Approximation Algorithms

02282
Inge Li Gørtz

Approximation algorithms

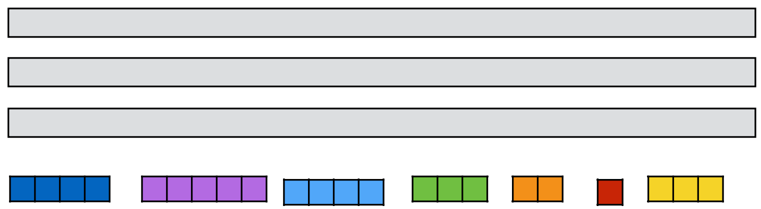
- Fast. Cheap. Reliable. Choose two.
- NP-hard problems: choose 2 of
  - optimal
  - polynomial time
  - all instances

- **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 *α-approximation algorithm* if for any instance I of the optimization problem:
  - A runs in polynomial time
  - A returns a valid solution
  - $A(I) \leq \alpha \cdot OPT$, where $\alpha \geq 1$, for minimization problems
  - $A(I) \geq \alpha \cdot OPT$, where $\alpha \leq 1$, for maximization problems

Load balancing

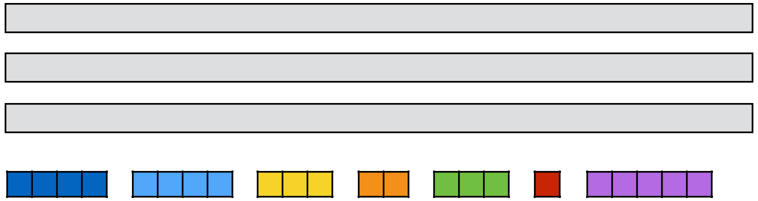
Scheduling on identical parallel machines



- 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 (list scheduling)



- *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

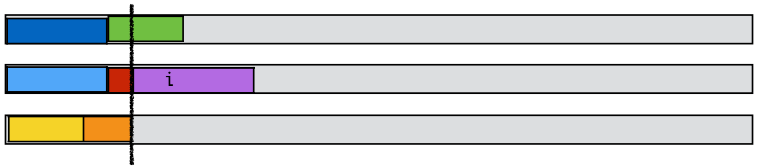
- Lower bounds:
  - Each job must be processed:

$$T^* \geq \max_j t_j$$

  - There is a machine that is assigned at least average load:

$$T^* \geq \frac{1}{m} \sum_j t_j$$

Approximation factor

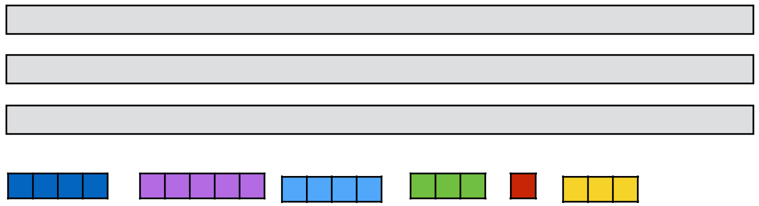


- i: job finishes last.
- All other machines busy until start time s of i. ($s = T_i - t_i$)
- Partition schedule into before and after s.
- After ≤ T*.
- Before:
  - All machines busy => total amount of work = m·s:

$$m \cdot s \leq \sum_j t_j \qquad \Rightarrow s \leq \frac{1}{m} \sum_j t_j \leq T^*$$

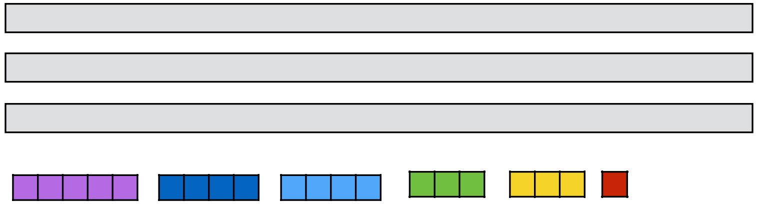
  - Length of schedule ≤ T*+ T* = 2T*.

Longest processing time rule



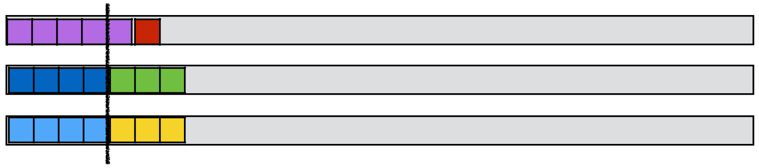
- *Longest processing time rule (LPT).* Sort jobs in non-increasing order. Assign next job on list to machine as soon as it becomes idle.

