally. The challenge here is that the distributed algorithm for covering can perform steps for many covering constraints in parallel. Moreover, each covering constraint, has just a local view of the ordering, that is, it only knows its relative order among the covering constraints with which it shares variables.

**Distributed Fractional Covering with  $\delta = 2$ .** Here is a short description of the distributed 2-approximation algorithm for fractional covering (Alg. 5 in appendix from [12]). In each round, the algorithm does steps on a large subset

of remaining edges (covering constraints), as follows. Each vertex (covering variable) randomly chooses to be a *leaf* or a *root*. A not-yet-satisfied edge  $e_j = (u_i, u_r)$  between a leaf  $u_i$  and a root  $u_r$  with  $b_i/A_{ij} \leq b_r/A_{rj}$  is *active* for the round. Each leaf  $u_i$  chooses a random *star* edge  $(u_i, u_r)$  from its active edges. These star edges form stars rooted at roots. Each root  $u_r$  then performs steps (of Alg. 1) on its star edges (in any order) until they are all satisfied.

Note that in a round the algorithm performs steps in parallel for edges not belonging to the same star. For edges belonging to the same star, their root performs steps for some of them one by one. There are edges for which the algorithm never performs steps because they are covered by steps done for adjacent edges.

In the distributed setting we define the time at which a step to satisfy  $C_j$  is done as a pair  $(t_j^R, t_j^S)$ , where  $t_j^R$  denotes the round in which the step was performed and  $t_j^S$  denotes that within the star this step is the  $t_j^S$ -th one. Let  $t_j^R = 0$  if no step was performed for  $C_j$ . Overloading Definition 1, we redefine “ $\prec$ ” as follows.

**Definition 3.** *Let  $C_{j'} \prec C_j$  (or equivalently  $x_{j'} \prec x_j$ ) if  $\text{Vars}(C_{j'}) \cap \text{Vars}(C_j) \neq \emptyset$  ( $j'$  and  $j$  are adjacent edges in the distributed network) and  $(\{0 < t_{j'}^R < t_j^R\}$  or  $\{t_{j'}^R = t_j^R \text{ and } t_{j'}^S < t_j^S\})$ .*

The pair  $(t_j^R, t_j^S)$  is enough to distinguish which of two adjacent edges had a step to satisfy its covering constraint performed first. Adjacent edges can have their covering constraints done in the same round only if they belong to the same star (they have a common root), thus they differ in  $t_j^S$ . Otherwise they are done in different rounds, so they differ in  $t_j^R$ . Thus the pair  $(t_j^R, t_j^S)$  and relation “ $\prec$ ” define a partially ordered set  $\mathcal{D}$  of all edges done by the distributed algorithm for covering.

**Lemma 4.** *([12]) Alg. 5 (for FRACTIONAL COVERING with  $\delta = 2$ ) finishes in  $T = O(\log m)$  rounds in expectation and with high probability. Simultaneously, Alg. 5 sets  $(t_j^R, t_j^S)$  for each edge  $e_j$  for which it performs a step ( $0 < t_j^R \leq T$ ), thus defining a poset of edges  $\mathcal{D}$ , ordered by “ $\prec$ ”.*

**Distributed Fractional Packing with  $\delta = 2$ .** Alg. 3 implements Alg. 2 in a distributed fashion. First, it runs Alg. 1 using the distributed implementation by [12] (Alg. 5) and recording  $\mathcal{D}$ . Meanwhile, as it discovers the partial order  $\mathcal{D}$ , it begins the second phase of Alg. 2, raising each packing variable as soon as it can. Specifically it waits to set a given  $x_j \in \mathcal{D}$  until after it knows that (a)  $x_j$  is in  $\mathcal{D}$ , (b) for each  $x_{j'} \in N(x_j)$  whether  $x_j \prec x_{j'}$ , and (c) each such  $x_{j'}$  is set. In other words, (a) a step has been done for the covering constraint  $C_j$ , (b) each adjacent covering constraint  $C_{j'}$  is satisfied and (c) for each adjacent  $C_{j'}$  for which a step was done after  $C_j$ , the variable  $x_{j'}$  has been set. Subject to these constraints it sets  $x_j$  as soon as possible. Note that some nodes will be executing the second phase of the algorithm (packing) while some other nodes are still executing the first phase (covering). This is necessary because a given node cannot know when distant nodes are done with the first phase.

