

CSCI 550: ADVANCED ALGORITHMS

3. APPROXIMATION ALGORITHMS

- *load balancing*
- *center selection*
- *weighted vertex cover: pricing method*
- *weighted vertex cover: LP rounding*
- *knapsack problem*

Coping with NP-completeness

- Q. Suppose I need to solve an **NP**-complete problem. What should I do?
- A. Theory says you're unlikely to find a poly-time algorithm.

Must sacrifice one of three desired features.

- Solve problem to **optimality**.
- Solve problem in polynomial time.
- Solve specific instances of the problem.

This lecture's focus. Solving problems as close to optimally as possible.

ρ -approximation algorithm.

- Runs in polynomial time.
- Solves arbitrary instances of the problem.
- Finds a solution that is within some ratio ρ of the optimum.

Challenge. Need to prove that a solution's value is close to optimum, without even knowing what the optimum value is.

2

CSCI 550: ADVANCED ALGORITHMS

3. APPROXIMATION ALGORITHMS

- *load balancing*
- *center selection*
- *weighted vertex cover: pricing method*
- *weighted vertex cover: LP rounding*
- *knapsack problem*

Load balancing

Input. m identical machines; $n \geq m$ jobs, each job j has a processing time t_j .

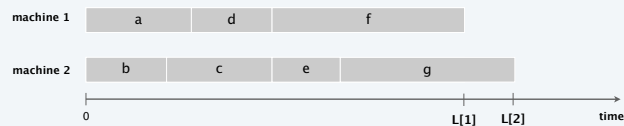
- Job j must run contiguously on one machine.
- Each machine can process at most one job at a time.

Def. Let $S[i]$ be the subset of jobs assigned to machine i .

The **load** of a machine i is $L[i] = \sum_{j \in S[i]} t_j$.

Def. The **makespan** is the maximum load on any machine $L = \max_i L[i]$.

Load balancing. Assign each job to a machine to minimize makespan.



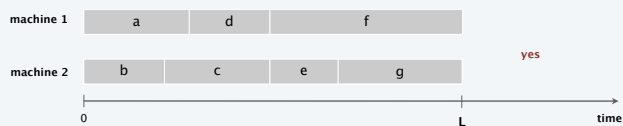
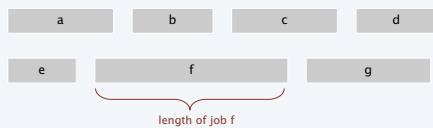
4

Load balancing on 2 machines is NP-hard

Claim. Load balancing is hard even if we have just $m = 2$ machines.

Pf. PARTITION $\leq_P$ LOAD-BALANCE.

NP-complete



5

Load balancing: list scheduling



List-scheduling algorithm.

- Consider n jobs in some fixed order.
- Assign each job j to the machine i whose load is smallest so far.

LIST-SCHEDULING ($m, n, t_1, t_2, \dots, t_n$)

FOR $i = 1$ TO m

$L[i] \leftarrow 0$. $\leftarrow$ load on machine i

$S[i] \leftarrow \emptyset$. $\leftarrow$ jobs assigned to machine i

FOR $j = 1$ TO n

$i \leftarrow \text{argmin}_k L[k]$. $\leftarrow$ machine i has smallest load

$S[i] \leftarrow S[i] \cup \{j\}$. $\leftarrow$ assign job j to machine i

$L[i] \leftarrow L[i] + t_j$. $\leftarrow$ update load of machine i

RETURN $S[1], S[2], \dots, S[m]$.

Implementation. $O(n \log m)$ using a priority queue for loads $L[k]$.

6

Load balancing: list scheduling analysis

Theorem. [Graham 1966] The greedy algorithm is a 2-approximation.

- First worst-case analysis of an approximation algorithm.
- Need to compare the resulting solution to the optimal makespan L^* .

Lemma 1. For all k , the optimal makespan is $L^* \geq t_k$.

Pf. Some machine must process the most time-consuming job. ▀

Lemma 2. The optimal makespan is $L^* \geq \frac{1}{m} \sum_k t_k$.

Pf.

