GREEDY ALGORITHMEN

(gierige Algorithmen)

a.) "Definition" von greedy Algorithmen

Welche greedy Algorithmen haben wir bisher gesehen?

Beispiele:

- LIST für SCHEDULING ($(2-\frac{1}{m})$-approximativ)

repeat {
  - bearbeite die Jobs in Reihenfolge der Eingabe
  - teile den nächsten Job einer Maschine zu
  - revidiere diese Entscheidung später nicht.

- LPT (Longest Processing Time) für SCHEDULING ($\frac{4}{3}$-approximativ)

repeat {
  - bearbeite die Jobs in absteigender Folge der Laufzeiten
  - teile den nächsten Job einer geeigneten Maschine zu
  - revidiere diese Entscheidung später nicht

- Greedy Algorithmus für CLIQUE ($\leq \frac{n}{3}$-approximativ)

repeat {
  - nehme den Knoten mit höchstem Grad
  - entferne nicht-adjazente Knoten
  - die ausgewählten Knoten werden nicht zurückgenommen

- Greedy Algorithmus für VERTEX COVER (2-approximativ)

repeat {
  - nehme beliebige Kante (ihre Endknoten)
  - entferne die adjazenten Kanten
  - die ausgewählten Endknoten werden später nicht zurückgenommen

- Greedy Algorithmen für COLORING

⋮

G2.

Grobe Definition:

Ein Greedy Algorithmus bestimmt eine Lösung iterativ; jedes mal wird eine Entscheidung getroffen, die lokal am vielversprechendsten ist. Getroffene Entscheidungen werden nicht revidiert.

Eine konkretere definierbare Sorte von Greedy Algorithmen:
Besser: ein Versuch Greedy Algorithmen konkreter zu definieren.
Priority Algorithmen

- Jede Eingabe besteht aus Datenelementen; es gibt eine festgelegte vollständige Ordnung auf allen möglichen Datenelementen (man kann zB. für einen Alg. festlegen, dass längere Jobs höhere Priorität haben als kürzere, unabhängig von konkreten Instanzen)
- In jedem Schritt erhält der Algorithmus das Datenelement das aktuell die höchste Priorität hat, und trifft eine nicht-revidierbare Entscheidung für dieses Datenelement. Die (Prioritäten der) restlichen Datenelemente können sich in jeder Iteration ändern.

Beispiele:

	Datenelement	Vollständige Ordnung	Entscheidung
LPT	Job mit Laufzeit (p_i)	nach fallenden Laufzeiten	Zuweisung zu einer Maschine
Greedy-CLIQUE	Knoten mit Grad $(v, \deg(v))$	nach fallendem Grad	Hinzunahme zur Clique, falls bisher adjazent
Greedy-VC	Kante (u,v)	Beliebig	Hinzunahme der Endpunkte zur Knotenüberdeckung falls bisher nicht adjazent

b.) Optimale Greedy Algorithmen (Wiederholung)

1. Intervall - Scheduling

Eingabe: n Aufgaben (Jobs) $A_1\ A_2\ A_3\ \dots\ A_n$

jeweils mit Startzeiten $s_1\ s_2\ s_3\ \dots\ s_n$

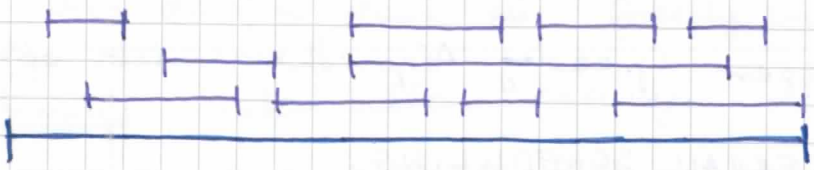
und Endpunkten $e_1\ e_2\ e_3\ \dots\ e_n$

Ausgabe: eine größtmögliche Teilmenge von Jobs (d.h. mit den meisten Jobs drin) die ohne Überlappen auf einer Maschine ausführbar sind

($A_{i_1}\ A_{i_2}\ \dots\ A_{i_m}$ s.d.

wenn $k \neq l$ dann $[s_{i_k}, e_{i_k}) \cap [s_{i_l}, e_{i_l}) = \emptyset$)

Jobs:



ganzes Intervall (Maschine)

Es gibt einen Greedy Algorithmus, der eine optimale Lösung ausgibt.

