Greedy Δ-Approximation Algorithm for Covering with Arbitrary Constraints and Submodular Cost

Christos Koufogiannakis, Neal E. Young[⋆]

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

Abstract. This paper describes a greedy 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, facilities location, and integer and mixed-integer covering linear programs with upper bound on the variables.
The algorithm is also the first 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.).

1 Introduction

The classification of general techniques is an important research program within the field of approximation algorithms. What are the scopes of, and the relationships between, the various algorithm-design techniques such as the primal-dual method, the local-ratio method [5], and randomized rounding? Within this research program, an important question is which problems admit optimal and fast *greedy* approximation algorithms, and by what techniques [25, 11]?

We give here a single online 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, $\Delta = 2$.)

For some problems in the class, no greedy (or other) Δ-approximation algorithms were known. For others, previous greedy Δ-approximation algorithms were known, but with non-trivial and seemingly problem-specific analyses. For VERTEX COVER and SET COVER, in the early 1980's, Hochbaum gave an algorithm that rounds a solution to the standard LP relaxation [32]; Bar-Yehuda and Even gave a linear-time greedy algorithm [6]. A few years later, for SET MULTICOVER, Hall and Hochbaum gave a quadratic-time primal-dual algorithm [26]. In the late 1990's, Bertsimas and Vohra generalized all of these results with a quadratic-time primal-dual algorithm for covering integer programs (CIP), restricted to $\{0,1\}$-variables and integer constraint matrix A, and with approximation ratio $\max_i \sum_j A_{ij} \ge \Delta$ [10]. Most recently, in 2000, Carr et al. gave the first (and only previous) Δ-approximation for general CIP with $\{0,1\}$ variables [15].[1] 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 this work), [44] gives details of an extension to CIP with general upper bounds on the variables. Both [15] and [44] 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.

Online paging and caching algorithms are also (online) monotone covering problems, as they can be formulated as online SET COVER [2]. These problems also have a rich history (see Fig. 1, and [12]).

All of the classical covering problems above (vertex cover, set cover, mixed integer linear programs with variable upper bounds (CMIP) and others (facility location, probabilistic variants of these problems, etc.), as well as online

[⋆] Partially supported by NSF awards CNS-0626912, CCF-0729071.

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

problem	approximation ratio	method	where	comment	
VERTEX COVER	$2 - \ln\ln\widehat{\Delta}/\ln\widehat{\Delta}$	local ratio	[28]	see also [31, 7, 42, 27, 29, 21, 36]	
SET COVER	Δ	greedy; LP	[6]; [32, 31]	$\Delta = \max_i \lvert\{j : A_{ij} > 0\}\rvert$	$\star$
CIP, 0/1-variables	$\max_i \sum_j A_{ij}$	greedy	[10, 26]		$\star$
CIP	Δ	ellipsoid	[15, 44]	KC-ineq., high-degree-poly time	$\star$
MONOTONE COVER	Δ	greedy	[our §**2**]	$\min\{c(x) : x \in S\ (\forall S \in \mathcal{C})\}$	new
CMIP	Δ	greedy	[our §**3**]	near-linear-time implementation	new
FACILITY LOCATION	Δ	greedy	[our §**4**]	linear-time implementation	new
PROBABILISTIC CMIP	Δ	greedy	[our §**4**]	quadratic-time implementation	new
online problem	competitive ratio	deterministic online			
PAGING	$k = \Delta$	potential function	[46, 45]	e.g. LRU, FIFO, FWF, Harmonic	$\star$
CONNECTION CACHING	$O(k)$	reduction to paging	[18, 1]		$\star$
WEIGHTED CACHING	k	primal-dual	[50, 45]	e.g. Harmonic, Greedy-Dual	$\star$
FILE CACHING	k	primal-dual	[51, 14]	e.g. Greedy-Dual-Size, Landlord	$\star$
UNW. SET COVER	$O(\log(\Delta)\log(n/\mathsf{opt}))$	primal-dual	[13]	unweighted	
CLP	$O(\log n)$	fractional	[13]	$\min\{c \cdot x : Ax \geq b; x \leq u\}$,	
MONOTONE COVER	Δ	potential function	[our §**2**]	includes the above and CMIP...	new
UPGRADABLE CACHING	$d + k$	reduction to mono. cover	[our §**5**]	d components, k files in cache	new

