Chapter 7
Approximation Algorithms

Algorithm Theory
WS 2013/14

Fabian Kuhn

Approximation Algorithms

- Optimization appears everywhere in computer science
- We have seen many examples, e.g.:
  - scheduling jobs
  - traveling salesperson (minimum)
  - 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

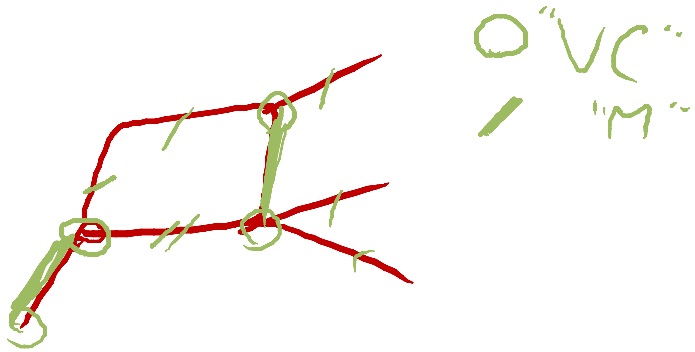
- **Metric TSP:** If distances are positive and satisfy the triangle inequality, the greedy tour is only by a log-factor longer than an optimal tour

M^* ≥ A NP-hard VC = M

- **Maximum Matching and Vertex Cover:** A maximal matching gives solutions that are within a factor of 2 for both problems.

$$\Big|\bigcup_{e \in M} e\Big| \le 2|VC| \qquad e=\{u,v\}$$

$$|M| \ge \frac{1}{2}|M^*|$$



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:

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

- **Approximation Ratio α:**

$$\textbf{Minimization}:\ \alpha := \max_{\text{input instances}} \frac{\text{ALG}}{\text{OPT}} = \max \frac{\overset{\geq MIN}{ALG}}{MIN} \geq 1$$

$$\textbf{Maximization}:\ \alpha := \max_{\text{input instances}} \frac{\text{OPT}}{\text{ALG}} = \max \frac{MAX}{\underset{\leq MAX}{ALG}} \geq 1$$

Example: Load Balancing

We are given:

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

$n \gg m$ usually

Goal:

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

makespan: largest total processing time of any machine

"earliest time at which we are done with everything"

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.

$m = 3$

Example: 3 machines, 12 jobs $n = 12$

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

Greedy Assignment:

$\overbrace{}^{16}$

$\boldsymbol{M_1}$: 3 | 1 | 6 | 1 | 5

$\boldsymbol{M_2}$: 4 | 4 | 3

$\boldsymbol{M_3}$: 2 | 3 | 4 | 2

Optimal Assignment:

$\boldsymbol{M_1}$: 3 | 4 | 2 | 3 | 1 $\boxed{13}$

average: $12\frac{1}{3}$

$\boldsymbol{M_2}$: 6 | 4 | 3 13

$\boldsymbol{M_3}$: 4 | 2 | 1 | 5 12

$16 \leq 2 \cdot 13$

$\frac{ALG}{OPT} \leq 2$

Greedy Analysis

- We will show that greedy gives a $\boxed{2}$-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^*:

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

average load on each machine

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

$n = m + 1$

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

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^*:

$$T^* \boxed{\geq} \frac{1}{m} \cdot \sum_{i=1}^{n} t_i \quad =: \overline{T}_1$$

- Second lower bound on optimal makespan T^*:

$$T^* \boxed{\geq} \max_{1 \leq i \leq n} t_i \quad =: T_2$$

$$T^* \geq \max\{\overline{T}_1, \overline{T}_2\}$$

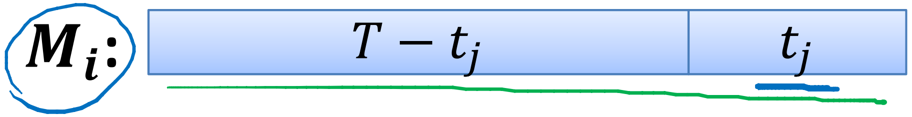
$$\boxed{\overset{\text{algo:}}{T} \leq 2T^*}$$

Greedy Analysis

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

Proof:

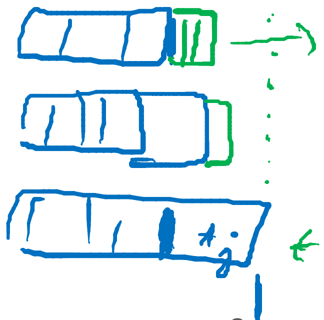
- For machine k, let $\boxed{T_k}$ be the time used by machine k
- Consider some machine M_i for which $\boxed{T_i = T}$ ← "troublemaker"
- Assume that job $\boxed{j}$ is the last one schedule on M_i:



$T = (T - t_j) + t_j$

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

$\forall k: \boxed{T_k \geq T - t_j}$



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 $T_k \geq T - t_j$

$$T_1 = \frac{1}{m} \cdot \sum_{i=1}^{n} t_i$$

$$m \cdot (T - t_j) = \sum_{k=1}^{m} (T - t_j) \leq \sum_{k=1}^{m} T_k = \sum_{i=1}^{n} t_i = m \cdot T_1 \leq m \cdot T^*$$

$$\Rightarrow T - t_j \leq T^*$$

$$t_j \leq T_2 \leq T^*$$

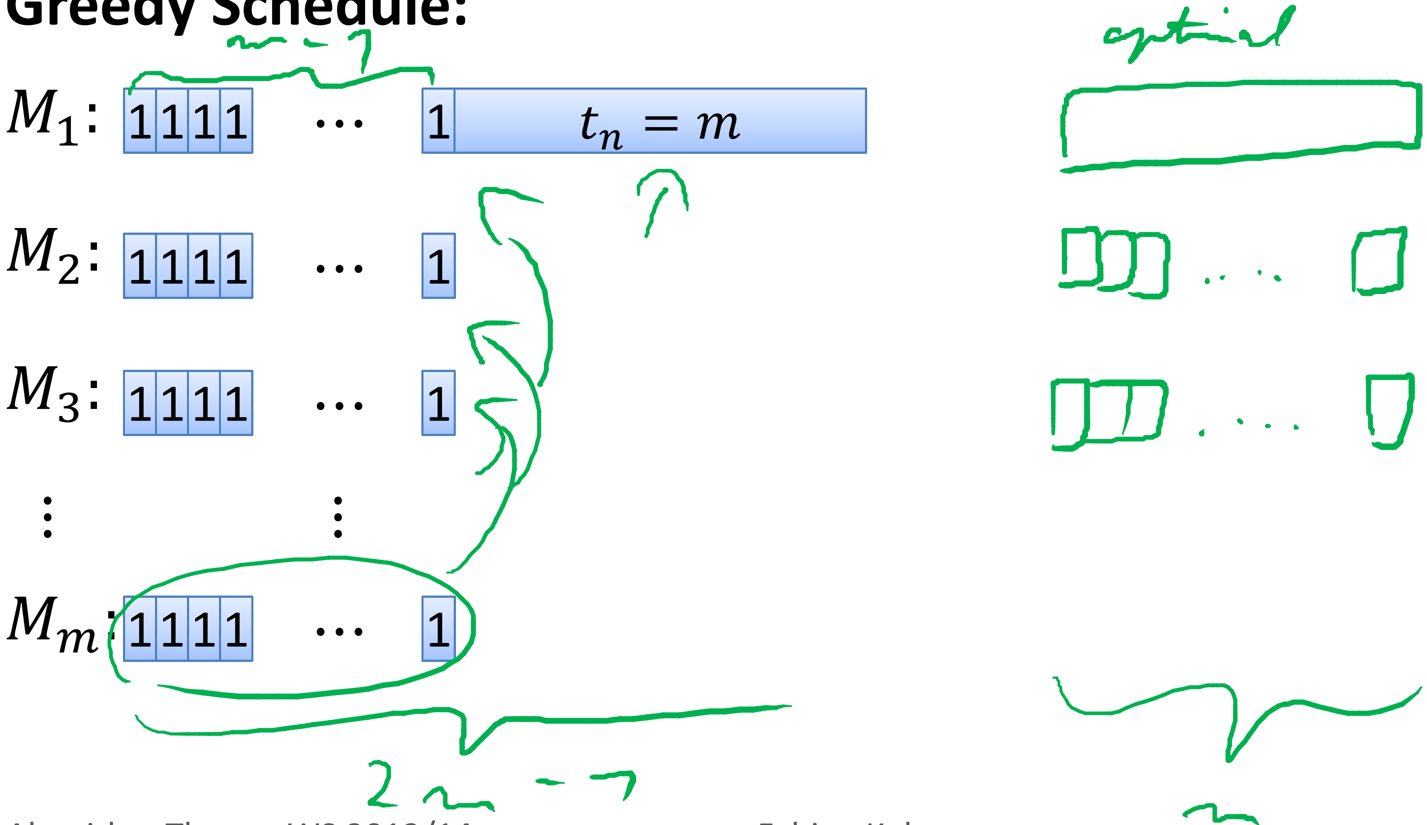
$$T = \underline{(T - t_j)} + \underline{t_j} \leq 2T^*$$

Can We Do Better?

The analysis of the greedy algorithm is almost tight:

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

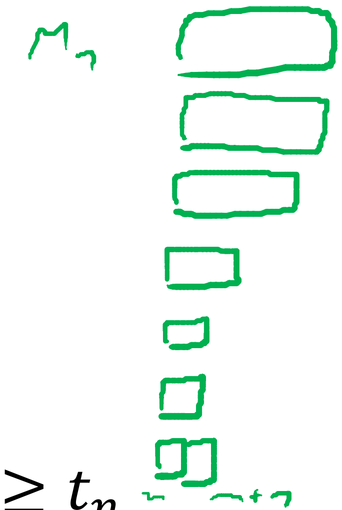
Greedy Schedule:



Improving Greedy

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

Idea: assign large jobs first



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

$\geq t_{m+1}$

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 $\leq {}^{3}/_{2}$, i.e., we have $T \leq {}^{3}/_{2} \cdot T^*$.