- The total processing time is $\sum_k t_k$.
- One of m machines must do at least $1/m$ fraction of total work. ▀

7

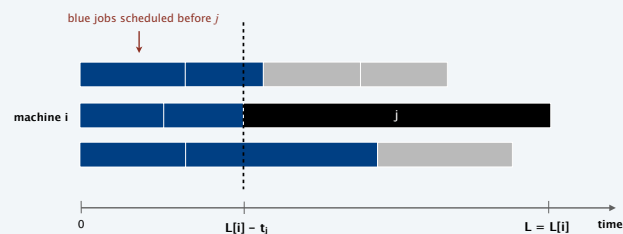
Load balancing: list scheduling analysis

Theorem. [Graham 1966] The greedy algorithm is a 2-approximation.

Pf. Consider the load $L[i]$ of the bottleneck machine i . ← machine that ends up with highest load

- Let j be the last job scheduled on machine i .
- When job j was assigned to machine i , i had the smallest load.

Its load before the assignment was $L[i] - t_j$; hence, $L[i] - t_j \leq L[k]$ for all $1 \leq k \leq m$.



8

Load balancing: list scheduling analysis

Theorem. [Graham 1966] The greedy algorithm is a 2-approximation.

Pf. Consider the load $L[i]$ of the bottleneck machine i . ← machine that ends up with highest load

- Let j be the last job scheduled on machine i .
- When job j was assigned to machine i , i had the smallest load.

Its load before the assignment was $L[i] - t_j$; hence, $L[i] - t_j \leq L[k]$ for all $1 \leq k \leq m$.

- Sum inequalities over all k and divide by m :

$$\begin{aligned} L[i] - t_j &\leq \frac{1}{m} \sum_k L[k] \\ &= \frac{1}{m} \sum_k t_k \end{aligned}$$

Lemma 2 → $\leq L^*$.

- Now, $L = L[i] = (L[i] - t_j) + t_j \leq 2L^*$ ▀

↑ ↑
 $\leq L^*$ $\leq L^*$
 above inequality Lemma 1

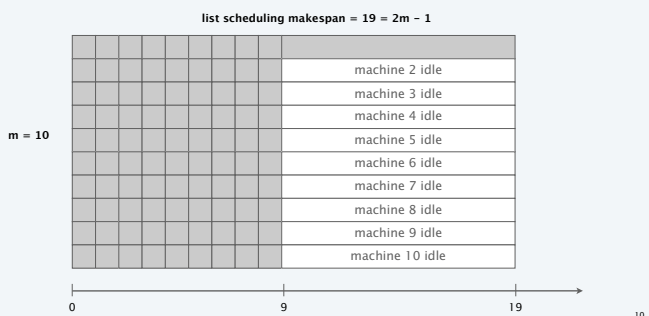
9

Load balancing: list scheduling analysis

Q. Is our analysis tight?

A. Essentially, yes.

Ex: m machines, first $m(m-1)$ jobs have length 1, last job has length m .

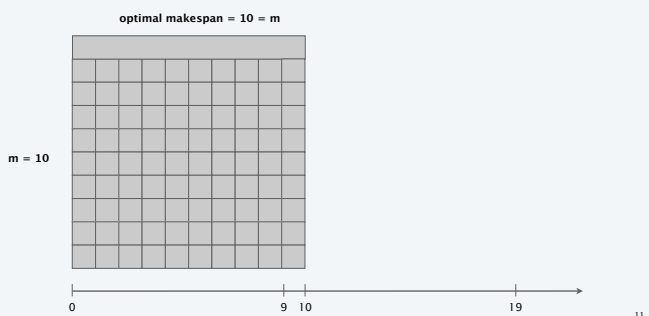


Load balancing: list scheduling analysis

Q. Is our analysis tight?

A. Essentially, yes.

Ex: m machines, first $m(m-1)$ jobs have length 1, last job has length m .



Load balancing: LPT rule

Longest processing time (LPT). Sort n jobs in decreasing order of processing times, then run the list scheduling algorithm.

LPT-LIST-SCHEDULING ($m, n, t_1, t_2, \dots, t_n$)

SORT jobs and renumber so that $t_1 \geq t_2 \geq \dots \geq t_n$.

FOR $i = 1$ TO m

$L[i] \leftarrow 0$. ← load on machine i

$S[i] \leftarrow \emptyset$. ← jobs assigned to machine i

FOR $j = 1$ TO n

$i \leftarrow \operatorname{argmin}_k L[k]$. ← machine i has smallest load

$S[i] \leftarrow S[i] \cup \{j\}$. ← assign job j to machine i

$L[i] \leftarrow L[i] + t_j$. ← update load of machine i

RETURN $S[1], S[2], \dots, S[m]$.

