Chapter 8
Approximation Algorithms

Algorithm Theory
WS 2018/19

Fabian Kuhn

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:**

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

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

Metric TSP

Input:

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

$$d(u, v) = d(v, u) \geq 0, \qquad d(u, v) = 0 \Leftrightarrow u = v$$
$$\forall u, v, w \in V : d(u, v) \leq d(u, w) + d(w, v)$$

Solution:

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

Goal:

- Minimize length of TSP path or TSP tour

Metric TSP

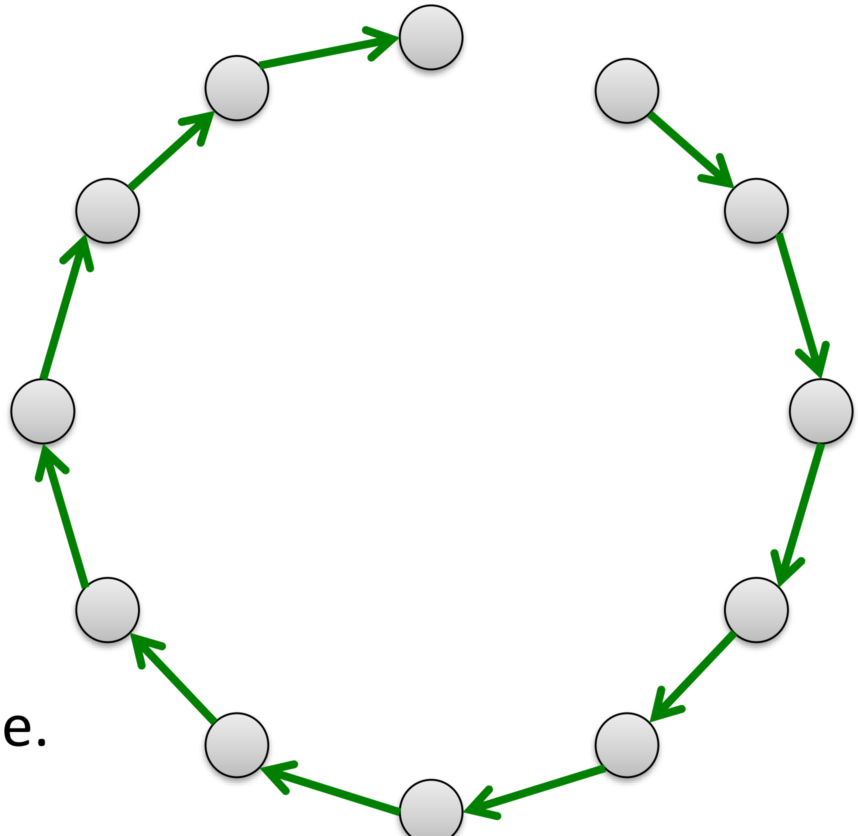
- The problem is NP-hard
- We have seen that the greedy algorithm (always going to the nearest unvisited node) gives an $O(\log n)$-approximation
- Can we get a constant approximation ratio?
- We will see that we can…

TSP and MST

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

Proof:

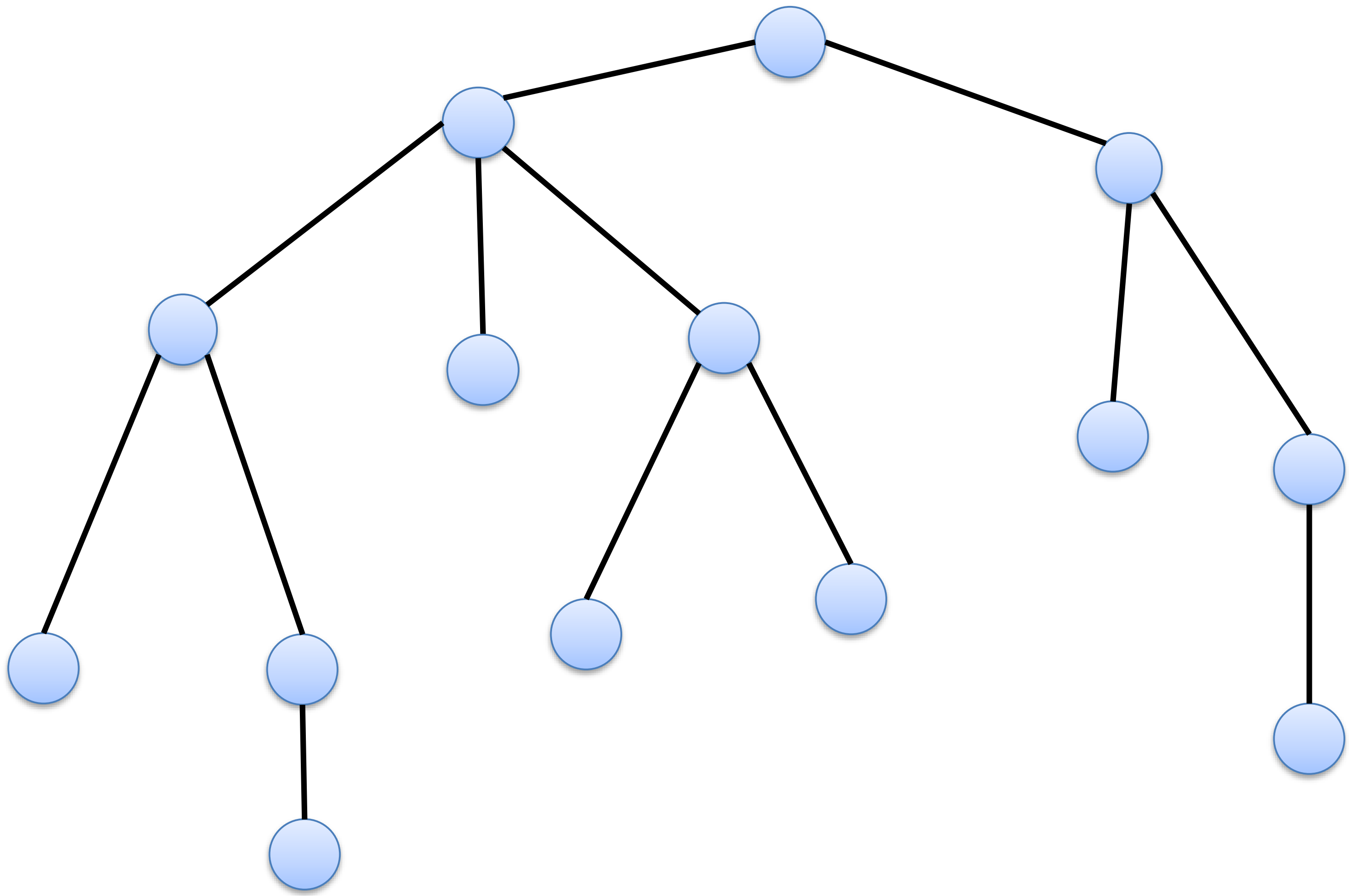
- A TSP path is a spanning tree, it's length is the weight of the tree



