I. Probleme / Instanzen

Problem: allgemeine Frage

↳ * allg. Beschreibung der Parameter
* Beschr. d. Eigenschaften der Lsg.

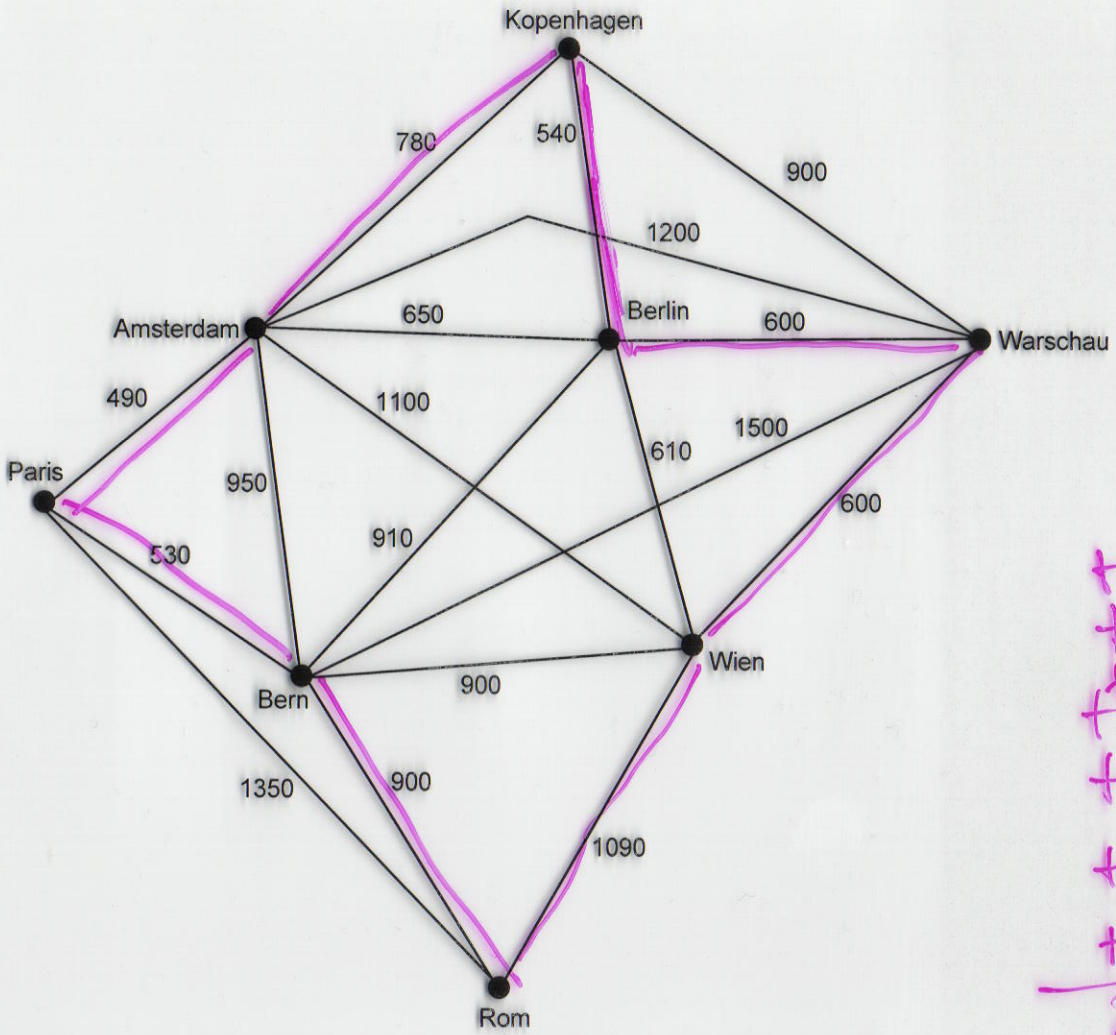
Instanz: definiert Werte für alle Problemparameter

Bsp. 1 TSP – Traveling Salesman Problem

Städte ⟷ Knoten
Straßen/Verbindungen ⟷ Kanten
Entfernung ⟷ Gewicht $c_{e_1}, \ldots, c_{e_m}$

↳ Problem:
- Gegeben: Graph $G = (V, E)$ mit Gewichtsfkt. c (Längenfkt.)
- Gesucht: Tour minimaler Länge, d.h. Rundreise die jeden Knoten genau einmal besucht

Instanz?



