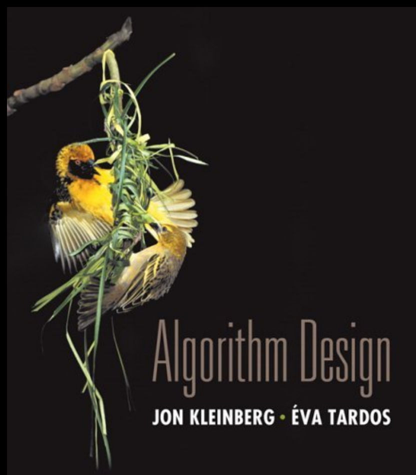


Chapter 11

Approximation Algorithms

Approximation Algorithms

Q. Suppose I need to solve an NP-hard 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 poly-time.
- Solve arbitrary instances of the problem.

ρ-approximation algorithm.
- Guaranteed to run in poly-time.
- Guaranteed to solve arbitrary instance of the problem
- Guaranteed to find solution within ratio ρ of true optimum.

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

2

11.1 Load Balancing

Load Balancing

Input. m identical machines; n jobs, job j has processing time t_j.
- Job j must run contiguously on one machine.
- A machine can process at most one job at a time.

Def. Let J(i) be the subset of jobs assigned to machine i. The load of machine i is $L_i = \Sigma_{j \in J(i)} t_j$.

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: List Scheduling

List-scheduling algorithm.

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

```
List-Scheduling(m, n, t1,t2,…,tn) {
   for i = 1 to m {
      Li ← 0          ← load on machine i
      J(i) ← ϕ        ← jobs assigned to machine i
   }

   for j = 1 to n {
      i = argmink Lk          ← machine i has smallest load
      J(i) ← J(i) ∪ {j}       ← assign job j to machine i
      Li ← Li + tj            ← update load of machine i
   }
   return J(1), …, J(m)
}
```

Note: in Textbook, Ti is used instead of Li

Implementation. O(n log m) using a priority queue.

5

Load Balancing: List Scheduling Analysis

Theorem. [Graham, 1966] Greedy algorithm is a 2-approximation.

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

Lemma 1. The optimal makespan $L^* \geq \max_j t_j$.
Pf. Some machine must process the most time-consuming job. ▪

Lemma 2. The optimal makespan $L^* \geq \frac{1}{m} \sum_j t_j$.
Pf.

- The total processing time is $\Sigma_j t_j$.
- One of m machines must do at least a 1/m fraction of total work. ▪

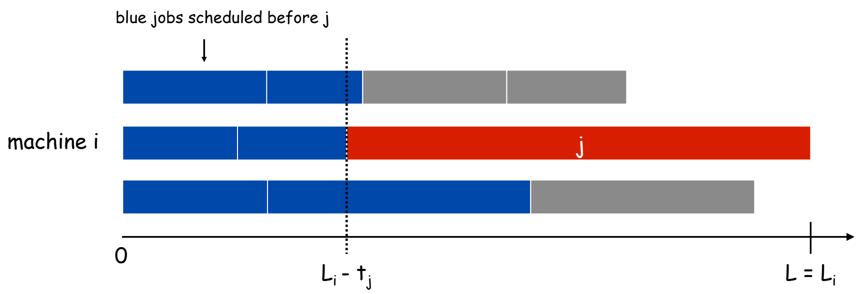
6

Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load L_i of bottleneck machine i.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L_i - t_j \Rightarrow L_i - t_j \le L_k$ for all $1 \le k \le m$.



Load Balancing: List Scheduling Analysis

Theorem. Greedy algorithm is a 2-approximation.

Pf. Consider load L_i of bottleneck machine i.

- Let j be last job scheduled on machine i.
- When job j assigned to machine i, i had smallest load. Its load before assignment is $L_i - t_j \Rightarrow L_i - t_j \le L_k$ for all $1 \le k \le m$.
- Sum inequalities over all k and divide by m:

$$\begin{aligned} L_i - t_j &\le \tfrac{1}{m} \sum_k L_k \\ &= \tfrac{1}{m} \sum_j t_j \\ &\le L^* \end{aligned}$$

Lemma 1 →

Sum of makespan L_k equals sum of all jobs' processing time t_j.