<p><b>Distributed 2-approximation Fractional Packing with <math>\delta = 2</math></b></p> <p>input: Graph <math>G = (V, E)</math> representing a fractional packing problem instance with <math>\delta = 2</math>.</p> <p>output: Feasible <math>x</math>, 2-approximately minimizing <math>w \cdot x</math>.</p> <ol style="list-style-type: none"> <li>1. Each edge <math>e_j \in E</math> initializes <math>x_j \leftarrow 0</math>.</li> <li>2. Each edge <math>e_j \in E</math> initializes <math>done_j \leftarrow \text{false}</math>. ... <i>this indicates if <math>x_j</math> has been set to its final value</i></li> <li>3. Until each edge <math>e_j</math> has set its variable <math>x_j</math> (<math>done_j == \text{true}</math>), perform a round:</li> <li>4.   Perform a round of Alg. 5. ... <i>covering with <math>\delta = 2</math> augmented to compute <math>(t_j^R, t_j^S)</math></i></li> <li>5.   For each node <math>u_r</math> that was a root (in Alg. 5) at any previous round, consider locally at <math>u_r</math> all stars <math>S_r^t</math> that were rooted by <math>u_r</math> at any previous round <math>t</math>. For each star <math>S_r^t</math> perform <b>IncreaseStar</b>(<math>S_r^t</math>).</li> </ol>	<p>alg. 3</p>
<p><b>IncreaseStar</b>(star <math>S_r^t</math>):</p> <ol style="list-style-type: none"> <li>6. For each edge <math>e_j \in S_r^t</math> in decreasing order of <math>t_j^S</math>:</li> <li>7.   If <b>IncreasePackingVar</b>(<math>e_j</math>) == <b>UNDONE</b> then <b>BREAK</b> (stop the for loop).</li> </ol>	
<p><b>IncreasePackingVar</b>(edge <math>e_j = (u_i, u_r)</math>):</p> <ol style="list-style-type: none"> <li>8. If <math>e_j</math> or any of its adjacent edges has a non-yet-satisfied covering constraint return <b>UNDONE</b>.</li> <li>9. If <math>t_j^R == 0</math> then:</li> <li>10.   Set <math>x_j = 0</math> and <math>done_j = \text{true}</math>.</li> <li>11.   Return <b>DONE</b>.</li> <li>12. If <math>done_{j'} == \text{false}</math> for any edge <math>e_{j'}</math> such that <math>x_j \prec x_{j'}</math> then return <b>UNDONE</b>.</li> <li>13. Set <math>x_j = \min \{ (b_i - \sum_{j'} A_{ij'} x_{j'}) / A_{ij}, (b_r - \sum_{j'} A_{rj'} x_{j'}) / A_{rj} \}</math> and <math>done_j = \text{true}</math>.</li> <li>14. Return <b>DONE</b>.</li> </ol>	

All  $x_j$ 's will be determined in  $2T$  rounds by the following argument. After round  $T$ ,  $\mathcal{D}$  is determined. Then by a straightforward induction on  $t$ , within  $T + t$  rounds, every constraint  $C_j$  for which a step was done at round  $T - t$  of the first phase, will have its variable  $x_j$  set.

**Theorem 1.** *For FRACTIONAL PACKING where each variable appears in at most two constraints there is a distributed 2-approximation algorithm running in  $O(\log m)$  rounds in expectation and with high probability, where  $m$  is the number of packing variables.*

*Proof.* By Lemma 4, Alg. 5 computes a covering solution  $y$  in  $T = O(\log m)$  rounds in expectation and with high probability. At the same time, the algorithm sets  $(t_j^R, t_j^S)$  for each edge  $e_j$  for which it performs a step to cover  $C_j$ , and thus defining a poset  $\mathcal{D}$  of edges. In the distributed setting the algorithm does not define a linear order because there can be edges with the same  $(t_j^R, t_j^S)$ , that is, edges that are covered by steps done in parallel. However, since these edges must be non-adjacent, we can still think that the algorithm gives a linear order (as in the sequential setting), where ties between edges with the same  $(t_j^R, t_j^S)$  are broken arbitrarily (without changing  $\mathcal{D}$ ). Similarly, we can analyze Alg. 3 as if

it considers the packing variables in a reverse order of  $\mathcal{D}$ . Then, by Lemma 1 and Lemma 2 the returned solution  $x$  is 2-approximate.

