Approximation algorithms part two
more constant factor approximations

Benjamin Sach
Approximation Algorithms Recap

An algorithm $A$ is an $\alpha$-approximation for problem $P$ if,

- $A$ runs in polynomial time
- $A$ always outputs a solution with value $s$
  
  within an $\alpha$ factor of $\text{Opt}$

Here $P$ is an optimisation problem with optimal solution of value $\text{Opt}$

- If $P$ is a maximisation problem, $\frac{\text{Opt}}{\alpha} \leq s \leq \text{Opt}$
- If $P$ is a minimisation problem, $\text{Opt} \leq s \leq \alpha \cdot \text{Opt}$

We have seen a $3/2$-approximation for Bin Packing (and a faster $2$-approximation)
Scheduling Jobs on Parallel Machines

Goal: minimise the (wall-clock) time taken to process all jobs

$m$ identical machines

$n$ jobs

time taken
Scheduling Jobs on Parallel Machines

Goal: minimise the (wall-clock) time taken to process all jobs

(it’s NP-hard)
Scheduling Jobs on Parallel Machines

**Goal:** minimise the *wall-clock* time taken to process all jobs

wall-clock time *(also called makespan)*
Scheduling Jobs on Parallel Machines

Goal: minimise the (wall-clock) time taken to process all jobs

- Job $j$ takes $t_j$ time units

wall-clock time (also called makespan)
Scheduling Jobs on Parallel Machines

1. Goal: minimise the (wall-clock) time taken to process all jobs

   - Job \( j \) takes \( t_j \) time units

   - We say that \( j \in J(i) \) iff job \( j \) is assigned to machine \( i \)

   - wall-clock time (also called makespan)
Goal: minimise the (wall-clock) time taken to process all jobs

- We say that $j \in J(i)$ iff job $j$ is assigned to machine $i$
- The load of machine $i$ is $L_i = \sum_{j \in J(i)} t_j$
Scheduling Jobs on Parallel Machines

Goal: minimise the (wall-clock) time taken to process all jobs

- We say that $j \in J(i)$ iff job $j$ is assigned to machine $i$
- The load of machine $i$ is $L_i = \sum_{j \in J(i)} t_j$

wall-clock time (also called makespan)
Scheduling Jobs on Parallel Machines

Goal: minimise the (wall-clock) time taken to process all jobs

- We say that \( j \in J(i) \) iff job \( j \) is assigned to machine \( i \)
- The load of machine \( i \) is \( L_i = \sum_{j \in J(i)} t_j \)
- So the wall-clock time is \( \max_i L_i \) (which we want to minimise)
Scheduling Jobs on Parallel Machines

Goal: minimise the (wall-clock) time taken to process all jobs

- We say that $j \in J(i)$ iff job $j$ is assigned to machine $i$
- The load of machine $i$ is $L_i = \sum_{j \in J(i)} t_j$
- So the wall-clock time is $\max_i L_i$ (which we want to minimise)
Scheduling Jobs on Parallel Machines
Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Scheduling Jobs on Parallel Machines

Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
**Algorithm:** Put job $j$ on the machine $i$ with smallest (current) load
Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Scheduling Jobs on Parallel Machines

Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Scheduling Jobs on Parallel Machines

Algorithm: Put job \( j \) on the machine \( i \) with smallest (current) load
Scheduling Jobs on Parallel Machines

Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Scheduling Jobs on Parallel Machines

Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Scheduling Jobs on Parallel Machines

Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Scheduling Jobs on Parallel Machines

Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Scheduling Jobs on Parallel Machines

Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Algorithm: Put job $j$ on the machine $i$ with smallest (current) load
Algorithm: Put job $j$ on the machine $i$ with smallest (current) load

*How long does it take to compute this schedule?*
Algorithm: Put job $j$ on the machine $i$ with smallest (current) load

How long does it take to compute this schedule?
Algorithm: Put job $j$ on the machine $i$ with smallest (current) load

How long does it take to compute this schedule?

$O(nm)$ time naively, $O(n \log m)$ time using a priority queue
Algorithm: Put job $j$ on the machine $i$ with smallest (current) load

How long does it take to compute this schedule?

$O(nm)$ time naively, $O(n \log m)$ time using a priority queue

(it's also an online solution)
Scheduling Jobs on Parallel Machines

Algorithm: Put job \( j \) on the machine \( i \) with smallest (current) load

How long does it take to compute this schedule?

- \( O(nm) \) time naively,
- \( O(n \log m) \) time using a priority queue

(it's also an online solution)

How good is it?
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs.

Let $T_g$ denote the time taken by the greedy schedule.

**Theorem** The greedy algorithm given is a 2-approximation.
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a $2$-approximation

- Before we prove this, we prove two useful facts,
The greedy approximation

Let $Opt$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a $2$-approximation

- Before we prove this, we prove two useful facts,

**Fact** $Opt \geq \max_j t_j$
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a 2-approximation

- Before we prove this, we prove two useful facts,

**Fact** $\text{Opt} \geq \max_j t_j$

  - Some machine must process the largest job

$L_i$ is the load of machine $i$

Job $j$ takes $t_{j}$ time units

$m$ machines
$n$ jobs
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs.
Let $T_g$ denote the time taken by the greedy schedule.

