15. Approximate Set Cover and Bin Packing

CPSC 535

Kevin A. Wortman

CALIFORNIA STATE UNIVERSITY FULLERTON

Set Cover: Intuition

- list of **needs**
- list of **services**
  - each service meets some of the needs
- **puzzle:** shortest list of products that meets all the needs?

Set Cover: Formal Definition

set cover problem
input: a universe set X, and family $\mathfrak{F}$ of subsets of X, such that $X = \bigcup_{S \in \mathfrak{F}} S$
output: a minimum size subfamily $\mathfrak{C} \subseteq \mathfrak{F}$ whose members cover all of X, so $X = \bigcup_{S \in \mathfrak{C}} S$

Application: Streaming Services

- **needs:** stream TV shows A, B, C, D, E, F
- $X = \{A, B, C, D, E, F\}$
- **services:** alternative streaming services; each offer only some shows
- $\mathfrak{F} = \{\{A, F\}, \{A, C, E\}, \{B, E\}, \ldots\}$
- **puzzle:** subscribe to smallest number of services that provide all desired shows

Application: Menu Design

- **needs:** menu has a food option available for various dietary needs
- $X = \{carnivore, vegan, kosher, halal, glutenfree, \ldots\}$
- **services:** alternative entrees
- $\mathfrak{F} = \{\{carnivore, halal\}, \{vegan\}, \{kosher, carnivore\}, \ldots\}$
- **puzzle:** design a menu with the fewest number of food options so that everyone can eat something

Set Cover Hardness

- set cover is *NP*-complete
- baseline algorithm: for each subset $\mathfrak{C} \subseteq \mathfrak{F}$, check if the sets in $\mathfrak{C}$ contain all elements, keep track of the smallest such $\mathfrak{C}$
- $\Theta(2^n \cdot n)$ time, slow

Set Cover Approximation Algorithm

```
function APPROX-SET-COVER(X, 𝔉)
    U_0 = ∅                                    ▷ still-uncovered elements
    ℭ = ∅
    i = 0
    while U_i ≠ ∅ do
        // choose set with most currently-uncovered elements
        Find S ∈ 𝔉 that maximizes |S ∩ U_i|
        U_{i+1} = U_i − S
        ℭ = ℭ ∪ {S}
        i = i + 1
    end while
    return ℭ
end function
```

Efficiency Analysis

- while loop: $\Theta(n)$ iterations
  - Find: $\Theta(n)$ time (assuming fast data structure to look up U_i)
  - $U_{i+1} = U_i - S$: $\Theta(n)$ time
- other steps: $\Theta(1)$ time each
- total time $\Theta(n^2)$
- can be sped up to $\Theta(n)$ (CLRS Exercise 35.3-3)

Approximation Ratio

Theorem: APPROX-SET-COVER is a $O(\lg n)$-approximation algorithm

Proof sketch:

- Let $\mathfrak{C}^\star$ be the optimal cover and $k^\star = |\mathfrak{C}^\star|$
- $\mathfrak{C}^\star$ covers all of X, and each $U_i \subseteq X$, so $\mathfrak{C}^\star$ covers each U_i
- each U_i can be covered with $\leq k^\star$ sets from $\mathfrak{F}$
- on average, $\mathfrak{C}^\star$ covers $n/k^\star$ elements/set
- so at least one set in $\mathfrak{F}$ covers $\geq n/k^\star$ elements
- APPROX-SET-COVER picks the set that covers the most elements, so each S covers at least $n/k^\star$ additional elements, and

$$U_{i+1} \leq |U_i| - |U_i|/k^\star = |U_i|(1 - 1/k^\star)$$

Approximation Ratio (continued)

$$U_{i+1} \leq |U_i|(1 - 1/k^\star)$$

- algorithm stops when some $|U_i| = 0$
- as a recurrence,
$$T(n) = (1 - 1/k^\star)n$$
- algebra and log rules show $T(n) \in O(k^\star \lg n)$
- each iteration adds one set to $\mathfrak{C}$, so APPROX-SET-COVER picks $O(k^\star \lg n)$ sets
- $k^\star$ is the optimal number of sets, so
- $\therefore$ APPROX-SET-COVER is a $O(\lg n)$-approximation algorithm $\square$