We prove that the  $x$  can be computed in at most  $T$  extra rounds after the initial  $T$  rounds to compute  $y$ . First note that within a star, even though its edges are ordered according to  $t_j^S$  they can all set their packing variables in a single round if none of them waits for some adjacent edge packing variable that belongs to a different star. So in the rest of the proof we only consider the case where edges are waiting for adjacent edges that belong to different stars. Note that  $1 \leq t_j^R \leq T$  for each  $x_j \in \mathcal{D}$ . Then, at round  $T$ , each  $x_j$  with  $t_j^R = T$  can be set in this round because it does not have to wait for any other packing variable to be set. At the next round, round  $T + 1$ , each  $x_j$  with  $t_j^R = T - 1$  can be set; they are dependent only on variables  $x_{j'}$  with  $t_{j'}^R = T$  which have been already set. In general, packing variables with  $t_j^R = t$  can be set once all adjacent  $x_{j'}$  with  $t_{j'}^R \geq t + 1$  have been set. Thus by induction on  $t = 0, 1, \dots$  a constraint  $C_j$  for which a step was done at round  $T - t$  may have to wait until at most round  $T + t$  until its packing variable  $x_j$  is set. Therefore, the total number of rounds until solution  $x$  is computed is  $2T = O(\log m)$  in expectation and with high probability.  $\square$

The following theorem is a direct result of Lemma 3 and Thm 1 and the fact that for this problem  $m = O(n^2)$ .

**Theorem 2.** *For MAXIMUM WEIGHTED  $b$ -MATCHING on graphs there is a distributed 2-approximation algorithm running in  $O(\log n)$  rounds in expectation and with high probability.*

## 4 Distributed Fractional Packing with General $\delta$

### 4.1 Distributed Model for General $\delta$

Here we assume that the distributed network has a node  $v_j$  for each covering constraint  $C_j$  (packing variable  $x_j$ ), with edges from  $v_j$  to each node  $v_{j'}$  if  $C_j$  and  $C_{j'}$  share a covering variable  $y_i$ <sup>4</sup>. The total number of nodes in the network is  $m$ . Note that in this model the role of nodes and edges is reversed as compared to the model used in Section 3. We assume the standard synchronous model with unbounded message size.

### 4.2 Distributed Algorithm

Koufogiannakis and Young [12] show a distributed  $\delta$ -approximation algorithm for (fractional) covering problems with at most  $\delta$  variables per covering constraint that runs in  $O(\log^2 m)$  rounds in expectation and with high probability. Similar to the  $\delta = 2$  case, here we use this algorithm to get a poset of packing variables which we then consider in a reverse order, raising them maximally.

---

<sup>4</sup> The computation can easily be simulated on a network with nodes for covering variables or nodes for covering variables and covering constraints.

**Distributed covering with general  $\delta$ .** Here is a brief description of the distributed  $\delta$ -approximation algorithm for (fractional) covering from [12]. To start each phase, the algorithm finds large independent subsets of covering constraints by running one phase of Linial and Saks’ (LS) decomposition algorithm, with any  $k$  such that  $k \in \Theta(\ln m)^5$  [16]. The LS algorithm, for a given  $k$ , takes  $O(k)$  rounds and produces a random subset  $\mathcal{R} \subseteq \{v_j | j = 1 \dots m\}$  of the covering constraints, and for each covering constraint  $v_j \in \mathcal{R}$  a “leader”  $\ell(v_j) \in \mathcal{R}$ , with the following properties:

- Each  $v_j \in \mathcal{R}$  is within distance  $k$  of its leader:  $(\forall v_j \in \mathcal{R}) d(v_j, v_{\ell(j)}) \leq k$ .
- Components do not share covering variables (edges do not cross components):  $(\forall v_j, v_{j'} \in \mathcal{R}) v_{\ell(j)} \neq v_{\ell(j')} \Rightarrow \text{Vars}(v_j) \cap \text{Vars}(v_{j'}) = \emptyset$ .
- Each covering constraint node has a chance to be in  $\mathcal{R}$ :  $(\forall j = 1 \dots m) \Pr[v_j \in \mathcal{R}] \geq 1/cm^{1/k}$  for some  $c > 1$ .