**Theorem** The greedy algorithm given is a 2-approximation.

- Before we prove this, we prove two useful facts,

**Fact** $\text{Opt} \geq \max_j t_j$

  - Some machine must process the largest job.

**Fact** $\text{Opt} \geq \frac{\sum_j t_j}{m}$

$L_i$ is the load of machine $i$.

Job $j$ takes $t_j$ time units.

$m$ machines

$n$ jobs.
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs.
Let $T_g$ denote the time taken by the greedy schedule.

**Theorem** The greedy algorithm given is a 2-approximation.

- Before we prove this, we prove two useful facts,

**Fact** $\text{Opt} \geq \max_j t_j$

  - Some machine must process the largest job.

**Fact** $\text{Opt} \geq \frac{\sum_j t_j}{m}$

  - There is a total of $\sum_j t_j$ time units of work to be done.

$L_i$ is the load of machine $i$.

Job $j$ takes $t_j$ time units.

$m$ machines

$n$ jobs.
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs

Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a 2-approximation

- Before we prove this, we prove two useful facts,

**Fact** $\text{Opt} \geq \max_j t_j$

  - Some machine must process the largest job

**Fact** $\text{Opt} \geq \frac{\sum_j t_j}{m}$

  - There is a total of $\sum_j t_j$ time units of work to be done
  - Some machine $i$ must have load $L_i$ at least $\frac{\sum_j t_j}{m}$
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a $2$-approximation

- Before we prove this, we prove two useful facts,

**Fact** $\text{Opt} \geq \max_j t_j$

  - Some machine must process the largest job

**Fact** $\text{Opt} \geq \frac{\sum_j t_j}{m}$

  - There is a total of $\sum_j t_j$ time units of work to be done

  - Some machine $i$ must have load $L_i$ at least $\frac{\sum_j t_j}{m}$

  *(the $m$ machines can’t all have below average load)*
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs

Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a 2-approximation

$L_i$ is the load of machine $i$

Job $j$ takes $t_{ij}$ time units

$m$ machines

$n$ jobs
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a $2$-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs.
Let $T_g$ denote the time taken by the greedy schedule.

**Theorem** The greedy algorithm given is a 2-approximation.

**Proof** Consider the machine $i$ with largest load $T_g = L_i$.

$L_i$ is the load of machine $i$.

<table>
<thead>
<tr>
<th>Machine</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

$T_g$ is the time taken by the greedy schedule.

Job $j$ takes $t_{ij}$ time units.

$m$ machines

$n$ jobs
The greedy approximation

Let $Opt$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a $2$-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$

- Let $j$ denote the last job machine $i$ completes

$m$ machines
$n$ jobs
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs.
Let $T_g$ denote the time taken by the greedy schedule.

**Theorem** The greedy algorithm given is a $2$-approximation.

**Proof** Consider the machine $i$ with largest load $T_g = L_i$.

- Let $j$ denote the last job machine $i$ completes.

$L_i$ is the load of machine $i$.

Job $j$ takes $t_j$ time units.

$m$ machines
$n$ jobs.
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a 2-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$

- Let $j$ denote the last job machine $i$ completes
- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$

$L_i$ is the load of machine $i$

Job $j$ takes $t_j$ time units

$m$ machines
$n$ jobs
Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs. Let $T_g$ denote the time taken by the greedy schedule.

**Theorem** The greedy algorithm given is a 2-approximation.

**Proof** Consider the machine $i$ with largest load $T_g = L_i$.

- Let $j$ denote the last job machine $i$ completes.
- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$.

$L_i$ is the load of machine $i$.

Job $j$ takes $t_j$ time units.

$m$ machines

$n$ jobs
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs.
Let $T_g$ denote the time taken by the greedy schedule.

**Theorem** The greedy algorithm given is a 2-approximation.

**Proof** Consider the machine $i$ with largest load $T_g = L_i$.

- Let $j$ denote the last job machine $i$ completes.
- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$.

$L_i$ is the load of machine $i$.

Job $j$ takes $t_j$ time units.

$m$ machines

$n$ jobs
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a 2-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$

- Let $j$ denote the last job machine $i$ completes
- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$

So…

$L_i$ is the load of machine $i$

Job $j$ takes $t_j$ time units

$m$ machines
$n$ jobs
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs

Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a $2$-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$

- Let $j$ denote the last job machine $i$ completes

- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leq L_k \text{ for all } 1 \leq k \leq m,$$
The greedy approximation

Let \( \text{Opt} \) denote the time taken by the optimal scheduling of jobs
Let \( T_g \) denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a 2-approximation

**Proof** Consider the machine \( i \) with largest load \( T_g = L_i \)

- Let \( j \) denote the last job machine \( i \) completes
- When job \( j \) was assigned, machine \( i \) had the smallest load, \( L_i - t_j \)

So...

\[
L_i - t_j \leq L_k \text{ for all } 1 \leq k \leq m,
\]

