Chapter 8
Approximation Algorithms

Algorithm Theory
WS 2019/20

Fabian Kuhn

Approximation Algorithms

- Optimization appears everywhere in computer science
- We have seen many examples, e.g.:
  - scheduling jobs
  - traveling salesperson
  - maximum flow, maximum matching
  - minimum spanning tree
  - minimum vertex cover
  - …
- Many discrete optimization problems are NP-hard
- They are however still important and we need to solve them
- As algorithm designers, we prefer algorithms that produce solutions which are provably good, even if we can't compute an optimal solution.

Approximation Algorithms: Examples

We have already seen two approximation algorithms

- **Metric TSP:** If distances are positive and satisfy the triangle inequality, the greedy tour is only by a log-factor longer than an optimal tour
- **Maximum Matching and Vertex Cover:** A maximal matching gives solutions that are within a factor of 2 for both problems.

Approximation Ratio

An approximation algorithm is an algorithm that computes a solution for an optimization with an objective value that is provably within a bounded factor of the optimal objective value.

Formally:

- $\mathrm{OPT} \geq 0$: optimal objective value
  $\mathrm{ALG} \geq 0$: objective value achieved by the algorithm

- **Approximation Ratio α:**

$$\textbf{Minimization: } \alpha := \max_{\text{input instances}} \frac{\mathrm{ALG}}{\mathrm{OPT}} \quad \geq 1$$

$$\textbf{Maximization: } \alpha := \min_{\text{input instances}} \frac{\mathrm{ALG}}{\mathrm{OPT}} \quad \leq 1$$

Example: Load Balancing

We are given:

- m machines $M_1, \dots, M_m$
- n jobs, processing time of job i is t_i

Goal:

- Assign each job to a machine such that the makespan is minimized

  makespan: largest total processing time of any machine

The above load balancing problem is NP-hard and we therefore want to get a good approximation for the problem.

Greedy Algorithm

There is a simple greedy algorithm:

- Go through the jobs in an arbitrary order
- When considering job i, assign the job to the machine that currently has the smallest load.

Example: 3 machines, 12 jobs

3	4	2	3	1	6	4	4	3	2	1	5

Greedy Assignment:

M_1:	3	1	6	1	5
M_2:	4	4	3		
M_3:	2	3	4	2	

makespan = 16

Optimal Assignment:

M_1:	3	4	2	3	1
M_2:	6	4	3		
M_3:	4	2	1	5	

makespan = 13

Greedy Analysis

- We will show that greedy gives a 2-approximation

- To show this, we need to compare the solution of greedy with an optimal solution (that we can't compute)

- Lower bound on the optimal makespan T^*:

$$T^* \geq \frac{1}{m} \cdot \sum_{i=1}^{n} t_i$$

- Lower bound can be far from T^*:
  - m machines, m jobs of size 1, 1 job of size m

$$T^* = m, \qquad \frac{1}{m} \cdot \sum_{i=1}^{n} t_i = 2$$

Greedy Analysis

- We will show that greedy gives a 2-approximation

- To show this, we need to compare the solution of greedy with an optimal solution (that we can't compute)

- Lower bound on the optimal makespan T^*:

$$T^* \geq \frac{1}{m} \cdot \sum_{i=1}^{n} t_i$$

- Second lower bound on optimal makespan T^*:

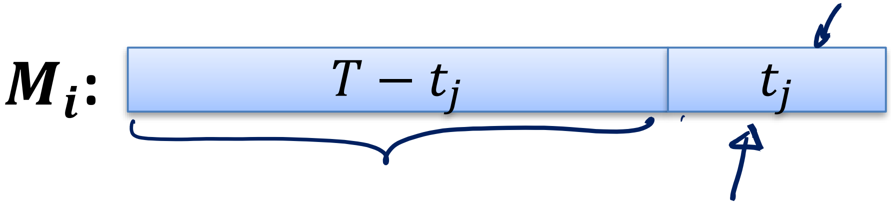
$$T^* \geq \max_{1 \leq i \leq n} t_i$$

Greedy Analysis

Theorem: The greedy algorithm has approximation ratio ≤ 2, i.e., for the makespan T of the greedy solution, we have $T \leq 2T^*$.

Proof:

- For machine k, let T_k be the time used by machine k
- Consider some machine M_i for which $T_i = T$
- Assume that job j is the last one schedule on M_i:



- When job j is scheduled, M_i has the minimum load

$\forall k: \; T_k \geq T - t_j \quad \Longrightarrow \quad \sum_{s=1}^{n} t_s \geq m \cdot (T - t_j)$

avg. load $\geq T - t_j$

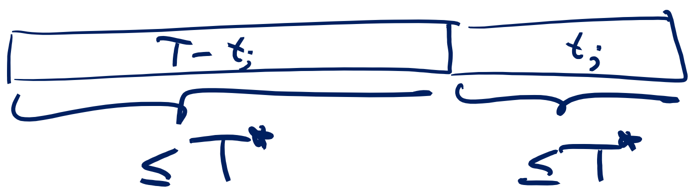
Greedy Analysis

Theorem: The greedy algorithm has approximation ratio ≤ 2, i.e., for the makespan T of the greedy solution, we have $T \leq 2T^*$.