Next, each node  $v_j \in \mathcal{R}$  sends its information (the constraint and its variables’ values) to its leader  $v_{\ell(j)}$ . This takes  $O(k)$  rounds because  $v_{\ell(j)}$  is at distance  $O(k)$  from  $v_j$ . Each leader then constructs (locally) the subproblem induced by the covering constraints that contacted it and the variables of those constraints, with their current values. Using this local copy, the leader does steps until all covering constraints that contacted it are satisfied. (Distinct leaders’ subproblems don’t share covering variables, so they can proceed simultaneously.) To end the phase, each leader  $u_\ell$  returns the updated variable information to the constraints that contacted  $v_\ell$ . Each covering constraint node in  $\mathcal{R}$  is satisfied in the phase.

To extend the algorithm to compute a solution to the dual packing problem the idea is similar to the  $\delta = 2$  case, substituting the role of stars by components and the role of roots by leaders. With each step done to satisfy the covering constraints  $C_j$ , the algorithm records  $(t_j^R, t_j^S)$ , where  $t_j^R$  is the round and  $t_j^S$  is the within-the-component iteration in which the step was performed. This defines a poset  $\mathcal{D}$  of covering constraints for which it performs steps.

**Lemma 5.** ([12]) *The distributed  $\delta$ -approximation algorithm for FRACTIONAL COVERING finishes in  $T = O(\log^2 m)$  rounds in expectation and with high probability, where  $m$  is the number of covering constraints (packing variables). Simultaneously, it sets  $(t_j^R, t_j^S)$  for each covering constraint  $C_j$  for which it performs a step ( $0 < t_j^R \leq T$ ), thus defining a poset of covering constraints (packing variables)  $\mathcal{D}$ , ordered by “ $\prec$ ”.*

**Distributed packing with general  $\delta$ .** (sketch) The algorithm (Alg. 4) is very similar to the case  $\delta = 2$ . First it runs the distributed algorithm for covering, recording  $(t_j^R, t_j^S)$  for each covering constraint  $C_j$  for which it performs a step. Meanwhile, as it discovers the partial order  $\mathcal{D}$ , it begins computing the packing solution, raising each packing variable as soon as it can. Specifically it waits to set a given  $x_j \in \mathcal{D}$  until after it knows that (a)  $x_j$  is in  $\mathcal{D}$ , (b) for each

---

<sup>5</sup> If nodes don’t know a  $k \in \Theta(\ln m)$ , a doubling technique can be used as a work-around [12].

<p><b>Distributed <math>\delta</math>-approximation Fractional Packing with general <math>\delta</math></b> <span style="float: right;">alg. 4</span></p> <p>input: Graph <math>G = (V, E)</math> representing a fractional packing problem instance.  output: Feasible <math>x</math>, <math>\delta</math>-approximately minimizing <math>w \cdot x</math>.</p> <ol style="list-style-type: none"> <li>1. Initialize <math>x \leftarrow 0</math>.</li> <li>2. For each <math>j = 1 \dots m</math> initialize <math>done_j \leftarrow \text{false}</math>. ... <i>this indicates if <math>x_j</math> has been set to its final value</i></li> <li>3. Until each <math>x_j</math> has been set (<math>done_j == \text{true}</math>) do:</li> <li>4.   Perform a phase of the <math>\delta</math>-approximation algorithm for covering by [12], recording <math>(t_j^R, t_j^S)</math>.</li> <li>5.   For each node <math>v_{\mathcal{K}}</math> that was a leader at any previous phase, consider locally at <math>v_{\mathcal{K}}</math> all components that chose <math>v_{\mathcal{K}}</math> as a leader at any previous phase. For each such component <math>\mathcal{K}_r</math> perform <b>IncreaseComponent</b>(<math>\mathcal{K}_r</math>).</li> </ol> <hr/> <p><b>IncreaseComponent</b>(component <math>\mathcal{K}_r</math>):</p> <ol style="list-style-type: none"> <li>6. For each <math>j \in \mathcal{K}_r</math> in decreasing order of <math>t_j^S</math>:</li> <li>7.   If <b>IncreasePackingVar</b>(<math>j</math>) == <b>UNDONE</b> then <b>BREAK</b> (stop the for loop).</li> </ol> <hr/> <p><b>IncreasePackingVar</b>(<math>j</math>):</p> <ol style="list-style-type: none"> <li>8. If <math>C_j</math> or any <math>C_{j'}</math> that shares covering variables with <math>C_j</math> is not yet satisfied return <b>UNDONE</b>.</li> <li>9. If <math>t_j^R == 0</math> then:</li> <li>10.   Set <math>x_j = 0</math> and <math>done_j = \text{true}</math>.</li> <li>11.   Return <b>DONE</b>.</li> <li>12. If <math>done_{j'} == \text{false}</math> for any <math>x_{j'}</math> such that <math>x_j \prec x_{j'}</math> then return <b>UNDONE</b>.</li> <li>13. Set <math>x_j = \min_{i \in \text{Cons}(x_j)} ((b_i - \sum_{j'} A_{ij'} x_{j'}) / A_{ij})</math> and <math>done_j = \text{true}</math>.</li> <li>14. Return <b>DONE</b>.</li> </ol>
--