- If we then sum over all \( k \),

\( m \) machines
\( n \) jobs
The greedy approximation

Let \( \text{Opt} \) denote the time taken by the optimal scheduling of jobs
Let \( T_g \) denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a 2-approximation

**Proof** Consider the machine \( i \) with largest load \( T_g = L_i \)

- Let \( j \) denote the last job machine \( i \) completes
- When job \( j \) was assigned, machine \( i \) had the smallest load, \( L_i - t_j \)

So...

\[
L_i - t_j \leq L_k \text{ for all } 1 \leq k \leq m,
\]

- If we then sum over all \( k \),

\[
m(L_i - t_j) \leq \sum_{k=1}^{m} L_k
\]
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a $2$-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$

- Let $j$ denote the last job machine $i$ completes

- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leq L_k \text{ for all } 1 \leq k \leq m,$$

- If we then sum over all $k$,

$$m(L_i - t_j) \leq \sum_{k=1}^{m} L_k$$

so $(L_i - t_j) \leq \frac{\sum_{k=1}^{m} L_k}{m} \leq \text{Opt}$ (by the second fact)
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs

Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a 2-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$

- Let $j$ denote the last job machine $i$ completes

- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leq L_k \text{ for all } 1 \leq k \leq m,$$

- If we then sum over all $k$,

$$m(L_i - t_j) \leq \sum_{k=1}^{m} L_k$$

so $$(L_i - t_j) \leq \frac{\sum_{k=1}^{m} L_k}{m} \leq \text{Opt} \text{ (by the second fact)}$$

**Fact** $\text{Opt} \geq \frac{\sum_j t_j}{m}$
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a 2-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$

- Let $j$ denote the last job machine $i$ completes
- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leq L_k \text{ for all } 1 \leq k \leq m,$$

- If we then sum over all $k$,

$$m(L_i - t_j) \leq \sum_{k=1}^{m} L_k$$

so

$$(L_i - t_j) \leq \frac{\sum_{k=1}^{m} L_k}{m} \leq \text{Opt} \text{ (by the second fact)}$$
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a $2$-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$

- Let $j$ denote the last job machine $i$ completes
- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leq L_k \text{ for all } 1 \leq k \leq m,$$

- If we then sum over all $k$,

$$m(L_i - t_j) \leq \sum_{k=1}^{m} L_k$$

so $(L_i - t_j) \leq \frac{\sum_{k=1}^{m} L_k}{m} \leq \text{Opt}$ (by the second fact)

also $t_j \leq \text{Opt}$ (by the first fact)
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a $2$-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$

- Let $j$ denote the last job machine $i$ completes

- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leq L_k \text{ for all } 1 \leq k \leq m,$$

- If we then sum over all $k$,

$$m(L_i - t_j) \leq \sum_{k=1}^{m} L_k$$

so $(L_i - t_j) \leq \frac{\sum_{k=1}^{m} L_k}{m} \leq \text{Opt}$ (by the second fact)

also $t_j \leq \text{Opt}$ (by the first fact)

**Fact** $\text{Opt} \geq \max_j t_j$
The greedy approximation

Let $\text{Opt}$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a $2$-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$

- Let $j$ denote the last job machine $i$ completes
- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$

  So...

  \[ L_i - t_j \leq L_k \quad \text{for all } 1 \leq k \leq m, \]

- If we then sum over all $k$,

  \[ m(L_i - t_j) \leq \sum_{k=1}^{m} L_k \]

  so \( (L_i - t_j) \leq \frac{\sum_{k=1}^{m} L_k}{m} \leq \text{Opt} \) (by the second fact)

  also \( t_j \leq \text{Opt} \) (by the first fact)
The greedy approximation

Let $Opt$ denote the time taken by the optimal scheduling of jobs
Let $T_g$ denote the time taken by the greedy schedule

**Theorem** The greedy algorithm given is a 2-approximation

**Proof** Consider the machine $i$ with largest load $T_g = L_i$

- Let $j$ denote the last job machine $i$ completes
- When job $j$ was assigned, machine $i$ had the smallest load, $L_i - t_j$

So...

$$L_i - t_j \leq L_k \text{ for all } 1 \leq k \leq m,$$

- If we then sum over all $k$,

$$m(L_i - t_j) \leq \sum_{k=1}^{m} L_k$$

so $$(L_i - t_j) \leq \frac{\sum_{k=1}^{m} L_k}{m} \leq Opt \text{ (by the second fact)}$$

also $t_j \leq Opt \text{ (by the first fact)}$

Therefore, $T_g = L_i = (L_i - t_j) + t_j \leq Opt + Opt = 2Opt$
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)
Step 1: Sort the jobs into non-increasing order (job 1 is now largest)
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job \( j \) on the machine \( i \) with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

1. Sort the jobs into non-increasing order (job 1 is now largest)
2. Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job \( j \) on the machine \( i \) with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job \( j \) on the machine \( i \) with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)
Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)
Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job $j$ on the machine $i$ with smallest (current) load
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)
Step 2: Put job \( j \) on the machine \( i \) with smallest (current) load

How long does it take to compute this schedule?
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)

Step 2: Put job \( j \) on the machine \( i \) with smallest (current) load

How long does it take to compute this schedule?