- Now $L_i = \underbrace{(L_i - t_j)}_{\le L^*} + \underbrace{t_j}_{\le L^*} \le 2L^*$. ▪

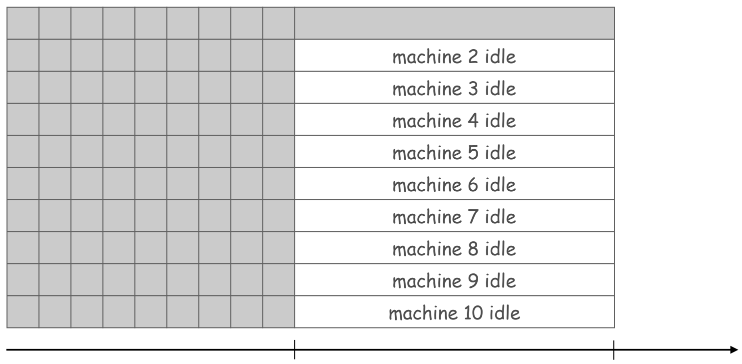
Lemma 2

Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m

m = 10



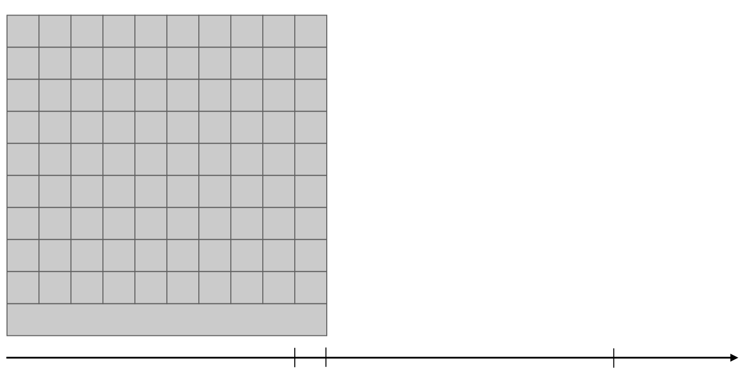
list scheduling makespan = 19

Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?
A. Essentially yes.

Ex: m machines, m(m-1) jobs length 1 jobs, one job of length m

m = 10



optimal makespan = 10

Load Balancing: LPT Rule

Longest processing time (LPT). Sort n jobs in **descending** order of processing time, and then run list scheduling algorithm.

```
LPT-List-Scheduling(m, n, t1,t2,…,tn) {
   Sort jobs so that t1 ≥ t2 ≥ … ≥ tn

   for i = 1 to m {
      Li ← 0           ← load on machine i
      J(i) ← φ         ← jobs assigned to machine i
   }

   for j = 1 to n {
      i = argmink Lk          ← machine i has smallest load
      J(i) ← J(i) ∪ {j}       ← assign job j to machine i
      Li ← Li + tj            ← update load of machine i
   }
   return J(1), …, J(m)
}
```

11

Load Balancing: LPT Rule

Observation. If at most m jobs, then list-scheduling is optimal.
Pf. Each job put on its own machine. ▪

Lemma 3. If there are more than m jobs, $L^* \geq 2\, t_{m+1}$.
Pf.
- Consider first m+1 jobs $t_1, \ldots, t_{m+1}$.
- Since the t_i's are in descending order, each takes at least t_{m+1} time.
- There are m+1 jobs and m machines, so by pigeonhole principle, at least one machine gets two jobs. ▪

Theorem. LPT rule is a 3/2 approximation algorithm.
Pf. Same basic approach as for list scheduling.

$$L_i \;=\; \underbrace{(L_i - t_j)}_{\leq\, L^*} \;+\; \underbrace{t_j}_{\leq\, \frac{1}{2}L^*} \;\leq\; \tfrac{3}{2} L^*. \qquad ▪$$