780
+ 540 1
+ 600
+ 600 2530
+ 1090 3620
+ 900 4520
+ 530
+ 490 5540
5530

AuD- Aufgabe

↳ wir hatten nicht bewiesen, dass es besser geht!

Also: warum geht es nicht besser?

∀ wir hatten: jede Lsg. braucht genau 8 Kanten (n Kanten)

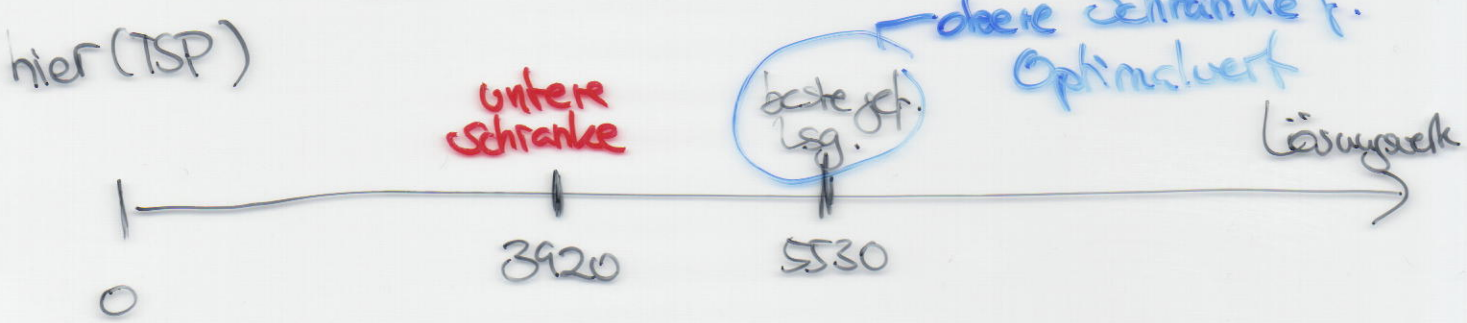
⇒ mindestens Kosten von

$$8 \cdot \min_{i=1,\dots,m} c_{e_i} = 8 \cdot 490 = 3920$$

untere Schranke

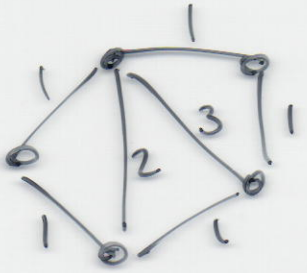
→ i. Allg.: $n \cdot \min_{i=1,\dots,m} c_{e_i}$ ist untere Schranke für jede Lsg., also auch für die Optimallsg.

Wir sehen: zum Beweisen von Optimalität einer Lsg. kann man z.B. Schranken verwenden



Finden wir eine (untere) Schranke und eine Lösung mit gleichem Wert, haben wir eine Optimallsg. gefunden.

Das kann beim TSP klappen (z.B.:



TSP-Lösung: 5
Schranke: 5

ABER: es klappt beim TSP leider nicht immer!

- in VL viele interessante Probleme, bei denen das klappt!

Laufzeit

↳ wie lange dauert es eine Lsg. zu finden?
↳ Aufwand hängt von Problem-/Inputgröße ab

Inputgröße beim TSP?

