Chapter 11
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.

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

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

**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 \( L = \max_i L_i \).

**Load balancing.** Assign each job to a machine to minimize makespan.
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, t_1, t_2, ..., t_n) {
    for i = 1 to m {
        L_i ← 0 ← load on machine i
        J(i) ← ∅ ← jobs assigned to machine i
    }

    for j = 1 to n {
        i = argmin_k L_k ← machine i has smallest load
        J(i) ← J(i) ∪ \{j\} ← assign job j to machine i
        L_i ← L_i + t_j ← update load of machine i
    }
}
```

Implementation. $O(n \log n)$ using a priority queue.
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 $\sum_j t_j$.
- One of $m$ machines must do at least a $1/m$ fraction of total work. ·
Load Balancing: List Scheduling Analysis

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

```plaintext
blue jobs scheduled before j
```

```plaintext
machine i
```

```plaintext
0 \quad L_i - t_j \quad L = L_i
```
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 $L_i - t_j \Rightarrow L_i - t_j \leq L_k$ for all $1 \leq k \leq m$.
- Sum inequalities over all $k$ and divide by $m$:

\[
L_i - t_j \leq \frac{1}{m} \sum_k L_k = \frac{1}{m} \sum_k t_k \leq L^* \]

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

\[\text{Lemma 2} \]
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 $m$

$m = 10$

<table>
<thead>
<tr>
<th></th>
<th>machine 2 idle</th>
<th>machine 3 idle</th>
<th>machine 4 idle</th>
<th>machine 5 idle</th>
<th>machine 6 idle</th>
<th>machine 7 idle</th>
<th>machine 8 idle</th>
<th>machine 9 idle</th>
<th>machine 10 idle</th>
</tr>
</thead>
</table>

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, t_1, t_2, ..., t_n) {
    Sort jobs so that t_1 ≥ t_2 ≥ ... ≥ t_n

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

    for j = 1 to n {
        i = argmin_k L_k ← machine i has smallest load
        J(i) ← J(i) ∪ {j} ← assign job j to machine i
        L_i ← L_i + t_j ← update load of machine i
    }
}
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, \( L^* \geq 2t_{m+1} \).

**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.

\[
L_i = \underbrace{(L_i - t_j)}_{\leq L^*} + \underbrace{t_j}_{\leq \frac{1}{2}L^*} \leq \frac{3}{2}L^*. ·
\]

( by observation, can assume number of jobs > m )
11.2 Center Selection
Input. Set of n sites $s_1, \ldots, s_n$.

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

Input. Set of $n$ sites $s_1, \ldots, s_n$.

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

Notation.
- $\text{dist}(x, y) =$ 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.
- $r(C) = \max_i \text{dist}(s_i, C) =$ 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)
Center Selection Example

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

**Remark:** search can be infinite!
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, s_1, s_2, ..., s_n) {
    C = ∅
    repeat k times {
        Select a site s_i with maximum dist(s_i, C)
        Add s_i to C
    }
    return C
}
```

**Observation.** Upon termination all centers in C are pairwise at least r(C) apart.

**Pf.** By construction of algorithm.
**Theorem.** 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 $C^*$.
- $\operatorname{dist}(s, C) \leq \operatorname{dist}(s, c_i) \leq \operatorname{dist}(s, c_i^*) + \operatorname{dist}(c_i^*, c_i) \leq 2r(C^*)$.
- Thus $r(C) \leq 2r(C^*)$. $\Delta$-inequality $\leq r(C^*)$ since $c_i^*$ is closest center.
11.4 The Pricing Method: Vertex Cover
Weighted Vertex Cover

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

```
vertex 1: 2 2 4
vertex 2: 2 9

weight = 2 + 2 + 4

vertex 1: 2 4
vertex 2: 2 9

weight = 9
```
Weighted Vertex Cover

**Pricing method.** Each edge must be covered by some vertex $i$. Edge $e$ pays price $p_e \geq 0$ to use vertex $i$.

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

For each vertex $i$:

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

**Claim.** For any vertex cover $S$ and any fair prices $p_e$: $\sum_e p_e \leq w(S)$.

**Proof.**

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

- each edge $e$ covered by at least one node in $S$
- sum fairness inequalities for each node in $S$
Pricing Method

Pricing method. Set prices and find vertex cover simultaneously.

Weighted-Vertex-Cover-Approx(G, w) {
    foreach e in E
        \[ p_e = 0 \]
        \[ \sum_{e=(i,j)} p_e = w_i \]

    while (\exists \text{edge } i-j \text{ such that neither } i \text{ nor } j \text{ are tight})
        select such an edge e
        increase \[ p_e \] without violating fairness

    S \leftarrow \text{set of all tight nodes}
    return S
}
Pricing Method

Figure 11.8

(a) 

(b) 

(c) 

(d)
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 $w(S) \leq 2w(S^*)$.

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

\[\uparrow \quad \uparrow \quad \uparrow \quad \uparrow \]

all nodes in $S$ are tight  \hspace{1cm} \hspace{1cm} \hspace{1cm} S \subseteq V, \hspace{1cm} \text{prices } \geq 0 \hspace{1cm} \text{each edge counted twice} \hspace{1cm} \text{fairness lemma}