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…

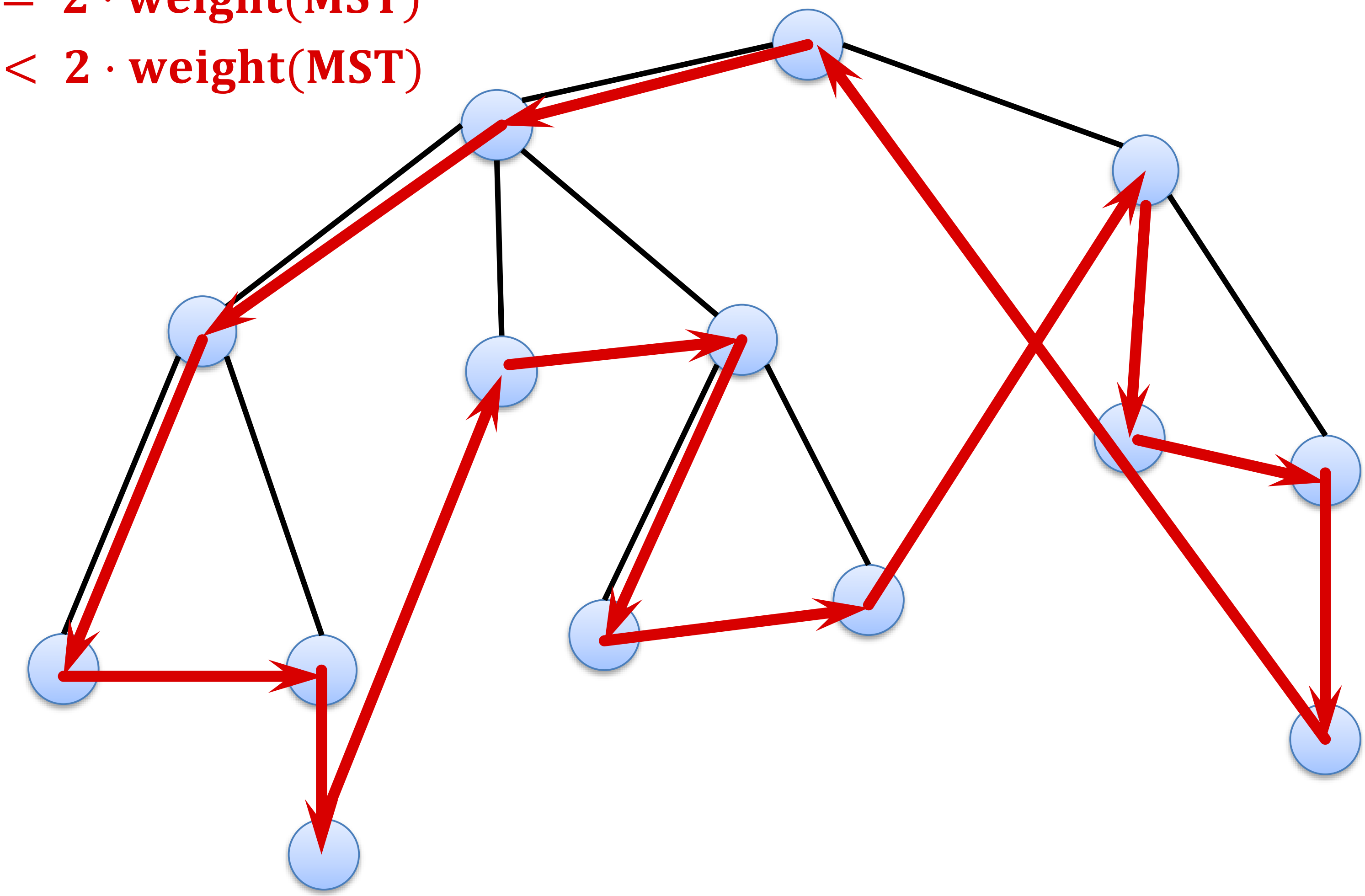


The MST Tour

Walk around the MST…

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

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



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?