Longest processing time rule



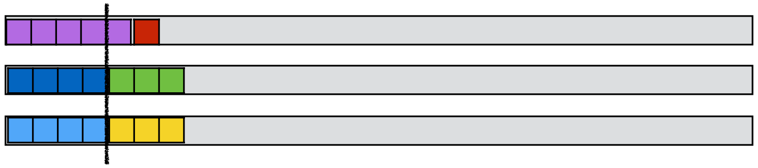
- *Longest processing time rule (LPT).* Sort jobs in non-increasing order. Assign next job on list to machine as soon as it becomes idle.
- LPT is a is a 3/2-approximation algorithm:
  - polynomial time ✓
  - valid solution ✓
  - factor 3/2

Longest processing time rule: 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 $t_1 \geq \ldots. \geq t_n$.
- If $n \leq m$ then optimal.
- Lower bound: If $n > m$ then $T^* \geq 2t_{m+1}$.
- Factor 3/2:
  - 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: factor 4/3

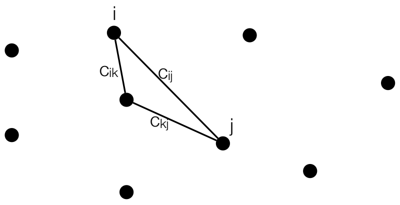


- **Longest processing time rule (LPT).** Sort jobs in non-increasing order. Assign next job on list to machine as soon as it becomes idle.
- Assume $t_1 \geq \ldots. \geq t_n$.
- Assume wlog that smallest job finishes last.
- If $t_n \leq T^*/3$ then $T \leq 4/3\ T^*$.
- If $t_n > T^*/3$ then each machine 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.*

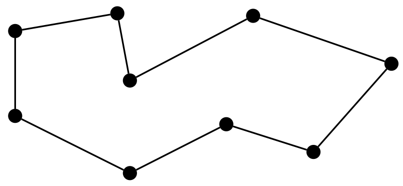
Traveling salesman problem

Traveling Salesman Problem (TSP)



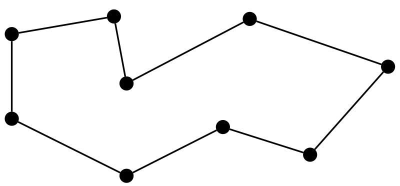
- Set of cities {1,…,n}
- $c_{ij} \geq 0$: cost of traveling from i to j.
- c_{ij} a metric:
  - $c_{ii} = 0$
  - $c_{ij} = c_{ji}$
  - $c_{ij} \leq c_{ik} + c_{kj}$ (triangle inequality)
- Goal: Find a *tour of minimum cost visiting every city exactly once*.

Traveling Salesman Problem (TSP)



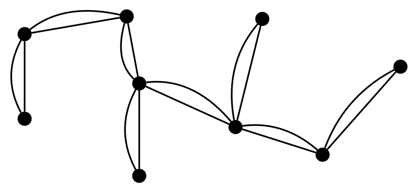
- Set of cities {1,…,n}
- $c_{ij} \geq 0$: cost of traveling from i to j.
- c_{ij} a metric:
  - $c_{ii} = 0$
  - $c_{ij} = c_{ji}$
  - $c_{ij} \leq c_{ik} + c_{kj}$
- Goal: Find a *tour of minimum cost visiting every city exactly once*.

Double tree algorithm



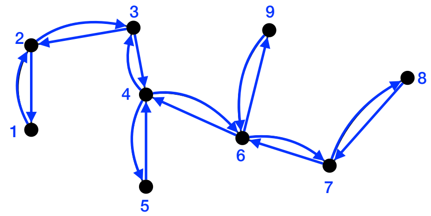
- MST is a lower bound on TSP.
  - Deleting an edge e from OPT gives a spanning tree.
  - OPT ≥ OPT - c_e ≥ MST.

Double tree algorithm



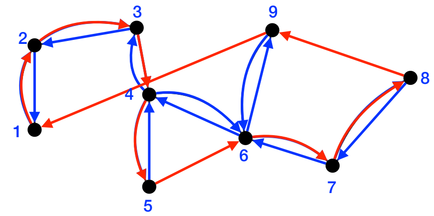
- Double tree algorithm
  - Compute MST T.
  - Double edges of T
  - Construct Euler tour τ (a tour visiting every edge exactly once).

