UC Riverside
UC Riverside Electronic Theses and Dissertations

Title
Approximation Algorithms for Covering Problems

Permalink
https://escholarship.org/uc/item/6s01g4z5

Author
Koufogiannakis, Christos

Publication Date
2009

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA
RIVERSIDE

Approximation Algorithms for Covering Problems

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Christos Koufogiannakis

December 2009

Dissertation Committee:

Dr. Neal E. Young, Chairperson
Dr. Marek Chrobak
Dr. Stefano Lonardi

Copyright by
Christos Koufogiannakis
2009

The Dissertation of Christos Koufogiannakis is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I am grateful to my advisor, without whose help, I would not have been here.

I also thank the Greek State Scholarships Foundation (IKY) for their financial support.

To my parents for all the support.

ABSTRACT OF THE DISSERTATION

Approximation Algorithms for Covering Problems

by

Christos Koufogiannakis

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2009
Dr. Neal E. Young, Chairperson

In this thesis we present sequential and distributed approximation algorithms for covering problems.

First, we give a sequential δ-approximation algorithm for MONOTONE COVERING, a generalization of many fundamental NP-hard covering problems. The approximation ratio δ is the maximum number of variables on which any constraint depends. (For example, for vertex cover, δ is 2.) The algorithm unifies, generalizes, and improves many previous algorithms for fundamental covering problems such as VERTEX COVER, SET COVER, FACILITY LOCATION, and COVERING MIXED-INTEGER LINEAR PROGRAMS with upper bound on the variables.

The algorithm is also a δ-competitive algorithm for *online* MONOTONE COVERING, which generalizes online versions of the above-mentioned covering problems as well as many fundamental online paging and caching problems. As such it also generalizes many classical online algorithms, including LRU, FIFO, FWF, BALANCE, GREEDY-DUAL, GREEDY-DUAL SIZE (a.k.a. LANDLORD), and algorithms for connection caching, where δ is the cache size. It also gives new δ-competitive algorithms for *upgradable* variants of these problems, which model choosing the caching strategy *and* an appropriate

hardware configuration (cache size, CPU, bus, network, etc.).

Then we show distributed versions of the sequential algorithm. For WEIGHTED VERTEX COVER, we give a distributed 2-approximation algorithm taking $O(\log n)$ rounds. The algorithm generalizes to covering mixed integer linear programs (CMIP) with two variables per constraint ($\delta = 2$). For any MONOTONE COVERING problem, we show a distributed δ-approximation algorithm taking $O(\log^2 |\mathcal{C}|)$ rounds, where $|\mathcal{C}|$ is the number of constraints.

Last, we extend the distributed algorithms for covering to compute δ-approximate solutions for FRACTIONAL PACKING and MAXIMUM WEIGHTED b-MATCHING in hypergraphs, where δ is the maximum number of packing constraints in which a variable appears (for MAXIMUM WEIGHTED b-MATCHING δ is the maximum edge degree — for graphs $\delta = 2$).

Contents

List of Figures **x**

1 Introduction **1**
1.1 Monotone Covering . . . 1
1.2 Related Problems . . . 2
1.2.1 Covering Problems . . . 3
1.2.1.1 Vertex Cover . . . 3
1.2.1.2 Set Cover . . . 3
1.2.1.3 Set Multicover . . . 4
1.2.1.4 Covering Integer Programs . . . 4
1.2.1.5 Hardness of Approximation Results . . . 5
1.2.1.6 Alternatives to δ-approximation . . . 5
1.2.2 Online Problems . . . 5
1.2.2.1 Online Covering Problems . . . 6
1.2.2.2 Paging and Caching Problems . . . 6
1.2.2.3 Upgradable Caching . . . 7
1.2.3 Distributed Covering Problems . . . 9
1.2.3.1 Distributed Vertex Cover . . . 10
1.2.3.2 Other Related Results . . . 11
1.2.4 Distributed Fractional Packing and Maximum Weighted Matching . . . 12
1.2.4.1 Distributed Maximum Weighted Matching . . . 12
1.2.4.2 Distributed Fractional Packing . . . 13
1.2.4.3 Other Related Results . . . 14
1.3 Summary of Results . . . 14
1.3.1 Sequential Algorithms . . . 14
1.3.2 Distributed Algorithms for Covering . . . 16
1.3.3 Distributed Algorithms for Fractional Packing and Maximum Weighted Matching . . . 17

2 Sequential Algorithm **18**
2.1 The Greedy Algorithm for Monotone Covering . . . 18
2.2 Linear time algorithm for Vertex Cover, Set Cover and Facility Location 22
2.3 Nearly Linear-Time Implementation for Covering Mixed Integer Linear Programs . . . 23
2.4 Online Monotone Covering and Caching with Upgradable Hardware . . 24

2.4.1 Upgradable online problems. 26
2.5 Randomized Variant of Alg. 1 and Stateless Online Algorithm 28
2.6 Relation to the Local-Ratio Method 30

3 Distributed Covering **34**
3.1 Distributed Weighted Vertex Cover 34
3.1.1 Distributed Model 34
3.1.2 Distributed Algorithms for Weighted Vertex Cover 35
3.2 Distributed Mixed Integer Programs with Two Variables per Constraint 39
3.3 Distributed Monotone Covering 48

4 Distributed Algorithm for Fractional Packing **52**
4.1 Covering and packing 52
4.2 Distributed Fractional Packing with $\delta = 2$ 60
4.2.1 Distributed model for $\delta = 2$ 60
4.2.2 Distributed algorithm for $\delta = 2$ 60
4.3 Distributed Fractional Packing with general δ 64
4.3.1 Distributed model for general δ 64
4.3.2 Distributed algorithm 65

5 Conclusions **67**
5.1 Summary of Results 67
5.2 Future Work 68
5.2.0.1 Sequential Setting 68
5.2.0.2 Distributed Setting 69

Bibliography **70**

A Proofs **78**

List of Figures

2.1 Greedy δ-approximation algorithm for Monotone Covering (Alg. 1). . . . 19
2.2 Implementation of stepsize(x, S) for CMIP(Alg. 2). 23
2.3 Randomized and Stateless implementation of stepsize(x, S) (Alg. 3, Alg. 4). 28

3.1 Distributed 2-approximation algorithm for Weighted Vertex Cover (Alg. 5). 35
3.2 Analysis of Alg. 5. 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). 37
3.3 Distributed 2-approximation algorithm for CMIP_2 (Alg. 6). 43
3.4 Distributed δ-approximation algorithm for Monotone Covering (Alg. 7). 50

4.1 Sequential δ-approximation algorithm for Fractional Covering (Alg. 8). . 54
4.2 Sequential δ-approximation algorithm for Fractional Packing (Alg. 9). . 56
4.3 Distributed 2-approximation algorithm for Fractional Packing where each variable appears in at most 2 constraints and Maximum Weighted Matching on graphs (Alg. 10). 63
4.4 Distributed δ-approximation algorithm for Fractional Packing and Maximum Weighted Matching on hypergraphs (Alg. 11). 66

Original Publications

This thesis is based on the following original publications:

- 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.

  DOI http://dx.doi.org/10.1007/978-3-642-02927-1_53.

  The original publication is available at www.springerlink.com.

  ©Springer-Verlag Berlin Heidelberg 2009. With kind permission of Springer Science+Business Media.

- C. Koufogiannakis and N.E. Young. Distributed and parallel algorithms for weighted vertex cover and other covering problems. *In the twenty-eighth ACM symposium Principles of Distributed Computing*, pages 171–179, 2009.

  DOI http://doi.acm.org/10.1145/1582716.1582746.

  ©2009 ACM.

- C. Koufogiannakis and N.E. Young. Distributed fractional packing and maximum weighted b-matching via tail-recursive duality. *In the twenty-third International Symposium on Distributed Computing. Lecture Notes in Computer Science*, LNCS 5805:221–238, 2009.

  DOI http://dx.doi.org/10.1007/978-3-642-04355-0_23

  The original publication is available at www.springerlink.com.

  ©Springer-Verlag Berlin Heidelberg 2009. With kind permission of Springer Science+Business Media.

Chapter 1

Introduction

In this thesis we present research done by the author on approximation algorithms for covering problems. In Section 1.1 we introduce the basic covering problem that we study. In Section 1.2 we describe specific problems and related results for each one of them. Finally in Section 1.3 we summarize our results.

1.1 Monotone Covering

The main result of this thesis is a simple greedy approximation algorithm for Monotone Covering. An algorithm is called c-approximation if the cost of the solution produced by the algorithm is at most $c \cdot \mathrm{OPT}$, where OPT is the cost of the optimal solution.

A Monotone Covering instance is specified by a collection $\mathcal{C}$ of constraints and a non-negative, non-decreasing, submodular[1] objective function, $c : \mathbb{R}_+^n \to \mathbb{R}_+$. The problem is to compute $x \in \mathbb{R}_+^n$ minimizing $c(x)$ and on the same time satisfying

[1]Formally, $c(x)+c(y) \geq c(x \wedge y)+c(x \vee y)$, where $x \wedge y$ (and $x \vee y$) are the component-wise minimum (and maximum) of x and y. Intuitively, there is no positive synergy between the variables: let $\partial_j c(x)$ denote the rate at which increasing x_j would increase $c(x)$; then, increasing x_i (for $i \neq j$) does not increase $\partial_j c(x)$. Any separable function $c(x) = \sum_j c_j(x_j)$ is submodular, the product $c(x) = \prod_j x_j$ is not. The maximum $c(x) = \max_j x_j$ is submodular, the minimum $c(x) = \min_j x_j$ is not.

all constraints $S \in \mathcal{C}$. We write the problem in the following more compact form, $\min\{c(x) : x \in \mathbb{R}_+^n, (\forall S \in \mathcal{C})\ x \in S\}$. Each constraint $S \in \mathcal{C}$ must be monotone, but can be non-convex.

Monotone Covering allows each variable to take values throughout $\mathbb{R}_+$, but can still model problems with restricted variable domains. For example, formulate Vertex Cover as $\min\{\sum_v c_v x_v : x \in \mathbb{R}_+^V, (\forall (u,w) \in E)\ \lfloor x_u \rfloor + \lfloor x_w \rfloor \geq 1\}$. 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. Generally, to model problems where each variable x_j should take values in some closed set $U_j \subset \mathbb{R}_+$ (e.g. $U_j = \{0,1\}$ or $U_j = \mathbb{Z}_+$), one allows $x \in \mathbb{R}_+^n$ but replaces each monotone constraint $x \in S$ by the monotone constraint $x \in \hat{\mu}(S)$, where $\hat{\mu}(S) = \{x : \mu(x) \in S\}$ and $\mu_j(x) = \max\{z \in U_j, z \leq x_j\}$. If $x \in \mathbb{R}_+^n$ is any δ-approximate solution to the modified problem, then $\mu(x)$ will be a δ-approximate solution respecting the variable domains. (For Vertex Cover each $U_j = \mathbb{Z}_+$ so $\mu_j(x) = \lfloor x_j \rfloor$.)[2]

We give here a single greedy δ-approximation algorithm for a combinatorially rich class of *monotone* covering problems, including many classical covering problems as well as online paging and caching problems. The approximation ratio, δ, is the maximum number of variables on which any constraint depends. (For Vertex Cover, $\delta = 2$.)

1.2 Related Problems

Monotone Covering generalizes many fundamental covering problems. For some problems in the class, no greedy (or other) δ-approximation algorithms were known.

[2] In this setting, if the cost is defined only on the restricted domain, it should be extended to $\mathbb{R}_+^n$ for the algorithm. One way is to take the cost of $x \in \mathbb{R}_+^n$ to be the expected cost of $\hat{x}$, where $\hat{x}_j$ is rounded up or down to its nearest elements a, b in U_j such that $a \leq x_j \leq b$: take $\hat{x}_j = b$ with probability $\frac{b - x_j}{b - a}$, otherwise take $\hat{x}_j = a$. If a or b does not exist, let $\hat{x}_j$ be the one that does.

For others, previous greedy δ-approximation algorithms were known, but with non-trivial and seemingly problem-specific analyses. In this section we describe problems that are special cases of MONOTONE COVERING and we discuss related work.

1.2.1 Covering Problems

1.2.1.1 Vertex Cover

Given a graph $G = (V, E)$, the VERTEX COVER problem asks for a minimum cardinality set of vertices C such that every edge $e \in E$ is touching a vertex in C. In the more general *weighted* version of the problem, vertices have non-negative cost, and one should find a set of vertices C of minimum total costs that covers all edges. We can formulate WEIGHTED VERTEX COVER as an integer program of the form $\min\{c \cdot x : x_u + x_v \geq 1 \ \ (\forall (u,v) \in E); x \in \{0,1\}^{|E|}\}$.

For WEIGHTED VERTEX COVER, in the early 1980's, Hochbaum gave a 2-approximation algorithm that rounds a solution to the standard LP relaxation [62]. Bar-Yehuda and Even gave a 2-approximation linear-time greedy algorithm based on the local-ratio technique [13].

1.2.1.2 Set Cover

SET COVER is a more general problem. Given a universe $\mathcal{U}$ of elements,a collection $\mathcal{S}$ of subsets of $\mathcal{U}$ find a subcollection $C \subseteq \mathcal{S}$ of sets whose union is $\mathcal{U}$. The integer linear programming formulation is $\min\{c \cdot x : \sum_{s \ni e} x_s \geq 1 \ \ (\forall e \in \mathcal{U}); x \in \{0,1\}^{|\mathcal{S}|}\}$. The previous algorithms for VERTEX COVER give straightforwardly δ-approximation algorithms for SET COVER, where δ is the maximum number of variables on which any constraint depends — the maximum number of sets that contain any element.

1.2.1.3 Set Multicover

SET MULTICOVER is a natural generalization of SET COVER. Here an element $e \in \mathcal{U}$ has to be covered at least b_e times, $\min\{c \cdot x : \sum_{s \ni e} x_s \geq b_e \;\; (\forall e \in \mathcal{U}); x \in \{0,1\}^{|\mathcal{S}|}\}$. In 1986, Hall and Hochbaum gave a quadratic-time primal-dual δ-approximation algorithm [54].

1.2.1.4 Covering Integer Programs

A problem that includes all the above as special cases is Covering Integer Programs (CIP). Given a non-negative cost function c, a matrix A, vectors b and u, all with non-negative entries, a CIP with upper bounds on variables is a problem of the form $\min\{c \cdot x : Ax \geq b; x \in \mathbb{Z}_+^m, x \leq u\})$. Again, let δ be the maximum number of variables on which any constraint depends.

In the late 1990's, Bertsimas and Vohra showed a quadratic-time primal-dual algorithm for CIP, restricted to $\{0,1\}$-variables and integer constraint matrix A, and with approximation ratio $\max_i \sum_j A_{ij} \geq \delta$ [19]. Most recently, in 2000, Carr et al. gave the first δ-approximation for general CIP with $\{0,1\}$ variables [24].[3] They state (without details) that their result extends to allow general upper bounds on the variables (restricting $x_j \in \{0,1,2,\ldots,u_j\}$). In 2009 (independently of our work), Pritchard gives details of an extension to CIP with general upper bounds on the variables [105]. Both [24] and [105] use exponentially many valid "Knapsack Cover" (KC) inequalities to reduce the integrality gap to δ. Their algorithms solve the LP using the ellipsoid method, so the running time is a high-degree polynomial.

[3] The standard LP relaxation has an arbitrarily large integrality gap (e.g. $\min\{x_1 : 10x_1 + 10x_2 \geq 11; x_2 \leq 1\}$ has gap 10).

1.2.1.5 Hardness of Approximation Results

The special case of CIP where all A, b, c, u take values 0/1, is the same as δ-Bounded Hypergraph Vertex Cover (a.k.a. Set Cover with maximum element frequency δ). For any fixed $\varepsilon > 0$, if P $\neq$ NP this problem is not approximable to a $(\delta - 1 - \varepsilon)$ factor [37]. In addition it is not approximable to $(\delta - \varepsilon)$ under the unique games conjecture [75].