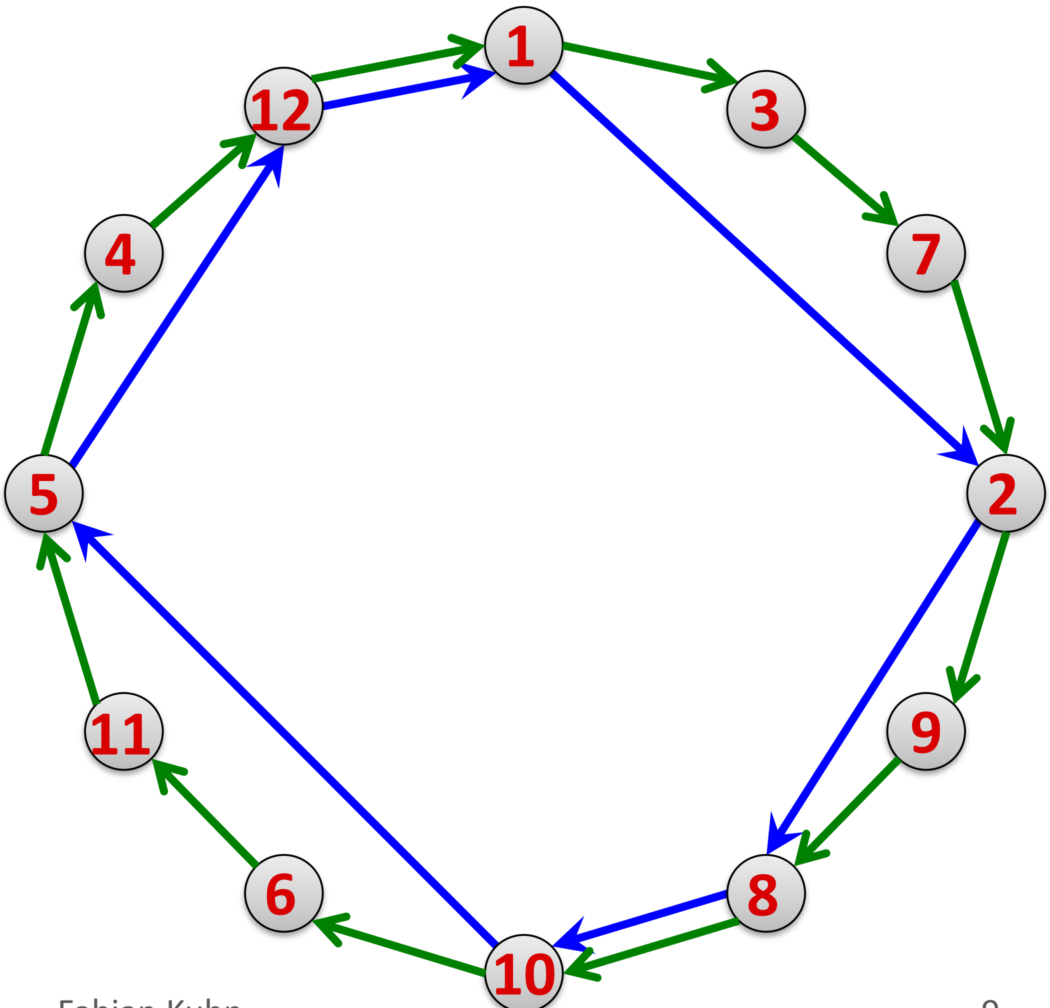
Metric TSP Subproblems

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 $1, 2, \dots, 12$

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

blue tour $\leq$ green tour



TSP and Matching

- Consider a metric TSP instance (V, d) with an 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:

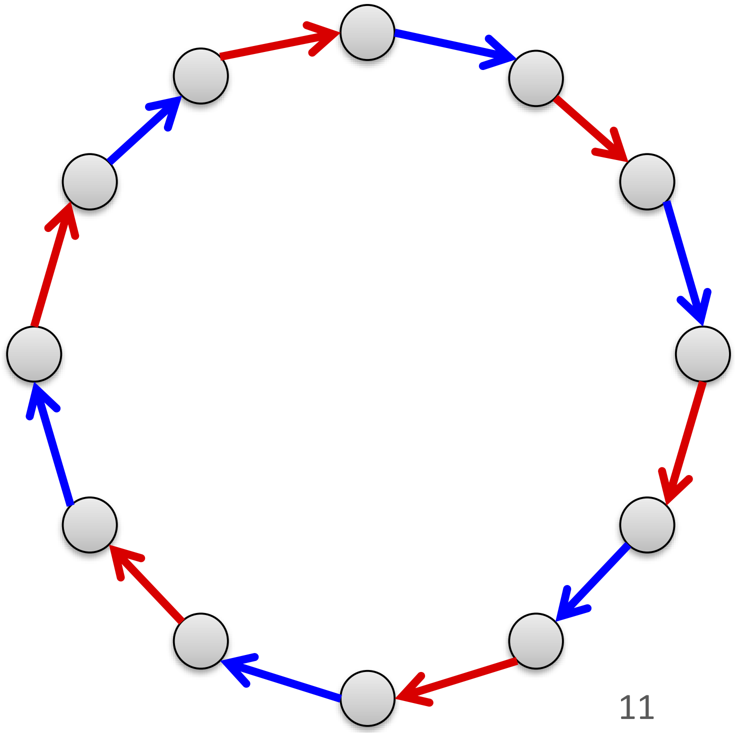
$$w(M) = \sum_{\{u,v\} \in M} d(u, v)$$

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).

Proof:

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



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 minimum weight perfect matching in a complete bipartite graph can be computed in polynomial time
- With a more complicated algorithm, also a minimum weight perfect matching in complete (non-bipartite) graphs can be computed in polynomial time