- $n \cdot \log n + m \cdot \log c$ $\quad C = \max_i c_i$

Ein naiver Alg. fürs TSP wäre:

- probiere alle zulässigen Lösungen
  (wieviele sind das?

$$(n \cdot (n-1) \cdot (n-2) \cdots \cdot 1) \cdot \frac{1}{2} = \frac{n!}{2}$$

n: # mögl. Startstädten

$$\approx \left(\frac{n}{e}\right)^n \cdot \sqrt{\frac{e^4}{8} \cdot n}$$

(Stirlingsche Formel)

$$= \mathcal{O}(n^n)$$

Einfacher: mindestens 2^n Möglichkeiten

⇒ eine Inputgröße im Exponenten

⇒ langsam...

inVL: Probleme, die Algorithmen mit Laufzeit polynomiell im Input erlauben!

Abteilung Algorithmik
Institut für Betriebssysteme und Rechnerverbund **SoSe 13**
TU Braunschweig

Prof. Dr. Sándor P. Fekete
Dr. Christiane Schmidt
Stephan Friedrichs

Netzwerkalgorithmen
Übung 0 vom 15.04.2013

Dieses Übungsblatt wird nicht abgegeben!
Besprechung der Aufgaben in den kleinen Übungen
am Donnerstag, 25.04.2013, und Freitag, 26.04.2013.

Ein bißchen Notation:
Ein ungerichteter Graph G ist *zusammenhängend*, wenn es für je 2 Knoten $v, w \in V(G)$ immer einen $v-w$–Pfad gibt. Ein gerichteter Graph ist zusammenhängend, wenn der ungerichtete Graph zusammenhängend ist.

$G \backslash e = (V(G), E(G) \backslash \{e\})$

Für $X, Y \subseteq V(G) : E(X;Y) = \{\{x,y\} \in E(G) : x \in X, y \in Y\}$
Für $X \subseteq V(G) : \delta(X) = E(X, V(G) \backslash X)$, ist der durch X *induzierte Schnitt.*

Ein *Schnitt* in einem ungerichteten Graphen ist eine Kantenmenge vom Typ $\delta(X)$ für $\emptyset \neq X \subsetneq V(G)$, die also X und $V \backslash X$ trennt.

Für gerichtete Graphen:
$E^+(X,Y) = \{\{x,y\} \in E(G) : x \in X \backslash Y, y \in Y \backslash X\}$
$\delta^+(X) = E^+(X, V(G) \backslash X)$ ("von X nach $V(G) \backslash X$")
$\delta^-(X) = \delta^+(V(G) \backslash X)$ ("von $V(G) \backslash X$ nach X")

Ein *gerichteter Schnitt* ist eine Kantenmenge vom Typ $\delta^+(X)$ für $\emptyset \neq X \subsetneq V(G)$ mit $\delta^-(X) = \emptyset$.

Aufgabe 1 (Graphen):
Eine Kante e eines Graphen G heißt *Brücke*, wenn $G \backslash e$ $(= (V(G), E(G) \backslash \{e\})$ mehr Zusammenhangskomponenten hat als G.

Ein zusammenhängender Graph heißt *unicyclic* wenn er genau einen Kreis enthält. Zeige, dass die folgenden Aussagen äquivalent sind:

Alg. Prinzipien

- Divide and Conquer
- Greedy
- Dynamische Programmierung

(16) Greedy

- in jedem Schritt die Wahl treffen, die „am besten" aussieht
- Idee: optimale lokale Entscheidungen führen zum globalen Optimum
- ABER: Greedy-Algorithmen führen nicht immer zum Optimum

↳ Beispiel: TSP

Greedy-TSP

- starte in Knoten v
- wähle günstigste Kante e = (v, w) mit w gehört noch nicht zur Tour
- mache bei w weiter

2. Divide an Conquer
- Zerlege Probleme in Teilprobleme
- löse Teilprobleme
- Kombiniere Teilprobleme zur Gesamtlösung

Funktioniert leider nicht immer!

3. Dynamische Programmierung
↳ Details in AuD2
+ ggf. im Laufe des Semesters!