Graphen und diskrete Optimierung

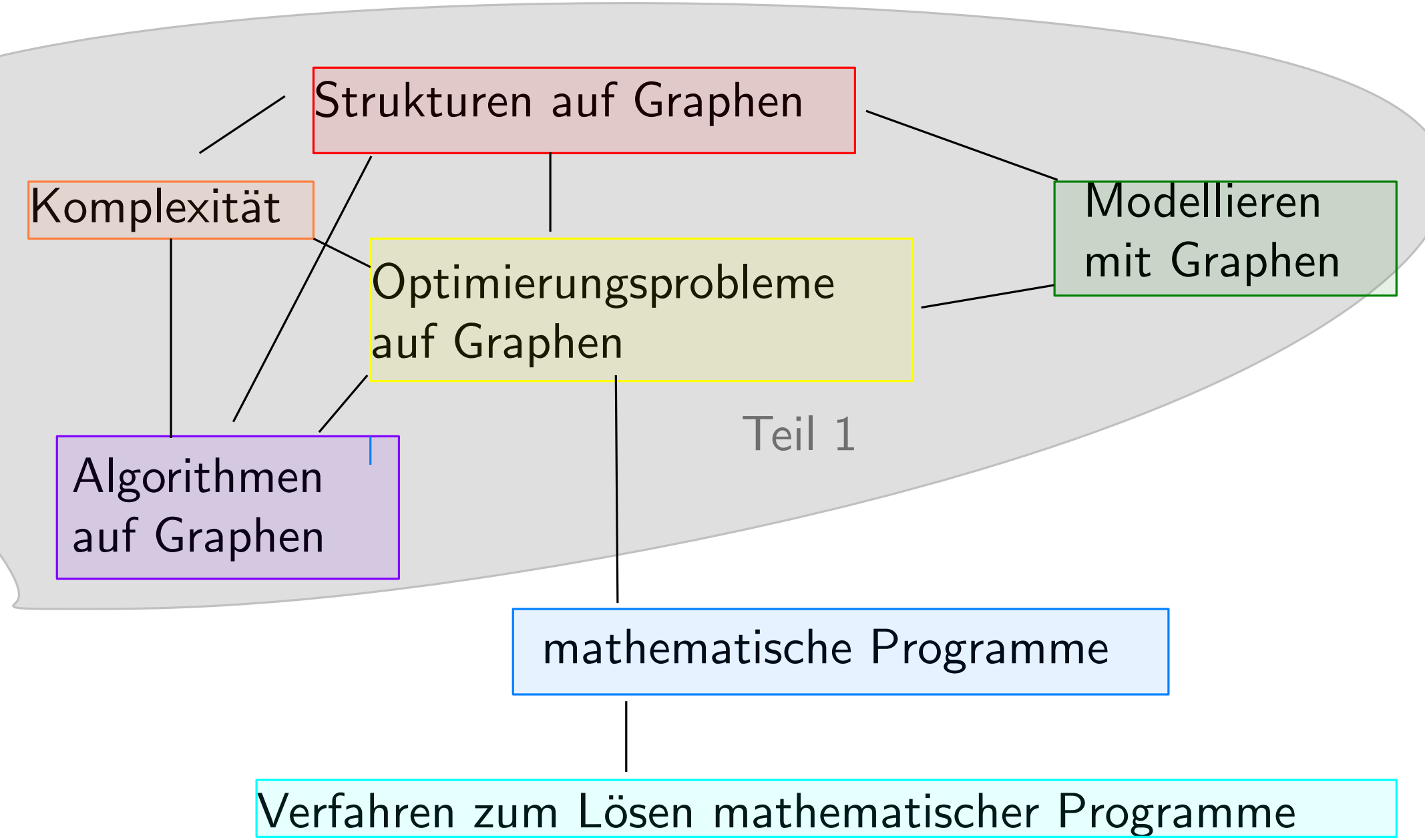
im Bachelorstudiengang 'Informatik und Nachhaltigkeit'

Algorithmen für NP-schwere Probleme

Marie Schmidt

31.05.2023

Worum soll er hier gehen?



Vorlesungsübersicht

- Einführung: Modellieren auf Graphen ✓
- Teil 1: Graphen und Algorithmen auf Graphen
  - Graphen modellieren räumliche Zusammenhänge✓
  - Graphen modellieren Beziehungen✓
  - Algorithmen und (Problem-)Komplexität
    - Problemklassen P und NP✓
    - NP-vollständig und NP-schwer✓
    - Algorithmen für NP-schwere Probleme
- Teil 2: Lineare und ganzzahlige Optimierung auf Graphen

Das macht ein NP-Schwere-Beweis

"I can't find an efficient algorithm, but neither can all these famous people."

Das macht ein NP-Schwere-Beweis

"I can't find an efficient algorithm, but neither can all these famous people."

Algorithmen für NP-schwere Probleme

Problem: für NP-schwere Probleme können wir (falls $P \neq NP$) nicht effizient Optimallösungen finden - alle bekannten Algorithmen haben exponentielle Worst-Case-Laufzeit

Algorithmen für NP-schwere Probleme

Was machen wir, wenn wir trotzdem ein NP-schweres Problem lösen müssen?

Algorithmen für NP-schwere Probleme

Was machen wir, wenn wir trotzdem ein NP-schweres Problem lösen müssen?

Algorithmen für NP-schwere Probleme

Was machen wir, wenn wir trotzdem ein NP-schweres Problem lösen müssen?

Idee 1: Wir finden (schnell/effizient) eine *gute* Lösung:

- Heuristiken
- Approximationsalgorithmen

Algorithmen für NP-schwere Probleme

Was machen wir, wenn wir trotzdem ein NP-schweres Problem lösen müssen?

Idee 1: Wir finden (schnell/effizient) eine *gute* Lösung:
- Heuristiken
- Approximationsalgorithmen

Idee 2: Wir nutzen ein Lösungsverfahren mit exponentieller Laufzeit, z.B.
- Brute Force
- Branch-and-Bound (evtl. 2. Vorlesungshälfte)
- lineare ganzzahlige Programmierung (2. Vorlesungshälfte)

Algorithmen für NP-schwere Probleme

Was machen wir, wenn wir trotzdem ein NP-schweres Problem lösen müssen?

Idee 1: Wir finden (schnell/effizient) eine *gute* Lösung:
- Heuristiken
- Approximationsalgorithmen

Idee 2: Wir nutzen ein Lösungsverfahren mit exponentieller Laufzeit, z.B.
- Brute Force
- Branch-and-Bound (evtl. 2. Vorlesungshälfte)
- lineare ganzzahlige Programmierung (2. Vorlesungshälfte)

Idee 3: Vielleicht ist unser konkretes Problem ein Spezialfall für den es polynomiellen Algorithmus gibt? (Bsp: chordaler Graph beim Färbungsproblem)

Algorithmen für NP-schwere Probleme

Was machen wir, wenn wir trotzdem ein NP-schweres Problem lösen müssen?

Idee 1: Wir finden (schnell/effizient) eine *gute* Lösung:
- Heuristiken
- Approximationsalgorithmen

Idee 2: Wir nutzen ein Lösungsverfahren mit exponentieller Laufzeit, z.B.
- Brute Force
- Branch-and-Bound (evtl. 2. Vorlesungshälfte)
- lineare ganzzahlige Programmierung (2. Vorlesungshälfte)

Idee 3: Vielleicht ist unser konkretes Problem ein Spezialfall für den es polynomiellen Algorithmus gibt? (Bsp: chordaler Graph beim Färbungsproblem)

Idee 4: Wir nutzen ein (eigentlich exponentielles) Lösungsverfahren, das für *spezielle Probleminstanzen* doch effizient ist:
- pseudopolynomielle Algorithmen
- Festparameter-Berechenbarkeit

Algorithmen für NP-schwere Probleme

Was machen wir, wenn wir trotzdem ein NP-schweres Problem lösen müssen?

Idee 1: Wir finden (schnell/effizient) eine *gute* Lösung:
- Heuristiken
- Approximationsalgorithmen

Idee 2: Wir nutzen ein Lösungsverfahren mit exponentieller Laufzeit, z.B.
- Brute Force
- Branch-and-Bound (evtl. 2. Vorlesungshälfte)
- lineare ganzzahlige Programmierung (2. Vorlesungshälfte)

Idee 3: Vielleicht ist unser konkretes Problem ein Spezialfall für den es polynomiellen Algorithmus gibt? (Bsp: chordaler Graph beim Färbungsproblem)

Idee 4: Wir nutzen ein (eigentlich exponentielles) Lösungsverfahren, das für *spezielle Probleminstanzen* doch effizient ist:
- pseudopolynomielle Algorithmen
- Festparameter-Berechenbarkeit

Idee 5: randomisierte Algorithmen & average-case Analyse

Algorithmen für NP-schwere Probleme

Was machen wir, wenn wir trotzdem ein NP-schweres Problem lösen müssen?

Idee 1: Wir finden (schnell/effizient) eine *gute* Lösung:
- Heuristiken
- Approximationsalgorithmen ← jetzt

Idee 2: Wir nutzen ein Lösungsverfahren mit exponentieller Laufzeit, z.B.
- Brute Force
- Branch-and-Bound (evtl. 2. Vorlesungshälfte)
- lineare ganzzahlige Programmierung (2. Vorlesungshälfte)

Idee 3: Vielleicht ist unser konkretes Problem ein Spezialfall für den es polynomiellen Algorithmus gibt? (Bsp: chordaler Graph beim Färbungsproblem)

Idee 4: Wir nutzen ein (eigentlich exponentielles) Lösungsverfahren, das für *spezielle Probleminstanzen* doch effizient ist:
- pseudopolynomielle Algorithmen
- Festparameter-Berechenbarkeit ← später

Idee 5: randomisierte Algorithmen & average-case Analyse

Approximationsalgorithmus

Sei X ein Minimierungsproblem mit Zielfunktion f.
Für eine Instanz I von X bezeichnen wir im folgenden den Zielfunktionswert einer Optimallösung mit $f_{\mathrm{OPT}(I)}$.

Approximationsalgorithmus

Sei X ein Minimierungsproblem mit Zielfunktion f.
Für eine Instanz I von X bezeichnen wir im folgenden den Zielfunktionswert einer Optimallösung mit $f_{\mathrm{OPT}(I)}$.

Ein Algorithmus A, der zu jeder Instanz I von X eine Lösung $A(I)$ mit $f(A(I)) \leq \alpha \cdot f_{\mathrm{OPT}(I)}$ findet, nennen wir α-**Approximationsalgorithmus** für X.

Approximationsalgorithmus

Sei X ein Minimierungsproblem mit Zielfunktion f.
Für eine Instanz I von X bezeichnen wir im folgenden den Zielfunktionswert einer Optimallösung mit $f_{\mathrm{OPT}(I)}$.

Ein Algorithmus A, der zu jeder Instanz I von X eine Lösung $A(I)$ mit $f(A(I)) \leq \alpha \cdot f_{\mathrm{OPT}(I)}$ findet, nennen wir α-**Approximationsalgorithmus** für X.

α nennen wir **Approximationsfaktor, Approximationsgüte, oder Approximationsgarantie.**

Approximationsalgorithmus

Sei X ein Minimierungsproblem mit Zielfunktion f.
Für eine Instanz I von X bezeichnen wir im folgenden den Zielfunktionswert einer Optimallösung mit $f_{\mathrm{OPT}(I)}$.

Ein Algorithmus A, der zu jeder Instanz I von X eine Lösung $A(I)$ mit $f(A(I)) \leq \alpha \cdot f_{\mathrm{OPT}(I)}$ findet, nennen wir α-**Approximationsalgorithmus** für X.

α nennen wir **Approximationsfaktor, Approximationsgüte, oder Approximationsgarantie.**

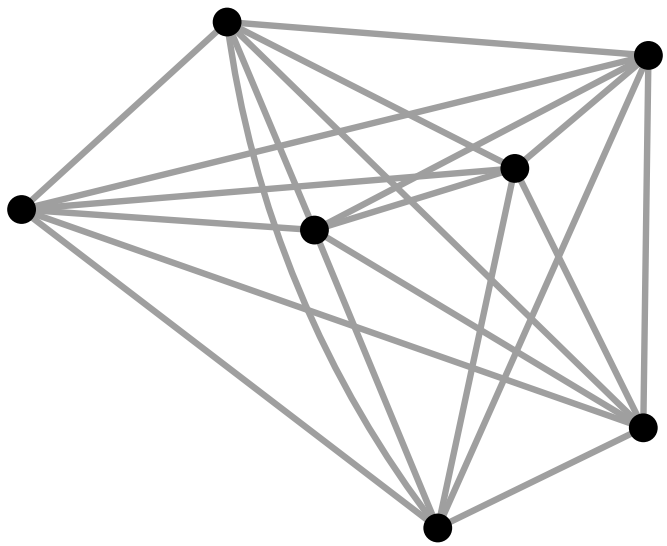
Wie definieren wir 'Approximationsalgorithmus' sinnvollerweise für Maximierungsprobleme?

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Tree-Doubling für das TSP

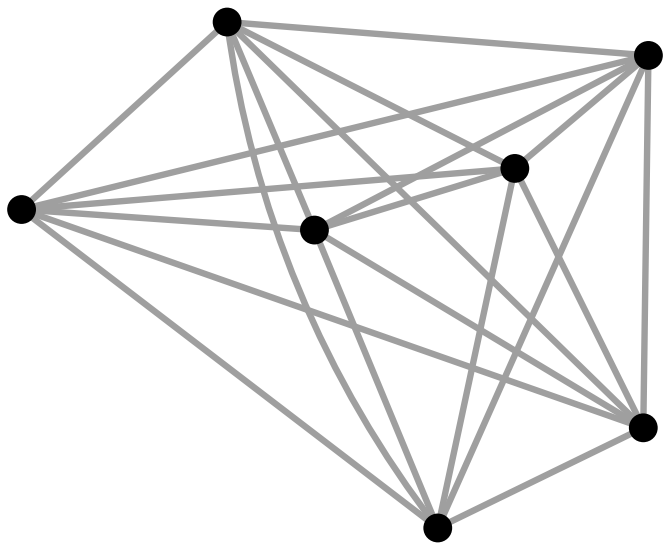
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal



Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Approximationsalgorithmus für das TSP

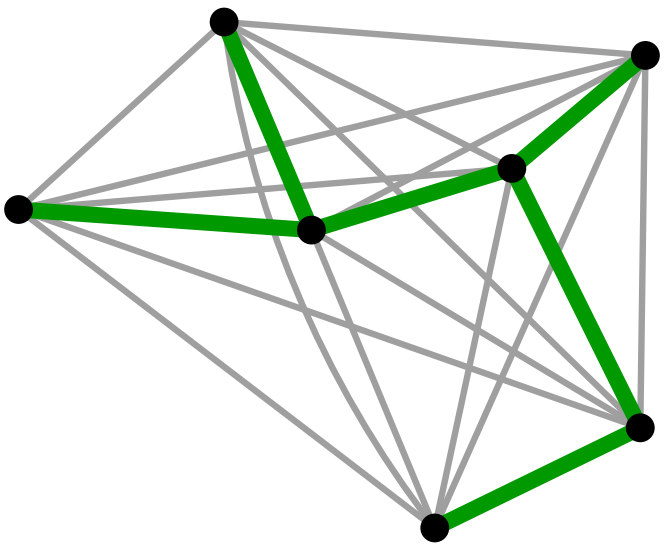


Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

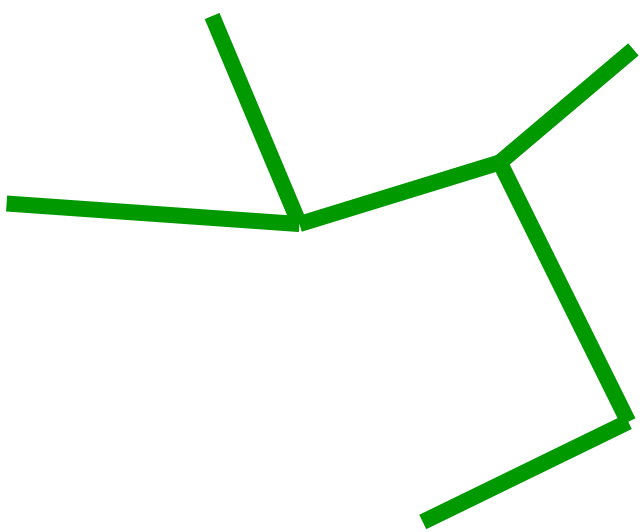


Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.



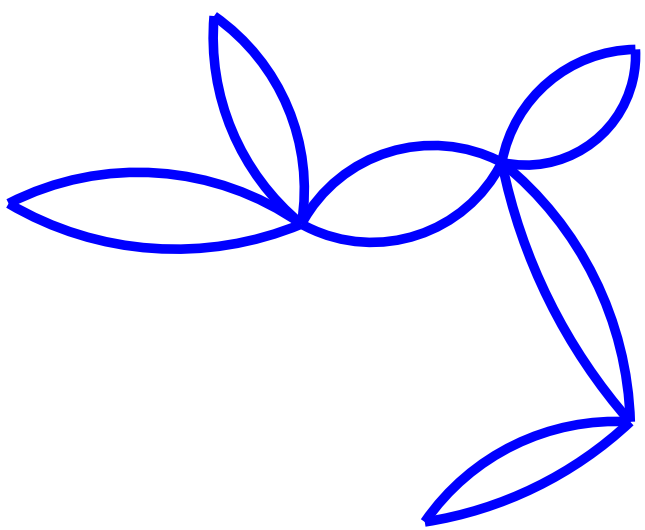
Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.



Tree-Doubling für das TSP

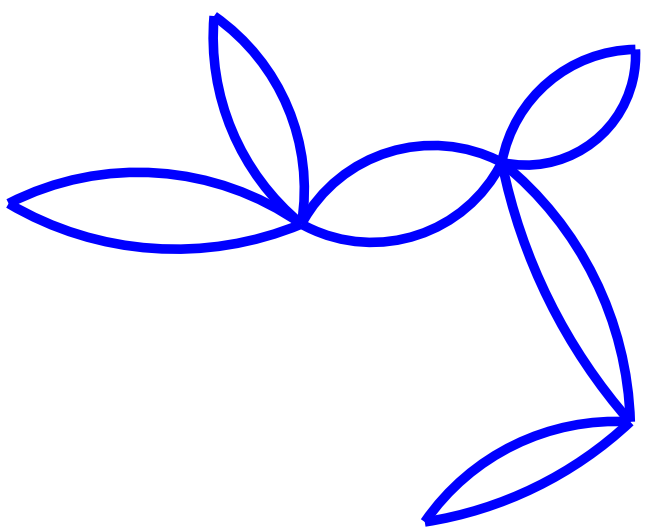
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.



Tree-Doubling für das TSP

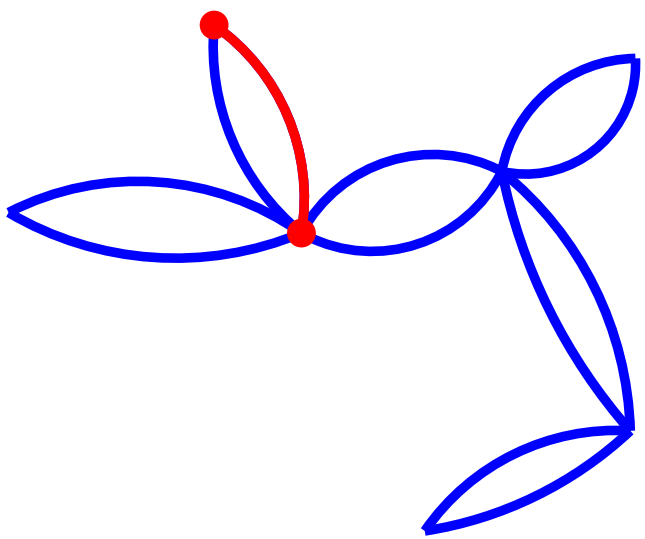
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.



Tree-Doubling für das TSP

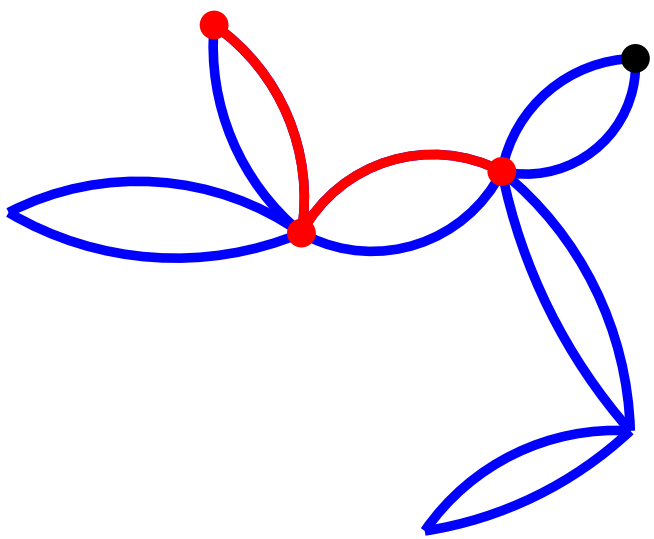
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.



Tree-Doubling für das TSP

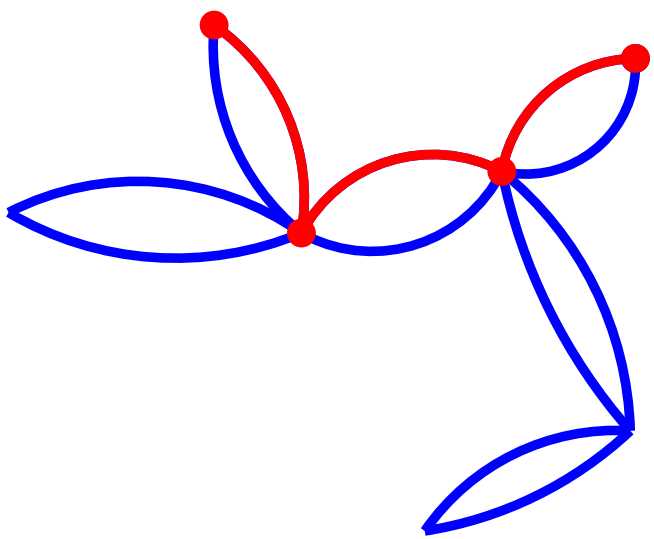
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.



Tree-Doubling für das TSP

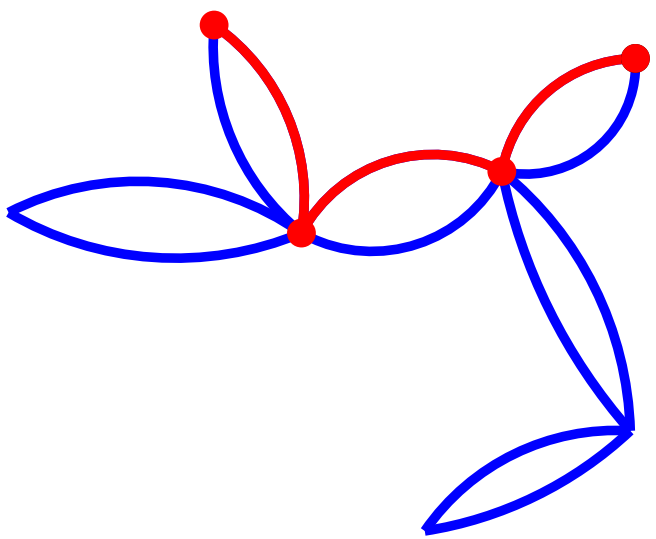
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

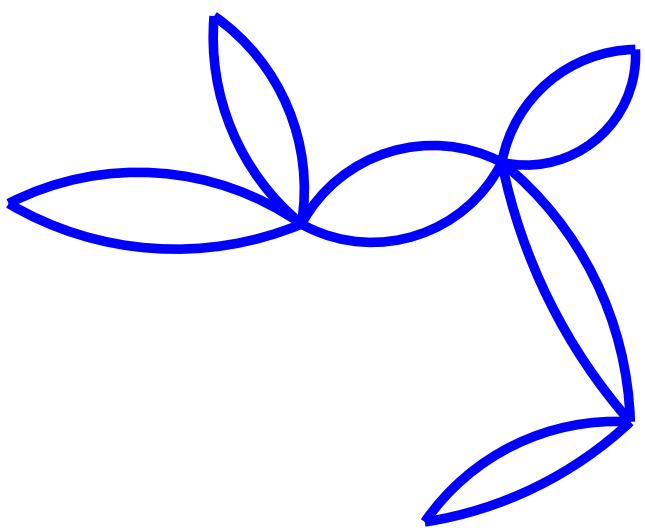


Laufzeit?

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse



Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

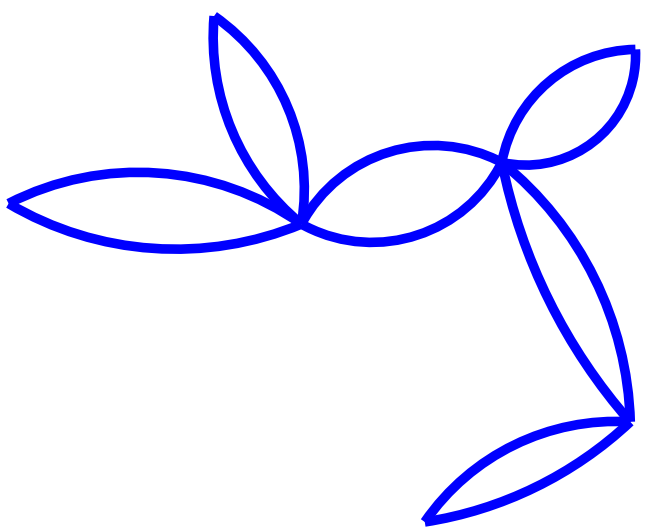
Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse
$w(A(I)) =$



Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

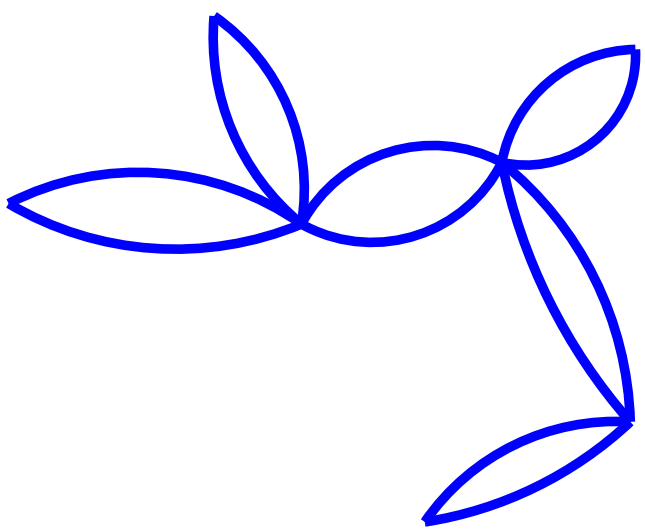
Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse
$w(A(I)) = w(\text{EK})$



Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

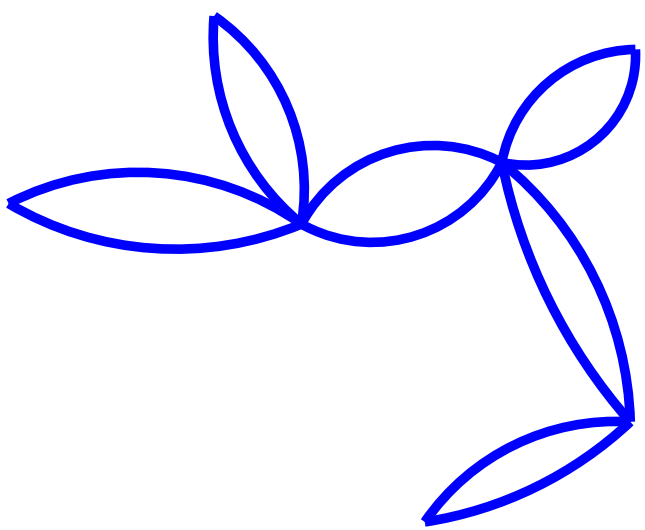
Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse
$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB})$



Approximationsalgorithmus für das TSP

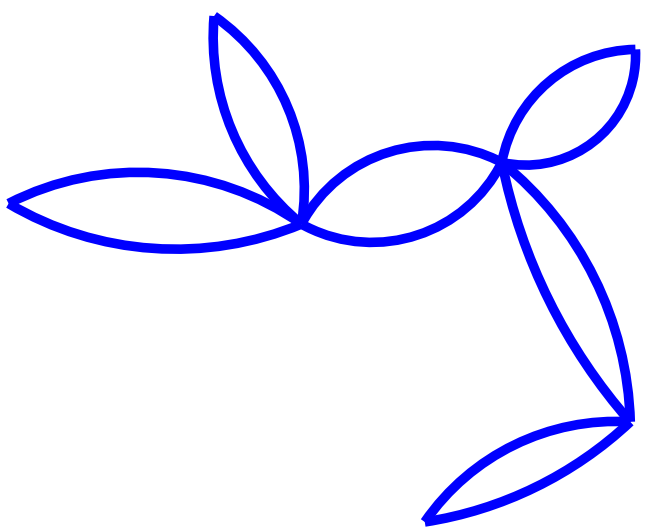
- Bestimme minmalen Spannbaum MSB von G.
- Verdopple minimalen Spannbaum.
- Durchlaufe Eulerkreis EK.

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB}) \leq 2 \cdot w_{\text{OPT}(I)}$$



Approximationsalgorithmus für das TSP

- Bestimme minmalen Spannbaum MSB von G.
- Verdopple minimalen Spannbaum.
- Durchlaufe Eulerkreis EK.

Tree-Doubling für das TSP

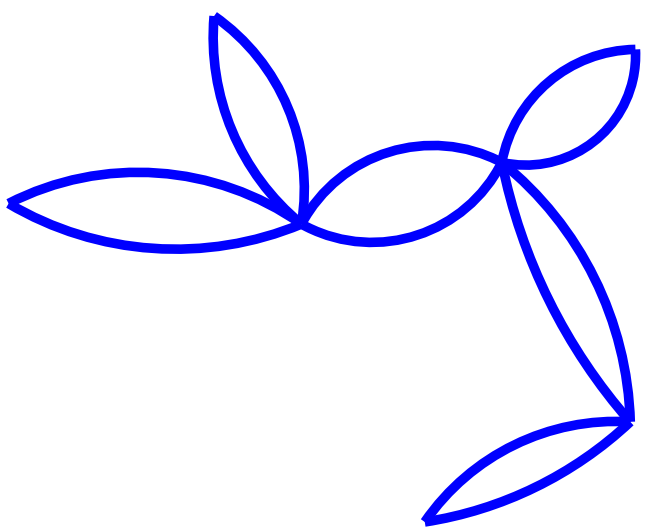
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB}) \leq 2 \cdot w_{\text{OPT}(I)}$$

Approximationsalgorithmus für das TSP

- Bestimme minmalen Spannbaum MSB von G.
- Verdopple minimalen Spannbaum.
- Durchlaufe Eulerkreis EK.