Fig. 1. Some Δ-approximation covering algorithms and deterministic online algorithms. "$\star$" = generalized or strengthened here.

variants (paging, weighted caching, file caching, (generalized) connection caching, etc.) are special cases of what we call **monotone covering**. Formally, a monotone covering instance is specified by a collection $\mathcal{C} \subset 2^{\mathbb{R}_+}$ of constraints and a non-negative, non-decreasing, submodular[2] objective function, $c : \mathbb{R}^n_+ \rightarrow \mathbb{R}_+$. The problem is to compute $\min\{c(x) : x \in \mathbb{R}^n_+, (\forall S \in \mathcal{C})\ x \in S\}$. Each constraint $S \in \mathcal{C}$ must be monotone (i.e., closed upwards), 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 $x \in \mu^{-1}(S)$, where $\mu^{-1}(S) = \{x : \mu(x) \in S\}$ 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 $\mu_j(x) = \lfloor x_j \rfloor$.)[3]

Section 2 describes our greedy Δ-approximation algorithm (Alg. 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 (see [38]).

The running time depends on the implementation, which is problem specific, but can be fast. Section 2 describes linear-time implementations for vertex cover, set cover, and (non-metric) facility location. Section 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 [15].) Section 4 describes an extension to a *probabilistic* (two-stage) variant of monotone covering, which naturally has submodular cost. The implementation for this case takes time

[2] 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.

[3] 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 doesn't exist, let $\hat{x}_j$ be the one that does.

$O(N\widehat{\Delta}\log\Delta)$, where N is the number of non-zeros in the constraint matrix and $\widehat{\Delta}$ is the maximum number of constraints in which any variable appears. (For comparison, [30] gives a $\ln(n)$-approximation algorithm for the special case of probabilistic set cover; the algorithm is based on submodular-function minimization [43], resulting in high-degree-polynomial run-time.[4])

Section 5 discusses *online* monotone covering. Following [13], an online algorithm must maintain a current x; as constraints $S \in \mathcal{C}$ are revealed one by one, the algorithm must increase coordinates of x to satisfy $x \in S$. The algorithm can't decrease coordinates of x. An algorithm is Δ-competitive if $c(x)$ is at most Δ times the minimum cost of any solution x^* that meets all the constraints.

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 [46], Balance and Greedy Dual for weighted caching [16, 50], Landlord [51], a.k.a. Greedy Dual Size [14], for file caching, and algorithms for connection caching [18–20, 1]. 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 5 also illustrates the generality of online monotone covering by describing a $(k+d)$-competitive algorithm for a new class of **upgradable** caching problems. In upgradable caching, the online algorithm chooses not only which pages to evict, but also how to configure and upgrade the relevant hardware components (determining such parameters as the cache size, CPU, bus, and network speeds, etc.) 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 [35] and its "multislope" generalization [39] 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 6 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 [45]) and Pitt's weighted vertex-cover algorithm [4].

Section 7 concludes by discussing the relation of the analysis here to the primal-dual and local-ratio methods. As a rule of thumb, greedy approximation algorithms can generally be analysed naturally via the primal-dual method, and sometimes even more naturally via the local-ratio method. The results here extend many primal-dual and local-ratio results. We conjecture that it is possible, but unwieldy, to recast the analysis here via primal-dual. It can be recast as a local-ratio analysis, but in a non-traditional form.

For distributed implementations of Alg. 1 running in $O(\log^2 n)$ rounds (or $O(\log n)$ for $\Delta = 2$), see [38].