Set Cover Summary

- set cover is NP-complete, exact algorithm takes exponential time
- fast $O(\lg n)$-approximate algorithm
- showed $\Theta(n^2)$ time
- $\Theta(n)$ time is possible

Big Idea: Linear Programming Relaxation

Recall:

- linear programming with real-valued variables is fast (polynomial time)
- integer linear programming (MIP) is *NP*-complete and slow (exponential time)

Idea:

- formulate our problem as a MIP
- "cheat" and solve it as a LP
- round off each solution variable to the nearest integer

Big Idea: Linear Programming Relaxation

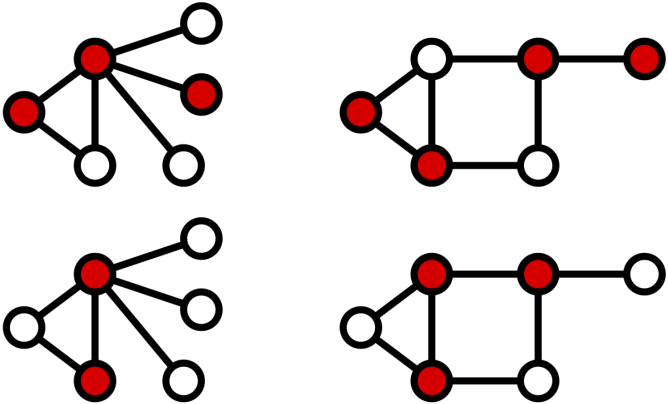
- **LP relaxation:** MIP formulation with integrality constraints removed
- not correct in general
- **but,** sometimes we can prove an approximate performance ratio
- next: algorithm that uses LP relaxation to solve vertex cover

Review: Vertex Cover

vertex cover problem
input: undirected graph $G = (V, E)$
output: set of vertices $C \subseteq V$, of minimal size $|C|$, such that every edge in E is incident on at least one vertex in C

- *NP*-complete
- previous deck: greedy algorithm, 2-approximate, $\Theta(m + n)$ time

Vertex Cover Example



Images credit: Wikipedia user Miym, CC BY-SA 3.0,

https://commons.wikimedia.org/wiki/File:Vertex-cover.svg,

https://commons.wikimedia.org/wiki/File:Minimum-vertex-cover.svg

Review: Formulating Vertex Cover

Variables: for each $v \in V$, create an integer variable x_v such that

$$x_v = 1 \Leftrightarrow v \in C$$

Objective: minimize

$$\sum_{v \in V} x_v$$

Constraints:

$$\begin{array}{lll} 0 \leq x_v \leq 1 & \forall v \in V & \text{(0 or 1 indicator)} \\ x_u + x_v \geq 1 & \forall (u, v) \in E & \text{(each edge is covered)} \end{array}$$

Review: Vertex Cover Outcomes

- **Infeasible:**
  - never happens
  - $\exists$ a solution: setting all $x_v = 1$ satisfies all constraints
- **Unbounded:**
  - never happens
  - objective is bounded: the objective function is to minimize

    $$\sum_{v \in V} x_v;$$

    since every $x_v \geq 0$, the minimum objective value is zero, which is finite, so the program is never unbounded
- **Solution:** Construct C as

$$C = \{v \mid v \in V \text{ and } x_v = 1\}$$

Vertex Cover LP Relaxation

Variables: for each $v \in V$, create a real-valued variable x_v such that

$$x_v = 1 \Leftrightarrow v \in C$$

Objective: minimize

$$\sum_{v \in V} x_v$$

Constraints:

$0 \leq x_v \leq 1$	$\forall v \in V$	(fuzzy 0 or 1 indicator)
$x_u + x_v \geq 1$	$\forall (u, v) \in E$	(each edge is covered)

LP Relaxation Vertex Cover Algorithm

function APX-VC-RELAX($G = (V, E)$)
    $C = \varnothing$
    LP = the linear program from the previous slide
    $\bar{x} = SOLVE - LP(LP)$ ▷ assume LP has solution
    for each vertex $v \in V$ **do**
        if $x_v \geq \frac{1}{2}$ **then** ▷ using $x_v \in \bar{x}$
            $C = C \cup \{v\}$
        end if
    end for
    return C
end function