Load balancing: LPT rule

Observation. If bottleneck machine i has only 1 job, then this is optimal.

Pf. Any solution must schedule that job. ■

Lemma 3. If there are more than m jobs, then $L^* \geq 2t_{m+1}$.

Pf.

- Consider the processing times of the first $m+1$ jobs $t_1 \geq t_2 \geq \dots \geq t_{m+1}$.
- Each job takes at least t_{m+1} time.
- There are $m+1$ jobs and m machines, so by the pigeonhole principle, at least one machine gets two jobs. ■

Theorem. The LPT rule is a $3/2$ -approximation algorithm.

Pf. [similar to proof for list scheduling]

- Consider the load $L[i]$ of bottleneck machine i . ← assuming machine i has at least 2 jobs, we have $j \geq m+1$
- Let j be the last job scheduled on machine i .

$$L = L[i] = \underbrace{(L[i] - t_j)}_{\text{as before} \rightarrow \leq L^*} + \underbrace{t_j}_{\leq \frac{1}{2} L^*} \leq \frac{3}{2} L^* \quad \leftarrow \text{Lemma 3 (since } t_{m+1} \geq t_j \text{)}$$

13

Load balancing: LPT rule

Q. Is our $3/2$ analysis tight?

A. No.

Theorem. [Graham 1969] LPT rule is a $4/3$ -approximation.

Pf. More sophisticated analysis of same algorithm.

Q. Is Graham's $4/3$ analysis tight?

A. Essentially, yes.

Ex.

- m machines; $n = 2m + 1$ jobs.
- 2 jobs of length $m, m+1, \dots, 2m-1$ and one more job of length m .
- Then, $L / L^* = (4m - 1) / (3m)$.

14

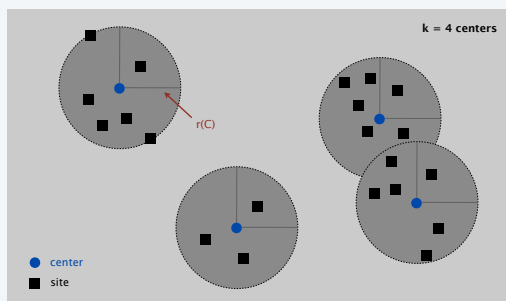
CSCI 550: ADVANCED ALGORITHMS 3. APPROXIMATION ALGORITHMS

- ▶ load balancing
- ▶ center selection
- ▶ weighted vertex cover: pricing method
- ▶ weighted vertex cover: LP rounding
- ▶ knapsack problem

Center selection problem

Input. Set of n sites $s_1, \dots, s_n$ and an integer $k > 0$.

Center selection problem. Select a set of k centers C so that the maximum distance $r(C)$ from a site to the nearest center is minimized.



16

Center selection problem

Input. Set of n sites $s_1, \dots, s_n$ and an integer $k > 0$.

Center selection problem. Select a set of k centers C so that the maximum distance $r(C)$ from a site to the nearest center is minimized.

Notation.

- $\text{dist}(x, y)$ = distance between sites x and y .
- $\text{dist}(s_i, C) = \min_{c \in C} \text{dist}(s_i, c)$ = distance from s_i to closest center.
- $r(C) = \max_i \text{dist}(s_i, C)$ = smallest covering radius.

Goal. Find set of centers C that minimizes $r(C)$, subject to $|C| = k$.

Distance function properties.

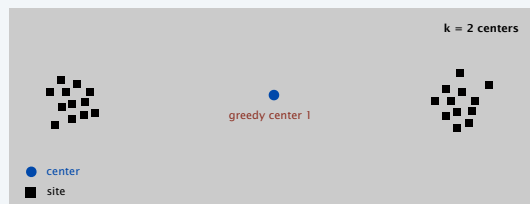
- $\text{dist}(x, x) = 0$ [identity]
- $\text{dist}(x, y) = \text{dist}(y, x)$ [symmetry]
- $\text{dist}(x, y) \leq \text{dist}(x, z) + \text{dist}(z, y)$ [triangle inequality]

17

Center selection: a false start

Greedy algorithm. Put the first center at the best possible location for a single center, and then keep adding centers so as to reduce the covering radius each time by as much as possible.