\( O(n \log n) \) time (to sort the jobs)
Longest Processing Time (LPT)

Step 1: Sort the jobs into non-increasing order (job 1 is now largest)
Step 2: Put job $j$ on the machine $i$ with smallest (current) load

How long does it take to compute this schedule?

$O(n \log n)$ time (to sort the jobs)

How good is it?
The LPT approximation

Let $T_l$ denote the time taken by the LPT schedule

**Theorem** The LPT algorithm is a $3/2$-approximation
The LPT approximation

- Let $T_l$ denote the time taken by the LPT schedule

**Theorem** The LPT algorithm is a $3/2$-approximation

- Before we prove this, we prove another useful fact and a Lemma

$L_i$ is the load of machine $i$

$m$ machines

$n$ jobs

Job $j$ takes $t_{.j}$ time units
The LPT approximation

- Let $T_l$ denote the time taken by the LPT schedule

**Theorem** The LPT algorithm is a $3/2$-approximation

- Before we prove this, we prove another useful fact and a Lemma

**Fact** If there are at most $m$ jobs ($n \leq m$) then LPT is optimal

$L_i$ is the load of machine $i$

$m$ machines

$n$ jobs

Job $j$ takes $t_j$

time units
The LPT approximation

- Let $T_l$ denote the time taken by the LPT schedule

**Theorem** The LPT algorithm is a $3/2$-approximation

- Before we prove this, we prove another useful fact and a Lemma

**Fact** If there are at most $m$ jobs ($n \leq m$) then LPT is optimal

If there are at most $m$ jobs then

- LPT gives each job its own machine so $\max_i L_i \leq \max_j t_j \leq \text{Opt}$

$L_i$ is the *load* of machine $i$  

$m$ machines  

$n$ jobs  

Job $j$ takes $t_j$ time units
The LPT approximation

- Let $T_l$ denote the time taken by the LPT schedule.

**Theorem** The LPT algorithm is a $3/2$-approximation.

- Before we prove this, we prove another useful fact and a Lemma.

**Fact** If there are at most $m$ jobs ($n \leq m$) then LPT is optimal.

If there are at most $m$ jobs then

LPT gives each job its own machine so $\max_i L_i \leq \max_j t_j \leq \text{Opt}$

**Lemma** If $n > m$ then $\text{Opt} \geq 2t(m+1)$ (after sorting).

$L_i$ is the load of machine $i$.

$m$ machines, $n$ jobs.

Job $j$ takes $t_j$ time units.
The LPT approximation

- Let $T_i$ denote the time taken by the LPT schedule

**Theorem** The LPT algorithm is a $3/2$-approximation

- Before we prove this, we prove another useful fact and a Lemma

**Fact** If there are at most $m$ jobs ($n \leq m$) then LPT is optimal

If there are at most $m$ jobs then

LPT gives each job its own machine so $\max_i L_i \leq \max_j t_j \leq \text{Opt}$

**Lemma** If $n > m$ then $\text{Opt} \geq 2t_{(m+1)}$ (after sorting)

**Proof**
The LPT approximation

- Let $T_i$ denote the time taken by the LPT schedule

**Theorem** The LPT algorithm is a $\frac{3}{2}$-approximation

- Before we prove this, we prove another useful fact and a Lemma

**Fact** If there are at most $m$ jobs ($n \leq m$) then LPT is optimal

If there are at most $m$ jobs then
LPT gives each job its own machine so $\max_i L_i \leq \max_j t_j \leq \text{Opt}$

**Lemma** If $n > m$ then $\text{Opt} \geq 2t(m+1)$ (after sorting)

**Proof**

- Note that $t_1 \geq t_2 \geq t_3 \geq \ldots t_m \geq t(m+1)$
The LPT approximation

- Let $T_i$ denote the time taken by the LPT schedule

**Theorem** The LPT algorithm is a $\frac{3}{2}$-approximation

- Before we prove this, we prove another useful fact and a Lemma

**Fact** If there are at most $m$ jobs ($n \leq m$) then LPT is optimal

- If there are at most $m$ jobs then
  - LPT gives each job its own machine so $\max_i L_i \leq \max_j t_j \leq \text{Opt}$

**Lemma** If $n > m$ then $\text{Opt} \geq 2t_{(m+1)}$ (after sorting)

**Proof**

- Note that $t_1 \geq t_2 \geq t_3 \geq \ldots t_m \geq t_{(m+1)}$

- One of the $m$ machines must be assigned
  (at least) two of these $m + 1$ jobs under any schedule
The LPT approximation

- Let $T_l$ denote the time taken by the LPT schedule

**Theorem** The LPT algorithm is a $3/2$-approximation

- Before we prove this, we prove another useful fact and a Lemma

**Fact** If there are at most $m$ jobs ($n \leq m$) then LPT is optimal

If there are at most $m$ jobs then

- LPT gives each job its own machine so $\max_i L_i \leq \max_j t_j \leq \text{Opt}$

