Approximation Algorithms

• NP-hard problems: choose 2 of
  • optimal
  • polynomial time
  • all instances

• Approximation algorithms. Trade-off between time and quality.

• Let $A(I)$ denote the value returned by algorithm $A$ on instance $I$. Algorithm $A$ is an $\alpha$-approximation algorithm if for any instance $I$ of the optimization problem:
  • $A$ runs in polynomial time
  • $A$ returns a valid solution
  • $A(I) \leq \alpha \cdot \text{OPT}$, where $\alpha \geq 1$, for minimization problems
  • $A(I) \geq \alpha \cdot \text{OPT}$, where $\alpha \leq 1$, for maximization problems

Scheduling jobs on a single machine

• $n$ jobs
• Each job $j$ has: processing time $p_j$, release date $r_j$, due date $d_j$.
• Once a job has begun processing it must be completed.
• Schedule starts at time 0.
• Lateness of job $j$ completed at time $C_j$: $L_j = C_j - d_j$.
• Goal. Schedule all jobs so as to minimize the maximum lateness:
  \[ \text{minimize } L_{\text{max}} = \max_{i=1...n} L_i \]
Scheduling jobs on a single machine

- \( n \) jobs
- Each job \( j \) has: processing time \( p_j \), release date \( r_j \), due date \( d_j \).
- Once a job has begun processing it must be completed.
- Schedule starts at time 0.
- Lateness of job \( j \) completed at time \( C_j \): \( L_j = C_j - d_j \).
- Goal. Schedule all jobs so as to minimize the maximum lateness:
  \[
  \text{minimize } L_{\text{max}} = \max_{i=1...n} L_i
  \]

NP-hard even to decide if all jobs can be completed by their due date.

- **Problem**: Assume optimal value is 0 then
  - \( \alpha \)-approximation algorithm must find a solution of value at most \( \alpha \cdot 0 = 0 \)
  - no such algorithm exists unless \( P=NP \).
- **Solution**: Assume all due dates are negative (optimal value always positive).

**Earliest Due Date Rule**

- Earliest due date rule (EDD). When machine idle: start processing an available job with earliest due date.

Earliest Due Date Rule

- **Earliest due date rule (EDD)**. When machine idle: start processing an available job with earliest due date.
Lower bound

- Let $S$ be a subset of jobs
  - $r(S) = \min_{j \in S} r_j$
  - $p(S) = \sum_{j \in S} p_j$
  - $d(S) = \max_{j \in S} d_j$
  - $L^*$ optimal value

- **Claim.** For any subset $S$ of jobs: $L^* \geq r(S) + p(S) - d(S)$.
- **Proof.**
  - Look at optimal schedule restricted to $S$.
  - No job can be processed before $r(S)$.
  - Needed processing time $p(S)$.
  - Latest job $i$ to be processed cannot complete earlier than $r(S) + p(S)$.
  - $d_i \leq d(S)$ => lateness of $i$ at least $r(S) + p(S) - d(S)$.
  - $L^* \geq L_i$

EDD: Approximation factor

- $j$: job with maximum lateness ($L_{max} = L_j = C_j - d_j$).
- $t$: earliest time before $C_j$ that machine idle (not idle in $[t,C_j]$).
- $S$: jobs processed in $[t,C_j]$.
- We have:
  - $r(S) = t$ and $p(S) = C_j - t$.
  - $C_i = p(S) + t = p(S) + r(S)$.
- Use Claim:
  - $L^* \geq r(S) + p(S) - d(S) \geq r(S) + p(S) = C_j$.
  - $L^* \geq r_j + p_j - d_j \geq - d_j$.
  - $L_{max} = C_j - d_j \leq 2L^*$.

Scheduling on identical parallel machines

- $n$ jobs to be scheduled on $m$ identical machines.
- Each job has a processing time $p$.
- Once a job has begun processing it must be completed.
- Schedule starts at time 0.
- Completion time of job $j = C_j$.
- Goal. Schedule all jobs so as to minimize the maximum completion time (makespan):
  $$\text{minimize } C_{max} = \max_{j=1,...,n} C_j$$

Local search

- Start with any schedule
- Consider job that finishes last:
  - If reassigning it to another machine can make it complete earlier, reassign it to the one that makes it finish earliest.
- Repeat until last finishing job cannot be transferred.
- The local search algorithm above is a 2-approximation algorithm:
  - polynomial time
  - valid solution ✓
  - factor 2
Each job must be processed: $C^* \geq \max_{i=1...n} p_i$.