Remark: This can be arbitrarily bad!



18

Center selection: greedy algorithm

Better greedy algorithm. Repeatedly choose the next center to be the site **farthest** from any existing centers.

GREEDY-CENTER-SELECTION ($k, n, s_1, s_2, \dots, s_n$)

$C \leftarrow \emptyset$.

REPEAT k times

 Select a site s_i with maximum distance $\text{dist}(s_i, C)$.

$C \leftarrow C \cup s_i$.

RETURN C .

↑
site farthest
from any center

Property. Upon termination, all centers in C are pairwise at least $r(C)$ apart.

Pf. By construction of algorithm.

19

Center selection: analysis of greedy algorithm

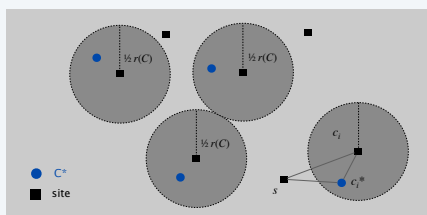
Lemma. Let C^* be an optimal set of centers. Then $r(C) \leq 2r(C^*)$.

Pf. [by contradiction] Assume $r(C^*) < \frac{1}{2} r(C)$.

- For each site $c_i \in C$, consider ball of radius $\frac{1}{2} r(C)$ around it.
- Exactly one c_i^* in each ball; let c_i be the site paired with c_i^* .
- Consider any site s and its closest center $c_i^* \in C^*$.
- $\text{dist}(s, C) \leq \text{dist}(s, c_i) \leq \text{dist}(s, c_i^*) + \text{dist}(c_i^*, c_i) \leq 2r(C^*)$.

↑
Δ-inequality
≤ $r(C^*)$ since c_i^* is closest center

- Thus, $r(C) \leq 2r(C^*)$. ▀



20

Center selection: analysis of greedy algorithm

Lemma. Let C^* be an optimal set of centers. Then $r(C) \leq 2r(C^*)$.

Theorem. The greedy algorithm is a 2-approximation for the center selection problem.

Remark. The greedy algorithm always places centers at sites, but is still within a factor of 2 of the best solution that is allowed to place centers anywhere.

↑
e.g., points in the plane

Question. Is there hope of a 3/2-approximation? 4/3?

21

Dominating set reduces to center selection

Theorem. Unless $P = NP$, there is no p -approximation for the center selection problem for any $p < 2$.

Pf. We can use a $(2 - \epsilon)$ -approximation algorithm for CENTER-SELECTION to solve DOMINATING-SET in poly-time.

- Let $G = (V, E)$, k be an instance of DOMINATING-SET.
- Construct an instance G' of CENTER-SELECTION with sites V and distances:
 - $\text{dist}(u, v) = 1$ if $(u, v) \in E$
 - $\text{dist}(u, v) = 2$ if $(u, v) \notin E$
- Note that G' satisfies the triangle inequality.
- G has a dominating set of size k iff there exist k centers C^* with $r(C^*) = 1$.
- Thus, if G has a dominating set of size k , a $(2 - \epsilon)$ -approximation algorithm for CENTER-SELECTION would find a solution C^* with $r(C^*) = 1$ since it cannot use any edge of distance 2. ■

22

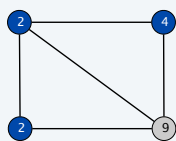
CSCI 550: ADVANCED ALGORITHMS 3. APPROXIMATION ALGORITHMS

- ▶ load balancing
- ▶ center selection
- ▶ weighted vertex cover: pricing method
- ▶ weighted vertex cover: LP rounding
- ▶ knapsack problem

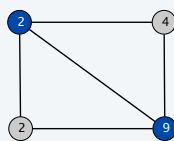
Weighted vertex cover

Vertex cover. Given a graph $G = (V, E)$, a vertex cover is a set $S \subseteq V$ such that each edge in E has at least one end in S .

Weighted vertex cover. Given a graph G with vertex weights, find a vertex cover of minimum weight.



weight = $2 + 2 + 4 = 8$



weight = $2 + 9 = 11$

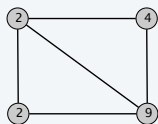
24

Pricing method

Pricing method. Each edge must be covered by some vertex.
Edge $e = (i, j)$ pays a price $p_e \geq 0$ to use both vertices i and j .