Tree-Doubling ist ein 2-Approximationsalgorithmus für das TSP.

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB}) \leq 2 \cdot w_{\text{OPT}(I)}$$

Approximationsalgorithmus für das TSP

- Bestimme minmalen Spannbaum MSB von G.
- Verdopple minimalen Spannbaum.
- Durchlaufe Eulerkreis EK.

Können wir das noch besser machen?

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB}) \leq 2 \cdot w_{\text{OPT}(I)}$$

Können wir das noch besser machen?

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

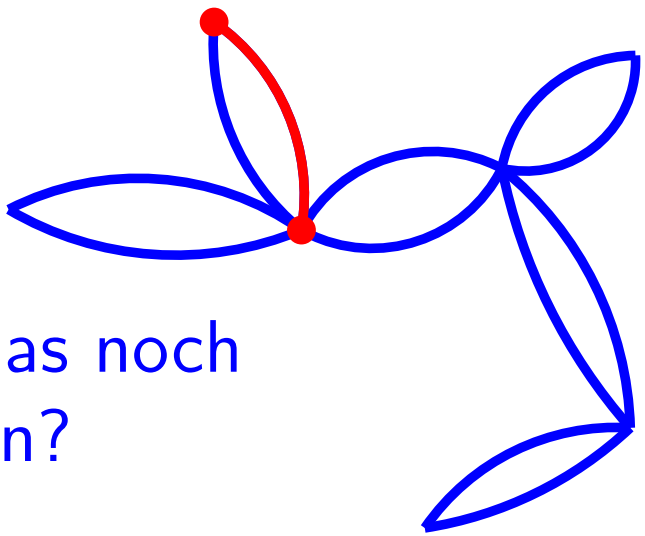
Durchlaufe Eulerkreis EK.

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB}) \leq 2 \cdot w_{\text{OPT}(I)}$$



Können wir das noch besser machen?

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

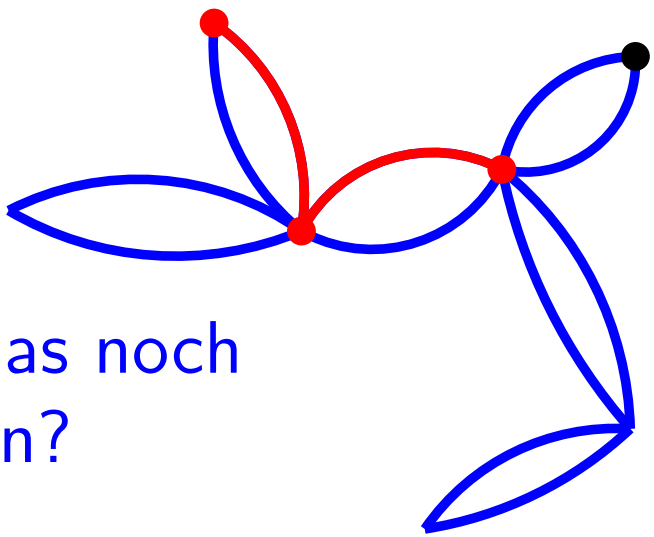
Durchlaufe Eulerkreis EK.

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB}) \leq 2 \cdot w_{\text{OPT}(I)}$$



Können wir das noch besser machen?

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

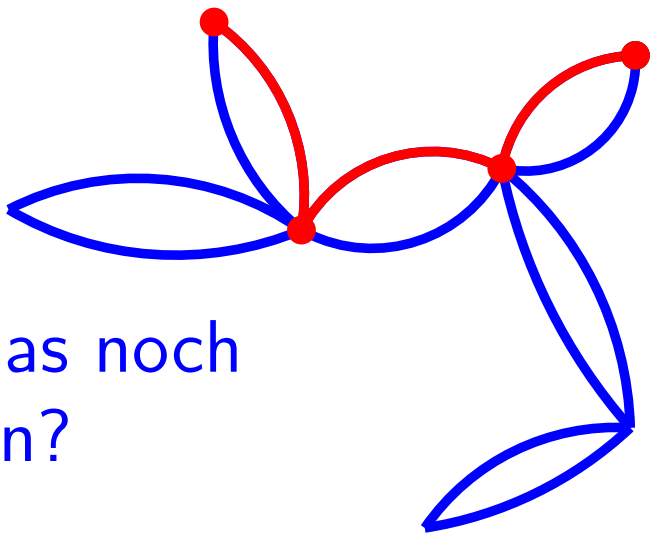
Durchlaufe Eulerkreis EK.

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB}) \leq 2 \cdot w_{\text{OPT}(I)}$$



Können wir das noch besser machen?

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

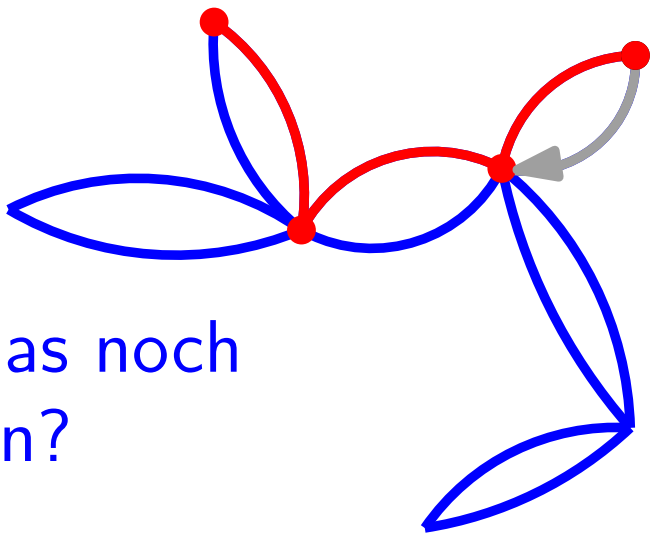
Durchlaufe Eulerkreis EK.

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB}) \leq 2 \cdot w_{\text{OPT}(I)}$$



Können wir das noch besser machen?

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

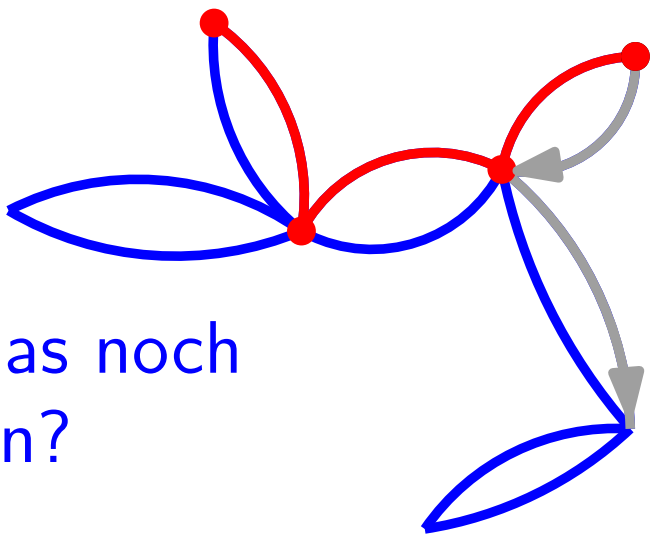
Durchlaufe Eulerkreis EK.

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB}) \leq 2 \cdot w_{\text{OPT}(I)}$$



Können wir das noch besser machen?

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

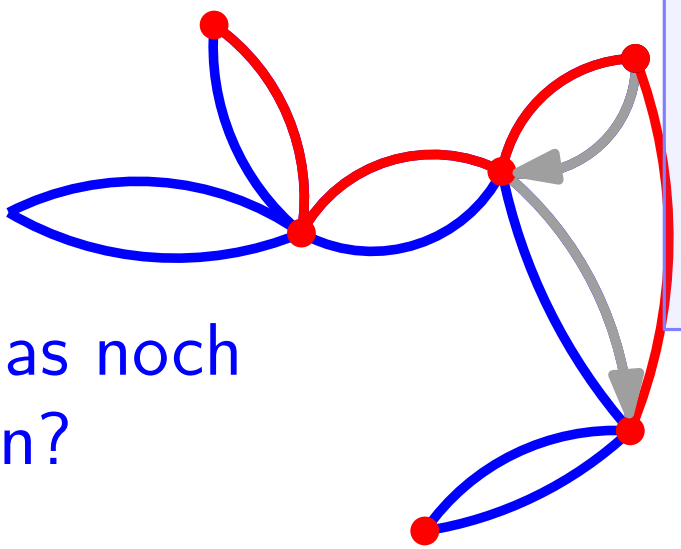
Überspringe dabei besuchte Knote

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB}) \leq 2 \cdot w_{\text{OPT}(I)}$$



Können wir das noch besser machen?

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

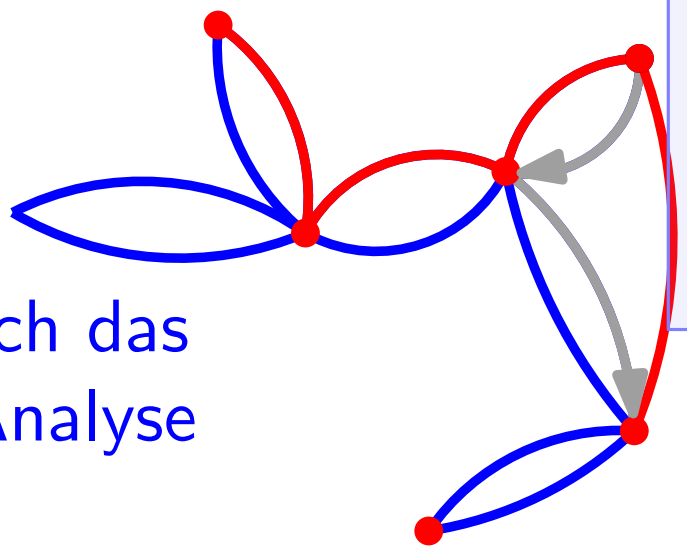
Überspringe dabei besuchte Knote
Füge „Abkürzungen“ ein.

Tree-Doubling für das TSP

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) = w(\text{EK}) = 2 \cdot w(\text{MSB}) \leq 2 \cdot w_{\text{OPT}(I)}$$



Wie wirkt sich das auf unsere Analyse aus?

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

Überspringe dabei besuchte Knote
Füge „Abkürzungen“ ein.

Tree-Doubling für das TSP

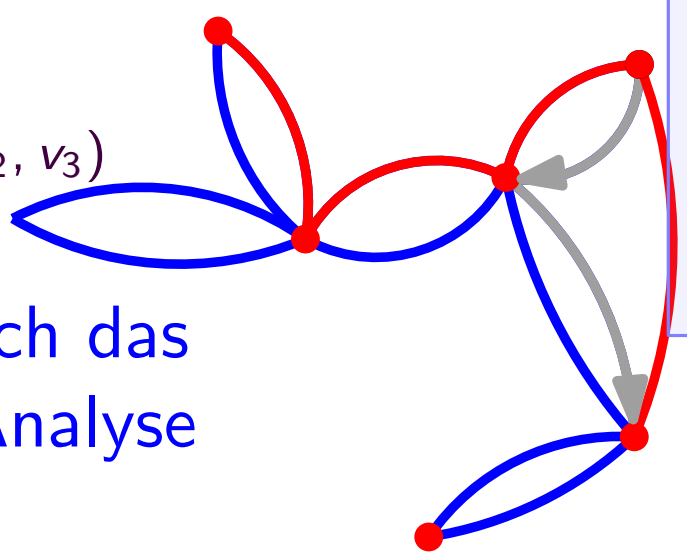
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) \leq w(\text{EK}) = 2 \cdot w(\text{MSB})$$
$$\leq 2 \cdot w_{\text{OPT}(I)}$$

falls Dreiecksungleichung gilt

$w(v_1, v_3) \leq w(v_1, v_2) + w(v_2, v_3)$
$\forall v_i \in V$



Wie wirkt sich das auf unsere Analyse aus?

Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

Überspringe dabei besuchte Knote
Füge „Abkürzungen“ ein.

Tree-Doubling für das TSP

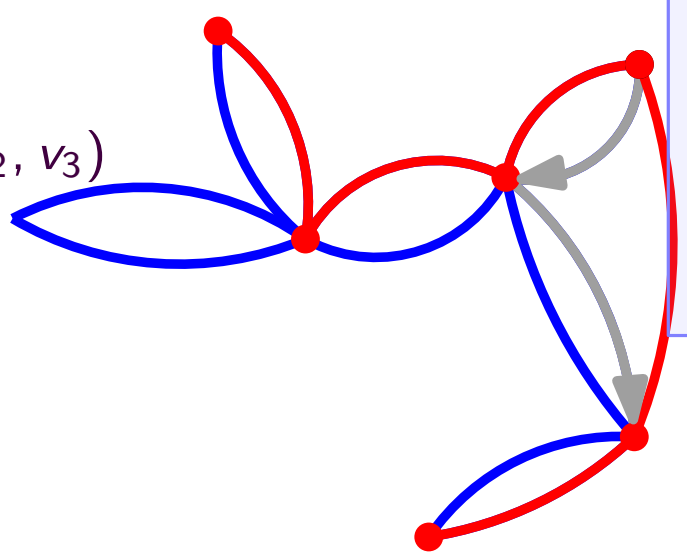
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) \leq w(\text{EK}) = 2 \cdot w(\text{MSB})$$
$$\leq 2 \cdot w_{\text{OPT}(I)}$$

falls Dreiecksungleichung gilt

$w(v_1, v_3) \leq w(v_1, v_2) + w(v_2, v_3)$
$\forall v_i \in V$



Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

Überspringe dabei besuchte Knote
Füge „Abkürzungen“ ein.

Tree-Doubling für das TSP

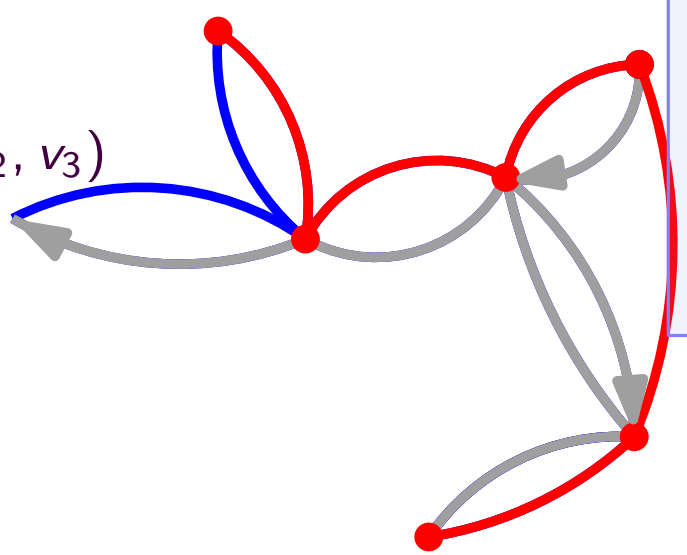
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) \leq w(\mathrm{EK}) = 2 \cdot w(\mathrm{MSB})$$
$$\leq 2 \cdot w_{\mathrm{OPT}(I)}$$

falls Dreiecksungleichung gilt

$w(v_1, v_3) \leq w(v_1, v_2) + w(v_2, v_3)$
$\forall v_i \in V$



Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

Überspringe dabei besuchte Knote
Füge „Abkürzungen“ ein.

Tree-Doubling für das TSP

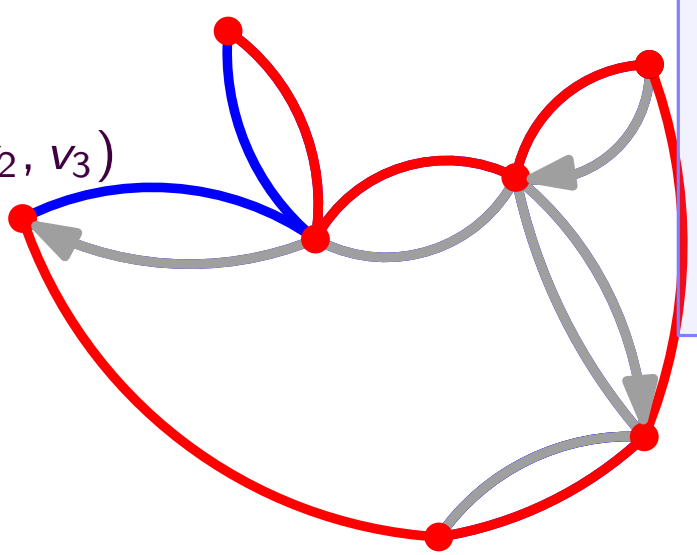
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) \leq w(\mathrm{EK}) = 2 \cdot w(\mathrm{MSB}) \leq 2 \cdot w_{\mathrm{OPT}(I)}$$

falls Dreiecksungleichung gilt

$w(v_1, v_3) \leq w(v_1, v_2) + w(v_2, v_3)$
$\forall v_i \in V$



Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

Überspringe dabei besuchte Knote
Füge „Abkürzungen“ ein.

Tree-Doubling für das TSP

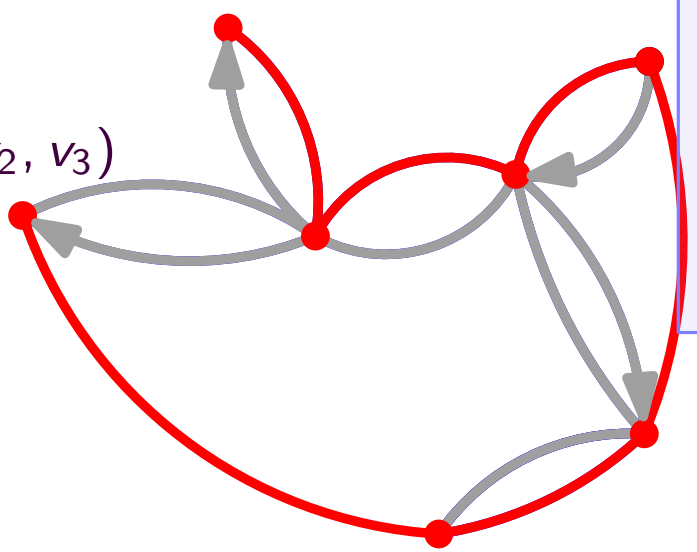
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) \leq w(\mathrm{EK}) = 2 \cdot w(\mathrm{MSB}) \leq 2 \cdot w_{\mathrm{OPT}(I)}$$

falls Dreiecksungleichung gilt

$w(v_1, v_3) \leq w(v_1, v_2) + w(v_2, v_3)$
$\forall v_i \in V$



Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

Überspringe dabei besuchte Knote
Füge „Abkürzungen“ ein.

Tree-Doubling für das TSP

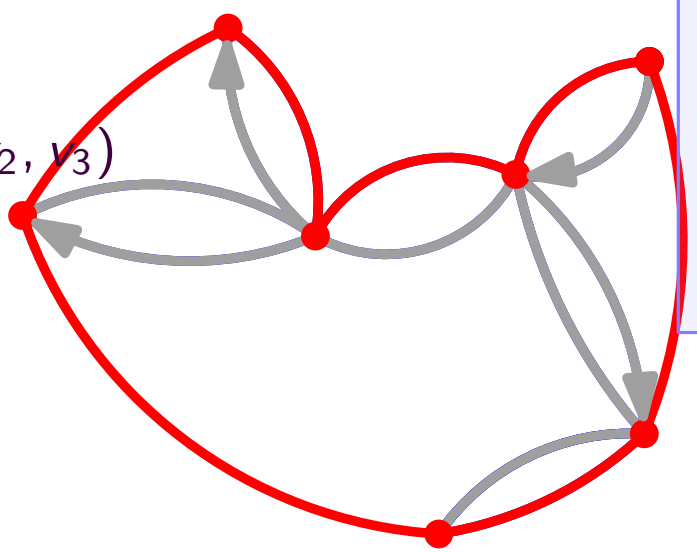
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) \leq w(\mathrm{EK}) = 2 \cdot w(\mathrm{MSB}) \leq 2 \cdot w_{\mathrm{OPT}(I)}$$

falls Dreiecksungleichung gilt

$w(v_1, v_3) \leq w(v_1, v_2) + w(v_2, v_3)$
$\forall v_i \in V$



Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

Überspringe dabei besuchte Knote
Füge „Abkürzungen“ ein.

Tree-Doubling für das TSP

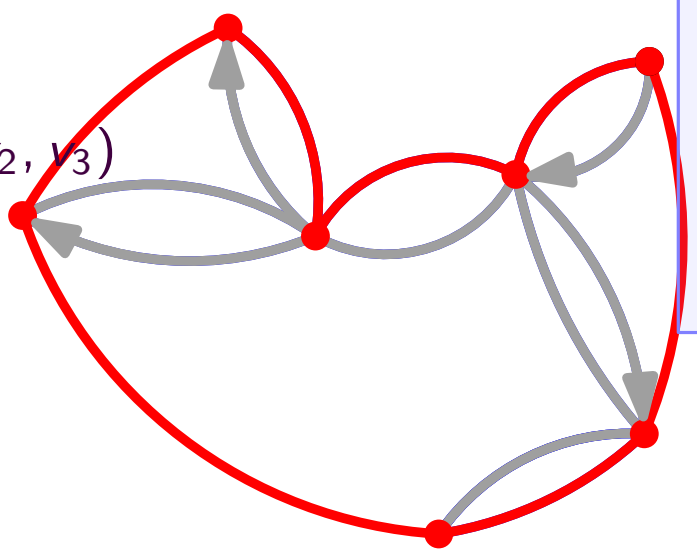
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Analyse

$$w(A(I)) \leq w(\text{EK}) = 2 \cdot w(\text{MSB})$$
$$\leq 2 \cdot w_{\text{OPT}(I)}$$

falls Dreiecksungleichung gilt

$w(v_1, v_3) \leq w(v_1, v_2) + w(v_2, v_3)$
$\forall v_i \in V$



Approximationsalgorithmus für das TSP

Bestimme minmalen Spannbaum MSB von G.

Verdopple minimalen Spannbaum.

Durchlaufe Eulerkreis EK.

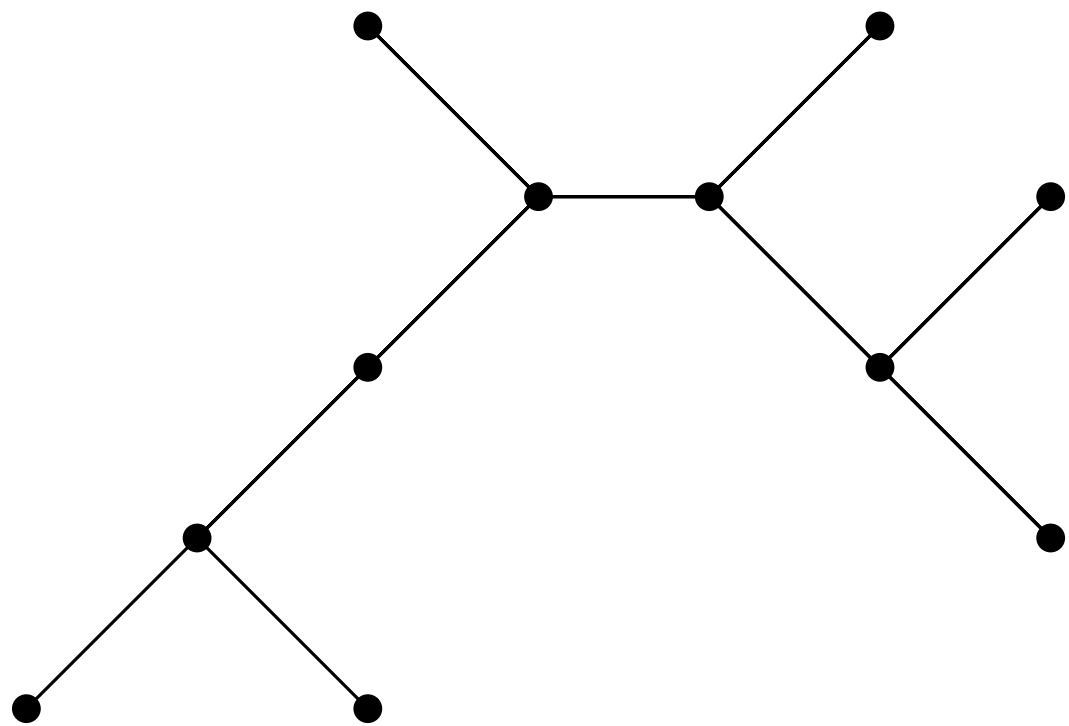
Überspringe dabei besuchte Knote
Füge „Abkürzungen“ ein.

Tree-Doubling ist ein 2-Approximationsalgorithmus für das TSP.

Christofides' Algorithmus

Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

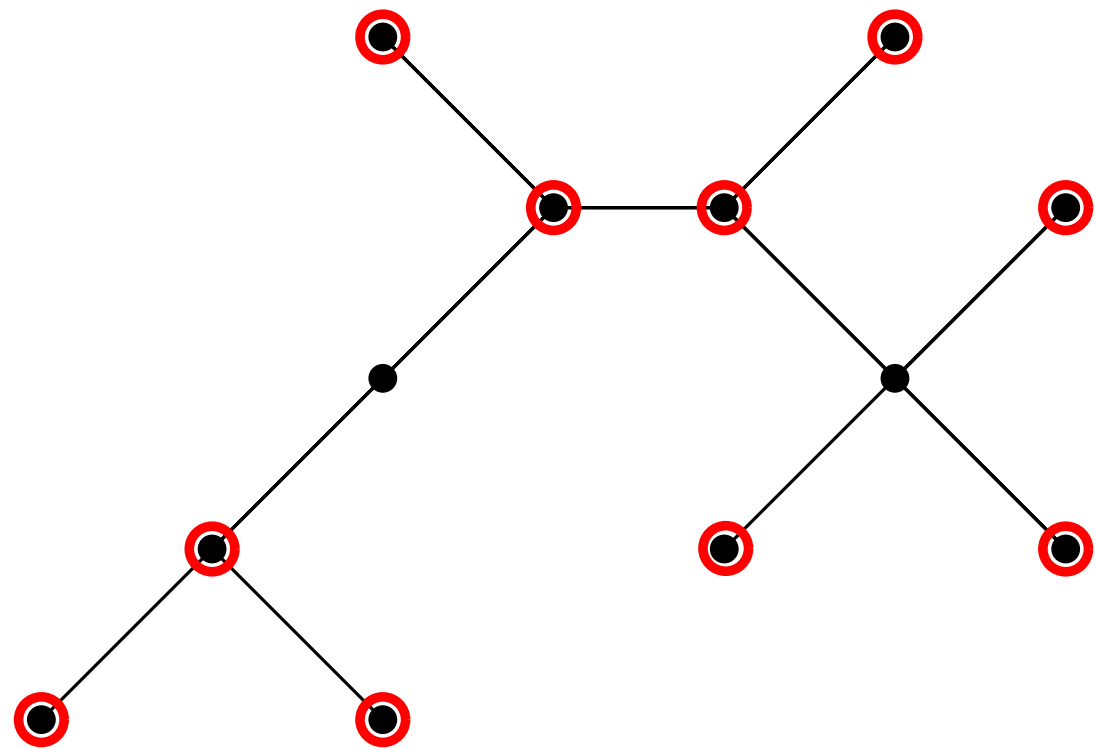


Christofides' Algorithmus

Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.



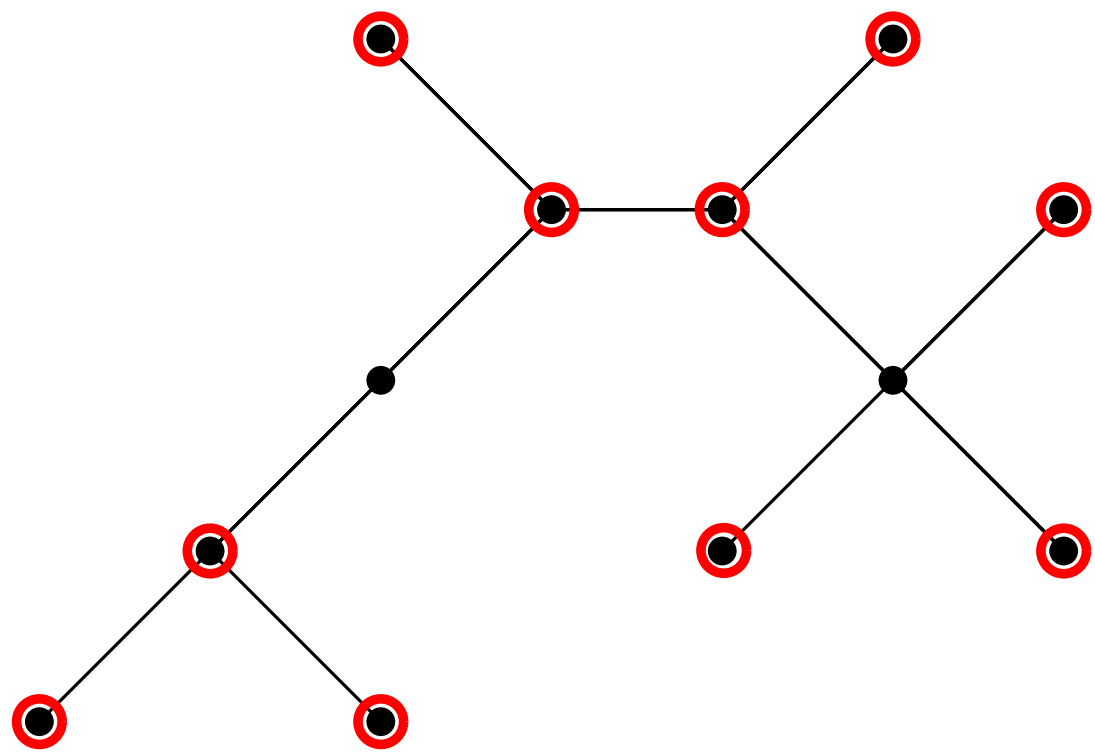
Christofides' Algorithmus

Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Das ist immer eine gerade Anzahl! Warum?

Sei U die Menge der Knoten ungeraden Grades in B.