Lemma 3
(by observation, can assume number of jobs > m)

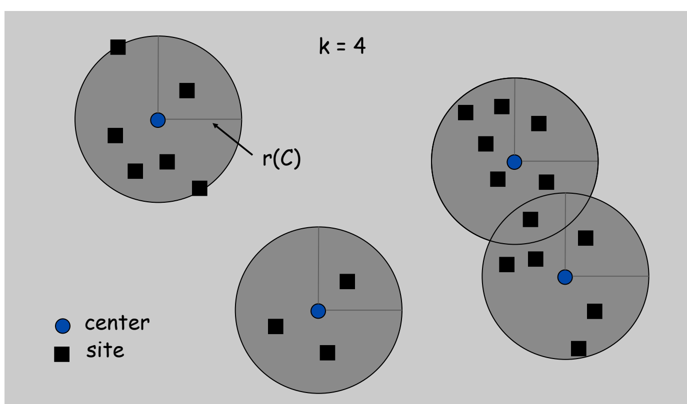
12

11.2 Center Selection

Center Selection Problem

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

Center selection problem. Select k centers C so that maximum distance from a site to nearest center is minimized.



15

Center Selection Problem

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

Center selection problem. Select k centers C so that maximum distance from a site to nearest center is minimized.

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

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

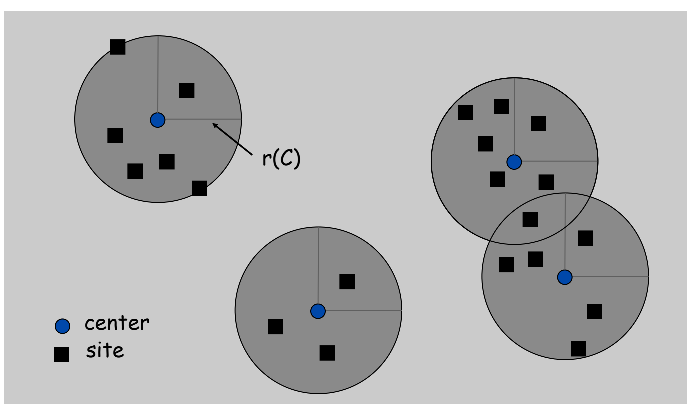
Distance function properties.
- dist(x, x) = 0 (identity)
- dist(x, y) = dist(y, x) (symmetry)
- dist(x, y) ≤ dist(x, z) + dist(z, y) (triangle inequality)

16

Center Selection Example

Ex: each site is a point in the plane, a center can be any point in the plane, dist(x, y) = Euclidean distance.

Remark: search can be infinite!

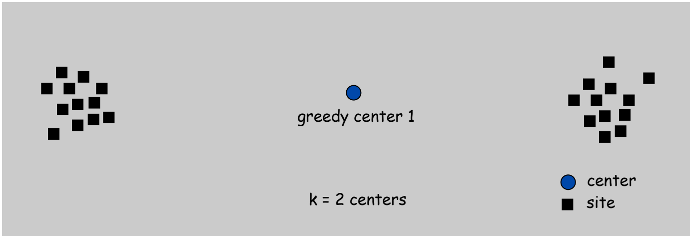


17

Greedy Algorithm: 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: arbitrarily bad!



Center Selection: Greedy Algorithm

Greedy algorithm. Repeatedly choose the next center to be the site farthest from any existing center.

```
Greedy-Center-Selection(k, n, s1,s2,…,sn) {

   C = φ
   Select any site s and add s to C;
   repeat k-1 times {
      Select a site si with maximum dist(si, C)
      Add si to C
   }                         site farthest from any center
   return C
}
```

Observation. Upon termination all centers in C are pairwise at least $r(C)$ apart.
Pf. By construction of algorithm.