Fairness. Edges incident to vertex i should pay $\leq w_i$ in total.

for each vertex i : $\sum_{e=(i,j)} p_e \leq w_i$



Fairness lemma. For any vertex cover S and any fair prices p_e , $\sum_e p_e \leq w(S)$.

Pf. $\sum_{e \in E} p_e \leq \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in S} w_i = w(S)$. ■

each edge e covered by at least one node in S sum fairness inequalities for each node in S

25

Pricing method

Pricing method. Set prices and find vertex cover simultaneously.

WEIGHTED-VERTEX-COVER (G, w)

$S \leftarrow \emptyset$.

FOREACH $e \in E$

$p_e \leftarrow 0$.

$$\sum_{e=(i,j)} p_e = w_i$$

WHILE (there exists an edge (i, j) such that neither i nor j is tight)

Select such an edge $e = (i, j)$.

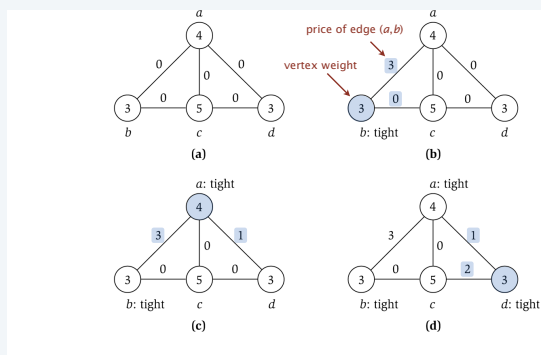
Increase p_e as much as possible until i or j is tight.

$S \leftarrow$ set of all tight nodes.

RETURN S .

26

Pricing method: example



27

Pricing method: analysis

Theorem. Pricing method is a 2-approximation for WEIGHTED-VERTEX-COVER.
pf.

- The algorithm terminates, since at least one new node becomes tight after each iteration of the while loop.
- Let S = set of all tight nodes upon termination of algorithm.
 S is a vertex cover: if some edge (i, j) is uncovered, then neither i nor j is tight. But then the while loop would not terminate.
- Let S^* be an optimal vertex cover. We show that $w(S) \leq 2 w(S^*)$.

$$w(S) = \sum_{i \in S} w_i = \sum_{i \in S} \sum_{e=(i,j)} p_e \leq \sum_{i \in V} \sum_{e=(i,j)} p_e = 2 \sum_{e \in E} p_e \leq 2 w(S^*). \quad \blacksquare$$

all nodes in S are tight $S \subseteq V$, prices ≥ 0 each edge counted twice fairness lemma

28

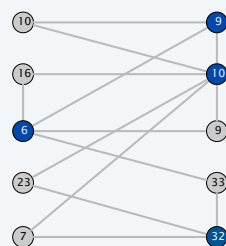
CSCI 550: ADVANCED ALGORITHMS

3. APPROXIMATION ALGORITHMS

- load balancing
- center selection
- weighted vertex cover: pricing method
- weighted vertex cover: LP rounding
- knapsack problem

Weighted vertex cover

Given a graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a min-weight subset of vertices $S \subseteq V$ such that every edge is incident to at least one vertex in S .



30

Weighted vertex cover: ILP formulation

Given a graph $G = (V, E)$ with vertex weights $w_i \geq 0$, find a min-weight subset of vertices $S \subseteq V$ such that every edge is incident to at least one vertex in S .

Integer linear programming formulation.

- Model inclusion of each vertex i using a 0/1 variable x_i .

$$x_i = \begin{cases} 0 & \text{if vertex } i \text{ is not in vertex cover} \\ 1 & \text{if vertex } i \text{ is in vertex cover} \end{cases}$$

Vertex covers are in 1-1 correspondence with 0/1 assignments:

$$S = \{i \in V : x_i = 1\}.$$

- Objective function: minimize $\sum_i w_i x_i$.
- For every edge (i, j) , we must take either vertex i or j (or both):
 $x_i + x_j \geq 1$.

