Noname manuscript No.
(will be inserted by the editor)

Distributed Algorithms for Covering, Packing and Maximum Weighted Matching

Christos Koufogiannakis · Neal E. Young

Received: date / Accepted: date

Abstract This paper gives poly-logarithmic-round, distributed δ-approximation algorithms for covering problems with submodular cost and monotone covering constraints (Submodular-cost Covering). The approximation ratio δ is the maximum number of variables in any constraint. Special cases include Covering Mixed Integer Linear Programs (CMIP), and Weighted Vertex Cover (with $\delta = 2$).

Via duality, the paper also gives poly-logarithmic-round, distributed δ-approximation algorithms for Fractional Packing linear programs (where δ is the maximum number of constraints in which any variable occurs), and for Max Weighted c-Matching in hypergraphs (where δ is the maximum size of any of the hyperedges; for graphs $\delta = 2$).

The paper also gives parallel (RNC) 2-approximation algorithms for CMIP with two variables per constraint and Weighted Vertex Cover.

The algorithms are randomized. All of the approximation ratios exactly match those of comparable centralized algorithms.[1]

Keywords Approximation algorithms · Integer linear programming · Packing and covering · Vertex cover · Matching

C. Koufogiannakis
Department of Computer Science and Engineering
University of California, Riverside

N. E. Young
Department of Computer Science and Engineering
University of California, Riverside

[1] Preliminary versions appeared in [34,35]. Work by the first author was partially supported by the Greek State Scholarship Foundation (IKY). Work by the second author was partially supported by NSF awards CNS-0626912, CCF-0729071.

1 Background and results

Many distributed systems are composed of components that can only communicate locally, yet need to achieve a global (system-wide) goal involving many components. The general possibilities and limitations of such systems are widely studied [42,49,54,38,39,12]. It is of specific interest to see which fundamental combinatorial optimization problems admit efficient distributed algorithms that achieve approximation guarantees that are as good as those of the best centralized algorithms. Research in this spirit includes works on Set Cover (Dominating Set) [26,40,41], capacitated dominating set [37], Capacitated Vertex Cover [16, 17], and many other problems. This paper presents distributed approximation algorithms for some fundamental covering and packing problems.

The algorithms use the standard synchronous communication model: in each round, nodes can exchange a constant number of messages with neighbors and perform local computation [54]. There is no restriction on message size or local computation. The algorithms are *efficient* — they finish in a number of rounds that is poly-logarithmic in the network size [42].

1.1 Covering Problems

Consider optimization problems of the following form: *given a non-decreasing, continuous, and submodular*[2] *cost function* $c : \mathbb{R}_+^n \to \mathbb{R}_+$, *and a set of constraints* $\mathcal{C}$ *where each constraint* $S \in \mathcal{C}$ *is closed upwards*[3],

$$\textit{find } x \in \mathbb{R}_+^n \textit{ minimizing } c(x) \textit{ s.t. } (\forall S \in \mathcal{C})\ x \in S.$$

[2] Formally, $c(x) + c(y) \geq c(x \wedge y) + c(x \vee y)$ where $\wedge$ and $\vee$ are component-wise minimum and maximum, respectively.

[3] If $x \in S$ and $y \geq x$, then $y \in S$.

Distributed Computing 24(1): 45-63 (2011). doi: 10.1007/s00446-011-0127-7

Submodular-cost Covering includes all problems of this form [36]. The (approximation) parameter δ is the maximum number of elements of x on which any constraint $x \in S$ depends.

In the above formulation, x ranges over $\mathbb{R}^n_+$, but, because the constraints can be non-convex, one can handle arbitrarily restricted variable domains by incorporating them into the constraints [36]. For example, Weighted Vertex Cover is equivalent to *minimize* $\sum_v c_v x_v$ *subject to* $x \in \mathbb{R}^n_+$ *and*

$$x_u \geq 1 \text{ or } x_w \geq 1 \qquad (\forall (u,w) \in E).$$

(Given any 2-approximate solution x to this formulation — which allows $x_u \in \mathbb{R}_+$ — rounding each x_u down to its floor gives a 2-approximate integer solution.)

Submodular-cost Covering includes the following problems as special cases:

CIP, covering integer programs with variable upper bounds: *given* $A \in \mathbb{R}^{m\times n}_+$, $w \in \mathbb{R}^m_+$, $u \in \mathbb{R}^n_+$,

minimize $c \cdot x$ *subject to* $x \in \mathbb{Z}^n_+$, $Ax \geq w$, *and* $x \leq u$.

CMIP, covering mixed integer linear programs: CIP with both integer and fractional variables.

Facility Location, Weighted Set Cover (that is, CIP with $A_{ij} \in \{0,1\}, w_i = 1$), Weighted Vertex Cover (that is, Weighted Set Cover with $\delta = 2$), and probabilistic (two-stage stochastic) variants of these problems.

In the centralized (non-distributed) setting there are two well-studied classes of polynomial-time approximation algorithms for covering problems:

(i) $O(\log \Delta)$-approximation algorithms where Δ is the maximum number of constraints in which any variable occurs (e.g. [27,46,10,56,57,32]), and

(ii) $O(\delta)$-approximation algorithms where δ is the maximum number of variables in any constraint (e.g. [4, 22,21,5,19,7,8,55,2,36]), including most famously 2-approximation algorithms for Weighted Vertex Cover.

The algorithms here match those in class (ii).

Related work. For Unweighted Vertex Cover, it is well known that a 2-approximate solution can be found by computing any maximal matching, then taking the cover to contain the endpoints of the edges in the matching. A maximal matching (and hence a 2-approximate vertex cover) can be computed deterministically in $O(\log^4 n)$ rounds using the algorithm of Hańćkowiak, Karonski and Panconesi [20] or in $O(\Delta + \log^* n)$ rounds using the algorithm of Panconesi and Rizzi [51], where Δ is the maximum vertex degree. Or, using randomization, a maximal matching (and hence a 2-approximate vertex cover) can be computed in an expected $O(\log n)$ rounds via the algorithm of Israeli and Itai [25].

Maximal Matching is also in NC [9,47,30] and in RNC — parallel poly-log time with polynomially many randomized processors [25] — hence so is 2-approximating Unweighted Vertex Cover.

For Weighted Vertex Cover, previous to this work, no efficient distributed 2-approximation algorithm was known. Similarly, no parallel NC or RNC 2-approximation algorithm was known. in 1994, Khuller, Vishkin and Young gave $2(1+\varepsilon)$-approximation algorithms: a distributed algorithm taking $O(\log n \log 1/\varepsilon)$ rounds and a parallel algorithm, in NC for any fixed ε [31].[4] Given integer vertex weights, their result gives distributed 2-approximation in $O(\log n \log n\hat{C}))$ rounds, where $\hat{C}$ is the average vertex weight. The required number of rounds is reduced to (expected) $O(\log n\hat{C})$ by Grandoni, Könemann and Panconesi [18,15].

As noted in [38], neither of these algorithms is efficient (taking a number of rounds that is polylogarithmic in the number of vertices).

Kuhn, Moscibroda and Wattenhofer describe distributed approximation algorithms for *fractional* covering and packing linear programs [39]. They show an $O(1)$-approximation with high probability (w.h.p.) in $O(\log m)$ rounds (m is the number of covering constraints). The approximation ratio is greater than 2 for Fractional Weighted Vertex Cover. For (integer) Weighted Vertex Cover and Weighted Set Cover (where each $A_{ij} \in \{0,1\}$) combining their algorithms with randomized rounding gives $O(\log \Delta)$-approximate integer solutions in $O(\log n)$ rounds, where Δ is the maximum number of constraints in which any variable occurs.

Distributed lower bounds. The best lower bounds known for Weighted Vertex Cover are by Kuhn, Moscibroda and Wattenhofer: to achieve even a *polylogarithmic* approximation ratio requires in the worst case $\Omega(\sqrt{\log n/\log\log n})$ rounds. In graphs of constant degree Δ, $\Omega(\log \Delta/\log\log\Delta)$ rounds are required [38].

New results for covering problems. This paper gives the following results for covering problems.

- Section 2 describes the first efficient distributed 2-approximation algorithm for Weighted Vertex Cover. The algorithm runs in $O(\log n)$ rounds in expectation and with high probability.
- Section 3, generalizing the above result, describes the first efficient distributed 2-approximation algo-

[4] Their result extends to $\delta(1+\varepsilon)$-approximating Weighted Set Cover: a distributed algorithm taking $O(\delta \log n \log 1/\varepsilon)$ rounds, and a parallel algorithm (in NC for fixed ε and δ).

problem	approx. ratio	# rounds	where	when
Unweighted Vertex Cover	2	$O(\log n)$ (random)	[25]	1986
	2	$O(\log^4 n)$	[20]	2001
Weighted Vertex Cover	$2+\varepsilon$	$O(\log \varepsilon^{-1} \log n)$	[31]	1994
	2	$O(\log n \log n\hat{C})$	[31]	1994
	2	$O(\log n\hat{C})$ (random)	[18,15]	2005
	2	$O(\log n)$ (random)	here	
Weighted Set Cover	$O(\log \Delta)$	$O(\log m)$ (random)	[39]	2006
	δ	$O(\log m)$ (random)	here	
CMIP (with $\delta = 2$)	2	$O(\log m)$ (random)	here	
CMIP	δ	$O(\log^2 m)$ (random)	here	

Table 1 Comparison of distributed algorithms for covering problems. δ is the maximum number of variables in any constraint. Δ is the maximum number of constraints in which any variable occurs.

rithm for CMIP (covering mixed integer linear programs with variable upper bounds) restricted to instances where each constraint has at most two variables ($\delta = 2$). The algorithm runs in $O(\log m)$ rounds in expectation and with high probability, where m is the number of constraints.

– Section 4 gives the first efficient distributed δ-approximation algorithm for Submodular-Cost Covering (generalizing both problems above). The algorithm runs in $O(\log^2 m)$ rounds in expectation and with high probability, where m is the number of constraints.

Previously, even for the simple special case of 2-approximating Weighted Vertex Cover, no efficient distributed δ-approximation algorithm was known.

Each of the algorithms presented here is a distributed implementation of a (centralized) δ-approximation algorithm for Submodular-cost Covering by Koufogiannakis and Young [36]. Each section describes how that centralized algorithm specializes for the problem in question, then describes an efficient distributed implementation.

1.2 Fractional Packing, Maximum Weighted Matching

Fractional Packing is the following problem: *given matrix $A \in \mathbb{R}_+^{n\times m}$ and vectors $w \in \mathbb{R}_+^m$ and $c \in \mathbb{R}_+^n$,*

$$\textit{maximize } w \cdot y \textit{ subject to } y \in \mathbb{R}_+^m \textit{ and } Ay \le c.$$

This is the linear-program dual of Fractional Covering problem *minimize $c{\cdot}x$ s.t. $x \in \mathbb{R}_+^n$ and $A^T x \ge w$.*

For packing, δ is the maximum number of packing constraints in which any variable appears, that is, $\max_j |\{i : A_{ij} \neq 0\}|$. In the centralized setting, Fractional Packing can be solved optimally in polynomial time using linear programming. Alternatively, one can use a faster approximation algorithm (e.g., [33]).

Max Weighted c-Matching on a graph (or hypergraph) is the variant where each $A_{ij} \in \{0,1\}$ and the solution y must take integer values. (Without loss of generality each vertex capacity c_j is also an integer.) An instance is defined by a given hypergraph $H(V,E)$ and cost vector $c \in \mathbb{Z}_+^{|V|}$; a solution is given by a vector $y \in \mathbb{Z}_+^{|E|}$ maximizing $\sum_{e\in E} w_e y_e$ and meeting all the vertex capacity constraints $\sum_{e\in E(u)} y_e \le c_u$ ($\forall u \in V$), where $E(u)$ is the set of edges incident to vertex u. For this problem, $n = |V|$, $m = |E|$ and δ is the maximum (hyper)edge degree (for graphs $\delta = 2$).

Max Weighted c-Matching is a cornerstone optimization problem in graph theory and Computer Science. As a special case it includes the ordinary Max Weighted Matching problem (MWM) (where $c_u = 1$ for all $u \in V$). Restricted to graphs, the problem belongs to the "well-solved class of integer linear programs" in the sense that it can be solved optimally in polynomial time in the centralized setting [13,14,48]; moreover the obvious greedy algorithm (repeatedly select the heaviest edge that not conflicting with already selected edges) gives a 2-approximation[5] in nearly linear time. For hypergraphs the problem is NP-hard — it generalizes set packing, one of Karp's 21 NP-complete problems [28].