1.2.1.6 Alternatives to δ-approximation

As an alternative when δ is large, many covering problems considered here also admit $O(\log \widehat{\delta})$-approximation algorithms, where $\widehat{\delta}$ is the maximum number of constraints in which any variable occurs. Examples include a greedy algorithm for Set Cover [69, 94, 28] (1975) and greedy $O(\log \max_j \sum_i A_{ij})$-approximation algorithms for CIP with $\{0,1\}$-variables and integer A [38, 43] (1982). Srinivasan gave $O(\log \widehat{\delta})$-approximation algorithms for general CIP without variable upper bounds [111, 112] (2000); these were extended to CIP with variable upper bounds by Kolliopoulos et al. [78] (2005). (The latter algorithm solves the CIP relaxation with KC inequalities, then randomly rounds the solution.) The class of $O(\log(\hat{\delta}))$-approximation algorithms for general CIP is not yet fully understood; these algorithms could yet be subsumed by a single fast greedy algorithm.

1.2.2 Online Problems

A problem is called online if the input is given and processed piece-by-piece. An algorithm for solving such a problem must give the solution to each piece knowing only the pieces that it has seen so far and ignoring the pieces to be given in the future. An algorithm is c-competitive if the cost of the solution produced by the algorithm is

bounded by $c \cdot \mathrm{OPT} + b$ where OPT is the cost of the optimal solution and b depends only on the starting configurations of the online and optimal (offline) algorithms.

1.2.2.1 Online Covering Problems

Following [22], in the online version of the classical covering problems above (Vertex Cover, Set Cover, Set Multicover, CIP) the algorithm knows the cost function (the cost of each variable) but it does not know all the covering constraints. Constraints are revealed one-by-one. Once the algorithm is given a constraint, it should increase some variables to satisfy the constraint, but it can never decrease any variable. Bar-Yehuda and Even's algorithm is δ-competitive for Vertex Cover and Set Cover [13]. However, to the best of our knowledge there was no previously known δ-competitive algorithm for online Set Multicover and CIP.

1.2.2.2 Paging and Caching Problems

Online paging and caching algorithms (paging, weighted caching, file caching, (generalized) connection caching, etc.) are also (online) Monotone Covering problems, as they can be formulated as online Set Cover [6].

Paging is a problem where the input is an online sequence of requests $r_1, r_2, \ldots$ for pages to be brought into a cache of size $k = \delta$. The goal is to minimize the number of page faults. Paging can be formulated as Monotone Covering as follows [6]. Let x_t indicate whether page r_t is evicted before the next request to r_t after time t, so the total cost is $\sum_t x_t$. For *any* k-subset $Q = \{r_s : s < t, r_s \neq r_t\}$, at least one page $r_s \in Q$ must have been evicted (s is the time of the most recent request to r_s), so the following constraint is met, $\sum_{r_s \in Q} \lfloor x_s \rfloor \geq 1$.

These problems also have a rich history (see [21]). They are known to have

deterministic δ(=k)-competitive algorithms [110, 106, 117, 23, 29, 2, 118]. For most online problems here, no *deterministic* online algorithm can be better than δ-competitive. But many online problems admit better-than-δ-competitive *randomized* algorithms. Examples include paging [42, 96], weighted caching [6, 23], connection caching [29], and file caching [7]. Some cases of online Monotone Covering (e.g. Vertex Cover) are unlikely to have better-than-δ-competitive randomized algorithms.

1.2.2.3 Upgradable Caching

Standard online caching problems model only the caching strategy. In practice other parameters (e.g., the size of the cache, the speed of the CPU, bus, network, etc.) must also be chosen well. In *upgradable* caching, the algorithm chooses not only the caching strategy, but also the hardware configuration. The hardware configuration is assumed to be determined by how much has been spent on each of some d components. The configuration is modeled by a vector $y \in \mathbb{R}_+^d$, where y_i has been spent so far on component i.

In response to each request, the algorithm can upgrade the hardware by increasing the y_i's. Then, if the requested item r_t is not in cache, it is brought in. Then items in cache must be selected for eviction until the set Q of items remaining in cache is cachable, as determined by some specified predicate $\mathsf{cachable}_t(Q, y)$. The cost of evicting an item r_s is specified by a function $\mathsf{cost}(r_s, y)$.

The $\mathsf{cachable}()$ predicate and $\mathsf{cost}()$ function can be specified arbitrarily, subject to the following restrictions. Predicate $\mathsf{cachable}_t(Q, y)$ must be non-decreasing in y (upgrading the hardware does not cause a cachable set to become uncachable) and non-increasing with Q (any subset of a cachable set is cachable). The function $\mathsf{cost}(r_s, y)$ must be non-increasing in y (upgrading the hardware does not increase the eviction cost

of any item). To model (standard, non-upgradable) file caching, take $\mathsf{cachable}_t(Q, y)$ to be true if $\sum_{r_s \in Q} \mathsf{size}(r_s) \leq k$.

In general, the adversary is free to constrain the cache contents at each step t in *any* way that depends on t and the hardware configuration, as long as upgrading the cache or removing items does not make a cachable set uncachable. Likewise, the cost of evicting any item can be determined by the adversary in *any* way that depends on the item and the hardware configuration, as long as upgrading the configuration does not increase any eviction cost. This gives a great deal of flexibility in comparison to the standard model. For example, the adversary could insist (among other constraints) that no set containing both of two (presumably conflicting) files can be cached. Or, upgrading the hardware could reduce the eviction cost of some items arbitrarily, even to zero.

The optimal cost is achieved by choosing an optimal hardware configuration at the start, then handling all caching decisions optimally. To be competitive, an algorithm must also choose a good hardware configuration: an algorithm is δ-competitive if its total cost (eviction cost plus final hardware configuration cost, $\sum_i y_i$) is at most δ times the optimum. Naturally, when the algorithm evicts an item, it pays the eviction cost in its *current* hardware configuration. Later upgrades do not reduce earlier costs.

Next we describe how to model the upgradable problem via online Monotone Covering with at most $\delta = k + d$ variables per constraint, where k is the maximum number of files ever held in cache and d is the number of hardware components.

Let variable y_i for $i = 1, \ldots, d$ denote the amount invested in component i, so that the vector y gives the current hardware configuration. Let x_t be the cost (if any) incurred for evicting the tth requested item r_t at any time before its next request. The total final cost is $\sum_i y_i + \sum_t x_t$. At time t, if some subset $Q \subseteq \{r_s : s \leq t\}$ of the items

is not cachable, then at least one item $r_s \in Q - \{r_t\}$ (where s is the time of the most recent request to r_s) must have been evicted, so the following constraint is met:

$$\mathsf{cachable}_t(Q, y) \;\text{ or }\; \textstyle\sum_{r_s \in Q - \{r_t\}} \lfloor x_s / \mathsf{cost}(r_s, y) \rfloor \geq 1. \qquad S_t(Q)$$

The restrictions on cachable and cost ensure that this constraint is monotone in x and y.

We know of no previous results for upgradable caching, although the classical online Rent-or-Buy (a.k.a. Ski Rental) problem [70] and its "multislope" generalization [93] have the basic characteristic (paying a fixed cost now can reduce many later costs; these are special cases of online Monotone Covering with $\delta = 2$).

1.2.3 Distributed Covering Problems

Assume that a covering problem is represented by a graph. There are several graph representations but they are equivalent in the sense that one can simulate the other. For example consider a graph in which nodes are covering variables and edges connect variables that appear in the same constraint. Or imagine a graph where nodes are constraints and there is an edge between two nodes if they share a variable. Another possible representation, is a graph where we have nodes for both variables ands constraints. There is an edge between a variable node to a constraint node if the variable appears in the constraint.

In the distributed setting, given such a representation of a covering problem, computation takes place in rounds at nodes using only local information. In the standard synchronous distributed model, in each round, nodes can exchange a constant number of messages with neighbors, and perform some local computation [104]. We assume no restriction on message size and local computation. (Note that a synchronous model algorithm can be transformed into an asynchronous algorithm with the same time

complexity [104].) The goal here is to compute a solution, that is, each node or edge knows its own value. An efficient computation should finish in a number of rounds that is poly-logarithmic in the network size [90]:

> 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 sequential algorithms. Research in this spirit includes works on Dominating Set (Set Cover) [68, 86, 87], Capacitated Dominating set [83], Capacitated Vertex Cover [48, 49], Weighted Matching [115, 64, 92, 91] and many other problems. Thus, studying distributed δ-approximation algorithms for covering problems, is also very important in this direction.

1.2.3.1 Distributed Vertex Cover

In the distributed setting, in the case of unweighted Vertex Cover, 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 [58] or in $O(\Delta + \log^* n)$ rounds using the algorithm of Panconesi and Rizzi [103], 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 [67]. Maximal matching is also in NC [25, 95, 74] and in RNC (parallel poly-log time with polynomially many randomized processors) [67] 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 [76]. 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 [50]. Their algorithm assumes integer vertex weights.

As noted in [84], no previous 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).

1.2.3.2 Other Related Results

Kuhn, Moscibroda and Wattenhofer describe distributed approximation algorithms for *fractional* covering (and packing) linear programs [85]. 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 $O(\log \hat{\delta})$-approximate integer solutions in $O(\log n)$ rounds, where $\hat{\delta}$ 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 [84]. They prove that to achieve even a poly-logarithmic approximation ratio for Vertex Cover, the number of rounds required is at least $\Omega(\sqrt{\log n/\log\log n})$ and $\Omega(\log \Delta/\log\log \Delta)$, where Δ is the maximum vertex degree.

1.2.4 Distributed Fractional Packing and Maximum Weighted Matching

1.2.4.1 Distributed Maximum Weighted Matching

Given a (hyper)graph $H(V, E)$, vertex capacities $c \in \mathbb{Z}_+^{|V|}$ and an edge weight vector $w \in \mathbb{R}_+^m$, the Maximum Weighted b-Matching problem is to compute a vector $y \in \mathbb{Z}_+^{|E|}$ maximizing $\sum_{e \in E} w_e y_e$ and meeting all the vertex capacity constraints $\sum_{e \in E(u)} y_e \leq c_u \ (\forall u \in V)$, where $E(u)$ is the set of edges incident to vertex u. For this problem, δ is the maximum (hyper)edge degree (for graphs $\delta = 2$).

Maximum Weighted b-Matching is a cornerstone optimization problem in graph theory and Computer Science. As a special case it includes the ordinary Maximum Weighted Matching problem ($c_u = 1$ for all $u \in V$). In the sequential setting, Maximum Weighted b-Matching on graphs belongs to the "well-solved class of integer linear programs" in the sense that it can be solved in polynomial time [40, 41, 99]. Moreover, finding a 2-approximate[4] solution for Maximum Weighted Matching is relatively easy, since the obvious greedy algorithm, which selects the heaviest edge that is not conflicting with already selected edges, gives a 2-approximation. For hypergraphs the problem is NP-hard, since it generalizes Set Packing, one of Karp's 21 NP-complete problems [73].

In the distributed setting, the problem seems to be more difficult. There are several works considering distributed Maximum Weighted Matching on edge-weighted graphs. Uehara and Chen present a constant time $O(\Delta)$-approximation algorithm [113], where Δ is the maximum vertex degree. Wattenhofer and Wattenhofer improve this result, showing a randomized 5-approximation algorithm taking $O(\log^2 n)$ rounds [116]. Hoepman shows a deterministic 2-approximation algorithm taking $O(m)$ rounds [64].

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

Lotker, Patt-Shamir and Rosén give a randomized $(4+\varepsilon)$-approximation algorithm running in $O(\varepsilon^{-1} \log \varepsilon^{-1} \log n)$ rounds [92]. Lotker, Patt-Shamir and Pettie improve this result to a randomized $(2+\varepsilon)$-approximation algorithm taking $O(\log \varepsilon^{-1} \log n)$ rounds [91]. Their algorithm uses as a black box any distributed constant-factor approximation algorithm for maximum weighted matching which takes $O(\log n)$ rounds (i.e. [92]). 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 [65]. Nieberg presents a $(1+\varepsilon)$-approximation algorithm in $O(\varepsilon^{-2} + \varepsilon^{-1} \log(\varepsilon^{-1} n) \log n)$ rounds [101]. The latter two results give randomized 2-approximation algorithms for Maximum Weighted Matching in $O(\log^2 n)$ rounds.

1.2.4.2 Distributed Fractional Packing

Given a weight vector $w \in \mathbb{R}_+^m$, a coefficient matrix $A \in \mathbb{R}_+^{m \times n}$ and a vector $c \in \mathbb{R}_+^n$, the Fractional Packing problem is to compute a vector $y \in \mathbb{R}_+^m$ to maximize $\sum_{i=1}^m w_i y_i$ and at the same time meet all the constraints $\sum_{i=1}^m A_{ij} y_i \leq c_j \ \ (\forall j = 1 \ldots n)$. We use δ to denote the maximum number of packing constraints in which a variable appears, that is, $\delta = \max_i |\{j | \ A_{ij} \neq 0\}|$. In the sequential setting, Fractional Packing can be solved optimally in polynomial time using linear programming. Alternatively, one can use a faster approximation algorithm (i.e. [79]).

In the distributed setting, Kuhn, Moscibroda and Wattenhofer show a $(1+\varepsilon)$-approximation algorithm for Fractional Packing with logarithmic message size, but the running time depends on the input coefficients. For unbounded message size they show a constant-factor approximation algorithm for Fractional Packing which takes $O(\log m)$ rounds. If an integer solution is desired, then distributed randomized rounding ([86]) can be used. This gives an $O(\delta)$-approximation for Maximum Weighted

b-Matching on (hyper)graphs with high probability in $O(\log m)$ rounds, where 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 $\delta = 2$) .

1.2.4.3 Other Related Results

The best lower bounds known for distributed packing and matching are given by Kuhn, Moscibroda and Wattenhofer [85]. They prove that to achieve a constant or even a poly-logarithmic approximation ratio for fractional maximum matching, any algorithms requires at least $\Omega(\sqrt{\log n / \log\log n})$ rounds and $\Omega(\log \Delta / \log\log \Delta)$, where Δ is the maximum vertex degree.

For unweighted Maximum Matching on graphs, Israeli and Itai give a randomized distributed 2-approximation algorithm running in $O(\log n)$ rounds [67]. Lotker, Patt-Shamir and Pettie improve this result giving a randomized $(1+\varepsilon)$-approximation algorithm taking $O(\varepsilon^{-3}\log n)$ rounds [91]. Czygrinow, Hańćkowiak, and Szymańska show a deterministic 3/2-approximation algorithm which takes $O(\log^4 n)$ rounds [33]. A $(1+\varepsilon)$-approximation for Maximum Weighted Matching on graphs is in NC [65].

1.3 Summary of Results

1.3.1 Sequential Algorithms

Chapter 2 describes our greedy δ-approximation algorithm (Alg. 1 in Section 2.1) for Monotone Covering. It is roughly the following: *consider the constraints in any order; to satisfy a constraint, raise each variable in the constraint continuously and simultaneously, at rate inversely proportional to its cost. At termination, round x*

down to $\mu(x)$ if appropriate.

The proof of the approximation ratio is relatively simple: with each step, the cost incurred by the algorithm is at most δ times the reduction in the *residual cost* — the minimum possible cost to augment the current x to feasibility. The algorithm is online (as described below), and admits distributed implementations (Chapter 3).