Proof:

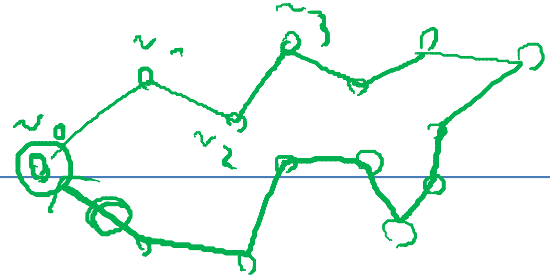
- As before, choose machine M_i with $T_i = T$ trouble maker
- Job t_j is the last one scheduled on machine M_i
- If there is only one job t_j on M_i, we have $T_i = t_j = T^* \geq T_2 = \max_j t_j$
- Otherwise, we have $j \geq m + 1$
  - The first m jobs are assigned to m distinct machines

$j \geq m+1$ $\quad\quad t_j \leq t_{m+1} \leq \frac{T^*}{2}$

$$T = \underbrace{(T - t_j)}_{\leq T_1 \leq T^*} + t_j \leq T^* + \frac{T^*}{2} \leq \frac{3}{2} T^*$$

$\blacksquare$

Metric TSP



Input:

- Set V of n nodes (points, cities, locations, sites)
- Distance function $d: V \times V \to \mathbb{R}$, i.e., $d(u,v)$: dist. from u to v
- Distances define a metric on V:

$$d(u,v) = d(v,u) \geq 0, \qquad d(u,v) = 0 \Leftrightarrow u = v$$

$$\boxed{d(u,v) \leq d(u,w) + d(v,w)}$$

triangle inequality

Solution:

- Ordering/permutation $v_1, v_2, \ldots, v_n$ of vertices
- Length of TSP path: $\sum_{i=1}^{n-1} d(v_i, v_{i+1})$
- Length of TSP tour: $d(v_n, v_1) + \sum_{i=1}^{n-1} d(v_i, v_{i+1})$

Goal:

- Minimize length of TSP path or TSP tour

Metric TSP

- The problem is NP-hard
- We have seen that the greedy algorithm (always going to the nearest unvisited node) gives an $O(\log n)$-approximation
- Can we get a constant approximation ratio?
- We will see that we can…

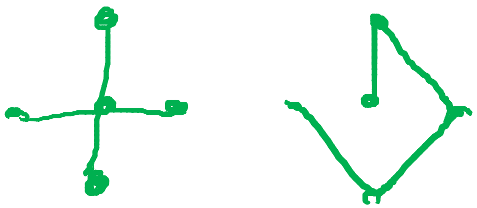
TSP and MST

Claim: The length of an optimal TSP path is lower bounded by the weight of a minimum spanning tree

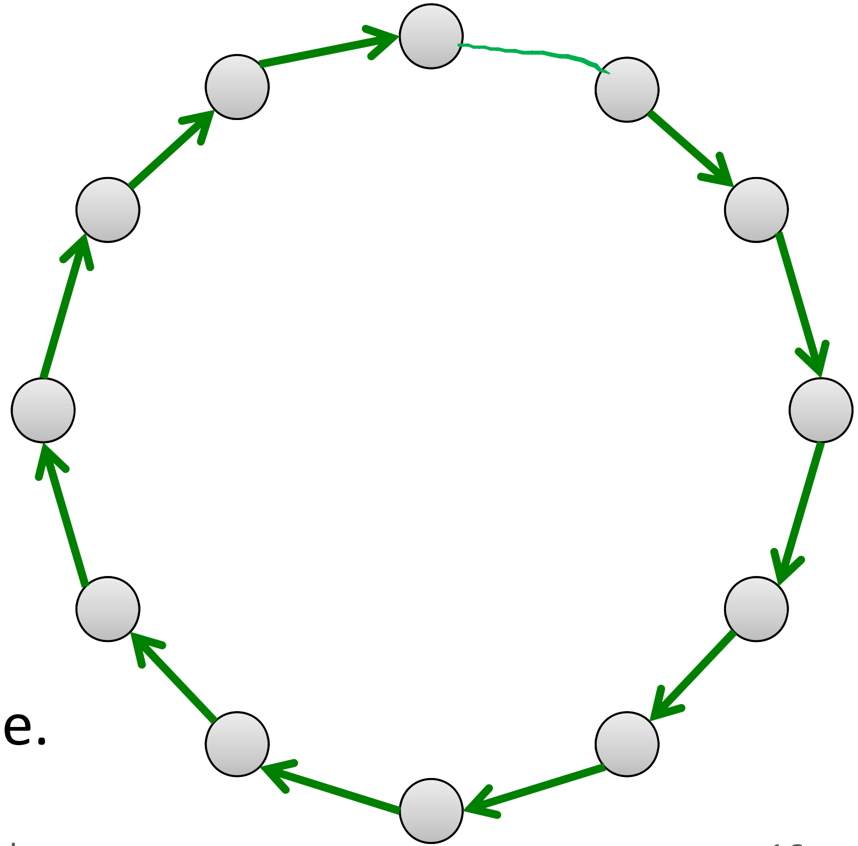
Proof:

- A TSP path is a spanning tree, it's length is the weight of the tree

⇒ "MST" ≤ "TSP path" ≤ "TSP tour"



Corollary: Since an optimal TSP tour is longer than an optimal TSP path, the length of an optimal TSP tour is also lower bounded by the weight of a minimum spanning tree.