Related work for distributed MWM and Fractional Packing. Several previous works consider distributed Max Weighted Matching in graphs. Uehara and Chen present an $O(\Delta)$-approximation algorithm running in a constant number of rounds [58], where Δ is the maximum vertex degree. Wattenhofer and Wattenhofer improve this result, giving a randomized 5-approximation algorithm taking $O(\log^2 n)$ rounds [59]. Hoepman gives a deterministic 2-approximation algorithm taking $O(m)$ rounds [23]. Lotker, Patt-Shamir and Rosén give a randomized $(4+\varepsilon)$-approximation algorithm running in $O(\varepsilon^{-1} \log \varepsilon^{-1} \log n)$ rounds [45]. Lotker, Patt-Shamir and Pettie improve this result to a randomized $(2+\varepsilon)$-approximation algorithm taking $O(\log \varepsilon^{-1} \log n)$ rounds [44]. Their algorithm uses as a black box any distributed constant-factor approxima-

[5] Since it is a maximization problem it is also referred to as a 1/2-approximation.

problem	approx. ratio	# rounds		where	when
Max Weighted Matching in graphs	$O(\Delta)$	$O(1)$		[58]	2000
	5	$O(\log^2 n)$	(random)	[59]	2004
	$O(1)(>2)$	$O(\log n)$	(random)	[39,40]	2006
	$4+\varepsilon$	$O(\varepsilon^{-1}\log\varepsilon^{-1}\log n)$	(random)	[45]	2007
	$2+\varepsilon$	$O(\log\varepsilon^{-1}\log n)$	(random)	[44]	2008
	$1+\varepsilon$	$O(\varepsilon^{-4}\log^2 n)$	(random)	[44]	2008
	$1+\varepsilon$	$O(\varepsilon^{-2}+\varepsilon^{-1}\log(\varepsilon^{-1}n)\log n)$	(random)	[50]	2008
	2	$O(\log^2 n)$	(random)	[44,50] ($\varepsilon=1$)	2008
	$6+\varepsilon$	$O(\varepsilon^{-1}\log^4 n\log(C_{\max}/C_{\min}))$		[53]	2010
	2	$O(\log n)$	(random)	here	
Max Weighted c-Matching in graphs	$6+\varepsilon$	$O(\varepsilon^{-3}\log^3 n\log^2(C_{\max}/C_{\min}))$	(random)	[53]	2010
	2	$O(\log n)$	(random)	here	
Max Weighted c-Matching in hypergraphs	$O(\delta)>\delta$	$O(\log m)$	(random)	[39,40]	2006
	δ	$O(\log^2 m)$	(random)	here	
Fractional Packing ($\delta=2$)	$O(1)(>2)$	$O(\log m)$	(random)	[39]	2006
	2	$O(\log m)$	(random)	here	
Fractional Packing (general δ)	$O(1)>12$	$O(\log m)$	(random)	[39]	2006
	δ	$O(\log^2 m)$	(random)	here	

Table 2 Comparison of distributed algorithms for Max Weighted Matching and Fractional Packing.

tion algorithm for MWM that takes $O(\log n)$ rounds (e.g., [45]). 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 [24]. Nieberg gives a $(1+\varepsilon)$-approximation algorithm that runs in $O(\varepsilon^{-2}+\varepsilon^{-1}\log(\varepsilon^{-1}n)\log n)$ rounds [50]. The latter two results give randomized 2-approximation algorithms for Max Weighted Matching in $O(\log^2 n)$ rounds.

Independently of this work, Panconesi and Sozio [53] give a randomized distributed $(6+\varepsilon)$-approximation algorithm for Max Weighted c-Matching in graphs, requiring $O(\frac{\log^3 n}{\varepsilon^3}\log^2\frac{C_{\max}}{C_{\min}})$ rounds, provided all edges weights lie in $[C_{\min}, C_{\max}]$. They also give a similar (but deterministic) result for Max Weighted Matching.

Kuhn, Moscibroda and Wattenhofer show efficient distributed approximation algorithms for Fractional Packing [39]. They first give a deterministic $(1+\varepsilon)$-approximation algorithm for Fractional Packing with logarithmic message size, but the number of rounds depends on the input coefficients. For unbounded message size they give an algorithm for Fractional Packing that finds a constant-factor approximation ratio (w.h.p.) in $O(\log m)$ rounds. If an integer solution is desired, then distributed randomized rounding ([40]) can be used. This gives an $O(\delta)$-approximation for Max Weighted c-Matching on (hyper)graphs (w.h.p.) in $O(\log m)$ rounds, where δ 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 δ, say $\delta=2$.)

Distributed lower bounds. The best lower bounds known for distributed packing and matching are given by Kuhn, Moscibroda and Wattenhofer: to achieve even a poly-logarithmic approximation ratio for Fractional Maximum Matching takes at least $\Omega(\sqrt{\log n/\log\log n})$ rounds. At least $\Omega(\log\Delta/\log\log\Delta)$ rounds are required in graphs of constant degree Δ [39].

Other related work. For Unweighted Maximum Matching in graphs, Israeli and Itai give a randomized distributed 2-approximation algorithm running in $O(\log n)$ rounds [25]. Lotker, Patt-Shamir and Pettie improve this result giving a randomized $(1+\varepsilon)$-approximation algorithm taking $O(\varepsilon^{-3}\log n)$ rounds [44]. Czygrinow, Hańćkowiak, and Szymańska show a deterministic 3/2-approximation algorithm that takes $O(\log^4 n)$ rounds [11]. A $(1+\varepsilon)$-approximation for Max Weighted Matching in graphs is in NC [24].

New results for Fractional Packing and MWM. This work presents efficient distributed δ-approximation algorithms for Fractional Packing and Max Weighted c-Matching. The algorithms are primal-dual extensions of the δ-approximation algorithms for covering. .

- Section 6 describes a distributed 2-approximation algorithm for Fractional Packing where each variable appears in at most two constraints ($\delta=2$), running in $O(\log m)$ rounds in expectation and with high probability. Here m is the number of packing variables. This improves the approximation ratio over the previously best known algorithm [39].
- Section 7 describes a distributed δ-approximation algorithm for Fractional Packing where each variable appears in at most δ constraints, running in $O(\log^2 m)$ rounds in expectation and with high probability, where m is the number of variables. For small δ, this improves over the best previously known constant factor approximation [39], but the number of rounds is bigger by a logarithmic-factor.

– Section 6 gives a distributed 2-approximation algorithm for Max Weighted c-Matching in graphs, running in $O(\log n)$ rounds in expectation and with high probability. Max Weighted c-Matching generalizes the well studied Max Weighted Matching problem. For 2-approximation, this algorithm is faster by at least a logarithmic factor than any previous algorithm. Specifically, in $O(\log n)$ rounds, the algorithm gives the best known approximation ratio. The best previously known algorithms compute a $(1+\varepsilon)$-approximation in $O(\varepsilon^{-4}\log^2 n)$ rounds [44] or in $O(\varepsilon^{-2}+\varepsilon^{-1}\log(\varepsilon^{-1}n)\log n)$ rounds [50]. For a 2-approximation each of these algorithms needs $O(\log^2 n)$ rounds.

– Section 7 also gives a distributed δ-approximation algorithm for Max Weighted c-Matching in hypergraphs with maximum hyperedge degree δ, running in $O(\log^2 m)$ rounds in expectation and with high probability, where m is the number of hyperedges. This result improves over the best previously known $O(\delta)$-approximation ratio by [39], but it is slower by a logarithmic factor.

2 Weighted Vertex Cover

2.1 Sequential Implementation

First, consider the following sequential 2-approximation algorithm for Weighted Vertex Cover.[6]

The algorithm starts with $x = \mathbf{0}$. To cover edge (v, w), it calls $\mathsf{Step}(x, (v, w))$, which raises x_v and x_w at rates inversely proportional to their respective costs, until x_v or x_w reaches 1 (increase x_v by β/c_v and x_w by β/c_w, where $\beta = \min\{(1-x_v)c_v, (1-x_w)c_w\}$). When a variable x_v reaches 1, v is added to the cover. The algorithm does $\mathsf{Step}(x, e)$ for not-yet-covered edges e, until all edges are covered.

2.2 Distributed and Parallel Implementations

In each round, the *distributed* algorithm simultaneously performs $\mathsf{Step}(x, e)$ on a large subset of the not-yet-covered edges, as follows. Each vertex randomly chooses to be a *leaf* or a *root*. A not-yet-satisfied edge (v, w) is called *active* if v is a leaf, w is a root, and, if $\mathsf{Step}(x, (v, w))$ were to be performed, v would enter the cover. Each leaf v chooses a random active edge (v, w). The edges chosen by the leaves are called *star edges*; they form stars with roots at their centers.

[6] For Weighted Vertex Cover, this sequential algorithm by Koufogiannakis and Young [36] is equivalent to the classic 2-approximation algorithm by Bar-Yehuda et al. [4].

Each root w then flips a coin. If heads comes up (with probability 1/2), w calls $\mathsf{heads}(w)$, which does $\mathsf{Step}(x, (v, w))$ for its star edges (v, w) in any order, until w enters the cover or all of w's star edges have steps done. Or, if tails comes up, w calls $\mathsf{tails}(w)$, which simulates $\mathsf{heads}(w)$, without actually doing any steps, to determine the *last* edge (v, w) that $\mathsf{heads}(w)$ would do a step for, and performs step $\mathsf{Step}(x, (v, w))$ for just *that* edge. For details see Alg. 1.

Theorem 1 *For 2-approximating* Weighted Vertex Cover*:*

(a) There is a distributed algorithm running in $O(\log n)$ rounds in expectation and with high probability.

(b) There is a parallel algorithm in "Las Vegas" RNC.

Proof (a) The proof starts by showing that, in each round of Alg. 1, at least a constant fraction (1/224) of the not-yet-covered edges are covered in expectation.

Any not-yet-covered edge (v, w) is active for the round with probability at least 1/4, because $\mathsf{Step}(x, (v, w))$ would bring at least one of v or w into the cover, and with probability 1/4 that node is a leaf and the other is a root. Thus, by a lower-tail bound, with constant probability (1/7) at least a constant fraction (one eighth) of the remaining edges are active.[7] Assume at least an eighth of the remaining edges are active. Next, condition on all the choices of leaves and roots (assume these are fixed).

It is enough to show that, for an arbitrary leaf v, in expectation at least a quarter of v's active edges will be covered.[8] To do so, condition on the star edges chosen by the *other* leaves. (Now the only random choices *not* conditioned on are v's star-edge choice and the coin flips of the roots.)

At least one of the following two cases must hold.

Case 1: *At least half of v's active edges (v, w) have the following property: if v were to choose (v, w) as its star edge, and w were to do $\mathsf{heads}(w)$, then $\mathsf{heads}(w)$ would* not *perform $\mathsf{Step}(x, (v, w))$.* That is, w would enter the cover before $\mathsf{heads}(w)$ would consider (v, w) (in Fig. 1, see the cost-10 node).

For such an edge (v, w), on consideration, $\mathsf{heads}(w)$ will bring w into the cover *whether or not* v chooses

[7] In expectation, at most 3/4 of the edges are inactive. By Markov's inequality, the probability that more than 7/8 of the edges are inactive is at most $(3/4)/(7/8) = 6/7$. Thus, with probability at least 1/7, at least 1/8 of the edges are active.

[8] If so, then by linearity of expectation (summing over the leaves), at least 1/4 of all active edges will be covered. Since a 1/8 of the remaining edges are active (with probability 1/7), this implies that at least a $1/7 * 8 * 4 = 1/224$ fraction of the remaining edges are covered in expectation.

Distributed 2-approximation algorithm for Weighted Vertex Cover ($G = (V, E)$, $c : V \to \mathbb{R}_+$) Alg. 1

1. At each node v: initialize $x_v \leftarrow 0$.
2. Until all vertices are finished, perform rounds as follows:
3. At each node v: if all of v's edges are covered, finish; else, choose to be a *leaf* or *root*, each with probability 1/2.
4. At each leaf node v: Label each not-yet covered edge (v, w) *active* if w is a root and $\mathsf{Step}(x, (v, w))$ (with the current x) would add v to the cover. Choose, among these active edges, a random *star edge* (v, w).
5. At each root node w, flip a coin, then run the corresponding subroutine below:
   heads(w)**:** For each star edge (v, w) (in some fixed order) do: if w is not yet in the cover, then do $\mathsf{Step}(x, (v, w))$.
   tails(w)**:** Do $\mathsf{Step}(x, (v, w))$ just for the *last* edge for which $\mathsf{heads}(w)$ would do $\mathsf{Step}(x, (v, w))$.

