Distributed and Parallel Algorithms for Weighted Vertex Cover and other Covering Problems

Christos Koufogiannakis
Dept. of Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521, USA
ckou@cs.ucr.edu

Neal E. Young[*]
Dept. of Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521, USA
neal@cs.ucr.edu

ABSTRACT

The paper presents distributed and parallel δ-approximation algorithms for covering problems, where δ is the maximum number of variables on which any constraint depends (for example, $\delta = 2$ for VERTEX COVER).

Specific results include the following.

• For WEIGHTED VERTEX COVER, the first distributed 2-approximation algorithm taking $O(\log n)$ rounds and the first parallel 2-approximation algorithm in RNC. The algorithms generalize to covering mixed integer linear programs (CMIP) with two variables per constraint ($\delta = 2$).

• For any covering problem with monotone constraints and submodular cost, a distributed δ-approximation algorithm taking $O(\log^2 |\mathcal{C}|)$ rounds, where $|\mathcal{C}|$ is the number of constraints. (Special cases include CMIP, facility location, and probabilistic (two-stage) variants of these problems.)

Categories and Subject Descriptors

G.2.2 [**DISCRETE MATHEMATICS**]: Graph Theory—*Graph algorithms, Hypergraphs*; F.2.2 [**ANALYSIS OF ALGORITHMS AND PROBLEM COMPLEXITY**]: Nonnumerical Algorithms and Problems—*Computations on discrete structures*

General Terms

Algorithms Theory

Keywords

distributed vertex cover, distributed covering

[*]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 [29, 34, 36, 24, 25, 7]. 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 DOMINATING SET (SET COVER) [17, 26, 27], CAPACITATED DOMINATING SET [23], CAPACITATED VERTEX COVER [8, 9], WEIGHTED MATCHING [39, 15, 31, 30] and many other problems (some discussed below).

Covering problems are of broad interest in the field of approximation algorithms. This work studies distributed δ-approximation algorithms for MONOTONE COVERING problems, that is, problems of the form

Find $x \in \mathbb{R}_+^n$ *minimizing* $c(x)$ *subject to* $(\forall S \in \mathcal{C})\ x \in S$.

The cost function $c : \mathbb{R}_+^n \rightarrow \mathbb{R}_+$ must be non-decreasing, continuous, and submodular[1] Each constraint set $S \in \mathcal{C}$ must be monotone (closed upwards) and closed under limit [22]. Here δ is the maximum number of variables (in x) on which any constraint $x \in S$ depends.

Although x ranges over $\mathbb{R}_+^n$, one can handle restrictions on the variable domains by incorporating them into the constraints [22].

Special cases include: CIP (covering integer programs with variable upper bounds, of the form $\min\{c \cdot x : Ax \geq b; x \in \mathbb{Z}_+^m, x \leq u\}$); CMIP (covering mixed integer linear programs — CIP with both integer and fractional variables); FACILITIES LOCATION; WEIGHTED SET COVER (CIP with $A_{ij} \in \{0, 1\}, b_i = 1$); WEIGHTED VERTEX COVER (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. [18, 32, 6, 37, 38, 21]), and (ii) $O(\delta)$-approximation algorithms where δ is the maximum number of variables in any constraint (e.g. [1, 13, 14, 2, 11, 4, 3, 22]), including most famously 2-approximation algorithms for WEIGHTED VERTEX COVER.

Our focus is on distributed algorithms in the second class.

[1]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.

Distributed δ-approximation algorithms. In the distributed setting, in the case of UNWEIGHTED VERTEX COVER (vertices have uniform weights), a 2-approximate solution can be found efficiently by computing any maximal matching and then taking the cover to contain the endpoints of the edges in the matching. A maximal matching can be computed deterministically in $O(\log^4 n)$ rounds using the algorithm of Hańćkowiak, Karonski and Panconesi [12] or in $O(\Delta + \log^* n)$ rounds using the algorithm of Panconesi and Rizzi [35], where Δ is the maximum vertex degree. Using randomization, a maximal matching can be computed in an expected $O(\log n)$ rounds via the algorithm of Israeli and Itai [16]. Maximal matching is also in NC [5, 33, 19] and in RNC (parallel poly-log time with polynomially many randomized processors) [16] and so is 2-approximate UNWEIGHTED VERTEX COVER.