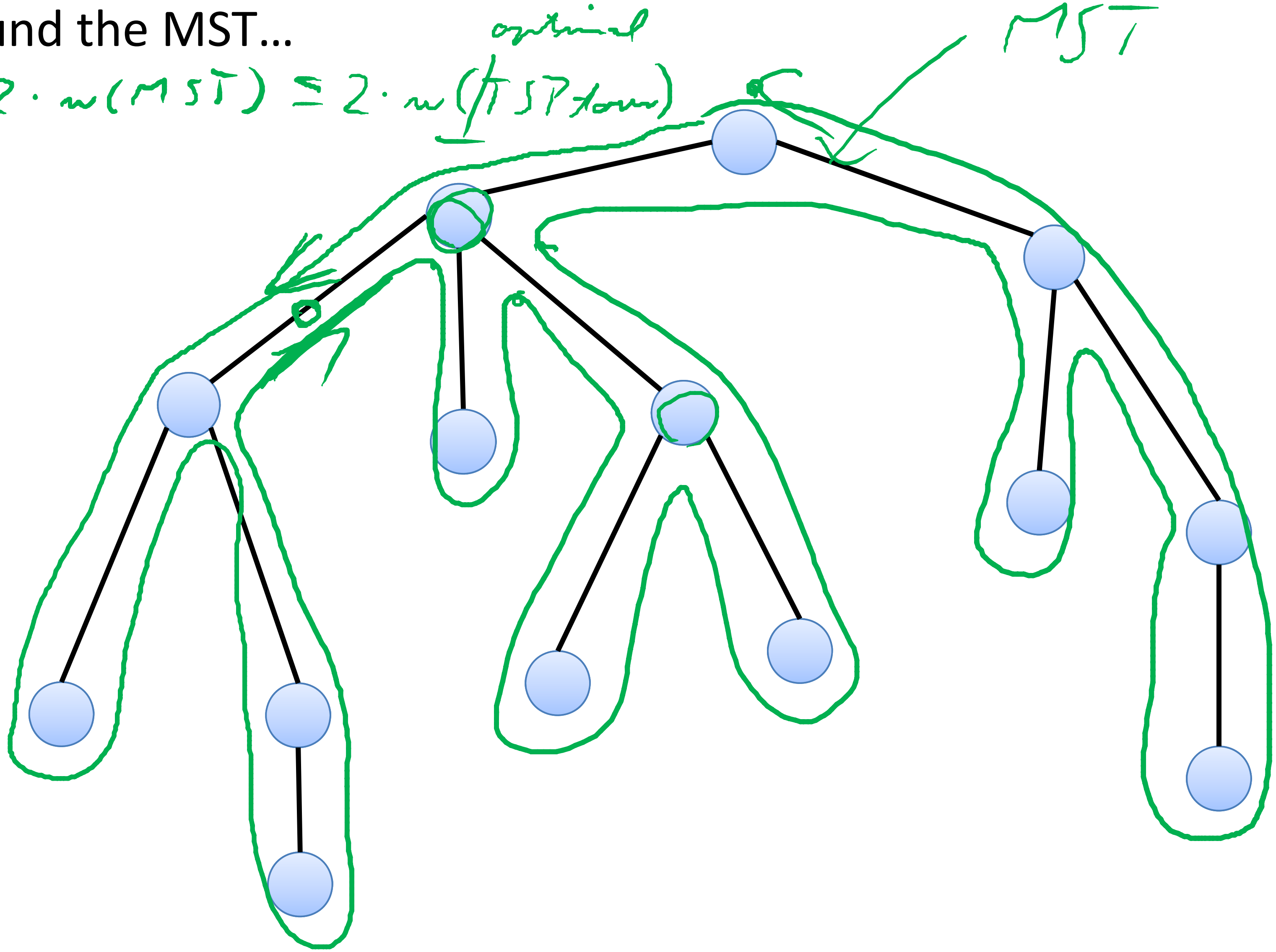


The MST Tour

Walk around the MST…

optimal

walk: $= 2 \cdot w(MST) \leq 2 \cdot w(\text{TSP tour})$

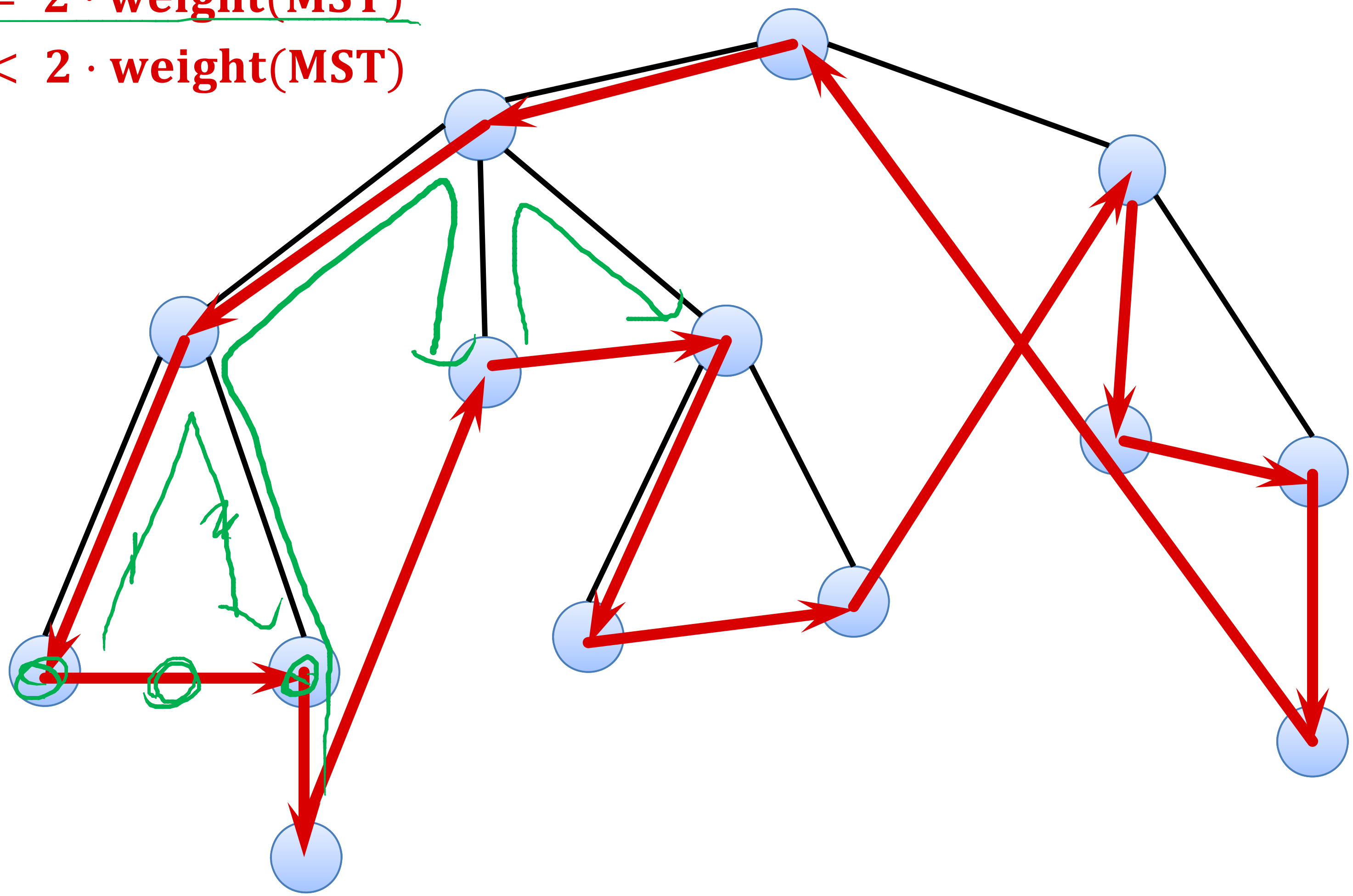


The MST Tour

Walk around the MST…

$\textbf{Cost (walk)} = \mathbf{2} \cdot \mathbf{weight(MST)}$

$\textbf{Cost (tour)} < \mathbf{2} \cdot \mathbf{weight(MST)}$



Approximation Ratio of MST Tour

Theorem: The MST TSP tour gives a 2-approximation for the metric TSP problem.

Proof:

- Triangle inequality → length of tour is at most $2 \cdot \text{weight}(\text{MST})$
- We have seen that $\text{weight}(\text{MST}) < \text{opt. tour length}$

Can we do even better?

1 2 3 $\Rightarrow \alpha = 3/2$

1 Metric TSP Subproblems

Claim: Given a metric (V, d) and (V', d) for $V' \subseteq V$, the optimal TSP path/tour of (V', d) is at most as large as the optimal TSP path/tour of (V, d).

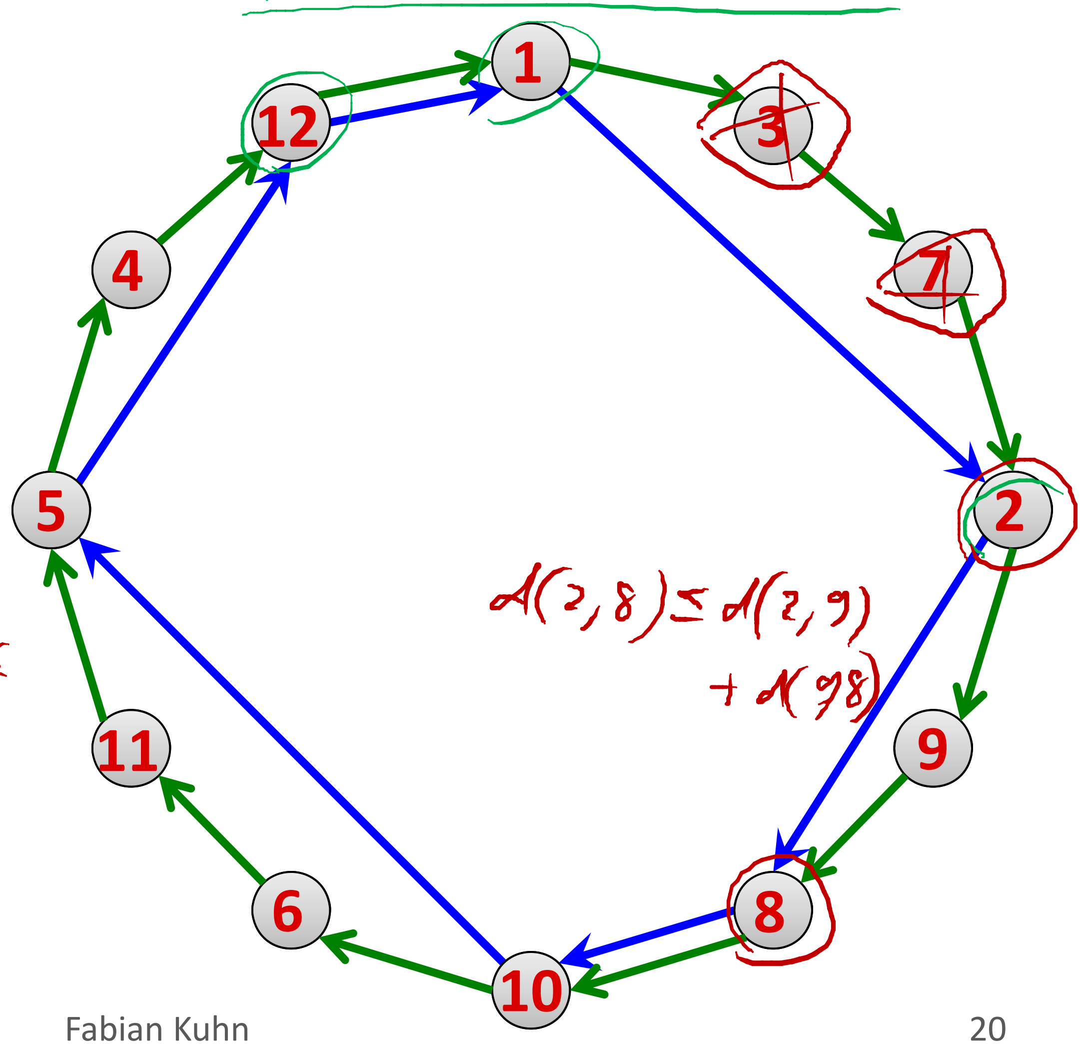
Optimal TSP tour of nodes $1, 2, \ldots, 12$