Christofides' Algorithmus

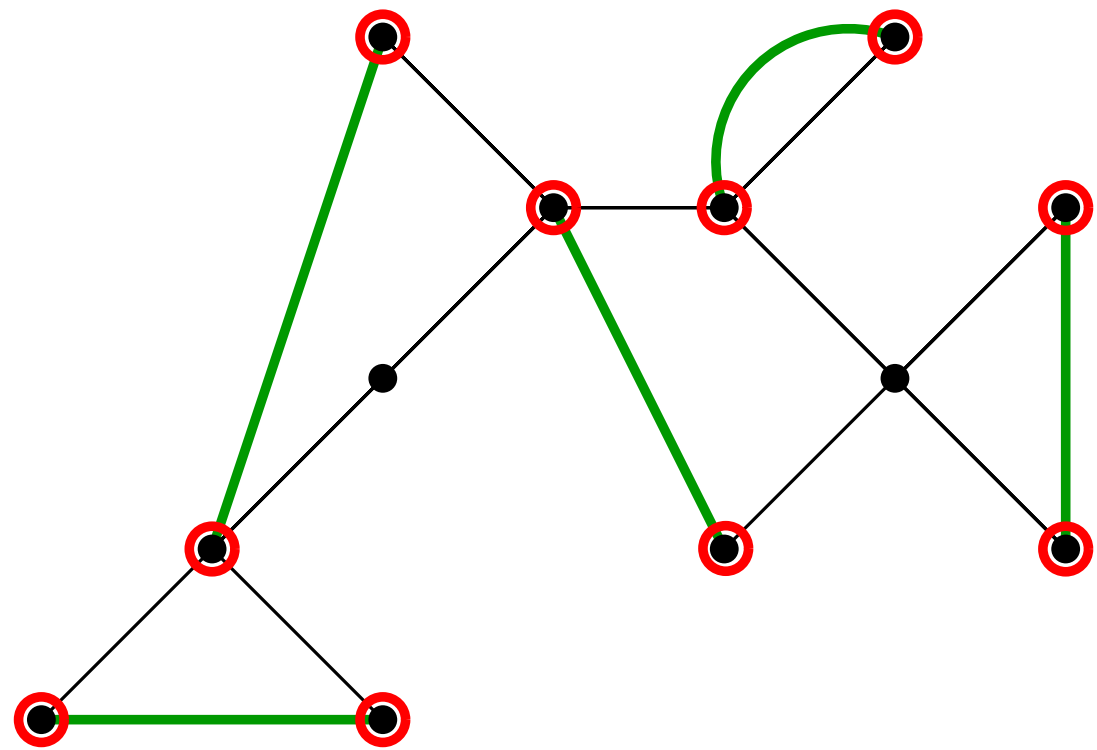
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Das ist immer eine gerade Anzahl! Warum?

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$



Christofides' Algorithmus

Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

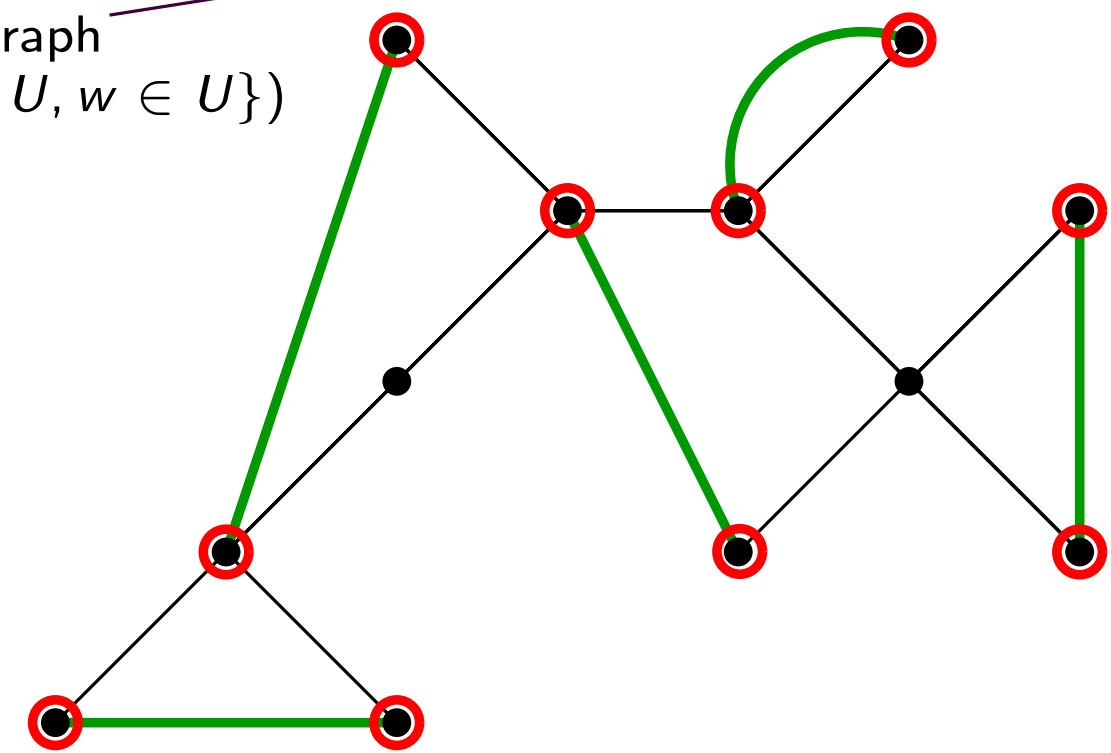
Bestimme minimalen Spannbaum B für G.

Das ist immer eine gerade Anzahl! Warum?

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

der von U *induzierte* Graph
$(U, \{vw \in E(G) \colon v \in U, w \in U\})$



Christofides' Algorithmus

Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

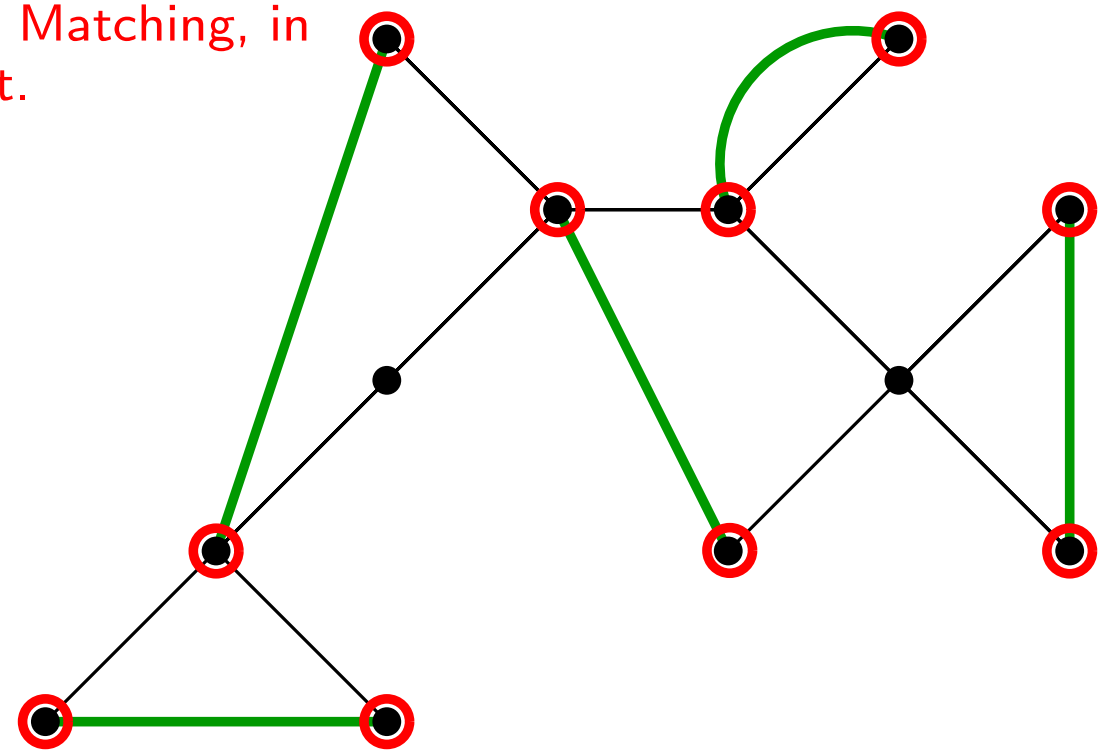
Bestimme minimalen Spannbaum B für G.

Das ist immer eine gerade Anzahl! Warum?

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Ein **perfektes** Matching ist ein Matching, in dem jeder Knoten gematcht ist.



Christofides' Algorithmus

Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

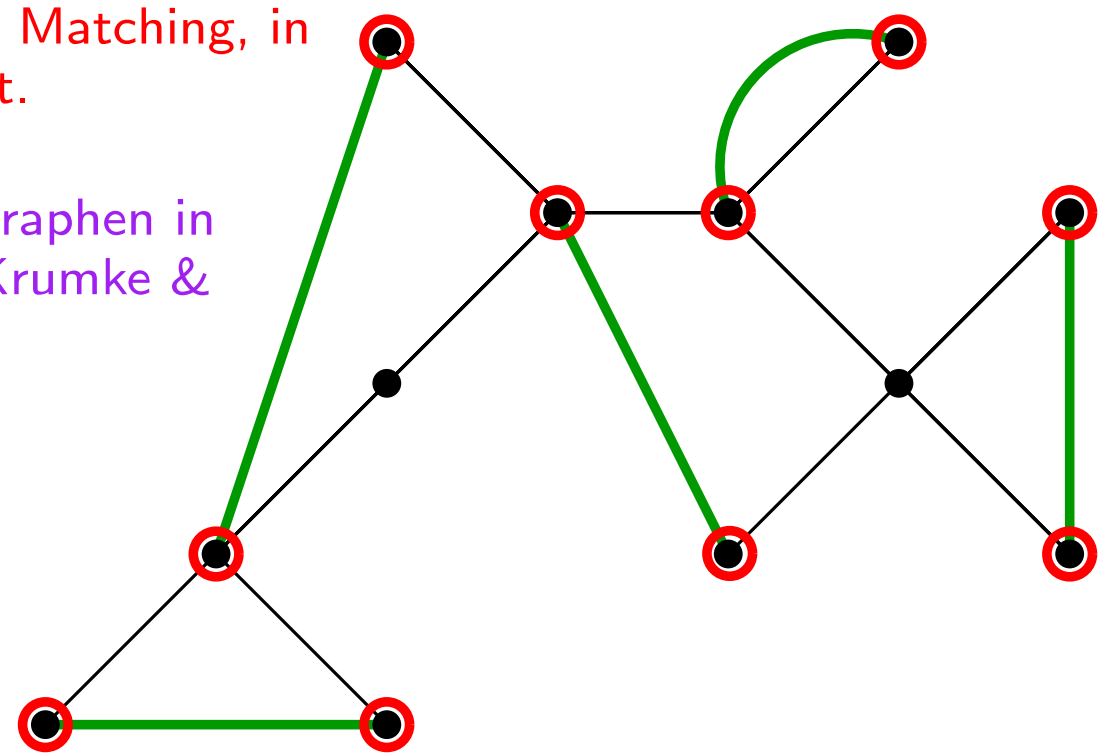
Das ist immer eine gerade Anzahl! Warum?

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Ein **perfektes** Matching ist ein Matching, in dem jeder Knoten gematcht ist.

Findet man auch im nicht-bipartiten Graphen in Polynomialzeit (siehe Kapitel 10.6 in Krumke & Noltemeier)



Christofides' Algorithmus

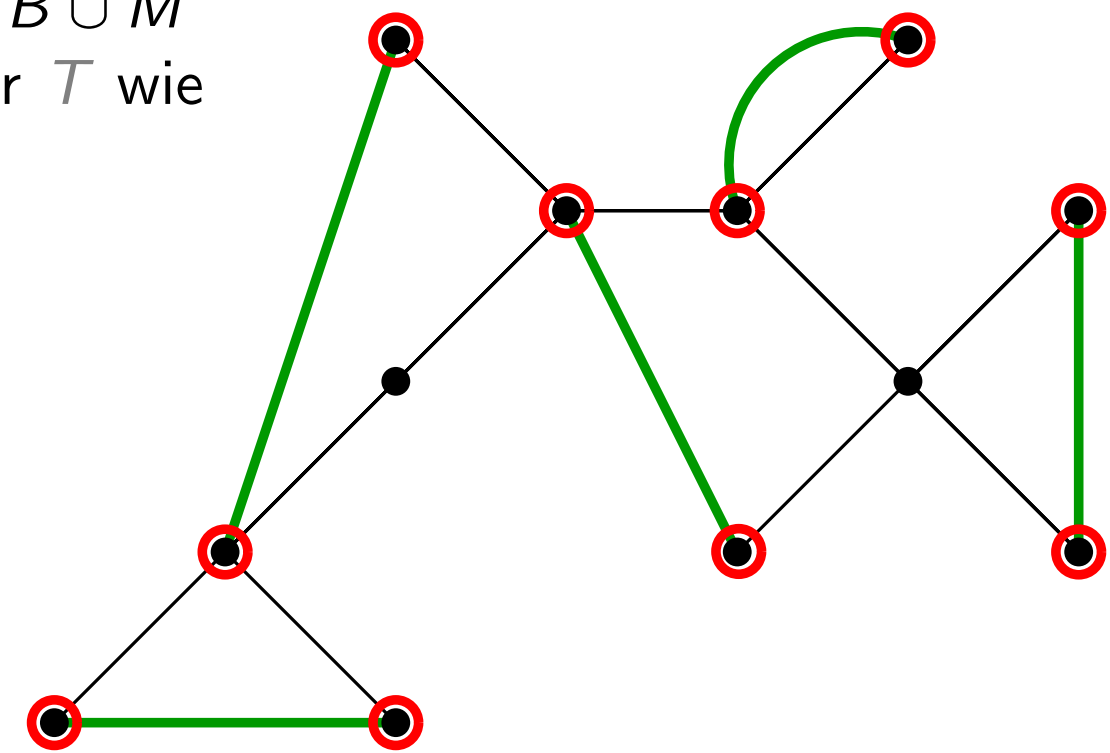
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

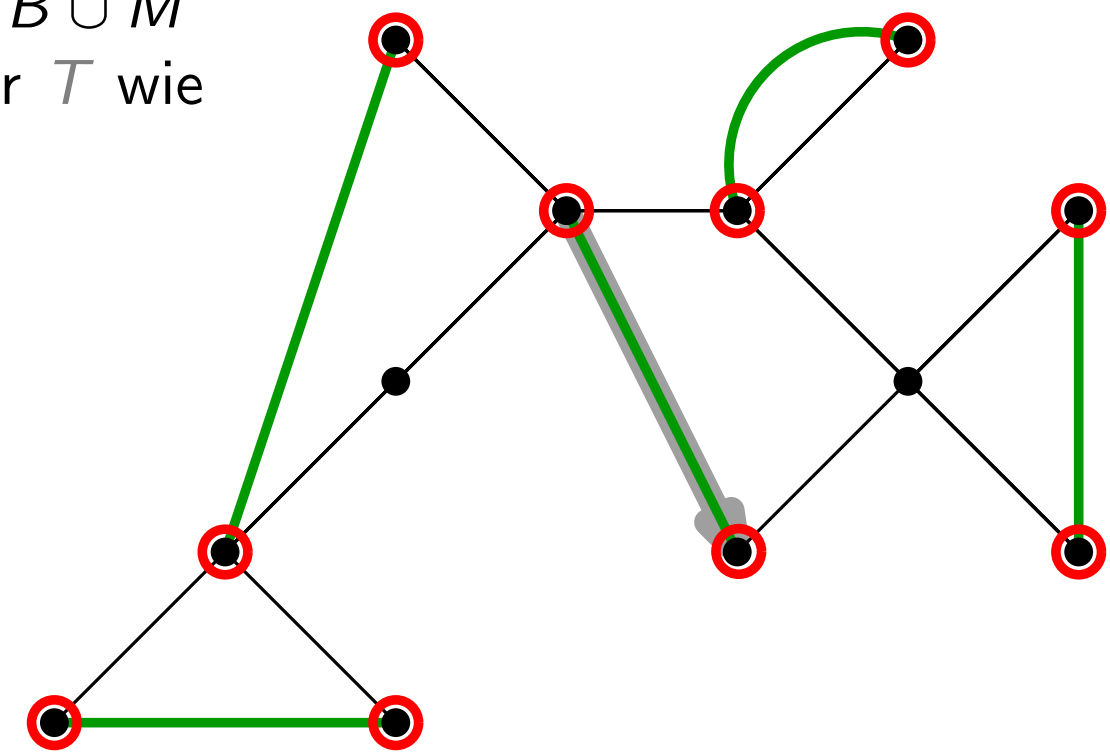
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

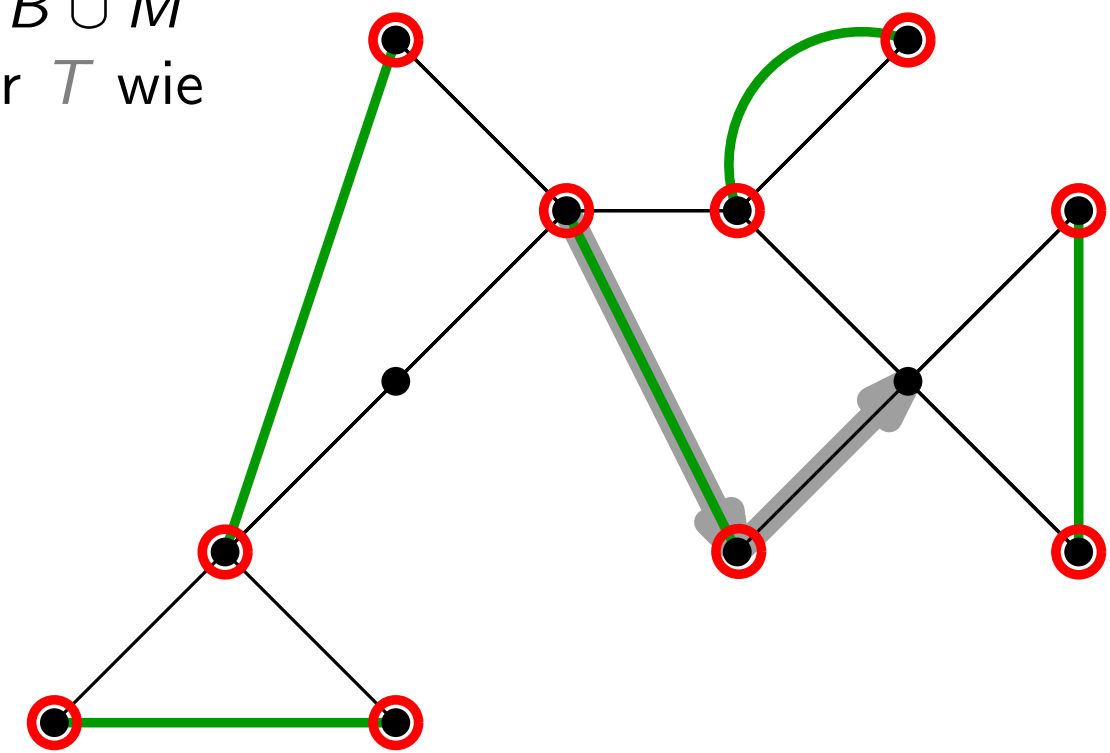
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

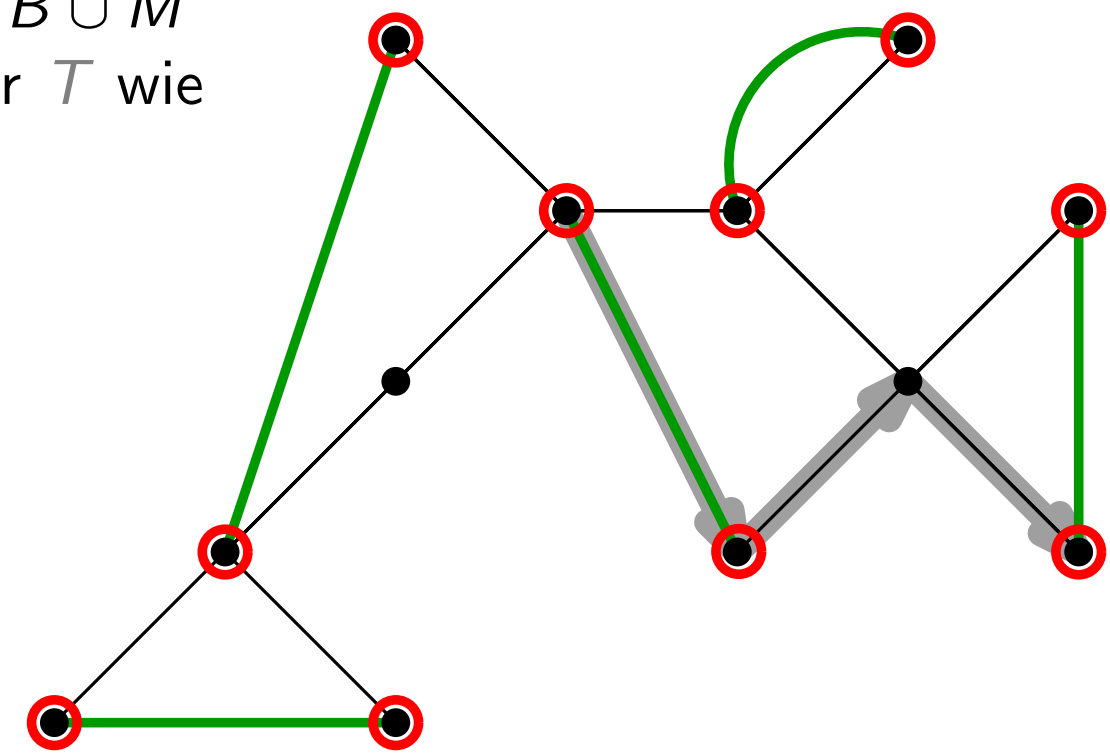
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

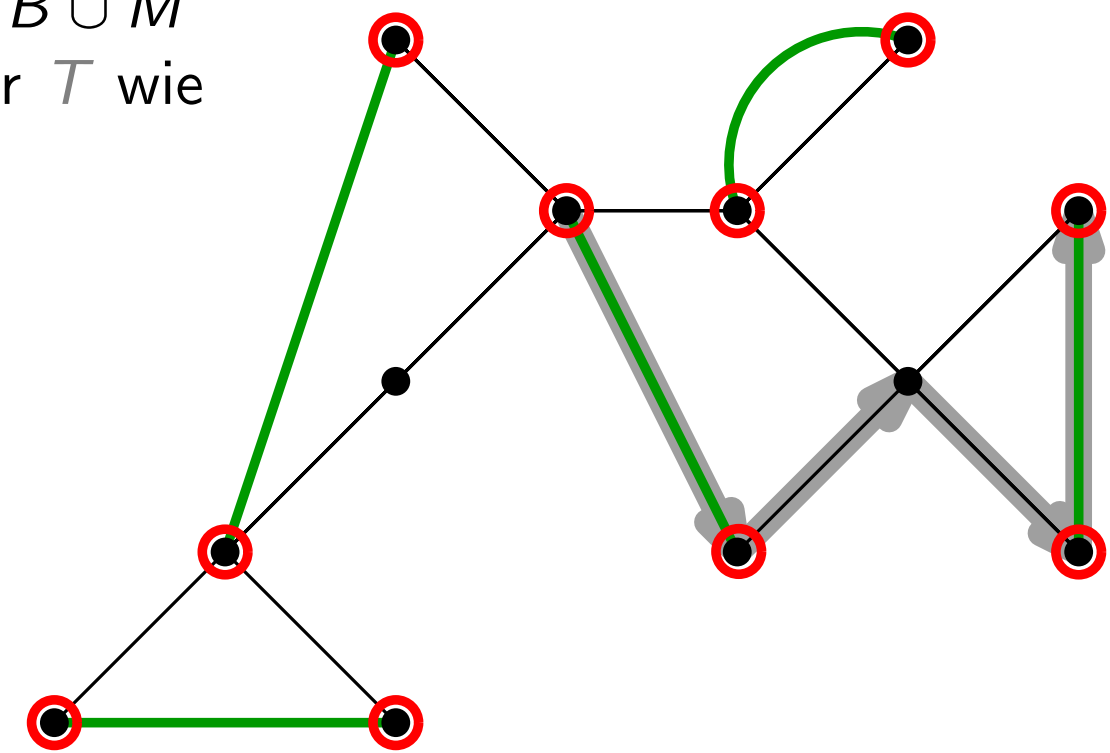
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

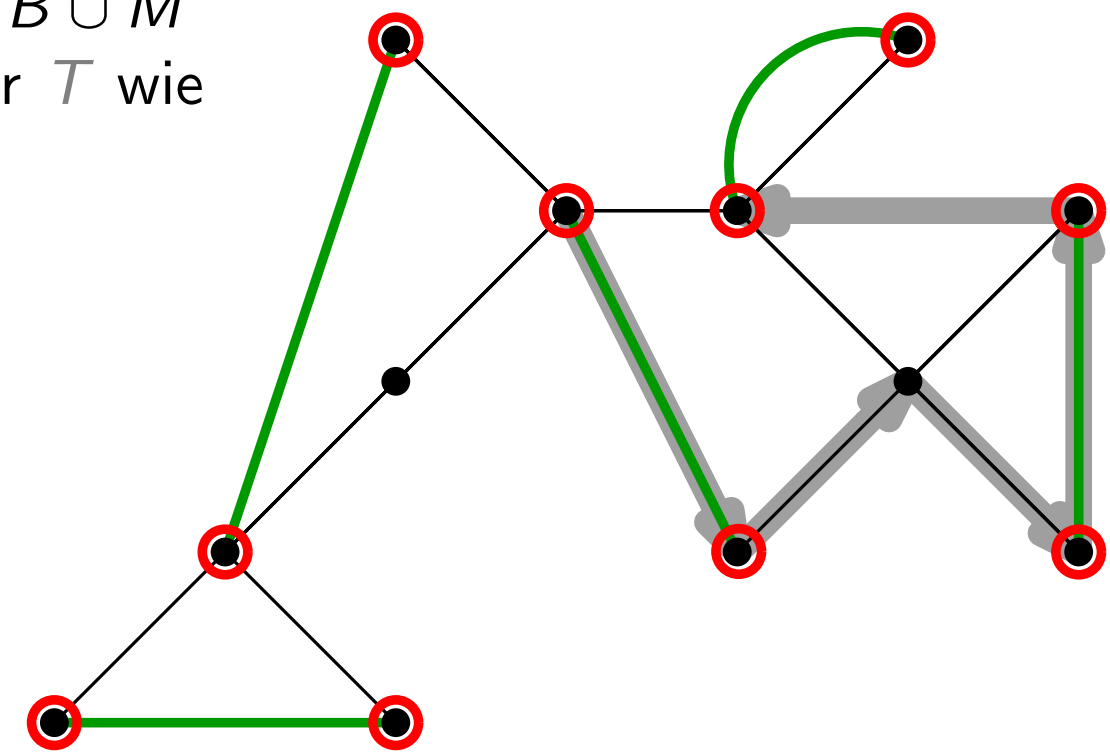
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

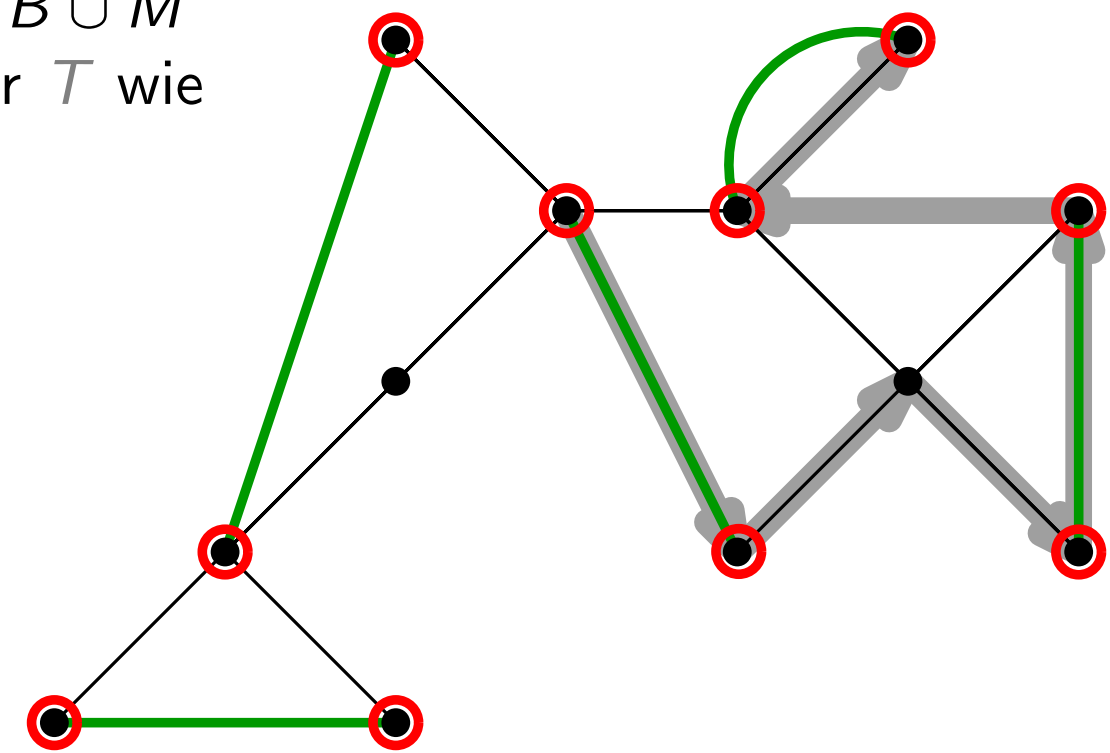
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

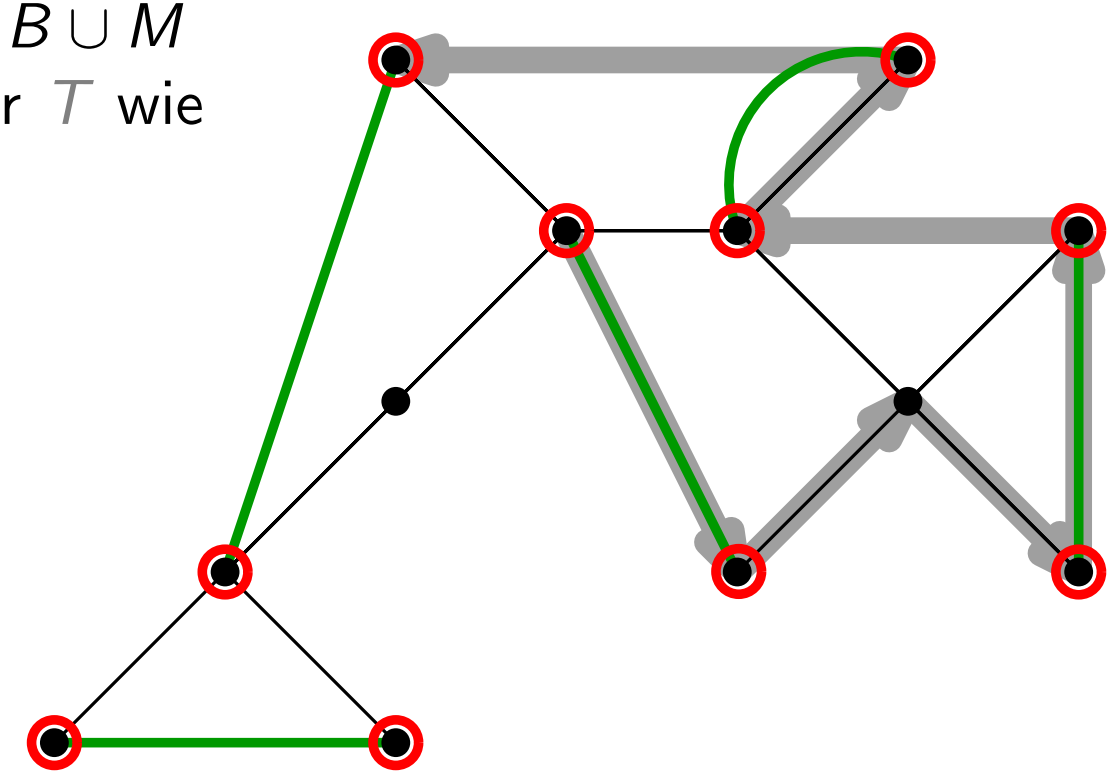
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

