Approximation Algorithms

02282
Inge Li Gørtz
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
Load balancing
• n jobs to be scheduled on m identical machines.
• Each job has a processing time $t_j$.
• Once a job has begun processing it must be completed.
• $T_j$: Load of machine j.
• Goal. Schedule all jobs so as to minimize the maximum load (makespan):

$$\text{minimize } T = \max_{i=1\ldots n} T_j$$
• **Simple greedy.** Process jobs in any order. Assign next job on list to machine with smallest current load.

• The greedy algorithm above is a 2-approximation algorithm:
  • polynomial time ✓
  • valid solution ✓
  • factor 2
Approximation factor

• Lower bounds:
  • Each job must be processed:
    \[ T^* \geq \max_j t_j \]
  • There is a machine that is assigned at least average load:
    \[ T^* \geq \frac{1}{m} \sum_j t_j \]
• i: job finishes last.
• All other machines busy until start time $s$ of i. ($s = T_i - t_i$)
• Partition schedule into before and after $s$.
• After $\leq T^*$. 
• Before:
  • All machines busy $\Rightarrow$ total amount of work $= m \cdot s$:

\[ m \cdot s \leq \sum_i t_i \quad \Rightarrow \quad s \leq \frac{1}{m} \sum_i t_i \leq T^* \]

• Length of schedule $\leq 2T^*$. 

Approximation factor
• *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 3/2-approximation algorithm:
- polynomial time ✓
- valid solution ✓
- factor 3/2
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 $t_1 \geq \ldots \geq t_n$.

Lower bound: If $n > m$ then $T^* \geq 2t_{m+1}$.

Factor 3/2:

- If $m \leq n$ then optimal.
- Before $\leq T^*$
  - After: $i$ job that finishes last.
    - $t_i \leq t_{m+1} \leq T^*/2$.
    - $T \leq T^* + T^*/2 \leq 3/2 T^*$.
- Tight?
• **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 $t_1 \geq \ldots \geq t_n$.

• Assume wlog that smallest job finishes last.

• If $t_n \leq T^*/3$ then $T \leq 4/3 T^*$.

• If $t_n > T^*/3$ then each machine can process at most 2 jobs in OPT.

• **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.

• **Theorem.** LPT is a 4/3-approximation algorithm.
Traveling salesman problem
• Set of cities \( \{1, \ldots, n\} \)
• \( c_{ij} \geq 0 \): cost of traveling from \( i \) to \( j \).
• \( c_{ij} \) a metric:
  • \( c_{ii} = 0 \)
  • \( c_{ij} = c_{ji} \)
  • \( c_{ij} \leq c_{ik} + c_{kj} \) (triangle inequality)
• Goal: Find a tour of minimum cost visiting every city exactly once.
Traveling Salesman Problem (TSP)

- Set of cities \{1,...,n\}
- \(c_{ij} \geq 0\): cost of traveling from i to j.
- \(c_{ij}\) a metric:
  - \(c_{ii} = 0\)
  - \(c_{ij} = c_{ji}\)
  - \(c_{ij} \leq c_{ik} + c_{kj}\)
- Goal: Find a tour of minimum cost visiting every city exactly once.
Double tree algorithm

- MST is a lower bound on TSP.
  - Deleting an edge $e$ from OPT gives a spanning tree.
  - $\text{OPT} \geq \text{OPT} - c_e \geq \text{MST}$.

- Eulerian graph
  - Graph Eulerian if there is a traversal of all edges visiting every edge exactly once.
  - $G$ Eulerian iff $G$ connected and all nodes have even degree.
  - Can construct Euler tour in polynomial time.
Double tree algorithm

- Double tree algorithm
  - Compute MST $T$.
  - Double edges of $T$
  - Construct Euler tour $\tau$
Double tree algorithm

- Double tree algorithm
  - Compute MST $T$.
  - Double edges of $T$
  - Construct Euler tour $\tau$
Double tree algorithm

- Double tree algorithm
  - Compute MST $T$.
  - Double edges of $T$
  - Construct Euler tour $\tau$
    - Shortcut $\tau$ such that each vertex only visited once ($\tau'$)
  - $\text{length}(\tau') \leq \text{length}(\tau) = 2 \cdot \text{weight}(T) \leq 2 \cdot \text{OPT}$.
- The double tree algorithm is a 2-approximation algorithm for TSP.
Christofides’ algorithm

• Christofides’ algorithm
  • Compute MST $T$.
  • No need to double all edges:
    • Consider set $O$ of all odd degree vertices in $T$.
    • Find minimum cost perfect matching $M$ on $O$.
      • Matching: no edges share an endpoint.
      • Perfect: all vertices matched.
      • Perfect matching on $O$ exists: Number of odd vertices in a graph is even.
    • $T + M$ is Eulerian (all vertices have even degree).
Christofides’ algorithm

- Compute MST $T$.
- $O = \{\text{odd degree vertices in } T\}$.
- Compute minimum cost perfect matching $M$ on $O$.
- Construct Euler tour $\tau$
- Shortcut such that each vertex only visited once ($\tau'$)
Christofides’ algorithm

- Compute MST $T$.
- $O = \{\text{odd degree vertices in } T\}$.
- Compute minimum cost perfect matching $M$ on $O$.
- Construct Euler tour $\tau$.
- Shortcut such that each vertex only visited once ($\tau'$)
Christofides’ algorithm

- Christofides’ algorithm
  - Compute MST $T$.
  - $O = \{\text{odd degree vertices in } T\}$.
  - Compute minimum cost perfect matching $M$ on $O$.
  - Construct Euler tour $\tau$
    - Shortcut such that each vertex only visited once ($\tau'$)
  - $\text{length}(\tau') \leq \text{length}(\tau) = \text{weight}(T) + \text{weight}(M) \leq \text{OPT} + \text{weight}(M)$. 
Analysis of Christofides’ algorithm

- $\text{weight}(M) \leq \frac{\text{OPT}}{2}$.
  - $\text{OPT}_o = \text{OPT}$ restricted to $O$.
  - $\text{OPT}_o \leq \text{OPT}$. 
Analysis of Christofides’ algorithm

- \text{weight}(M) \leq \text{OPT}/2.
  - \text{OPT}_o = \text{OPT} \text{ restricted to } O.
  - \text{OPT}_o \leq \text{OPT}.
Analysis of Christofides’ algorithm

- \( \text{weight}(M) \leq \text{OPT}/2. \)
  - \( \text{OPT}_O = \text{OPT} \) restricted to \( O \).
  - \( \text{OPT}_O \leq \text{OPT} \).
  - can partition \( \text{OPT}_O \) into two perfect matchings \( O_1 \) and \( O_2 \).
  - \( \text{weight}(M) \leq \min(\text{cost}(O_1), \text{cost}(O_2)) \leq \text{OPT}/2. \)
- \( \text{length}(\tau') \leq \text{length}(\tau) = \text{weight}(T) + \text{weight}(M) \leq \text{OPT} + \text{OPT}/2 = 3/2 \text{ OPT}. \)
- Christofides’ algorithm is a 3/2-approximation algorithm for TSP.