For WEIGHTED VERTEX COVER, no such result is known. The first distributed approximation algorithm for WEIGHTED VERTEX COVER (and weighted SET COVER) appeared in 1994 by Khuller, Vishkin and Young [20]. It takes $O(\delta \log n \log 1/\varepsilon)$ rounds to produce a $\delta(1+\varepsilon)$-approximation ($\delta = 2$ for WEIGHTED VERTEX COVER). Assuming the vertex weights are integers, taking $\varepsilon = 1/(n\hat{C}+1)$, where $\hat{C}$ is the average vertex weight, the algorithm can compute a δ-approximate cover in $O(\delta \log n(\log n + \log \hat{C}))$ rounds.

Grandoni, Konemann and Panconesi give a distributed 2-approximation algorithm for WEIGHTED VERTEX COVER that takes $O(\log n + \log \hat{C})$ rounds [10]. Their algorithm assumes integer vertex weights.

As noted in [24], neither algorithm takes a number of rounds that is poly-logarithmic in the number of vertices. Also, there was no previously known *parallel* 2-approximation NC or RNC algorithm (running in time poly-logarithmic in the number of vertices with polynomially many processors).

Kuhn, Moscibroda and Wattenhofer describe distributed approximation algorithms for *fractional* covering (and packing) linear programs [25]. Their algorithms give constant-factor approximations in $O(\log |\mathcal{C}|)$ rounds, where $|\mathcal{C}|$ is the number of covering constraints. The approximation ratio is greater than 2 for the case of 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.

The best known lower bounds for WEIGHTED VERTEX COVER are by Kuhn, Moscibroda and Wattenhofer [24]. They prove that to achieve even a polylogarithmic approximation ratio for VERTEX COVER, the number of rounds required is at least $\Omega(\sqrt{\log n / \log\log n})$ and $\Omega(\log \Delta / \log\log \Delta)$.

Results. For MONOTONE COVERING, we give distributed δ-approximation algorithms that are *efficient*, in that they finish in a number of rounds that is poly-logarithmic in the network size [29]:

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. This algorithm is easily parallelized, giving the first 2-approximation algorithm for WEIGHTED VERTEX COVER in RNC.

Section 3, generalizing the above result, describes the first efficient distributed 2-approximation algorithm for CMIP (covering mixed integer linear programs with variable upper bounds) where each constraint depends on at most two variables ($\delta = 2$). The algorithm runs in $O(\log |\mathcal{C}|)$ rounds in expectation and with high probability, where $|\mathcal{C}|$ is the number of constraints. This also gives a parallel (RNC) 2-approximation algorithm.

Section 4 describes the first efficient distributed δ-approximation algorithm for MONOTONE COVERING. The algorithm runs in $O(\log^2 |\mathcal{C}|)$ rounds in expectation and with high probability, where $|\mathcal{C}|$ is the number of constraints.

Special cases include CMIP, FACILITIES LOCATION, WEIGHTED SET COVER, WEIGHTED VERTEX COVER (WEIGHTED SET COVER with $\delta = 2$), and probabilistic (two-stage stochastic) variants of these problems. Previously, no efficient distributed $O(\delta)$-approximation algorithm was known for CIP, and no efficient distributed δ-approximation algorithm was known even for WEIGHTED VERTEX COVER.

Each of the algorithms we present here is a distributed implementation of a (centralized) δ-approximation algorithm for MONOTONE COVERING by Koufogiannakis and Young [22]. In each section, we describe how that centralized algorithm specializes for the problem in question, then describe an efficient distributed implementation.

2. WEIGHTED VERTEX COVER

THEOREM 1. *For* WEIGHTED VERTEX COVER*:*

(a) There is a distributed 2-approximation algorithm running in $O(\log n)$ rounds in expectation and with high probability.

(b) There is a parallel 2-approximation algorithm in "Las Vegas" RNC.

Centralized algorithm

For WEIGHTED VERTEX COVER, the centralized algorithm by Koufogiannakis and Young [22] is equivalent to a classical 2-approximation algorithm by Bar-Yehuda et al. [1].

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 stops when all edges are covered.

The algorithm returns a 2-approximate cover by the following argument. Fix any optimal integer solution x^*. Each call to $\mathsf{step}(x,e)$ increases the cost of x by at most 2β but decreases the potential $\sum_v c_v \max(0, x^*_v - x_v)$ by at least β.