The running time depends on the implementation, which is problem specific, but can be fast. Section 2.2 describes linear-time implementations for Vertex Cover, Set Cover, and (non-metric) Facility Location. Section 2.3 describes a nearly linear-time implementation for covering mixed integer linear programs with variable upper bounds (CMIP). (In contrast, the only previous δ-approximation algorithm (for CIP, a slight restriction of CMIP) uses the ellipsoid method; its running time is a high-degree polynomial [24].)

Section 2.4 discusses *online* Monotone Covering. The greedy algorithm (Alg. 1) is an online algorithm. Thus, it gives δ-competitive algorithms for online versions of all of the covering problems mentioned above. It also generalizes many classical deterministic online algorithms for paging and caching, including LRU, FIFO, FWF for paging [110], Balance and Greedy Dual for weighted caching [26, 117], Landlord [118], a.k.a. Greedy Dual Size [23], for file caching, and algorithms for Connection Caching [29, 30, 31, 2]. The competitive ratio δ is the cache size, commonly denoted k, or, in the case of file caching, the maximum number of files ever held in cache — at most k or $k+1$, depending on the specification. This is the best possible competitive ratio for deterministic online algorithms for these problems.

Section 2.4.1 illustrates the generality of online Monotone Covering by describing a $(k+d)$-competitive algorithm for a new class of **upgradable** caching problems. In the competitive ratio, d is the number of configurable hardware parameters. We know

of no previous results for upgradable caching, although the classical online RENT-OR-BUY (a.k.a. SKI RENTAL) problem [70] and its "multislope" generalization [93] have the basic characteristic (paying a fixed cost now can reduce many later costs; these are special cases of online Monotone Covering with $\delta = 2$).

Section 2.5 describes a natural randomized generalization of Alg. 1, with more flexibility in incrementing the variables. This yields a *stateless* online algorithm, generalizing the HARMONIC k-server algorithm (as it specializes for paging and weighted caching [106]) and Pitt's WEIGHTED VERTEX COVER algorithm [10].

Section 2.6 concludes by discussing the relation of the analysis here to the local-ratio method. Greedy approximation algorithms can sometimes be analyzed naturally via the local-ratio method. The results here extend many local-ratio results. The analysis of our algorithm can be recast as a local-ratio analysis, but in a non-traditional form.

1.3.2 Distributed Algorithms for Covering

In Chapter 3 we present distributed implementations of the greedy δ-approximation algorithm.

Section 3.1 gives a true 2-approximation algorithm for the well-studied WEIGHTED VERTEX COVER that takes $O(\log n)$ rounds. It also gives a parallel versions take poly-$\log n$ time. The best previous comparable works gave only $(2+\varepsilon)$-approximate solutions for any fixed $\varepsilon > 0$ or had weight-dependent run times.

In Section 3.3 we describe 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, FACILITY LOCATION, WEIGHTED SET COVER and WEIGHTED VERTEX COVER. Pre-

viously, 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.

1.3.3 Distributed Algorithms for Fractional Packing and Maximum Weighted Matching

In Chapter 4 we present efficient distributed δ-approximation algorithms for Fractional Packing and Maximum Weighted Matching. Here we use a non-standard primal-dual extension of our algorithm for covering.

In Section 4.2, for Fractional Packing where each variable appears in at most two constraints ($\delta = 2$), we show a distributed 2-approximation algorithm running in $O(\log m)$ rounds in expectation and with high probability, where m is the number of packing variables. This is the first 2-approximation algorithm requiring only $O(\log m)$ rounds. This also gives the first 2-approximation algorithm for Maximum Weighted b-Matching on graphs that takes $O(\log n)$ rounds.

In Section 4.3, for Fractional Packing where each variable appears in at most δ constraints, we give a distributed δ-approximation algorithm 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 [85], but the running time is slower by a logarithmic-factor. For Maximum Weighted b-Matching on hypergraphs with maximum hyperedge degree δ this is also a distributed δ-approximation algorithm running in $O(\log^2 m)$ rounds in expectation and with high probability, where m is the number of hyperedges. Our result improves over the best previously known $O(\delta)$-approximation ratio by [85], but it is slower by a logarithmic factor.

Chapter 2

Sequential Algorithm

In this chapter we describe the core δ-approximation algorithm for MONOTONE COVERING[1]. We also show specific implementations of VERTEX COVER, SET COVER, FACILITY LOCATION and CMIP. We also describe applications in the online setting and we give randomized and stateless implementations. Finally we discuss the connection to the local-ratio technique.

2.1 The Greedy Algorithm for Monotone Covering

Fix an instance of MONOTONE COVERING, $\min\{c(x) : x \in \mathbb{R}^n_+, (\forall S \in \mathcal{C})\ x \in S\}$. Let $\mathsf{Vars}(S)$ denote the variables in x that constraint $x \in S$ depends on, so that $\delta = \max_{S \in \mathcal{C}} |\mathsf{Vars}(S)|$.

The algorithm (Alg. 1) starts with $x = \mathbf{0}$, then repeats the following step until all constraints are satisfied: *choose any unmet constraint and a* step size $\beta > 0$*; for*

[1]The results in this chapter appear in:
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. DOI http://dx.doi.org/10.1007/978-3-642-02927-1_53. The original publication is available at www.springerlink.com. ©Springer-Verlag Berlin Heidelberg 2009. With kind permission of Springer Science+Business Media.

Greedy algorithm for Monotone Covering (monotone constraints $\mathcal{C}$, submodular objective c) alg. 1
output: feasible $x \in S$ ($\forall S \in \mathcal{C}$), δ-approximately minimizing $c(x)$ (see Thm. 1)
1. Let $x \leftarrow \mathbf{0}$. *... δ is the max # of vars any constraint depends on.*
2. While $\exists\ S \in \mathcal{C}$ such that $x \notin S$, do **step**(x, S) for any S such that $x \notin S$.
3. Return x. *... or $\mu(x)$ in the case of restricted variable domains.*

subroutine step$_c(x, S)$: *... makes progress towards satisfying $x \in S$.*
1. Choose a scalar *step size* $\beta \geq 0$. *... choose β subject to restriction in Thm. 1.*
2. For $j \in \mathsf{Vars}(S)$, let $x'_j \in \mathbb{R}_+ \cup \{\infty\}$ be the maximum such that raising x_j to x'_j would raise $c(x)$ by at most β.
3. For $j \in \mathsf{Vars}(S)$, let $x_j \leftarrow x'_j$. *... if c is linear, then $x'_j = x_j + \beta/c_j$*

Figure 2.1: Greedy δ-approximation algorithm for Monotone Covering (Alg. 1).

each variable x_j that the constraint depends on ($j \in \mathsf{Vars}(S)$), raise that variable so as to increase the cost $c(x)$ by at most β. (The step increases the total cost by at most $\delta\beta$.) The algorithm returns x (or, if variable domains are restricted as described in the introduction, $\mu(x)$).

The algorithm returns a δ-approximation, as long as each step size β is *at most* the minimum cost to optimally augment x to satisfy S, that is, $\min\{c(\hat{x}) - c(x) : \hat{x} \in S, \hat{x} \geq x\}$. Denote this cost $\mathsf{distance}_c(x, S)$. Also, let $\mathsf{residual}_c(x)$ be the *residual cost* of x — the minimum cost to augment x to full feasibility, i.e., $\mathsf{distance}_c(x, \cap_{S \in \mathcal{C}} S)$.

Theorem 1 *For* Monotone Covering*, the greedy algorithm (Alg. 1) returns a δ-approximate solution as long as it chooses step size $\beta \leq \mathsf{distance}_c(x, S)$ in each step (and eventually terminates).*

Proof. First, a rough intuition. 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 $x_k < x^*_k$. By raising all x_j for $j \in \mathsf{Vars}(S)$, the algorithm makes progress "covering" at least that coordinate x^*_k of x^*. Provided the step increases x_k to $x'_k \leq x^*_k$, the cost incurred can be charged to a corresponding portion of the cost of x^*_k (intuitively, to the

cost of the part of x^*_k in the interval $[x_k, x'_k]$; formally, to the *decrease in the residual cost* from increasing x_k, provably at least β). Since the step increases $c(x)$ by at most $\beta\delta$, and results in a charge to $c(x^*)$ of at least β, this proves the δ-approximation.

Here is the formal proof. By inspection (using that c is submodular) each step of the algorithm increases $c(x)$ by at most $\beta|\mathsf{Vars}(S)| \le \beta\delta$. We show that $\mathsf{residual}(x)$ decreases by at least β, so the invariant $c(x)/\delta + \mathsf{residual}(x) \le \mathrm{OPT}$ holds, proving the theorem.

Let x and x', respectively, be x before and after a given step. Let feasible $x^* \ge x$ be an optimal augmentation of x to full feasibility, so $c(x^*) - c(x) = \mathsf{residual}(x)$. Let $x \wedge y$ (resp. $x \vee y$) denote the component-wise minimum (resp. maximum) of x and y. By the submodularity of c, $c(x') + c(x^*) \ge c(x' \vee x^*) + c(x' \wedge x^*)$. (Equality holds if c is separable (e.g. linear).)

Rewriting gives $[c(x^*) - c(x)] - [c(x' \vee x^*) - c(x')] \ge c(x' \wedge x^*) - c(x)$.

The first bracketed term is $\mathsf{residual}(x)$. The second is at least $\mathsf{residual}(x')$, because $x^* \vee x' \ge x'$ is feasible. Thus,

$$\mathsf{residual}(x) - \mathsf{residual}(x') \ge c(x' \wedge x^*) - c(x). \qquad (2.1)$$

To complete the proof, we show the right-hand side of (2.1) is at least β.

Case 1. Suppose $x'_k < x^*_k$ for some $k \in \mathsf{Vars}(S)$. (In this case it must be that increasing x_k to x'_k costs β.)

Let y be x with just x_k raised to x'_k. Then $c(x' \wedge x^*) \ge c(y) = c(x) + \beta$.

Case 2. Otherwise $x' \wedge x^* \in S$, because $x^* \in S$ and $x'_j \ge x^*_j$ for all $j \in \mathsf{Vars}(S)$. Also $x' \wedge x^* \ge x$.

Thus, the right-hand side of (2.1) is at least $\mathsf{distance}_c(x, S)$. By assumption this is at least β. ■

Choosing the step size, β. In a sense, the algorithm reduces the given problem to a sequence of subproblems, each of which requires computing a lower bound on $\mathsf{distance}(x, S)$ for the current x and a given unmet constraint S. To completely specify the algorithm, one must specify how to choose β in each step.

Thm. 1 allows β to be small. At a minimum, $\mathsf{distance}(x, S) > 0$ when $x \notin S$, so one can take β to be infinitesimal. Then Alg. 1 raises x_j for $j \in \mathsf{Vars}(S)$ continuously at rate inversely proportional to $\partial c(x)/\partial x_j$ (at most until $x \in S$).

Another, generic, choice is to take β just large enough to satisfy $x \in S$. This also satisfies the theorem:

Observation 2 *Let β be the minimum step size so that $\mathsf{Step}(x, S)$ brings x into S. Then $\beta \le \mathsf{distance}_c(x, S)$.*

Thm. 1 can also allow β to be *more* than large enough to satisfy the constraint. Consider $\min\{x_1 + 2x_2 : x \in S\}$ where $S = \{x : x_1 + x_2 \ge 1\}$. Start with $x = \mathbf{0}$. Then $\mathsf{distance}(x, S) = 1$. The theorem allows $\beta = 1$. A single step with $\beta = 1$ gives $x_1 = 1$ and $x_2 = 1/2$, so that $x_1 + x_2 = 3/2 > 1$.

Generally, one has to choose β small enough to satisfy the theorem, but large enough so that the algorithm does not take too many steps. The computational complexity of doing this has to be addressed on a per-application basis. Consider a simple Subset-Sum example: $\min\{c \cdot x : x \in S\}$ where the single constraint S contains $x \ge 0$ such that $\sum_j c_j \min(1, \lfloor x_j \rfloor) \ge 1$. Computing $\mathsf{distance}(\mathbf{0}, S)$ is NP-hard, but it is easy to compute a useful β, for example $\beta = \min_{j : x_j < 1} c_j(1 - x_j)$. With this choice, the algorithm will satisfy S within δ steps.

2.2 Linear time algorithm for Vertex Cover, Set Cover and Facility Location

Here we give linear-time implementations for FACILITY LOCATION, SET COVER, and VERTEX COVER.

Theorem 3 *For (non-metric)* FACILITY LOCATION*,* SET COVER*, and* VERTEX COVER*, the greedy δ-approximation algorithm (Alg. 1) has a linear-time implementation. For* FACILITY LOCATION *δ is the maximum number of facilities that might serve any given customer.*

Proof. Formulate (non-metric) FACILITY LOCATION as minimizing the submodular objective $\sum_j f_j \max_i x_{ij} + \sum_{ij} d_{ij} x_{ij}$ subject to, for each customer i, $\sum_{j \in N(i)} \lfloor x_{ij} \rfloor \geq 1$ (where $j \in N(i)$ if customer i can use facility j).[2]

The implementation starts with all $x_{ij} = 0$. It considers the customers i in any order. For each it does the following: let $\beta = \min_{j \in N(i)}[d_{ij} + f_j(1 - \max_{i'} x_{i'j})]$ (the minimum cost to raise x_{ij} to 1 for any $j \in N(i)$). Then, for each $j \in N(i)$, raise x_{ij} by $\min[\beta/d_{ij}, (\beta + f_j \max_{i'} x_{i'j})/(d_{ij} + f_j)]$ (just enough to increase the cost by β). By maintaining, for each facility j, $\max_{i'} x_{i'j}$, the above can be done in linear time, $O(\sum_i |N(i)|)$.

VERTEX COVER and SET COVER are the special cases when $d_{ij} = 0$. ■

[2]The standard ILP is not a covering ILP due to constraints $x_{ij} \leq y_j$. The standard reduction to Set Cover increases Δ exponentially.

subroutine $\mathsf{stepsize}_c(x, S(I, A_i, u, b_i))$ **(for CMIP)** alg. 2

1. Order $I = (j_1, j_2, \ldots, j_k)$ by decreasing A_{ij}. ... *So* $A_{ij_1} \geq A_{ij_2} \geq \cdots \geq A_{ij_k}$.
   Let $J = J(x, S)$ contain the minimal prefix of I such that $x \notin S(J, A_i, u, b_i)$.
   Let S' denote the relaxed constraint $S(J, A_i, u, b_i)$.
2. Let $U = U(x, S) = \{j : x_j \geq u_j; A_{ij} > 0\}$ contain the variables that have hit their upper bounds.
3. Let $\beta_J = \min_{j \in J-U}(1 - x_j + \lfloor x_j \rfloor)c_j$ be the minimum cost to increase any floored term in S'.
4. Let $\beta_{\overline{J}} = \min_{j \in \overline{J}-U} c_j b'_i / A_{ij}$, where b'_i is the slack (b_i minus the value of the left-hand side of S'),
   be the min cost to increase the sum of fractional terms in S' to satisfy S'.
5. Return $\beta = \min\{\beta_J, \beta_{\overline{J}}\}$.

Figure 2.2: Implementation of $\mathsf{stepsize}(x, S)$ for CMIP(Alg. 2).

2.3 Nearly Linear-Time Implementation for Covering Mixed Integer Linear Programs

Theorem 4 *For* CMIP *(covering mixed integer linear programs with upper bounds), the greedy algorithm (Alg. 1) can be implemented to return a δ-approximation in $O(N \log \delta)$ time, where δ is the maximum number of non-zeros in any constraint and N is the total number of non-zeros in the constraint matrix.*

Proof. Fix any CMIP instance $\min\{c \cdot x : x \in \mathbb{R}^n_+; Ax \geq b; x \leq u; x_j \in \mathbb{Z}\ (j \in I)\}$. Model each constraint $A_i x \geq b_i$ using a monotone constraint $S \in \mathcal{C}$ of the form

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

where set I contains the indexes of the integer variables.

