COMPSCI 311: Introduction to Algorithms

Lecture 23: Approximation Algorithms

Marius Minea

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. Use randomization
   (efficient in expectation and/or optimal with high probability)

Approximation Algorithms

- ρ-approximation algorithm
  - Runs in polynomial time
  - Solves arbitrary instance of the problem
  - Guaranteed to find a solution within ratio ρ of optimum:
    $$\frac{\text{value of our solution}}{\text{value of optimum solution}} \le \rho$$

Today:

- Load Balancing
- Clustering

Load Balancing

There are m machines and n jobs to be done.

Assign jobs to *balance load* (largest load is minimal)
$\Rightarrow$ minimum completion time (*makespan*)

Machines: $1, 2, \ldots, m$
Job times: $t_1, t_2, \ldots, t_n$
Assignment: $A_i \subseteq \{1, 2, \ldots, n\}$ = set of jobs for machine i

Preliminary Analysis

Let T^* be the optimal makespan (smallest possible completion time)
What can we say about T^*?

- $T^* \ge \dfrac{1}{m}\displaystyle\sum_{j=1}^{n} t_j$

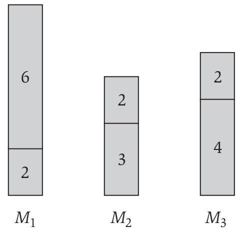
otherwise, total processing time $< m \cdot \frac{1}{m}\sum_{j=1}^{n} t_j = \sum_{j=1}^{n} t_j$

- $T^* \ge \max\limits_{j} t_j$ (at least as much as largest job time)

Simple Algorithm: Assign to lightest load

```
for i = 1 to m do $T_i = 0, A_i = \emptyset$
for j = 1 to n do
    choose minimum $T_i$
    $T_i = T_i + t_j, A_i = A_i \cup \{j\}$
```

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

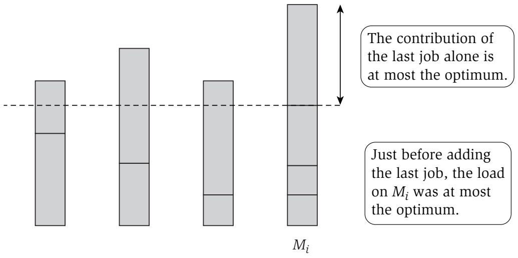


Example: result for jobs 2, 3, 4, 6, 2, 2 (in order)

What if order is 6, 4, 3, 2, 2, 2 ?

Analysis

Consider moment when last job is added, leading to highest load



The contribution of the last job alone is at most the optimum.

Just before adding the last job, the load on M_i was at most the optimum.

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.

new load = old load + t_i

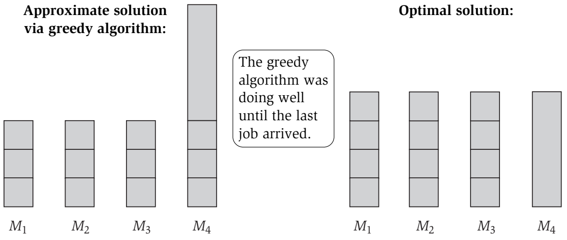
The old load was smallest among all machines at the time and the total load included at most all jobs without i:

$$\text{old load} \le \frac{1}{m} \cdot ((\sum_{j=1}^{n} t_j) - t_i) < \frac{1}{m} \sum_{j=1}^{n} t_j \le T^*$$

new load = old load + $t_i < T^* + T^* = 2T^*$

The algorithm gives a 2-approximation.

Worst Case



Approximate solution via greedy algorithm:

Optimal solution:

The greedy algorithm was doing well until the last job arrived.

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

Worst case is arbitrarily close to 2:
Consider $m(m-1)$ jobs of time 1. They will be perfectly balanced.
Then a huge job of time m comes along ⇒ makespan $2m - 1$

Optimal distribution would have job of size m by itself, makespan m

Improved: Large Jobs First

Intuition: large job coming last is worst case ⇒ sort jobs by time: $t_1 \ge t_2 \ge \ldots \ge t_n$. Again, assign next job to smallest load.