$\mathsf{Step}(x, (v, w))$:

6. Let scalar $\beta \leftarrow \min\big((1 - x_v)c_v, (1 - x_w)c_w\big)$. *. . . just enough to ensure v or w is added to the cover below*
7. Set $x_v \leftarrow x_v + \beta/c_v$. If $x_v = 1$, add v to the cover, covering all of v's edges.
8. Set $x_w \leftarrow x_w + \beta/c_w$. If $x_w = 1$, add w to the cover, covering all of w's edges.

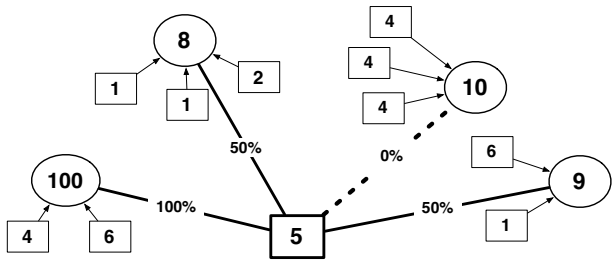

Fig. 1 Analysis of Alg. 1. Each node is labeled with its cost. Roots are circles; leaves are squares; star edges from leaves other than v (the cost-5 leaf) are determined as shown. Each edge (v, w) is labeled with the chance that v would enter the cover if v were to choose (v, w) for its star edge (assuming each $x_w = x_v = 0$ and each root w considers its star edges counter-clockwise).

(v, w) for its star edge. So, edge (v, w) will be covered in this round, regardless of v's choice of star edge, as long as w does $\mathsf{heads}(w)$. Since w does $\mathsf{heads}(w)$ with probability 1/2, edge (v, w) will be covered with probability 1/2.

Since this is true for at least half of v's active edges, in expectation, at least a quarter of v's active edges will be covered during the round.

Case 2: *At least half of v's active edges (v, w) have the following property: if v were to choose (v, w) as its star edge, and w were to do* $\mathsf{heads}(w)$*, then* $\mathsf{heads}(w)$ would *perform* $\mathsf{Step}(x, (v, w))$.

For such an edge (v, w), $\mathsf{heads}(w)$ would bring v into the cover as long as $\mathsf{Step}(x, (v, w))$ would not be the *last* step performed by $\mathsf{heads}(w)$ (in Fig. 1, the cost-8 and cost-100 nodes). Or, if $\mathsf{Step}(x, (v, w))$ would be the last step performed by $\mathsf{heads}(w)$, then $\mathsf{tails}(w)$ would do *only* $\mathsf{Step}(x, (v, w))$, which would bring v into the cover (by the assumption that, at the start of the round (v, w) is active so that $\mathsf{Step}(x, (v, w))$ would bring v into the cover) (in Fig. 1, the cost-9 node). Thus, for such an edge (v, w), one of $\mathsf{heads}(w)$ or $\mathsf{tails}(w)$ would bring v into the cover. Recall that w has probability 1/2 of doing $\mathsf{heads}(w)$ and probability 1/2 of doing $\mathsf{tails}(w)$. Thus, if v chooses such an edge, v enters the cover with at least probability 1/2.

In the case under consideration, v has probability at least 1/2 of choosing such an edge. Thus, with probability at least 1/4, v will enter the cover and all of v's edges will be deleted. Thus, in this case also, a quarter of v's edges are covered in expectation during the round.

Thus, in each round, in expectation at least 1/224 of the remaining edges are covered. By standard arguments, this implies that the expected number of rounds is at most about $224 \ln(n^2) = 448 \ln n$, and that the number of rounds is $O(\log n)$ with high probability. [9]

This completes the proof of Thm. 1, Part (a).

Next is the proof of Part (b). To obtain the parallel algorithm, implement $\mathsf{heads}(w)$ as follows. For w's kth star edge e_k, let β_k be the β that $\mathsf{Step}(x, e_k)$ would use if given x at the *start* of the round. If $\mathsf{heads}(w)$ eventually does $\mathsf{Step}(x, e_k)$ for edge e_k, the step will increase x_w by β_k/c_w, unless e_k is the last edge $\mathsf{heads}(w)$ does a step for. Thus, the edges for which $\mathsf{heads}(w)$ will do $\mathsf{Step}(x, e_k)$ are those for which $x_w + \sum_{j=1}^{k-1} \beta_j/c_w < 1$. These steps can be identified by a prefix-sum computation, then all but the last can be done in parallel. This gives an NC implementation of $\mathsf{heads}(w)$. The RNC algorithm simulates the distributed algorithm for $O(\log n)$ rounds; if the simulated algorithm halts, the RNC algorithm returns x, and otherwise it returns "fail". This completes the proof of Thm. 1. □

[9] For the high-probability bound, see e.g. [29, Thm. 1.1] (the example following that theorem). Alternatively, note that, in each round, by a lower-tail bound, with some constant probability at least some constant fraction of the remaining edges are covered. Say that a round is *good* if this happens. By Azuma's inequality, with high probability, for some suitably large constants $a < b$, at least $a \log m$ of the first $b \log m$ rounds are good, in which case all edges are covered in $b \log m$ rounds.

3 Mixed Integer Programs with Two Variables per Constraint

This section generalizes the results of Section 2 to CMIP_2 (CMIP with at most two non-zero coefficients A_{ij} in each constraint).

3.1 Sequential Implementation for CMIP (any δ)

First, consider the following sequential δ-approximation algorithm for CMIP [36]. Model the CMIP constraints (including the upper bounds and integrality constraints) by allowing each x_j to range freely in $\mathbb{R}_+$ but replacing each constraint $A_i x \geq b$ by the following equivalent constraint S_i:

$$\sum_{j \in I} A_{ij} \lfloor \min(x_j, u_j) \rfloor + \sum_{j \in \overline{I}} A_{ij} \min(x_j, u_j) \geq b_i$$

where set I contains the indexes of the integer variables.

The algorithm starts with $x = \mathbf{0}$, then repeatedly does $\mathsf{Step}(x, S)$, defined below, for any unsatisfied constraint S:

subroutine $\mathsf{Step}(x, S)$**:**

1. *Let* $\beta \leftarrow \mathsf{stepsize}(x, S)$.
2. *For each* j *with* $A_{ij} > 0$, *increase* x_j *by* β / c_j.

Here $\mathsf{stepsize}(x, S)$ (to be defined shortly) can be smaller than the β in Alg. 1. (Each step might not satisfy its constraint.)

After satisfying all constraints, the algorithm rounds each x_j down to $\lfloor \min(x_j, u_j) \rfloor$ and returns the rounded x. For this to produce a δ-approximate solution, it suffices for $\mathsf{stepsize}(x, S)$ to return a lower bound on the minimum cost of augmenting x to satisfy S, that is, on $\mathsf{distance}_c(x, S) = \min\{c(\hat{x}) - c(x) | \hat{x} \in S, \hat{x} \geq x\}$:

Observation 2 *([36]) If* $\mathsf{stepsize}(x, S) \leq \mathsf{distance}_c(x, S)$ *in each step, and the algorithm above terminates, then it returns a* δ*-approximate solution.*

Proof (sketch) Let x^* be any optimal solution. A step increases the cost $c \cdot x$ by $\delta\beta$, but decreases the potential $\sum_{v \in V} c_v \max(0, x_v^* - x_v)$ by at least β. Details in [36]. □

Compute $\mathsf{stepsize}(x, S_i)$ as follows. Consider the relaxations of S_i that can be obtained from S_i by relaxing any subset of the integrality constraints or variable upper bounds. (That is, replace $\lfloor \min(x_j, u_j) \rfloor$ by $\min(x_j, u_j)$ for any subset of the j's in I, and then replace $\min(x_j, u_j)$ by x_j for any subset of the j's.) Since there are at most δ variables per constraint and each can be relaxed (or not) in 4 ways, there are at most 4^δ such relaxed constraints.

Define the potential $\Phi(x, S_i)$ of constraint S_i to be the number of these relaxed constraints not satisfied by the current x. Compute $\mathsf{stepsize}(x, S_i)$ as *the minimum cost to increase just* **one** *variable enough to reduce* $\Phi(x, S_i)$.

Observation 3 *With this* $\mathsf{stepsize}()$, $\mathsf{Step}(x, S_i)$ *is done at most* 4^δ *times before constraint* S_i *is satisfied.*

(More efficient $\mathsf{stepsize}$ functions are described in [36].)

This step size satisfies the necessary condition for the algorithm to produce a δ-approximate solution:

Lemma 1 $\mathsf{stepsize}(x, S_i) \leq \mathsf{distance}_c(x, S_i)$

Proof Consider a particular relaxed constraint S_i' obtained by relaxing the upper-bound constraints for all x_j with $x_j < u_j$ and enforcing only a minimal subset J of the floor constraints (while keeping the constraint unsatisfied). This gives S_i', which is of the form

$$\sum_{j \in J} A_{ij} \lfloor x_j \rfloor + \sum_{j \in J'} A_{ij} x_j \;\geq\; b_i - \sum_{j \in J''} u_j$$

for some J, J', and J''.

What is the cheapest way to increase x to satisfy S_i'? Increasing any *one* term $\lfloor x_j \rfloor$ for $j \in J$ is enough to satisfy S_i' (increasing the left-hand side by A_{ij}, which by the minimality of J must be enough to satisfy the constraint).

Or, if no such term increases, then $\sum_{j \in J'} A_{ij} x_j$ must be increased by enough so that increase alone is enough to satisfy the constraint. The cheapest way to do that is to increase just one variable (x_j for $j \in J'$ maximizing A_{ij}/c_j).

In sum, for this S_i', $\mathsf{distance}(x, S_i')$ is the minimum cost to increase just *one* variable so as to satisfy S_i'. Thus, by its definition, $\mathsf{stepsize}(x, S_i) \leq \mathsf{distance}(x, S_i')$. It follows that

$$\mathsf{stepsize}(x, S_i) \leq \mathsf{distance}(x, S_i') \leq \mathsf{distance}(x, S_i).$$

□

Example. Minimize $x_1 + x_2$ subject to $0.5x_1 + 3x_2 \geq 5$, $x_2 \leq 1$, and $x_1, x_2 \in \mathbb{Z}_+$. Each variable has cost 1, so each step will increase each variable equally. There are eight relaxed constraints:

$$0.5x_1 + 3x_2 \geq 5 \tag{1}$$
$$0.5x_1 + 3\lfloor x_2 \rfloor \geq 5 \tag{2}$$
$$0.5x_1 + 3\min\{x_2, 1\} \geq 5 \tag{3}$$
$$0.5x_1 + 3\lfloor \min\{x_2, 1\} \rfloor \geq 5 \tag{4}$$
$$0.5\lfloor x_1 \rfloor + 3x_2 \geq 5 \tag{5}$$
$$0.5\lfloor x_1 \rfloor + 3\lfloor x_2 \rfloor \geq 5 \tag{6}$$
$$0.5\lfloor x_1 \rfloor + \min\{x_2, 1\} \geq 5 \tag{7}$$
$$0.5\lfloor x_1 \rfloor + 3\lfloor \min\{x_2, 1\} \rfloor \geq 5 \tag{8}$$

At the beginning, $x_1 = x_2 = 0$. No relaxed constraint is satisfied, so $\Phi(x, S) = 8$. Then $\mathsf{stepsize}(x, S) = 5/3$ (Constraint (1) or (5) would be satisfied by raising x_2 by 5/3). The first step raises x_1 and x_2 to 5/3, reducing $\Phi(x, S)$ to 6.

For the second step, $\mathsf{stepsize}(x, S) = 1/3$ (Constraint (2) or (6) would be satisfied by raising x_2 by 1/3). The step raises both variables by 1/3 to 2, lowering $\Phi(x, S)$ to 4.