Given such a constraint S and an $x \notin S$, the subroutine $\mathsf{stepsize}(x, S)$ (Alg. 2) computes a step size β satisfying Thm. 1 as follows. Let S', J, U, β_J, $\beta_{\overline{J}}$, and β be as in Alg. 2. That is, $S' = S(J, A_i, u, b_i)$ is the relaxation of $S(I, A_i, u, b_i)$ obtained by relaxing the floors in S (in order of increasing A_{ij}) as much as possible, while maintaining $x \notin S'$; $J \subseteq I$ contains the indexes j of variables whose floors are not relaxed. Increasing x to

satisfy S' requires (at least) either: (i) increasing $\sum_{j\in J-U} A_{ij}\lfloor x_j\rfloor$, at cost at least β_J, or (ii) increasing $\sum_{j\in \overline{J}-U} A_{ij}x_j$ by at least the slack b'_i of the constraint S', at cost at least $\beta_{\overline{J}}$. Thus, $\mathsf{distance}(x,S) \ge \mathsf{distance}(x,S') \ge \min\{\beta_J, \beta_{\overline{J}}\} = \beta$. This choice satisfies Thm. 1, so the algorithm returns a δ-approximate solution.

Lemma 5 *For any S, Alg. 1 calls $\mathsf{Step}(x,S)$ with $\beta = \mathsf{stepsize}(x,S)$ (from Alg. 2) at most $2|\mathsf{Vars}(S)|$ times.*

Proof. Let j be the index of the variable x_j that determines β in the algorithm (β_J in case (i) of the previous proof, or $\beta_{\overline{J}}$ in case (ii)). The step increases x_j by β/c_j. This may bring x_j to (or above) its upper bound u_j. If not, then, in case (i), the left-hand side of S' increases by at least A_{ij}, which, by the minimality of $J(x)$ and the ordering of I, is enough to satisfy S'. Or, in case (ii), the left-hand side increases by the slack b'_i (also enough to satisfy S'). Thus the step either increases the set $U(x)$ or satisfies S', increasing the set $J(x)$. ■

The naive implementations of $\mathsf{stepsize}()$ and $\mathsf{Step}()$ run in time $O(|\mathsf{Vars}(S)|)$ (after the A_{ij}'s within each constraint are sorted in preprocessing). By the lemma, with this implementation, the total time for the algorithm is $O(\sum_S |\mathsf{Vars}(S)|^2) \le O(N\delta)$. By a careful heap-based implementation, this time can be reduced to $O(N \log \delta)$ (Lemma 30 in the appendix). ■

2.4 Online Monotone Covering and Caching with Upgradable Hardware

Recall that in *online* MONOTONE COVERING, each constraint $S \in \mathcal{C}$ is revealed one at a time; an online algorithm must raise variables in x to bring x into the given S, without

knowing the remaining constraints. Alg. 1 (with, say, $\mathsf{Step}(x, S)$ taking β just large enough to bring $x \in S$; see Observation 2) can do this, so it yields a δ-competitive online algorithm.[3]

Corollary 6 *The greedy algorithm (Alg. 1) gives a δ-competitive online* Monotone Covering *algorithm.*

Example: generalized connection caching. As discussed in the introduction (following the formulation of weighted caching as online Set Cover from [6]) this result naturally generalizes a number of known results for paging, weighted caching, file caching, connection caching, etc. To give just one example, consider connection caching. A request sequence r is given online. Each request $r_t = (u_t, w_t)$ activates the connection (u_t, w_t) (if not already activated) between nodes u_t and w_t. If either node has more than k active connections, then one of them other than r_t (say r_s) must be closed at cost $\mathsf{cost}(r_s)$. Model this problem as follows. Let variable x_t indicate whether connection r_t is closed before the next request to r_t after time t, so the total cost is $\sum_t \mathsf{cost}(r_t) x_t$. For each node u and each time t, for any $(k+1)$-subset $Q \subseteq \{r_s : s \le t; u \in r_s\}$, at least one connection $r_s \in Q - \{r_t\}$ (where s is the time of the most recent request to r_s) must have been closed, so the following constraint[4] is met: $\sum_{r_s \in Q - \{r_t\}} \lfloor x_s \rfloor \ge 1$.

Corollary 6 gives the following k-competitive algorithm for online connection caching. When a connection request (u, w) occurs at time t, the connection is activated and x_t is set to 0. If a node, say u, has more than k active connections, the current x violates the constraint above for the set Q containing u's active connections. Node u applies the $\mathsf{Step}()$ subroutine for this constraint: it raises x_s for all the connections

[3]If the cost function is linear, in responding to S this algorithm needs to know S and the values of variables in S and their cost coefficients. For general submodular costs, the algorithm may need to know not only S, but *all* variables' values and the whole cost function.

[4]This presentation assumes that the last request must stay in cache. If not, don't subtract $\{r_t\}$ from Q in the constraints. The competitive ratio goes from k to $k+1$.

$r_s \in Q - \{r_t\}$ at rate $1/\mathsf{cost}(r_s)$ simultaneously, until some x_s reaches 1. It closes any such connection r_s.

2.4.1 Upgradable online problems.

Standard online caching problems model only the caching strategy. In practice other parameters (e.g., the size of the cache, the speed of the CPU, bus, network, etc.) must also be chosen well. In *upgradable* caching, the algorithm chooses not only the caching strategy, but also the hardware configuration. The hardware configuration is assumed to be determined by how much has been spent on each of some d components. The configuration is modeled by a vector $y \in \mathbb{R}_+^d$, where y_i has been spent so far on component i. Upgradable caching allows a fair ammount of flexibility as described in Section 1.2.2.3. For this problem, our greedy algorithm gives a $(d+k)$-competitive online algorithm.

Theorem 7 Upgradable caching *has a $(d+k)$-competitive online algorithm, where d is the number of upgradable components and k is the maximum number of files that can be held in the cache.*

Proof. Let variable y_i for $i = 1, \ldots, d$ denote the amount invested in component i, so that the vector y gives the current hardware configuration. Let x_t be the cost (if any) incurred for evicting the tth requested item r_t at any time before its next request. The total final cost is $\sum_i y_i + \sum_t x_t$. At time t, if some subset $Q \subseteq \{r_s : s \leq t\}$ of the items is not cachable, then at least one item $r_s \in Q - \{r_t\}$ (where s is the time of the most recent request to r_s) must have been evicted, so the following constraint is met:

$$\mathsf{cachable}_t(Q, y) \ \text{ or } \ \textstyle\sum_{r_s \in Q - \{r_t\}} \lfloor x_s / \mathsf{cost}(r_s, y) \rfloor \geq 1. \qquad S_t(Q)$$

The restrictions on $\mathsf{cachable}$ and cost ensure that this constraint is monotone in x and y.

The greedy algorithm initializes $y = \mathbf{0}$, $x = \mathbf{0}$ and $Q = \emptyset$. It caches the subset Q of requested items r_s with $x_s < \mathsf{cost}(r_s, y)$. To respond to request r_t (which adds r_t to the cache if not present), the algorithm raises each y_i and each x_s for r_s in $Q - \{r_t\}$ at unit rate. It evicts any r_s with $x_s \geq \mathsf{cost}(r_s, y)$, until $\mathsf{cachable}_t(Q, y)$ holds for the cached set Q. The degree[5] δ is the maximum size of $Q - \{r_t\}$, plus d for y. ■

This result generalizes easily to "upgradable" MONOTONE CACHING, where investing in some d components can relax constraints or reduce costs.

Restricting groups of items (such as segments within files). The `http` protocol allows retrieval of segments of files. To model this in this setting, consider each file f as a group of arbitrary segments (e.g. bytes or pages). Let x_t be the *number of segments* of file r_t evicted before its next request. Let $c(x_t)$ be the cost to retrieve the cheapest x_t segments of the file, so the total cost is $\sum_t c(x_t)$. Then, for example, to say that the cache can hold at most k segments total, add constraints of the form (for appropriate subsets Q of requests) $\sum_{s \in Q} \mathsf{size}(r_s) - \lfloor x_s \rfloor \leq k$ (where $\mathsf{size}(r_s)$ is the number of segments in r_s). When the greedy algorithm increases x_s to x'_s, the online algorithm evicts segments $\lfloor x_s \rfloor + 1$ through $\lfloor x'_s \rfloor$ of file r_s (assuming segments are ordered by cheapest retrieval).

Generally, any monotone restriction that is a function of just the *number* of segments evicted from each file (as opposed to which specific segments are evicted), can be modeled. (For example, *"evict at least 3 segments of r_s or at least 4 segments from r_t"*: $\lfloor x_s/3 \rfloor + \lfloor x_t/4 \rfloor \geq 1$.) Although the caching constraints constrain file segments, the competitive ratio will be the maximum number of files (as opposed to segments)

[5]The algorithm enforces just *some* constraints $S_t(Q)$; δ is defined w.r.t. the problem defined by those constraints.

subroutine rstep$_c(x, S)$ alg. 3

1. Fix an arbitrary probability $p_j \in [0, 1]$ for each $j \in \mathsf{Vars}(S)$. *. . . taking each* $p_j = 1$ *gives Alg. 1*
2. Choose a scalar step size $\beta \geq 0$.
3. For $j \in \mathsf{Vars}(S)$ with $p_j > 0$, let X_j be the max. s.t. raising x_j to X_j would raise $c(x)$ by $\leq \beta/p_j$.
4. For $j \in \mathsf{Vars}(S)$ with $p_j > 0$, with probability p_j, let $x_j \leftarrow X_j$. *. . . these events can be dependent if desired!*

---

subroutine stateless-rstep$_c(x, S, U)$: $\cdots$ *do rstep, and keep each* x_j *in its (countable) domain* U_j $\cdots$ alg. 4

1. For $j \in \mathsf{Vars}(S)$, let $X_j = \min\{z \in U_j; z > x_j\}$ (or $X_j = x_j$ if the minimum is undefined).
2. Let α_j be the increase in $c(x)$ that would result from increasing just x_j to X_j.
3. Do $\mathsf{rstep}_c(x, S)$, choosing any $\beta \in (0, \min_j \alpha_j]$ and $p_j = \beta/\alpha_j$ (or $p_j = 0$ if $X_j = x_j$).

Figure 2.3: Randomized and Stateless implementation of $\mathsf{stepsize}(x, S)$ (Alg. 3, Alg. 4).

referred to in any constraint.

2.5 Randomized Variant of Alg. 1 and Stateless Online Algorithm

This section describes a randomized, online generalization of Alg. 1. It has more flexibility than Alg. 1 in how it increases variables. This can be useful, for example, in distributed settings, in dealing with numerical precision issues, and in obtaining *stateless* online algorithms (an example follows).

The algorithm is Alg. 1, modified to call subroutine $\mathsf{rstep}_c(x, S)$ (shown in Alg. 3) instead of $\mathsf{Step}_c(x, S)$. The subroutine has more flexibility in incrementing x. Its step-size requirement is a bit more complicated.

Theorem 8 *For* Monotone Covering *suppose the randomized greedy algorithm terminates, and, in each step,* β *is at most* $\min\{E[c(x \uparrow_p \hat{x}) - c(x)] : \hat{x} \geq x; \hat{x} \in S\}$, *where* $x \uparrow_p \hat{x}$ *is a random vector obtained from* x *by raising* x_j *to* $\hat{x}_j$ *with probability* p_j *for*

each $j \in \mathsf{Vars}(S)$*. Then the algorithm returns a* δ*-approximate solution in expectation.*

If the objective $c(x)$ is linear, the required upper bound on β above simplifies to $\mathsf{distance}_{c'}(x, S)$ where $c'_j = p_j c_j$.

Proof. We claim that, in each step, the expected increase in $c(x)$ is at most δ times the expected decrease in $\mathsf{residual}(x)$. This implies (by the optional stopping theorem) that $E[c(x_{\text{final}})] \leq \delta \times \mathsf{residual}(\mathbf{0})$, proving the theorem.

Fix any step starting with a given x. Let (r.v.) x' be x after the step. Fix feasible $x^* \geq x$ s.t. $\mathsf{residual}(x) = c(x^*) - c(x)$. Inequality (2.1) holds; to prove the claim we show $E_{x'}[c(x' \wedge x^*) - c(x)] \geq \beta$. Since $x^* \geq x$ and $x' = x \uparrow_p X$, this is equivalent to $E[c(x \uparrow_p X) - c(x)] \geq \beta$.

(**Case 1.**) Suppose $X_k < x^*_k$ for some $k \in \mathsf{Vars}(S)$ with $p_k > 0$. Let y be obtained from x by raising just x_k to X_k. Then with probability p_k or more, $c(x \uparrow_p X) \geq c(y) \geq c(x) + \beta/p_k$. Thus the expectation is at least β.

(**Case 2.**) Otherwise, $X_j \geq x^*_j$ for all j with $p_j > 0$. Then $E[c(x \uparrow_p X) - c(x)] \geq E[c(x \uparrow_p x^*) - c(x)]$. Since $x^* \geq x$ and $x^* \in S$, this is at least β by the assumption on β. ■

A stateless online algorithm. As described in the introduction, when the variables have restricted domains ($x_j \in U_j$), Alg. 1 constructs x and then "rounds" x down to $\mu(x)$. In the online setting, Alg. 1 maintains x as constraints are revealed; meanwhile, it uses $\mu(x)$ as its current online solution. In this sense, it is not *stateless*. A stateless algorithm can maintain only one online solution, each variable of which should stay in its restricted domain.

Next we use Thm. 8 to give a stateless online algorithm. The algorithm generalizes the HARMONIC k-server algorithm as it specializes for paging and caching [106],

and Pitt's WEIGHTED VERTEX COVER algorithm [10]. Given an unsatisfied constraint S, the algorithm increases each x_j for $j \in \mathsf{Vars}(S)$ to its next largest allowed value, with probability inversely proportional to the resulting increase in cost. (The algorithm can be tuned to increase just one, or more than one, x_j. It repeats the step until the constraint is satisfied.)

Formally, the stateless algorithm is the randomized algorithm from Thm. 8, but with the subroutine $\mathsf{rstep}_c(x, S)$ replaced by **stateless-rstep**$_c(x, S, U)$ (in Alg. 4), which executes $\mathsf{rstep}_c(x, S)$ in a particular way. (A¡ technicality: if $0 \notin U_j$, then x_j should be initialized to $\min U_j$ instead of 0. This does not affect the approximation ratio.)

Theorem 9 *For* MONOTONE COVERING *with discrete variable domains as described above, there is a stateless randomized online δ-approximation algorithm.*

Proof. By inspection **stateless-rstep**$_c(x, S, U)$ maintains each $x_j \in U_j$.

We show that **stateless-rstep**$_c(x, S, U)$ performs $\mathsf{rstep}_c(x, S)$ in a way that satisfies the requirement on β in Thm. 8. Let $\hat{x}$ be as in the proof of Thm. 8, with the added restriction that each $\hat{x}_j \in U_j$. Since $\hat{x} \in S$ but $x \notin S$, there is a $k \in \mathsf{Vars}(S)$ with $\hat{x}_k > x_k$. Since $\hat{x}_k \in U_k$, the choice of X_k ensures $\hat{x}_k \geq X_k$. Let y be obtained from x by raising x_k to X_k. Then, $E[c(x \uparrow_p \hat{x}) - c(x)] \geq p_k[c(y) - c(x)] = p_k \alpha_k = \beta$, satisfying Thm. 8. ■

2.6 Relation to the Local-Ratio Method

The local-ratio method has most commonly been applied to problems with variables x_j taking values in $\{0, 1\}$ and with linear objective function $c \cdot x$ (see [14, 10, 16, 12]; for one exception, see [15]). In these cases, each step of the algorithm is typically

interpreted as modifying the problem by repeatedly *reducing* selected objective function weights c_j by some β. At the end, the x, where x_j is raised from 0 to 1 if $c_j = 0$, gives the solution. At each step, the weights to lower are chosen so that the change must decrease OPT's cost by at least β, while increasing the cost for the algorithm's solution by at most $\delta\beta$. This guarantees a δ-approximate solution.