Proof:

- For all machines M_k: load $T_k \geq T - t_j$

$$T^* \geq \frac{1}{m}\sum_{s=1}^{n} t_s \geq T - t_j$$



$$T = T - t_j + t_j \leq 2 \cdot T^*$$

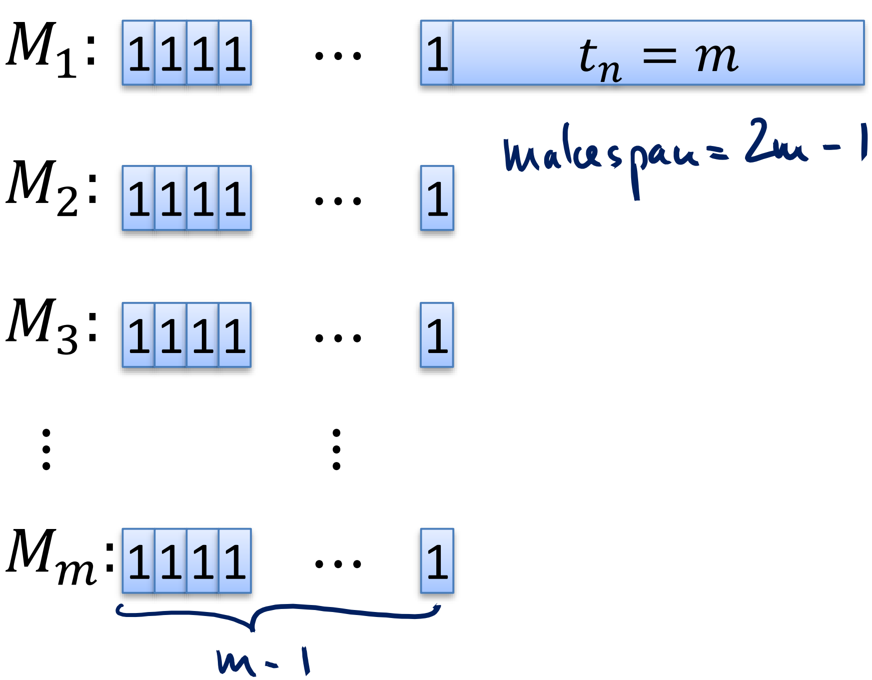
□

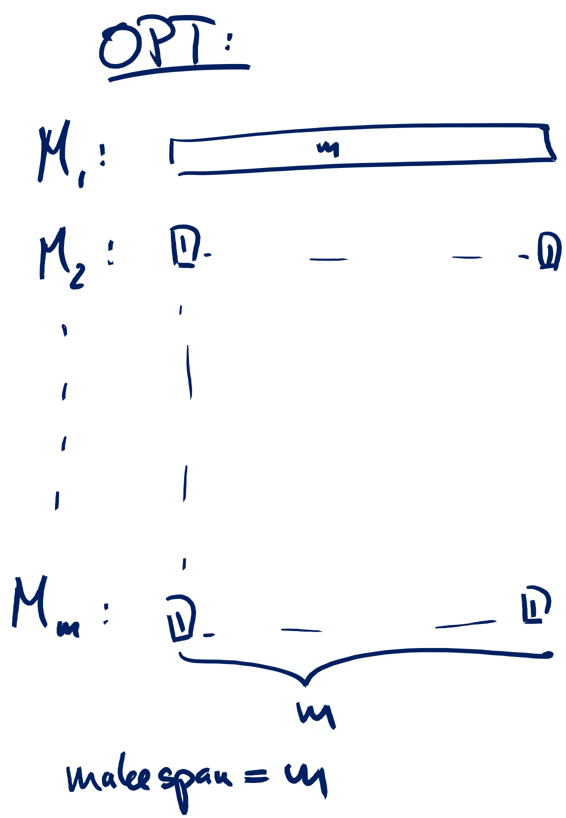
Can We Do Better?

The analysis of the greedy algorithm is almost tight:

- Example with $n = m(m-1) + 1$ jobs
- Jobs $1, \dots, n-1 = m(m-1)$ have $t_i = 1$, job n has $t_n = m$

Greedy Schedule:





Improving Greedy

Bad case for the greedy algorithm:
One large job in the end can destroy everything

Idea: assign large jobs first

if $n \leq m$ → optimal

Modified Greedy Algorithm:

1. Sort jobs by decreasing length s.t. $t_1 \geq t_2 \geq \cdots \geq t_n$
2. Apply the greedy algorithm as before (in the sorted order)

Lemma: If $n > m$: $T^* \geq t_m + t_{m+1} \geq 2t_{m+1}$

Proof:

- Two of the first $m + 1$ jobs need to be scheduled on the same machine
- Jobs m and $m + 1$ are the shortest of these jobs

Analysis of the Modified Greedy Alg.

Theorem: The modified algorithm has approximation ratio $\leq {}^{3}/_{2}$.

Proof:

- We show that $T \leq {}^{3}/_{2} \cdot T^*$
- As before, we consider the machine M_i with $T_i = T$
- Job j (of length t_j) is the last one scheduled on machine M_i
- If j is the only job on M_i, we have $T = T^*$
- Otherwise, we have $j \geq m + 1$
  - The first m jobs are assigned to m distinct machines