Double tree algorithm



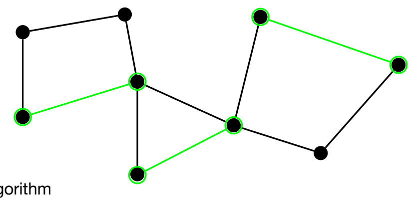
- Double tree algorithm
  - Compute MST T.
  - Double edges of T
  - Construct Euler tour τ (a tour visiting every edge exactly once).

Double tree algorithm



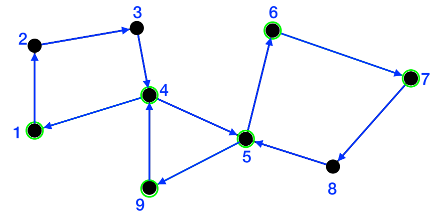
- Double tree algorithm
  - Compute MST T.
  - Double edges of T
  - Construct Euler tour τ (a tour visiting every edge exactly once).
  - Shortcut τ such that each vertex only visited once (τ')
- length(τ') ≤ length(τ) = 2 cost(T) ≤ 2 OPT.
- The double tree algorithm is a 2-approximation algorithm for TSP.

Christofides' algorithm



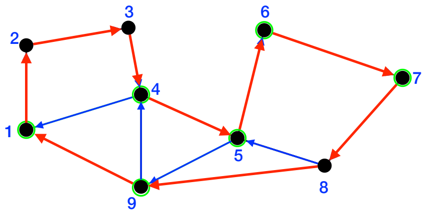
- Christofides' algorithm
  - Compute MST T.
  - No need to double all edges:
    - Enough to turn it into an Eulerian graph: *A graph Eulerian if there is a traversal of all edges visiting every edge exactly once.*
      - G Eulerian iff G connected and all nodes have even degree.
    - Consider set 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



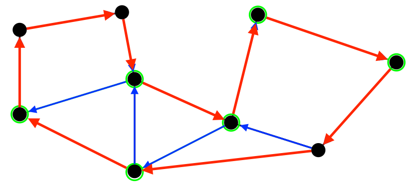
- Christofides' algorithm
  - Compute MST T.
  - O = {odd degree vertices in T}.
  - Compute minimum cost perfect matching M on O.
  - Construct Euler tour τ
  - Shortcut such that each vertex only visited once (τ')

Christofides' algorithm



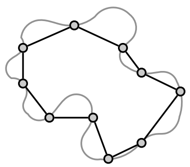
- Christofides' algorithm
  - Compute MST T.
  - O = {odd degree vertices in T}.
  - Compute minimum cost perfect matching M on O.
  - Construct Euler tour τ
  - Shortcut such that each vertex only visited once (τ')

Christofides' algorithm



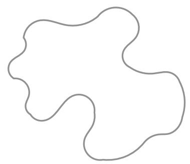
- Christofides' algorithm
  - Compute MST T.
  - O = {odd degree vertices in T}.
  - Compute minimum cost perfect matching **M** on O.
  - Construct Euler tour τ
  - Shortcut such that each vertex only visited once (τ')
- length(τ') $\leq$ length(τ) = cost(T) + cost(**M**) $\leq$ OPT + weight(M).

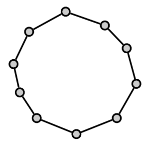
Analysis of Christofides' algorithm



- weight(M) $\leq$ OPT/2.
  - OPT_O = OPT restricted to O.
  - $\text{OPT}_O \leq$ OPT.

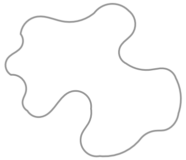
Analysis of Christofides' algorithm

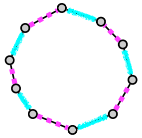




- weight(M) $\leq$ OPT/2.
  - OPT_O = OPT restricted to O.
  - $\text{OPT}_O \leq$ OPT.

Analysis of Christofides' algorithm





- weight(M) ≤ OPT/2:
  - OPT_o = OPT restricted to O.
  - $OPT_o \leq OPT$.
  - can partition OPT_o into two perfect matchings O_1 and O_2.
  - cost(M) ≤ min(cost(O_1), cost(O_2)) ≤ OPT/2.
- length(τ') ≤ length(τ) = cost(T) + cost(M) ≤ OPT + OPT/2 = 3/2 OPT.
- Christofides' algorithm is a 3/2-approximation algorithm for TSP.