One more observation:
if $m < n$, one machine must do two jobs from set $t_1, t_2, \ldots, t_{m+1}$

$\Rightarrow T^* \ge t_m + t_{m+1} \ge 2t_{m+1}$

Largest Jobs First: Analysis

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

new load = old load + t_i

If $i \le m$, will be added to empty machine

new load $= 0 + t_i \le T^*$

If $i > m$, we have $t_i \le t_{m+1}$

old load $< \frac{1}{m} \sum_{j=1}^{n} t_j \le T^*$

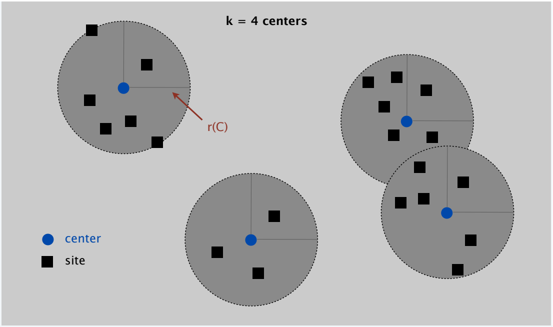
new load $< \frac{1}{m} \sum_{j=1}^{n} t_j + t_i \le T^* + t_{m+1} \le T^* + 1/2T* = 1.5T^*$

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

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

Clustering or Center Selection

Find k centers covering all given points, with *minimal radius* (k is fixed and given)



Problem Setup

- **Input:** set of n points $P = \{p_1, p_2, \ldots, p_n\}$ in $\mathbb{R}^2$. A number k.
- **Goal:** Find k centers $C = \{c_1, c_2, \ldots, c_k\}$ in $\mathbb{R}^2$ such that every point $p \in P$ is close to some center $c \in C$.

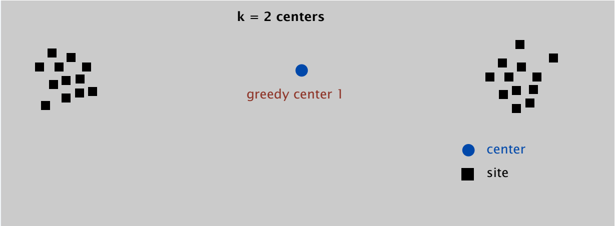
Want to minimize $\max_{p \in P} d(p, C)$
where $d(p, C) = \min_i d(p, c_i)$

Equivalent statement: find minimum value R such that all points can be covered with k discs of radius R.

Can apply to other notions of distance, as long as it's symmetric and satisfies triangle inequality.

Greedy can be arbitrarily bad!

Place first center at best location, each next one to get best reduction of radius



First center will be placed in the middle, $R \simeq 1/2 \cdot maxDist$

No matter where you place second center, R does not decrease (one cluster still closer to first center)

But: could have placed centers within the two clusters.

Knowing Optimal Radius Helps

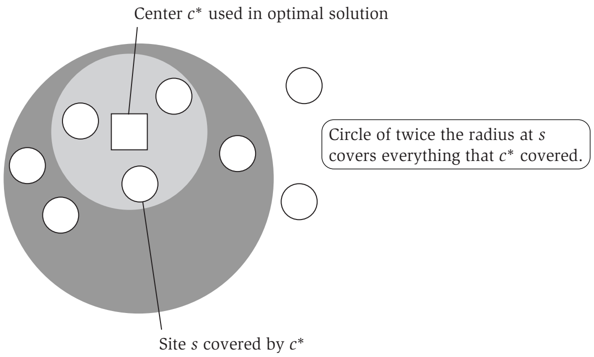

Center c^* used in optimal solution
Circle of twice the radius at s covers everything that c^* covered.
Site s covered by c^*

Figure 11.4 Everything covered at radius r by c^* is also covered at radius $2r$ by s.

Algorithm assuming optimal radius

```
Let $C = \emptyset$
**while** $P \neq \emptyset$ **do**
    choose $p \in P$, let $C = C \cup \{p\}$
    delete from $P$ all points at distance $\leq 2r$ from $p$
**if** $|C| \leq k$ **then** solution found
**else** there is no solution with radius $\leq r$
```

Correctness Proof

Any solution found has radius $\leq 2r$ – by design

Assume algorithm returns more than k centers.
Then any cover with $\leq k$ centers has radius $> r$.

