The Design and Analysis of Algorithms

Lecture 20 Approximation Algorithm I

Zhenbo Wang

Department of Mathematical Sciences, Tsinghua University

Content

Load Balancing

Center Selection

Approximation Algorithms

Q. Suppose I need to solve an NP-hard problem. What should I do?

A. Theory says it is unlikely to find a poly-time algorithm.

Practice. Must sacrifice one of three desired features.

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

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!

Load Balancing

Input. m identical machines; n jobs, job j has processing time t_j.

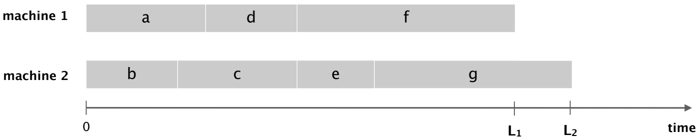
- Job j must run contiguously on one machine.
- A machine can process at most one job at a time.

Def. Let $J(i)$ be the subset of jobs assigned to machine i.

The *load* of machine i is $L_i = \sum_{j \in J(i)} t_j$.

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

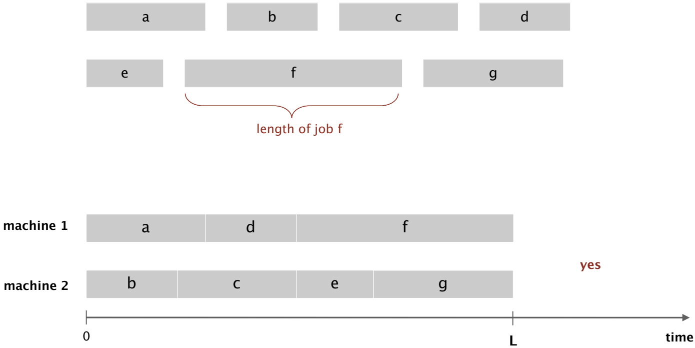
Def. *Load balancing.* Assign each job to a machine to minimize makespan.



Load Balancing on 2 machines is NP-hard

Claim. Load balancing is NP-hard even if only 2 machines.

Pf. NUMBER-PARTITIONING $\leq_P$ LOAD-BALANCE.



Load Balancing: List Scheduling Algorithm

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

for $i = 1$ to m **do**
    $L_i \leftarrow 0$
    $J(i) \leftarrow \emptyset$
end for
for $j = 1$ to n **do**
    $i = argmin_k L_k$
    $J(i) \leftarrow J(i) \cup j$
    $L_i \leftarrow L_i + t_j$
end for
return $J(1), \cdots, J(m)$.

- Running time: $O(n \log m)$.

Load Balancing: List Scheduling Analysis

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 $\sum_j t_j$.

One of m machines must do at least a $1/m$ fraction of total work. □

Load Balancing: List Scheduling Analysis

Theorem 3 (Graham 1966)

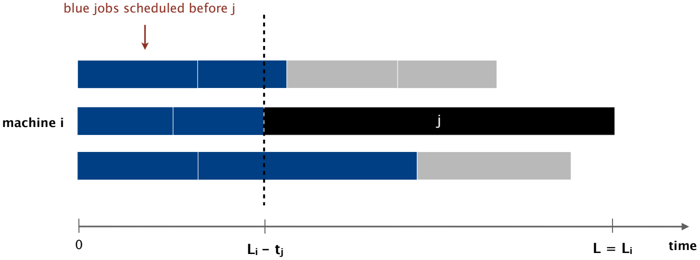
List Scheduling is a 2-approximation algorithm.

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 $L_i - t_j \Rightarrow L_i - t_j \leq L_k$ for all $1 \leq k \leq m$.



Load Balancing: List Scheduling Analysis

Sum inequalities over all k and divide by m, by lemma 2:

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

Now, by lemma 1:

$$L_i = (L_i - t_j) + t_j \leq 2L^*. \square$$

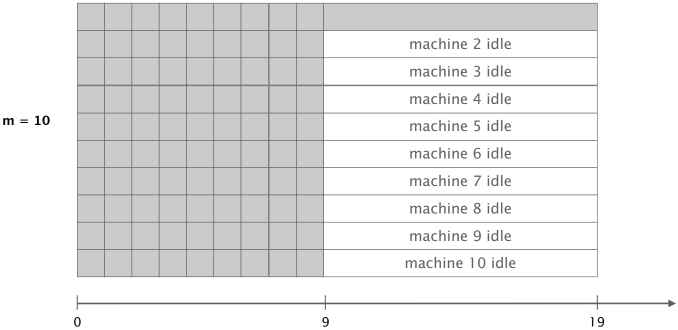
Load Balancing: List Scheduling Analysis

Q. Is our analysis tight?

A. Essentially yes.

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

list scheduling makespan = 19



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.