31

Weighted vertex cover: ILP formulation

Integer linear programming formulation.

$$\begin{aligned} (ILP) \quad & \min \quad \sum_{i \in V} w_i x_i \\ \text{s.t.} \quad & x_i + x_j \geq 1 \quad (i, j) \in E \\ & x_i \in \{0, 1\} \quad i \in V \end{aligned}$$

Observation. If x^* is an optimal solution to ILP , then $S = \{i \in V : x_i^* = 1\}$ is a min-weight vertex cover.

32

Integer linear programming

Given integers a_{ij} , b_i , and c_j , find integers x_j that satisfy:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geq b \\ & x \geq 0 \\ & x \text{ integral} \end{aligned} \quad \begin{aligned} \min \quad & \sum_{j=1}^n c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^n a_{ij} x_j \geq b_i \quad 1 \leq i \leq m \\ & x_j \geq 0 \quad 1 \leq j \leq n \\ & x_j \text{ integral} \quad 1 \leq j \leq n \end{aligned}$$

Observation. Formulating VERTEX-COVER as an integer linear program proves that INTEGER-PROGRAMMING is an NP-hard optimization problem.

33

Integer linear programming

Given integers a_{ij} , b_i , and c_j , find **real numbers** x_j that satisfy:

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \geq b \\ & x \geq 0 \end{array} \qquad \begin{array}{ll} \min & \sum_{j=1}^n c_j x_j \\ \text{s.t.} & \sum_{j=1}^n a_{ij} x_j \geq b_i \quad 1 \leq i \leq m \\ & x_j \geq 0 \quad 1 \leq j \leq n \end{array}$$

Linear. No x^2 , xy , $\arccos(x)$, $x(1-x)$, etc.

Simplex algorithm. [Dantzig 1947] Can solve LP in practice.

Ellipsoid algorithm. [Khachiyan 1979] Can solve LP in poly-time.

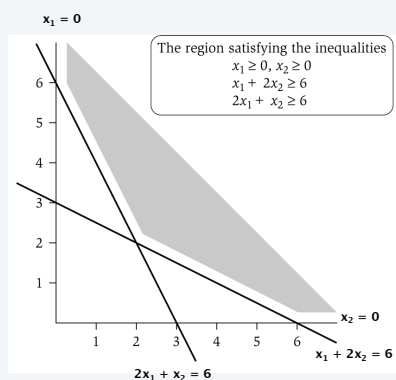
Interior point algorithms. [Karmarkar 1984, Renegar 1988, ...]

Can solve LP both in practice and in poly-time.

34

LP feasible region

LP geometry in 2D.



35

Weighted vertex cover: LP relaxation

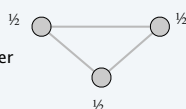
Linear programming relaxation.

$$\begin{array}{ll} (LP) \min & \sum_{i \in V} w_i x_i \\ \text{s.t.} & x_i + x_j \geq 1 \quad (i, j) \in E \\ & x_i \geq 0 \quad i \in V \end{array}$$

Observation. Optimal value of $LP \leq$ optimal value of ILP .

Pf. LP has fewer constraints.

Note. LP solution x^* may not correspond to a vertex cover (even if all weights are 1).



Q. How can solving LP help us find a low-weight vertex cover?

A. Solve LP and **round** fractional values in x^* .

36

Weighted vertex cover: LP rounding algorithm

Lemma. If x^* is an optimal solution to LP , then $S = \{i \in V : x_i^* \geq \frac{1}{2}\}$ is a vertex cover whose weight is at most twice the min possible weight.

Pf. [S is a vertex cover]

- Consider an edge $(i, j) \in E$.
- Since $x_i^* + x_j^* \geq 1$, either $x_i^* \geq \frac{1}{2}$ or $x_j^* \geq \frac{1}{2}$ (or both) $\Rightarrow (i, j)$ covered.

Pf. [S has desired weight]

- Let S^* be an optimal vertex cover. Then

$$\sum_{i \in S^*} w_i \geq \sum_{i \in S} w_i x_i^* \geq \frac{1}{2} \sum_{i \in S} w_i$$

$\uparrow$ LP is a relaxation $\uparrow$ $x_i^* \geq \frac{1}{2}$

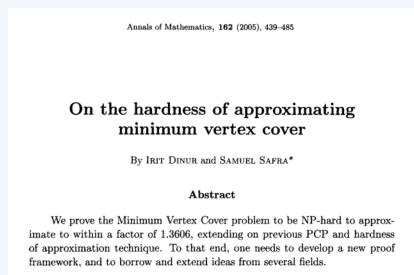
Theorem. The rounding algorithm is a 2-approximation algorithm.