Induced TSP tour for nodes $1, 2, 5, 8, 10, 12$

optimal tour on $(1, 2, 5, 8, 10, 12) \leq$

blue tour $\leq$ green tour

not necessarily



2 TSP and Matching

- Consider a metric TSP instance (V, d) with an $\boxed{\text{even}}$ number of nodes $|V|$
- Recall that a perfect matching is a matching $M \subseteq V \times V$ such that every node of V is incident to an edge of M.
- Because $|V|$ is even and because in a metric TSP, there is an edge between any two nodes $u, v \in V$, any partition of V into $|V|/2$ pairs is a perfect matching.
- The weight of a matching M is the sum of the distances represented by all edges in M:

$$w(M) = \sum_{\{u,v\} \in M} d(u, v)$$

TSP and Matching

Lemma: Assume we are given a TSP instance (V, d) with an even number of nodes. The length of an optimal TSP tour of (V, d) is at least twice the weight of a minimum weight perfect matching of (V, d). $=: W$

$\boxed{\text{opt TSP} \geq 2W}$

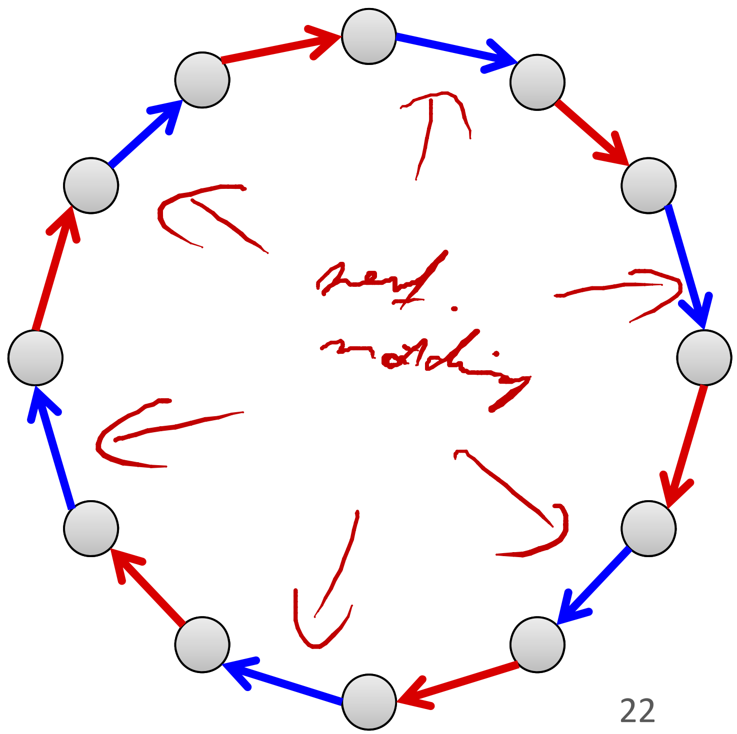
Proof:

- The edges of a TSP tour can be partitioned into 2 perfect matchings

$W \leq$ blue edges

$W \leq$ red edges

$2W \leq$ red & blue edges $=$ opt. TSP tour



2 Minimum Weight Perfect Matching

Claim: If $|V|$ is even, a minimum weight perfect matching of (V, d) can be computed in polynomial time

Proof Sketch:

- We have seen that a maximum matching in an unweighted graph can be computed in polynomial time
- With a more complicated algorithm, also a maximum weighted matching can be computed in polynomial time ([illegible])
- In a complete graph, a maximum weighted matching is also a (maximum weight) perfect matching   weights ≥ 0
- gives us graph G'
  Define weight $w(u,v) := D - d(u,v)$ (also needed: $|V|$ even)
  $D = \max d(u,v)$
- A maximum weight perfect matching for (V, w) is a minimum weight perfect matching for (V, d) $W_G^{min} = \frac{|V|}{2} \cdot D - W_{G'}^{max}$

$\square$

3 Algorithm Outline

$\overbrace{\text{MST}}^{G'}$

Problem of MST algorithm:

- Every edge has to be visited twice (before taking shortcuts)

Goal:

$MST \subset G''$

- Get a $\overbrace{\text{graph}}^{G''}$ on which every edge only has to be visited once (and where still the total edge weight is small compared to an optimal TSP tour)

unlike G', G'' is not ~~(necessarily)~~ a tree

$\Rightarrow$ #edges $> n-1$

Euler Tours:

- A tour that visits each edge of a graph exactly once is called an Euler tour
- An Euler tour in a (multi-)graph exists if and only if every node of the graph has even degree

- That's definitely not true for a tree, but can we modify our MST suitably?