We assume throughout that the reader is familiar with classical covering problems [49, 33] as well as classical online paging and caching problems and algorithms [12].

Alternatives to Δ-Approximation: log-Approximations, Randomized Online Algorithms. In spite of extensive work, no $(2-\varepsilon)$-approximation algorithm for constant $\varepsilon > 0$ is yet known for vertex cover [28, 31, 7, 42, 27, 29, 21, 36]. For small Δ, it seems that Δ-approximation may be the best possible in polynomial time.

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 [34, 40, 17] (1975) and greedy $O(\log\max_j \sum_i A_{ij})$-approximation algorithms for CIP with $\{0,1\}$-variables and integer A [22, 24] (1982). Srinivasan gave $O(\log\widehat{\Delta})$-approximation algorithms for general CIP without variable upper bounds [47, 48] (2000); these were extended to CIP with variable upper bounds by Kolliopoulos et al. [37] (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.

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 rent-or-buy [35, 39], paging [23, 41], weighted caching [2, 14], connection caching [18], and file caching [3]. Some cases of online monotone covering (e.g. vertex cover) are unlikely to have better-than-Δ-competitive randomized algorithms. It would interesting to classify which cases admit better-than-Δ-competitive randomized online algorithms.

[4] [30] also mentions a 2-approximation for probabilistic vertex cover, without details.

```
greedy algorithm for monotone covering (monotone constraints $\mathcal{C}$, submodular objective $c$)    alg. 1
output: feasible $x \in S$ ($\forall S \in \mathcal{C}$), $\Delta$-approximately minimizing $c(x)$ (see Thm. 1)
 1. Let $x \leftarrow \mathbf{0}$.    ... $\Delta = \max_{S \in \mathcal{C}} |\mathsf{vars}(S)|$ 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; see the introduction.
subroutine step$_c(x, S)$:    ... makes progress towards satisfying $x \in S$.
 1. Choose a scalar step size $\beta \geq 0$.    ... choose $\beta$ 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 $\beta$.
 3. For $j \in \mathsf{vars}(S)$, let $x_j \leftarrow x'_j$.    ... if $c$ is linear, then $x'_j = x_j + \beta/c_j$ for $j \in \mathsf{vars}(S)$.
```

2 The Greedy Algorithm for Monotone Covering (Alg. 1)

Fix an instance of monotone covering. 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 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 $\beta|\mathsf{vars}(S)| \leq \beta\Delta$. We show that $\mathsf{residual}(x)$ decreases by at least β, so the invariant $c(x)/\Delta + \mathsf{residual}(x) \leq \mathsf{opt}$ holds, proving the theorem.

Let x and x', respectively, be x before and after a given step. Let feasible $x^* \geq x$ 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, $c(x') + c(x^*) \geq c(x' \vee x^*) + c(x' \wedge x^*)$. (Equality holds if c is separable (e.g. linear).)

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

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

$$\mathsf{residual}(x) - \mathsf{residual}(x') \geq c(x' \wedge x^*) - c(x). \tag{1}$$

To complete the proof, we show the right-hand side of (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 $c(x' \wedge x^*) \geq c(y) = c(x) + \beta$.

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

Thus, the right-hand side of (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 1 *Let β be the minimum step size so that* $\mathsf{step}(x, S)$ *brings x into S. Then* $\beta \leq \mathsf{distance}_c(x, S)$.

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 $S = \{x : x_1 + x_2 \geq 1\}$. 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 doesn't 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 $x \geq 0$ such that $\sum_j c_j \min(1, \lfloor x_j \rfloor) \geq 1$. 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.

As a warm-up, here are linear-time implementations for facility location, set cover, and vertex cover.

Theorem 2. *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 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).[5]

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 $d_{ij} = 0$. □

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

Theorem 3. *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-zeroes in any constraint and N is the total number of non-zeroes in the constraint matrix.*

Proof (sketch). 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

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

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 indices 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, $\mathsf{distance}(x, S) \geq \mathsf{distance}(x, S') \geq \min\{\beta_J, \beta_{\overline{J}}\} = \beta$. This choice satisfies Thm. 1, so the algorithm returns a Δ-approximate solution.

