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:** \( \rho \)-approximation algorithm
  - Runs in polynomial time
  - Solves arbitrary instances of the problem
  - Guaranteed to find a solution within ratio \( \rho \) of optimum:
    \[ \frac{\text{value of our solution}}{\text{value of optimum solution}} \leq \rho \] (if goal = minimum)

Today:
- Load Balancing
- Clustering

Load Balancing

Input:
- Machines 1, 2, ..., \( m \) (identical)
- Jobs 1, 2, ..., \( n \) with time \( t_j \) for \( j \)th 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”

Preliminary Analysis

Two lower bounds for optimal solution:
1. \( T^* \geq \frac{1}{m} \sum_{j=1}^{n} t_j \) (at least as big as the average job time)
2. \( T^* \geq \max_j t_j \) (at least as big as the largest job time)

**Proof** of 1. Otherwise, total processing time \( \leq mT^* \)
\[
< m \frac{1}{m} \sum_{j=1}^{n} t_j = \sum_{j=1}^{n} t_j = \text{total processing time}
\]
**Chapter 11 Approximation Algorithms**

The greedy algorithm was doing well until the last job arrived. Consider moment when job leading to highest load is added; call this \( j \) object.

\[
\text{new load} = \text{old load} + t_j
\]

At that time:

- old load was smallest among all machines
  
  \[
  \text{old load} \leq \frac{1}{m} \sum_{k=1}^{n} t_k \leq T^* 
  \]

- Therefore
  
  \[
  \text{new load} = \text{old load} + t_j < T^* + T^* = 2T^*
  \]

The algorithm gives a 2-approximation.

---

**Simple Algorithm: Assign to lightest load**

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

\[
\text{for } i = 1 \text{ to } n \text{ do } T_i = 0, A_i = \emptyset \\
\text{for } j = 1 \text{ to } n \text{ do } \\
\text{Choose } i \text{ s.t. } T_i \text{ is minimum} \\
T_i = T_i + t_j \\
A_i = A_i \cup \{j\}
\]

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

---

**Clicker**

Suppose the jobs with times 6, 4, 3, 2, 2 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

---

**Worst Case**

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 \( \Rightarrow \) makespan \( 2m - 1 \)

Optimal distribution would have job of size \( m \) by itself, makespan \( m \).
**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 $d(p, C) = \min_{c \in C} d(p, c)$

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

Use any distance measure, if symmetric and satisfies triangle inequality.

---

**Largest Jobs First: Analysis**

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

- new load = old load + $t_j$

If $j \leq m$, job will be added to empty machine
new load = $0 + t_j \leq T^*$

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

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

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

Algorithm is a 1.5-approximation (no load is $> 1.5 \times$ optimum)
More careful analysis can improve bound to 4/3 (tight)
Knowing Optimal Radius Helps

First step: assume optimal radius known

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.

Greedy Algorithm that Works

Original algorithm avoids overlap by choosing a new center that is at least \( 2r \) away from all selected centers.

New: choose 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!
Can we do better? Not if $P \neq NP$!

**Theorem:** If $P \neq NP$, there is no $\rho$-approximation for center selection for $\rho < 2$.

**Proof:** If so, could solve Dominating-Set in polynomial time.

**Dominating-Set:** each node covers itself and all connected nodes. Is there a cover of size $\leq k$?

Construct center selection instance with same nodes. Set distances:
- $d(u, v) = 1$ if $(u, v) \in E$ (edge in original graph)
- $d(u, v) = 2$ otherwise

$G$ has dominating set of size $k$ iff $G'$ has $k$ centers with radius 1.

A $(2 - \epsilon)$-approximation algorithm could find such a set, and thus solve Dominating-Set in polynomial time!