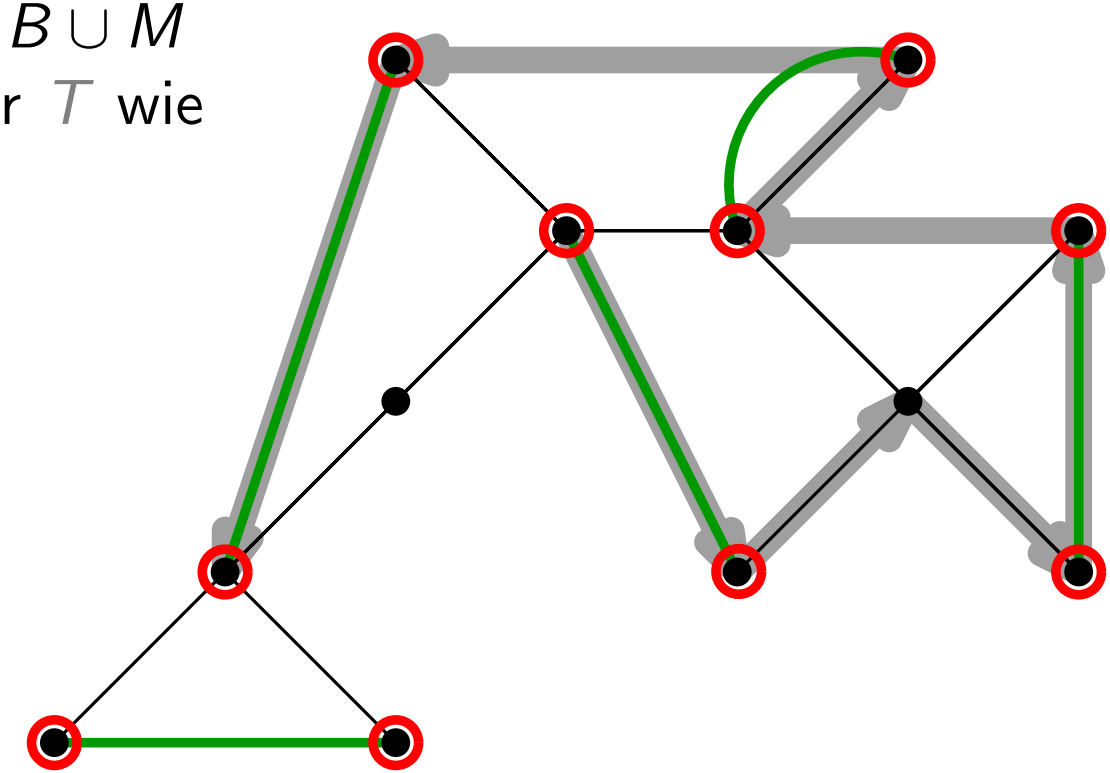
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

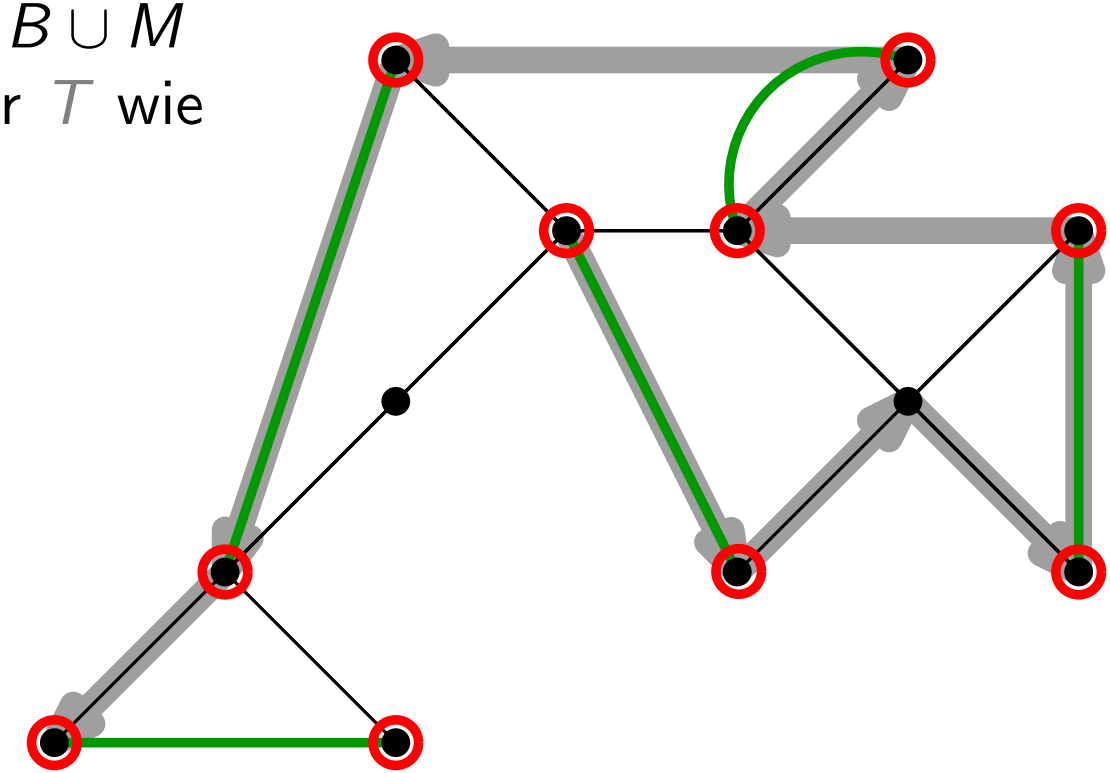
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

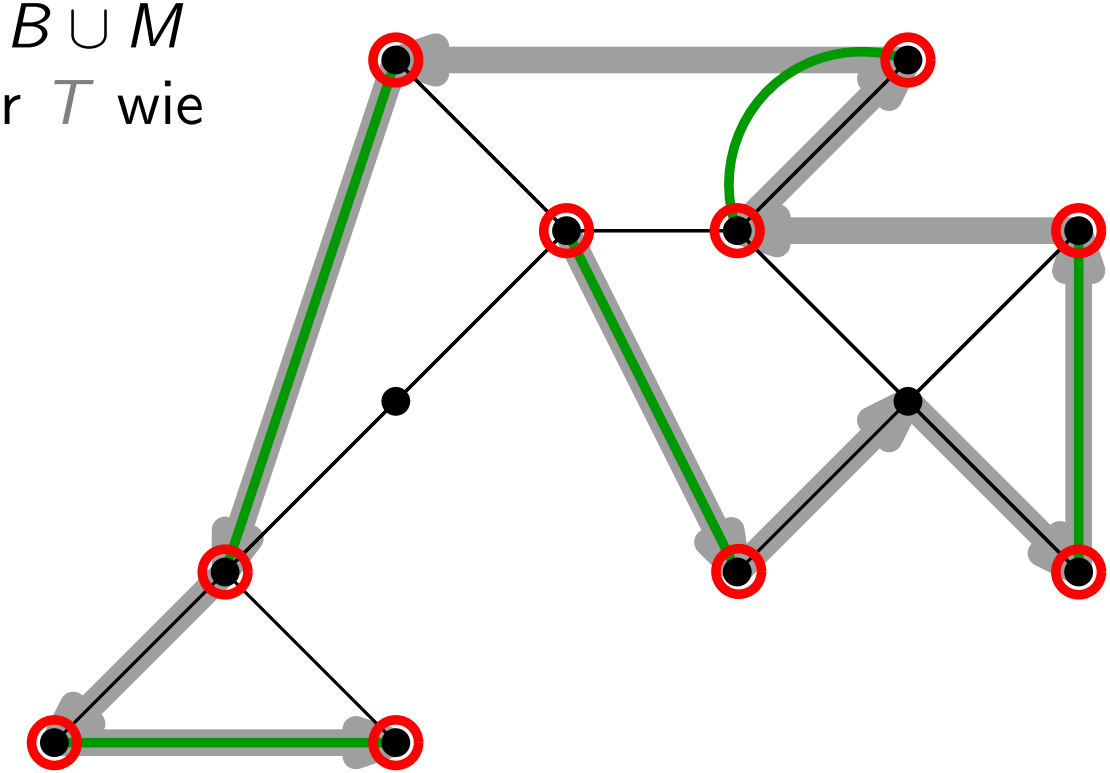
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

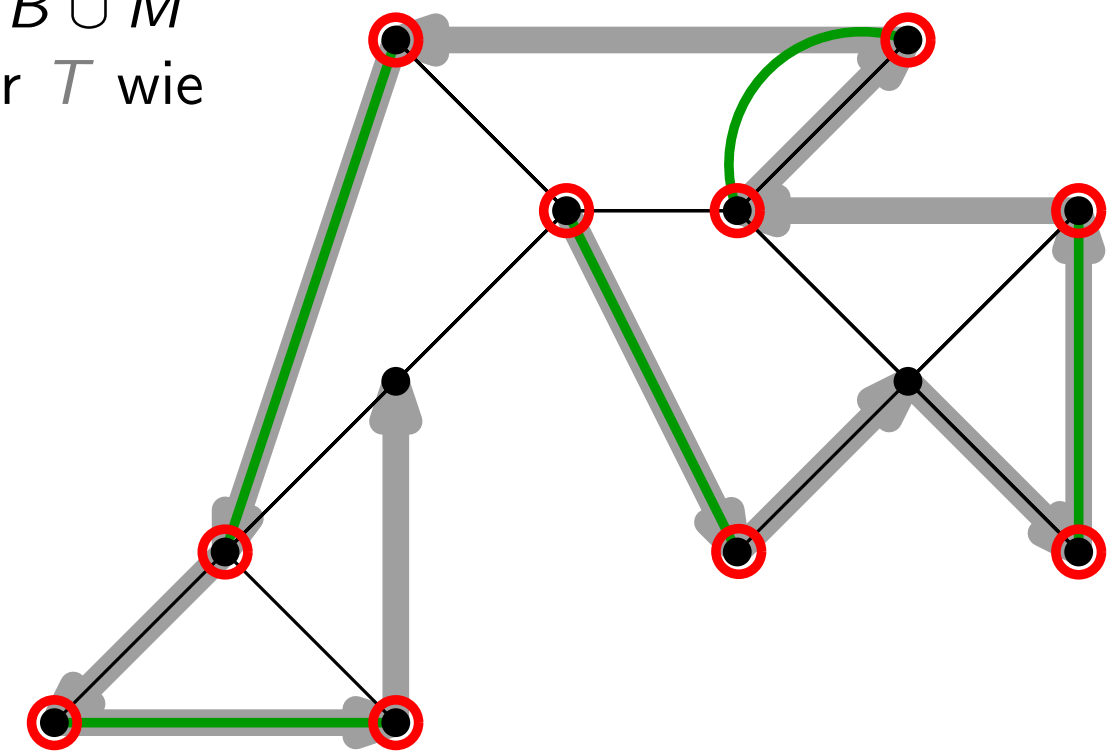
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

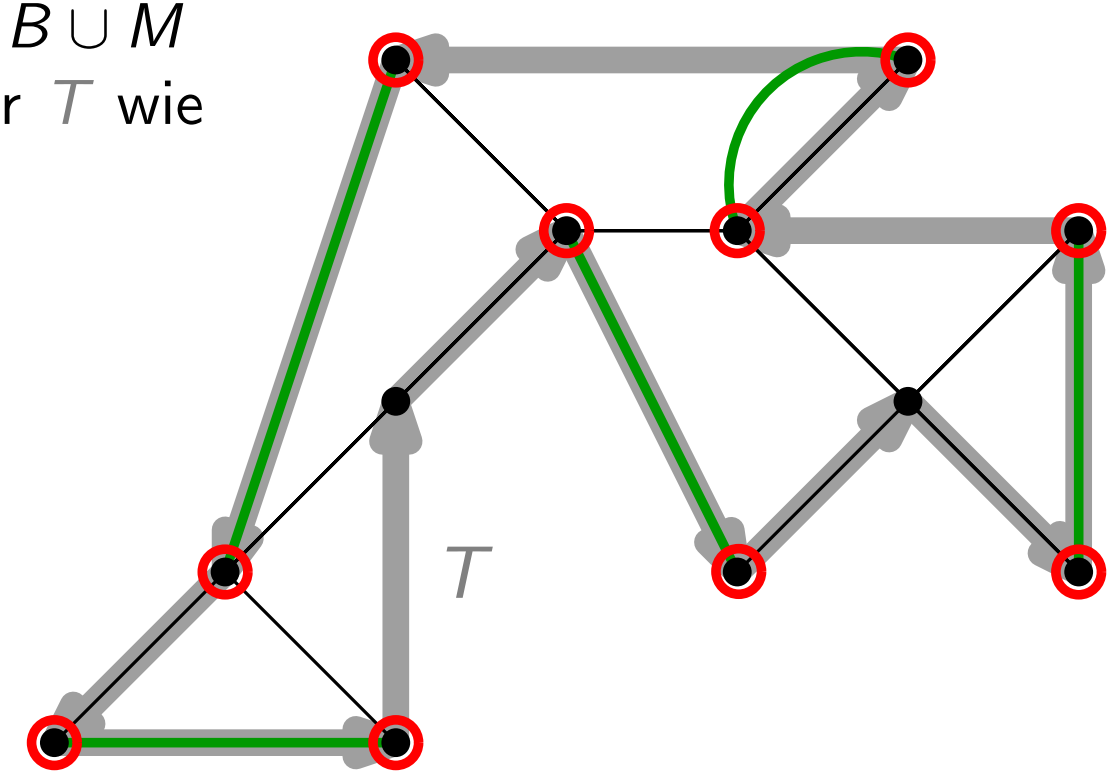
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Christofides' Algorithmus

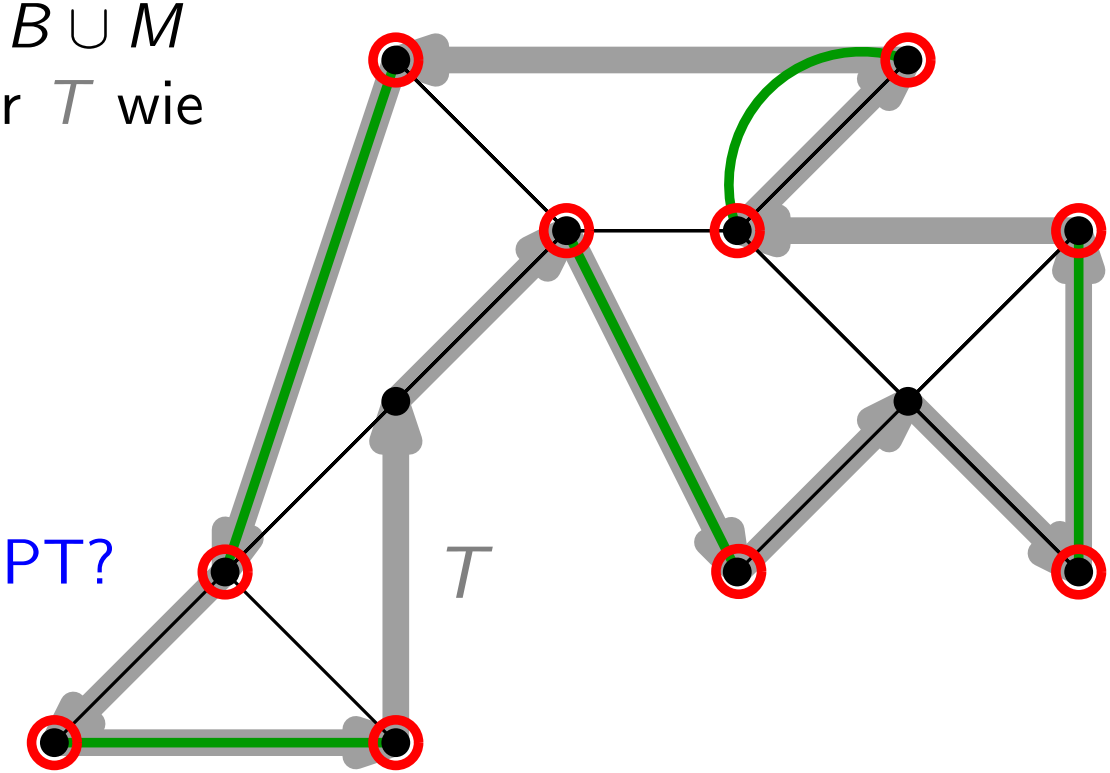
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Wie gut ist T im Vergleich zu OPT?

Christofides' Algorithmus

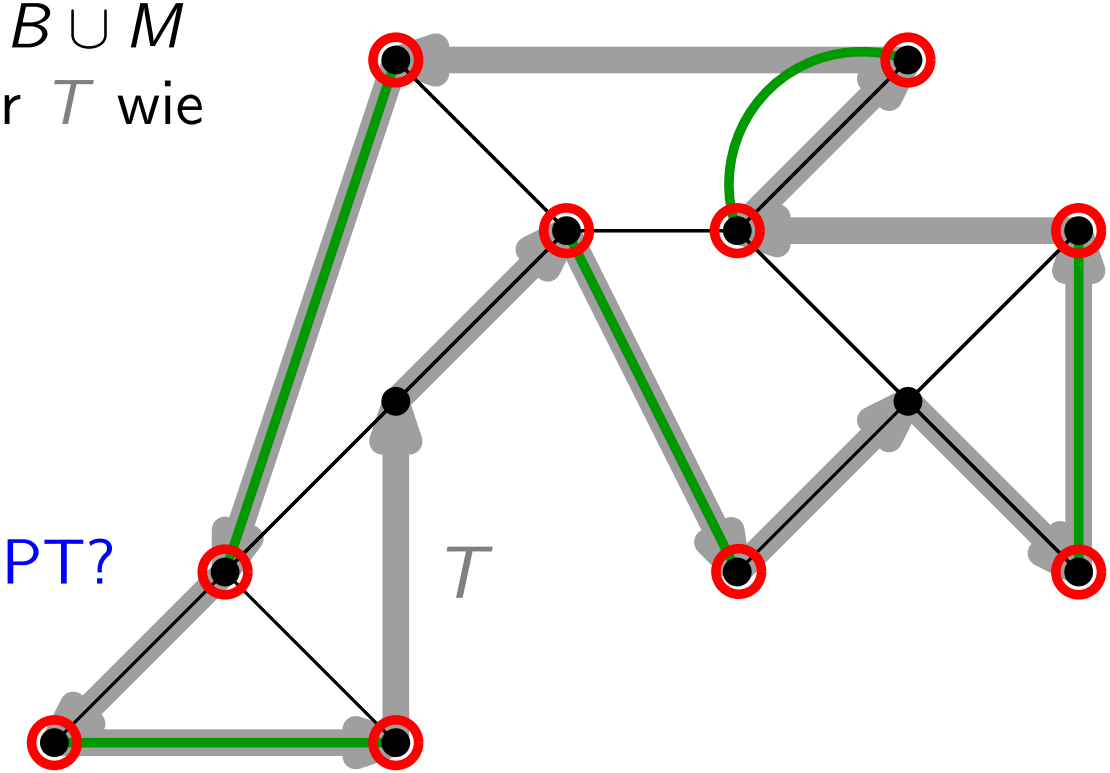
Gegeben: vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$

Bestimme minimalen Spannbaum B für G.

Sei U die Menge der Knoten ungeraden Grades in B.

Bestimme kostenminimales perfektes Matching M für $G[U]$

Berechne im eulerschen Graphen $B \cup M$ erst Eulertour und dann Rundtour T wie bei Tree-Doubling.



Wie gut ist T im Vergleich zu OPT?

Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathsf{OPT}}$, da $w(T) \leq w(B \cup M)$.

Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathrm{OPT}}$, da $w(T) \leq w(B \cup M)$.

Da $w(B) \leq w_{\mathrm{OPT}}$, genügt es zu zeigen, dass $w(M) \leq w_{\mathrm{OPT}}/2$.

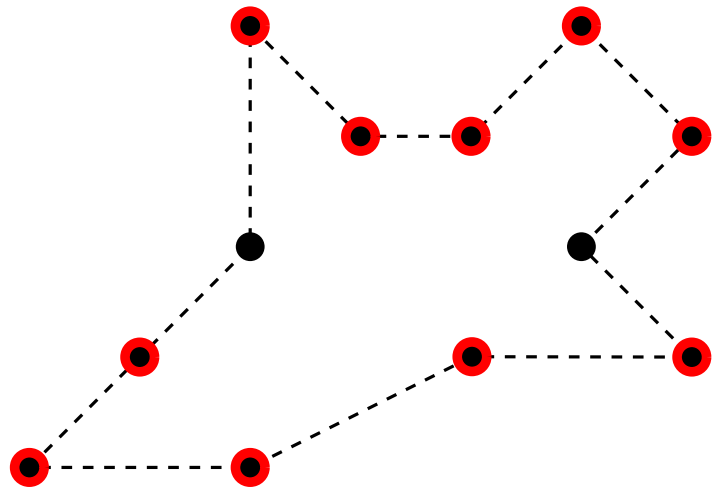
Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathrm{OPT}}$, da $w(T) \leq w(B \cup M)$.

Da $w(B) \leq w_{\mathrm{OPT}}$, genügt es zu zeigen, dass $w(M) \leq w_{\mathrm{OPT}}/2$.

- Sei T^* eine optimale Tour.



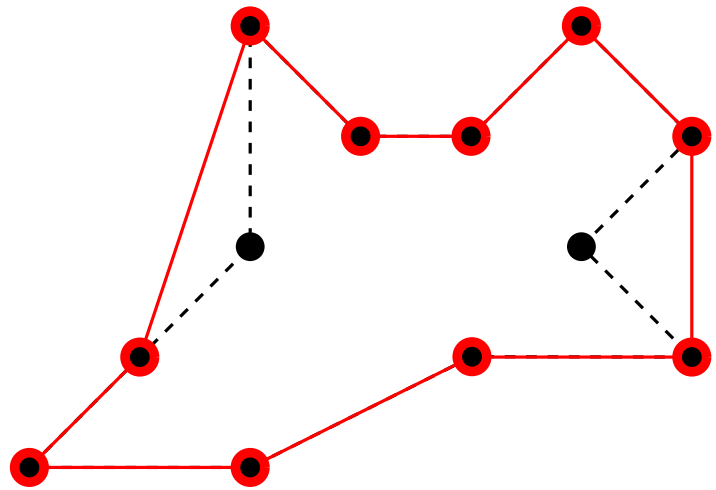
Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathrm{OPT}}$, da $w(T) \leq w(B \cup M)$.

Da $w(B) \leq w_{\mathrm{OPT}}$, genügt es zu zeigen, dass $w(M) \leq w_{\mathrm{OPT}}/2$.

- Sei T^* eine optimale Tour.
- Abkürzen (wenn möglich) von T^* liefert Tour T^{**} in $G[U]$ mit $w(T^{**}) \leq w(T^*) = w_{\mathrm{OPT}}$.



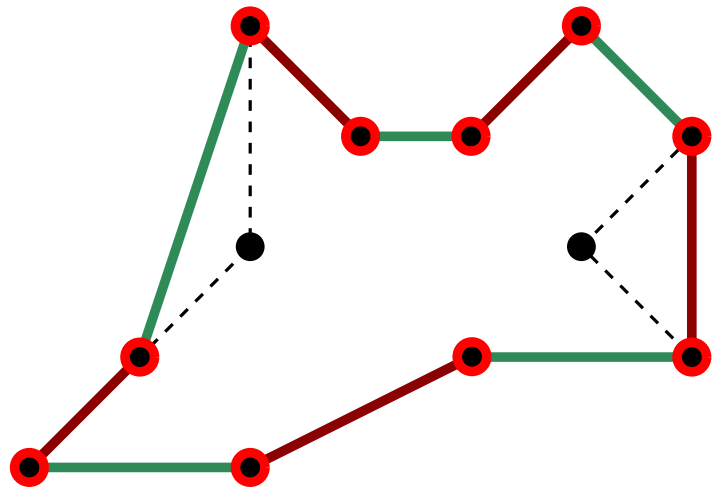
Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathrm{OPT}}$, da $w(T) \leq w(B \cup M)$.

Da $w(B) \leq w_{\mathrm{OPT}}$, genügt es zu zeigen, dass $w(M) \leq w_{\mathrm{OPT}}/2$.

- Sei T^* eine optimale Tour.
- Abkürzen (wenn möglich) von T^* liefert Tour T^{**} in $G[U]$ mit $w(T^{**}) \leq w(T^*) = w_{\mathrm{OPT}}$.
- Zerlege in T^{**} in zwei disjunkte perfekte Matchings M', M'' für $G[U]$.



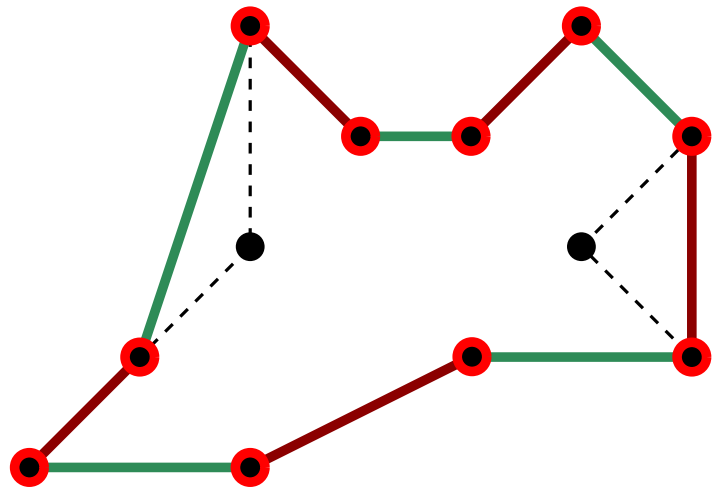
Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathrm{OPT}}$, da $w(T) \leq w(B \cup M)$.

Da $w(B) \leq w_{\mathrm{OPT}}$, genügt es zu zeigen, dass $w(M) \leq w_{\mathrm{OPT}}/2$.

- Sei T^* eine optimale Tour.
- Abkürzen (wenn möglich) von T^* liefert Tour T^{**} in $G[U]$ mit $w(T^{**}) \leq w(T^*) = w_{\mathrm{OPT}}$.
- Zerlege in T^{**} in zwei disjunkte perfekte Matchings M', M'' für $G[U]$.
- O.B.d.A. $w(M') \leq w(M'')$.



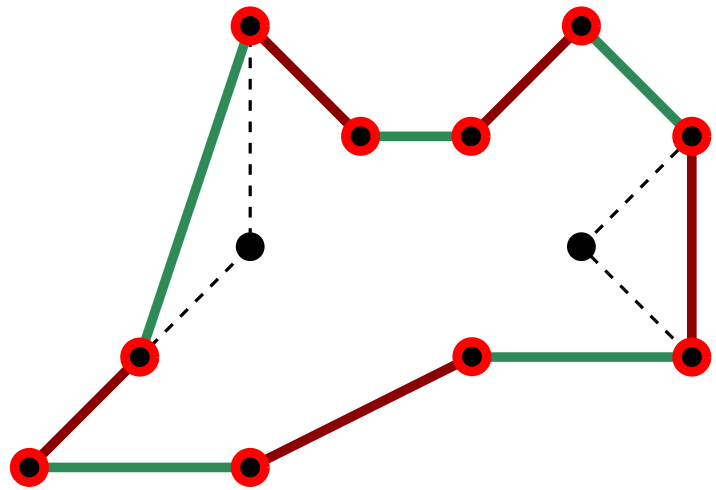
Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathrm{OPT}}$, da $w(T) \leq w(B \cup M)$.

Da $w(B) \leq w_{\mathrm{OPT}}$, genügt es zu zeigen, dass $w(M) \leq w_{\mathrm{OPT}}/2$.

- Sei T^* eine optimale Tour.
- Abkürzen (wenn möglich) von T^* liefert Tour T^{**} in $G[U]$ mit $w(T^{**}) \leq w(T^*) = w_{\mathrm{OPT}}$.
- Zerlege in T^{**} in zwei disjunkte perfekte Matchings M', M'' für $G[U]$.
- O.B.d.A. $w(M') \leq w(M'')$.
- Also gilt $w(M') \leq w(T^{**})/2 \leq f_{\mathrm{OPT}}/2$.



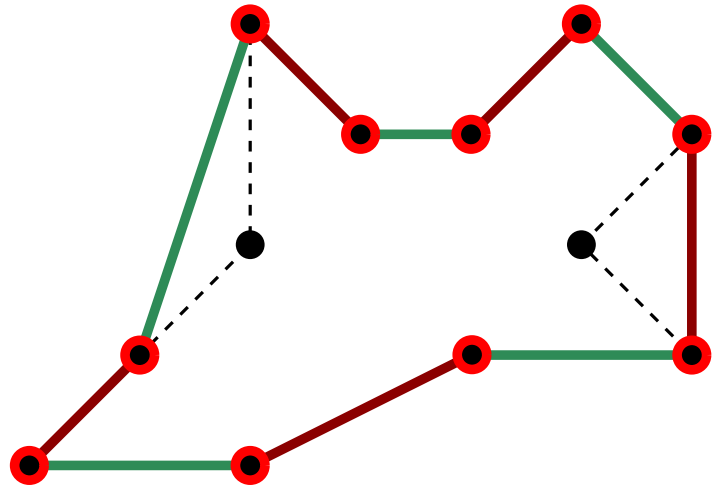
Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathrm{OPT}}$, da $w(T) \leq w(B \cup M)$.

Da $w(B) \leq w_{\mathrm{OPT}}$, genügt es zu zeigen, dass $w(M) \leq w_{\mathrm{OPT}}/2$.