**Lemma** If $n > m$ then $\text{Opt} \geq 2t(m+1)$ (after sorting)

**Proof**

- Note that $t_1 \geq t_2 \geq t_3 \geq \ldots t_m \geq t(m+1)$

- One of the $m$ machines must be assigned
  - (at least) two of these $m + 1$ jobs under any schedule

- So we have that any schedule takes at least $2t(m+1)$ time
The LPT approximation

- Let $T_l$ denote the time taken by the LPT schedule

**Theorem** The LPT algorithm is a $3/2$-approximation

- Before we prove this, we prove another useful fact and a Lemma

**Fact** If there are at most $m$ jobs ($n \leq m$) then LPT is optimal

  If there are at most $m$ jobs then
  LPT gives each job its own machine so $\max_i L_i \leq \max_j t_j \leq \text{Opt}$

**Lemma** If $n > m$ then $\text{Opt} \geq 2t(m+1)$ (after sorting)

**Proof**

- Note that $t_1 \geq t_2 \geq t_3 \geq \ldots t_m \geq t(m+1)$

- One of the $m$ machines must be assigned
  (at least) two of these $m + 1$ jobs under any schedule

- So we have that any schedule takes at least $2t(m+1)$ time
  in particular $\text{Opt} \geq 2t(m+1)$
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

$L_i$ is the load of machine $i$

$m$ machines

$n$ jobs

Job $j$ takes $t_j$

time units
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes

$L_i$ is the load of machine $i$

1 2 3 4 5

$m$ machines

$n$ jobs

Job $j$ takes $t_j$ time units
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

\[(L_i - t_j) \leq \text{Opt}\]
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

$$(L_i - t_j) \leq \text{Opt}$$

- If $n \leq m$ then we are done so assume $n > m$
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_i = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

  $$(L_i - t_j) \leq \text{Opt}$$

- If $n \leq m$ then we are done so assume $n > m$

because LPT is optimal in this case
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_i = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

$$ (L_i - t_j) \leq \text{Opt} $$

- If $n \leq m$ then we are done so assume $n > m$
**Theorem** The LPT algorithm is a $3/2$-approximation.

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes.
- Using the same argument as before, we have that,

$$(L_i - t_j) \leq \text{Opt}$$

- If $n \leq m$ then we are done so assume $n > m$.
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$
Theorem The LPT algorithm is a $3/2$-approximation.

Proof Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

$$ (L_i - t_j) \leq \text{Opt} $$

- If $n \leq m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$

Fact $\text{Opt} \geq \max_j t_j$
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

  $$(L_i - t_j) \leq \text{Opt}$$

- If $n \leq m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$

$L_i$ is the load of machine $i$

$m$ machines

$n$ jobs

Job $j$ takes $t_j$ time units
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

  $$(L_i - t_j) \leq \text{Opt}$$

- If $n \leq m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$

  so assume that $(L_i - t_j) > 0$
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

$$ (L_i - t_j) \leq \text{Opt} $$

- If $n \leq m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$
  
  so assume that $(L_i - t_j) > 0$

- Therefore machine $i$ was assigned at least two jobs
Theorem  The LPT algorithm is a $3/2$-approximation

Proof  Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

$$ (L_i - t_j) \leq \text{Opt} $$

- If $n \leq m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$
  so assume that $(L_i - t_j) > 0$

- Therefore machine $i$ was assigned at least two jobs
  By the algorithm description, we have that $j \geq m + 1$
The LPT approximation

Theorem  The LPT algorithm is a $\frac{3}{2}$-approximation

Proof  Consider the machine $i$ with largest load $T_l = L_i$

• Let $j$ denote the last job machine $i$ completes

• Using the same argument as before, we have that,

$$(L_i - t_j) \leq \text{Opt}$$

• If $n \leq m$ then we are done so assume $n > m$

• Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$

  so assume that $(L_i - t_j) > 0$

• Therefore machine $i$ was assigned at least two jobs

  By the algorithm description, we have that $j \geq m + 1$

  it doesn’t assign a second job to any machine until

  every machine has at least one job
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

$$ (L_i - t_j) \leq \text{Opt} $$

- If $n \leq m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$
  
  so assume that $(L_i - t_j) > 0$

- Therefore machine $i$ was assigned at least two jobs

  By the algorithm description, we have that $j \geq m + 1$
The LPT approximation

**Theorem** The LPT algorithm is a $\frac{3}{2}$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,
  
  $$(L_i - t_j) \leq \text{Opt}$$

- If $n \leq m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$
  
  so assume that $(L_i - t_j) > 0$

- Therefore machine $i$ was assigned at least two jobs
  
  By the algorithm description, we have that $j \geq m + 1$
  
  $$t_j \leq t_{m+1} \leq \text{Opt}/2 \text{ (by the Lemma)}$$
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

\[ (L_i - t_j) \leq \text{Opt} \]