Distributed implementation

In each round, the distributed algorithm performs $\mathsf{step}(x,e)$ simultaneously on a large subset of the not-yet-covered edges, as follows. Each node 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.

```
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 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 step$(x, (v, w))$.
           tails($w$): Do step$(x, (v, w))$ just for the last edge for which heads$(w)$ would do step$(x, (v, w))$.

step$(x, (v, w))$:
1. 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
2. Set $x_v = x_v + \beta/c_v$. If $x_v = 1$, add $v$ to the cover, covering all of $v$'s edges.
3. Set $x_w = x_w + \beta/c_w$. If $x_w = 1$, add $w$ to the cover, covering all of $w$'s edges.
```

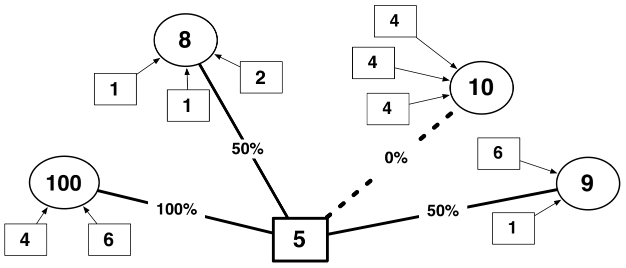

Figure 1: Each node is labeled with its cost. Roots are circles; leaves are squares; star edges from leaves other than v (the cost-5 leaf) are determined as shown. Each edge (v, w) is labeled with the chance that v would enter the cover if v were to choose (v, w) for its star edge (assuming each $x_w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

Each root w then flips a coin. If heads comes up (with probability 1/2), w does heads(w): that is, it does 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 does tails(w): that is, it simulates heads(w), without actually doing any steps, to determine the *last* edge (v, w) that heads(w) would do a step for, and performs step step$(x, (v, w))$ for just *that* edge. For details see Alg. 1.

Analysis of the number of rounds. We show that, in each round, a constant fraction of the not-yet-covered edges are covered in expectation, proving part (a) of the theorem.

Any not-yet-covered edge (v, w) is active for the round with constant probability, because 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. So, with constant probability a constant fraction of the remaining edges are active. Assume this happens. 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 a constant fraction of v's active edges will be covered. 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: *A constant fraction 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* heads(w)*, then* heads(w) *would* not *perform* step$(x, (v, w))$. That is, w would enter the cover before heads(w) would consider (v, w) (in Fig. 1, see the cost-10 node).

For such an edge (v, w), on consideration, heads(w) will bring w into the cover *whether or not* v chooses (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 heads(w). Since w does heads(w) with probability 1/2, edge (v, w) will be covered with probability 1/2.

Since this is true for a constant fraction of v's active edges, in expectation, a constant fraction of v's active edges will be covered during the round.
Case 2: *A constant fraction 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* heads(w)*, then* heads(w) would *perform* step$(x, (v, w))$.

For such an edge (v, w), heads(w) would bring v into the cover as long as step$(x, (v, w))$ would not be the *last* step performed by heads(w) (in Fig. 1, the cost-8 and cost-100 nodes). Or, if step$(x, (v, w))$ would be the last step performed by heads(w), then tails(w) would do *only* 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 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 heads(w) or tails(w) would bring v into the cover. Recall that w has a 50% chance of doing heads(w) and a 50% chance of doing tails(w). Thus, if v chooses such an edge, v enters the cover with at least a 50% chance.

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

Thus, in each round, in expectation a constant fraction of the remaining edges are covered. By standard arguments, this implies that the number of rounds is $O_d(\log n)$, both in expectation and with probability $1 - 1/n^d$, for any fixed $d > 0$. This completes the proof of Thm. 1, part (a).

Parallel (RNC) implementation

Proof of Thm. 1, part (b). To obtain the parallel algorithm, implement heads(w) as follows. For w's kth star edge e_k, let β_k be the β that step(x, e_k) would use if given x at the *start* of the round. If heads(w) eventually does 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 $O_d(\log n)$ rounds; if the simulated algorithm halts, the RNC algorithm returns x, and otherwise it returns "fail". This completes the proof of Thm. 1.

3. MIXED INTEGER PROGRAMS (TWO VARIABLES PER CONSTRAINT)

Next we generalize the results of Section 2 to CMIP$_2$ (CMIP with at most two non-zero coefficients A_{ij} in each constraint).

THEOREM 2. *For* COVERING MIXED INTEGER LINEAR PROGRAMS *with at most two variables per constraint (CMIP$_2$):*

(a) there is a distributed 2-approximation 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 2-approximation algorithm in "Las Vegas" RNC.