- Sei T^* eine optimale Tour.
- Abkürzen (wenn möglich) von T^* liefert Tour T^{**} in $G[U]$ mit $w(T^{**}) \leq w(T^*) = w_{\mathrm{OPT}}$.
- Zerlege in T^{**} in zwei disjunkte perfekte Matchings M', M'' für $G[U]$.
- O.B.d.A. $w(M') \leq w(M'')$.
- Also gilt $w(M') \leq w(T^{**})/2 \leq f_{\mathrm{OPT}}/2$.
- Außerdem gilt $w(M) \leq w(M')$, da



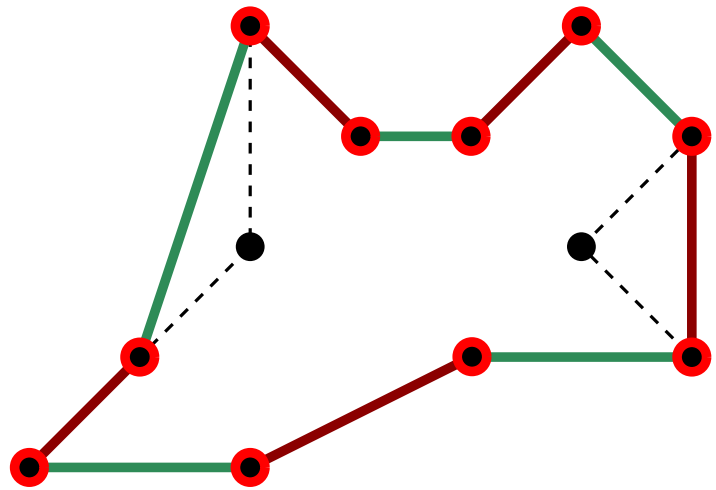
Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathrm{OPT}}$, da $w(T) \leq w(B \cup M)$.

Da $w(B) \leq w_{\mathrm{OPT}}$, genügt es zu zeigen, dass $w(M) \leq w_{\mathrm{OPT}}/2$.

- Sei T^* eine optimale Tour.
- Abkürzen (wenn möglich) von T^* liefert Tour T^{**} in $G[U]$ mit $w(T^{**}) \leq w(T^*) = w_{\mathrm{OPT}}$.
- Zerlege in T^{**} in zwei disjunkte perfekte Matchings M', M'' für $G[U]$.
- O.B.d.A. $w(M') \leq w(M'')$.
- Also gilt $w(M') \leq w(T^{**})/2 \leq f_{\mathrm{OPT}}/2$.
- Außerdem gilt $w(M) \leq w(M')$, da
  - M' perfektes Matching in $G[U]$



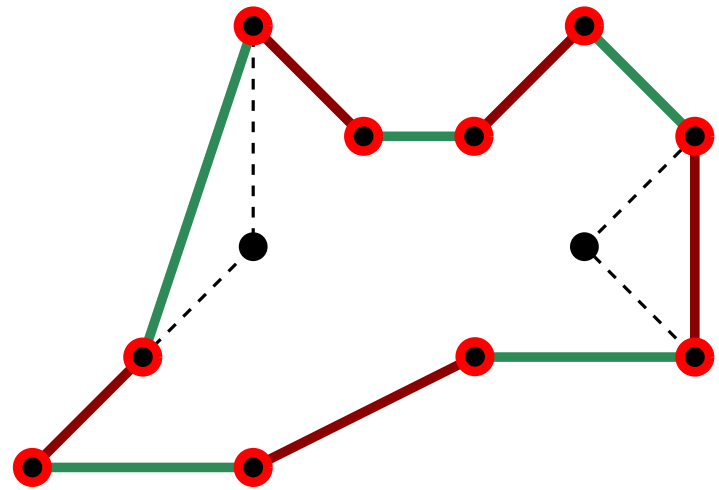
Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathrm{OPT}}$, da $w(T) \leq w(B \cup M)$.

Da $w(B) \leq w_{\mathrm{OPT}}$, genügt es zu zeigen, dass $w(M) \leq w_{\mathrm{OPT}}/2$.

- Sei T^* eine optimale Tour.
- Abkürzen (wenn möglich) von T^* liefert Tour T^{**} in $G[U]$ mit $w(T^{**}) \leq w(T^*) = w_{\mathrm{OPT}}$.
- Zerlege in T^{**} in zwei disjunkte perfekte Matchings M', M'' für $G[U]$.
- O.B.d.A. $w(M') \leq w(M'')$.
- Also gilt $w(M') \leq w(T^{**})/2 \leq f_{\mathrm{OPT}}/2$.
- Außerdem gilt $w(M) \leq w(M')$, da
  - M' perfektes Matching in $G[U]$
  - M *kostenminimales* perfektes Matching in $G[U]$



Analyse von Christofides' Algorithmus

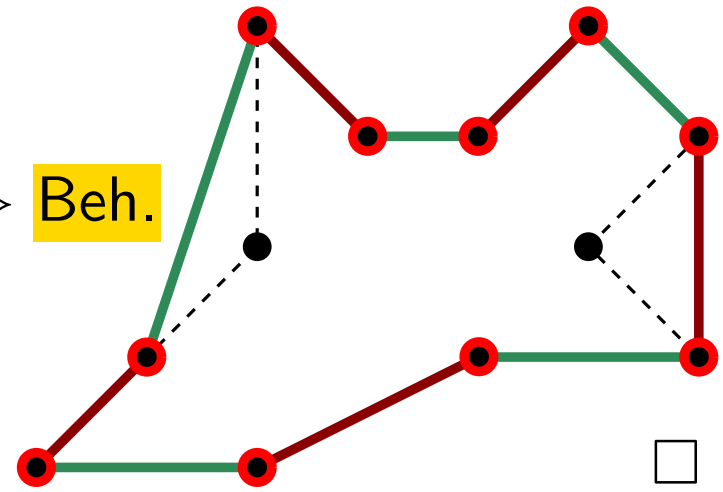
Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathrm{OPT}}$, da $w(T) \leq w(B \cup M)$.

Da $w(B) \leq w_{\mathrm{OPT}}$, genügt es zu zeigen, dass $w(M) \leq w_{\mathrm{OPT}}/2$.

- Sei T^* eine optimale Tour.
- Abkürzen (wenn möglich) von T^* liefert Tour T^{**} in $G[U]$ mit $w(T^{**}) \leq w(T^*) = w_{\mathrm{OPT}}$.
- Zerlege in T^{**} in zwei disjunkte perfekte Matchings M', M'' für $G[U]$.
- O.B.d.A. $w(M') \leq w(M'')$.
- Also gilt $w(M') \leq w(T^{**})/2 \leq f_{\mathrm{OPT}}/2$.
- Außerdem gilt $w(M) \leq w(M')$, da
  - M' perfektes Matching in $G[U]$
  - M *kostenminimales* perfektes Matching in $G[U]$

$\Rightarrow$ Beh.



□

Analyse von Christofides' Algorithmus

Christofides' Algorithmus liefert eine 3/2-Approximation für das TSP.

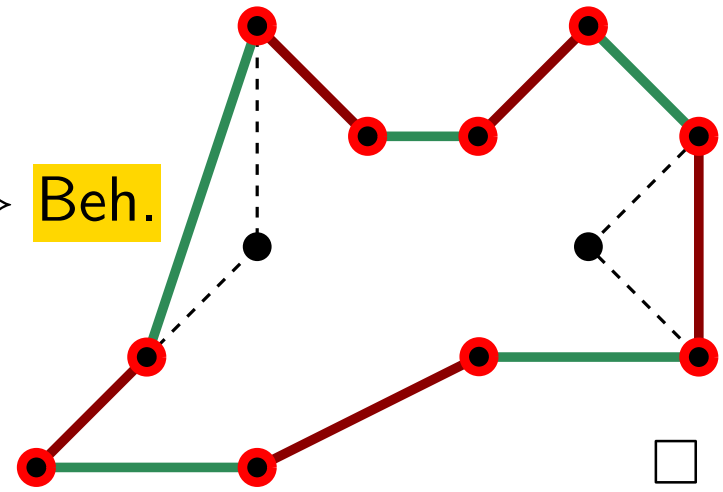
Beweis: Es genügt zu zeigen, dass $w(B \cup M) \leq 3/2 \cdot f_{\mathrm{OPT}}$, da $w(T) \leq w(B \cup M)$.

Da $w(B) \leq w_{\mathrm{OPT}}$, genügt es zu zeigen, dass $w(M) \leq w_{\mathrm{OPT}}/2$.

- Sei T^* eine optimale Tour.
- Abkürzen (wenn möglich) von T^* liefert Tour T^{**} in $G[U]$ mit $w(T^{**}) \leq w(T^*) = w_{\mathrm{OPT}}$.
- Zerlege in T^{**} in zwei disjunkte perfekte Matchings M', M'' für $G[U]$.
- O.B.d.A. $w(M') \leq w(M'')$.
- Also gilt $w(M') \leq w(T^{**})/2 \leq f_{\mathrm{OPT}}/2$.
- Außerdem gilt $w(M) \leq w(M')$, da
  - M' perfektes Matching in $G[U]$
  - M *kostenminimales* perfektes Matching in $G[U]$

$\Rightarrow$ Beh.

Und wenn die Dreiecksungleichung nicht gilt?



□

Varianten des Handlungsreisendenproblems

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Varianten des Handlungsreisendenproblems

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

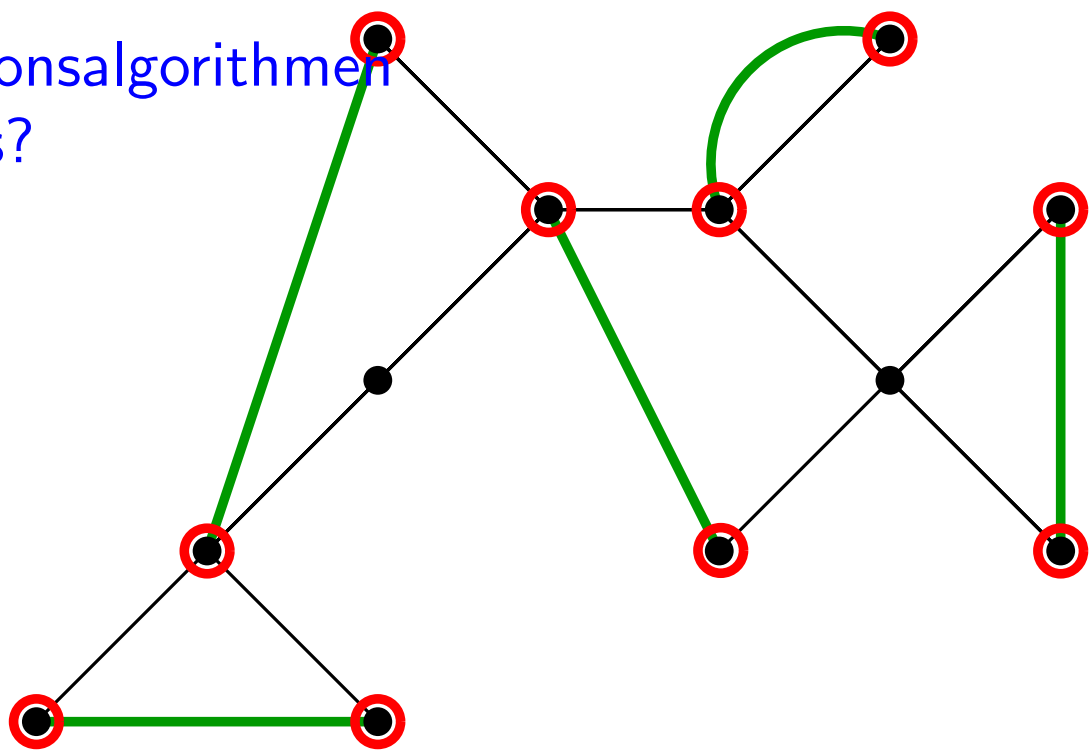
Funktionieren unsere Approximationsalgorithmen
für das ein-Besuchs-TSP ebenfalls?

Varianten des Handlungsreisendenproblems

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Funktionieren unsere Approximationsalgorithmen für das ein-Besuchs-TSP ebenfalls?

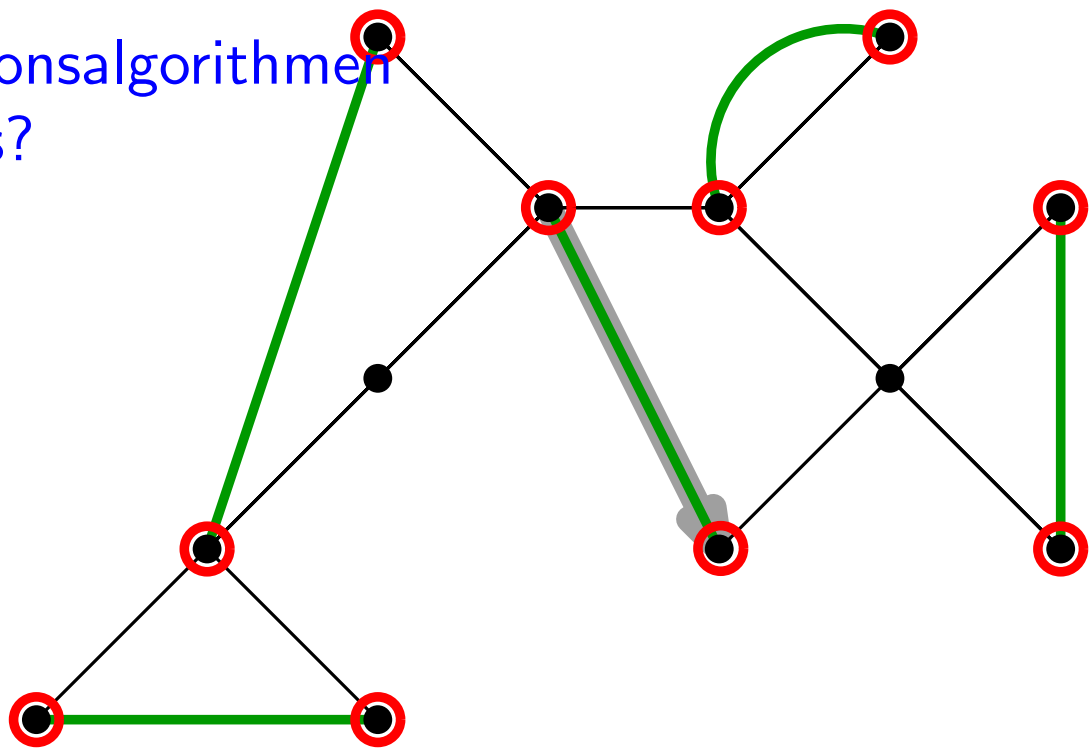


Varianten des Handlungsreisendenproblems

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Funktionieren unsere Approximationsalgorithmen für das ein-Besuchs-TSP ebenfalls?

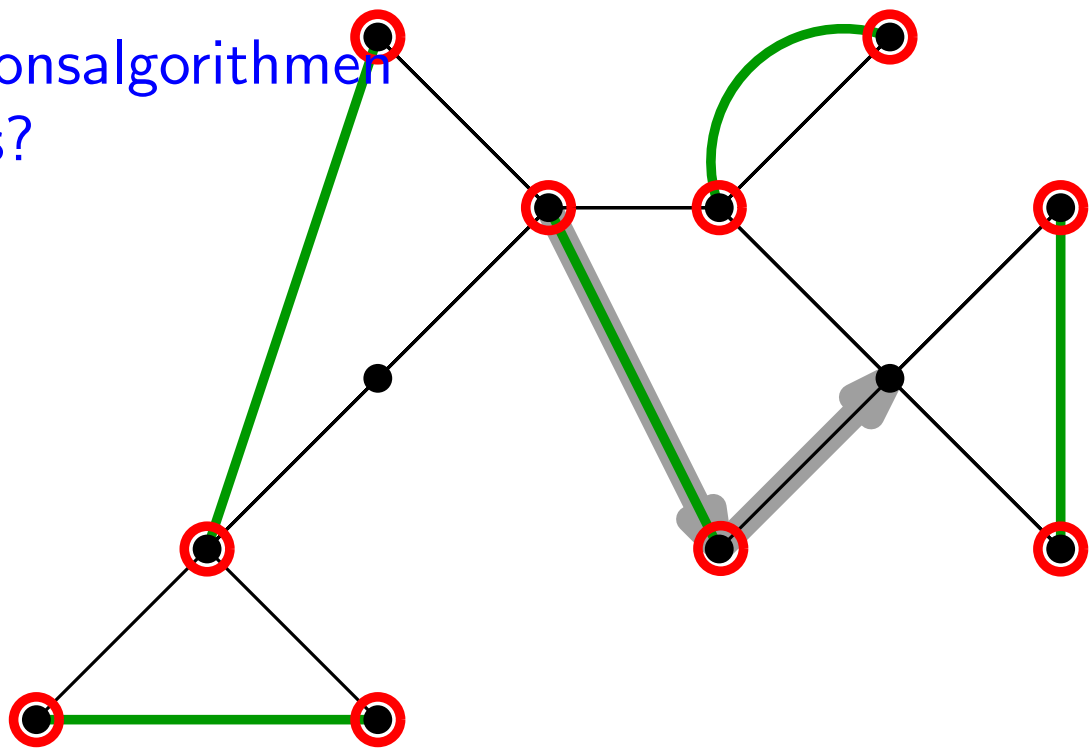


Varianten des Handlungsreisendenproblems

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Funktionieren unsere Approximationsalgorithmen für das ein-Besuchs-TSP ebenfalls?

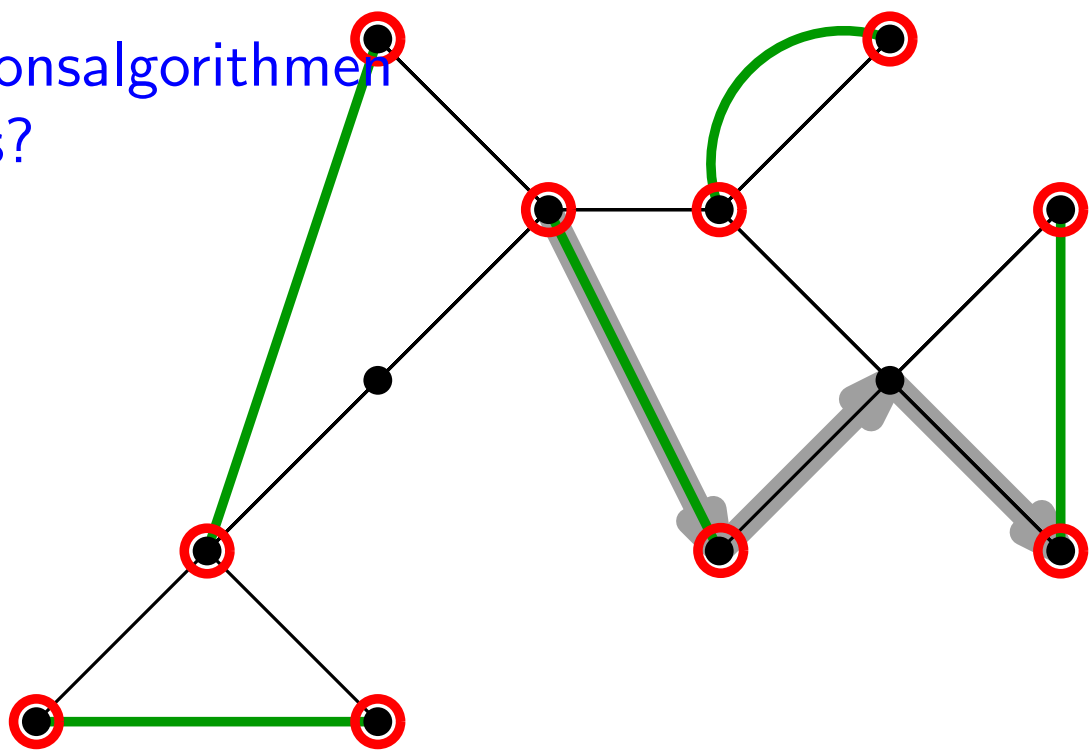


Varianten des Handlungsreisendenproblems

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Funktionieren unsere Approximationsalgorithmen für das ein-Besuchs-TSP ebenfalls?

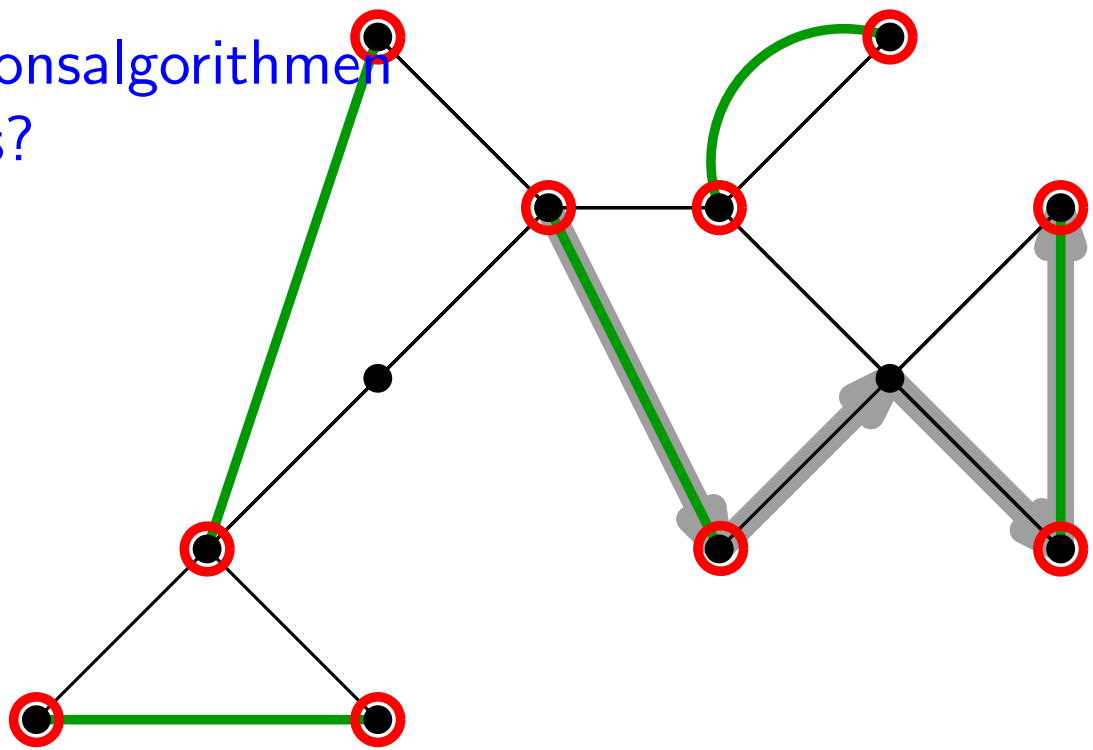


Varianten des Handlungsreisendenproblems

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Funktionieren unsere Approximationsalgorithmen für das ein-Besuchs-TSP ebenfalls?

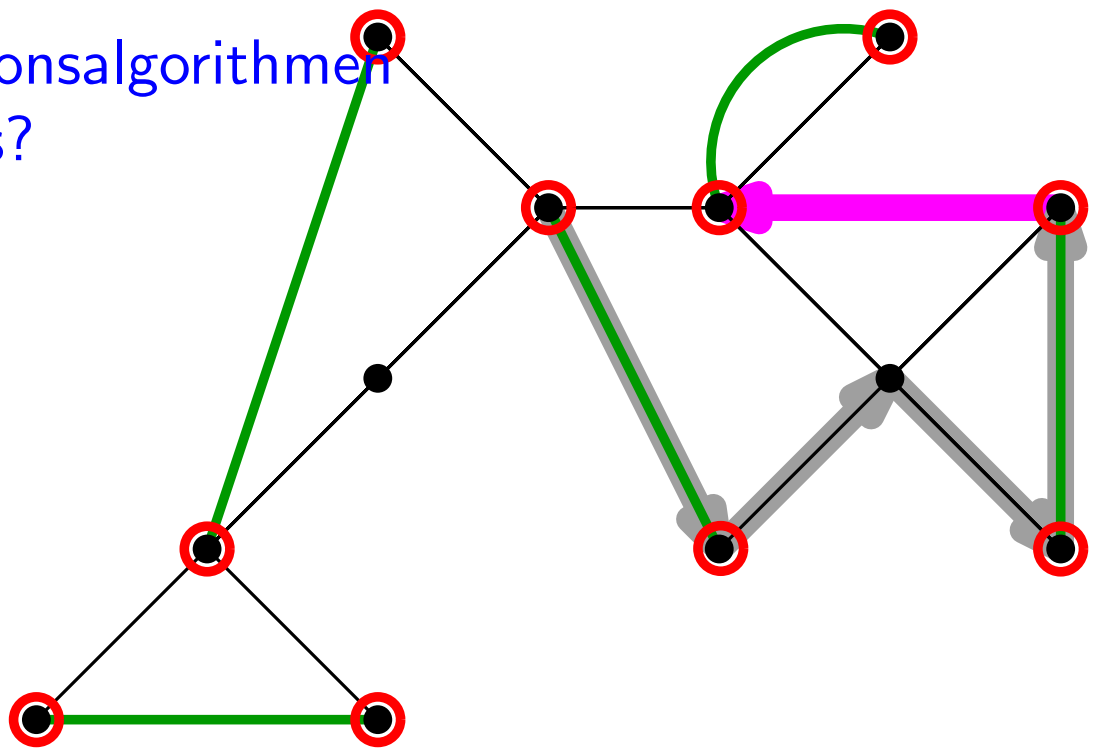


Varianten des Handlungsreisendenproblems

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Funktionieren unsere Approximationsalgorithmen für das ein-Besuchs-TSP ebenfalls?



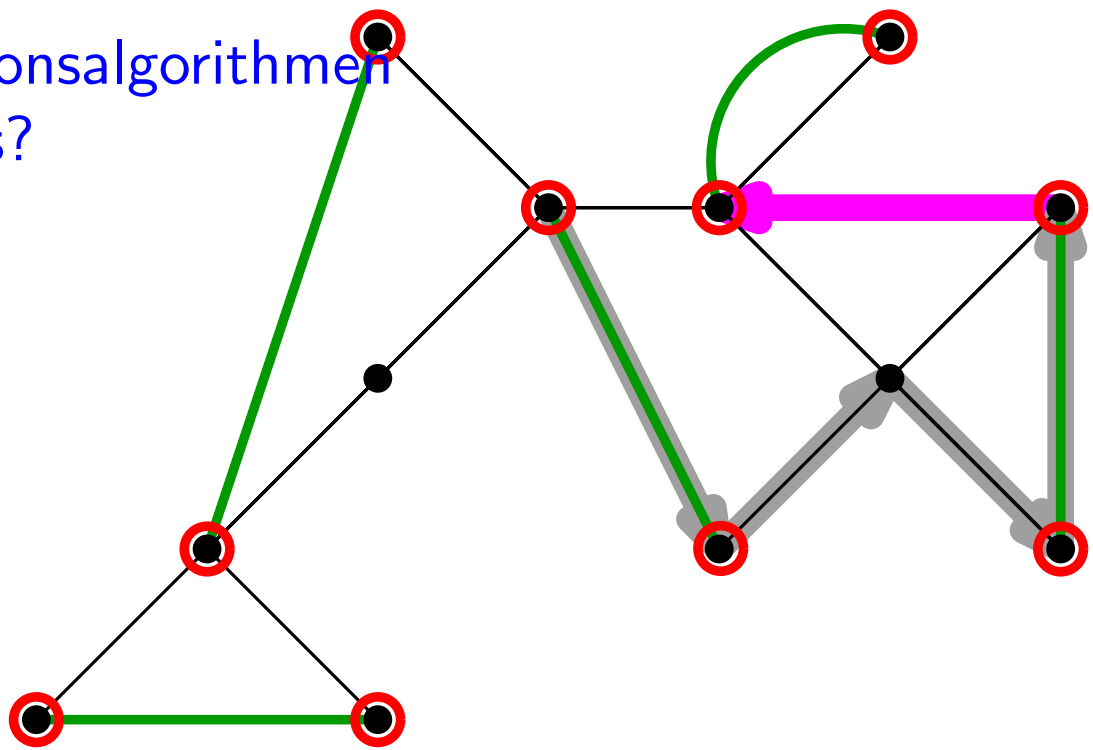
Varianten des Handlungsreisendenproblems

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Funktionieren unsere Approximationsalgorithmen für das ein-Besuchs-TSP ebenfalls?

Ja, falls $w_{u,w} \leq w_{u,v} + w_{v,w}$ für alle Knoten $u, v, w \in V$



Varianten des Handlungsreisendenproblems

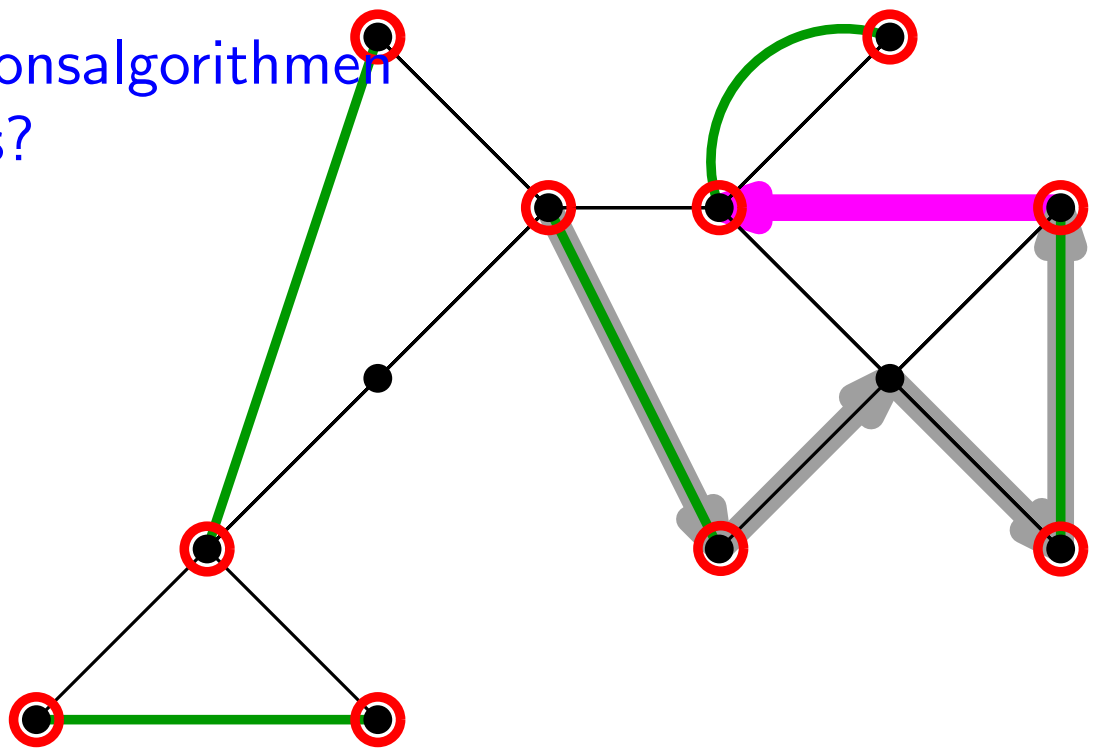
Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Funktionieren unsere Approximationsalgorithmen für das ein-Besuchs-TSP ebenfalls?