Lemma 1. *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 (sketch). 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 the increases the set $U(x)$ or satisfies S', increasing the set $J(x)$. □

[5] 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 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 minimum cost to increase the sum of fractional terms in $S'$ to satisfy $S'$.
  5. Return $\beta = \min\{\beta_J, \beta_{\overline{J}}\}$.
```

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 $O(\sum_S |\mathsf{vars}(S)|^2) \leq O(N\Delta)$. By a careful heap-based implementation, this time can be reduced to $O(N \log \Delta)$ (proof omitted). □

4 (Two-Stage) Probabilistic Monotone Covering

An instance of *probabilistic* monotone covering is specified by an instance $(c, \mathcal{C})$ of monotone covering, along with *activation* probabilities p_S for each constraint $S \in \mathcal{C}$ and a non-decreasing, submodular *first-stage* objective W. The first stage requires the algorithm to commit to a vector $x^S \in S$ for each $S \in \mathcal{C}$. In the second stage, the algorithm must pay to satisfy the activated constraints, where each constraint S is independently activated with probability p_S. The algorithm pays $c(\hat{x})$, where $\hat{x}$ is the minimal vector such that $\hat{x} \geq x^S$ for each active S ($\hat{x}_j = \max\{x^S_j : S \text{ active}\}$). The problem is to choose the first-stage vectors to minimize the first-stage cost $W(x^S : S \in \mathcal{C})$ plus the *expected* second-stage cost, $E[c(\hat{x})]$. This (expected) cost is submodular as long as c is.

Observation 2 *Probabilistic monotone covering reduces to monotone covering.*

Probabilistic CMIP is the special case where W is linear and the pair $(c, \mathcal{C})$ define a CMIP.

For example, consider a two-stage probabilistic facilities location problem specified by first-stage costs f^1, d^1, an activation probability p_i for each customer i, and second-stage costs f^2, c^2. The algorithm assigns to each customer i a facility $j(i) \in N(i)$ (those that can serve i), by setting $x_{ij(i)} = 1$ (satisfying constraints $\sum_{j \in N(i)} \lfloor x_{ij} \rfloor \geq 1$), then paying the first-stage cost $\sum_j f^1_j \max_i x_{ij} + \sum_{ij} d^1_{ij} x_{ij}$. Then, each customer i is activated with probability p_i. Facilities assigned to activated customers are opened by setting $\hat{x}_{ij} = 1$ if $x_{ij} = 1$ and i is active. The algorithm then pays the second-stage cost $\sum_j f^2_j \max_i \hat{x}_{ij} + \sum_{ij} d^2_{ij} \hat{x}_{ij}$. The algorithm should minimize its total expected payment. The degree $\Delta = \max_i |N(i)|$ is the maximum number of facilities that any given customer is eligible to use.

Theorem 4. *For probabilistic CMIP,*

(a) The greedy Δ-approximation algorithm can be implemented to run in $O(N\widehat{\Delta} \log \Delta)$ time, where $\widehat{\Delta}$ is the maximum number of constraints per variable and $N = \sum_{S \in \mathcal{C}} |\mathsf{vars}(S)|$ is the input size.

(b) When $p = \mathbf{1}$, it can be implemented to run in time $O(N \log \Delta)$ (generalizes CMIP and facilities location).

Proof (sketch). Let $X = (x^S)_{S \in \mathcal{C}}$ be the matrix formed by the first-stage vectors. Let random variable $\hat{x}$ be as described in the problem definition ($\hat{x}_j = \max\{\hat{x}^S_j : S \text{ active}\}$), so the problem is to choose X subject to $x^S \in S$ for each S to minimize $C(X) = W \cdot X + E[c \cdot \hat{x}]$. This function is submodular, increasing, and continuous in X.

To satisfy Thm. 1, the subroutine $\mathsf{step}(X, S)$ must compute the step size β to be at most $\mathsf{distance}(X, S)$ (the minimum possible increase in $C(X)$ required to satisfy S). For a given X and S, have $\mathsf{step}(X, S)$ compute β as follows. For a given X, the rate at which increasing x^S_j would increase $C(X)$ is