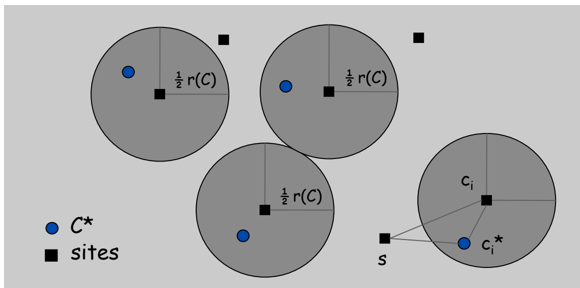
Center Selection: Analysis of Greedy Algorithm

Theorem. Let C^* be an optimal set of centers. Then $r(C) \le 2r(C^*)$.
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^*.
- $dist(s, C) \le dist(s, c_i) \le dist(s, c_i^*) + dist(c_i^*, c_i) \le 2r(C^*)$.
- Thus $r(C) \le 2r(C^*)$. ▪

Δ-inequality

$\le r(C^*)$ since c_i^* is closest center



20

Center Selection

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

Theorem. Greedy algorithm is a 2-approximation for center selection problem.

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

e.g., points in the plane

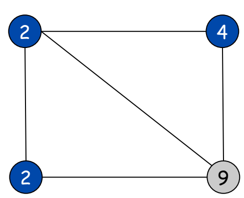
21

11.4 The Pricing Method: Vertex Cover

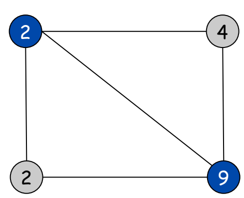
Weighted Vertex Cover

Definition. 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



weight = 11

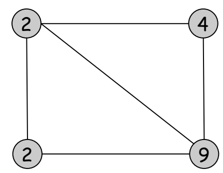
24

Pricing Method

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

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

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



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.

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

```
Weighted-Vertex-Cover-Approx(G, w) {
   foreach e in E
      pe = 0

   while (∃ edge i-j such that neither i nor j are tight)
      select such an edge e
      increase pe as much as possible until i or j tight
   }

   S ← set of all tight nodes
   return S
}
```

26

Pricing Method

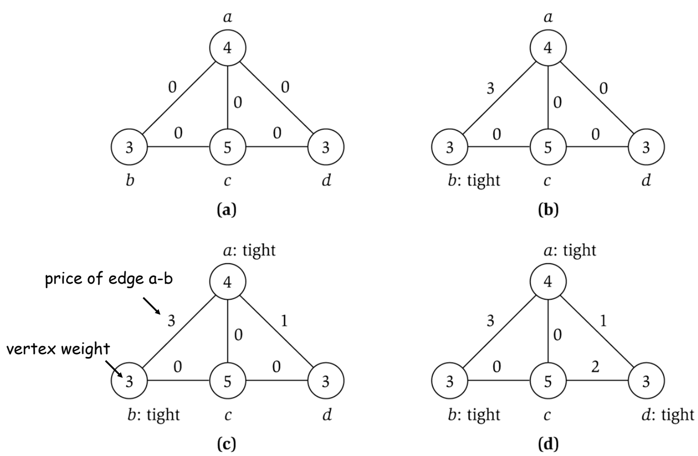


Figure 11.8

27

Pricing Method: Analysis

Theorem. Pricing method is a 2-approximation.
Pf.

- Algorithm terminates since at least one new node becomes tight after each iteration of 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 while loop would not terminate.
- Let S* be optimal vertex cover. We show w(S) ≤ 2w(S*).

$$w(S) = \underbrace{\sum_{i \in S} w_i = \sum_{i \in S} \sum_{e=(i,j)} p_e}_{\text{all nodes in S are tight}} \;\underset{S \subseteq V,\ \text{prices} \geq 0}{\leq}\; \sum_{i \in V} \sum_{e=(i,j)} p_e \;\underset{\text{each edge counted twice}}{=}\; 2 \sum_{e \in E} p_e \;\underset{\text{fairness lemma}}{\leq}\; 2w(S^*). \quad \blacksquare$$

28