Ja, falls $w_{u,w} \leq w_{u,v} + w_{v,w}$ für alle Knoten $u, v, w \in V$

Diese Eigenschaft heißt '**Dreiecksungleichung**'.



Varianten des Handlungsreisendenproblems

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

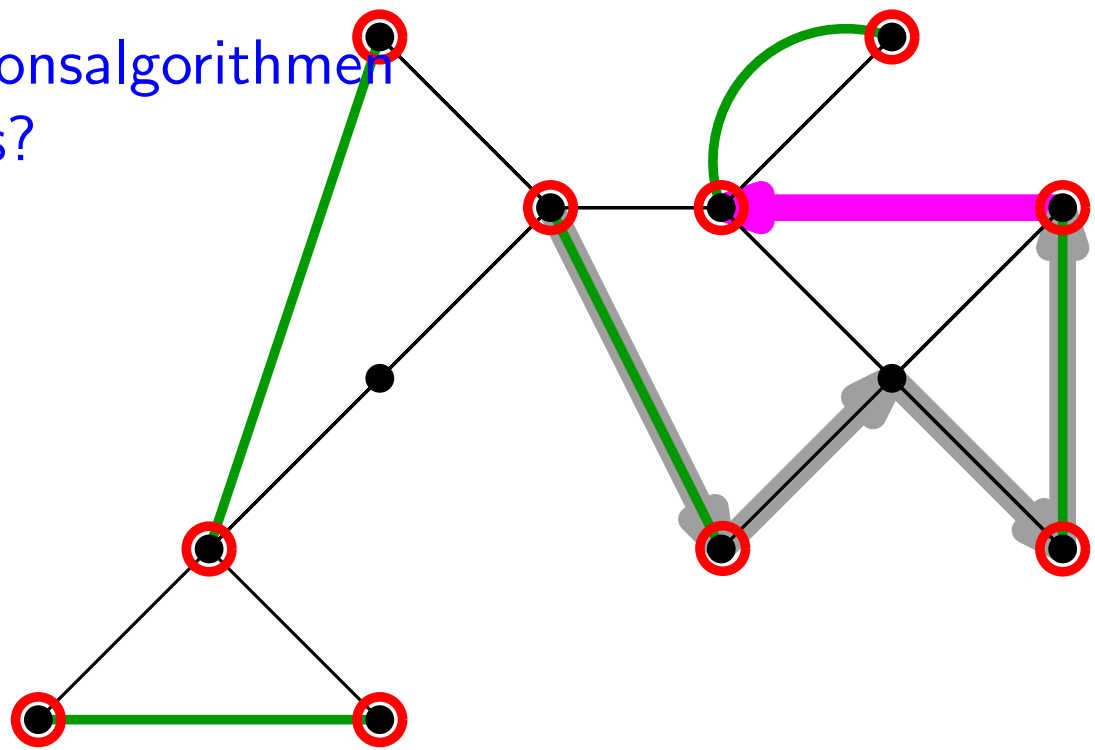
Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Funktionieren unsere Approximationsalgorithmen für das ein-Besuchs-TSP ebenfalls?

Ja, falls $w_{u,w} \leq w_{u,v} + w_{v,w}$ für alle Knoten $u, v, w \in V$

Diese Eigenschaft heißt '**Dreiecksungleichung**'.

Das ein-Besuch-TSP mit Dreiecksungleichung wird '**metrisches** TSP' genannt.



Varianten des Handlungsreisendenproblems

Handlungsreisendenproblem (TSP) (eine Variante):
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten besucht, so dass $\sum_{e \in E_K} w_e$ minimal

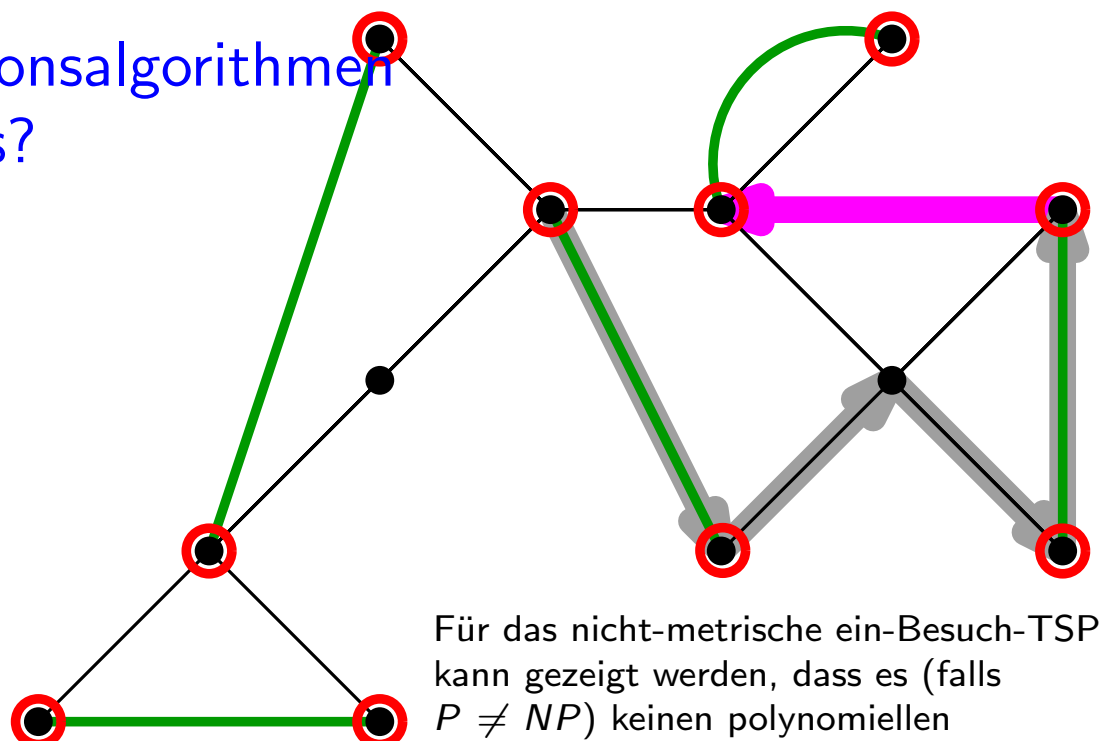
Handlungsreisendenproblem (ebenfalls übliche Variante) - hier: **ein-Besuch-TSP**
Gegeben: Vollständiger Graph $G = (V, E)$, Kantenlabels w_e für $e \in E$
Gesucht: Ein Kreis $K = (V_K, E_K)$, der alle Knoten genau einmal besucht, so dass $\sum_{e \in E_K} w_e$ minimal

Funktionieren unsere Approximationsalgorithmen für das ein-Besuchs-TSP ebenfalls?

Ja, falls $w_{u,w} \leq w_{u,v} + w_{v,w}$ für alle Knoten $u, v, w \in V$

Diese Eigenschaft heißt '**Dreiecksungleichung**'.

Das ein-Besuch-TSP mit Dreiecksungleichung wird '**metrisches** TSP' genannt.



Für das nicht-metrische ein-Besuch-TSP kann gezeigt werden, dass es (falls $P \neq NP$) keinen polynomiellen Approximationsalgorithmus gibt.

Algorithmen für NP-schwere Probleme

Was machen wir, wenn wir trotzdem ein NP-schweres Problem lösen müssen?

Idee 1: Wir finden (schnell/effizient) eine *gute* Lösung:
- Heuristiken
- Approximationsalgorithmen

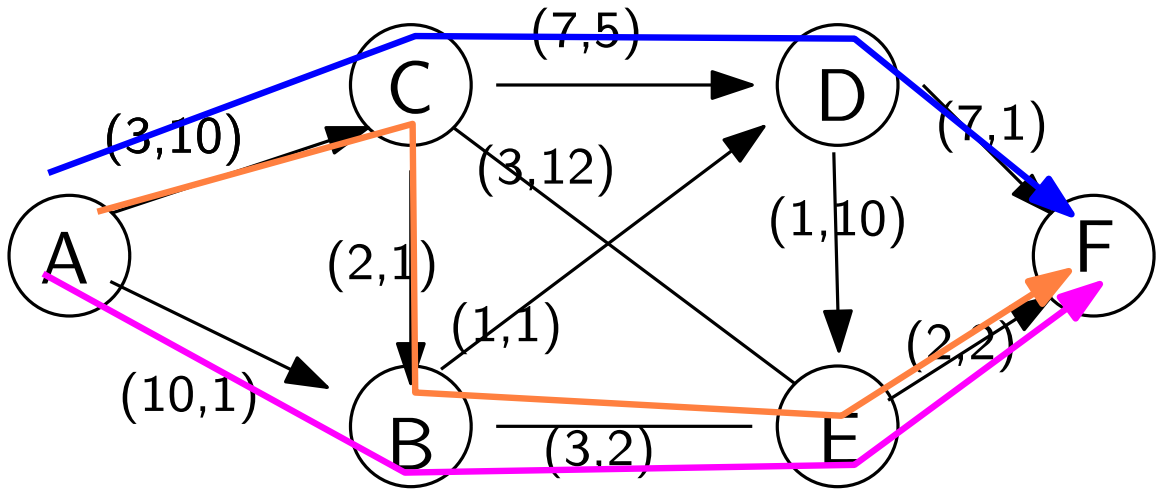
Idee 2: Wir nutzen ein Lösungsverfahren mit exponentieller Laufzeit, z.B.
- Brute Force
- Branch-and-Bound (evtl. 2. Vorlesungshälfte)
- lineare ganzzahlige Programmierung (2. Vorlesungshälfte)

Idee 3: Vielleicht ist unser konkretes Problem ein Spezialfall für den es polynomiellen Algorithmus gibt? (Bsp: chordaler Graph beim Färbungsproblem)

Idee 4: Wir nutzen ein (eigentlich exponentielles) Lösungsverfahren, das für *spezielle Probleminstanzen* doch effizient ist:
- pseudopolynomielle Algorithmen
- Festparameter-Berechenbarkeit ← jetzt

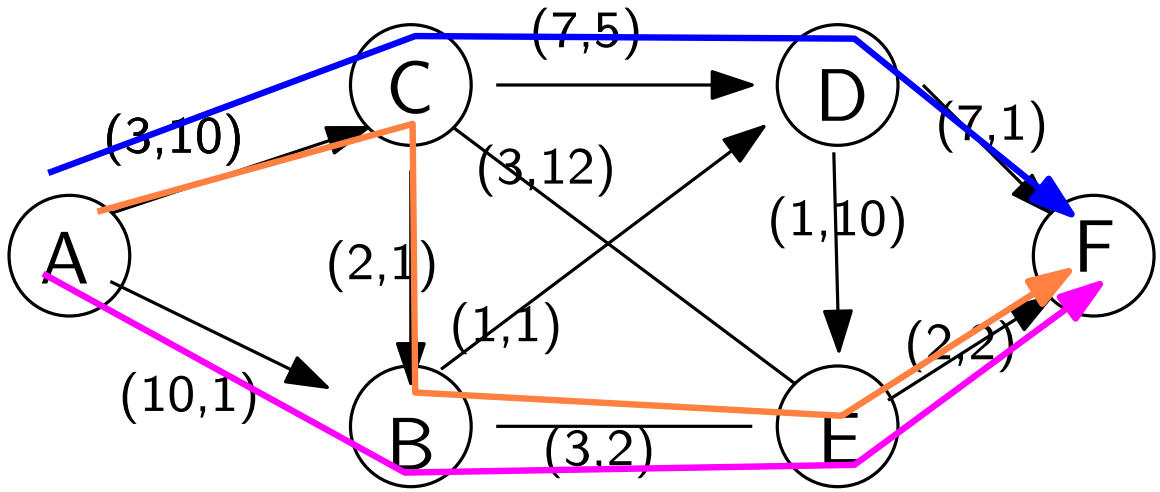
Idee 5: randomisierte Algorithmen & average-case Analyse

Entscheidungsproblem: bikriteriell kürzeste Wege



Optimierungsproblem:bikriteriell kürzeste-Wege
Gegeben: ein (gerichteter oder ungerichteter) Graph mit 2 Kantenlabels w_e^i auf jeder Kante e, Startknoten s und Zielknoten t
Gesucht: Menge alles **nichtdominierten** s-t-Wege

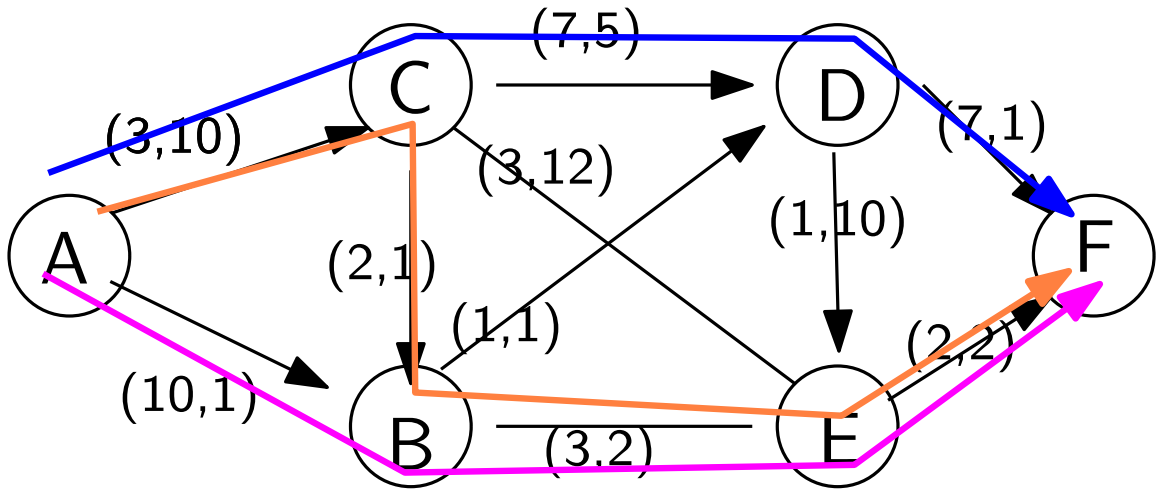
Entscheidungsproblem: bikriteriell kürzeste Wege



Optimierungsproblem:bikriteriell kürzeste-Wege
Gegeben: ein (gerichteter oder ungerichteter) Graph mit 2 Kantenlabels w_e^i auf jeder Kante e, Startknoten s und Zielknoten t
Gesucht: Menge alles **nichtdominierten** s-t-Wege

Entscheidungsproblem: multikriterielle kürzeste-Wege
Gegeben: ein (gerichteter oder ungerichteter) Graph mit 2 Kantenlabels w_e^i auf jeder Kante e, Startknoten s und Zielknoten t

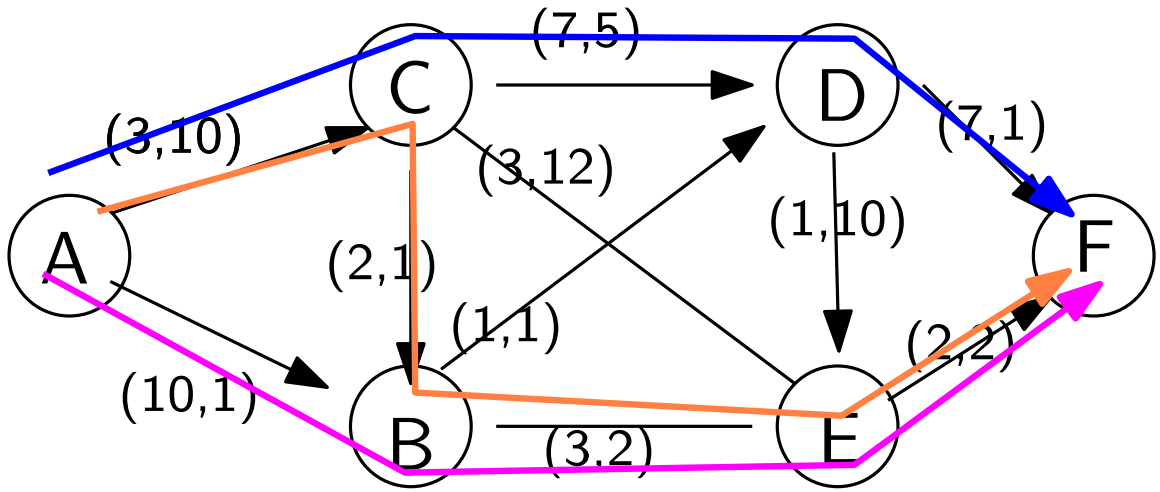
Entscheidungsproblem: bikriteriell kürzeste Wege



Optimierungsproblem:bikriteriell kürzeste-Wege
Gegeben: ein (gerichteter oder ungerichteter) Graph mit 2 Kantenlabels w_e^i auf jeder Kante e, Startknoten s und Zielknoten t
Gesucht: Menge alles **nichtdominierten** s-t-Wege

Entscheidungsproblem: multikriterielle kürzeste-Wege
Gegeben: ein (gerichteter oder ungerichteter) Graph mit 2 Kantenlabels w_e^i auf jeder Kante e, Startknoten s und Zielknoten t
Schranken W^1, W^2
Frage: Gibt es einen s-t-Weg mit $\sum_{e \in E} w_e^i \leq W^i$?

Entscheidungsproblem: bikriteriell kürzeste Wege



Optimierungsproblem:bikriteriell kürzeste-Wege
Gegeben: ein (gerichteter oder ungerichteter) Graph mit 2 Kantenlabels w_e^i auf jeder Kante e, Startknoten s und Zielknoten t
Gesucht: Menge alles **nichtdominierten** s-t-Wege

Entscheidungsproblem: multikriterielle kürzeste-Wege
Gegeben: ein (gerichteter oder ungerichteter) Graph mit 2 Kantenlabels w_e^i auf jeder Kante e, Startknoten s und Zielknoten t
Schranken W^1, W^2
Frage: Gibt es einen s-t-Weg mit $\sum_{e \in E} w_e^i \leq W^i$?

Man kann zeigen: Dieses EP ist NP-vollständig.

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Require: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit positiven Kantenlabels w_e^1, w_e^2 für $e \in E$, Startknoten s, Zielknoten t
Ensure: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)
Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ für $i = 1, 2$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$
while L_{open} nicht leer **do**
  wähle $l \in L_{open}$,
  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$
  for $v \in Adj(l.knoten)$ **do**
    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für $i = 1, 2$, $l'.\pi = l$
    Dominanzcheck$(l', L_{open}, L_{closed})$
  end for
end while
return Labels $l \in L_{closed}$ mit $l.knoten = t$

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Require: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit positiven Kantenlabels w_e^1, w_e^2 für $e \in E$, Startknoten s, Zielknoten t
Ensure: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)
Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ für $i = 1, 2$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$
while L_{open} nicht leer **do**
  wähle $l \in L_{open}$,
  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$
  for $v \in Adj(l.knoten)$ **do**
    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für $i = 1, 2$, $l'.\pi = l$
    Dominanzcheck$(l', L_{open}, L_{closed})$
  end for
end while
return Labels $l \in L_{closed}$ mit $l.knoten = t$

Label l repräsentiert einen Weg $p(l)$ von s zu einem Knoten u.
l setzt sich zusammen aus:
- $l.knoten = u$ - Endknoten des Weges
- d_i für $i = 1, 2$: 'Länge' des Weges bzgl Kantenlabels w_i
- (Pointer auf) Vorgängerlabel $l.\pi$

Label-Setting Algorithmus zum Finden der Menge der nicht-dominierten Wege

Require: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit positiven Kantenlabels w_e^1, w_e^2 für $e \in E$, Startknoten s, Zielknoten t
Ensure: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)
Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ für $i = 1, 2$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$
while L_{open} nicht leer **do**
  wähle $l \in L_{open}$,
  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$
  for $v \in Adj(l.knoten)$ **do**
    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für $i = 1, 2$, $l'.\pi = l$
    Dominanzcheck$(l', L_{open}, L_{closed})$
  end for
end while
return Labels $l \in L_{closed}$ mit $l.knoten = t$

Label l repräsentiert einen Weg $p(l)$ von s zu einem Knoten u.
l setzt sich zusammen aus:
- $l.knoten = u$ - Endknoten des Weges
- d_i für $i = 1, 2$: 'Länge' des Weges bzgl Kantenlabels w_i
- (Pointer auf) Vorgängerlabel $l.\pi$

Können wir mit diesem Algorithmus auch das Entscheidungsproblem lösen?

Label-Setting Algorithmus für das EP 'multikriterielle kürzeste Wege'

Require: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit positiven Kantenlabels w_e^1, w_e^2 für $e \in E$, Startknoten s, Zielknoten t, Schranken W^1, W^2
Ensure: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)
Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ für $i = 1, 2$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$
while L_{open} nicht leer **do**
  wähle $l \in L_{open}$,
  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$
  for $v \in Adj(l.knoten)$ **do**
    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für $i = 1, 2$, $l'.\pi = l$
    Dominanzcheck$(l', L_{open}, L_{closed})$
  end for
end while
return Labels $l \in L_{closed}$ mit $l.knoten = t$

Teste: Ist $l'.d_i \leq W^i$?

Können wir mit diesem Algorithmus auch das Entscheidungsproblem lösen?

Label-Setting Algorithmus für das EP 'multikriterielle kürzeste Wege'

Require: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit positiven Kantenlabels w_e^1, w_e^2 für $e \in E$, Startknoten s, Zielknoten t, Schranken W^1, W^2
Ensure: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)
Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ für $i = 1, 2$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$
while L_{open} nicht leer **do**
  wähle $l \in L_{open}$,
  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$
  for $v \in Adj(l.knoten)$ **do**
    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für $i = 1, 2$, $l'.\pi = l$
    Dominanzcheck$(l', L_{open}, L_{closed})$
  end for
end while
return Labels $l \in L_{closed}$ mit $l.knoten = t$

Teste: Ist $l'.d_i \leq W^i$?

Wie viele Labels werden (maximal) erstellt?

Label-Setting Algorithmus für das EP 'multikriterielle kürzeste Wege'

Require: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit positiven Kantenlabels w_e^1, w_e^2 für $e \in E$, Startknoten s, Zielknoten t, Schranken W^1, W^2
Ensure: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)
Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ für $i = 1, 2$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$
while L_{open} nicht leer **do**
  wähle $l \in L_{open}$,
  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$
  for $v \in Adj(l.knoten)$ **do**
    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für $i = 1, 2$, $l'.\pi = l$
    Dominanzcheck$(l', L_{open}, L_{closed})$
  end for
end while
return Labels $l \in L_{closed}$ mit $l.knoten = t$

Teste: Ist $l'.d_i \leq W^i$?

Wie viele Labels werden (maximal) erstellt?

Angenommen $w_e^i \in \mathbb{N}$:

Label-Setting Algorithmus für das EP 'multikriterielle kürzeste Wege'

Require: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit positiven Kantenlabels w_e^1, w_e^2 für $e \in E$, Startknoten s, Zielknoten t, Schranken W^1, W^2
Ensure: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)
Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ für $i = 1, 2$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$
while L_{open} nicht leer **do**
  wähle $l \in L_{open}$,
  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$
  for $v \in Adj(l.knoten)$ **do**
    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für $i = 1, 2$, $l'.\pi = l$
    Dominanzcheck$(l', L_{open}, L_{closed})$
  end for
end while
return Labels $l \in L_{closed}$ mit $l.knoten = t$

Teste: Ist $l'.d_i \leq W^i$?

Wie viele Labels werden (maximal) erstellt?

Angenommen $w_e^i \in \mathbb{N}$:

Maximal $W^1 \cdot W^2$ unterschiedliche Label pro Knoten möglich.

Label-Setting Algorithmus für das EP 'multikriterielle kürzeste Wege'

Require: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit positiven Kantenlabels w_e^1, w_e^2 für $e \in E$, Startknoten s, Zielknoten t, Schranken W^1, W^2
Ensure: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)
Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ für $i = 1, 2$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$
while L_{open} nicht leer **do**
  wähle $l \in L_{open}$,
  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$
  for $v \in Adj(l.knoten)$ **do**
    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für $i = 1, 2$, $l'.\pi = l$
    Dominanzcheck$(l', L_{open}, L_{closed})$
  end for
end while
return Labels $l \in L_{closed}$ mit $l.knoten = t$

Teste: Ist $l'.d_i \leq W^i$?

Wie viele Labels werden (maximal) erstellt?

Angenommen $w_e^i \in \mathbb{N}$:

Maximal $W^1 \cdot W^2$ unterschiedliche Label pro Knoten möglich.

$\Rightarrow$ Problem ist in $O(W^1 \cdot W^2 \cdot |V|)$.

Label-Setting Algorithmus für das EP 'multikriterielle kürzeste Wege'

Require: (gerichteter oder ungerichteter) Graph $G = (V, E)$ mit positiven Kantenlabels w_e^1, w_e^2 für $e \in E$, Startknoten s, Zielknoten t, Schranken W^1, W^2
Ensure: Menge der nicht-dominierten Labels in t (repräsentieren nicht-dominierte s-t-Wege)
Initialisiere: Erstelle Label l_0 mit $l_0.knoten = s$, $l_0.d_i = 0$ für $i = 1, 2$, $l_0.\pi = ()$, $L_{open} = \{l_0\}$, $L_{closed} = \{\}$
while L_{open} nicht leer **do**
  wähle $l \in L_{open}$,
  setze $L_{open} = L_{open} \setminus \{l\}$, $L_{closed} = L_{closed} \cup \{l\}$
  for $v \in Adj(l.knoten)$ **do**
    Create l' mit $l'.knoten = v$, $l'.d_i = l.d_i + w^i_{l.knoten,v}$ für $i = 1, 2$, $l'.\pi = l$
    Dominanzcheck$(l', L_{open}, L_{closed})$
  end for
end while
return Labels $l \in L_{closed}$ mit $l.knoten = t$

Teste: Ist $l'.d_i \leq W^i$?

Wie viele Labels werden (maximal) erstellt?

Angenommen $w_e^i \in \mathbb{N}$:

Maximal $W^1 \cdot W^2$ unterschiedliche Label pro Knoten möglich.

$\Rightarrow$ Problem ist in $O(W^1 \cdot W^2 \cdot |V|)$.

Ist das polynomiell in der Eingabelänge $|I|$?

Pseudo-polynomielle Zeit

Ein Algorithmus für ein (Entscheidungs-/Optimierungs-)Problem benötigt **pseudo-polynomielle** Zeit, wenn seine Laufzeit polynomiell in $|I|$ und number(I) ist, wobei number(I) die größte in I vorkommende Zahl ist.

Pseudo-polynomielle Zeit

Ein Algorithmus für ein (Entscheidungs-/Optimierungs-)Problem benötigt **pseudo-polynomielle** Zeit, wenn seine Laufzeit polynomiell in $|I|$ und number(I) ist, wobei number(I) die größte in I vorkommende Zahl ist.

Für feste m (z.B. $m = 2$) und ganzzahlige Kantengewichte ist das Entscheidungsproblem 'multikriterielle kürzeste Wege' pseudo-polynomiell lösbar, nämlich in $O(\text{number}(I)^m|V|)$ lösbar.

Pseudo-polynomielle Zeit

Ein Algorithmus für ein (Entscheidungs-/Optimierungs-)Problem benötigt **pseudo-polynomielle** Zeit, wenn seine Laufzeit polynomiell in $|I|$ und number(I) ist, wobei number(I) die größte in I vorkommende Zahl ist.

Für feste m (z.B. $m = 2$) und ganzzahlige Kantengewichte ist das Entscheidungsproblem 'multikriterielle kürzeste Wege' pseudo-polynomiell lösbar, nämlich in $O(\text{number}(I)^m|V|)$ lösbar.
Mehr dazu auf dem Übungszettel!

Stark NP-schwer

Für welche der bisher in der Vorlesung (und Übung) betrachteten NP-schweren Problem *könnte* es pseudo-polynomielle Algorithmen geben?

Stark NP-schwer

Für welche der bisher in der Vorlesung (und Übung) betrachteten NP-schweren Problem *könnte* es pseudo-polynomielle Algorithmen geben?

$\rightarrow$ Konzept ergibt nur Sinn für Probleme, wo number(I) nicht sowieso polynomiell in $|I|$ ist!

Stark NP-schwer

Für welche der bisher in der Vorlesung (und Übung) betrachteten NP-schweren Problem *könnte* es pseudo-polynomielle Algorithmen geben?

$\rightarrow$ Konzept ergibt nur Sinn für Probleme, wo number(I) nicht sowieso polynomiell in $|I|$ ist!

Ein Problem, dass schon bei Beschränkung auf Instanzen mit number(I) polynomiell in $|I|$ NP-schwer ist, nennen wir **stark NP-schwer**.

Stark NP-schwer

Für welche der bisher in der Vorlesung (und Übung) betrachteten NP-schweren Problem *könnte* es pseudo-polynomielle Algorithmen geben?

$\rightarrow$ Konzept ergibt nur Sinn für Probleme, wo number(I) nicht sowieso polynomiell in $|I|$ ist!

Ein Problem, dass schon bei Beschränkung auf Instanzen mit number(I) polynomiell in $|I|$ NP-schwer ist, nennen wir **stark NP-schwer**.

Beispiele: Knotenüberdeckung, Färbungsproblem, Clique, Hamiltonkreis, etc

Stark NP-schwer

Für welche der bisher in der Vorlesung (und Übung) betrachteten NP-schweren Problem *könnte* es pseudo-polynomielle Algorithmen geben?

$\rightarrow$ Konzept ergibt nur Sinn für Probleme, wo number(I) nicht sowieso polynomiell in $|I|$ ist!

Ein Problem, dass schon bei Beschränkung auf Instanzen mit number(I) polynomiell in $|I|$ NP-schwer ist, nennen wir **stark NP-schwer**.

Beispiele: Knotenüberdeckung, Färbungsproblem, Clique, Hamiltonkreis, etc

Ist das Handlungsreisendenproblem stark NP-schwer oder pseudo-polynomiell lösbar?

VL 6: Beispiel Polynomialzeitreduktion

Satz: Entscheidungsproblem Hamiltonscher Kreis (HC) $\leq_p$ Entscheidungsproblem Handlungsreisendenproblem (TSP)

Beweis:

Sei $G = (V, E)$ eine Instanz von HC.