$$c'_j \;=\; w^S_j + c_j \Pr[x^S_j = \hat{x}_j] \;=\; w^S_j + c_j p_S \prod \{1 - p_R : x^R_j > x^S_j, j \in \mathsf{vars}(R)\}.$$

This rate does not change until x^S_j reaches $t_j = \min\{x^R_j : x^R_j > x^S_j, j \in \mathsf{vars}(R)\}$.

Take $\beta = \min(\beta_t, \mathsf{stepsize}_{c'}(x^S, S))$, where $\beta_t = \min\{(t_j - x^S_j)c'_j : j \in \mathsf{vars}(S)\}$ is the minimum cost to bring any x^S_j to its threshold, and $\mathsf{stepsize}()$ is the subroutine from Section 3, using the (linear) cost vector c' defined above.

This β is a valid lower bound on $\mathsf{distance}(X, S)$, because β_t is a lower bound on the cost to bring any x_j^S to its next threshold, while $\mathsf{stepsize}_{c'}(x^S, S)$ is a lower bound on the cost to satisfy S without bringing any x_j^S to its threshold.

If $\mathsf{step}(X, S)$ uses this β, the number of steps to satisfy S is at most $O(|\mathsf{vars}(S)|\widehat{\Delta})$. Each step either (i) makes some x_j^S reach its next threshold (and each x_j^S crosses at most $\widehat{\Delta}$ thresholds), or (ii) increases the number of "floored" variables or increases the number of variables at their upper bounds (which by the analysis of $\mathsf{stepsize}()$ from Section 3, can happen at most $2|\mathsf{vars}(S)|$ times). Thus, the total number of steps is $O(\sum_S |\mathsf{vars}(S)|\widehat{\Delta})$, that is, $O(N\widehat{\Delta})$. (Implementation details needed to achieve amortized time $O(\log \Delta)$ per step are omitted.) This completes the proof sketch for part (a).

For part (b) of the theorem, note that in this case the product in the equation for $C_j^S(X)$ is 1 if $x_j^S = \max_R x_j^R$ and 0 otherwise. Each variable has at most one threshold to reach, so the number of calls to $\mathsf{step}(X, S)$ is reduced to $O(|\mathsf{vars}(S)|)$. This allows an implementation in total time $O(N \log \Delta)$. □

5 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 1) can do this, so it yields a Δ-competitive online algorithm.[6]

Corollary 1. *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 [2]) 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[7] is met: $\sum_{r_s \in Q - \{r_t\}} \lfloor x_s \rfloor \ge 1$.

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

Remark on $k/(k-h+1)$-competitiveness. The classic ratio of $k/(k-h+1)$ (versus opt with cache size $h \le k$) can be reproduced in such a setting as follows. For any set Q as described above, opt must meet the stronger constraint $\sum_{r_s \in Q - \{r_t\}} \lfloor x_s \rfloor \ge k - h + 1$. In this scenario, the proof of Thm. 1 extends to show a ratio of $k/(k-h+1)$ (use that the variables are $\{0, 1\}$, so there are at least $k - h + 1$ variables x_j such that $x_j < x_j^*$).

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.

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 $\mathsf{cost}(r_s, y)$.

[6] 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.

[7] 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 $k + 1$.

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 doesn't 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 doesn't 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 degree $\Delta = k + d$, where k is the maximum number of files ever held in cache and d is the number of hardware components. This gives a simple $(k + d)$-competitive online algorithm for upgradable caching.

Theorem 5. *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 (sketch). 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[8] 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) referred to in any constraint.

[8] 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 $\alpha_j$ 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 $X_j = x_j$).
```