Pf. Above lemma + the fact that LP can be solved in poly-time.

37

Weighted vertex cover inapproximability

Theorem. [Dinur-Safra 2004] If $P \neq NP$, then there exists no ρ -approximation algorithm for WEIGHTED-VERTEX-COVER for any $\rho < 1.3606$ (even if all weights are 1).

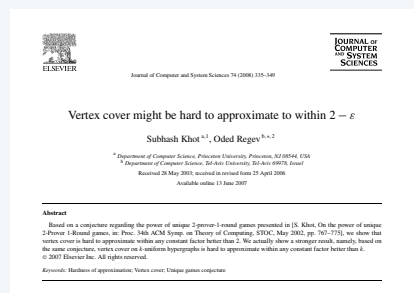


Open research problem. Close the gap.

38

Weighted vertex cover inapproximability

Theorem. [Khot-Regev 2008] If the Unique Games Conjecture is true, then there exists no $2 - \epsilon$ approximation algorithm for WEIGHTED-VERTEX-COVER for any $\epsilon > 0$.



Open research problem. Prove the Unique Games Conjecture.

39

CSCI 550: ADVANCED ALGORITHMS

3. APPROXIMATION ALGORITHMS

- *load balancing*
- *center selection*
- *weighted vertex cover: pricing method*
- *weighted vertex cover: LP rounding*
- *knapsack problem*

Polynomial-time approximation schemes

PTAS. $(1 + \epsilon)$ -approximation algorithm for any constant $\epsilon > 0$.

- Load balancing. [Hochbaum–Shmoys 1987]
- Euclidean TSP. [Arora, Mitchell 1996]
- Many more!

Consequence. PTAS produces an arbitrarily high quality solution, but at the expense of trading off accuracy for time.

41

Knapsack problem

Knapsack problem.

- Given n objects and a knapsack.
- Item i has value $v_i > 0$ and weighs $w_i > 0$. $\leftarrow$ assume $w_i \leq W$ for each i
- Knapsack has weight limit W .
- Goal: fill knapsack so as to maximize total value.

Ex: $\{3, 4\}$ has value 40.

item	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

original instance ($W = 11$)

42

Knapsack problem is NP-complete

KNAPSACK. Given a set X , weights $w_i \geq 0$, values $v_i \geq 0$, a weight limit W , and a target value V , is there a subset $S \subseteq X$ such that:

$$\begin{aligned}\sum_{i \in S} w_i &\leq W \\ \sum_{i \in S} v_i &\geq V\end{aligned}$$

SUBSET-SUM. Given a set X , values $u_i \geq 0$, and an integer U , is there a subset $S \subseteq X$ whose elements sum to exactly U ?

Theorem. SUBSET-SUM $\leq_P$ KNAPSACK.

Pf. Given an instance $(u_1, \dots, u_n, U)$ of SUBSET-SUM, create the following KNAPSACK instance:

$$\begin{aligned}v_i &= w_i = u_i & \sum_{i \in S} u_i &\leq U \\ V = W = U & & \sum_{i \in S} u_i &\geq U\end{aligned}$$

43

Knapsack problem: dynamic programming

Def. $OPT(i, w)$ = **max value** subset of items $1, \dots, i$ with **weight** limit w .

Case 1. OPT does not select item i .

- OPT selects best of $1, \dots, i-1$ using up to weight limit w .

Case 2. OPT selects item i .

- New weight limit = $w - w_i$.
- OPT selects best of $1, \dots, i-1$ using up to weight limit $w - w_i$.

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{OPT(i-1, w), v_i + OPT(i-1, w - w_i)\} & \text{otherwise} \end{cases}$$

Theorem. The algorithm computes the optimal value in $O(nW)$ time.

- Not polynomial in input size.
- Polynomial time if weights are small integers.

44

Knapsack problem: dynamic programming revisited

Def. $OPT(i, v)$ = **min weight** of a knapsack for which we can obtain a solution of **value** $\geq v$ using a subset of items $1, \dots, i$.