Wir konstruieren eine Instanz $f(G) = (\tilde{G}, k)$ von TSP wie folgt:

Knoten $\tilde{V} := V$, Kanten $\tilde{E} := \{\{u, v\} \in V \times V, u \neq v\}$ $\tilde{G} := (\tilde{V}, \tilde{E})$

Kantenlabel: $l_e = 1$ falls $e \in E$, $l_e = |V| + 2$ falls $e \in \tilde{E} \setminus E$

$k := |V| + 1$

G ist eine JA-Instanz von HC genau dann, wenn $(\tilde{G}, k)$ eine JA-Instanz von TSP ist.

Stark NP-schwer

Für welche der bisher in der Vorlesung (und Übung) betrachteten NP-schweren Problem *könnte* es pseudo-polynomielle Algorithmen geben?

$\rightarrow$ Konzept ergibt nur Sinn für Probleme, wo nicht number(I) sowieso in $O(|I|)$ ist!

Ein Problem, dass schon bei Beschränkung auf Instanzen mit number(I) polynomiell in $|I|$ NP-schwer ist, nennen wir **stark NP-schwer**.

Beispiele: Knotenüberdeckung, Färbungsproblem, Clique, Hamiltonkreis, etc

Satz: Das Handlungsreisendenproblem ist stark NP-schwer.
Beweis: Reduktion von Hamiltonscher Kreis (VL6).

Stark NP-schwer

Für welche der bisher in der Vorlesung (und Übung) betrachteten NP-schweren Problem *könnte* es pseudo-polynomielle Algorithmen geben?

$\rightarrow$ Konzept ergibt nur Sinn für Probleme, wo nicht number(I) sowieso in $O(|I|)$ ist!

Ein Problem, dass schon bei Beschränkung auf Instanzen mit number(I) polynomiell in $|I|$ NP-schwer ist, nennen wir **stark NP-schwer**.

Beispiele: Knotenüberdeckung, Färbungsproblem, Clique, Hamiltonkreis, etc

Satz: Das Handlungsreisendenproblem ist stark NP-schwer.
Beweis: Reduktion von Hamiltonscher Kreis (VL6).

Korollar: Es gibt keinen pseudo-polynomiellen Lösungsalgorithmus für das Handlungsreisendenproblem.

Brute Force für EP Knotenüberdeckung

BruteForce(Graph $G = (V, E)$,Zahl k)

Brute Force für EP Knotenüberdeckung

BruteForce(Graph $G = (V, E)$,Zahl k)

foreach $V' \in \binom{V}{k}$ **do**

  // teste, ob V' Knotenüb.

  vc $= true$

  if vc $= true$ **then**

    return ("JA", V')

return ("NEIN", $\emptyset$)

Brute Force für EP Knotenüberdeckung

```
BruteForce(Graph G = (V, E),Zahl k)
  foreach V' ∈ (V choose k) do
    // teste, ob V' Knotenüb.
    vc = true
    foreach {u, v} ∈ E do
      if {u, v} ∩ V' = ∅ then
        vc = false
    if vc = true then
      return ("JA", V')
  return ("NEIN", ∅)
```

Brute Force für EP Knotenüberdeckung

```
BruteForce(Graph G = (V, E),Zahl k)
  foreach V' ∈ (V choose k) do
    // teste, ob V' Knotenüb.
    vc = true
    foreach {u, v} ∈ E do
      if {u, v} ∩ V' = ∅ then
        vc = false
    if vc = true then
      return ("JA", V')
  return ("NEIN", ∅)
```

Laufzeit.

Brute Force für EP Knotenüberdeckung

```
BruteForce(Graph G = (V, E),Zahl k)
  foreach V' ∈ (V choose k) do
    // teste, ob V' Knotenüb.
    vc = true
    foreach {u, v} ∈ E do
      if {u, v} ∩ V' = ∅ then
        vc = false
    if vc = true then
      return ("JA", V')
  return ("NEIN", ∅)
```

Laufzeit.

Brute Force für EP Knotenüberdeckung

```
BruteForce(Graph G = (V, E),Zahl k)
  foreach V' ∈ (V choose k) do
    // teste, ob V' Knotenüb.
    vc = true
    foreach {u, v} ∈ E do
      if {u, v} ∩ V' = ∅ then
        vc = false
    if vc = true then
      return ("JA", V')
  return ("NEIN", ∅)
```

$\binom{|V|}{k} = O(|V|^k)$

Laufzeit.

Brute Force für EP Knotenüberdeckung

```
BruteForce(Graph G = (V, E),Zahl k)
  foreach V' ∈ (V choose k) do
    // teste, ob V' Knotenüb.
    vc = true
    foreach {u, v} ∈ E do
      if {u, v} ∩ V' = ∅ then
        vc = false
    if vc = true then
      return ("JA", V')
  return ("NEIN", ∅)
```

$\binom{|V|}{k} = O(|V|^k)$

Laufzeit.

Brute Force für EP Knotenüberdeckung

```
BruteForce(Graph G = (V, E),Zahl k)
  foreach V' ∈ (V choose k) do
    // teste, ob V' Knotenüb.
    vc = true
    foreach {u, v} ∈ E do
      if {u, v} ∩ V' = ∅ then
        vc = false
    if vc = true then
      return ("JA", V')
  return ("NEIN", ∅)
```

$\binom{|V|}{k} = O(|V|^k)$

$O(|E|)$

Laufzeit.

Brute Force für EP Knotenüberdeckung

```
BruteForce(Graph G = (V, E),Zahl k)
  foreach V' ∈ (V choose k) do
    // teste, ob V' Knotenüb.
    vc = true
    foreach {u, v} ∈ E do
      if {u, v} ∩ V' = ∅ then
        vc = false
    if vc = true then
      return ("JA", V')
  return ("NEIN", ∅)
```

$\binom{|V|}{k} = O(|V|^k)$

$O(|E|)$

Laufzeit. $O(|V|^k|E|)$

Brute Force für EP Knotenüberdeckung

```
BruteForce(Graph G = (V, E),Zahl k)
  foreach V' ∈ (V choose k) do
    // teste, ob V' Knotenüb.
    vc = true
    foreach {u, v} ∈ E do
      if {u, v} ∩ V' = ∅ then
        vc = false
    if vc = true then
      return ("JA", V')
  return ("NEIN", ∅)
```

$\binom{|V|}{k} = O(|V|^k)$

$O(|E|)$

Laufzeit. $O(|V|^k|E|)$ Dies ist *nicht* polynomiell in der Größe der Eingabe ($= |V| + |E| + \log(k)$), da k keine Konstante, sondern Teil der Eingabe.

Festparameter-Berechenbarkeit

Finde einen Algorithmus, der das EP Knotenüberdeckung in Zeit

$$O(f(k) + |I|^c)$$

Festparameter-Berechenbarkeit

Finde einen Algorithmus, der das EP Knotenüberdeckung in Zeit

$$O(f(k) + |I|^c)$$

löst, wobei $f : \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

Festparameter-Berechenbarkeit

Finde einen Algorithmus, der das EP Knotenüberdeckung in Zeit

$$O(f(k) + |I|^c)$$

löst, wobei $f : \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k,
– polynomiell von der Größe $|I|$ der Instanz I.

Festparameter-Berechenbarkeit

Finde einen Algorithmus, der das EP Knotenüberdeckung in Zeit

$$O(f(k) + |I|^c)$$

löst, wobei $f : \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k,
– polynomiell von der Größe $|I|$ der Instanz I.

Schwierigkeit der Instanz

Festparameter-Berechenbarkeit

Finde einen Algorithmus, der das EP Knotenüberdeckung in Zeit

$$O(f(k) + |I|^c)$$

löst, wobei $f : \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k,
– polynomiell von der Größe $|I|$ der Instanz I.

Schwierigkeit der Instanz

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

Festparameter-Berechenbarkeit

Finde einen Algorithmus, der das EP Knotenüberdeckung in Zeit

$$O(f(k) + |I|^c)$$

löst, wobei $f : \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k,
– polynomiell von der Größe $|I|$ der Instanz I.

Schwierigkeit der Instanz

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

$\mathcal{FPT}$ = Klasse der festparameterberechenbaren Probleme.

Festparameter-Berechenbarkeit

Bemerkung.
Die Klasse $\mathcal{FPT}$ ändert sich nicht, wenn statt $+$ hier $\cdot$ steht.

Finde einen Algorithmus, der das EP Knotenüberdeckung in Zeit

$$O(f(k) + |I|^c)$$

löst, wobei $f : \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k,
– polynomiell von der Größe $|I|$ der Instanz I.

Schwierigkeit der Instanz

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

$\mathcal{FPT}$ = Klasse der festparameterberechenbaren Probleme.

Festparameter-Berechenbarkeit

Bemerkung.
Die Klasse $\mathcal{FPT}$ ändert sich nicht, wenn statt $+$ hier $\cdot$ steht.

Finde einen Algorithmus, der das EP Knotenüberdeckung in Zeit

$$O(f(k) + |I|^c) =: O^\star(f(k))$$

löst, wobei $f : \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k,
– polynomiell von der Größe $|I|$ der Instanz I.

Schwierigkeit der Instanz

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

$\mathcal{FPT}$ = Klasse der festparameterberechenbaren Probleme.

Festparameter-Berechenbarkeit

Bemerkung.
Die Klasse $\mathcal{FPT}$ ändert sich nicht, wenn statt $+$ hier $\cdot$ steht.

Finde einen Algorithmus, der das EP Knotenüberdeckung in Zeit

$$O(f(k) + |I|^c) =: O^\star(f(k))$$

ignoriert polynomielle Faktoren!

löst, wobei $f : \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k,
– polynomiell von der Größe $|I|$ der Instanz I.

Schwierigkeit der Instanz

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

$\mathcal{FPT}$ = Klasse der festparameterberechenbaren Probleme.

Festparameter-Berechenbarkeit

Bemerkung.
Die Klasse $\mathcal{FPT}$ ändert sich nicht, wenn statt $+$ hier $\cdot$ steht.

Finde einen Algorithmus, der das EP Knotenüberdeckung in Zeit

$$O(f(k) + |I|^c) =: O^\star(f(k))$$

ignoriert polynomielle Faktoren!

löst, wobei $f : \mathbb{N} \to \mathbb{N}$ berechenbare Funktion (unabh. von I),
I gegebene Instanz, c Konstante (unabh. von I)

D.h. die Laufzeit soll abhängen
– beliebig vom Parameter k,
– polynomiell von der Größe $|I|$ der Instanz I.

Schwierigkeit der Instanz

Ein Problem, das in dieser Zeit gelöst werden kann, heißt *festparameterberechenbar* (*fixed-parameter tractable*) bzgl. k.

$\mathcal{FPT}$ = Klasse der festparameterberechenbaren Probleme.

BruteForceVC hat *nicht* die gewünschte Laufzeit.

Ein paar einfache Beobachtungen...

Sei $G = (V, E)$ Graph, V' Knotenüberdeckung für G, $v \notin V'$.
Welche Knoten liegen dann sicher in V'?

Ein paar einfache Beobachtungen...

Sei $G = (V, E)$ Graph, V' Knotenüberdeckung für G, $v \notin V'$.
Welche Knoten liegen dann sicher in V'?

Beob. 1: V' Knotenüberdeckung für G, $v \in V$ ein Knoten.
Dann gilt: $v \in V'$ oder $N_G(v) \subseteq V'$.

'Nachbarn' von v in G
= inzidente Knoten

Ein paar einfache Beobachtungen...

Sei $G = (V, E)$ Graph, V' Knotenüberdeckung für G, $v \notin V'$.
Welche Knoten liegen dann sicher in V'?

Beob. 1: V' Knotenüberdeckung für G, $v \in V$ ein Knoten.
Dann gilt: $v \in V'$ oder $N_G(v) \subseteq V'$.

'Nachbarn' von v in G
= inzidente Knoten

Betrachte Entscheidungsproblem k-Knotenüberdeckung: 'Gibt es Knotenüberdeckung mit k Knoten?' .
Was gilt für Knoten mit Grad $> k$?

Ein paar einfache Beobachtungen...

Sei $G = (V, E)$ Graph, V' Knotenüberdeckung für G, $v \notin V'$.
Welche Knoten liegen dann sicher in V'?

Beob. 1: V' Knotenüberdeckung für G, $v \in V$ ein Knoten.
Dann gilt: $v \in V'$ oder $N_G(v) \subseteq V'$.

'Nachbarn' von v in G = inzidente Knoten

Betrachte Entscheidungsproblem k-Knotenüberdeckung: 'Gibt es Knotenüberdeckung mit k Knoten?' .
Was gilt für Knoten mit Grad $> k$?

Beob. 2: Jeder Knoten mit Grad $> k$ ist in jeder k-Knotenüberdeckung von G enthalten.

Ein paar einfache Beobachtungen...

Sei $G = (V, E)$ Graph, V' Knotenüberdeckung für G, $v \notin V'$.
Welche Knoten liegen dann sicher in V'?

Beob. 1: V' Knotenüberdeckung für G, $v \in V$ ein Knoten.
Dann gilt: $v \in V'$ oder $N_G(v) \subseteq V'$.

'Nachbarn' von v in G = inzidente Knoten

Betrachte Entscheidungsproblem k-Knotenüberdeckung: 'Gibt es Knotenüberdeckung mit k Knoten?' .
Was gilt für Knoten mit Grad $> k$?

Beob. 2: Jeder Knoten mit Grad $> k$ ist in jeder k-Knotenüberdeckung von G enthalten.

Was gilt, falls $|E| > k^2$ und alle Knoten Grad $\leq k$ haben?

Ein paar einfache Beobachtungen...

Sei $G = (V, E)$ Graph, V' Knotenüberdeckung für G, $v \notin V'$.
Welche Knoten liegen dann sicher in V'?

Beob. 1: V' Knotenüberdeckung für G, $v \in V$ ein Knoten.
Dann gilt: $v \in V'$ oder $N_G(v) \subseteq V'$.

'Nachbarn' von v in G = inzidente Knoten

Betrachte Entscheidungsproblem k-Knotenüberdeckung: 'Gibt es Knotenüberdeckung mit k Knoten?' .
Was gilt für Knoten mit Grad $> k$?

Beob. 2: Jeder Knoten mit Grad $> k$ ist in jeder k-Knotenüberdeckung von G enthalten.

Was gilt, falls $|E| > k^2$ und alle Knoten Grad $\leq k$ haben?

Beob. 3: Falls $|E| > k^2$ und $\Delta(G) := \max_{v \in V} \deg v \leq k$, so hat G keine k-Knotenüberdeckung.

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

II) Lösung der Restinstanz mit Brute Force Ansatz

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und
Knotenüberdeckung $\tilde{V}'$ liefert **then**
return ('JA', $V_{>k} \cup \tilde{V}'$)
else
return ('NEIN',$\emptyset$)

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

Benutzen Sie Beobachtungen 2 & 3, um eine Instanz (G, k) in eine Instanz $(\tilde{G}, \tilde{k})$ zu verwandeln, so dass $|\tilde{G}| := |\tilde{V}| + |\tilde{E}| \in O(k^2)$!

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und Knotenüberdeckung $\tilde{V}'$ liefert **then**
  return ('JA', $V_{>k} \cup \tilde{V}'$)
else
  return ('NEIN',$\emptyset$)

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und
Knotenüberdeckung $\tilde{V}'$ liefert **then**
return ('JA', $V_{>k} \cup \tilde{V}'$)
else
return ('NEIN',$\emptyset$)

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und
Knotenüberdeckung $\tilde{V}'$ liefert **then**
return ('JA', $V_{>k} \cup \tilde{V}'$)
else
return ('NEIN',$\emptyset$)

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und
Knotenüberdeckung $\tilde{V}'$ liefert **then**
return ('JA', $V_{>k} \cup \tilde{V}'$)
else
return ('NEIN',$\emptyset$)

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und
Knotenüberdeckung $\tilde{V}'$ liefert **then**
return ('JA', $V_{>k} \cup \tilde{V}'$)
else
return ('NEIN',$\emptyset$)

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und
Knotenüberdeckung $\tilde{V}'$ liefert **then**
return ('JA', $V_{>k} \cup \tilde{V}'$)
else
return ('NEIN',$\emptyset$)

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und
Knotenüberdeckung $\tilde{V}'$ liefert **then**
return ('JA', $V_{>k} \cup \tilde{V}'$)
else
return ('NEIN',$\emptyset$)

Laufzeit.

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

Laufzeit.

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und
Knotenüberdeckung $\tilde{V}'$ liefert **then**
return ('JA', $V_{>k} \cup \tilde{V}'$)
else
return ('NEIN',$\emptyset$)

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

} $O(|V| + |E|)$ Zeit

Laufzeit.

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und Knotenüberdeckung $\tilde{V}'$ liefert **then**
  return ('JA', $V_{>k} \cup \tilde{V}'$)
else
  return ('NEIN',$\emptyset$)

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

} $O(|V| + |E|)$ Zeit

Laufzeit.

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und Knotenüberdeckung $\tilde{V}'$ liefert **then**
  return ('JA', $V_{>k} \cup \tilde{V}'$)
else
  return ('NEIN',$\emptyset$)

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

$O(|V| + |E|)$ Zeit

Laufzeit.

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und Knotenüberdeckung $\tilde{V}'$ liefert **then**
  return ('JA', $V_{>k} \cup \tilde{V}'$)
else
  return ('NEIN',$\emptyset$)

$O(|\tilde{E}| \cdot (|\tilde{V}|)^{\tilde{k}})$ Zeit

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

$O(|V| + |E|)$ Zeit

Laufzeit.

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und Knotenüberdeckung $\tilde{V}'$ liefert **then return** ('JA', $V_{>k} \cup \tilde{V}'$)

else

  return ('NEIN',$\emptyset$)

$O(|\tilde{E}| \cdot (|\tilde{V}|)^{\tilde{k}})$ Zeit

wobei $|\tilde{E}| \leq k^2$

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

$O(|V| + |E|)$ Zeit

Laufzeit.

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und Knotenüberdeckung $\tilde{V}'$ liefert **then return** ('JA', $V_{>k} \cup \tilde{V}'$)

else

  return ('NEIN',$\emptyset$)

$O(|\tilde{E}| \cdot (|\tilde{V}|)^{\tilde{k}})$ Zeit

wobei $|\tilde{E}| \leq k^2$

$\Rightarrow \; |\tilde{V}| \leq 2k^2$

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

Laufzeit.

$O(|V| + |E|)$ Zeit

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und Knotenüberdeckung $\tilde{V}'$ liefert **then return** ('JA', $V_{>k} \cup \tilde{V}'$)

else

  return ('NEIN',$\emptyset$)

$O(|\tilde{E}| \cdot (|\tilde{V}|)^{\tilde{k}})$ Zeit

wobei $|\tilde{E}| \leq k^2$

$\Rightarrow |\tilde{V}| \leq 2k^2$

$O(|V| + |E| + k^2 \cdot (2k^2)^k)$

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

Laufzeit.

$O(|V| + |E|)$ Zeit

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und Knotenüberdeckung $\tilde{V}'$ liefert **then return** ('JA', $V_{>k} \cup \tilde{V}'$)

else

  return ('NEIN',$\emptyset$)

$O(|\tilde{E}| \cdot (|\tilde{V}|)^{\tilde{k}})$ Zeit

wobei $|\tilde{E}| \leq k^2$

$\Rightarrow |\tilde{V}| \leq 2k^2$

$$O\left(|V| + |E| + k^2 \cdot (2k^2)^k\right)$$

$$= O\left(|V| + |E| + 2^k k^{2k+2}\right)$$

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

Laufzeit.

} $O(|V| + |E|)$ Zeit

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und Knotenüberdeckung $\tilde{V}'$ liefert **then return** ('JA', $V_{>k} \cup \tilde{V}'$)

else

  return ('NEIN',$\emptyset$)

} $O(|\tilde{E}| \cdot (|\tilde{V}|)^{\tilde{k}})$ Zeit
wobei $|\tilde{E}| \leq k^2$
$\Rightarrow |\tilde{V}| \leq 2k^2$

$$O(|V| + |E| + k^2 \cdot (2k^2)^k)$$
$$= O(\underbrace{|V| + |E|}_{|I|^1} + 2^k k^{2k+2})$$

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

Laufzeit.

$O(|V| + |E|)$ Zeit

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und Knotenüberdeckung $\tilde{V}'$ liefert **then return** ('JA', $V_{>k} \cup \tilde{V}'$)

else

  return ('NEIN',$\emptyset$)

$O(|\tilde{E}| \cdot (|\tilde{V}|)^{\tilde{k}})$ Zeit

wobei $|\tilde{E}| \leq k^2$

$\Rightarrow |\tilde{V}| \leq 2k^2$

$$O(|V| + |E| + k^2 \cdot (2k^2)^k)$$

$$= O(\underbrace{|V| + |E|}_{|I|^1} + \underbrace{2^k k^{2k+2}}_{f(k)})$$

Algorithmus von Buss

Algorithmus von Buss(Graph $G = (V, E)$, Zahl k)

I) Reduktion der Instanz auf ihren harten Kern

$V_{>k} = \{v \in V \mid \deg_G(v) > k\}$

if $|V_{>k}| > k$ **then return** ("NEIN", $\emptyset$)

$V_{\text{is}} =$ isolierte Knoten in $G[V \setminus V_{>k}]$

$\tilde{V} = V \setminus (V_{>k} \cup V_{\text{is}})$, $\tilde{G} := G[\tilde{V}]$

if $|E(\tilde{G})| > k^2$ **then return** ("NEIN", $\emptyset$)

Laufzeit.

} $O(|V| + |E|)$ Zeit

II) Lösung der Restinstanz mit Brute Force Ansatz

$\tilde{k} = k - |V_{>k}|$

if BruteForce($\tilde{G}, \tilde{k}$) Antwort 'JA' und Knotenüberdeckung $\tilde{V}'$ liefert **then return** ('JA', $V_{>k} \cup \tilde{V}'$)

else

  return ('NEIN',$\emptyset$)

} $O(|\tilde{E}| \cdot (|\tilde{V}|)^{\tilde{k}})$ Zeit
wobei $|\tilde{E}| \leq k^2$
$\Rightarrow |\tilde{V}| \leq 2k^2$

$$O(|V| + |E| + k^2 \cdot (2k^2)^k)$$
$$= O(\underbrace{|V| + |E|}_{|I|^1} + \underbrace{2^k k^{2k+2}}_{f(k)})$$

Also: k-Knotenüberd. $\in \mathcal{FPT}$!

Stichworte heute

Algorithmen: Christofides' Algorithmus, Algorithmus von Buss, Bikriterielles Labelsetting (revisited)
Approximationsalgorithmen, pseudopolynomielle Algorithmen, Festparameteralgorithmen

Komplexität: FPT, stark NP-schwer, pseudopolynomiell

Optimierungsprobleme: Knotenüberdeckung, TSP (unterschiedliche Versionen), bikriteriell kürzeste Wege

Stichworte heute

Algorithmen: Christofides' Algorithmus, Algorithmus von Buss, Bikriterielles Labelsetting (revisited)
Approximationsalgorithmen, pseudopolynomielle Algorithmen, Festparameteralgorithmen

Komplexität: FPT, stark NP-schwer, pseudopolynomiell

Optimierungsprobleme: Knotenüberdeckung, TSP (unterschiedliche Versionen), bikriteriell kürzeste Wege

Nächste Woche geht es los mit Teil 2: mathematische Optimierung!

Stichworte heute

Algorithmen: Christofides' Algorithmus, Algorithmus von Buss, Bikriterielles Labelsetting (revisited)
Approximationsalgorithmen, pseudopolynomielle Algorithmen, Festparameteralgorithmen

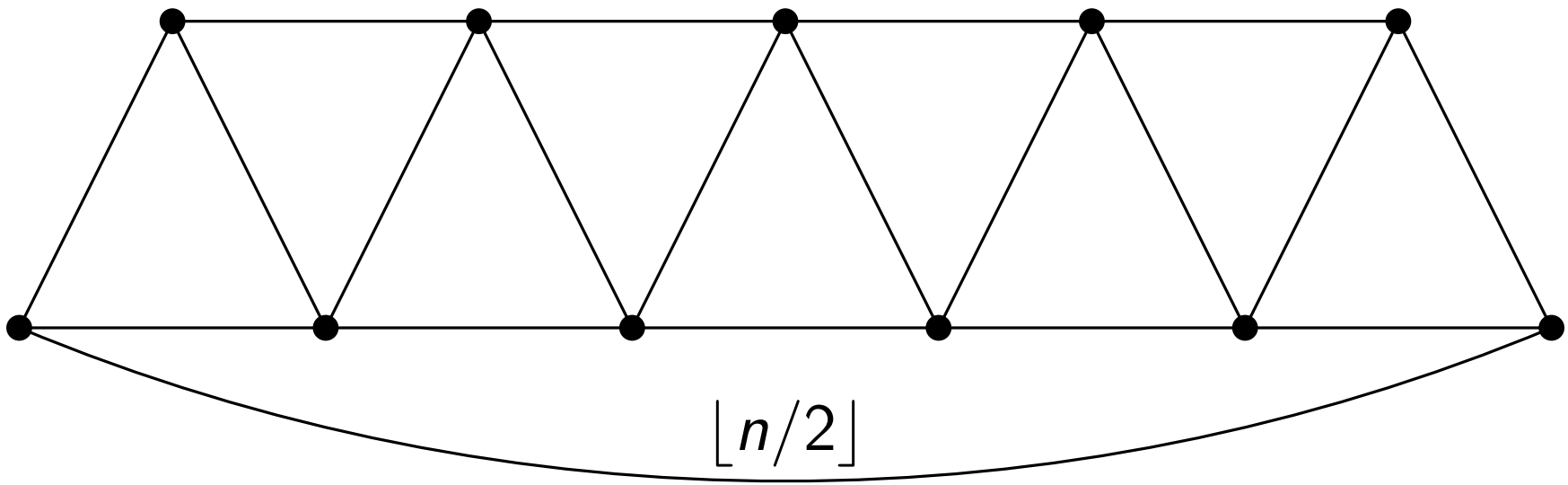
Komplexität: FPT, stark NP-schwer, pseudopolynomiell

Optimierungsprobleme: Knotenüberdeckung, TSP (unterschiedliche Versionen), bikriteriell kürzeste Wege

Wer sowas spannend findet,
der ist in den Vorlesungen von
Prof. Glaßer gut aufgehoben
(oder
https://tomvanderzanden.nl/LI
AC/)

Der Approximationsfaktor ist scharf

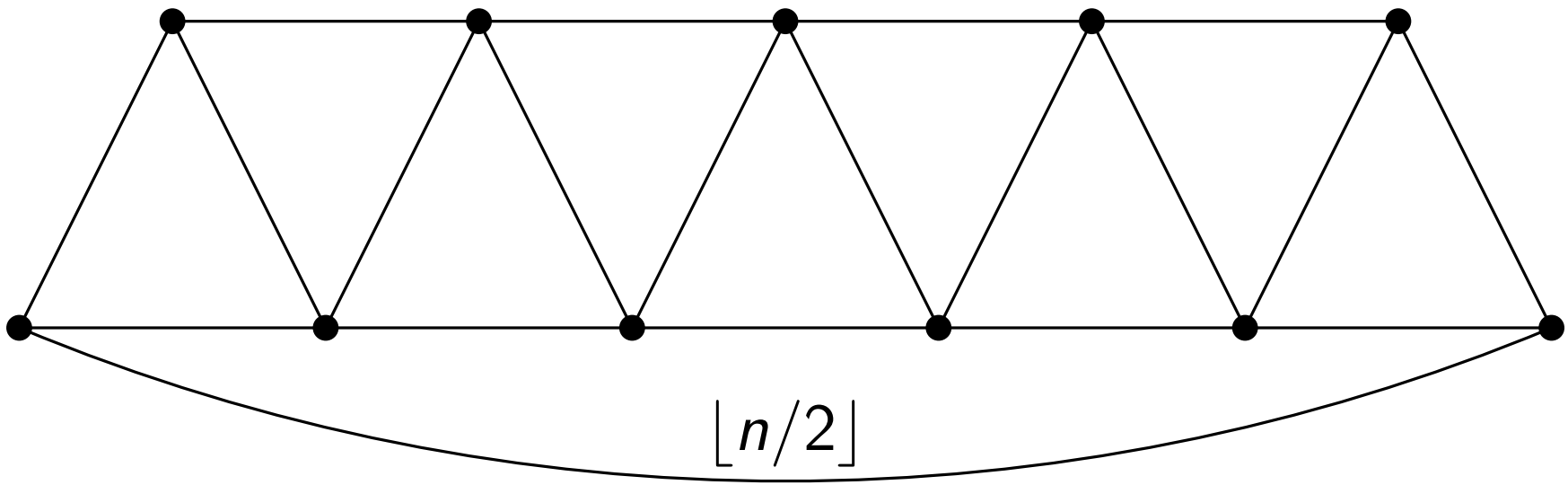
Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:



Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

– Unbeschriftete Kanten haben Kosten 1.

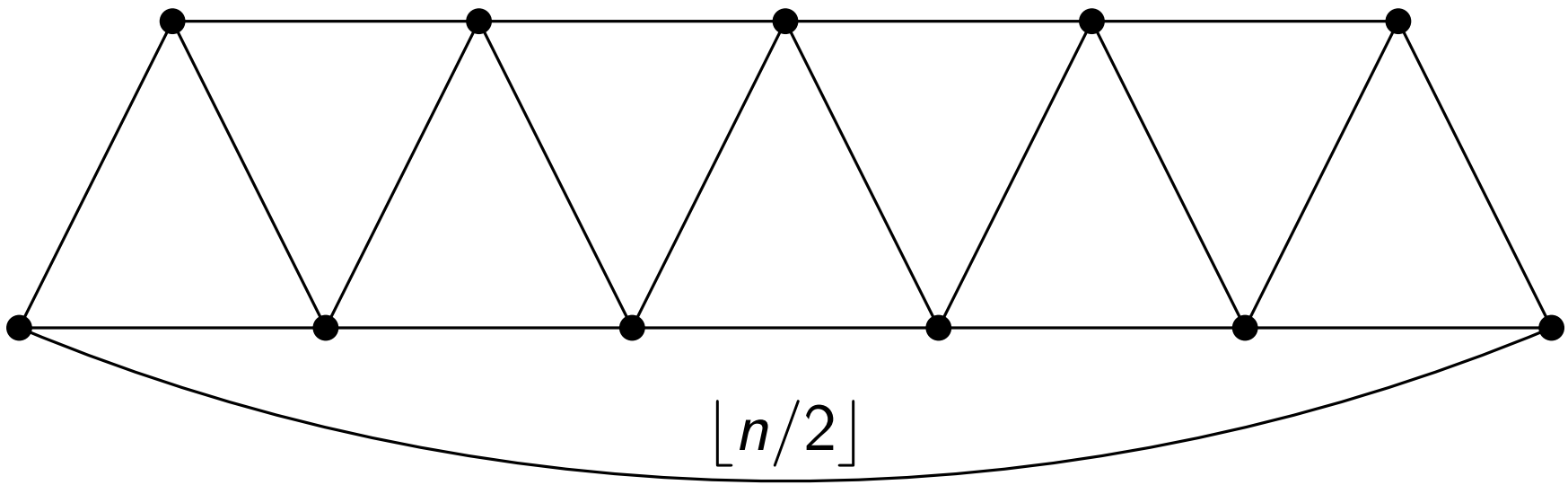


Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

– Unbeschriftete Kanten haben Kosten 1.