Centralized algorithm

First we describe an implementation of the centralized δ-approximation algorithm for MONOTONE COVERING [22] for the special case of CMIP$_2$.

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 $A_i x \ge b$ by the following equivalent monotone constraint S_i:

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

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

Once all constraints are satisfied, the algorithm rounds each x_j down to $\lfloor \min(x_j, u_j) \rfloor$ and returns the rounded x.

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

For the algorithm to produce a 2-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 $\mathsf{distance}_c(x, S) = \min\{c(\hat{x}) - c(x) | \hat{x} \in S, \hat{x} \ge x\}$:

OBSERVATION 1. *([22]) If $\mathsf{stepsize}(x, S) \le \mathsf{distance}_c(x, S)$ in each step, and the algorithm above terminates, then it returns a 2-approximate solution.*

(*Partial proof:* Let x^* be any optimal solution. A step increases the cost $c \cdot x$ by 2β, but decreases the potential $\sum_{v\in V} c_v \max(0, x^*_v - x_v)$ by at least β. Details in [22].)

Compute $\mathsf{stepsize}(x, S_i)$ as follows. Consider any relaxation 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 two variables per constraint there are at most sixteen 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 $\mathsf{stepsize}(x, S_i)$ (in constant time) as *the minimum cost to increase just one variable enough to reduce $\Phi(x, S_i)$.*

OBSERVATION 2. *With this* $\mathsf{stepsize}()$, $\mathsf{step}(x, S_i)$ *is done at most sixteen times before constraint S_i is satisfied.*

Also, this step size satisfies the condition in Observation 1:

LEMMA 1. $\mathsf{stepsize}(x, S_i) \le \mathsf{distance}_c(x, S_i)$

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

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

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 the sum $\sum_{j\in J'} A_{ij} x_j$ 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, $\mathsf{stepsize}(x, S_i) \le \mathsf{distance}(x, S_i')$. It follows that $\mathsf{stepsize}(x, S_i) \le \mathsf{distance}(x, S_i') \le \mathsf{distance}(x, S_i)$. $\square$

Example. Minimize $x_1 + x_2$ subject to $0.5x_1 + 3x_2 \ge 5$, $x_2 \le 1$, 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:

$$0.5x_1 + 3x_2 \ge 5 \qquad (1)$$
$$0.5x_1 + 3\lfloor x_2 \rfloor \ge 5 \qquad (2)$$
$$0.5x_1 + 3\min\{x_2, 1\} \ge 5 \qquad (3)$$
$$0.5x_1 + 3\lfloor \min\{x_2, 1\} \rfloor \ge 5 \qquad (4)$$
$$0.5\lfloor x_1 \rfloor + 3x_2 \ge 5 \qquad (5)$$
$$0.5\lfloor x_1 \rfloor + 3\lfloor x_2 \rfloor \ge 5 \qquad (6)$$
$$0.5\lfloor x_1 \rfloor + \min\{x_2, 1\} \ge 5 \qquad (7)$$
$$0.5\lfloor x_1 \rfloor + 3\lfloor \min\{x_2, 1\} \rfloor \ge 5 \qquad (8)$$

At the begining, $x_1 = x_2 = 0$. 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 $x_2 = \lfloor \min\{x_2, 1\} \rfloor = 1$.

Related $\mathsf{stepsize}()$ functions for general CMIP are in [22].

Distributed implementation

To prove part (a) of Thm. 2, we describe a distributed implementation of the above centralized algorithm. 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 were likely to be covered each round because a step done for one edge could cover not just that edge, but many others also. Here we take a similar approach. Recall the definition of $\Phi(x, S)$ in the definition of $\mathsf{stepsize}()$. We want the total potential of all constraints, $\Phi(x) = \sum_S \Phi(x, S)$, to 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).*

We want a constant fraction of the unmet constraints to be hit in each round.

Note that the observation implies, for example, that, among constraints S 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 $\mathsf{stepsize}(x, S') \le \mathsf{stepsize}(x, S)$).

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 $x_w \ge t_S$. At that point, each of w's not-yet-hit star constraints S has $t_S \le x_w$, 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.

Analysis of the number of rounds

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 one fourth, 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 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 $\mathsf{heads}(w)$ wouldn't 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 $x_w \ge t_S$, 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 $\mathsf{stepsize}(x, S_r)/c_w \ge \mathsf{stepsize}(x, S)/c_w$).