Euler Tour

Theorem: A connected (multi-)graph G has an Euler tour if and only if every node of G has even degree.

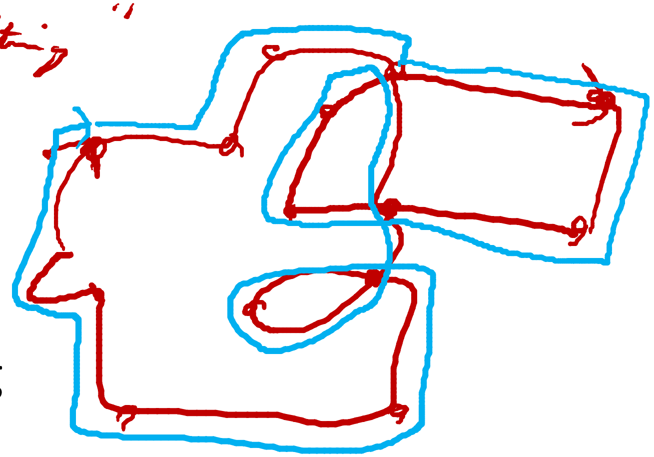
"⟵"

Proof:

- If G has an odd degree node, it clearly cannot have an Euler tour
- If G has only even degree nodes, a tour can be found recursively:

"# times entering = # times exiting"

1. Start at some node
2. As long as possible, follow an unvisited edge
   - Gives a partial tour, the remaining graph still has even degree



3. Solve problem on remaining components recursively
4. Merge the obtained tours into one tour that visits all edges

TSP Algorithm

1 + 2 + 3

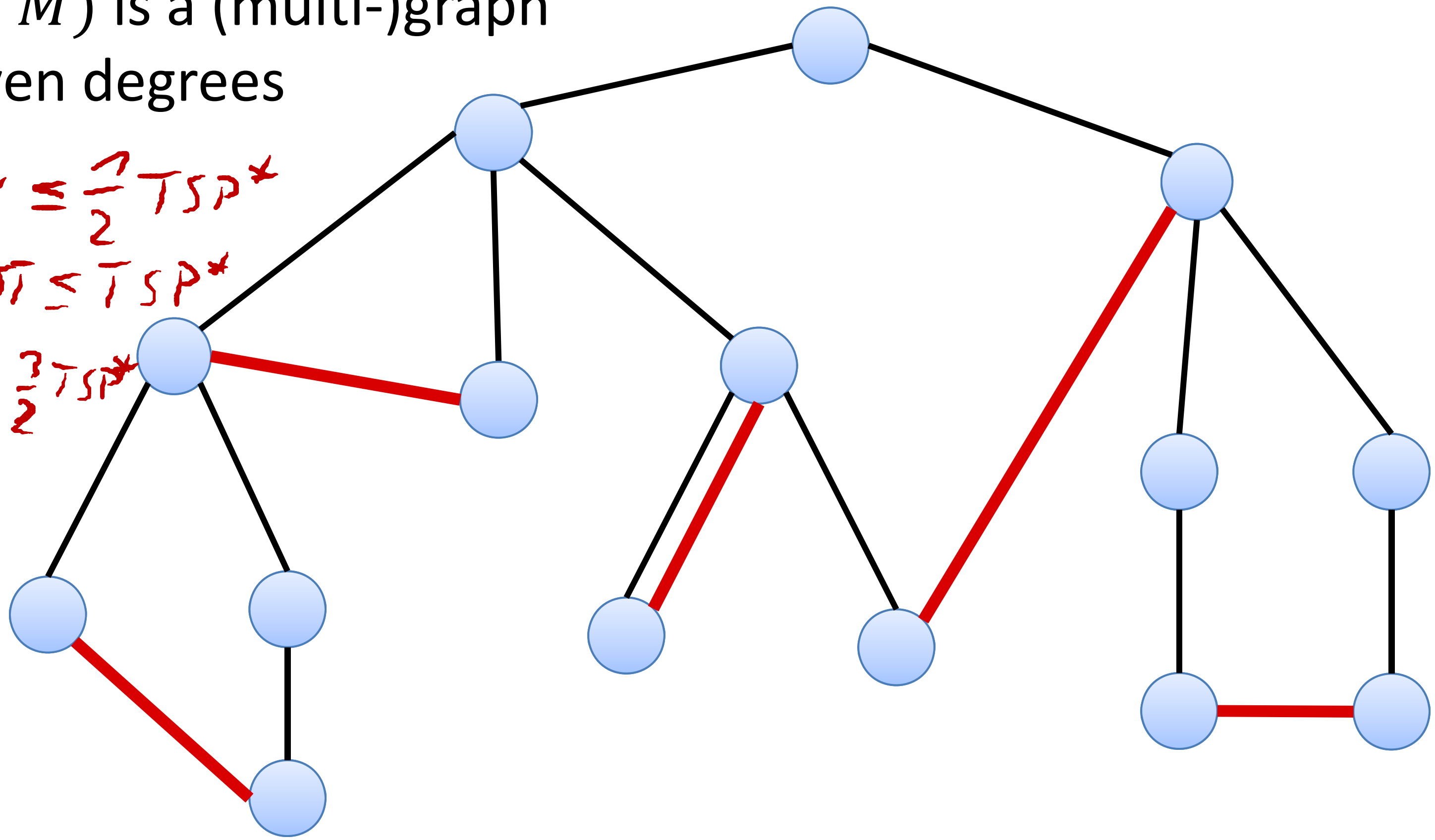
$\sum_{v \in V} \deg(v) = \boxed{2 \cdot |E|}$