- If $n \leq m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$
  
  so assume that $(L_i - t_j) > 0$

- Therefore machine $i$ was assigned at least two jobs
  
  By the algorithm description, we have that $j \geq m + 1$

\[ t_j \leq t_{m+1} \leq \text{Opt}/2 \] (by the Lemma)

**Lemma** If $n > m$ then $\text{Opt} \geq 2t_{(m+1)}$ (after sorting)
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

$$ (L_i - t_j) \leq \text{Opt} $$

- If $n \leq m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$
  
  so assume that $(L_i - t_j) > 0$

- Therefore machine $i$ was assigned at least two jobs
  
  By the algorithm description, we have that $j \geq m + 1$

  $$ t_j \leq t_{m+1} \leq \text{Opt}/2 \text{ (by the Lemma)} $$
The LPT approximation

**Theorem** The LPT algorithm is a $3/2$-approximation

**Proof** Consider the machine $i$ with largest load $T_l = L_i$

- Let $j$ denote the last job machine $i$ completes
- Using the same argument as before, we have that,

$$ (L_i - t_j) \leq \text{Opt} $$

- If $n \leq m$ then we are done so assume $n > m$
- Further if $(L_i - t_j) = 0$ then $T_l = L_i = t_j \leq \text{Opt}$
  so assume that $(L_i - t_j) > 0$

- Therefore machine $i$ was assigned at least two jobs
  
  By the algorithm description, we have that $j \geq m + 1$
  
  $$ t_j \leq t_{m+1} \leq \text{Opt}/2 \text{ (by the Lemma)} $$

Therefore, $T_l = L_i = (L_i - t_j) + t_j \leq \text{Opt} + \text{Opt}/2 = (3/2) \cdot \text{Opt}$
Scheduling conclusions

**Theorem** The greedy algorithm is a $2$-approximation which runs in $O(n \log m)$ time and it’s online.

**Theorem** The LPT algorithm is a $3/2$-approximation which runs in $O(n \log n)$ time.

In fact, LPT is a $4/3$-approximation (using better analysis).
$k$-centers

Goal: Minimise the largest distance from any site to the closest center.
$k$-centers

Goal
Minimise the largest distance from any site to the closest center
\(k\)-centers

\(n\) points (\textit{sites}) in 2D space

\textbf{Goal}

Minimise the largest distance from any site to the closest center
$k$-centers

$n$ points (sites) in 2D space

The distance between points $s_i, s_j$ is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Goal
Minimise the largest distance from any site to the closest center
$k$-centers

$n$ points (*sites*) in 2D space

The distance between points $s_i, s_j$ is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

(i.e. ‘normal’ euclidean distance)
$k$-centers

$n$ points (sites) in 2D space

Select $k$ sites to be centers

The distance between points $s_i, s_j$ is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

(i.e. ‘normal’ euclidean distance)
$k$-centers

$n$ points (sites) in 2D space

Select $k$ sites to be centers

The distance between points $s_i, s_j$ is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

(i.e. ‘normal’ euclidean distance)
$k$-centers

$n$ points (sites) in 2D space

Select $k$ sites to be centers

The distance between points $s_i, s_j$ is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
\( k \)-centers

\( n \) points (sites) in 2D space

The distance between points \( s_i, s_j \) is

\[
\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}
\]

**Goal** Minimise the largest distance from any site to the closest center
**k-centers**

*n* points (*sites*) in 2D space

The distance between points \( s_i, s_j \) is \( \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \)

**Goal** Minimise the largest distance from any site to the closest center
\( k\)-centers

\( n \) points (sites) in 2D space

The distance between points \( s_i, s_j \) is

\[
\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}
\]

**Goal** Minimise the largest distance from any site to the closest center.
$k$-centers

$n$ points (sites) in 2D space

Select $k$ sites to be centers

The distance between points $s_i, s_j$ is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

**Goal** Minimise the largest distance from any site to the closest center
$k$-centers

$n$ points (sites) in 2D space

Select $k$ sites to be centers

The distance between points $s_i, s_j$ is $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

Goal Minimise the largest distance from any site to the closest center
\textbf{\textit{k}-centers}

\textit{n} points (\textit{sites}) in 2D space

Select \textit{k} sites to be centers

The distance between points \( s_i, s_j \) is
\[
\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}
\]

\textbf{Goal} Minimise the largest distance from any site to the closest center
\( k \)-centers

\( n \) points (sites) in 2D space

The distance between points \( s_i, s_j \) is

\[
\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}
\]

Goal
Minimise the largest distance from any site to the closest center

(in general it’s NP-hard)

Select \( k \) sites to be centers
A Greedy approximation
A Greedy approximation

Start by picking any point to be a center
A Greedy approximation

Start by picking any point to be a center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center

This takes $O(nk)$ time
A Greedy approximation

Start by picking any point to be a center

Repeatedly pick the site which is furthest from any existing center

This takes $O(nk)$ time

but is it any good?
The Greedy approximation

**Theorem** The Greedy algorithm for \( k \)-center is a 2-approximation

**Proof**

Let \( C_g \) (resp. \( C_{\text{Opt}} \)) denote the set of centers selected by Greedy (resp. Optimal)