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 a constraint S' with a root other than w as its star constraint. (Or, if v chooses a constraint $S' \ne S$ 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 $h(S) \ge h(S')$. 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 $h(S') \ge h(S)$ 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).

```
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 step$(x, S)$ for the one maximizing 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.    At each leaf $v$: For each unmet constraint $S$ that can be hit by $x_v$ (per 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.    At each root node $w$: do either heads$(w)$ or tails$(w)$ below, each with probability $1/2$.

heads$(w)$
1. 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 $t_S = \infty$.
2. For each star constraint $S$ rooted at $w$, in order of decreasing $t_S$, do the following:
       If $x_w < t_S$ then do 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 stepsize$(x, S_r)$. Do step$(x, S_r)$ (hitting $S_r$ and all not-yet-hit star constraints rooted at $w$).

tails$(w)$
1. Determine which constraint $S_r$ would be the runt in heads$(w)$. Do step$(x, S_r)$.
```

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 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 $h(S) \le h(S')$ and the root w of S also does 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* tails(w)*, then* tails(w) *would do* 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 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 tails(w)).

Let S and S' be any two constraints in $\mathcal{T}$ where $t(S') \ge t(S)$. 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 tails(w'), then (because x_v increases to at least $t(S') \ge t(S)$) 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 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 $t(S) \le t(S')$).

This proves the lemma. □

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 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. 2, part (a).

Parallel (RNC) implementation

Proof of Thm. 2, part (b). To adapt the proof of (a) to prove part (b), the only difficulty is implementing step (2) of heads(w) in NC. This can be done using the following observation. When heads(w) does 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 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 step(x, S). This completes the description of the NC simulation of 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. 2.

4. MONOTONE COVERING

Recall that the instance of MONOTONE COVERING, defined by a cost function c and constraint collection $\mathcal{C}$, is

Find $x \in \mathbb{R}^n_+$ minimizing $c(x)$ subject to $(\forall S \in \mathcal{C})\ x \in S$.

The cost function $c : \mathbb{R}^n_+ \rightarrow \mathbb{R}_+$ must be non-decreasing, continuous, and submodular. Each constraint set $S \in \mathcal{C}$ must be monotone (closed upwards) and closed under limit.

Say that the cost function $c(x)$ is *locally computable* if the increase in $c(x)$ due to raising x_j can be determined knowing only the values of the variables that S depends on. Any linear or separable cost function is locally computable.

Theorem 3. *For* monotone covering *with a locally computable cost function there is a distributed δ-approximation algorithm taking $O(\log^2 |\mathcal{C}|)$ communication rounds in expectation and with high probability, where $|\mathcal{C}|$ is the number of constraints.*

Centralized algorithm

The centralized δ-approximation algorithm for monotone covering is as follows [22]. 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 $x \in S$ 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 $\beta$.
  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 3. *([22]) 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.*

(*Partial proof:* Each step starts with $x \notin S$. Since the optimal solution x^* is in S and S is monotone, 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. Details in [22].)

The function $\mathsf{stepsize}(x, S)$**.** In what follows, we take $\mathsf{stepsize}(x, S)$ to be the minimum β such that $\mathsf{step}(x, S)$ will satisfy S in one step. As observed in [22], this choice satisfies the requirement $\mathsf{stepsize}(x, S) \leq \mathsf{distance}_c(x, S)$ in Observation 3.

If this $\mathsf{stepsize}(x, S)$ is not easy to compute, more computationally tractable alternatives are sometimes possible. For example, for CMIP, a method similar to the one in the previous section for CMIP_2 can be used.

Distributed implementation

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.

Proof of Thm. 3. To start each phase, the algorithm finds large independent subsets of constraints by running one phase of Linial and Saks' (LS) decomposition algorithm [28][2], below, with any k such that $k \in \Theta(\log |\mathcal{C}|)$ (in case the nodes don't 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 $\mathcal{R} \subseteq \mathcal{S}$ 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 don't 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 $\mathcal{R}$: $(\forall S \in \mathcal{S})\ \Pr[S \in \mathcal{R}] \geq 1/c|\mathcal{C}|^{1/k}$ 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.