In contrast, recall that Alg. 1 raises selected x_j's fractionally by β/c_j. At the end, x_j is rounded down to $\lfloor x_j \rfloor$. Each step costs $\beta\delta$, but reduces the *residual cost* by at least β.

For problems with variables x_j taking values in $\{0, 1\}$ and with linear objective function $c \cdot x$, Alg. 1 can be given the following straightforward local-ratio interpretation. Instead of raising x_j by β/c_j, reduce c_j by β. At the end, instead of setting x_j to $\lfloor x_j \rfloor$, set $x_j = 1$ if $c_j = 0$. With this reinterpretation, a standard local-ratio analysis applies.

To understand the relation between the two interpretations, let c' denote the modified weights in the above reinterpretation. The reinterpreted algorithm maintains the following invariants: Each modified weight c'_j stays equal to $c_j(1 - x_j)$ (for c and x in the original interpretation; this is the cost to raise x_j the rest of the way to 1). Also, the residual cost $\mathsf{residual}(x)$ in the original interpretation equals (in the reinterpreted algorithm) the minimum cost to solve the original problem but with weights c'.

This local-ratio reinterpretation is straightforward and intuitive for problems with $\{0, 1\}$ variables and a linear objective. But for problems whose variables take values in more general domains, it does not extend cleanly. For example, suppose a variable x_j takes values in $\{0, 1, 2, \ldots, u\}$. The algorithm cannot afford to reduce the weight c_j, and then, at termination, set x_j to u for j with $c_j = 0$ (this can lose a factor of u in the approximation). Instead, one has to reinterpret the modified weight c'_j as a vector

of weights $c'_j : \{1, \ldots, u\} \to \mathbb{R}_+$ where $c'_j(i)$ is the cost to raise x_j from $\max\{x_j, i-1\}$ to $\min\{x_j, i\}$ (initially $c'_j(i) = c_j$). When the original algorithm increases x_j by β/c_j, reinterpret this as leaving x_j at zero, but lowering the non-zero $c'_j(i)$ with minimum i by β. At the end, take x_j to be the maximum i such that $c'_j(i) = 0$. We show next that this approach is doable (if less intuitive) for MONOTONE COVERING.

At a high level, the local-ratio method requires only that the objective be decomposed into "locally approximable" objectives. The common weight-reduction presentation of local ratio described above gives one decomposition, but others have been used. A local-ratio analysis for an integer programming problem with non-$\{0,1\}$ variable domains, based on something like $\mathsf{residual}(x)$, is used in [15]. Here, the following decomposition (different than [15]) works:

Lemma 10 *Any algorithm returns a δ-approximate solution x provided there exist $\{c^t\}$ and r such that*

(a) for any x, $c(x) = c(\mathbf{0}) + r(x) + \sum_{t=1}^{T} c^t(x)$,

(b) for all t, and any x and feasible x^*, $c^t(x) \le c^t(x^*)\delta$,

(c) the algorithm returns x such that $r(x) = 0$.

Proof. Let x^* be an optimal solution. Applying properties (a) and (c), then (b), then (a),

$$c(x) \;=\; c(\mathbf{0}) + \textstyle\sum_{t=1}^{T} c^t(x) \;\le\; c(\mathbf{0})\delta + \sum_{t=1}^{T} c^t(x^*)\delta \;+\; r(x^*)\delta \;=\; c(x^*)\delta. \quad \blacksquare$$

Next we describe how to use the proof of Thm. 1 (based on residual cost) to generate such a decomposition.

Let $\mathsf{distance}(x, y) = c(x \vee y) - c(x)$ (the cost to raise x to dominate y).

For any x, define $c^t(x) = \mathsf{distance}(x^{t-1}, x) - \mathsf{distance}(x^t, x)$, where x^t is Alg. 1's x after t calls to $\mathsf{Step}()$.

Define $r(x) = \mathsf{distance}(x^T, x)$, where x^T is the algorithm's solution.

For linear c note $c^t(x) = \sum_j c_j \big|[0, x_j] \cap [x_j^{t-1}, x_j^t]\big|$, the cost for x "between" x^{t-1} and x^t.

Lemma 11 *These c^t and r have properties (a-c) from Lemma 10, so the algorithm gives a δ-approximation.*

Proof. Part (a) holds because the sum in (a) telescopes to

$\mathsf{distance}(\mathbf{0}, x) - \mathsf{distance}(x^T, x) = c(x) - c(\mathbf{0}) - r(x)$.

Part (c) holds because the algorithm returns x^T, and

$r(x^T) = \mathsf{distance}(x^T, x^T) = 0$.

For (b), consider the tth call to $\mathsf{Step}()$. Let β be as in that call.

The triangle inequality holds for $\mathsf{distance}()$, so, for any $\hat{x}$,

$c^t(\hat{x}) \le \mathsf{distance}_c(x^{t-1}, x^t) = c(x^t) - c(x^{t-1})$.

As proved in the proof of Thm. 1, $c(x^t) - c(x^{t-1})$ is at most $\beta\delta$.

Also in the proof of Thm. 1, it is argued that

$\beta \le \mathsf{distance}(x^{t-1}, \cap_{S \in \mathcal{C}} S) - \mathsf{distance}(x^t, \cap_{S \in \mathcal{C}} S)$.

By inspection that argument holds for any $x^* \in \cap_{S \in \mathcal{C}} S$, giving

$\beta \le \mathsf{distance}(x^{t-1}, x^*) - \mathsf{distance}(x^t, x^*)$.

The latter quantity is $c^t(x^*)$. Thus, $c^t(\hat{x}) \le \beta\delta \le c^t(x^*)\delta$. ■

Chapter 3

Distributed Covering

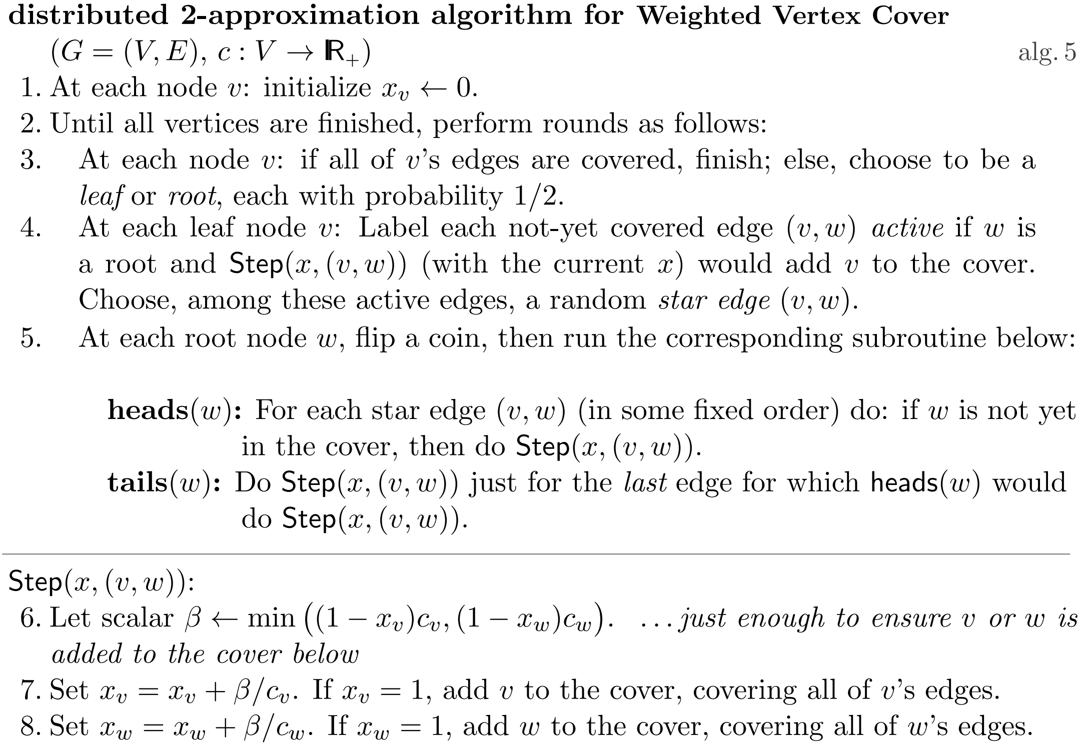
In this chapter we give distributed implementations of the basic sequential algorithm[1]. We start by giving a 2-approximation algorithm for WEIGHTED VERTEX COVER and CMIP with at most two variables per constraint. Then we show a distributed δ-approximation algorithm for MONOTONE COVERING.

3.1 Distributed Weighted Vertex Cover

3.1.1 Distributed Model

We assume the the standard synchronous communication model, where in each round, nodes can exchange a constant number of messages with neighbors, and perform some local computation [104]. We assume no restriction on message size and local computation. The goal here is to finish in a poly-logarithmic number of rounds.

[1]The results in this chapter appear in:
C. Koufogiannakis and N.E. Young. Distributed and parallel algorithms for weighted vertex cover and other covering problems. *In the twenty-eighth ACM symposium Principles of Distributed Computing*, pages 171–179, 2009. DOI http://doi.acm.org/10.1145/1582716.1582746. ©2009 ACM.



distributed 2-approximation algorithm for Weighted Vertex Cover $(G = (V, E),\ c : V \to \mathbb{R}_+)$ alg. 5

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 $x_v = x_v + \beta/c_v$. If $x_v = 1$, add v to the cover, covering all of v's edges.
8. Set $x_w = x_w + \beta/c_w$. If $x_w = 1$, add w to the cover, covering all of w's edges.

Figure 3.1: Distributed 2-approximation algorithm for Weighted Vertex Cover (Alg. 5).

3.1.2 Distributed Algorithms for Weighted Vertex Cover

Theorem 12 *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.

First, consider the sequential 2-approximation algorithm (Alg. 1) for WEIGHTED VERTEX COVER. 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.

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.

Each root w then flips a coin. If heads comes up (with probability 1/2), w does $\mathsf{heads}(w)$: that is, it 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 does $\mathsf{tails}(w)$: that is, it 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. 5.

Proof of Thm. 12, part (a). 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 $\mathsf{Step}(x, (v, w))$ 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. 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.)

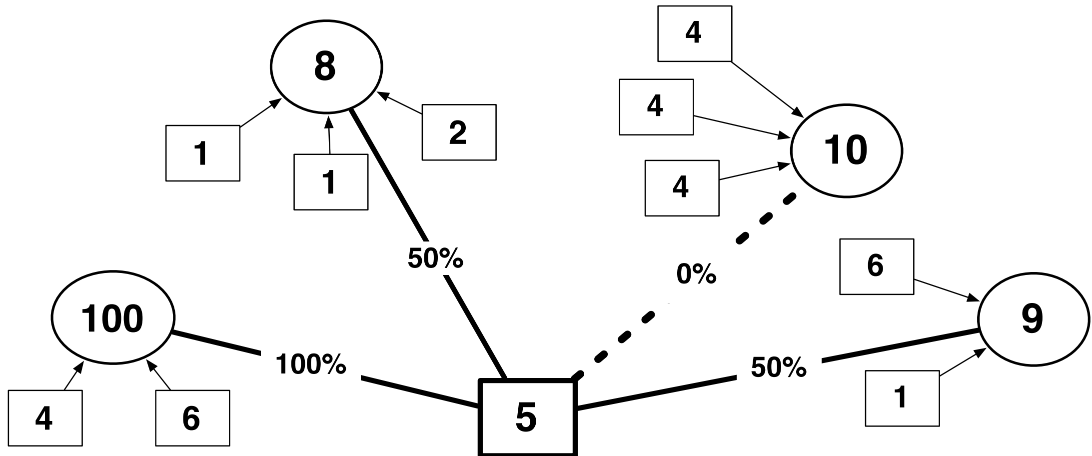


Figure 3.2: Analysis of Alg. 5. Each node is labeled with its cost. Roots are circles; leaves are squares; star edges from leaves other than v (the cost-5 leaf) are determined as shown. Each edge (v, w) is labeled with the chance that v would enter the cover if v were to choose (v, w) for its star edge (assuming each $x_w = x_v = 0$ and each root w considers its star edges counter-clockwise).

(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. 3.2, 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), $\mathsf{heads}(w)$ 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. 3.2, 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. 3.2, 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 a 50% chance of doing $\mathsf{heads}(w)$ and a 50% chance of doing $\mathsf{tails}(w)$. 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 $O(\log n)$, both in expectation and with high probability. This completes the proof of Thm. 12, part (a).

Parallel (RNC) implementation

Proof of Thm. 12, 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. 12.

3.2 Distributed Mixed Integer Programs with Two Variables per Constraint

Next we generalize the results of Section 3.1 to CMIP$_2$ (CMIP with at most two non-zero coefficients A_{ij} in each constraint).

Sequential Implementation

Consider first an implementation of the sequential δ-approximation algorithm (Alg. 1) 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 $A_i x \geq b$ by the following equivalent monotone constraint S_i:

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

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.

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.

As discussed in Section 2.1, 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 $\mathsf{distance}_c(x, S) = \min\{c(\hat{x}) - c(x) | \hat{x} \in S, \hat{x} \geq x\}$:

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 13 *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 necessary condition for the algorithm to produce a 2-approximate solution:

Lemma 14 $\mathsf{stepsize}(x, S_i) \leq \mathsf{distance}_c(x, S_i)$

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 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 $\mathsf{stepsize}(x, S_i) \le \mathsf{distance}(x, S'_i) \le \mathsf{distance}(x, S_i)$. ■

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:

$$0.5x_1 + 3x_2 \ge 5 \tag{3.1}$$

$$0.5x_1 + 3\lfloor x_2 \rfloor \ge 5 \tag{3.2}$$

$$0.5x_1 + 3\min\{x_2, 1\} \ge 5 \tag{3.3}$$

$$0.5x_1 + 3\lfloor \min\{x_2, 1\} \rfloor \ge 5 \tag{3.4}$$

$$0.5\lfloor x_1 \rfloor + 3x_2 \ge 5 \tag{3.5}$$

$$0.5\lfloor x_1 \rfloor + 3\lfloor x_2 \rfloor \ge 5 \tag{3.6}$$

$$0.5\lfloor x_1 \rfloor + \min\{x_2, 1\} \ge 5 \tag{3.7}$$

$$0.5\lfloor x_1 \rfloor + 3\lfloor \min\{x_2, 1\} \rfloor \ge 5 \tag{3.8}$$

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 (3.1) or (3.5) 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 (3.2) or (3.6) 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 (3.3), (3.4), (3.7), or (3.8) 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 $x_2 = \lfloor \min\{x_2, 1\} \rfloor = 1$.

Distributed implementation

Theorem 15 *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.

To prove part (a) of Thm. 15, we describe a distributed implementation of the above sequential algorithm. The algorithm (Alg. 6) generalizes Alg. 5.

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. 5, a constant fraction of the edges were likely to be covered each round

distributed 2-approximation algorithm for CMIP$_2$ (c, A, b, u, I) alg. 6

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. 16), 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 $\mathsf{Step}(x, S_r)$.

Figure 3.3: Distributed 2-approximation algorithm for CMIP$_2$ (Alg. 6).

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 16 *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 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. 6 for details.

Lemma 17 *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 $\mathsf{heads}(w)$ 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. ■

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. 15, part (a).

Parallel (RNC) implementation

Proof of Thm. 15, 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. 15.

3.3 Distributed Monotone Covering

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

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

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. In what follows we describe a distributed implementation of Alg. 1.