$x_{j'} \in N(x_j)$  whether  $x_j \prec x_{j'}$ , and (c) each such  $x_{j'}$  is set. In other words, (a) a step has been done for the covering constraint  $C_j$ , (b) each adjacent covering constraint  $C_{j'}$  is satisfied and (c) for each adjacent  $C_{j'}$  for which a step was done after  $C_j$ , the variable  $x_{j'}$  has been set. Subject to these constraints it sets  $x_j$  as soon as possible.

To do so, the algorithm considers all components that have been done by leaders in previous rounds. For each component, the leader considers the component's packing variables  $x_j$  in order of decreasing  $t_j^S$ . When considering  $x_j$  it checks if each  $x_{j'}$  with  $x_j \prec x_{j'}$  is set, and if yes, then  $x_j$  can be set and the algorithm continues with the next component's packing variable (in order of decreasing  $t_j^S$ ). Otherwise the algorithm cannot yet decide about the remaining component's packing variables.

**Theorem 3.** *For FRACTIONAL PACKING where each variable appears in at most  $\delta$  constraints there is a distributed  $\delta$ -approximation algorithm running in  $O(\log^2 m)$  rounds in expectation and with high probability, where  $m$  is the number of packing variables.*

The proof is omitted because it is similar to the proof of Thm 1; the  $\delta$ -approximation ratio is given by Lemma 1 and Lemma 2, and the running time uses  $T = O(\log^2 m)$  by Lemma 5.

The following theorem is a direct result of Lemma 3 and Thm 3.

**Theorem 4.** *For MAXIMUM WEIGHTED b-MATCHING on hypergraphs, there is a distributed  $\delta$ -approximation algorithm running in  $O(\log^2 m)$  rounds in expectation and with high probability, where  $\delta$  is the maximum hyperedge degree and  $m$  is the number of hyperedges.*

## 5 Conclusions

We show a new non-standard primal-dual method, which extends the (local-ratio related) algorithms for fractional covering by [13,12] to compute approximate solutions to the dual fractional packing problem without increasing the time complexity (even in the distributed setting).

Using this new technique, we show a distributed 2-approximation algorithm for FRACTIONAL PACKING where each packing variable appears in at most 2 constraints and a distributed 2-approximation algorithm for MAXIMUM WEIGHTED b-MATCHING on graphs, both running in a logarithmic number of rounds. We also present a distributed  $\delta$ -approximation algorithm for FRACTIONAL PACKING where each variable appears in at most  $\delta$  constraints and a distributed  $\delta$ -approximation algorithm for MAXIMUM WEIGHTED b-MATCHING on hypergraphs, both running in  $O(\log^2 m)$  rounds, where  $m$  is the number of packing variables and hyperedges respectively.