- The algorithm uses similar ideas as the bipartite weighted matching algorithm and it uses the Blossom algorithm as a subroutine

Algorithm Outline

Problem of MST algorithm:

- Every edge has to be visited twice

Goal:

- Get a graph 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)

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.

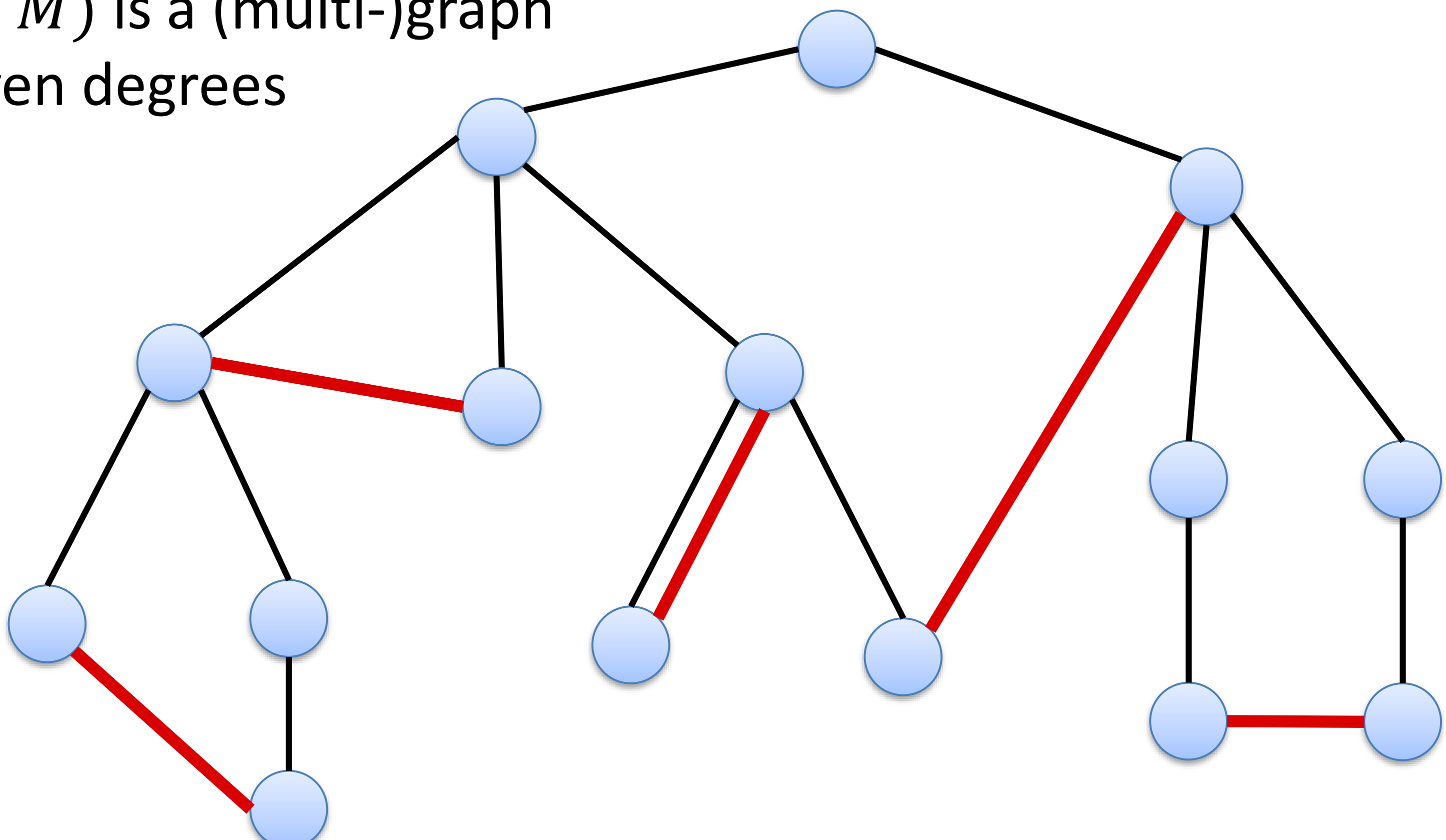
Proof:

- 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:

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. 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

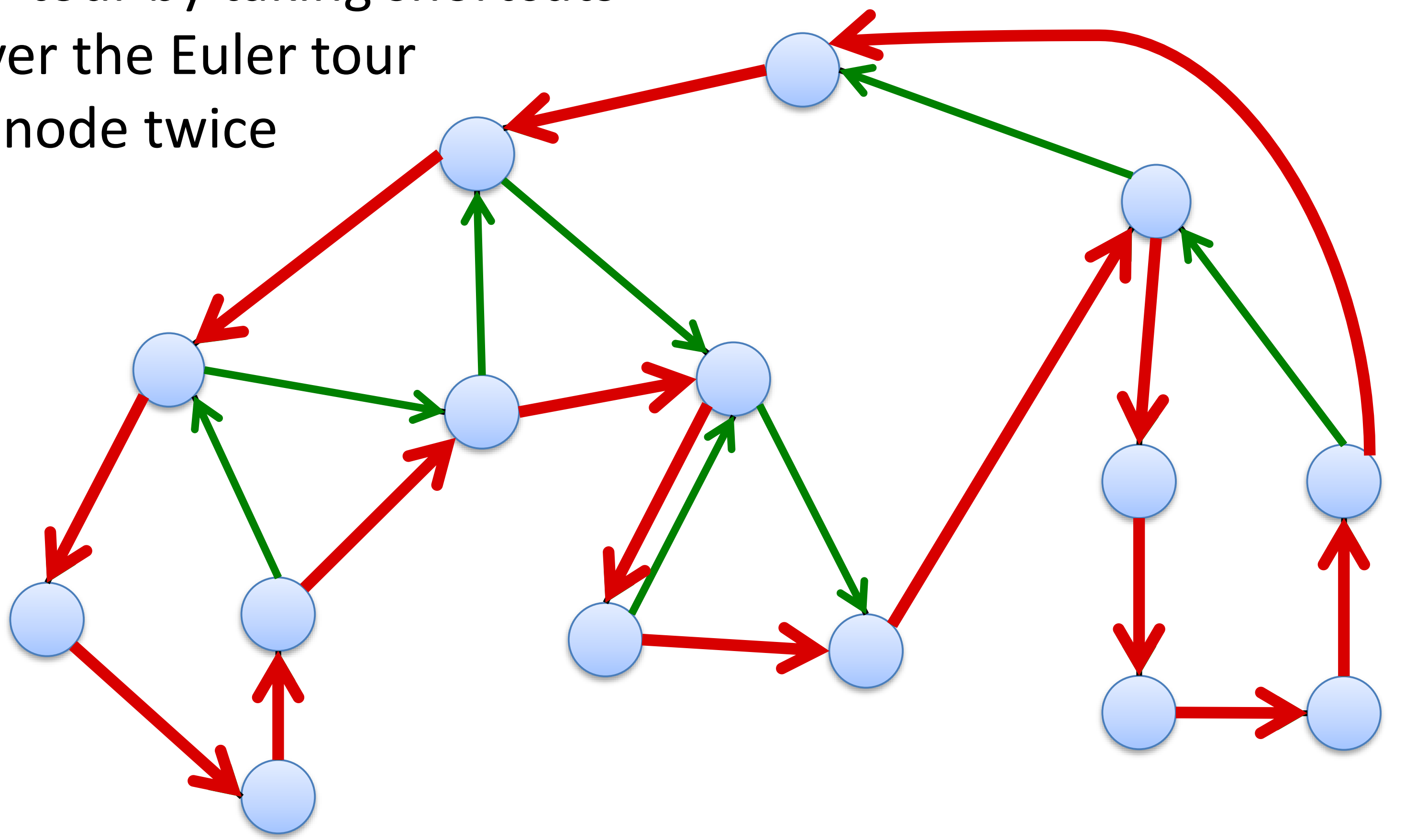


TSP Algorithm

5. Compute Euler tour on $(V, T \cup M)$

6. Total length of Euler tour $\leq \frac{3}{2} \cdot \mathbf{TSP_{OPT}}$

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

Knapsack

- n items $1, \dots, n$, each item has weight $w_i > 0$ and value $v_i > 0$
- Knapsack (bag) of capacity W

- Goal: pack items into knapsack such that total weight is at most W and total value is maximized:

$$\max \sum_{i \in S} v_i$$

$$\text{s.t. } S \subseteq \{1, \dots, n\} \text{ and } \sum_{i \in S} w_i \leq W$$

- E.g.: jobs of length w_i and value v_i, server available for W time units, try to execute a set of jobs that maximizes the total value

Knapsack: Dynamic Programming Alg.

We have shown:

- If all item weights w_i are integers, using dynamic programming, the knapsack problem can be solved in time $O(nW)$
- If all values v_i are integers, there is another dynamic progr. algorithm that runs in time $O(n^2V)$, where V is the max. value.

Knapsack: Dynamic Programming Alg.

We have shown:

- If all item weights w_i are integers, using dynamic programming, the knapsack problem can be solved in time $O(nW)$
- If all values v_i are integers, there is another dynamic progr. algorithm that runs in time $O(n^2V)$, where V is the max. value.

Problems:

- If W and V are large, the algorithms are not polynomial in n
- If the values or weights are not integers, things are even worse (and in general, the algorithms cannot even be applied at all)

Idea:

- Can we adapt one of the algorithms to at least compute an approximate solution?

Approximation Algorithm

- The algorithm has a parameter $\varepsilon > 0$
- We assume that each item alone fits into the knapsack
- We define:

$$V := \max_{1 \le i \le n} v_i, \qquad \forall i: \hat{v}_i := \left\lceil \frac{v_i n}{\varepsilon V} \right\rceil, \qquad \hat{V} := \max_{1 \le i \le n} \hat{v}_i$$

- We solve the problem with integer values $\hat{v}_i$ and weights w_i using dynamic programming in time $O(n^2 \cdot \hat{V})$
- If solution value $< V$, we take item with value V instead

Theorem: The described algorithm runs in time $O(n^3/\varepsilon)$.