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 18 *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.*

Proof. To start each phase, the algorithm finds large independent subsets of constraints by running one phase of Linial and Saks' (LS) decomposition algorithm [89][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 $\mathcal{R} \subseteq \mathcal{C}$ 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

[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 [85].

distributed algorithm for Monotone Covering alg. 7

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 $\mathsf{Step}(x, S)$ to satisfy constraints $S \in \mathcal{K}$.
7. Broadcast the variables to constraints in $\mathcal{K}$ and their neighbors.

Figure 3.4: Distributed δ-approximation algorithm for Monotone Covering (Alg. 7).

$O(k)$ 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, $\mathsf{stepsize}(x, S)$ 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 $O_d(\log|\mathcal{C}|)\sum_{i=0}^{\infty} 4^i/|\mathcal{C}|^{di} = O_d(\log|\mathcal{C}|)$. ■

The following corollary is a subsequent result of Thm. 18.

Corollary 19 *There is a distributed δ-approximation algorithm for* Set Cover*,* CMIP *and (non-metric)* Facility Location *taking $O(\log^2|\mathcal{C}|)$ communication rounds in expectation and with high probability.*

Chapter 4

Distributed Algorithm for Fractional Packing

In this chapter we describe how to extend the distributed algorithms for covering to compute solution for Fractional Packing and Maximum Weighted b-Matching[1]. In Section 4.1 we show a primal-dual algorithm that computes a δ-approximate solution for these problems. Then in Section 4.2 we show a distributed implementation for $\delta = 2$ and in Section 4.3 we show a distributed implementation for general δ.

4.1 Covering and packing

As a special case, Alg. 1 computes δ-approximate solutions for Fractional Covering problems of the form $\min\{\sum_{j=1}^{n} c_j x_j : \sum_{j=1}^{n} A_{ij} x_j \geq w_i \quad (\forall i = 1..m),\ x \in \mathbb{R}_+^n\}$. The linear programming dual of such a problem is the following Fractional

[1] The results in this chapter appear in:
C. Koufogiannakis and N.E. Young. Distributed fractional packing and maximum weighted b-matching via tail-recursive duality. *In the twenty-third International Symposium on Distributed Computing. Lecture Notes in Computer Science*, LNCS 5805:221–238, 2009. DOI http://dx.doi.org/10.1007/978-3-642-04355-0_23 The original publication is available at www.springerlink.com. ©Springer-Verlag Berlin Heidelberg 2009. With kind permission of Springer Science+Business Media.

PACKING problem: $\max\{\sum_{j=1}^{m} w_i y_i : \sum_{i=1}^{m} A_{ij} y_i \leq c_j \quad (\forall j = 1 \ldots n),\ y \in \mathbb{R}_+^m\}$. For packing, δ is the maximum number of packing constraints in which a packing variable appears, $\delta = \max_i |\{j|\ A_{ij} \neq 0\}|$.

Here we extend the sequential and distributed approximation algorithms for FRACTIONAL COVERING by [80] to compute δ-approximate solutions for FRACTIONAL PACKING using a non-standard primal-dual approach.

Notation. Let C_i denote the i-th covering constraint ($\sum_{j=1}^{n} A_{ij} c_j \geq w_i$) and P_j denote the j-th packing constraint ($\sum_{i=1}^{m} A_{ij} y_i \leq c_j$). Let $\mathsf{Vars}(S)$ denote the set of (covering or packing) variable indexes that appear in (covering or packing) constraint S. Let $\mathsf{Cons}(z)$ denote the set of (covering or packing) constraint indexes in which (covering or packing) variable z appears. Let $N(y_s)$ denote the set of packing variables that appear in the packing constraints in which y_s appears, that is, $N(y_s) = \{y_i | i \in \mathsf{Vars}(P_j)$ for some $j \in \mathsf{Cons}(y_s)\} = \mathsf{Vars}(\mathsf{Cons}(y_s))$.

Fractional Covering. We start by repeating Alg. 1 tailored for FRACTIONAL COVERING and in a way that eases the subsequent primal-dual analysis. Let x^t be the solution after the first t steps have been performed. (Initially $x^0 = \mathbf{0}$.) Given x^t, let $w_i^t = w_i - \sum_{j=1}^{n} A_{ij} x_j^t$ be the slack of C_i after the first t steps. (Initially $w^0 = w$.) The algorithm is given by Alg. 8.

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

Our general approach. Our general approach is to recast the analysis as a primal-dual analysis, showing that the algorithm (Alg. 8) implicitly computes a solution to the

Greedy δ-approximation algorithm for Fractional Covering alg. 8

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

Figure 4.1: Sequential δ-approximation algorithm for Fractional Covering (Alg. 8).

dual packing problem of interest here. To do this we use the tail-recursive approach implicit in previous local-ratio analyses [16].

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

Standard Primal-Dual approach does not work. For even simple instances, generating a δ-approximate primal-dual pair for the above greedy algorithm requires a non-standard approach. For example, consider $\min\{x_1 + x_2 + x_3 : x_1 + x_2 \geq 1,\ x_1 + x_3 \geq 5,\ x_1, x_2 \geq 0\}$. If the greedy algorithm (Alg. 8) does the constraints in *either* order and chooses β maximally, it gives a solution of cost 10. In the dual $\max\{y_{12} + 5y_{13} : y_{12} + y_{13} \leq 1,\ y_{12}, y_{13} \geq 0\}$, the only way to generate a solution of cost 5 is to set $y_{13} = 1$ and $y_{12} = 0$. A standard primal-dual approach would raise the dual variable for each covering constraint when that constraint is processed (essentially allowing a dual solution to be generated in an *online* fashion, constraint by constraint). That can't work here. For example, if the constraint $x_1 + x_2 \geq 1$ is covered first by setting $x_1 = x_2 = 1$,

then the dual variable y_{12} would be increased, thus preventing y_{13} from reaching 1.

Instead, assuming the step to cover $x_1 + x_2 \geq 1$ is done first, the algorithm should not increase any packing variable until a solution to the residual dual problem is computed. After this step the residual primal problem is $\min\{x'_1 + x'_2 + x'_3 : x'_1 + x'_2 \geq -1,\ x'_1 + x'_3 \geq 4,\ x'_1, x'_2 \geq 0\}$, and the residual dual problem is $\max\{-y'_{12} + 4y'_{13} : y'_{12} + y'_{13} \leq 1,\ y'_{12}, y'_{13} \geq 0\}$. Once a solution y' to the residual dual problem is computed (either recursively or as shown later in this section) *then* the dual variable y'_{12} for the current covering constraint should be raised maximally, giving the dual solution y for the current problem. In detail, the residual dual solution y' is $y'_{12} = 0$ and $y'_{13} = 1$ and the cost of the residual dual solution is 4. Then the variable y'_{12} is raised maximally to give y_{12}. However, since $y'_{13} = 1$, y'_{12} cannot be increased, thus $y = y'$. Although neither dual coordinate is increased at this step, the dual cost is increased from 4 to 5, because the weight of y_{13} is increased from $w'_{13} = 4$ to $w_{13} = 5$. In what follows we present this formally.

Fractional Packing. We show that the greedy algorithm for covering creates an ordering of the covering constraints for which it performs steps, which we can then use to raise the corresponding packing variables. Let t_i denote the time[2] at which a step to cover C_i was performed. Let $t_i = 0$ if no step was performed for C_i. We define the relation "$C_{i'} \prec C_i$" on two covering constraints $C_{i'}$ and C_i *which share a variable* and *for which the algorithm performed steps* to indicate that constraint $C_{i'}$ was done first by the algorithm.

Definition 20 *Let* $C_{i'} \prec C_i$ *if* $\mathsf{Vars}(C_{i'}) \cap \mathsf{Vars}(C_i) \neq \emptyset$ *and* $0 < t_{i'} < t_i$.

[2] In general by "time" we mean some reasonable way to distinguish in which order steps were performed to satisfy covering constraints. For now, the time at which a step was performed can be thought as the step number (line 3 at Alg. 8). It will be slightly different in the distributed setting.

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

Definition 21 *A reverse order of poset $\mathcal{D}$ is an order $C_{i_1}, C_{i_2}, \ldots, C_{i_k}$ (or equivalently $y_{i_1}, y_{i_2}, \ldots, y_{i_k}$) such that for $l > j$ either we have $C_{i_l} \prec C_{i_j}$ or the relation "$\prec$" is not defined for constraints C_{i_l} and C_{i_j} (because they do not share a variable).*

Then the following figure (Alg. 9) shows the sequential δ-approximation algorithm for Fractional Packing.

Greedy δ-approximation algorithm for fractional packing alg. 9

1. Run Alg. 8, recording the poset $\mathcal{D}$.
2. Let T be the number of steps performed by Alg. 8.
3. Initialize $y^T \leftarrow \mathbf{0}$, $t \leftarrow T$. *... note that t will be decreasing from T to 0*
4. Let Π be some reverse order of $\mathcal{D}$. *... any reverse order works, see Lemma 22*
5. For each variable $y_s \in \mathcal{D}$ in the order given by Π do:
6. Set $y^{t-1} = y^t$.
7. Raise y_s^{t-1} until a packing constraint that depends on y_s^{t-1} is tight, that is, set $y_s^{t-1} = \max_{j \in \mathsf{Cons}(y_i)} (c_j - \sum_{i=1}^m A_{ij} y_i^{t-1})$.
8. Set $t = t - 1$.
9. Return $y = y^0$.

Figure 4.2: Sequential δ-approximation algorithm for Fractional Packing (Alg. 9).

The algorithm simply considers the packing variables corresponding to covering constraints that Alg. 8 did steps for, and raises each variable maximally without violating the packing constraints. The order in which the variables are considered matters: *the variables should be considered in the reverse of the order in which steps were done for*

the corresponding constraints, or an order which is "equivalent" (see Lemma 22). (This flexibility is necessary for the distributed setting.)

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

Lemma 22 *Alg. 9 returns the same solution y using (at line 4) any reverse order of $\mathcal{D}$.*

Proof. Let $\Pi = y_{j_1}, y_{j_2}, \ldots, y_{j_m}$ and $\Pi' = y_{j'_1}, y_{j'_2}, \ldots, y_{j'_m}$ be two different reverse orders of $\mathcal{D}$. Let $y^{\Pi,1\ldots k}$ be the solution computed so far by Alg. 9 after raising the first k packing variables of order Π. We prove that $y^{\Pi,1\ldots m} = y^{\Pi',1\ldots m}$.

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

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

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

So in any case, for each $y_j \in N(y_s)$, we have $r_j^{\Pi,1\ldots q} = y_j^{\Pi',1\ldots l}$ and thus $y_s^{\Pi,1\ldots q} = y_s^{\Pi',1\ldots l}$.

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

The following lemma and weak duality prove that the solution y returned by Alg. 9 is δ-approximate.

Lemma 23 *For the solutions x and y returned by Alg. 8 and Alg. 9 respectively, we have $\sum_{i=1}^{m} w_i y_i \geq 1/\delta \sum_{j=1}^{n} c_j x_j$.*

Proof. Lemma 22 shows that any reverse order of $\mathcal{D}$ produces the same solution, so w.l.o.g. here we assume that the reverse order Π used by Alg. 9 is the reverse of the order in which steps to satisfy covering constraints were performed by Alg. 8.

When Alg. 8 does a step to satisfy the covering constraint C_s (by increasing x_j by β_s/c_j for all $j \in \mathsf{Vars}(C_s)$), the cost of the covering solution $\sum_j c_j x_j$ increases by at most $\delta\beta_s$, since C_s depends on at most δ variables ($|\mathsf{Vars}(C_s)| \leq \delta$). Thus the final cost of the cover x is at most $\sum_{s \in \mathcal{D}} \delta\beta_s$.

Define $\Psi^t = \sum_i w_i^t y_i^t$ to be the cost of the packing y^t. Recall that $y^T = \mathbf{0}$ so $\Psi^T = 0$, and that the final packing solution is given by vector y^0, so the the cost of the final packing solution is Ψ^0. To prove the theorem we have to show that $\Psi^0 \geq \sum_{s \in \mathcal{D}} \beta_s$. We have that $\Psi^0 = \Psi^0 - \Psi^T = \sum_{t=1}^{T} \Psi^{t-1} - \Psi^t$ so it is enough to show that $\Psi^{t-1} - \Psi^t \geq \beta_s$ where C_s is the covering constraint done at the t-th step of Alg. 8. Then, $\Psi^{t-1} - \Psi^t$ is

$$\sum_i (w_i^{t-1} y_i^{t-1} - w_i^t y_i^t) \tag{4.1}$$

$$= w_s^{t-1}\ y_s^{t-1}\ +\ \sum_{i \neq s} (w_i^{t-1} - w_i^t) y_i^{t-1} \tag{4.2}$$

$$= w_s^{t-1}\ y_s^{t-1}\ +\ \sum_{j \in \mathsf{Cons}(y_s)} \sum_{i \in \{\mathsf{Vars}(P_i) - s\}} A_{ij} \frac{\beta_s}{c_j} y_i^{t-1} \tag{4.3}$$

$$= \beta_s y_s^{t-1} \max_{j \in \mathsf{Cons}(y_s)} \frac{A_{sj}}{c_j}\ +\ \sum_{j \in \mathsf{Cons}(y_s)} \sum_{i \in \{\mathsf{Vars}(P_j) - s\}} A_{ij} \frac{\beta_s}{c_j} y_i^{t-1} \tag{4.4}$$

$$\geq \beta_s\ \frac{1}{c_j} \sum_{i=1}^{m} A_{ij} y_i^{t-1} \qquad \text{(for } j \text{ s.t. constraint } P_j \text{ becomes tight after raising } y_s) \tag{4.5}$$

$$= \beta_s \tag{4.6}$$

In equation (4.2) we use the fact that $y_s^t = 0$ and $y_i^{t-1} = y_i^t$ for all $i \neq s$. For equation (4.3), we use the fact that the residual weights of packing variables in $N(y_s)$ are increased. If $x_i > 0$ for $i \neq s$, then x_i was increased before y_s ($y_s \prec y_i$) so at the current step $w_i^{t-1} > w_i^t > 0$, and $w_i^{t-1} - w_i^t = \sum_{j \in \mathsf{Cons}(y_s)} A_{ij} \frac{\beta_s}{c_j}$. For equation (4.4), by the definition of β_s we have $w_s^{t-1} = \beta_s \max_{j \in \mathsf{Cons}(y_s)} \frac{A_{sj}}{c_j}$. In inequality (4.5) we keep only the terms that appear in the constraint P_j that gets tight by raising y_s. The last equality holds because P_j is tight, that is, $\sum_{i=1}^m A_{ij} y_i = c_j$. ∎

The following lemma shows that Alg. 9 returns integral solutions if the coefficients A_{ij} are 0/1 and the c_j's are integers, thus giving a δ-approximation algorithm for Maximum Weighted b-Matching.

Lemma 24 *If $A \in \{0,1\}^{m \times n}$ and $c \in \mathbb{Z}_+^n$ then the returned packing solution y is integral, that is, $y \in \mathbb{Z}_+^m$.*

Proof. Since all non-zero coefficients are 1, the packing constraints are of the form $\sum_{i \in \mathsf{Vars}(P_j)} y_i \leq c_j \quad (\forall i)$. We prove by induction that $y \in \mathbb{Z}_+^m$. The base case is trivial

since the algorithm starts with a zero solution. Assume that at some point we have $y^t \in \mathbb{Z}_+^m$. Let $y_s \in \mathcal{D}$, be the next packing variable to be raised by the algorithm. We show that $y_s^{t-1} \in \mathbb{Z}_+$ and thus the resulting solution remains integral. The algorithm sets $y_s^{t-1} = \min_{j \in \mathsf{Cons}(y_s)} \{c_j - \sum_{i \in \mathsf{Vars}(P_j)} y_i^{t-1}\} = \min_{j \in \mathsf{Cons}(y_s)} \{c_j - \sum_{i \in \mathsf{Vars}(P_j)} y_i^t\} \geq 0$. By the induction hypothesis, each $y_i^t \in \mathbb{Z}_+$, and since $c \in \mathbb{Z}_+^n$, then y_s^{t-1} is also a non-negative integer. ■

4.2 Distributed Fractional Packing with $\delta = 2$

4.2.1 Distributed model for $\delta = 2$

We assume the network in which the distributed computation takes place has vertices for covering variables (packing constraints) and edges for covering constraints (packing variables). So, the network has a node u_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 $A_{ij}x_j + A_{ij'}x_{j'} \geq w_i$ ($\delta = 2$ so there can be at most 2 variables in each covering constraint). We assume the standard synchronous communication model, where in each round, nodes can exchange messages with neighbors, and perform some local computation [104]. We also assume no restriction on message size and local computation.