For the third step, $\mathsf{stepsize}(x, S) = 2$, (Constraint (3), (4), (7), or (8) would be satisfied by raising x_1 by 2). The step raises both variables by 2, to 4, decreasing $\Phi(x, S)$ to 0.

All constraints are now satisfied, and the algorithm returns $x_1 = \lfloor x_1 \rfloor = 4$ and $x_2 = \lfloor \min\{x_2, 1\} \rfloor = 1$.

3.2 Distributed and Parallel Implementations for $\delta = 2$

This section describes a distributed implementation of the above sequential algorithm for CMIP_2 — the special case of CMIP with $\delta = 2$. The algorithm (Alg. 2) generalizes Alg. 1.

We assume the network in which the distributed computation takes place has a node v for every variable x_v, with an edge (v, w) for each constraint S that depends on variables x_v and x_w. (The computation can easily be simulated on, say, a network with vertices for constraints and edges for variables, or a bipartite network with vertices for constraints and variables.)

In Alg. 1, a constant fraction of the edges are likely to be covered each round because a step done for one edge can cover not just that edge, but many others also. The approach here is similar. Recall the definition of $\Phi(x, S)$ in the definition of $\mathsf{stepsize}()$. The goal is that the total potential of all constraints, $\Phi(x) = \sum_S \Phi(x, S)$, should decrease by a constant fraction in each round.

Definition 1 *Say that a constraint S is* hit *during the round when its potential $\Phi(x, S)$ decreases as the result of some step.*

By the definition of $\mathsf{stepsize}()$*, for any x and any constraint S there is at least one variable x_v such that raising* **just** *x_v to $x_v + \mathsf{stepsize}(x, S)/c_v$ would be enough to hit S. Say such a variable x_v* can hit *S (given the current x).*

The goal is that a constant fraction of the unmet constraints should be hit in each round.

Note that the definition implies, for example, that, among constraints that can be hit by a given variable x_v, doing a single step for the constraint S maximizing $\mathsf{stepsize}(x, S)$ will hit all such constraints. Likewise, doing a single step for a random such constraint will hit in expectation at least half of them (those with $\mathsf{stepsize}(x, S') \leq \mathsf{stepsize}(x, S)$).

In each round of the algorithm, each node randomly chooses to be a *leaf* or a *root*. Each (two-variable) constraint is *active* if one of its variables x_v is a leaf and the other, say x_w, is a root, and the leaf x_v can hit the constraint at the start of the round. (Each unmet constraint is active with probability at least 1/4.) Each leaf v chooses one of its active constraints at random to be a *star constraint*. Then each root w does (randomly) either $\mathsf{heads}(w)$ or $\mathsf{tails}(w)$, where $\mathsf{heads}(w)$ does steps for the star constraints rooted at w in a particular order; and $\mathsf{tails}(w)$ does just one step for the last star constraint that $\mathsf{heads}(w)$ would have done a step for (called w's "runt").

As $\mathsf{heads}(w)$ does steps for the star constraints rooted at w, x_w increases. As x_w increases, the status of a star constraint S rooted at w can change: it can be hit by the increase in x_w or it can cease to be hittable by x_v (and instead become hittable by x_w). For each constraint S, define threshold t_S to be the minimum value of x_w at which S's would have such a status change. Then $\mathsf{heads}(w)$ does steps in order of decreasing t_S until it reaches a constraint S with $x_w \geq t_S$. At that point, each of w's not-yet-hit star constraints S has $t_S \leq x_w$, and can still be hit by x_w. (As x_w increases, once S changes status, S will be hittable by x_w at least until S is hit.) Then $\mathsf{heads}(w)$ does $\mathsf{Step}(x, S_r)$ for the "runt" constraint S_r — the one, among w's not-yet-hit star constraints, maximizing $\mathsf{stepsize}(x, S_r)$. This step hits all of w's not-yet-hit star constraints. See Alg. 2 for details.

Theorem 4 *For 2-approximating* COVERING MIXED INTEGER LINEAR PROGRAMS *with at most two variables per constraint (CMIP_2):*

(a) there is a distributed algorithm running in $O(\log |\mathcal{C}|)$ rounds in expectation and with high probability, where $|\mathcal{C}|$ is the number of constraints.

(b) there is a parallel algorithm in "Las Vegas" RNC.

The next lemma is useful in the proof of the theorem to follow.

Lemma 2 *The total potential $\sum_{S_i} \Phi(x, S_i)$ decreases by a constant factor in expectation with each round.*

Proof Any unmet constraint is active with probability at least 1/4, so with constant probability the potential of the active edges is a constant fraction of the total potential. Assume this happens. Consider an arbitrary leaf v. It is enough to show that in expectation a constant fraction of v's active constraints are hit (have

Distributed 2-approximation algorithm for CMIP$_2$ (c, A, b, u, I) Alg. 2

1. At each node $v \in V$: initialize $x_v \leftarrow 0$;
   if there are unmet constraints S that depend only on x_v, do $\mathsf{Step}(x, S)$ for the one maximizing $\mathsf{stepsize}(x, S)$.
2. Until all vertices are finished, perform rounds as follows:
3. At each node v: if v's constraints are all met, finish (round x_v down to $\min(x_v, u_v)$, or $\lfloor \min(x_v, u_v) \rfloor$ if $v \in I$); otherwise, choose to be a *leaf* or a *root*, each with probability 1/2.
4. Each leaf v does: for each unmet constraint S that can be hit by x_v (Defn. 1), label S *active* if S's other variable is x_w for a root w; choose, among these active constraints, a random one to be x_v's *star constraint* (rooted at w).
5. Each root w does either $\mathsf{heads}(w)$ or $\mathsf{tails}(w)$ below, each with probability 1/2.

$\mathsf{heads}(w)$:

6. For each star constraint S rooted at w, let t_S be the minimum threshold such that increasing x_w to t_S would either hit S (i.e., decrease $\Phi(x, S)$) or make it so S's leaf variable x_v could no longer hit S (and x_w could).
   If there is no such value, then take $t_S = \infty$.
7. For each star constraint S rooted at w, in order of decreasing t_S, do the following:
   If $x_w < t_S$ then do $\mathsf{Step}(x, S)$ (hitting S); otherwise, stop the loop and do the following:
   Among the star constraints rooted at w that have not yet been hit this round, let S_r (the "runt") be one maximizing $\mathsf{stepsize}(x, S_r)$. Do $\mathsf{Step}(x, S_r)$ (hitting S_r and all not-yet-hit star constraints rooted at w).

$\mathsf{tails}(w)$:

8. Determine which constraint S_r would be the runt in $\mathsf{heads}(w)$. Do $\mathsf{Step}(x, S_r)$.

their potentials decrease) during the round. To do so, condition on any set of choices of star constraints by the *other* leaves, so the only random choices left to be made are v's star-constraint choice and the coin flips of the roots. Then (at least) one of the following three cases must hold:

Case 1. *A constant fraction of v's active constraints S have the following property: if v were to choose S as its star constraint, and the root w of S were to do* $\mathsf{heads}(w)$*, then* $\mathsf{heads}(w)$ *would* not *do* $\mathsf{Step}(x, S)$*.*

Although $\mathsf{heads}(w)$ would not do $\mathsf{Step}(x, S)$ for such an S, it nonetheless would hit S: just before $\mathsf{heads}(w)$ does $\mathsf{Step}(x, S_r)$, then $x_w \geq t_S$, so either S has already been hit (by the increases in x_w) or will be hit by $\mathsf{Step}(x, S_r)$ (because x_w can hit S and, by the choice of S_r, $\mathsf{Step}(x, S_r)$ increases x_w by $\mathsf{stepsize}(x, S_r)/c_w \geq \mathsf{stepsize}(x, S)/c_w$).

On consideration, for a constraint S with the assumed property, the steps done by $\mathsf{heads}(w)$ will be the same even if v chooses some constraint S' with a root other than w as its star constraint. (Or, if v chooses a constraint $S' \neq S$ that shares root w with S, the steps done by $\mathsf{heads}(w)$ will still raise x_w by as much as they would have had v chosen S for its star constraint.) Thus, for such a constraint S, $\mathsf{heads}(w)$ (which w does with probability at least 1/2) will hit S *whether or not* v chooses S as its star constraint.

If a constant fraction of v's active constraints have the assumed property, then a constant fraction of v's active constraints will be hit with probability at least 1/2, so in expectation a constant fraction of v's active constraints will be hit.

Case 2. *A constant fraction of v's active constraints S have the following property: if v were to choose S as its star constraint, and the root w of S were to do* $\mathsf{heads}(w)$*, then* $\mathsf{heads}(w)$ *would do* $\mathsf{Step}(x, S)$ *when $x_w < t_S$ (S would not be the runt).*

Let $\mathcal{H}$ denote the set of such constraints. For $S \in \mathcal{H}$ let $h(S)$ be the value to which $\mathsf{heads}(w)$ (where w is the root of S) would increase x_v. Whether or not v chooses S as its star constraint, if x_v increases to $h(S)$ in the round and w does $\mathsf{heads}(w)$, then S will be hit.

Let S and S' be any two constraints in $\mathcal{H}$ where $h(S) \geq h(S')$. Let w and w', respectively, be the root vertices of S and S'. (Note that $w = w'$ is possible.) If v chooses S' as its star constraint and w and w' both do $\mathsf{heads}()$, then S will be hit (because x_v increases to at least $h(S') \geq h(S)$ and $\mathsf{heads}(w)$ still increases x_w at least to the value it would have had just before $\mathsf{heads}(w)$ would have done $\mathsf{Step}(x, S)$, if v *had* chosen S as its star constraint).

Since (in the case under consideration) a constant fraction of v's active constraints are in $\mathcal{H}$, with constant probability v chooses some constraint $S' \in \mathcal{H}$ as its star constraint and the root w' of S' does $\mathsf{heads}(w')$. Condition on this happening. Then the chosen constraint S' is uniformly random in $\mathcal{H}$, so, in expectation, a constant fraction of the constraints S in $\mathcal{H}$ are hit (because $h(S) \leq h(S')$ and the root w of S also does $\mathsf{heads}(w)$).

Case 3. *A constant fraction of v's active constraints S have the following property: if v were to choose S as its star constraint, and the root w of S were to do* $\mathsf{tails}(w)$*, then* $\mathsf{tails}(w)$ *would do* $\mathsf{Step}(x, S)$ *(S would be the runt).*

Let $\mathcal{T}$ denote the set of such constraints. For $S \in \mathcal{T}$ let $t(S)$ be the value to which $\mathsf{tails}(w)$ (where w is the root of S) would increase x_v. Whether or not v chooses S as its star constraint, if x_v increases to $t(S)$ in the round then S will be hit (whether or not w does $\mathsf{tails}(w)$).

Let S and S' be any two constraints in $\mathcal{T}$ where $t(S') \geq t(S)$. Let w and w', respectively, be the root vertices of S and S'. (Again $w = w'$ is possible.) If v chooses S' as its star constraint and w' does $\mathsf{tails}(w')$, then (because x_v increases to at least $t(S') \geq t(S)$) S will be hit.

Since (in the case under consideration) a constant fraction of v's active constraints are in $\mathcal{T}$, with constant probability v chooses some constraint $S' \in \mathcal{T}$ as its star constraint and the root w' of S' does $\mathsf{tails}(w')$. Condition on this happening. Then the chosen constraint S' is uniformly random in $\mathcal{T}$, so, in expectation, a constant fraction of the constraints S in $\mathcal{T}$ are hit (because $t(S) \leq t(S')$). This proves the lemma. □

Proof (Thm. 4, part (a)) The lemma implies that the potential decreases in expectation by a constant factor each round. As the potential is initially $O(|\mathcal{C}|)$ and non-increasing, standard arguments (see Footnote 9) imply that the number of rounds before the potential is less than 1 (and so x must be feasible) is $O(\log |\mathcal{C}|)$ in expectation and with high probability.