6 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 6. *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 (sketch). 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 $E[c(x_{\mathsf{final}})] \leq \Delta \times \mathsf{residual}(\mathbf{0})$, 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 (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. 6 to give a stateless online algorithm. The algorithm generalizes the Harmonic k-server algorithm as it specializes for paging and caching [45], and Pitt's weighted vertex cover algorithm [4]. 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. 6, 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 7. *For monotone covering with discrete variable domains as described above, there is a stateless randomized online Δ-approximation algorithm.*

Proof (sketch). 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. 6. Let $\hat{x}$ be as in the proof of Thm. 6, 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. 6. □

7 Relation to Primal-Dual and Local-Ratio Methods

Primal-Dual. Here we speculate about how Thm. 1 might be cast as a primal-dual analysis. Given a vector v, consider its "shadow" $s(v) = \{x : \exists_j x_j \geq v_j\}$. Any monotone set S is the intersection of the shadows of its boundary points: $S = \bigcap_{v \in \partial S} s(v)$. Thus, any monotone covering instance can be recast to use only shadow sets for constraints. Any shadow set $s(v)$ is of the form $s(v) = \{x : \sum_j \lfloor x_j / v_j \rfloor \geq 1\}$, a form similar to that of the CMIP constraints $S(I, A_i, u, b_i, d)$ in Section 3. We conjecture that the Knapsack Cover (KC) inequalities from [15] for CIP can be generalized to give valid inequalities with integrality gap Δ for constraints of this form. (Indeed, the result in Section 3 easily extends to handle such constraints.) This could yield an appropriate relaxation on which a primal-dual analysis could be based.

For even simple instances, generating a Δ-approximate primal-dual pair for the greedy algorithm here requires a "tail-recursive" dual solution implicit in some local-ratio analyses [9], as opposed to the typical forward-greedy dual solution.[9] Even if the above program (extended to non-linear cost functions!) can be carried out, it seems likely to lead to a less intuitive proof than that of Thm. 1.

Local-Ratio. 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 [7, 4, 9, 5]; for one exception, see [8]). 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 lowers 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 [8]. Here, the following decomposition (different than [8]) works:

Lemma 2. *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^*, $c^t(x) \leq c^t(x^*)\Delta$,
(c) the algorithm returns x such that $r(x) = 0$.

[9] For example, consider $\min\{x_1 + x_2 + x_3 : x_1 + x_2 \geq 1, x_1 + x_3 \geq 2\}$. If the greedy algorithm does the constraints in *either* order and chooses β maximally, it gives a solution of cost 4. In the dual $\max\{y_{12} + 2y_{13} : y_{12} + y_{13} \leq 1\}$, the only way to generate a solution of cost 2 is to set $y_{13} = 1$ and $y_{12} = 0$. If the primal constraint for y_{12} is considered first, y_{12} cannot be assigned a non-zero value. Instead, one should consider the dual variables for constraints for which steps were done, in the *reverse* order of those steps, raising each until a constraint is tight.

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

$$c(x) \;=\; c(\mathbf{0}) + \textstyle\sum_{t=1}^{T} c^t(x) \;\le\; c(\mathbf{0})\Delta + \textstyle\sum_{t=1}^{T} c^t(x^*)\Delta \;+\; r(x^*)\Delta \;=\; c(x^*)\Delta. \qquad \square$$

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 3. *These c^t and r have properties (a-c) from Lemma 2, 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, $c^t(\hat{x}) \le \beta\Delta \le c^t(x^*)\Delta$. □