4.2.2 Distributed algorithm for $\delta = 2$

In this section we augment Alg. 6 (distributed algorithm for covering with $\delta = 2$) to compute 2-approximate solutions to the dual fractional packing problem without increasing the time complexity. The high level idea is similar to that in the previous section: run the distributed algorithm for covering to get a partial order of the

covering constraints for which steps were performed, then consider the corresponding dual packing variables in "some reverse" order raising them maximally. The challenge here is that the distributed algorithm for covering can perform steps for many covering constraints in parallel. Moreover, each covering constraint, has just a local view of the ordering, that is, it only knows its relative order among the covering constraints with which it shares variables.

Note that Alg. 6 proceeds in rounds, and within each round it covers a number of edges. Then, 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 20, we redefine "$\prec$" as follows.

Definition 25 *Let $C_{i'} \prec C_i$ (or equivalently $y_{i'} \prec y_i$) if $\mathsf{Vars}(C_{i'}) \cap \mathsf{Vars}(C_i) \neq \emptyset$ (i' and i are adjacent edges in the distributed network) and ([$0 < t_{i'}^R < t_i^R$] or [$t_{i'}^R = t_i^R$ and $t_{i'}^S < t_i^S$]).*

The pair (t_i^R, t_i^S) is adequate to distinguish which of two adjacent edges had a step to satisfy its covering constraint performed first. Adjacent edges can have their covering constraints done in the same round only if they belong to the same star (they have a common root), thus they differ in t_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 done by the distributed algorithm for covering.

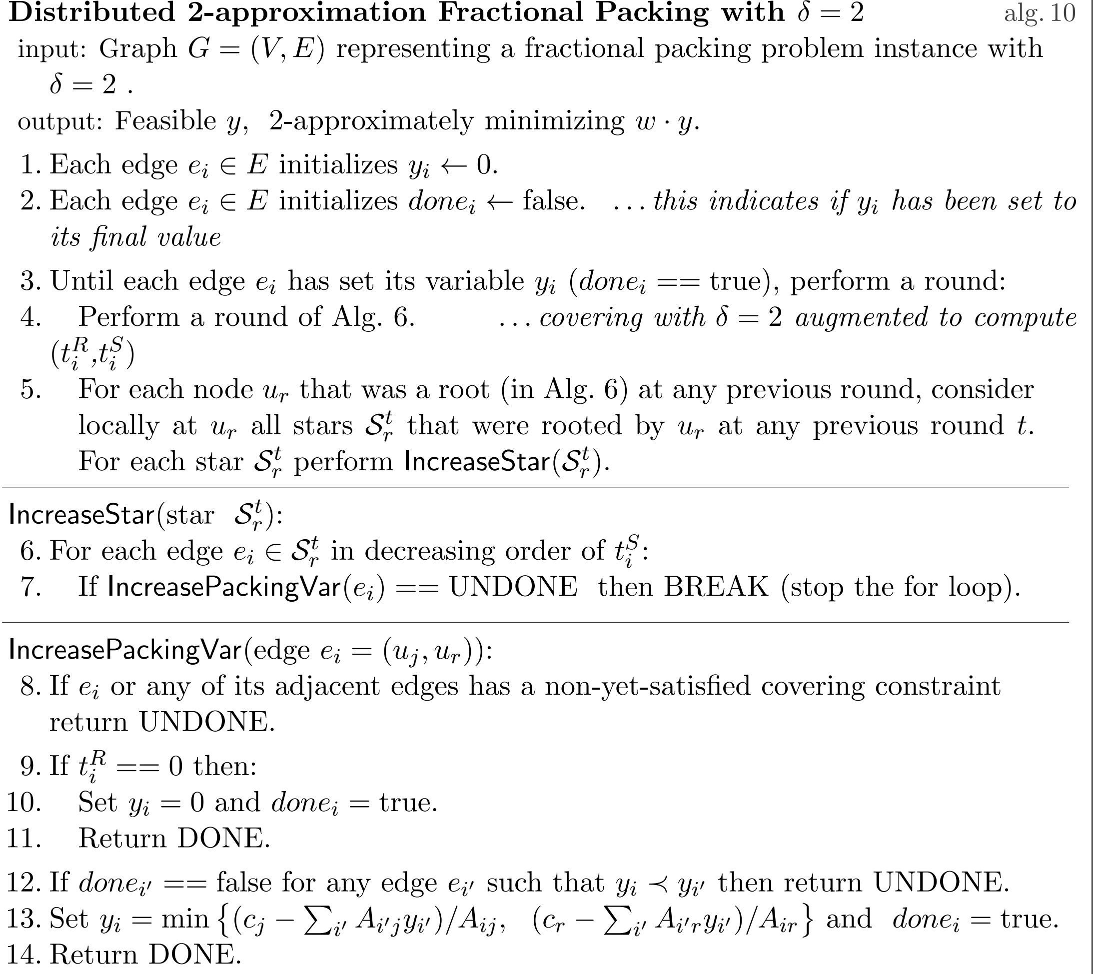
Distributed Fractional Packing with $\delta = 2$. Alg. 10 implements Alg. 9 in a distributed fashion. First, it runs Alg. 6 and recording $\mathcal{D}$. Meanwhile, as it discovers the partial order $\mathcal{D}$, it begins the second phase of Alg. 9, raising each packing variable as soon as it can. Specifically it waits to set a given $y_i \in \mathcal{D}$ until after it knows that

(a) y_i is in $\mathcal{D}$, (b) for each $y_{i'} \in N(y_i)$ whether $y_i \prec y_{i'}$, and (c) each such $y_{i'}$ is set. In other words, (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. Subject to these constraints it sets y_i as soon as possible. Note that some nodes will be executing the second phase of the algorithm (packing) while some other nodes are still executing the first phase (covering). This is necessary because a given node cannot know when distant nodes are done with the first phase.

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

Theorem 26 *For* Fractional Packing *where each variable appears in at most two constraints there is a distributed 2-approximation algorithm running in $O(\log m)$ rounds in expectation and with high probability, where m is the number of packing variables.*

Proof. By Thm 15, Alg. 6 computes a covering solution x in $T = O(\log m)$ rounds in expectation and with high probability. At the same time, the algorithm sets (t_i^R, t_i^S) for each edge e_i for which it performs a step to cover C_i, and thus defining a poset $\mathcal{D}$ of edges. In the distributed setting the algorithm does not define a linear order because there can be edges with the same (t_i^R, t_i^S), that is, edges that are covered by steps done in parallel. However, since these edges must be non-adjacent, we can still think that the algorithm gives a linear order (as in the sequential setting), where ties between edges with the same (t_i^R, t_i^S) are broken arbitrarily (without changing $\mathcal{D}$). Similarly, we can analyze Alg. 10 as if it considers the packing variables in a reverse order of $\mathcal{D}$. Then,



Distributed 2-approximation Fractional Packing with $\delta = 2$ alg. 10

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. 6. *... 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. 6) 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 IncreaseStar($\mathcal{S}_r^t$).

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 IncreasePackingVar(e_i) == UNDONE then BREAK (stop the for loop).

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 UNDONE.
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 edge $e_{i'}$ such that $y_i \prec y_{i'}$ then return UNDONE.
13. Set $y_i = \min\left\{(c_j - \sum_{i'} A_{i'j} y_{i'})/A_{ij},\ \ (c_r - \sum_{i'} A_{i'r} y_{i'})/A_{ir}\right\}$ and $done_i =$ true.
14. Return DONE.

Figure 4.3: Distributed 2-approximation algorithm for Fractional Packing where each variable appears in at most 2 constraints and Maximum Weighted Matching on graphs (Alg. 10).

by Lemma 22 and Lemma 23 the returned solution y is 2-approximate.

We prove that the y can be computed in at most T extra rounds after the initial T rounds to compute x. First note that within a star, even though its edges are ordered according to t_i^S they can all set their packing variables in a single round if none of them waits for some adjacent edge packing variable that belongs to a different star. So in the rest of the proof we only consider the case were edges are waiting for adjacent edges that belong to different stars. Note that $1 \le t_i^R \le T$ for each $y_i \in \mathcal{D}$. Then, at round T, each y_i with $t_i^R = T$ can be set in this round because it does not have to wait

for any other packing variable to be set. At the next round, round $T+1$, each y_i with $t_i^R = T-1$ can be set; they are dependent only on variables $y_{i'}$ with $t_{i'}^R = T$ which have been already set. In general, packing variables with $t_i^R = t$ can be set once all adjacent $y_{i'}$ with $t_i^R \geq t+1$ have been set. Thus by induction on $t = 0, 1, \ldots$ a constraint C_i for which a step was done at round $T-t$ may have to wait until at most round $T+t$ until its packing variable y_i is set. Therefore, the total number of rounds until solution y is computed is $2T = O(\log m)$ in expectation and with high probability. ■

The following theorem is a direct result of Lemma 24 and Thm 26 and the fact that for this problem $|E| = O(|V|^2)$.

Theorem 27 *For* MAXIMUM WEIGHTED b-MATCHING *on graphs there is a distributed* 2-*approximation algorithm running in* $O(\log |V|)$ *rounds in expectation and with high probability.*

4.3 Distributed Fractional Packing with general δ

4.3.1 Distributed model for general δ

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[3]. 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 4.2. We assume the standard synchronous model with unbounded message size.

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

4.3.2 Distributed algorithm

We extend the distributed δ-approximation algorithm for (fractional) covering problems Alg. 7) to compute δ-approximation for packing. Similar to the $\delta = 2$ case, here we use the covering algorithm to get a poset of packing variables which we then consider in a reverse order, raising them maximally. 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 $\mathcal{D}$ of covering constraints for which it performs steps.

Distributed packing with general δ. The algorithm is very similar to the case $\delta = 2$. First it runs the distributed algorithm for covering, 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 it waits to set a given $y_i \in \mathcal{D}$ until after it knows that (a) y_i is in $\mathcal{D}$, (b) for each $y_{i'} \in N(y_i)$ whether $y_i \prec y_{i'}$, and (c) each such $y_{i'}$ is set. In other words, (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. Subject to these constraints it sets y_i as soon as possible.

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

Distributed δ-approximation Fractional Packing with general δ alg. 11

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. 7), 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 IncreaseComponent($\mathcal{K}_r$).

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

6. For each $i \in \mathcal{K}_r$ in decreasing order of t_i^S:
7. If IncreasePackingVar(i) == UNDONE then BREAK (stop the for loop).

IncreasePackingVar(i):

8. If C_i or any $C_{i'}$ that shares covering variables with C_i is not yet satisfied return UNDONE.
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 $y_i \prec y_{i'}$ then return UNDONE.
13. Set $y_i = \min_{j \in \mathsf{Cons}(y_i)} \big((c_j - \sum_{i'} A_{i'j} y_{i'})/A_{ij}\big)$ and $done_i =$ true.
14. Return DONE.

Figure 4.4: Distributed δ-approximation algorithm for Fractional Packing and Maximum Weighted Matching on hypergraphs (Alg. 11).

Theorem 28 *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.*

The proof is omitted because it is similar to the proof of Thm 26.

The following theorem is a direct result of Lemma 24 and Thm 28.

Theorem 29 *For* MAXIMUM WEIGHTED b-MATCHING *on hypergraphs, there is a distributed δ-approximation algorithm running in $O(\log^2 |E|)$ rounds in expectation and with high probability, where δ is the maximum hyperedge degree and $|E|$ is the number of hyperedges.*

Chapter 5

Conclusions

5.1 Summary of Results

In this thesis we present sequential and distributed approximation algorithms for covering problems.

First, we give a sequential δ-approximation algorithm for MONOTONE COVERING. The algorithm unifies, generalizes, and improves many previous algorithms for fundamental covering problems such as VERTEX COVER, SET COVER, FACILITY LOCATION, and COVERING MIXED-INTEGER LINEAR PROGRAMS with upper bound on the variables. The algorithm is also a δ-competitive algorithm for *online* MONOTONE COVERING, which generalizes online versions of the above-mentioned covering problems as well as many fundamental online paging and caching problems. As such it also generalizes many classical online algorithms, including LRU, FIFO, FWF, BALANCE, GREEDY-DUAL, GREEDY-DUAL SIZE (a.k.a. LANDLORD), and algorithms for connection caching, where δ is the cache size. It also gives new δ-competitive algorithms for *upgradable* variants of these problems, which model choosing the caching strategy *and* an appropriate hardware configuration (cache size, CPU, bus, network, etc.).

Then we show distributed versions of the sequential algorithm. For WEIGHTED VERTEX COVER, we give the first distributed 2-approximation algorithm running in $O(\log n)$ rounds. The algorithm generalize to CMIP with two variables per constraint ($\delta = 2$). For any MONOTONE COVERING problem, we show the first distributed δ-approximation algorithm running $O(\log^2 |\mathcal{C}|)$ rounds, where $|\mathcal{C}|$ is the number of constraints.

Finally, we show distributed δ-approximate algorithms for FRACTIONAL PACKING and MAXIMUM WEIGHTED b-MATCHING, where δ is the maximum number of constraints in which a variable appears (for MAXIMUM WEIGHTED b-MATCHING δ is the maximum edge degree — for graphs $\delta = 2$). For $\delta = 2$ the algorithm runs in $O(\log m)$ rounds, where m is the number of packing variables. For general δ the algorithm runs in $O(\log^2 m)$ rounds.

5.2 Future Work

There are several open problems related to the problems presented in this thesis.

5.2.0.1 Sequential Setting

Our greedy δ-approximation for MONOTONE COVERING has a very simple analysis. However, we do not know of any primal-dual interpretation of the analysis. We already know how to analyze the algorithm using primal-dual for some special cases, and it seems that traditional primal-dual techniques do not work. Even simple special cases require a "tail-recursive" primal-dual approach to compute a δ-approximate primal-dual pair.

5.2.0.2 Distributed Setting

A long lasting question is how powerful randomization is in the distributed setting. There are cases where randomized algorithms are faster or give a better approximation ratio compared to the corresponding deterministic algorithms. For example, for Maximal Matching (and thus for a 2-approximation algorithm for Vertex Cover) the fastest known deterministic algorithm takes $O(\log^4 n)$ rounds [58], while the fastest randomized algorithm runs in $O(\log n)$ rounds (Alg. 5). Figuring out when randomization is necessary to provide better algorithms, or finding out deterministic algorithms that match the performance of the corresponding randomized ones is a very interesting research problem.

There are several more important open problems in the area of distributed algorithms. Among others, for Maximum Weighted Matching find an $(1+\varepsilon)$-approximation algorithm that runs in $O(\log n)$ number of rounds. For Weighted Vertex Cover find a deterministic 2-approximation algorithm that runs in a number of rounds that is independent of the total number of nodes in the network (i.e. a running time that is only a function of the maximum node degree).

Bibliography

[1] S. Albers. Generalized connection caching. *In the twelfth ACM Symposium on Parallel Algorithms and Architectures*, pages 70–78, 2000.