$LPT(m, n, t_1, t_2, \cdots, t_n)$

```
Sort jobs so that t_1 ≥ t_2 ≥ ··· ≥ t_n
for i = 1 to m do
    L_i ← 0
    J(i) ← ∅
end for
for j = 1 to n do
    i = argmin_k L_k
    J(i) ← J(i) ∪ j
    L_i ← L_i + t_j
end for
return J(1), ··· , J(m).
```

Load Balancing: LPT Rule

Claim. If at most m jobs, then list-scheduling is optimal.

Pf. Each job put on its own machine.

Lemma 4

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

Pf. Consider first $m+1$ jobs $t_1, \cdots, t_{m+1}$.

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. □

Load Balancing: LPT Rule

Theorem 5

LPT rule is a 3/2-approximation algorithm.

Pf. Same basic approach as for list scheduling.

$$L_i = (L_i - t_j) + t_j \leq \frac{3}{2} L^*. \square$$

Q. Is our 3/2 analysis tight?

A. No.

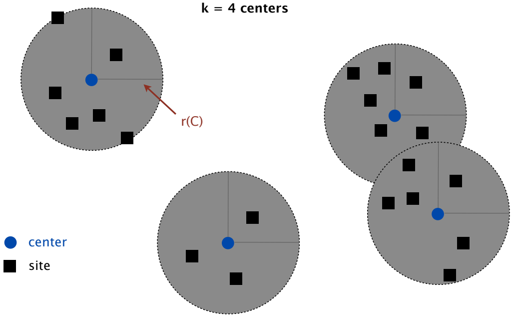
Theorem 6 (Graham 1969)

LPT rule is a $(\frac{4}{3} - \frac{1}{3m})$-approximation algorithm.

Center Selection Problem

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

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



Center Selection Problem

Notation.

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

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

Distance function properties.

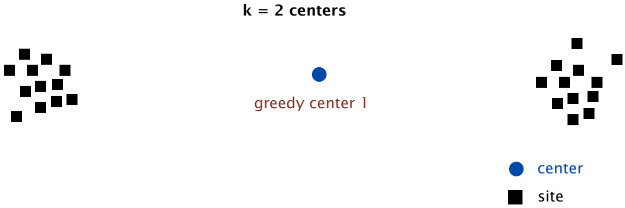
- $dist(x, x) = 0$
- $dist(x, y) = dist(y, x)$
- $dist(x, y) \leq dist(x, z) + dist(z, y)$

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

- Repeatedly choose next center to be site *farthest* from any existing center.

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

$C \leftarrow \emptyset$.
for $i = 1$ to k **do**
    Select a site s_i with maximum distance $dist(s_i, C)$
    $C \leftarrow C \cup s_i$
end for
return C

Lemma 7

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

Pf. By construction of algorithm. □

Center Selection: Analysis of Greedy Algorithm

Lemma 8

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

Pf. [by contradiction] Assume $r(C^*) < 1/2r(C)$.

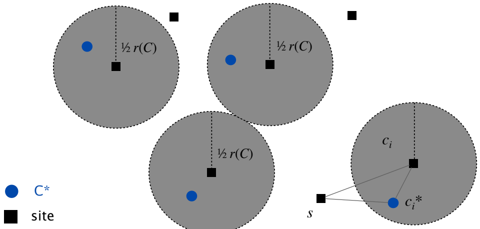
For each site $c_i \in C$, consider ball of radius $1/2r(C)$ around it.

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 $c_i^* \in C^*$.

$dist(s, C) \le dist(s, c_i) \le dist(s, c_i^*) + dist(c_i^*, c_i) \le 2r(C^*) < r(C)$.

Thus, $dist(s, C) < r(C)$ for any s, a contradiction. $\square$



Center Selection

Theorem 9

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.

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

Theorem 10

Unless P = NP, there no ρ-approximation for center selection problem for any $\rho < 2$.

Dominating Set Reduces to Center Selection

- *Dominating-SET*. Given a graph $G = (V, E)$ and integer k, is there a subset D of V such that every vertex not in D is adjacent to at least one member of D and $|D| = k$?
- *Dominating-SET* is *NP-Complete*.

Claim If there is a $(2 - \epsilon)$ approximation algorithm for *CENTER-SELECTION*, we can solve *DOMINATING-SET* in poly-time.

Pf. Let $G = (V, E)$, k be an instance of *DOMINATING-SET*.

Construct instance G' of *CENTER-SELECTION* with sites V and distances

$$dist(u, v) = 1 \text{ if } (u, v) \in E$$
$$dist(u, v) = 2 \text{ if } (u, v) \notin E$$

Dominating Set Reduces to Center Selection

Note that G' satisfies the triangle inequality.

G has dominating set of size k iff there exists k centers C^* with $r(C^*) = 1$.

Thus, G has a dominating set of size k iff a $(2 - \epsilon)$-approximation algorithm for CENTER-SELECTION find a solution C^* with $r(C^*) = 1$. $\square$

Homework

- Read Chapter 11 of the textbook.
- Exercises 5 & 6 in Chapter 11.