References

1. S. Albers. On generalized connection caching. *Theory of Computing Systems*, 35(3):251–267, 2002.
2. 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.
3. 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.
4. R. Bar-Yehuda. One for the price of two: A unified approach for approximating covering problems. *Algorithmica*, 27(2):131–144, 2000.
5. 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.
6. 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.
7. 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.
8. R. Bar-Yehuda and D. Rawitz. Efficient algorithms for integer programs with two variables per constraint. *Algorithmica*, 29(4):595–609, 2001.
9. 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.
10. D. Bertsimas and R. Vohra. Rounding algorithms for covering problems. *Mathematical Programming: Series A and B*, 80(1):63–89, 1998.
11. 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.
12. A. Borodin and R. El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press New York, NY, USA, 1998.
13. N. Buchbinder and J. Naor. Online primal-dual algorithms for covering and packing problems. *Lecture Notes in Computer Science*, 3669:689–701, 2005.
14. 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.
15. 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.
16. M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. *SIAM J. Discrete Math.*, 4(2):172–181, 1991.
17. V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.
18. E. Cohen, H. Kaplan, and U. Zwick. Connection caching. *In the thirty-first ACM Symposium on Theory Of Computing*, pages 612–621, 1999.

19. 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.
20. E. Cohen, H. Kaplan, and U. Zwick. Connection caching: Model and algorithms. *Journal of Computer and System Sciences*, 67(1):92–126, 2003.
21. I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162:439–486, 2005.
22. G. Dobson. Worst-case analysis of greedy heuristics for integer programming with nonnegative data. *Mathematics of Operations Research*, 7(4):515–531, 1982.
23. 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.
24. 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.
25. Teo Gonzales, editor. *Approximation Algorithms and Metaheuristics*, chapter 4 (Greedy Methods). Taylor and Francis Books (CRC Press), 2007.
26. N.G. Hall and D.S. Hochbaum. A fast approximation algorithm for the multicovering problem. *Discrete Applied Mathematics*, 15(1):35–40, 1986.
27. 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.
28. E. Halperin. Improved approximation algorithm for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31(5):1608–1623, 2002.
29. J. Håstad. Some optimal inapproximability results. *Journal of the ACM*, 48(4):798–859, 2001.
30. 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.
31. D. S. Hochbaum. Efficient bounds for the stable set, vertex cover, and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.
32. D.S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11:555–556, 1982.
33. D.S. Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Co. Boston, MA, USA, 1996.
34. D. S. Johnson. Approximation algorithms for combinatorial problems. *In the fifth ACM Symposium On Theory Of Computing*, 25:38–49, 1973.
35. A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator. Competitive snoopy caching. *Algorithmica*, 3:77–119, 1988.
36. 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.
37. 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.
38. Christos Koufogiannakis and Neal E. Young. Distributed and parallel algorithms for weighted vertex cover and other covering problems. In *the twenty-eighth ACM symposium Principles of Distributed Computing*, 2009.
39. Zvi Lotker, Boaz Patt-Shamir, and Dror Rawitz. Rent, lease or buy: Randomized algorithms for multislope ski rental. In Susanne Albers and Pascal Weil, editors, *the twenty-fifth Symposium on Theoretical Aspects of Computer Science*, volume 08001 of *Dagstuhl Seminar Proceedings*, pages 503–514. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.
40. L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Math*, 13:383–390, 1975.
41. L.A. McGeoch and D.D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(1):816–825, 1991.
42. B. Monien and E. Speckenmeyer. Ramsey numbers and an approximation algorithm for the vertex cover problem. *Acta Informatica*, 22:115–123, 1985.
43. 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.
44. David Pritchard. Approximability of sparse integer programs. Technical report, arxiv.org, 2009. http://arxiv.org/abs/0904.0859.
45. P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. *IBM Journal of Research and Development*, 38(6):683–707, 1994.
46. D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
47. A. Srinivasan. Improved approximation guarantees for packing and covering integer programs. *SIAM Journal on Computing*, 29:648–670, 1999.
48. A. Srinivasan. New approaches to covering and packing problems. *In the twelveth ACM-SIAM Symposium On Discrete Algorithms*, pages 567–576, 2001.
49. V.V. Vazirani. *Approximation algorithms*. Springer, 2001.
50. N. E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11:525–541, 1994.
51. N. E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002.