Note. Optimal value is the largest value v such that $OPT(n, v) \leq W$.

Case 1. OPT does not select item i .

- OPT selects best of $1, \dots, i-1$ that achieves value $\geq v$.

Case 2. OPT selects item i .

- Consumes weight w_i , need to achieve value $\geq v - v_i$.
- OPT selects best of $1, \dots, i-1$ that achieves value $\geq v - v_i$.

$$OPT(i, v) = \begin{cases} 0 & \text{if } v \leq 0 \\ \infty & \text{if } i = 0 \text{ and } v > 0 \\ \min\{OPT(i-1, v), w_i + OPT(i-1, v - v_i)\} & \text{otherwise} \end{cases}$$

45

Knapsack problem: dynamic programming revisited

Theorem. The algorithm computes the optimal value in $O(n^2 v_{\max})$ time, where $v_{\max}$ is the maximum of any value.

Pf.

- The optimal value is $V^* \leq n v_{\max}$.
- There is one subproblem for each item and for each value $v \leq V^*$.
- We take $O(1)$ time per subproblem. ▀

Remarks.

- Still not polynomial in input size!
- Polynomial time if values are small integers.

46

Knapsack problem: polynomial-time approximation scheme

Intuition for approximation algorithm.

- Round all values up to lie within smaller range.
- Run second DP algorithm on rounded/scaled instance.
- Return optimal items from rounded instance.

item	value	weight
1	934221	1
2	5956342	2
3	17810013	5
4	21217800	6
5	27343199	7

original instance ($W = 11$)

item	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

rounded instance ($W = 11$)

47

Knapsack problem: polynomial-time approximation scheme

Round up all values:

- $0 < \epsilon \leq 1$ = precision parameter.
- $v_{\max}$ = largest value in original instance.
- θ = scaling factor = $\epsilon v_{\max} / 2n$. $\bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta$, $\hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil$

Observation. Optimal solutions to problem with $\bar{v}$ are equivalent to optimal solutions to problem with $\hat{v}$.

Intuition. $\bar{v}$ is close to v so the optimal solution using $\bar{v}$ is nearly optimal; $\hat{v}$ is small and integral so the DP algorithm is fast.

48

Knapsack problem: polynomial-time approximation scheme

Theorem. If S is the solution found by the rounding algorithm and S^* is any other feasible solution, then $(1 + \epsilon) \sum_{i \in S} v_i \geq \sum_{i \in S^*} v_i$.

Pf. Let S^* be any feasible solution satisfying the weight constraint.

$$\begin{aligned}
 \sum_{i \in S^*} v_i &\leq \sum_{i \in S^*} \bar{v}_i && \text{always round up} \\
 &\leq \sum_{i \in S} \bar{v}_i && \text{solve rounded instance optimally} \\
 &\leq \sum_{i \in S} (v_i + \theta) && \text{never round up by more than } \theta \\
 &\leq \sum_{i \in S} v_i + n\theta && |S| \leq n \\
 &= \sum_{i \in S} v_i + \frac{1}{2} \epsilon v_{\max} && \theta = \epsilon v_{\max} / 2n \\
 &\leq (1 + \epsilon) \sum_{i \in S} v_i && v_{\max} \leq 2 \sum_{i \in S} v_i
 \end{aligned}$$

subset containing only the item of largest value

choosing $S^* = \{ \max \}$

$$\begin{aligned}
 v_{\max} &\leq \sum_{i \in S} v_i + \frac{1}{2} \epsilon v_{\max} \\
 &\leq \sum_{i \in S} v_i + \frac{1}{2} v_{\max} \\
 \text{thus} \quad v_{\max} &\leq 2 \sum_{i \in S} v_i
 \end{aligned}$$

49

Knapsack problem: polynomial-time approximation scheme

Theorem. For any $\epsilon > 0$, the rounding algorithm computes a feasible solution whose value is within a $(1 + \epsilon)$ factor of the optimum in $O(n^3 / \epsilon)$ time.

Pf.

- We have already proved the accuracy bound.
- DP algorithm running time is $O(n^2 \hat{v}_{\max})$, where $\hat{v}_{\max} = \left\lceil \frac{v_{\max}}{\theta} \right\rceil = \left\lceil \frac{2n}{\epsilon} \right\rceil$

50