COMPSCI 311: Introduction to Algorithms

Lecture 25: Approximation Algorithms

Dan Sheldon

University of Massachusetts Amherst

Coping With NP-Completeness

Suppose you want to solve an NP-complete problem?
What should you do?

You can't design an algorithm to do *all* of the following:

1. Solve arbitrary instances of the problem
2. Solve problem to optimality
3. Solve problem in polynomial time

Coping strategies

1. Design algorithms for special cases of problem.
2. Design approximation algorithms or heuristics.
3. Design algorithms that run efficiently for some, but not all, problem instances

Approximation Algorithms

- **Def**: ρ-approximation algorithm
  - Runs in polynomial time
  - Solves arbitrary instances of the problem
  - Guaranteed to find a solution within ratio ρ of optimum:

$$\frac{\text{value of our solution}}{\text{value of optimum solution}} \leq \rho \qquad \text{(if goal = minimum)}$$

Today: load balancing

Load Balancing

Input:

- Machines $1, 2, \ldots, m$ (identical)
- Jobs $1, 2, \ldots, n$ with time t_j for jth job
- Any job can run on any machine

Goal:

- Assign jobs to *balance load*
- A_i = set of jobs assigned to machine i
- Minimize completion time = largest load of any machine = "makespan"

Clicker

Let T^* be the optimal makespan, i.e., the smallest possible completion time of any assignment. What can we say about T^*?

A. $T^* \geq \frac{1}{m} \sum_{j=1}^{n} t_j$ (at least as big as the average machine load)

B. $T^* \geq \max_j t_j$ (at least as big as the largest job time)

C. Both A and B.

D. Neither A nor B.

Preliminary Analysis

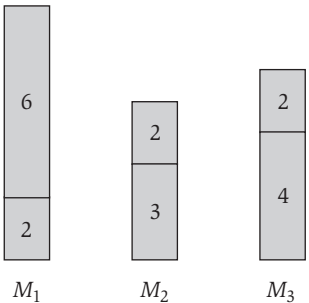
Two lower bounds for optimal solution:

1. $T^* \geq \frac{1}{m} \sum_{j=1}^{n} t_j$
2. $T^* \geq \max_j t_j$

Proof of 1. Otherwise,

$$\begin{aligned} \text{total processing time} &\leq mT^* \\ &< m\frac{1}{m}\sum_{j=1}^{n} t_j \\ &= \sum_{j=1}^{n} t_j \\ &= \text{total processing time} \end{aligned}$$

Simple Algorithm: Assign to lightest load



Example: jobs with times 2, 3, 4, 6, 2, 2 arrive in order

for i $= 1$ to m **do** $T_i = 0, A_i = \emptyset$

for j $= 1$ to n **do**

Choose i s.t. T_i is minimum

$T_i = T_i + t_j$

$A_i = A_i \cup \{j\}$

Complexity? $O(n \log m)$ with priority queue

Clicker

Suppose the jobs with times 6, 4, 3, 2, 2, 2 arrive in the order listed, and are scheduled on three machines by the simple algorithm. What will the final makespan be?

A. 6

B. 7

C. 8

D. 9

Analysis

Consider moment when job leading to highest load is added

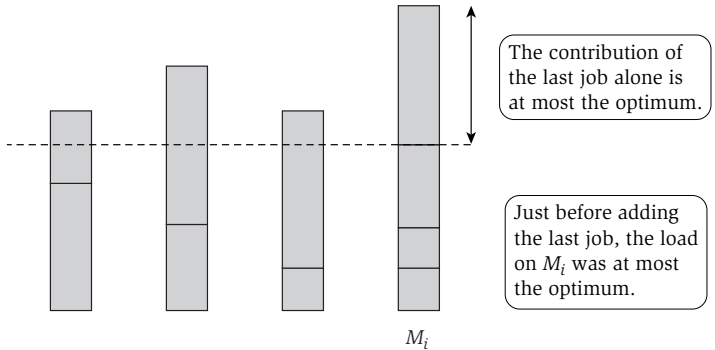


Figure 11.2 Accounting for the load on machine M_i in two parts: the last job to be added, and all the others.

Analysis

Consider moment when job leading to highest load is added; call this job j

$$\text{new load} = \text{old load} + t_j$$

At that time:

- old load was smallest among all machines

$$\text{old load} < \frac{1}{m} \sum_{k=1}^{n} t_k \leq T^*$$

- Therefore

$$\text{new load} = \text{old load} + t_j < T^* + T^* = 2T^*$$

The algorithm gives a **2-approximation**.

Clicker

Our lightest load algorithm immediately assigns each job received.
Considering all possible orderings of the same set of jobs, which of the following is true?

(Hint: consider jobs with times 4, 3, 3, 2 on two machines.)

A. Getting the largest job first is always best.

B. Getting the largest job last is always best.

C. None of the above

Worst Case

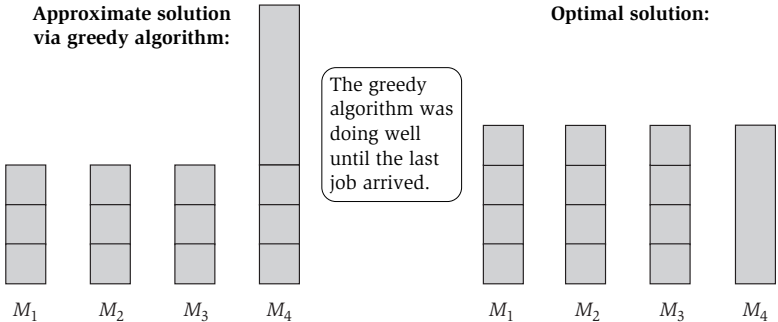


Figure 11.3 A bad example for the greedy balancing algorithm with $m = 4$.

Worst case is arbitrarily close to 2: with $m(m-1)$ jobs of time 1 followed by one of time m, lightest load gives makespan $2m-1$, but optimal makespan is m.

Improved Algorithm: Large Jobs First

Intuition: large job coming last is worst case $\implies$ sort jobs by time: $t_1 \geq t_2 \geq \ldots \geq t_n$. Then follow same algorithm as before (assign each job to machine with lightest load).

Observation: if $n > m$, then one machine must do two jobs from set $t_1, t_2, \ldots, t_{m+1}$, so

$$T^* \geq t_m + t_{m+1} \geq 2t_{m+1} \quad \implies t_{m+1} \leq T^*/2$$

Largest Jobs First: Analysis

Again, consider moment when job j leading to highest load is added.

$$\text{new load} = \text{old load} + t_j$$

If $j \leq m$, job will be added to empty machine

$$\text{new load} = 0 + t_j \leq T^*$$

If $j > m$, we have $t_j \leq t_{m+1}$

$$\text{old load} < \frac{1}{m} \sum_{k=1}^{n} t_k \leq T^*$$

$$\text{new load} < \frac{1}{m} \sum_{k=1}^{n} t_k + t_j \leq T^* + t_{m+1} \leq T^* + 1/2T* = 1.5T^*$$

Algorithm is a 1.5-approximation (no load is $>$ 1.5 x optimum)

More careful analysis can improve bound to 4/3 (tight)