This completes the proof of Thm. 4, part (a). □

Parallel (RNC) implementation

Proof (Thm. 4, part (b)) To adapt the proof of (a) to prove part (b), the only difficulty is implementing step (2) of $\mathsf{heads}(w)$ in NC. This can be done using the following observation. When $\mathsf{heads}(w)$ does $\mathsf{Step}(x, S_k)$ for its kth star constraint (except the runt), the effect on x_w is the same as setting $x_w \leftarrow f_k(x_w)$ for a linear function f_k that can be determined at the start of the round. By a prefix-sum-like computation, compute, in NC, for all i's, the functional composition $F_k = f_k \circ f_{k-1} \circ \cdots \circ f_1$. Let x_w^0 be x_w at the start of the round. Simulate the steps for all constraints S_k in parallel by computing $x_w^k = F_k(x_w^0)$, then, for each k with $x_w^{k-1} < t_{S_k}$, set the variable x_v of S_k's leaf v by simulating $\mathsf{Step}(x, S_k)$ with $x_w = x_w^{k-1}$. Set x_w to x_w^k for the largest k with $x_w^{k-1} < t_{S_k}$. Finally, determine the runt S and do $\mathsf{Step}(x, S)$. This completes the description of the NC simulation of $\mathsf{heads}(w)$.

The RNC algorithm will simulate some $c \log |\mathcal{C}|$ rounds of the distributed algorithm, where c is chosen so the probability of termination is at least 1/2. If the distributed algorithm terminates in that many rounds, the RNC algorithm will return the computed x. Otherwise the RNC algorithm will return "fail".

This concludes the proof of Thm. 4. □

4 Submodular-cost Covering

This section describes an efficient distributed algorithm for Submodular-cost Covering. Given a cost function c and a collection of constraints $\mathcal{C}$, the problem is to find $x \in \mathbb{R}_+^n$ to

minimize $c(x)$, *subject to* $(\forall S \in \mathcal{C})\ x \in S$.

The cost function $c : \mathbb{R}_+^n \to \mathbb{R}_+$ is non-decreasing, continuous, and submodular. Each constraint $S \in \mathcal{C}$ is closed upwards.

4.1 Sequential Implementation

Here is a brief description of the centralized δ-approximation algorithm for this problem (for a complete description see [36]).

The algorithm starts with $x = \mathbf{0}$, then repeatedly does the following $\mathsf{Step}(x, S)$ for any not-yet-satisfied constraint S (below $\mathsf{vars}(S)$ denotes the variables in x that constraint S depends on):

subroutine $\mathsf{Step}(x, S)$**:**

1. *Let* $\beta \leftarrow \mathsf{stepsize}(x, S)$.
2. *For* $j \in \mathsf{vars}(S)$, *let* $x'_j \in \mathbb{R}_+ \cup \{\infty\}$ *be maximal s.t. raising* x_j *to* x'_j *would raise* $c(x)$ *by at most* β.
3. *For* $j \in \mathsf{vars}(S)$, *let* $x_j \leftarrow x'_j$.

Let $\mathsf{distance}_c(x, S)$ denote *the minimum cost of augmenting* x *to satisfy* S, $\min\{c(\hat{x}) - c(x) : \hat{x} \in S, \hat{x} \geq x\}$.

Observation 5 *([36]) If* $\mathsf{stepsize}(x, S) \leq \mathsf{distance}_c(x, S)$ *in each step, and the algorithm terminates, then it returns a* δ*-approximate solution.*

Proof (sketch, for linear c) Each step starts with $x \notin S$. Since the optimal solution x^* is in S and S is closed upwards, there must be *at least one* $k \in \mathsf{vars}(S)$ such that $x_k < x_k^*$. Thus, while the algorithm increases the cost of x by at most $\delta\beta$, it decreases the potential $\sum_j c_j \max(0, x_j^* - x_j)$ by at least β. Full proof in [36]. □

The function $\mathsf{stepsize}(x, S)$**.** One generic way to define $\mathsf{stepsize}(x, S)$ is as the minimum β such that $\mathsf{Step}(x, S)$ will satisfy S in one step. This choice satisfies the requirement $\mathsf{stepsize}(x, S) \leq \mathsf{distance}_c(x, S)$ in Observation 5 [36]. But the sequential algorithm above, and the distributed algorithm described next, will work correctly with any $\mathsf{stepsize}()$ satisfying the condition in Observation 5. For an easy-to-compute and efficient $\mathsf{stepsize}()$ function for CMIP, see [36].

4.2 Distributed Implementation

Say that the cost function $c(x)$ is *locally computable* if, given any assignment to x and any constraint S, the increase in $c(x)$ due to $\mathsf{Step}(x, S)$ raising the variables in S can be determined knowing only the values of those variables ($\{x_j \mid j \in \mathsf{vars}(S)\}$). Any linear or separable cost function is locally computable.

We assume the distributed network has a node for each constraint $S \in \mathcal{C}$, with edges from S to each node whose constraint S' shares variables with S ($\mathsf{vars}(S) \cap \mathsf{vars}(S') \neq \emptyset$). (The computation can easily be simulated on a network with nodes for variables or nodes for variables and constraints.) We assume unbounded message size.

Theorem 6 *For δ-approximating any* Submodular-cost Covering *problem with a locally computable cost function, there is a randomized distributed algorithm taking $O(\log^2 |\mathcal{C}|)$ communication rounds in expectation and with high probability, where $|\mathcal{C}|$ is the number of constraints.*

Proof To start each phase, the algorithm finds large independent subsets of constraints by running one phase of Linial and Saks' (LS) decomposition algorithm [43], [10] below, with any k such that $k \in \Theta(\log |\mathcal{C}|)$ (in case the nodes do not know such a value see the comment at the end of this subsection). A phase of the LS algorithm, for a given k, takes $O(k)$ rounds and produces a random subset $\mathcal{R} \subseteq \mathcal{C}$ of the constraints (nodes), and for each constraint $S \in \mathcal{R}$ a "leader" node $\ell(S) \in \mathcal{S}$, with the following properties:

- Each constraint in $\mathcal{R}$ is within distance k of its leader:
  $(\forall S \in \mathcal{R})\ d(S, \ell(S)) \leq k$.
- Edges do not cross components:
  $(\forall S, S' \in \mathcal{R})\ \ell(S) \neq \ell(S') \rightarrow \mathsf{vars}(S) \cap \mathsf{vars}(S') = \emptyset$.
- Each constraint has a chance to be in $\mathcal{R}$:
  $(\forall S \in \mathcal{C})\ \Pr[S \in \mathcal{R}] \geq 1/c|\mathcal{C}|^{1/k}$ for some $c > 1$.

Next, each constraint $S \in \mathcal{R}$ sends its information (the constraint and its variables' values) to its leader $\ell(S)$. This takes $O(k)$ rounds because $\ell(S)$ is at distance $O(k)$ from S. Each leader then constructs (locally) the subproblem induced by the constraints that contacted it and the variables of those constraints, with their current values. Using this local copy, the leader repeatedly does $\mathsf{Step}(x, S)$ for any not-yet-met constraint S that contacted it, until all constraints that contacted it are satisfied.

[10] The LS decomposition was also used in approximation algorithms for fractional packing and covering by Kuhn et al. [39].

Distributed algorithm for problems with arbitrary covering constraints and submodular cost Alg. 3

1. Initialize $x \leftarrow 0$.
2. Compute the Linial/Saks decomposition of the constraint graph G. Denote it $B_1, B_2, \ldots, B_{O(\log |\mathcal{C}|)}$.
3. For $b = 1, 2, \ldots, O(\log |\mathcal{C}|)$, do:
4. Within each connected component $\mathcal{K}$ of block B_b:
5. Gather all constraints in $\mathcal{K}$ at the leader $v_{\mathcal{K}}$.
6. At $v_{\mathcal{K}}$, do $\mathsf{Step}(x, S)$ until $x \in S$ for all $S \in \mathcal{K}$.
7. Broadcast variables' values to all constraints in $\mathcal{K}$.

(By the assumption that the cost is locally computable, the function $\mathsf{stepsize}(x, S)$ and the subroutine $\mathsf{Step}(x, S)$ can be implemented knowing only the constraint S and the values of the variables on which S depends. Thus, the leader can perform $\mathsf{Step}(x, S)$ for each constraint that contacted it in this phase. Moreover, distinct leaders' subproblems do not share variables, so they can proceed simultaneously.)

To end the phase, each leader ℓ returns the updated variable information to the constraints that contacted ℓ. Each constraint in $\mathcal{R}$ is satisfied in the phase and drops out of the computation (it can be removed from the network and from $\mathcal{C}$; its variables' values will stabilize once the constraint and all its neighbors are finished).

Analysis of the number of rounds. In each phase (since each constraint is in $\mathcal{R}$, and thus satisfied, with probability $1/c|\mathcal{C}|^{1/k}$), the number of remaining constraints decreases by at least a constant factor $1 - 1/c|\mathcal{C}|^{1/k} \leq 1 - 1/\Theta(c)$ in expectation. Thus, the algorithm finishes in $O(c \log |\mathcal{C}|)$ phases in expectation and with high probability $1 - 1/|\mathcal{C}|^{O(1)}$. Since each phase takes $O(k)$ rounds, this proves the theorem.

Comment. If the nodes do not know a value $k \in \Theta(\log |\mathcal{C}|)$, use a standard doubling trick. Fix any constant $d > 0$. Start with $x = \mathbf{0}$, then run the algorithm as described above, except doubling values of k as follows. For each $k = 1, 2, 4, 8, \ldots$, run $O_d(k)$ phases as described above with that k. (Make the number of phases enough so that, if $k \geq \ln |\mathcal{C}|$, the probability of satisfying all constraints is at least $1 - 1/|\mathcal{C}|^d$.) The total number of rounds is proportional to the number of rounds in the last group of $O_d(k)$ phases.

To analyze this modification, consider the first $k \geq \log |\mathcal{C}|$. By construction, with probability at least $1 - 1/|\mathcal{C}|^d$, all constraints are satisfied after the $O_d(k)$ phases with this k. So the algorithm finishes in $O_d(\log |\mathcal{C}|)$ phases with probability at least $1 - 1/|\mathcal{C}|^d$.

To analyze the expected number of rounds, note that the probability of not finishing in each subsequent group of phases is at most $1/|\mathcal{C}|^d$, while the number of rounds increases by a factor of four for each increase in

k, so the expected number of subsequent rounds is at most $O_d(\log|\mathcal{C}|)\sum_{i=0}^{\infty} 4^i/|\mathcal{C}|^{di} = O_d(\log|\mathcal{C}|)$. □

We remark without proof that the above algorithm can be derandomized at the expense of increasing the number of rounds to super-polylogarithmic (but still sub-linear), using Panconesi and Srinivasan's deterministic variant of the Linial-Saks decomposition [52].

Also, note that although there are parallel (NC) variants of the Linial-Saks decomposition [1, Thm. 5], this does not yield a parallel algorithm. For example, if a single variable occurs in all constraints, the underlying network is complete, so has diameter 1, and thus can be "decomposed" into a single cluster. In the distributed setting, this case can be handled in $O(1)$ rounds (by doing all the computation at a single node). But it is not clear how to handle it in the parallel setting.

4.3 Applications

As mentioned in the introduction, the covering problem considered in this section generalizes many covering problems. For all of these, Thm. 6 gives a distributed δ-approximation algorithm running in $O(\log^2|\mathcal{C}|)$ communication rounds in expectation and with high probability.

Corollary 1 *There is a distributed δ-approximation algorithm for* Set Cover, CMIP *and (non-metric)* Facility Location *that runs in $O(\log^2|\mathcal{C}|)$ communication rounds in expectation and with high probability.*

If the complexity of the computation (as opposed to just the number of communication rounds) is important, these problems have appropriate stepsize() functions that can be computed efficiently (generally so that each constraint can be satisfied with overall work nearly linear in the problem size) [36].

Here is a brief description of the problems mentioned above but not previously defined.

(Non-metric) Facility Location, for a bipartite graph $G = (C, F, E)$ of customers C and facilities F, with assignment costs d and opening costs f, asks to find $x \geq 0$ such that $\sum_{j\in N(i)} \lfloor x_{ij} \rfloor \geq 1$ for each customer i, while minimizing the cost to *open* facilities $\sum_{j\in F} f_j \max_{i\in N(j)} x_{ij}$ plus the cost to *assign* customers to them $\sum_{ij\in E} d_{ij}x_{ij}$. The total cost is submodular. Each customer has at most δ accessible facilities.[11]

In probabilistic CMIP, the constraints are CMIP constraints and each constraint has a probability p_S of being active. The stage-one and stage-two costs are specified by a matrix w and a vector c, respectively. In stage one, the problem instance is revealed. The algorithm computes, for each constraint $S \in \mathcal{C}$, a "commitment" vector $y^S \in S$ for that constraint. The cost for stage one is $w \cdot y = \sum_{S, j\in \mathsf{vars}(S)} w_j^S y_j^S$. In stage two, each constraint S is (independently) *active* with probability p_S. Let $\mathcal{A}$ denote the active constraints. The final solution x is the minimal vector covering the active-constraint commitments, i.e. with $x_j = \max\{y_j^S : S \in \mathcal{A}, j \in \mathsf{vars}(S)\}$. The cost for stage two is the random variable $c \cdot x = \sum_j c_j x_j$. The problem is to choose y to minimize the total expected cost $C(y) = w\cdot y + E_{\mathcal{A}}[c\cdot x]$.