– Jede nicht gezeichnete Kante uv hat Kosten...

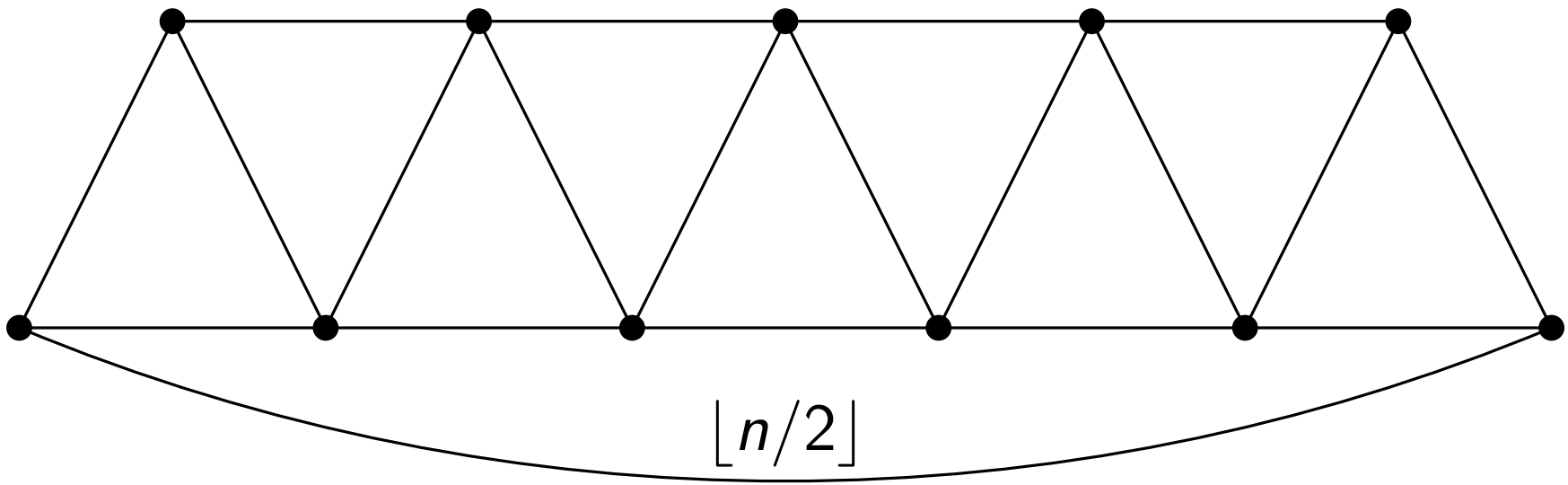


Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

– Unbeschriftete Kanten haben Kosten 1.

– Jede nicht gezeichnete Kante uv hat Kosten...
  Länge eines kürzesten u-v-Weges

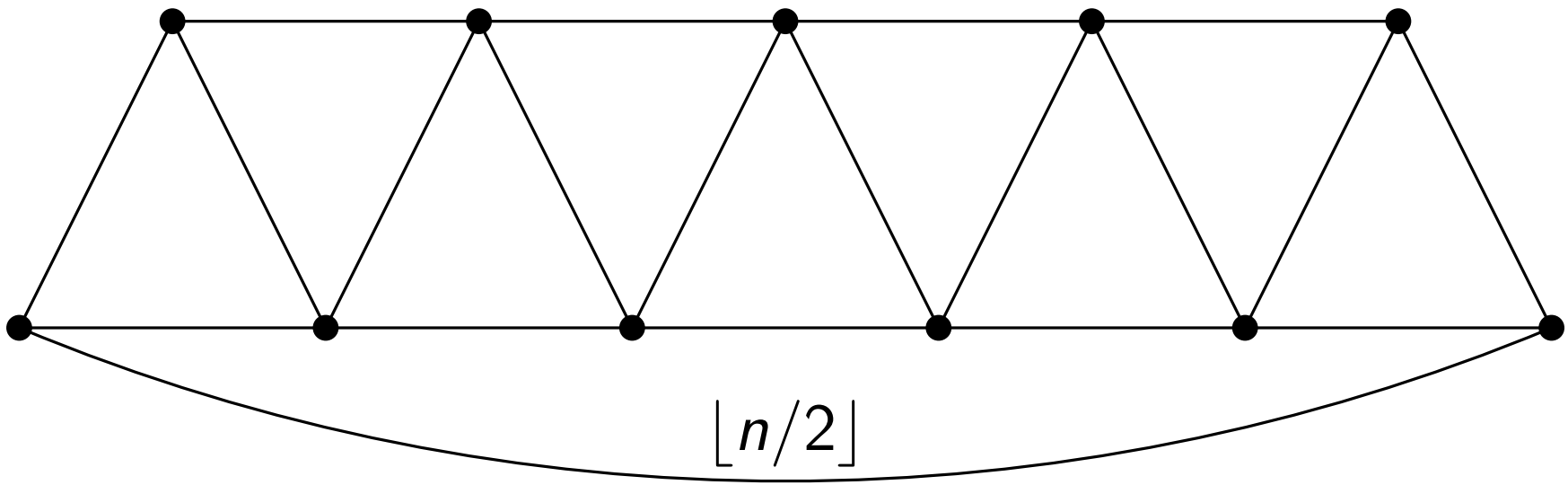


Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

– Unbeschriftete Kanten haben Kosten 1.

– Jede nicht gezeichnete Kante uv hat Kosten...
Länge eines kürzesten u-v-Weges ($\Rightarrow$ G metrisch!)

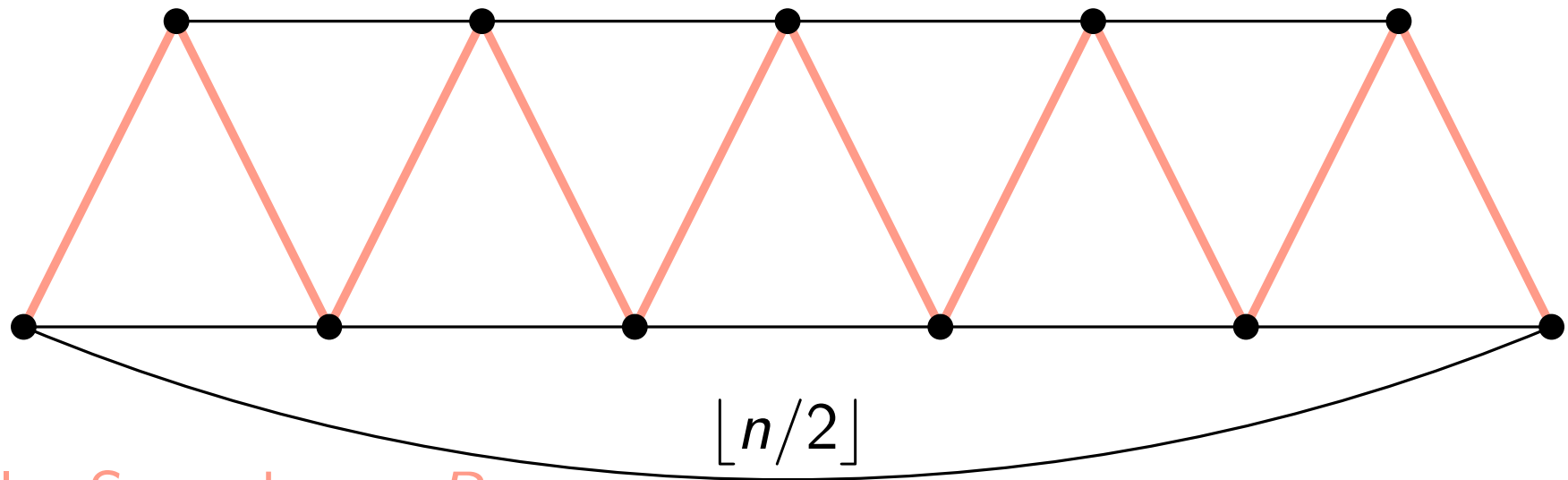


Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

– Unbeschriftete Kanten haben Kosten 1.

– Jede nicht gezeichnete Kante uv hat Kosten...
Länge eines kürzesten u-v-Weges ($\Rightarrow$ G metrisch!)



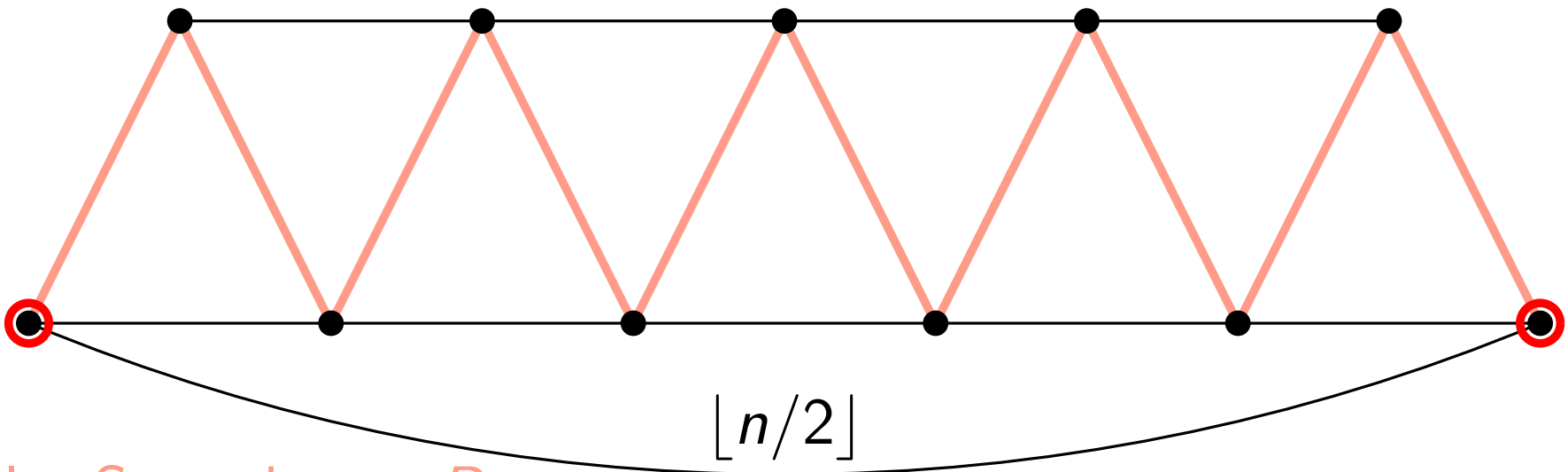
minimaler Spannbaum B

Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

– Unbeschriftete Kanten haben Kosten 1.

– Jede nicht gezeichnete Kante uv hat Kosten...
Länge eines kürzesten u-v-Weges ($\Rightarrow$ G metrisch!)



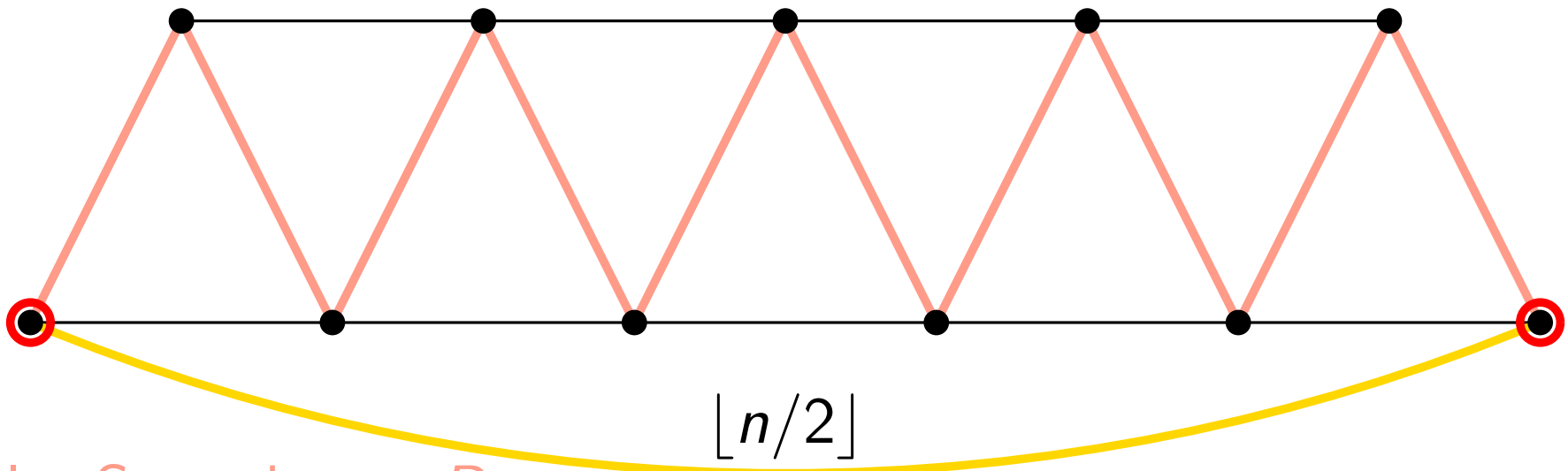
minimaler Spannbaum B

Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

– Unbeschriftete Kanten haben Kosten 1.

– Jede nicht gezeichnete Kante uv hat Kosten...
Länge eines kürzesten u-v-Weges ($\Rightarrow$ G metrisch!)

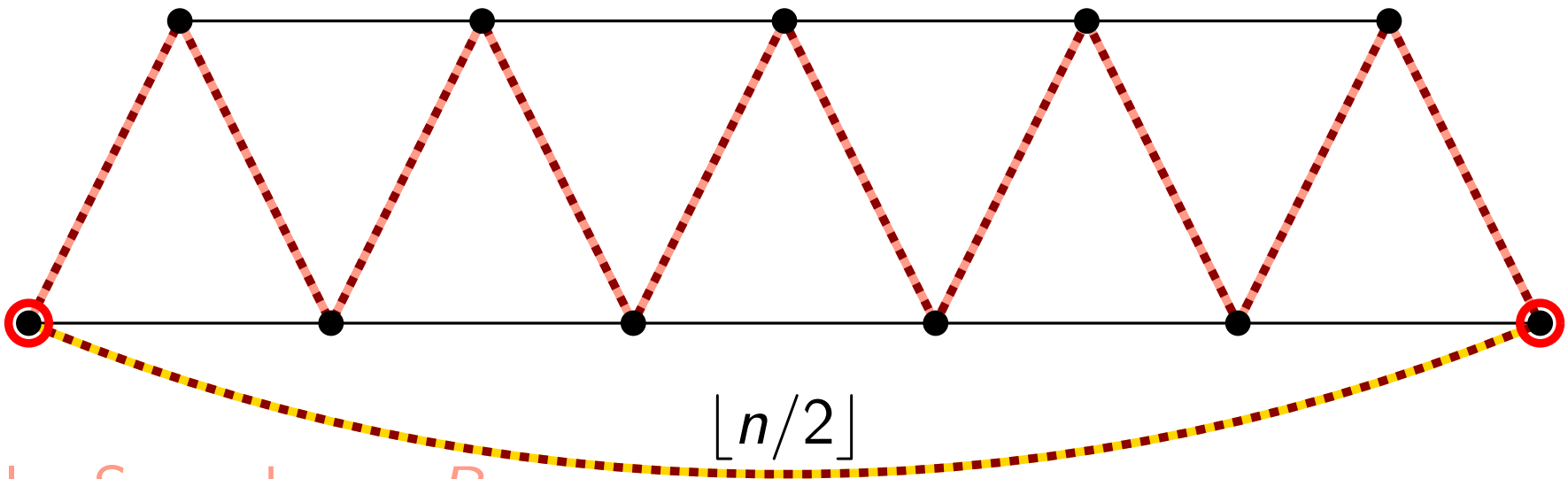


minimaler Spannbaum B
perfektes Matching M in $G[U]$

Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

- Unbeschriftete Kanten haben Kosten 1.
- Jede nicht gezeichnete Kante uv hat Kosten...
  Länge eines kürzesten u-v-Weges ($\Rightarrow$ G metrisch!)



minimaler Spannbaum B

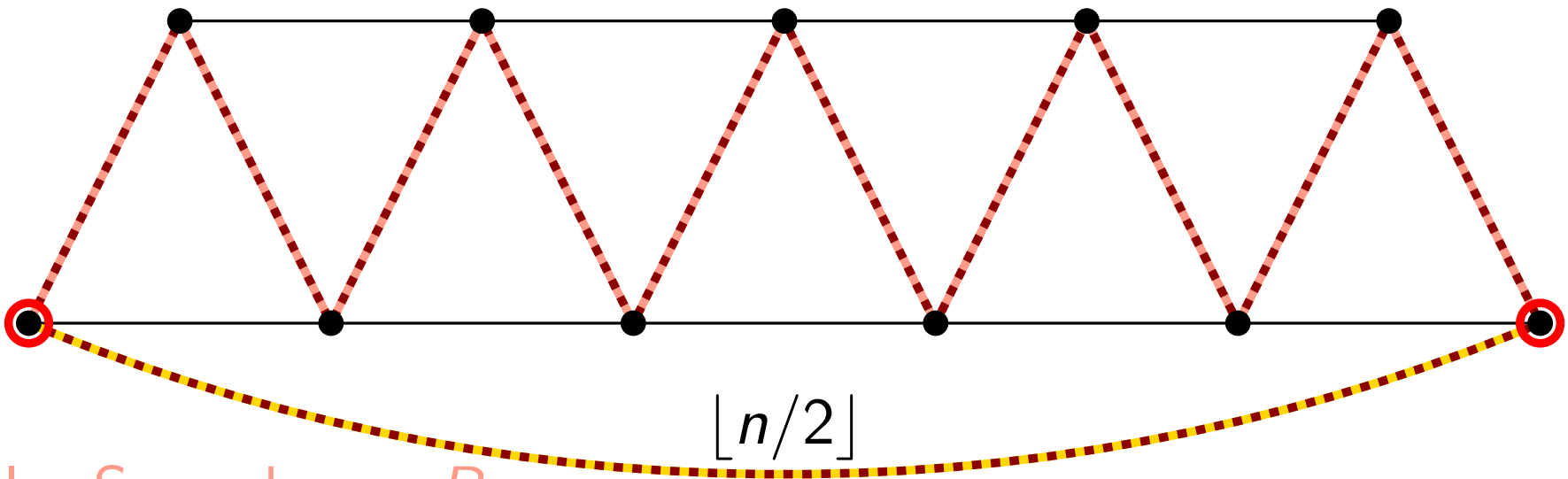
perfektes Matching M in $G[U]$

Christofides-Tour T mit Kosten ALG $=$

Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

- Unbeschriftete Kanten haben Kosten 1.
- Jede nicht gezeichnete Kante uv hat Kosten...
  Länge eines kürzesten u-v-Weges ($\Rightarrow$ G metrisch!)



minimaler Spannbaum B

perfektes Matching M in $G[U]$

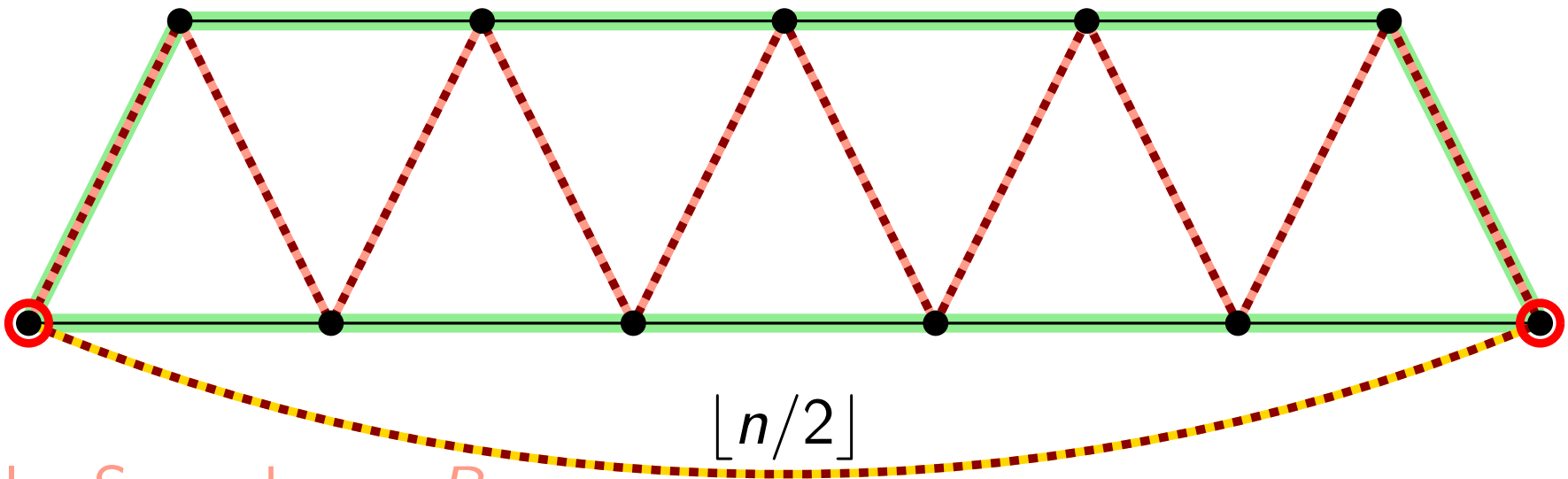
Christofides-Tour T mit Kosten ALG $= n + \lfloor n/2 \rfloor - 1$

Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

– Unbeschriftete Kanten haben Kosten 1.

– Jede nicht gezeichnete Kante uv hat Kosten...
Länge eines kürzesten u-v-Weges ($\Rightarrow$ G metrisch!)



minimaler Spannbaum B

perfektes Matching M in $G[U]$

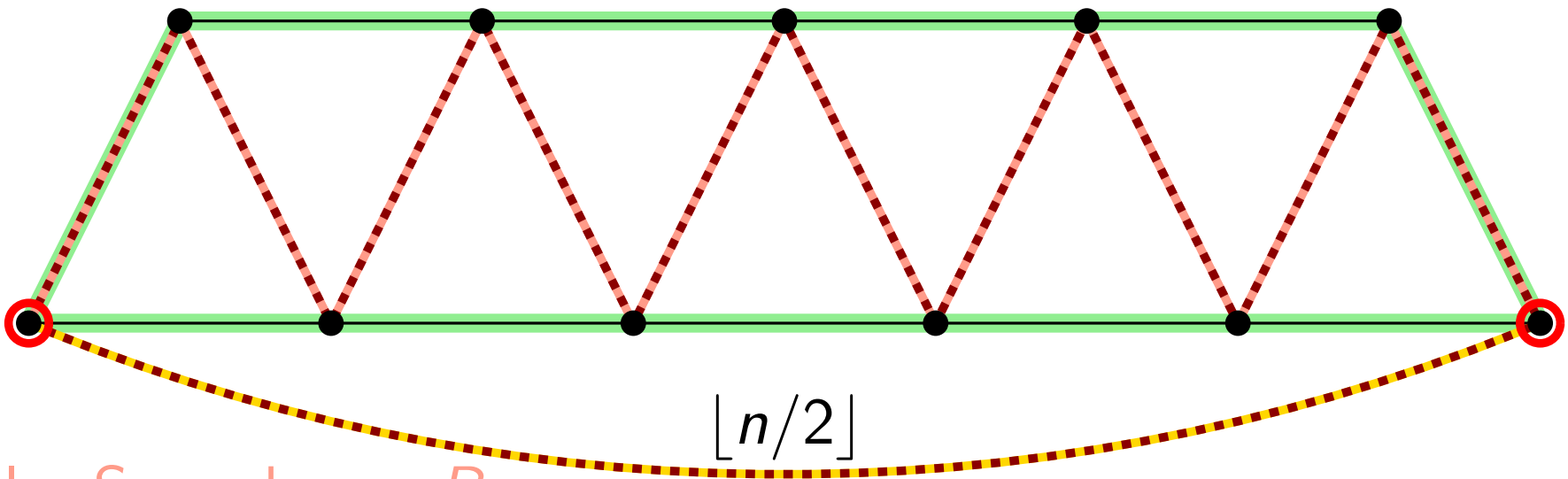
Christofides-Tour T mit Kosten ALG $= n + \lfloor n/2 \rfloor - 1$

OPT $=$

Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

- Unbeschriftete Kanten haben Kosten 1.
- Jede nicht gezeichnete Kante uv hat Kosten...
  Länge eines kürzesten u-v-Weges ($\Rightarrow$ G metrisch!)



minimaler Spannbaum B

perfektes Matching M in $G[U]$

Christofides-Tour T mit Kosten ALG $= n + \lfloor n/2 \rfloor - 1$

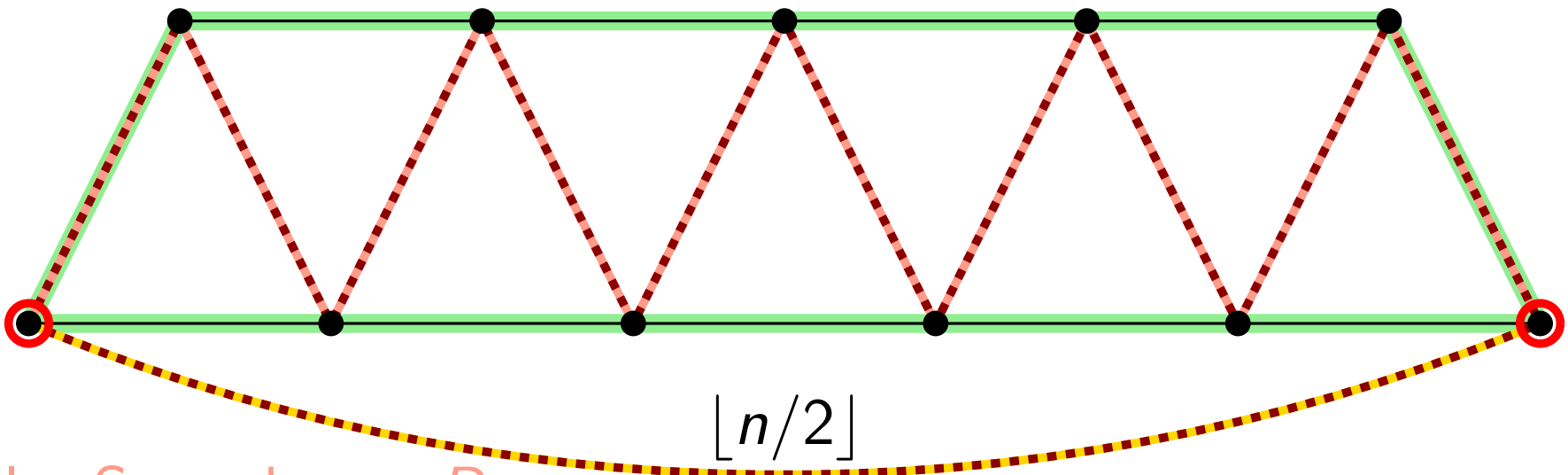
OPT $= n$

Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

– Unbeschriftete Kanten haben Kosten 1.

– Jede nicht gezeichnete Kante uv hat Kosten...
Länge eines kürzesten u-v-Weges ($\Rightarrow$ G metrisch!)



minimaler Spannbaum B

perfektes Matching M in $G[U]$

Christofides-Tour T mit Kosten ALG $= n + \lfloor n/2 \rfloor - 1$

OPT $= n$

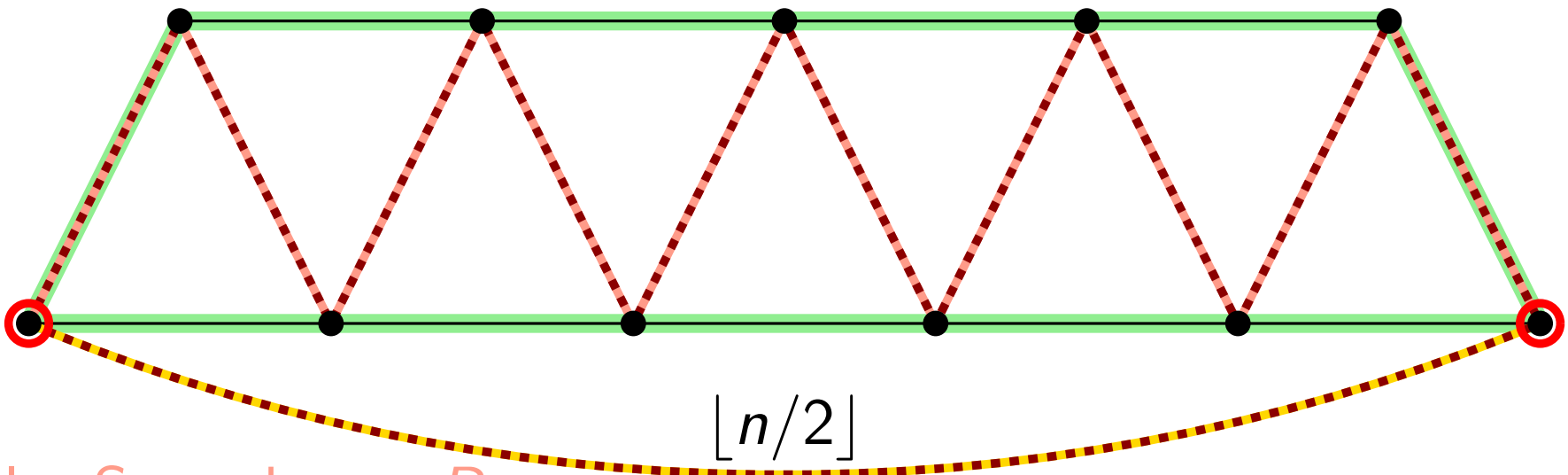
ALG / OPT $= (n + \lfloor n/2 \rfloor - 1)/n \rightarrow$

Der Approximationsfaktor ist scharf

Konstruiere metrische Instanz G, für die Christofides möglichst *schlecht* ist:

– Unbeschriftete Kanten haben Kosten 1.

– Jede nicht gezeichnete Kante uv hat Kosten...
Länge eines kürzesten u-v-Weges ($\Rightarrow$ G metrisch!)



minimaler Spannbaum B

perfektes Matching M in $G[U]$

Christofides-Tour T mit Kosten ALG $= n + \lfloor n/2 \rfloor - 1$

OPT $= n$

ALG / OPT $= (n + \lfloor n/2 \rfloor - 1)/n \to 3/2$