Let \( r_g \) (resp. \( r_{\text{Opt}} \)) denote largest site-center distance using Greedy (resp. Optimal)
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)

Let $r_g$ (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

**Case 1:** No $s_i, s_i' \in C_g$ are closest to the same $s_j \in C_{Opt}$
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)
Let $r_g$ (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

**Case 1:** No $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{Opt}$
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)

Let $r_g$ (resp. $Opt$) denote largest site-center distance using Greedy (resp. Optimal)

**Case 1:** No $s_i, s_i' \in C_g$ are closest to the same $s_j \in C_{Opt}$
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a $2$-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)

Let $r_g$ (resp. $Opt$) denote largest site-center distance using Greedy (resp. Optimal)

**Case 1:** No $s_i, s_i' \in C_g$ are closest to the same $s_j \in C_{Opt}$
The Greedy approximation

**Theorem** The Greedy algorithm for \( k \)-center is a 2-approximation

**Proof**

Let \( C_g \) (resp. \( C_{Opt} \)) denote the set of centers selected by Greedy (resp. Optimal)

Let \( r_g \) (resp. \( Opt \)) denote largest site-center distance using Greedy (resp. Optimal)

**Case 1:** No \( s_i, s_{i'} \in C_g \) are closest to the same \( s_j \in C_{Opt} \)
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)

Let $r_g$ (resp. $Opt$) denote largest site-center distance using Greedy (resp. Optimal)

**Case 1:** No $s_i, s_i' \in C_g$ are closest to the same $s_j \in C_{Opt}$
**The Greedy approximation**

**Theorem** The Greedy algorithm for \(k\)-center is a 2-approximation

**Proof**

Let \(C_g\) (resp. \(C_{\text{Opt}}\)) denote the set of centers selected by Greedy (resp. Optimal)

Let \(r_g\) (resp. \(\text{Opt}\)) denote largest site-center distance using Greedy (resp. Optimal)

**Case 1:** No \(s_i, s_i' \in C_g\) are closest to the same \(s_j \in C_{\text{Opt}}\)
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)
Let $r_g$ (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

**Case 1:** No $s_i, s_i' \in C_g$ are closest to the same $s_j \in C_{Opt}$

Disclaimer: for illustrative purposes only
**The Greedy approximation**

**Theorem** The Greedy algorithm for \( k \)-center is a 2-approximation

**Proof**

Let \( C_g \) (resp. \( C_{\text{Opt}} \)) denote the set of centers selected by Greedy (resp. Optimal)

Let \( r_g \) (resp. \( \text{Opt} \)) denote largest site-center distance using Greedy (resp. Optimal)

**Case 1**: No \( s_i, s_{i'} \in C_g \) are closest to the same \( s_j \in C_{\text{Opt}} \)

\[ r_g \leq 2\text{Opt} \]

\( \text{Distance at most } 2\text{Opt} \)

\( \text{so } r_g \leq 2\text{Opt} \)

Disclaimer: for illustrative purposes only
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)

Let $r_g$ (resp. $Opt$) denote largest site-center distance using Greedy (resp. Optimal)

**Case 2**: Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{Opt}$
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation

**Proof**

Let $C_g$ (resp. $C_{\text{Opt}}$) denote the set of centers selected by Greedy (resp. Optimal)

Let $r_g$ (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

**Case 2:** Some $s_i, s_i' \in C_g$ are closest to the same $s_j \in C_{\text{Opt}}$
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)

Let $r_g$ (resp. $Opt$) denote largest site-center distance using Greedy (resp. Optimal)

**Case 2:** Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{Opt}$

Assume wlog. that Greedy made $s_i$ a center after $s_{i'}$
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a $2$-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal).


**Case 2:** Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{Opt}$

Assume wlog. that Greedy made $s_i$ a center after $s_{i'}$

$s_i$ was added as a center because it was
**The Greedy approximation**

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)

Let $r_g$ (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

**Case 2:** Some $s_i, s_{i'} \in C_g$ are closest to the same $s_j \in C_{Opt}$

Assume wlog. that Greedy made $s_i$ a center after $s_{i'}$

$s_i$ was added as a center because it was the furthest from any existing Greedy center
The Greedy approximation

**Theorem** The Greedy algorithm for \( k \)-center is a 2-approximation.

**Proof**

Let \( C_g \) (resp. \( C_{Opt} \)) denote the set of centers selected by Greedy (resp. Optimal).

Let \( r_g \) (resp. \( Opt \)) denote largest site-center distance using Greedy (resp. Optimal).

**Case 2:** Some \( s_i, s_i' \in C_g \) are closest to the same \( s_j \in C_{Opt} \).

Assume wlog. that Greedy made \( s_i \) a center after \( s_i' \).

\( s_i \) was added as a center because it was the furthest from any existing Greedy center.