Probabilistic Facility Location is a special case of probabilistic CMIP, where each customer i is also given a probability p_i. In stage one, the algorithm computes x and is charged the assignment cost for x. In stage two, each customer i is *active* ($i \in \mathcal{A}$) with probability p_i, independently. The algorithm is charged opening costs f_j only for facilities with active customers (cost $\sum_j f_j \max_{i\in\mathcal{A}} x_{ij}$). The (submodular) cost $c(x)$ is the total *expected* charge.

[11] The standard linear-cost formulation is not a covering one. The standard reduction to set cover increases δ exponentially.

5 Sequential Primal-Dual Algorithm for Fractional Packing and MWM

This section gives a sequential δ-approximation algorithm for Fractional Packing which is the basis of subsequent distributed algorithms for Fractional Packing and Max Weighted c-Matching. The algorithm is a primal-dual extension of the previous covering algorithms.

Notation. Fix any Fractional Packing instance and its Fractional Covering dual:

$$\textit{maximize } w\cdot y \textit{ subject to } y \in \mathbb{R}_+^m \textit{ and } A^\intercal y \leq c,$$
$$\textit{minimize } c\cdot x \textit{ subject to } x \in \mathbb{R}_+^n \textit{ and } Ax \geq w.$$

Let C_i denote the i-th covering constraint ($A_i x \geq w_i$) and P_j denote the j-th packing constraint ($A_j^T y \leq c_j$). Let $\mathsf{vars}(S)$ contain the indices of variables in constraint S. Let $\mathsf{cons}(z)$ contain the indices of constraints in which variable z appears. Let $N(y_s)$ denote the set of packing variables that share constraints with y_s: $N(y_s) = \mathsf{vars}(\mathsf{cons}(y_s))$.

Fractional Covering. Alg. 4 shows a sequential algorithm for Fractional Covering. Each iteration of the algorithm does one step for an unsatisfied constraint C_s: taking the step size β_s to be the minimum such that raising just *one* variable $x_j \in \mathsf{vars}(C_s)$ by β_s/c_j is enough to make x satisfy C_s, then raising *each* variable x_j for $j \in \mathsf{vars}(C_s)$ by β_s/c_j.

Sequential algorithm for Fractional Covering Alg. 4

1. Initialize $x^0 \leftarrow \mathbf{0}$, $w^0 \leftarrow w$, $t \leftarrow 0$.
2. While $\exists$ an unsatisfied covering constraint C_s:
3. Set $t \leftarrow t+1$. *... do a step for C_s*
4. Let $\beta_s \leftarrow w_s^{t-1} \cdot \min_{j \in \mathsf{vars}(C_s)} c_j / A_{sj}$.
5. For each $j \in \mathsf{vars}(C_s)$:
6. Set $x_j^t \leftarrow x_j^{t-1} + \beta_s / c_j$.
7. For each $i \in \mathsf{cons}(x_j)$ set $w_i^t \leftarrow w_i^{t-1} - A_{ij}\beta_s/c_j$.
8. Let $x \leftarrow x^t$. Return x.

Fractional Covering is CMIP restricted to instances with no integer variables and no upper bounds on variables. For this special case, in the sequential algorithm for CMIP in Section 3.1, each constraint has no proper relaxation. Hence, that sequential algorithm reduces to Alg. 4. In turn, the sequential algorithm for CMIP in Section 3.1 is a special case of the sequential algorithm for Submodular-cost Covering in Section 4.1:

Observation 7 *Alg. 4 is a special case of both of the following two sequential algorithms:*

- *the algorithm for* CMIP *in Section 3.1, and*
- *the algorithm for* Submodular-cost Covering *in Section 4.1.*

The explicit presentation of Alg. 4 eases the analysis.

Fractional packing. Fix a particular execution of Alg. 4 on the Fractional Covering instance. In addition to computing a Fractional Covering solution x, the algorithm also (implicitly) defines a Fractional Packing solution y, as follows:[12]

Let T be the number of steps performed by Alg. 4. For $0 \le t \le T$, after the tth step of Alg. 4, let x^t be the covering solution so far, and let $w^t = w - Ax^t$ be the current slack vector. Define the *residual covering problem* to be

$$\textit{minimize } c \cdot x \textit{ subject to } x \in \mathbb{R}_+^n \textit{ and } Ax \ge w^t;$$

define the *residual packing problem* to be its dual:

$$\textit{maximize } w^t \cdot y \textit{ subject to } y \in \mathbb{R}_+^m \textit{ and } A^\intercal y \le c.$$

The residual packing constraints are independent of t. Now define a sequence $y^T, y^{T-1}, \ldots, y^1, y^0$, where y^t is a solution to the residual packing problem after step t, inductively, as follows: Take $y^T = \mathbf{0}$. Then, for each $t = T-1, T-2, \ldots, 0$, define y^t from y^{t+1} as follows: let C_s be the constraint for which Alg. 4 did a step in iteration t; start with $y^t = y^{t+1}$, then raise the single packing variable y_s^t maximally, subject to the packing constraints $A^T y \le c$. Finally, define y to be y^0.

The next lemma and weak duality prove that this y is a δ-approximation for Fractional Packing.

Lemma 3 *The cost of the covering solution x returned by Alg. 4 is at most δ times the cost of the packing solution y defined above: $w \cdot y \ge \frac{1}{\delta} c \cdot x$.*

Proof When Alg. 4 does a step to satisfy the covering constraint C_s (increasing x_j by β_s/c_j for the at most δ variables in $\mathsf{vars}(C_s)$), the covering cost $c \cdot x$ increases by at most $\delta\beta_s$, so at termination $c \cdot x$ is at most $\sum_{s \in \mathcal{D}} \delta\beta_s$. Since $w \cdot y = w^0 y^0 - w^T y^T = \sum_{t=1}^T (w^{t-1}y^{t-1} - w^t y^t)$, it is enough to show that $w^{t-1}y^{t-1} - w^t y^t \ge \beta_s$, where C_s is the covering constraint used in the tth step of Alg. 4. Show this as follows:

$$\begin{aligned}
w^{t-1}y^{t-1} - w^t y^t &= \sum_i (w_i^{t-1} y_i^{t-1} - w_i^t y_i^t) \\
&= w_s^{t-1}\, y_s^{t-1} + \sum_{i \ne s} (w_i^{t-1} - w_i^t) y_i^{t-1} && (9) \\
&= y_s^{t-1} \beta_s \max_{j \in \mathsf{cons}(y_s)} \frac{A_{sj}}{c_j} + \sum_{i \ne s; j \in \mathsf{cons}(y_s)} A_{ij} \frac{\beta_s}{c_j} y_i^{t-1} && (10) \\
&\ge \beta_s \frac{1}{c_j} \sum_{i=1}^m A_{ij} y_i^{t-1} \\
&\quad \text{(for } j \text{ s.t. constraint } P_j \text{ is tight after raising } y_s) && (11) \\
&= \beta_s
\end{aligned}$$

Eq. (9) follows from $y_s^t = 0$ and $y_i^{t-1} = y_i^t$ ($\forall i \ne s$). For (10), the definition of β_s gives $w_s^{t-1} = \beta_s \max_{j \in \mathsf{cons}(y_s)} \frac{A_{sj}}{c_j}$, and, by inspecting Alg. 4, $w_i^{t-1} - w_i^t$ is $\sum_{j \in \mathsf{vars}(C_s): i \in \mathsf{cons}(x_j)} A_{ij} \frac{\beta_s}{c_j} = \sum_{j \in \mathsf{cons}(y_s)} A_{ij} \frac{\beta_s}{c_j}$. Then (11) follows by dropping all but the terms for the index $j \in \mathsf{cons}(y_s)$ s.t. constraint P_j gets tight ($A_j^\intercal y^{t-1} = c_j$). The last equality holds because P_j is tight. □

By the next lemma, the packing solution y has integer entries as long as the coefficients A_{ij} are 0/1 and the c_j's are integers:

Lemma 4 *If $A \in \{0,1\}^{m \times n}$ and $c \in \mathbb{Z}_+^n$ then the packing solution y lies in $\mathbb{Z}_+^m$.*

Proof Since all non-zero coefficients are 1, the packing constraints are of the form $\sum_{i \in \mathsf{vars}(P_j)} y_i \le c_j$ ($\forall i$). By (reverse) induction on t, each y^t is in $\mathbb{Z}_+^m$. This is true initially, because $y^T = \mathbf{0}$. Assume it is true for y^{t-1}. Then it is true for y^t because, to obtain y^t from y^{t-1}, one entry y_i^t is raised maximally subject to $A^\intercal y \le c$. □

[12] This tail-recursive definition follows local-ratio analyses [6]. The more standard primal-dual approach — setting the packing variable for a covering constraint when a step for that constraint is done — doesn't work. See the appendix for an example.

Sequential algorithm for Fractional Packing Alg. 5

1. Run Alg. 4; record poset $\mathcal{D}$ of covering constraint indices.
2. Let Π be a total order of $\mathcal{D}$ respecting the partial order.
3. Initialize $y^T \leftarrow \mathbf{0}$ (where Alg. 4 does T steps).
4. For $t = T-1, T-2, \ldots, 0$ do:
5. Set $y^t \leftarrow y^{t+1}$.
6. For $s = \Pi_t$, raise y^t_s maximally subject to $A^\intercal y^t \le c$:
$$y^t_s \leftarrow \min_{j \in \mathsf{cons}(y_s)} (c_j - A^\intercal_j y^t)/A_{sj} \ .$$
7. Set $y \leftarrow y^0$. Return y.

Corollary 2 *For both* Fractional Packing *and* Max Weighted c-Matching*, the packing solution y defined above is a δ-approximation.*

The covering solution x computed by Alg. 4 is somewhat independent of the order in which Alg. 4 considers constraints. For example, if Alg. 4 does a step for a constraint C_s, and then (immediately) a step for a different constraint $C_{s'}$ that shares no variables with C_s, then doing those steps in the opposite order would not change the final covering solution x. Not surprisingly, it also would not change the packing solution y as defined above. This flexibility is important for the distributed algorithm. The partial order defined next captures this flexibility precisely:

Definition 2 *Let t_i denote the time at which Alg. 4 does a step to cover C_i.*[13] *Let $t_i = 0$ if no step was performed for C_i. Let $i' \prec i$ denote the predicate*

$$\mathsf{vars}(C_{i'}) \cap \mathsf{vars}(C_i) \neq \emptyset \text{ and } 0 < t_{i'} < t_i.$$

(That is, the two constraints share a variable and Alg. 4 did steps for both constraints — the first constraint sometime before the second constraint.)

Let $\mathcal{D}$ be the poset of indices of covering constraints for which Alg. 4 performed a step, (partially) ordered by the transitive closure of "$\prec$".

Sequential Alg. 5 computes the Fractional Packing solution y as defined previously, but considering the variables in an order obtained by reversing *any* total ordering Π of $\mathcal{D}$.[14] The next lemma shows that this still leads to the same packing solution y.

Lemma 5 *Alg. 5 returns the same solution y regardless of the total order Π of $\mathcal{D}$ that it uses in line 2.*

[13] For now, the time at which a step was performed can be thought as the step number t (line 3 at Alg. 4). It will be slightly different in the distributed setting.

[14] The partial order $\mathcal{D}$ is still defined with respect to a particular fixed execution of Alg. 4. The goal is to allow Alg. 4 to do steps for the constraints in any order, thus defining the partial order $\mathcal{D}$, and then to allow Alg. 5 to use any total ordering Π of $\mathcal{D}$.

Proof Observe first that, if indices s' and s are independent in the poset $\mathcal{D}$, then in line 6 of Alg. 5, the value computed for y_s does not depend on $y_{s'}$ (and vice versa). This is because, if it did, there would be a j with $A_{sj} \neq 0$ and $A_{s'j} \neq 0$, so the covering variable x_j would occur in both covering constraints C_s and $C_{s'}$. In this case, since C_s and $C_{s'}$ share a variable, s and s' would be ordered in $\mathcal{D}$.