(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 don't 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 don't 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 $O_d(\log |\mathcal{C}|) \sum_{i=0}^{\infty} 4^i / |\mathcal{C}|^{di} = O_d(\log |\mathcal{C}|)$. $\square$

Applications

As mentioned in the introduction, monotone covering generalizes many covering problems. For all of these, Thm. 3 gives a distributed δ-approximation algorithm running in $O(\log^2 |\mathcal{C}|)$ communication rounds in expectation and with high probability.

[2]Decomposing the graph for packing and covering problems has been also used by Kuhn et al. to compute distributively an approximate solution for fractional covering [25]

COROLLARY 1. *There is a distributed δ-approximation algorithm for* CMIP*,* PROBABILISTIC CMIP*,* FACILITY LOCATION *and* PROBABILISTIC FACILITY LOCATION *taking* $O(\log^2 |\mathcal{C}|)$ *communication rounds in expectation and with high probability.*

The corollary follows immediately from Thm. 3 and the observation that the problems in question (see descriptions below) are special cases on MONOTONE COVERING.

If the complexity of the computation (as opposed to just the number of communication rounds) is important, these problems have appropriate $\mathsf{stepsize}()$ functions that can be computed efficiently (generally so that each constraint can be satisfied with overall work nearly linear in the problem size) [22].

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 *opening* cost $\sum_{j \in F} f_j \max_{i \in N(j)} x_{ij}$ plus the *assignment* cost $\sum_{ij \in E} d_{ij} x_{ij}$. The total cost is submodular. Each customer has at most δ accessible facilities.[3]

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.

5. REFERENCES

[1] 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.

[2] 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.

[3] D. Bertsimas and R. Vohra. Rounding algorithms for covering problems. *Mathematical Programming: Series A and B*, 80(1):63–89, 1998.

[4] 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.

[5] Z.-Z. Chen. A fast and efficient nc algorithm for maximal matching. *Information Processing Letters*, 55:303–307, 1995.

[6] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[7] 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.

[8] 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.

[9] 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.

[10] F. Grandoni, J. Könemann, J., and A. Panconesi. Distributed weighted vertex cover via maximal matchings. *Lecture Notes in Computer Science*, 3595:839–848, 2005.

[11] N. Hall and D. Hochbaum. A fast approximation algorithm for the multicovering problem. *Discrete Applied Mathematics*, 15(1):35–40, 1986.

[12] 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.

[13] D. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11:555–556, 1982.

[14] D. S. Hochbaum. Efficient bounds for the stable set, vertex cover, and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.

[15] J. Hoepman. Simple distributed weighted matchings. *Arxiv preprint cs.DC/0410047*, 2004.

[16] A. Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22:77–80, 1986.

[17] 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.

[18] D. S. Johnson. Approximation algorithms for combinatorial problems. *In the fifth ACM Symposium On Theory Of Computing*, 25:38–49, 1973.

[19] P. Kelsen. An optimal parallel algorithm for maximal matching. *Information Processing Letters*, 52:223–228, 1994.

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

[20] S. Khuller, U. Vishkin, and N. Young. A primal-dual parallel approximation technique applied to weighted set and vertex covers. *Journal of Algorithms*, 17:280–289, 1994.

[21] S. Kolliopoulos and N. Young. Approximation algorithms for covering/packing integer programs. *Journal of Computer and System Sciences*, 71(4):495–505, 2005.

[22] C. Koufogiannakis and N. 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] C. Lenzen, Y. 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.

[28] N. Linial and M. Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993.

[29] N. Linian. Locality in distributed graph algorithms. *SIAM Journal on Computing*, 21:193–201, 1992.

[30] 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.

[31] 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.

[32] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Math*, 13:383–390, 1975.

[33] 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.

[34] M. Naor and L. Stockmeyer. What can be computed locally? *SIAM Journal on Computing*, 24:1259–1277, 1995. STOC' 93.

[35] A. Panconesi and R. Rizzi. Some simple distributed algorithms for sparse networks. *Distributed Computing*, 14:97–100, 2001.

[36] D. Peleg. Distributed computing: a locality-sensitive approach. *Society for Industrial and Applied Mathematics*, 2000.

[37] A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM Journal on Computing*, 29:648–670, 1999.

[38] A. Srinivasan. New approaches to covering and packing problems. *In the twelveth ACM-SIAM Symposium On Discrete Algorithms*, pages 567–576, 2001.

[39] M. Wattenhofer and R. Wattenhofer. Distributed weighted matching. *Lecture Notes in Computer Science*, 3274:335–348, 2004.