1. Compute MST T
2. V_{odd}: nodes that have an odd degree in T ($|V_{\text{odd}}|$ is even)
3. Compute min weight perfect matching M of (V_{odd}, d)
4. $(V, T \cup M)$ is a (multi-)graph with even degrees

red edges: $W \leq \frac{1}{2} TSP^*$

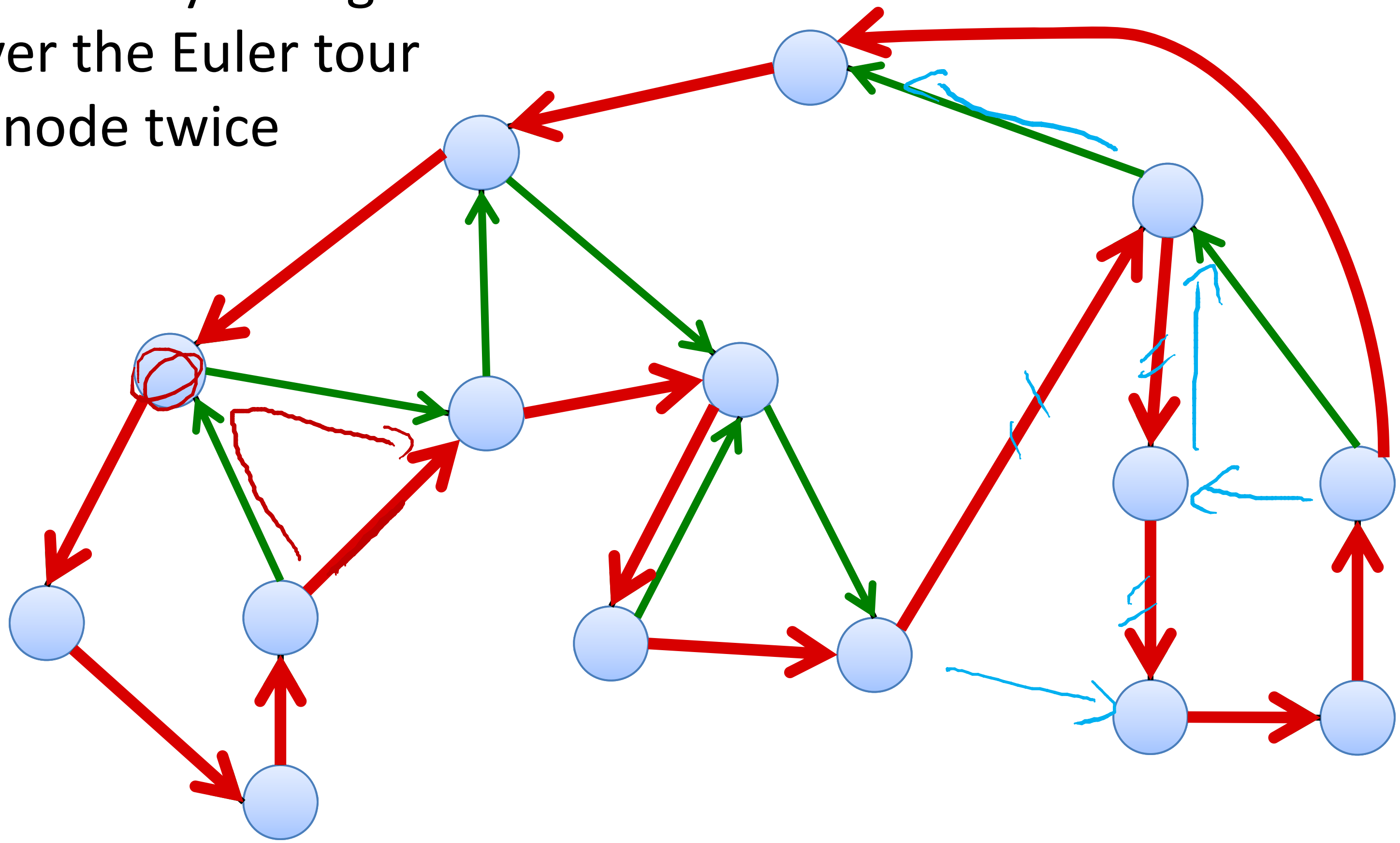
black edges: $MST \leq TSP^*$

black & red: $\leq \frac{3}{2} TSP^*$



TSP Algorithm

5. Compute Euler tour on $(V, T \cup M)$
6. Total length of Euler tour $\boxed{\leq \frac{3}{2} \cdot \mathbf{TSP_{OPT}}}$
7. Get TSP tour by taking shortcuts wherever the Euler tour visits a node twice



TSP Algorithm

- The described algorithm is by Christofides

Theorem: The Christofides algorithm achieves an approximation ratio of at most ${}^{3}/_{2}$.

Proof:

- The length of the Euler tour is $\leq {}^{3}/_{2} \cdot \mathrm{TSP}_{\mathrm{OPT}}$
- Because of the triangle inequality, taking shortcuts can only make the tour shorter