Consider running Alg. 5 using any total order Π of $\mathcal{D}$. Suppose that, in some iteration t, the algorithm raises a variable y_s ($s = \Pi_t$), and then, in the next iteration, raises a variable $y_{s'}$ ($s' = \Pi_{t-1}$) where $y_{s'}$ is independent of y_s in the poset $\mathcal{D}$. By the previous observation, the value computed for y_s does not depend on $y_{s'}$, and vice versa. Thus, *transposing* the order in which Alg. 5 considers just these two variables (so that Alg. 5 raises $y_{s'}$ in iteration t and y_s in the next iteration, instead of the other way around) would not change the returned vector y. The corresponding total order Π' (Π, but with s and s' transposed) is also a total ordering of $\mathcal{D}$. Thus, for any order Π' that can be obtained from Π by such a transposition, Alg. 5 gives the same solution y.

To complete the proof, observe that *any* total ordering Π' of $\mathcal{D}$ can be obtained from Π by finitely many such transpositions. Consider running Bubblesort on input Π, ordering elements (for the sort) according to Π'. This produces Π' in at most $\binom{T}{2}$ transpositions of adjacent elements. Bubblesort only transposes two adjacent elements s', s, if s' occurs before s in Π but after s in Π', so s and s' must be independent in $\mathcal{D}$. Thus, each intermediate order produced along the way during the sort is a valid total ordering of $\mathcal{D}$. □

Corollary 3 *For both* Fractional Packing *and* Max Weighted c-Matching*, Alg. 5 is δ-approximation algorithm.*

6 Fractional Packing with $\delta = 2$

This section gives a 2-approximation for Fractional Packing with $\delta = 2$ (each variable occurs in at most two constraints).

Distributed model. 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_j for every covering variable x_j. An edge e_i connects vertices u_j and $u_{j'}$ if x_j and $x_{j'}$ belong to the same covering constraint C_i, that is, there exists a constraint $A_{ij}x_j + A_{ij'}x_{j'} \ge w_i$ ($\delta = 2$ so there can be at most 2 variables in each covering constraint).

Distributed 2-approximation algorithm for Fractional Packing with $\delta = 2$ Alg. 6

input: Graph $G = (V, E)$ representing a fractional packing problem instance with $\delta = 2$.
output: Feasible y, 2-approximately minimizing $w \cdot y$.

1. Each edge $e_i \in E$ initializes $y_i \leftarrow 0$.
2. Each edge $e_i \in E$ initializes $done_i \leftarrow$ false. *... this indicates if y_i has been set to its final value*
3. Until each edge e_i has set its variable y_i ($done_i =$ true), perform a round:
4. Perform a round of Alg. 2. *... covering with $\delta = 2$ augmented to compute (t_i^R, t_i^S)*
5. For each node u_r that was a root (in Alg. 2) at any previous round, consider locally at u_r all stars $\mathcal{S}_r^t$ that were rooted by u_r at any previous round t. For each star $\mathcal{S}_r^t$ perform $\mathsf{IncreaseStar}(\mathcal{S}_r^t)$.

$\mathsf{IncreaseStar}$(star $\mathcal{S}_r^t$):

6. For each edge $e_i \in \mathcal{S}_r^t$ in decreasing order of t_i^S:
7. If $\mathsf{IncreasePackingVar}(e_i) =$ NOT DONE then BREAK (stop the for loop).

$\mathsf{IncreasePackingVar}$(edge $e_i = (u_j, u_r)$):

8. If e_i or any of its adjacent edges has a non-yet-satisfied covering constraint return NOT DONE.
9. If $t_i^R = 0$ then:
10. Set $y_i \leftarrow 0$ and $done_i \leftarrow$ true.
11. Return DONE.
12. If $done_{i'} =$ false for any edge $e_{i'}$ such that $i \prec i'$ then return NOT DONE.
13. Set $y_i \leftarrow \min\left\{(c_j - \sum_{i'} A_{i'j} y_{i'})/A_{ij},\ (c_r - \sum_{i'} A_{i'r} y_{i'})/A_{ir}\right\}$ and $done_i \leftarrow$ true.
14. Return DONE.

Algorithm. Recall Alg. 2, which 2-approximates CMIP_2 in $O(\log m)$ rounds. Fractional Covering with $\delta = 2$ is the special case of CMIP_2 with no integer variables and no upper bounds. Thus, Alg. 2 also 2-approximates Fractional Covering with $\delta = 2$ in $O(\log m)$ rounds.

Using the results in the previous section, this section extends Alg. 2 (as it specializes for Fractional Covering with $\delta = 2$) to compute a solution for the dual problem, Fractional Packing with $\delta = 2$, in $O(\log m)$ rounds.

Alg. 2 proceeds in rounds, and within each round it covers a number of edges. Define the time at which a step to cover constraint C_i (edge e_i) is done as a pair (t_i^R, t_i^S), where t_i^R denotes the round in which the step was performed and t_i^S denotes that within the star this step is the t_i^S-th one. Let $t_i^R = 0$ if no step was performed for C_i. Overloading Definition 2, define "$\prec$" as follows.

Definition 3 *Let $i' \prec i$ denote that $\mathsf{vars}(C_{i'}) \cap \mathsf{vars}(C_i) \neq \emptyset$ (i' and i are adjacent edges in the network) and the pair $(t_{i'}^R, t_{i'}^S)$ is lexicographically less than (t_i^R, t_i^S) (but $t_{i'}^R > 0$).*

For any two *adjacent* edges i and i', the pairs (t_i^R, t_i^S) and $(t_{i'}^R, t_{i'}^S)$ are adequate to distinguish which edge 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_i^S. Otherwise they are done in different rounds, so they differ in t_i^R. Thus the pair (t_i^R, t_i^S) and relation "$\prec$" define a partially ordered set $\mathcal{D}$ of all edges for which Alg. 2 did a step.

The extended algorithm, Alg. 6, runs Alg. 2 to compute a cover x, recording the poset $\mathcal{D}$. Meanwhile, it computes a packing y as follows: as it discovers $\mathcal{D}$, it starts raising packing variable as soon as it can (while emulating Alg. 5). Specifically it sets a given $y_i \in \mathcal{D}$ as soon as (a) Alg. 2 has done a step for the covering constraint C_i, (b) the current cover x satisfies each adjacent covering constraint $C_{i'}$, and (c) for each adjacent $C_{i'}$ for which Alg. 2 did a step *after* C_i, the variable $y_{i'}$ has already been set. When it sets the packing variable y_i, it does so following Alg. 5: it raises y_i maximally subject to the packing constraint $A^\intercal y \leq c$.

Some nodes will be executing the second phase of the algorithm (computing the y_i's) while some other nodes are still executing the first phase (computing the x_j's). This is necessary because a given node cannot know when distant nodes are done computing x.

Theorem 8 *For* Fractional Packing *where each variable appears in at most two constraints ($\delta = 2$) 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 Thm. 4, Alg. 2 computes a covering solution x in $T = O(\log m)$ rounds in expectation and with high probability. Then by a straightforward induction on t, within $T + t$ rounds, for every constraint C_i for which Alg. 2 did a step in round $T - t$, Alg. 6 will have set the variable y_i. Thus, Alg. 6 computes the entire packing y in $2T = O(\log m)$ rounds with in expectation and with high probability.

As Alg. 2 performs steps for covering constraints, order the indices of those constraints by the time in

which the constraints' steps were done, breaking ties arbitrarily. Let Π be the resulting order.

As Alg. 6 performs steps for packing variables, order the indices of those variables by the time in which the variables were raised, breaking ties arbitrarily. Let Π' be the reverse of this order.

Then the poset $\mathcal{D}$ is the same as it would be if defined by executing the sequential algorithm Alg. 4 for the Fractional Covering problem and considering the constraints in the order in which their indices occur in Π. Also, the covering solution x is the same as would be computed by Alg. 4.

Likewise, Π' is a total ordering of $\mathcal{D}$, and the packing solution y is the same as would be computed by Alg. 5 using the order Π'.

By Lemmas 3 and 5, and weak duality, the Fractional Packing solution y is 2-approximate. □

The following corollary is a direct result of Lemma 4 and Thm. 8 and the fact that for this problem $|E| = O(|V|^2)$.

Corollary 4 *There is a distributed* 2*-approximation algorithm for* Max Weighted c-Matching *on graphs running in* $O(\log |V|)$ *rounds in expectation and with high probability.*

7 Fractional Packing with general δ

Distributed model. Here we assume that the distributed network has a node v_i for each covering constraint C_i (packing variable y_i), with edges from v_i to each node $v_{i'}$ if C_i and $C_{i'}$ share a covering variable x_j. [15]. 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 6.

Algorithm. Fix any Fractional Packing instance and its Fractional Covering dual:

$$\textit{maximize } w \cdot y \textit{ subject to } y \in \mathbb{R}^m_+ \textit{ and } A^{\mathsf{T}} y \le c,$$
$$\textit{minimize } c \cdot x \textit{ subject to } x \in \mathbb{R}^n_+ \textit{ and } Ax \ge w.$$

Recall Alg. 3, which δ-approximates Submodular-cost Covering in $O(\log^2 m)$ rounds. Fractional Covering (with general δ) is a special case. Thus, Alg. 3 also δ-approximates Fractional Covering in $O(\log^2 m)$ rounds.

For this special case, make Alg. 3 use the particular stepsize() function defined in Alg. 4 so that the specialization of Alg. 3 defined thusly is also a distributed implementation of Alg. 4.

[15] The computation can easily be simulated on a network with nodes for covering variables or nodes for covering variables and covering constraints.

Following the approach in the previous two sections, this section extends this specialization of Alg. 3 for Fractional Covering to compute a solution for Fractional Packing, the dual problem, in $O(\log^2 m)$ rounds.

Similar to the $\delta = 2$ case in the previous section, Alg. 3 defines a poset $\mathcal{D}$ of the indices of covering constraints; the extended algorithm sets raises packing variables maximally, in a distributed way consistent with some total ordering of $\mathcal{D}$. Here, the role of stars is substituted by components and the role of roots by leaders. With each step done to satisfy the covering constraints C_i, the algorithm records (t_i^R, t_i^S), where t_i^R is the round and t_i^S is the within-the-component iteration in which the step was performed. This defines a poset $\mathcal{D}$ on the indices of covering constraints for Alg. 3 performs steps.

The extended algorithm, Alg. 7, runs the specialized version of Alg. 3 on the Fractional Covering instance, recording the poset $\mathcal{D}$ (that is, recording (t_i^R, t_i^S) for each covering constraint C_i 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 sets a given $y_i \in \mathcal{D}$ once (a) a step has been done for the covering constraint C_i, (b) each adjacent covering constraint $C_{i'}$ is satisfied and (c) for each adjacent $C_{i'}$ for which a step was done after C_i, the variable $y_{i'}$ has been set.

To implement this, 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 y_i in order of decreasing t_i^S. When considering y_i it checks if each $y_{i'}$ with $i \prec i'$ is set, and, if so, the algorithm sets y_i and continues with the next component's packing variable (in order of decreasing t_i^S).

Theorem 9 *For* Fractional Packing *where each variable appears in at most* δ *constraints there is a distributed* δ*-approximation algorithm running in* $O(\log^2 m)$ *rounds in expectation and with high probability, where* m *is the number of packing variables.*

Proof The proofs of correctness and running time are essentially the same as in the proofs of Thm. 8, except that in this case $T = O(\log^2 m)$ in expectation and w.h.p. (by Thm. 6), so the running time is $O(\log^2 m)$. □

The following corollary is a direct result of Lemma 4 and Thm. 9.

Corollary 5 *For* Max Weighted c-Matching *on hypergraphs, there is a distributed* δ*-approximation algorithm running in* $O(\log^2 |E|)$ *rounds in expectation*

Distributed δ-approximation algorithm for Fractional Packing with general δ Alg. 7

input: Graph $G = (V, E)$ representing a fractional packing problem instance.
output: Feasible y, δ-approximately minimizing $w \cdot y$.