[2] S. Albers. On generalized connection caching. *Theory of Computing Systems*, 35(3):251–267, 2002.

[3] Susanne Albers and Pascal Weil, editors. *STACS 2008, 25th Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, February 21-23, 2008, Proceedings*, volume 08001 of *Dagstuhl Seminar Proceedings*. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

[4] N. Alon, B. Awerbuch, and Y. Azar. The online set cover problem. *In the thirty-fifth ACM Symposium on Theory Of Computing*, pages 100–105, 2003.

[5] E. Balas. A sharp bound on the ratio between optimal integer and fractional covers. *Mathematics of Operations Research*, 9(1):1–5, 1984.

[6] N. Bansal, N. Buchbinder, and J. Naor. A primal-dual randomized algorithm for weighted paging. *In the forty-third IEEE symposium on Foundations Of Computer Science*, pages 507–517, 2007.

[7] N. Bansal, N. Buchbinder, and S. Naor. Randomized competitive algorithms for generalized caching. In *the fortieth ACM Symposium on Theory Of Computing*, pages 235–244, 2008.

[8] J. Bar-Ilan, G. Kortsarz, and D. Peleg. Generalized submodular cover problems and applications. *Theoretical Computer Science*, 250(1-2):179–200, 2001.

[9] R. Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. *Approximation Algorithms for Combinatorial Optimization: International Workshop, APPROX'98*, 1998.

[10] R. Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. *Algorithmica*, 27(2):131–144, 2000.

[11] R. Bar-Yehuda. Using homogeneous weights for approximating the partial cover problem. *Journal of Algorithms*, 25:137–144, 2001.

[12] 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.

[13] 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.

[14] 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.

[15] R. Bar-Yehuda and D. Rawitz. Efficient algorithms for integer programs with two variables per constraint. *Algorithmica*, 29(4):595–609, 2001.

[16] 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.

[17] B. Berger, J. Rompel, and P. Shor. Efficient nc algorithms for set cover with applications to learning and geometry. *In the thirty-fifth IEEE symposium on Foundations Of Computer Science*, pages 454–477, 1994.

[18] P. Berman and B. DasGupta. Approximating the online set multicover problems via randomized winnowing. *Theoretical Computer Science*, 393(1-3):54–71, 2008.

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

[20] A. Borodin, D. Cashman, and A. Magen. How well can primal-dual and local-ratio algorithms perform? In *the thirty-second International Colloquium on Automata, Languages and Programming*, pages 943–955. Springer, 2005.

[21] A. Borodin and R. El-Yaniv. *Online computation and competitive analysis.* Cambridge University Press New York, NY, USA, 1998.

[22] N. Buchbinder and J. Naor. Online primal-dual algorithms for covering and packing problems. *Lecture Notes in Computer Science*, 3669:689–701, 2005.

[23] P. Cao and S. Irani. Cost-aware www proxy caching algorithms. *In the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, pages 193–206, 1997.

[24] 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.

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

[26] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991.

[27] Fabián A. Chudak and Kiyohito Nagano. Efficient solutions to relaxations of combinatorial problems with submodular penalties via the lovász extension and non-smooth convex optimization. In *the eighteenth ACM-SIAM Symposium On Discrete Algorithms*, pages 79–88, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

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

[29] E. Cohen, H. Kaplan, and U. Zwick. Connection caching. *In the thirty-first ACM Symposium on Theory Of Computing*, pages 612–621, 1999.

[30] E. Cohen, H. Kaplan, and U. Zwick. Connection caching under various models of communication. *In the twelfth ACM Symposium on Parallel Algorithms and Architectures*, pages 54 –63, 2000.

[31] E. Cohen, H. Kaplan, and U. Zwick. Connection caching: Model and algorithms. *Journal of Computer and System Sciences*, 67(1):92–126, 2003.

[32] A. Czygrinow and M. Hańćkowiak. Distributed algorithm for better approximation of the maximum matching. *In the ninth international Computing and Combinatorics Conference*, pages 242–251, 2003.

[33] 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.

[34] 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.

[35] M. Demange and V.T. Paschos. Algorithms and models for the on-line vertex-covering. *Lecture Notes In Computer Science*, 2573:102–113, 2002.

[36] M. Demange and V.T. Paschos. On-line vertex-covering. *Theoretical Computer Science*, 332(1-3):83–108, 2005.

[37] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162:439–486, 2005.

[38] G. Dobson. Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Mathematics of Operations Research*, 7(4):515–531, 1982.

[39] D. Dubhashi, F. Grandoni, and A. Panconesi. *Distributed Approximation Algorithms via LP-duality and Randomization*, chapter 13. Taylor and Francis, 2007.

[40] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[41] J. Edmonds and E. L. Johnson. Matching: A well-solved class of integer linear programs. *Combinatorial Structures and Their Applications*, pages 89–92, 1970.

[42] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator, and N.E. Young. Competitive paging algorithms. *J. Algorithms*, 12:685–699, 1991.

[43] M.L. Fisher and L.A. Wolsey. On the Greedy Heuristic for Continuous Covering and Packing Problems. *SIAM Journal on Algebraic and Discrete Methods*, 3:584–591, 1982.

[44] T. Fujito. On approximation of the submodular set cover problem. *Operations Research Letters*, 25(4):169–174, 1999.

[45] R. Gandhi, S. Khuller, and A. Srinivasan. Approximation algorithms for partial covering problems. *Journal of Algorithms*, 53:55–84, 2004.

[46] Teo Gonzales, editor. *Approximation Algorithms and Metaheuristics*, chapter 4 (Greedy Methods). Taylor and Francis Books (CRC Press), 2007.

[47] P. Gopalan, H. Karloff, A. Mehta, M. Mihail, and N. Vishnoi. Caching with expiration times. *In the thirteenth ACM-SIAM Symposium On Discrete Algorithms*, pages 540–547, 2002.

[48] 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.

[49] 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.

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

[51] F. Grandoni, J. Könemann, J., and A. Panconesi. Distributed weighted vertex cover via maximal matchings. *ACM Transactions on Algorithms*, 1, 2008.

[52] S. Guha, R. Hassin, S. Khuller, and E. Or. Capacitated vertex covering. *Journal of Algorithms*, 48(1):257 – 270, 2003.

[53] A. Gupta, M. Pal, R. Ravi, and A. Sinha. Boosted sampling: approximation algorithms for stochastic optimization. *In the thirty-sixth ACM Symposium on Theory Of Computing*, pages 417–426, 2004.

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

[55] M. M. Halldórsson and J. Radhakrishnan. Greed is good: Approximating independent sets in sparse and bounded-degree graphs. *In the twenty-sixth ACM Symposium on Theory Of Computing*, pages 439–448, 1994.

[56] E. Halperin. Improved approximation algorithm for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31(5):1608–1623, 2002.

[57] Eran Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. In *the eleventh ACM-SIAM Symposium On Discrete Algorithms*, pages 329–337, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[58] 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.

[59] J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.

[60] A. Hayrapetyan, C. Swamy, and É. Tardos. Network design for information networks. In *the sixteenth ACM-SIAM Symposium On Discrete Algorithms*, pages 933–942. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2005.

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

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

[63] D.S. Hochbaum. *Approximation algorithms for NP-hard problems.* PWS Publishing Co. Boston, MA, USA, 1996.

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

[65] S. Hougardy and D.E. Vinkemeier. Approximating weighted matchings in parallel. *Information Processing Letters*, 99(3):119–123, 2006.

[66] S. Irani. Page replacement with multi-size pages and applications to web caching. *Algorithmica*, 33(3):384–409, 2002.

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

[68] 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.

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

[70] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.

[71] A.R. Karlin, C. Kenyon, and D. Randall. Dynamic tcp acknowledgment and other stories about e/(e-1). *Algorithmica*, 36(3):209–224, 2003.

[72] H. J. Karloff. A las vegas rnc algorithm for maximum matching. *Combinatorica*, 6(4):387–391, 1986.

[73] 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.

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

[75] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-ε. *Journal of Computer and System Sciences*, 74:335–349, 2008.

[76] 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.

[77] T. Kimbrel. Online paging and file caching with expiration times. *Theoretical Computer Science*, 268:119–131, 2001.

[78] 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.

[79] 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.

[80] 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.

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] N. Linial and M. Saks. Decomposing graphs into regions of small diameter. *In the second ACM-SIAM Symposium On Discrete Algorithms*, pages 320–330, 1991.

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

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

[91] 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.

[92] 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.

[93] Zvi Lotker, Boaz Patt-Shamir, and Dror Rawitz. Rent, lease or buy: Randomized algorithms for multislope ski rental. In Albers and Weil [3], pages 503–514.

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

[95] 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.

[96] L.A. McGeoch and D.D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(1):816–825, 1991.

[97] J. Mestre. Adaptive local ratio. In *the nineteenth ACM-SIAM Symposium On Discrete Algorithms*, pages 152–160. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 2008.

[98] B. Monien and E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22:115–123, 1985.

[99] 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.

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

[101] 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.

[102] J. B. Orlin. A faster strongly polynomial time algorithm for submodular function minimization. *In the twelfth conference on Integer Programming and Combinatorial Optimization*, pages 240–251, 2007.

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

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

[105] David Pritchard. Approximability of sparse integer programs. *In the seventeenth European Symposium on Algorithms*, Lecture Notes in Computer Science 5757:83–94, 2009.

[106] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. *IBM Journal of Research and Development*, 38(6):683–707, 1994.

[107] S. Rajagopalan and V. V. Vazirani. Primal-dual rnc approximation algorithms for set cover and covering integer programs. *SIAM Journal on Computing*, 28:525–540, 1999.

[108] R. Ravi and A. Sinha. Hedging uncertainty: Approximation algorithms for stochastic optimization problems. *Mathematical Programming*, 108(1):97–114, 2006.

[109] D. Shmoys and C. Swamy. Stochastic optimization is (almost) as easy as deterministic optimization. In *Forty-fifth IEEE symposium on Foundations Of Computer Science*, volume 45, pages 228–237. IEEE Computer Society Press, 2004.

[110] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

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

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

[113] R. Uehara and Z. Chen. Parallel approximation algorithms for maximum weighted matching in general graphs. *Information Processing Letters*, 76(1-2):13–17, 2000.

[114] V.V. Vazirani. *Approximation algorithms.* Springer, 2001.

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

[116] M. Wattenhofer and R. Wattenhofer. Distributed weighted matching. *In the eighteenth international symposium on Distributed Computing*, pages 335–348, 2004.

[117] N. E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11:525–541, 1994.

[118] N. E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.

[119] Neal E. Young. Greedy set-cover algorithms (part 7 of encyclopedia of algorithms). *Springer Encyclopedia of Algorithms*, 2008.

Appendix A

Proofs

Lemma 30 *The sequence of conservative basic steps needed to satisfy S_i can be implemented in time $O(|\mathsf{Vars}(S_i)|\log(\mathsf{Vars}(S_i))$.*

Proof. In the following, the universe of indexes j is restricted without further comment to $\mathsf{Vars}(S_i)$. The algorithm maintains the following data structures:

- A vector x^0 recording x at the start of the sequence and the elapsed time s since the start, from which $x_j = x_j^0 + s/c_j$ can be calculated.
- The slack of the relaxed constraint, $b' = b'_i(x) = b_i - \sum_{i \in J} A_{ij} \lfloor \min(x_j, u_j) \rfloor - \sum_{i \in \overline{J}} A_{ij} \min(x_j, u_j)$.
- A min-heap $q(\overline{J} - U)$ holding indices $j \in \overline{J} - U$, keyed by c_j/A_{ij}. Then $\beta_{\overline{J}}$ is the minimum key in $q(\overline{J} - U)$, times the slack b', so $\beta_{\overline{J}}$ can be found in $O(\log \delta)$ time.
- The current rate of change $\theta = \sum_{j \in \overline{J} - U} A_{ij}/c_j$ of the fractional part with respect to s. When s increases by β, this allows the reduction in the slack b' to be computed in constant time, as the reduction in the slack is $\theta\beta$.

- A max-heap $q(I - J)$ holding indices $j \in I - J$, keyed by A_{ij}. This allows indices to be added to J in order of decreasing A_{ij}.

- A min-heap $q(J - U)$ holding indices $j \in J - U$, keyed by the value of s when $\lfloor x_j \rfloor$ next increases, that is, by $s + c_j(1 + \lfloor x_j \rfloor - x_j) = c_j(1 + \lfloor x_j^0 + s/c_j \rfloor - x_j^0)$. Then β_J is the minimum key in $q(J - U)$ minus s, so β_J can be found in $O(\log \delta)$ time. Also, when s increases, all variables x_j with $j \in J - U$ whose terms $\lfloor x_j \rfloor$ increase can also be found in $O(\log \delta)$ time per variable.

- A min-heap $q(\overline{U})$ holding indexes $j \in \mathsf{Vars}(S_i) - U$, keyed by the value of s that will cause x_j to reach its upper bound u_j, that is, by $c_j(u_j - x_j^0)$. Then, when s increases, each variable reaching its upper bound can be found in $O(\log \delta)$ time.

At the start of the sequence of steps for S_i, the algorithm initializes the above data structures for $J = \emptyset$, $U = \{j : x_j \geq u_j\}$. This can be done in $O(|\mathsf{Vars}(S_i)| \log \delta)$ time.

The algorithm then repeats the conservative basic step, by repeating the following two steps in order until S_i is satisfied:

Augment J**.** While x satisfies the constraint for J (i.e., while $b' \leq 0$), add the next index to J as follows. If $I = J$ (that is, $q(I - J)$ is empty), halt, as the original constraint S_i is satisfied. Otherwise, find the element j in $q(I - J)$ with maximum key. Add j to J; update the queues, b', and θ accordingly. (Delete j from $q(I - J)$. If $j \notin U$ (i.e., $j \in q(\overline{U})$) add j to $q(J - U)$, delete j from $q(\overline{J} - U)$, increase the slack b' by $A_{ij}(x_j - \lfloor x_j \rfloor)$ and decrease the rate of change θ by A_{ij}/c_j. Otherwise ($j \in U$), increase b' by $A_{ij}(u_j - \lfloor u_j \rfloor)$.)

Increase s**.** Compute $\beta = \min(\beta_J, \beta_{\overline{J}})$ using $q(J - U)$, $q(\overline{J} - U)$, and b' as in their descriptions. Increase s by β. Decrease b' by $\theta\beta$. For each j in $q(J - U)$ with

key s (if any; these are the variables whose floors increase) decrease b' by A_{ij} and increase the key of j to $s + 1/c_j$.

For each variable x_j that reaches its upper bound (that is, has key s or less in $q(\overline{U})$), correct the updates in the previous two paragraphs as follows. If $j \in J$ (i.e., in $q(J - U)$), then reduce b' by $A_{ij}(\lfloor x_j \rfloor - \lfloor u_j \rfloor)$. Otherwise ($j \in \overline{J}$), reduce b' by $A_{ij}(x_j - u_j)$. Remove j from $q(\overline{U})$, and $q(J - U)$ or $q(\overline{J} - U)$.

For a given S_i, each loop within the augmentation step takes $O(\log \delta)$ time and adds a variable to J. The total time for all augmentation steps until S_i is satisfied is thus $O(|\mathsf{Vars}(S_i)| \log \delta)$.

The first paragraph of the step increasing s takes $O(\log \delta)$ time plus $O(\log \delta)$ time per variable whose floor increases. For each variable j whose floor increases, the slack decreases by at least A_{ij}, causing at least one variable to be added to J. This can happen $O(|\mathsf{Vars}(S_i)|)$ times (Lemma 5). Thus, the total time spent for a given S_i in the first paragraph of the step increasing s is $O(|\mathsf{Vars}(S_i)| \log \delta)$.

Since each variable reaches its upper bound only once, the total time spent in the second paragraph of the step throughout the course of the algorithm is $O(N \log \delta)$.

■