## References

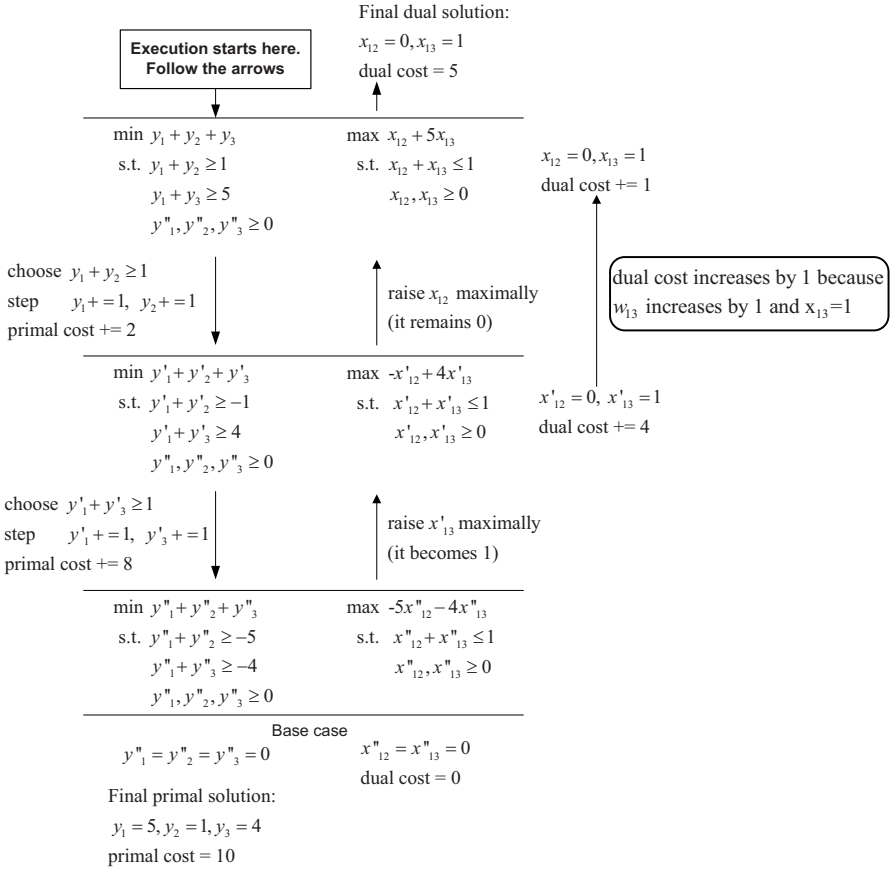
1. Bar-Yehuda, R.: One for the price of two: A unified approach for approximating covering problems. *Algorithmica* 27(2), 131–144 (2000)
2. Bar-Yehuda, R., Bendel, K., Freund, A., Rawitz, D.: Local ratio: a unified framework for approximation algorithms. *ACM Computing Surveys* 36(4), 422–463 (2004)
3. Bar-Yehuda, R., Rawitz, D.: On the equivalence between the primal-dual schema and the local-ratio technique. *SIAM Journal on Discrete Mathematics* 19(3), 762–797 (2005)
4. Czygrinow, A., Hańćkowiak, M., Szymańska, E.: A fast distributed algorithm for approximating the maximum matching. In: *The twelfth European Symposium on Algorithms*, pp. 252–263 (2004)
5. Edmonds, J.: Paths, trees, and flowers. *Canadian Journal of Mathematics* 17, 449–467 (1965)
6. Edmonds, J., Johnson, E.L.: Matching: A well-solved class of integer linear programs. In: *Combinatorial Structures and Their Applications*, pp. 89–92 (1970)
7. Hoepman, J.H.: Simple distributed weighted matchings. Arxiv preprint cs.DC/0410047 (2004)
8. Hougardy, S., Vinkemeier, D.E.: Approximating weighted matchings in parallel. *Information Processing Letters* 99(3), 119–123 (2006)
9. Israeli, A., Itai, A.: A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters* 22, 77–80 (1986)

10. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) *Complexity of Computer Computations*. The IBM Research Symposia Series, pp. 85–103. Plenum Press, New York (1972)
11. Koufogiannakis, C., Young, N.E.: Beating simplex for fractional packing and covering linear programs. In: *The forty-eighth IEEE symposium on Foundations of Computer Science*, pp. 494–504 (2007)
12. Koufogiannakis, C., Young, N.E.: Distributed and parallel algorithms for weighted vertex cover and other covering problems. In: *The twenty-eighth ACM symposium on Principles of Distributed Computing* (2009)
13. Koufogiannakis, C., Young, N.E.: Greedy  $\Delta$ -approximation algorithm for covering with arbitrary constraints and submodular cost. In: *The thirty-sixth International Colloquium on Automata, Languages and Programming*. LNCS, vol. 5555, pp. 634–652. Springer, Heidelberg (2009), <http://arxiv.org/abs/0807.0644>
14. Kuhn, F., Moscibroda, T., Wattenhofer, R.: The price of being near-sighted. In: *The seventeenth ACM-SIAM Symposium on Discrete Algorithm*, pp. 980–989 (2006)
15. Kuhn, F., Wattenhofer, R.: Constant-time distributed dominating set approximation. In: *The twetwenty-second ACM symposium on Principles of Distributed Computing*, pp. 25–32 (2003)
16. Linial, N., Saks, M.: Low diameter graph decompositions. *Combinatorica* 13(4), 441–454 (1993)
17. Lotker, Z., Patt-Shamir, B., Pettie, S.: Improved distributed approximate matching. In: *The 12th ACM Symposium on Parallelism in Algorithms and Architectures*, pp. 129–136 (2008)
18. Lotker, Z., Patt-Shamir, B., Rosén, A.: Distributed approximate matching. In: *The 26th ACM symposium on Principles Of Distributed Computing*, pp. 167–174 (2007)
19. Müller-Hannemann, M., Schwartz, A.: Implementing weighted b-matching algorithms: Towards a flexible software design. In: Goodrich, M.T., McGeoch, C.C. (eds.) *ALLENEX 1999*. LNCS, vol. 1619, pp. 18–36. Springer, Heidelberg (1999)
20. Nieberg, T.: Local, distributed weighted matching on general and wireless topologies. In: *The 5th ACM Joint Workshop on the Foundations of Mobile Computing, DIALM-POMC*, pp. 87–92 (2008)
21. Peleg, D.: *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics (2000)
22. Uehara, R., Chen, Z.: Parallel approximation algorithms for maximum weighted matching in general graphs. *Information Processing Letters* 76(1-2), 13–17 (2000)
23. Wattenhofer, M., Wattenhofer, R.: Distributed weighted matching. In: *The 18th international symposium on Distributed Computing*, pp. 335–348 (2004)



## Appendix

Alg. 5 shows the distributed 2-approximation algorithm for FRACTIONAL COVERING with  $\delta = 2$ .



**Fig. 2.** Example of the execution of our greedy primal-dual algorithm (assuming constraint  $y_1 + y_2 \geq 1$  is chosen first)

**Distributed 2-approximation Fractional Covering with  $\delta = 2$  ([12])** alg. 5

input: Graph  $G = (V, E)$  representing a fractional covering problem instance with  $\delta = 2$ .

output: Feasible  $y$ , 2-approximately minimizing  $b \cdot y$ .

1. Each node  $u_i \in V$  initializes  $y_i \leftarrow 0$ .
2. Each edge  $e_j \in E$  initializes  $t_j^R \leftarrow 0$  and  $t_j^S \leftarrow 0$ . *... auxiliary variables for Alg. 3*
3. Until there is a vertex with unsatisfied incident edges, perform a round:
  4. Each node  $u_i$ , randomly and independently chooses to be a *leaf* or a *root* for the round, each with probability  $1/2$ .
  5. Each leaf-to-root edge  $e_j = (u_i, u_r)$  with unmet covering constraint is *active* at the start of the round if  $u_i$  is a leaf,  $u_r$  is a root and  $b_i/A_{ij} \leq b_r/A_{rj}$ . Each leaf  $u_i$  chooses, among its active edges, a random one for the round. Communicate that choice to the neighbors. The chosen edges form independent *stars* — rooted trees of depth 1 whose leaves are leaf nodes and whose roots are root nodes.
  6. For each root node  $u_r$ , do:
    - (a) Let  $\mathcal{S}_r^t$  contain the star edges sharing variable  $y_r$  (at this round  $t$ ).
    - (b) Until there exist an unsatisfied edge (covering constraint)  $e_j = (u_i, u_r) \in \mathcal{S}_r^t$ , perform **Step**( $y, e_j$ ).

**Step**( $y, e_j = (u_i, u_r)$ ):

7. Let  $\beta_j \leftarrow (w_j - A_{ij}y_i - A_{rj}y_r) \cdot \min\{b_i/A_{ij}, b_r/A_{rj}\}$ .
8. Set  $y_i = y_i + \beta_j/b_i$  and  $y_r = y_r + \beta_j/b_r$ .
9. Set  $t_j^R$  to the number of rounds performed so far.
10. Set  $t_j^S$  to the number of steps performed by root  $u_r$  so far at this round.