Proof:

$$\hat{V} = \max_{1 \le i \le n} \hat{v}_i = \max_{1 \le i \le n} \left\lceil \frac{v_i n}{\varepsilon V} \right\rceil = \left\lceil \frac{V n}{\varepsilon V} \right\rceil = \left\lceil \frac{n}{\varepsilon} \right\rceil$$

Approximation Algorithm

Theorem: The approximation algorithm computes a feasible solution with approximation ratio at least $1 - \varepsilon$.

Proof:

- Define the set of all feasible solutions (subsets of $[n]$)

$$\mathcal{S} := \left\{ S \subseteq \{1, \dots, n\} : \sum_{i \in S} w_i \leq W \right\}$$

- $v(S)$: value of solution S w.r.t. values $v_1, v_2, \dots$
  $\hat{v}(S)$: value of solution S w.r.t. values $\hat{v}_1, \hat{v}_2, \dots$
- S^*: an optimal solution w.r.t. values $v_1, v_2, \dots$
  $\hat{S}$: an optimal solution w.r.t. values $\hat{v}_1, \hat{v}_2, \dots$

- Weights are not changed at all, hence, $\hat{S}$ is a feasible solution

Approximation Algorithm

Theorem: The approximation algorithm computes a feasible solution with approximation ratio at least $1 - \varepsilon$.

Proof:

- We have

$$v(S^*) = \sum_{i \in S^*} v_i = \max_{S \in \mathcal{S}} \sum_{i \in S} v_i\,,$$

$$\hat{v}(\hat{S}) = \sum_{i \in \hat{S}} \hat{v}_i = \max_{S \in \mathcal{S}} \sum_{S \in \mathcal{S}} \hat{v}_i$$

- Because every item fits into the knapsack, we have

$$\forall i \in \{1, \dots, n\}\colon\ v_i \le V \le \sum_{j \in S^*} v_j$$

- Also: $\widehat{v_i} = \left\lfloor \frac{v_i n}{\varepsilon V} \right\rfloor \implies v_i \le \frac{\varepsilon V}{n} \cdot \widehat{v_i},$ and $\widehat{v_i} \le \frac{v_i n}{\varepsilon V} + 1$

Approximation Algorithm

Theorem: The approximation algorithm computes a feasible solution with approximation ratio at least $1-\varepsilon$.

Proof:

- We have

$$v(S^*) = \sum_{i\in S^*} v_i \le \frac{\varepsilon V}{n}\cdot\sum_{i\in S^*}\widehat{v}_i \le \frac{\varepsilon V}{n}\cdot\sum_{i\in \hat{S}}\widehat{v}_i \le \frac{\varepsilon V}{n}\cdot\sum_{i\in \hat{S}}\left(1+\frac{v_i n}{\varepsilon V}\right)$$

- Therefore

$$v(S^*) = \sum_{i\in S^*} v_i \le \frac{\varepsilon V}{n}\cdot\left|\hat{S}\right| + \sum_{i\in \hat{S}} v_i \le \varepsilon V + v\left(\hat{S}\right)$$

- We have $v(S^*) \ge V$ and therefore

$$\boldsymbol{(1-\varepsilon)\cdot v(S^*) \le v\left(\widehat{S}\right)}$$

Approximation Schemes

- For every parameter $\varepsilon > 0$, the knapsack algorithm computes a $(1 + \varepsilon)$-approximation in time $O(n^3/\varepsilon)$.
- For every fixed ε, we therefore get a polynomial time approximation algorithm
- An algorithm that computes an $(1 + \varepsilon)$-approximation for every $\varepsilon > 0$ is called an approximation scheme.
- If the running time is polynomial for every fixed ε, we say that the algorithm is a polynomial time approximation scheme (PTAS)
- If the running time is also polynomial in $1/\varepsilon$, the algorithm is a fully polynomial time approximation scheme (FPTAS)
- Thus, the described alg. is an FPTAS for the knapsack problem