Correctness

- LP is never unbounded or infeasible
- must prove that C is a valid vertex cover
- need, for each edge $\{u, v\} \in E$, that $u \in C$ or $v \in C$ (or both)
- LP relaxation has constraints
  $x_u + x_v \geq 1 \quad \forall (u, v) \in E \quad$ (each edge is covered)
- solution $\bar{x}$ satisfies all constraints, so

  $$x_u + x_v \geq 1$$

  is true and

  $$x_u \geq \frac{1}{2} \text{ or } x_v \geq \frac{1}{2} \text{ (or both)}$$
- so the **for** loop adds at least one of u, v to C

Efficiency Analysis

- create LP : $\Theta(n + m)$
- solve LP : polynomial
- post-processing **for** loop: $\Theta(n)$
- total time

$$\Theta(n + m) + \Theta(\text{solve LP}) + \Theta(n) = \Theta(\text{solve LP})$$

- polynomial time

Approximation Ratio

Theorem: APX-VC-RELAX is a 2-approximation algorithm

Proof sketch:

- let $C^\star$ be an optimal vertex cover for G
- need to prove $|C| \leq 2|C^\star|$
- use a "common ground" comparison between $|C|$ and $|C^\star|$
- let

$$\begin{aligned} z^\star &= \text{objective function value of LP} \\ &= \sum_{v \in V} x_v \text{ using each } x_v \in \bar{x} \end{aligned}$$

- we use $z^\star$ to relate $|C|$ to $|C^\star|$

Relating $z^\star$ to $|C^\star|$

- $z^\star$ is objective value $f(C)$ for our relaxed LP
- $C^\star$ is solution to MIP, with more constraints (integer x_v)
- so

$$\begin{aligned} z^\star &\le f(C^\star) \\ &= |C^\star| \end{aligned}$$

Relating $z^\star$ to $|C|$

- now relate $z^\star$ to $|C|$:

$$
\begin{aligned}
z^\star &= \sum_{v \in V} x_v \\
&\geq \sum_{v \in V, x_v \geq 1/2} x_v \\
&\geq \sum_{v \in V, x_v \geq 1/2} \left(\frac{1}{2}\right) \\
&= \sum_{v \in C} \frac{1}{2} \\
&= \frac{1}{2}|C|
\end{aligned}
$$

Completing the Proof of Approximation Ratio

Combine

$$z^\star \leq |C^\star|$$

with

$$z^\star \geq \frac{1}{2}|C|$$

to obtain

$$\frac{1}{2}|C| \leq z^\star \leq |C^\star|$$

or

$$|C| \leq 2 \cdot |C^\star|.$$

QED.

Vertex Cover LP Relaxation Summary

- LP relaxation approach:
  - formulate vertex cover as MIP
  - remove integer constraints, solve as LP
  - round each solution variable to nearest integer
- same polynomial runtime as linear programming
- 2-approximation
- compared to greedy algorithm in previous slides, this algo. is
  - simpler
  - slower
- generalizes to **weighted** case (see textbook section 35.5)

Bin Packing: Intuition

- have a collection of **items**
- want to **pack** them tightly into containers
- **puzzle**: which items go together in each container?

Bin Packing: Formal Definition

bin packing problem
input: a multiset $U = \{u \in \mathbb{Q}, 0 < u \leq 1\}$ of item sizes
output: a partition $B_1, B_2, \ldots, B_k$ of U into k multisets, such that the sum of each B_i is at most 1

- bin capacity is 1
- each size $u \in U$ is a fraction between 0 and 1
- ex. $U = \{\frac{2}{3}, \frac{1}{2}, \frac{1}{9}, \frac{1}{2}, \ldots\}$
- k = number of bins used

Example Applications

- Given a sink full of dirty dishes, how to load the dishwasher to clean all the dishes in the fewest loads?
- Given an Amazon order for items of varying weights, how to pack the items into the fewest shipping boxes?
- Given a set of virtual machines (VMs) of varying memory sizes, how to host them on the fewest physical servers?