There is a machine that is assigned at least average load: $C^* \geq \sum_{i=1...n} p_i/m$.

All other machines busy until start time $s$ of job $i$. ($s = C_i - p_i$)

Partition schedule into before and after $s$.

After $s \leq C^*$.

Before:

- All machines busy $\Rightarrow$ total amount of work $= m \cdot s$.
- $m \cdot s \leq \sum_{i=1...n} p_i \Rightarrow s \leq \sum_{i=1...n} p_i/m \leq C^*$.

Length of schedule $\leq 2C^*$.

Running time

- Polynomial time. Does it terminate?
- Minimum completion time of machines $C_{\min}$ never decreases.
- Remains same $\Rightarrow$ number of machines with minimum completion time decreases.
- No job transferred more than once:
  - Proof by contradiction. Assume job transferred twice.

Length of schedule $\leq 2C^*$.

• Longest processing time rule (LPT). Sort jobs in non-increasing order. Assign next job on list to machine as soon as it becomes idle.

• Longest processing time rule (LPT). Sort jobs in non-increasing order. Assign next job on list to machine as soon as it becomes idle.

• LPT is a 4/3-approximation algorithm:
  - polynomial time ✓
  - valid solution ✓
  - factor 4/3
• Longest processing time rule (LPT). Sort jobs in non-increasing order. Assign next job on list to machine as soon as it becomes idle.
• Assume $p_1 \geq \ldots \geq p_n$.
• Assume wlog that smallest job finishes last.
• If $p_n \leq C^*/3$ then $C_{\max} \leq 4/3 C^*$.
• If $p_n > C^*/3$ then each machine can process at most 2 jobs.
• Lemma. For any input where the processing time of each job is more than a third of the optimal makespan, LPT computes an optimal schedule.

The k-center problem

• Input. An integer $k$ and a complete, undirected graph $G=(V,E)$, with distance $d(i,j)$ between each pair of vertices $i,j \in V$.
• $d$ is a metric:
  • $d(i,i) = 0$
  • $d(i,j) = d(j,i)$
  • $d(i,j) \leq d(i,l) + d(l,j)$
• Goal. Choose a set $S \subseteq V$, $|S| = k$, of $k$ centers so as to minimize the maximum distance of a vertex to its closest center.

$$S = \arg\min_{|S| = k} \max_{i \in V} d(i,S)$$

• Covering radius. Maximum distance of a vertex to its closest center.

k-center: Greedy algorithm

• Greedy algorithm.
  • Pick arbitrary $i \in V$.
  • Set $S = \{i\}$
  • while $|S| < k$ do
    • Find vertex $j$ farthest away from any cluster center in $S$
    • Add $j$ to $S$

• Greedy is a 2-approximation algorithm:
  • polynomial time ✓
  • valid solution ✓
  • factor 2
**k-center: analysis**

- \( r^* \) optimal radius.
- Show all vertices within distance \( 2r^* \) from a center.
- Consider optimal clusters. 2 cases.
  - Algorithm picked one center in each optimal cluster
    - distance from any vertex to its closest center \( \leq 2r^* \) (triangle inequality)
  - Some optimal cluster does not have a center.
    - Some cluster have more than one center.
    - distance between these two centers \( \leq 2r^* \).
    - when second center in same cluster picked it was the vertex farthest away from any center.
    - distance from any vertex to its closest center at most \( 2r^* \).

**k-center: Inapproximability**

- There is no \( \alpha \)-approximation algorithm for the k-center problem for \( \alpha < 2 \) unless P=NP.
- **Proof.** Reduction from dominating set.
  - Dominating set: Given \( G=(V,E) \) and \( k \). Is there a (dominating) set \( S \subseteq V \) of size \( k \), such that each vertex is either in \( S \) or adjacent to a vertex in \( S \).
  - Given instance of the dominating set problem construct instance of k-center problem:
    - Complete graph \( G' \) on \( V \).
    - All edges from \( E \) has weight 1, all new edges have weight 2.
    - Radius in k-center instance 1 or 2.
    - \( G \) has an dominating set of size \( k \) \( \iff \) opt solution to the k-center problem has radius 1.
  - Use \( \alpha \)-approximation algorithm \( A \):
    - \( \text{opt} = 1 \) \( \implies \) \( A \) returns solution with radius at most \( \alpha < 2 \)
    - \( \text{opt} = 2 \) \( \implies \) \( A \) returns solution with radius 2.
    - Can use \( A \) to distinguish between the 2 cases.