Proof by contradiction. Assume cover $|C^*| \leq k$ with radius $r^* \leq r$

Each greedy center $c \in C$ is covered by some *close* optimal center $c^* \in C^*$, with $d(c, c^*) \leq r^*$.

Each optimal center $c*$ can't be close to two greedy centers c, c'.
Triangle inequality would give
$$d(c, c') \leq d(c, c^*) + d(c^*, c') \leq r^* + r^* \leq 2r$$
but $d(c, c') > 2r$ since greedy algorithm eliminates closer points.

Thus, each greedy center c has a *distinct* optimal center c^*, and $|C| \leq |C^*|$, contradiction.

What if we don't know the optimal radius?

It's not reasonable to assume we know the solution

We know $0 < r^* \leq maxDist$ between two points

Refine interval for covering radius by binary search.
Start with $maxDist/2$

Each try: there is a set with radius $2r$ or there is no set with radius r

But there is a greedy algorithm without knowing optimal radius!

Greedy Algorithm that Works

Our algorithm avoids overlap by choosing a new center that is at least $2r$ away from all selected centers.

Replace this condition by choosing a center that is *furthest away* from all selected centers!

```
if $k \geq |P|$ then return $P$
choose $p \in P$, let $C = \{p\}$
while $|C| < k$ do
    choose $p \in P$ maximizing $d(p, C)$
    $C = C \cup \{p\}$
return $C$
```

Claim: algorithm returns C with $r(C) \leq 2r^*$
(at most twice optimal radius)

Correctness Proof

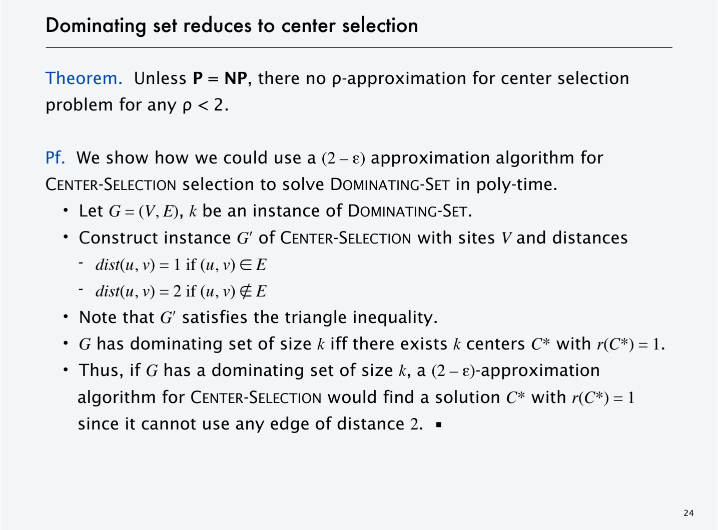
Similar argument: assume $r(C) > 2r^*$.

There must be a point p more than $2r^*$ away from any center in C.

Claim: whenever the algorithm adds a center c' to current C', it is at least $2r^*$ away from all selected centers (because we choose the farthest, and p is $> 2r^*$ away):

$$d(c', C') \geq d(s, C') \geq d(s, C) > 2r^*.$$

So our algorithm is a correct implementation of the previous one, but that algorithm would still not have selected p after k iterations, so no cover with $\leq r^*$ would exist, contradiction!



Dominating set reduces to center selection

Theorem. Unless **P** = **NP**, there no ρ-approximation for center selection problem for any ρ < 2.

Pf. We show how we could use a $(2 - \varepsilon)$ approximation algorithm for CENTER-SELECTION selection to solve DOMINATING-SET in poly-time.

- Let $G = (V, E)$, k be an instance of DOMINATING-SET.
- Construct instance G' of CENTER-SELECTION with sites V and distances
  - $dist(u, v) = 1$ if $(u, v) \in E$
  - $dist(u, v) = 2$ if $(u, v) \notin E$
- 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, if G has a dominating set of size k, a $(2 - \varepsilon)$-approximation algorithm for CENTER-SELECTION would find a solution C^* with $r(C^*) = 1$ since it cannot use any edge of distance 2. ▪

slide credit: Kevin Wayne / Pearson