Generalizations of Bin Packing

Problem statement can be generalized to be more realistic:

- items are 2D shapes instead of numbers (physical object shipping)
- items can partially overlap (VM shared memory can overlap)
- one bin, different values: *knapsack problem*
- minimize bins, and also waste: *cutting stock problem*

Bin Packing Hardness

- bin packing is *NP*-complete
- generalizations (ex. 2D shapes) are even harder
- baseline algorithm:
  - loop through each possible number of bins $k = 1, \ldots, n$
  - for each item $u \in U$: try placing u in all k bins, then recursively place the remaining items
- $\Theta(n \cdot n!)$ time, extremely slow
- (exponential time is possible too)

Greedy Algorithm Idea

- keep a list of bins
- for each item u : find any bin with enough room for u, and put it there
- if no bin has enough room: start a new bin holding just u
- **"first fit"** algorithm

First-Fit Algorithm

```
function FIRST-FIT-BIN-PACK(U)
    B = ∅                                        ▷ the bins
    for u ∈ U do
        packed = false
        for B_i ∈ B do
            if (u + ∑ B_i) ≤ 1 then              ▷ does u fit in B_i?
                B_i = B_i ∪ {u}
                packed = true
                break loop
            end if
        end for
        if packed == false then
            B_k = {u}                            ▷ u in its own new bin
            B = B ∪ {B_k}
        end if
    end for
    return B
end function
```

Efficiency Analysis

- outer **for** loop: $\Theta(n)$ iterations
- $|B| \leq n$, so
- inner **for** loop: $\Theta(n)$ iterations
- $\sum B_i : \Theta(n)$ time
- other statements are $\Theta(1)$ time
- $\therefore\ \Theta(n^3)$ total time
- can speed up to $\Theta(n \log n)$ by caching totals and storing bins in a BST

Approximation Ratio

Theorem: FIRST-FIT-BIN-PACK is a 2-approximation algorithm.
Proof sketch:

- Recall $k = |B|$
- Let $B^\star$ be an optimal multiset of bins, and $k^\star = |B^\star|$
- (again) use "common ground" comparison between k and $k^\star$
- Let $t = \sum U$ be the sum of all items; use t as common ground
- Best possible packing fills every single bin with no leftover space, so

$$k^\star \geq \frac{\text{total size of items}}{\text{size of each bin}} = \frac{(t)}{(1)} = t$$

There Is At Most One Light Bin

- Call a bin B_i **"light"** when $\sum B_i \leq \frac{1}{2}$, otherwise **"heavy"**
- Invariant: there is at most one light bin
- Induction on number of items in bins
- Base case: zero items $\implies$ zero bins $\implies$ no light bin
- Inductive case: given new item u, four cases:

	$u \leq \frac{1}{2}$	$u > \frac{1}{2}$
no light bin	u starts only light bin	u starts new heavy bin
$\exists$ light bin B_i	u joins B_i	u either joins B_i making it heavy; or starts a new heavy bin

Approximation Ratio (continued)

- worst case is one light bin and $k - 1$ barely-heavy bins
- k is at most

$$k \leq \frac{\text{total size of items}}{\text{size of each bin}} + 1 < \frac{(t)}{(1/2)} + 1 = 2t + 1$$

- we showed $k^\star \geq t$, so

$$k < 2t + 1 \leq 2(k^\star) + 1$$

- for large n,

$$k \propto 2k^\star$$

- QED

Tight Approximation Ratio

- (Johnson, Demers, Ullman, Garey, Graham 1974)
- FIRST-FIT-BIN-PACK is a 1.7-approximation algorithm
  - More elaborate case analysis
- There is a U for which FIRST-FIT-BIN-PACK achieves only a 1.7 performance ratio
  - $U = \{\frac{6}{101} \times 7, \frac{10}{101} \times 7, \frac{16}{101} \times 3, \frac{34}{101} \times 10, \frac{51}{101} \times 10\}$
  - $k^\star = 10$
  - $k = 17$
- $\therefore$ the $1.7k^\star$ bound is tight

Bin Packing Summary

- bin packing is *NP*-complete, exact algorithm takes exponential time
- $\Theta(n \log n)$ time, 1.7-approximation algorithm
- we only proved $\Theta(n^3)$ time, 2-approximation