However, \( s_i \) is at most \( 2Opt \) away from \( s_i' \).
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)

Let $r_g$ (resp. $Opt$) denote largest site-center distance using Greedy (resp. Optimal)

**Case 2:** Some $s_i, s_i' \in C_g$ are closest to the same $s_j \in C_{Opt}$

Assume wlog. that Greedy made $s_i$ a center after $s_i'$

$s_i$ was added as a center because it was

the furthest from any existing Greedy center

However, $s_i$ is at most $2Opt$ away from $s_i'$

![Diagram showing the Greedy algorithm and its approximation properties](diagram.png)
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a $2$-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)

Let $r_g$ (resp. $Opt$) denote largest site-center distance using Greedy (resp. Optimal)

**Case 2:** Some $s_i, s_i' \in C_g$ are closest to the same $s_j \in C_{Opt}$

Assume wlog. that Greedy made $s_i$ a center after $s_i'$

$s_i$ was added as a center because it was the furthest from any existing Greedy center

However, $s_i$ is at most $2Opt$ away from $s_i'$

So even before adding $s_i$ as a center, all sites were $\leq 2Opt$ away from a Greedy center
The Greedy approximation

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation

**Proof**

Let $C_g$ (resp. $C_{Opt}$) denote the set of centers selected by Greedy (resp. Optimal)

Let $r_g$ (resp. Opt) denote largest site-center distance using Greedy (resp. Optimal)

**Case 2:** Some $s_i, s_i' \in C_g$ are closest to the same $s_j \in C_{Opt}$

Assume wlog. that Greedy made $s_i$ a center after $s_i'$

$s_i$ was added as a center because it was the furthest from any existing Greedy center

However, $s_i$ is at most $2Opt$ away from $s_i'$

So even before adding $s_i$ as a center, all sites were $\leq 2Opt$ away from a Greedy center

Therefore, $r_g \leq 2Opt$
Theorem The Greedy algorithm for $k$-center is a 2-approximation which runs in $O(nk)$ time.
The Greedy algorithm for $k$-center is a $2$-approximation which runs in $O(nk)$ time.

- The approximation works for any (metric) distance function,
**k-center Conclusions**

**Theorem** The Greedy algorithm for $k$-center is a 2-approximation which runs in $O(nk)$ time

- The approximation works for any (metric) distance function,

$$d(s_i, s_j) = L_1 \text{ or } L_\infty \text{ for example}$$
\textbf{Theorem} The Greedy algorithm for \( k \)-center is a 2-approximation which runs in \( O(nk) \) time

- The approximation works for any (metric) distance function,
  \[ d(s_i, s_j) = L_1 \text{ or } L_\infty \text{ for example} \]

- Distance function \( d \) is a metric iff
  \[ d(x, y) = d(y, x), \quad d(x, y) \geq 0 \]
  \[ (d(x, y) = 0 \text{ iff } x = y) \text{ and } d(x, z) \leq d(x, y) + d(y, z) \]
\(k\)-center Conclusions

**Theorem** The Greedy algorithm for \(k\)-center is a 2-approximation which runs in \(O(nk)\) time

- The approximation works for any (metric) distance function,
  \[d(s_i, s_j) = L_1 \text{ or } L_\infty \text{ for example}\]

- Distance function \(d\) is a metric iff
  \[d(x, y) = d(y, x), d(x, y) \geq 0\]
  \[(d(x, y) = 0 \text{ iff } x = y) \text{ and } d(x, z) \leq d(x, y) + d(y, z)\]

- For a general (metric) \(d\), the problem is not \(\alpha\)-approximable with \(\alpha < 2\)
**Theorem** The Greedy algorithm for \( k \)-center is a 2-approximation which runs in \( O(nk) \) time

- The approximation works for any (metric) distance function,
  
  \[ d(s_i, s_j) = L_1 \text{ or } L_\infty \text{ for example} \]

- Distance function \( d \) is a metric iff
  
  \[ d(x, y) = d(y, x), d(x, y) \geq 0 \]
  
  \[ (d(x, y) = 0 \text{ iff } x = y) \text{ and } d(x, z) \leq d(x, y) + d(y, z) \]

- For a general (metric) \( d \), the problem is not \( \alpha \)-approximable with \( \alpha < 2 \)

- For \( d = L_2 \), the problem is not \( \alpha \)-approximable with \( \alpha < \sqrt{3} \approx 1.73 \)
**Theorem** The Greedy algorithm for $k$-center is a 2-approximation which runs in $O(nk)$ time

- The approximation works for any (metric) distance function,
  \[ d(s_i, s_j) = L_1 \text{ or } L_{\infty} \text{ for example} \]

- Distance function $d$ is a metric iff
  \[
  d(x, y) = d(y, x), \quad d(x, y) \geq 0
  \]
  \[ (d(x, y) = 0 \text{ iff } x = y) \text{ and } d(x, z) \leq d(x, y) + d(y, z) \]

- For a general (metric) $d$, the problem is not $\alpha$-approximable with $\alpha < 2$
- For $d = L_2$, the problem is not $\alpha$-approximable with $\alpha < \sqrt{3} \approx 1.73$
- For $d = L_1$ or $d = L_{\infty}$, the problem is not $\alpha$-approximable with $\alpha < 2$