1. Initialize $y \leftarrow 0$.
2. For each $i = 1 \ldots m$ initialize $done_i \leftarrow$ false. *... this indicates if y_i has been set to its final value*
3. Until each y_i has been set ($done_i =$ true) do:
4. Perform a phase of the δ-approximation algorithm for covering (Alg. 3), recording (t_i^R, t_i^S).
5. For each node $v_{\mathcal{K}}$ that was a leader at any previous phase, consider locally at $v_{\mathcal{K}}$ all components that chose $v_{\mathcal{K}}$ as a leader at any previous phase. For each such component $\mathcal{K}_r$ perform $\mathsf{IncreaseComponent}(\mathcal{K}_r)$.

$\mathsf{IncreaseComponent}$(component $\mathcal{K}_r$):

6. For each $i \in \mathcal{K}_r$ in decreasing order of t_i^S:
7. If $\mathsf{IncreasePackingVar}(i) =$ NOT DONE then BREAK (stop the for loop).

$\mathsf{IncreasePackingVar}(i)$:

8. If C_i or any $C_{i'}$ that shares covering variables with C_i is not yet satisfied return NOT DONE.
9. If $t_i^R = 0$ then:
10. Set $y_i = 0$ and $done_i =$ true.
11. Return DONE.
12. If $done_{i'} =$ false for any $y_{i'}$ such that $i \prec i'$ then return NOT DONE.
13. Set $y_i \leftarrow \min_{j \in \mathsf{cons}(y_i)} \left((c_j - \sum_{i'} A_{i'j} y_{i'}) / A_{ij} \right)$ and $done_i \leftarrow$ true.
14. Return DONE.

and with high probability, where δ is the maximum hyperedge degree and $|E|$ is the number of hyperedges.

Acknowledgements

Thanks to two anonymous referees for their helpful comments.

Appendix

Generate a δ-approximate primal-dual pair for the greedy algorithm for Fractional Covering (Alg. 4 in Section 5) in some sense requires a tail-recursive approach. The following example demonstrates this. Consider (i) $\min\{x_1 + x_2 + x_3 : x_1 + x_2 \geq 1,\ x_1 + x_3 \geq 5,\ x \geq 0\}$.

If the greedy algorithm (Alg. 4) does the constraints of (i) in either order (choosing β maximally), it gives a solution of cost 10.

The dual is $\max\{y_{12} + 5y_{13} : y_{12} + y_{13} \leq 1,\ y \geq 0\}$. The only way to generate a dual solution of cost 5 is to set $y_{12} = 0$ and $y_{13} = 1$.

Now consider the covering problem (ii) $\min\{x_1 + x_2 + x_3 : x_1 + x_2 \geq 1,\ x_1 + x_3 \geq 0,\ x \geq 0\}$ (the right-hand-side of the second constraint is 0).

If the greedy algorithm does the constraints of (ii) in either order (choosing β maximally), it gives a solution of cost 2.

The dual is $\max\{y_{12} : y_{12} + y_{13} \leq 1,\ y \geq 0\}$. The only way to generate a dual solution of cost 1 is to set $y_{12} = 1$ and $y_{12} = 0$ (the opposite of the y for (i)).

Consider running Alg. 4 on each problem, and giving it the shared constraint $x_1 + x_2 \geq 1$ first. This constraint (and the cost) are the same in both problems, and the algorithm sets x to satisfy the constraint in the same way for both problems. A standard primal-dual approach would set the dual variable y_{12} at this point, as a function of the treatment of x_1 and x_2 (essentially in an *online* fashion, constraint by constraint). That can't work here: in order to get a 2-approximate dual solution, the dual variable y_{12} has to be set differently in the two instances: to 0 for the first instance, and to 1 for the second instance.

References

1. B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Low-diameter graph decomposition is in NC. *Random Structures and Algorithms*, 5(3):441–452, 1994.
2. N. Bansal, N. Korula, V. Nagarajan, and A. Srinivasan. On k-column sparse packing programs. *In Proceedings of MPS Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 369–382, 2010.
3. R. Bar-Yehuda, K. Bendel, A. Freund, and D. Rawitz. Local ratio: a unified framework for approximation algorithms. *ACM Computing Surveys*, 36(4):422–463, 2004.
4. R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, 2(2):198–203, 1981.
5. R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25(27-46):50, 1985.
6. R. Bar-Yehuda and D. Rawitz. On the equivalence between the primal-dual schema and the local-ratio technique. *SIAM Journal on Discrete Mathematics*, 19(3):762–797, 2005.
7. D. Bertsimas and R. Vohra. Rounding algorithms for covering problems. *Mathematical Programming: Series A and B*, 80(1):63–89, 1998.

8. R. D. Carr, L. K. Fleischer, V. J. Leung, and C. A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *the eleventh ACM-SIAM Symposium On Discrete Algorithms*, pages 106–115, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.
9. Z.-Z. Chen. A fast and efficient nc algorithm for maximal matching. *Information Processing Letters*, 55:303–307, 1995.
10. V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.
11. A. Czygrinow, M. Hańćkowiak, and E. Szymańska. A fast distributed algorithm for approximating the maximum matching. *In the twelfth European Symosium on Algorithms*, pages 252–263, 2004.
12. A. Czygrinow, M. Hańćkowiak, and W. Wawrzyniak. Distributed packing in planar graphs. *In the twentieth ACM Symposium on Parallel Algorithms and Architectures*, pages 55–61, 2008.
13. J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.
14. J. Edmonds and E. L. Johnson. Matching: A well-solved class of integer linear programs. *Combinatorial Structures and Their Applications*, pages 89–92, 1970.
15. F. Grandoni, J. Könemann, and A. Panconesi. Distributed weighted vertex cover via maximal matchings. *ACM Transactions on Algorithms (TALG)*, 5(1):1–12, 2008.
16. F. Grandoni, J. Könemann, A. Panconesi, and M. Sozio. Primal-dual based distributed algorithms for vertex cover with semi-hard capacities. *In the twenty-fourth ACM symposium on Principles Of Distributed Computing*, pages 118–125, 2005.
17. F. Grandoni, J. Könemann, A. Panconesi, and M. Sozio. A primal-dual bicriteria distributed algorithm for capacitated vertex cover. *SIAM Journal on Computing*, 38(3):825–840, 2008.
18. F. Grandoni, J. Könemann, J., and A. Panconesi. Distributed weighted vertex cover via maximal matchings. *Lecture Notes in Computer Science*, 3595:839–848, 2005.
19. N.G. Hall and D.S. Hochbaum. A fast approximation algorithm for the multicovering problem. *Discrete Applied Mathematics*, 15(1):35–40, 1986.
20. M. Hańćkowiak, M. Karonski, and A. Panconesi. On the distributed complexity of computing maximal matchings. *SIAM Journal of Discrete Mathematics*, 15(1):41–57, 2001.
21. D. S. Hochbaum. Efficient bounds for the stable set, vertex cover, and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.
22. D.S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11:555–556, 1982.
23. J.H. Hoepman. Simple distributed weighted matchings. *Arxiv preprint cs.DC/0410047*, 2004.
24. S. Hougardy and D.E. Vinkemeier. Approximating weighted matchings in parallel. *Information Processing Letters*, 99(3):119–123, 2006.
25. A. Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22:77–80, 1986.
26. L. Jia, R. Rajaraman, and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *In the twentieth ACM symposium on the Principles Of Distributed Computing*, pages 33–42, 2001.
27. D. S. Johnson. Approximation algorithms for combinatorial problems. *In the fifth ACM Symposium On Theory Of Computing*, 25:38–49, 1973.
28. R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds., The IBM Research Symposia Series, New York, NY: Plenum Press:85–103, 1972.
29. R.M. Karp. Probabilistic recurrence relations. *Journal of the ACM (JACM)*, 41(6):1136–1150, 1994.
30. P. Kelsen. An optimal parallel algorithm for maximal matching. *Information Processing Letters*, 52:223–228, 1994.
31. S. Khuller, U. Vishkin, and N.E. Young. A primal-dual parallel approximation technique applied to weighted set and vertex covers. *Journal of Algorithms*, 17:280–289, 1994.
32. S.G. Kolliopoulos and N.E. Young. Approximation algorithms for covering/packing integer programs. *Journal of Computer and System Sciences*, 71(4):495–505, 2005.
33. C. Koufogiannakis and N.E. Young. Beating simplex for fractional packing and covering linear programs. *In the forty-eighth IEEE symposium on Foundations of Computer Science*, pages 494–504, 2007.
34. C. Koufogiannakis and N.E. Young. Distributed and parallel algorithms for weighted vertex cover and other covering problems. *the twenty-eighth ACM symposium Principles of Distributed Computing*, pages 171–179, 2009.
35. C. Koufogiannakis and N.E. Young. Distributed fractional packing and maximum weighted b-matching via tail-recursive duality. *the twenty-third International Symposium on Distributed Computing. Lecture Notes in Computer Science*, LNCS 5805:221–238, 2009.
36. C. Koufogiannakis and N.E. Young. Greedy Δ-approximation algorithm for covering with arbitrary constraints and submodular cost. *In the thirty-sixth International Colloquium on Automata, Languages and Programming*, LNCS 5555:634–652, 2009. See also http://arxiv.org/abs/0807.0644.
37. F. Kuhn and T. Moscibroda. Distributed approximation of capacitated dominating sets. *In the nineteenth ACM Symposium on Parallelism in Algorithms and Architectures*, pages 161–170, 2007.
38. F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! *In the twenty-third ACM symposium on Principles Of Distributed Computing*, pages 300–309, 2004.
39. F. Kuhn, T. Moscibroda, and R. Wattenhofer. The price of being near-sighted. *In the seventeenth ACM-SIAM Symposium On Discrete Algorithm*, pages 980–989, 2006.
40. F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. *In the twenty-second ACM symposium on the Principles Of Distributed Computing*, pages 25–32, 2003.
41. C. Lenzen, Y.A. Oswald, and R. Wattenhofer. What can be approximated locally?: case study: dominating sets in planar graphs. *In the twentieth ACM Symposium on Parallel Algorithms and Architectures*, pages 46–54, 2008.
42. N. Linial. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21:193–201, 1992.
43. N. Linial and M. Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993.
44. Z. Lotker, B. Patt-Shamir, and S. Pettie. Improved distributed approximate matching. *In the twelfth ACM Symposium on Parallelism in Algorithms and Architectures*, pages 129 –136, 2008.
45. Z. Lotker, B. Patt-Shamir, and A. Rosén. Distributed approximate matching. *In the twenty-sixth ACM symposium on Principles Of Distributed Computing*, pages 167–174, 2007.

46. L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Math*, 13:383–390, 1975.
47. N. Luby. A simple parallel algorithm for the maximal independent set problem. *In the seventh ACM Symposium on Theory Of Computing*, pages 1–10, 1985.
48. M. Müller-Hannemann and A. Schwartz. Implementing weighted b-matching algorithms: Towards a flexible software design. *In the Workshop on Algorithm Engineering and Experimentation (ALENEX)*, pages 18–36, 1999.
49. M. Naor and L. Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24:1259–1277, 1995. STOC' 93.
50. T. Nieberg. Local, distributed weighted matching on general and wireless topologies. *In the fifth ACM Joint Workshop on the Foundations of Mobile Computing, DIALM-POMC*, pages 87–92, 2008.
51. A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14:97–100, 2001.
52. A. Panconesi and A. Srinivasan. On the complexity of distributed network decomposition. *Journal of Algorithms*, 20(2):356–374, 1996.
53. Alessandro Panconesi and Mauro Sozio. Fast primal-dual distributed algorithms for scheduling and matching problems. *Distributed Computing*, 22:269–283, 2010.
54. D. Peleg. Distributed computing: a locality-sensitive approach. *Society for Industrial and Applied Mathematics*, 2000.
55. David Pritchard. Approximability of sparse integer programs. *In the seventeenth European Symposium on Algorithms*, Lecture Notes in Computer Science 5757:83–94, 2009.
56. A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM Journal on Computing*, 29:648–670, 1999.
57. A. Srinivasan. New approaches to covering and packing problems. *In the twelveth ACM-SIAM Symposium On Discrete Algorithms*, pages 567–576, 2001.
58. R. Uehara and Z. Chen. Parallel approximation algorithms for maximum weighted matching in general graphs. *Information Processing Letters*, 76(1-2):13–17, 2000.
59. M. Wattenhofer and R. Wattenhofer. Distributed weighted matching. *In the eighteenth international symposium on Distributed Computing*, pages 335–348, 2004.