Die Datenelemente werden offensichtlich die Jobs sein; aber was bestimmt die Priorität?

Man kann leicht einsehen, dass nach (aufsteigender) Startzeit, Länge oder Kürze zu sortieren, kein optimales Ergebnis resultiert.

G4.

Greedy Intervall-Scheduling

- sortiere die Aufgaben gemäß aufsteigenden Endpunkten

  Bezeichne $e^1 \leq e^2 \leq e^3 \leq \ldots \leq e^n$ die sortierten Endpunkte,

  und $A^1\ A^2\ A^3 \ldots A^n$ die entsprechenden Aufgaben

  und $s^1\ s^2\ s^3 \ldots s^n$ die jeweiligen Startpunkte

- Bezeichne J die Menge der ausgewählten Jobs

  Setze $J = \emptyset$, und $E = 0$

- FOR $k = 1$ to n DO

  IF $s_k \geq E$ THEN $J = J \cup \{A^k\}$ und $E = e^k$

  ELSE verwerfe A^k

Theorem: Dieser Greedy Algorithmus ist optimal für INTERVALL SCHEDULING.

Beweis: Wir zeigen durch Induktion über die Anzahl n der Jobs in der Eingabe, dass Greedy eine maximale Anzahl nicht kollidierender Aufgaben in der Lösung erzielt.

Basisfall:
- für $n=1$ Aufgabe ist Greedy optimal –

Induktionsschritt:
- Angenommen, Greedy ist optimal für $\leq n-1$ Jobs in der Eingabe, wir zeigen, dass er auch für n Jobs optimal ist:

Teil 1: A^1 darf in die Lösungsmenge

Behauptung 1: Es gibt mindestens eine optimale Lösung, die A^1 enthält.

Warum? Nehmen wir eine beliebige optimale Lösung, und sei A^* mit Startpunkt s^* und Endpunkt e^* die erste Aufgabe in dieser Lösung. Falls $A^* = A^1$, die Behauptung gilt. Falls $A^* \neq A^1$, dann $e^* \geq e^1$, weil e^1 so definiert wurde. (e^1 ist minimal).

Deshalb kollidiert A^1 mit anderen Aufgaben der optimalen Lösung nicht (sonst würde A^* auch kollidieren).

Wir tauschen A^* gegen A^1 in der optimalen Lösung, und erhalten somit eine optimale Lösung, die A^1 enthält.

Teil 2: Wir dürfen dann mit Greedy weitermachen

Behauptung 2: Unter allen Lösungen die A^1 enthalten, gibt Greedy eine optimale Lösung aus.

Warum? Lösungen, die A^1 enthalten, enthalten keinen Job der mit A^1 kollidiert; wir dürfen also solche kollidierenden Aufgaben aus der Eingabe I entfernen. Wir entfernen auch noch A^1, weil er in allen diesen Lösungen drin ist. Sei I' diese neue Instanz mit $\leq n-1$ Aufgaben. Laut Induktionsannahme gibt Greedy eine maximale Anzahl nicht-kollidierender Aufgaben aus I' aus.

□

- Die Datenelemente sind hier die Knoten v mit den $D_F(v)$ Werten als Prioritäten. Beachte, dass die $D_F(v)$ Werte sich ändern im Laufe des Algorithmus und streng genommen, sogar ihre Definition ändert sich, weil F wächst. (adaptiver Priority Algorithmus)

- Deshalb werden die Datenelemente, wie beim Huffman-Algorithmus, in einem Heap (oder in einem sog. Fibonacci-Heap) verwaltet.

Laufzeit: $O(|E| \cdot \log|V|)$ (bzw. $O(|E| + |V| \cdot \log|V|)$)

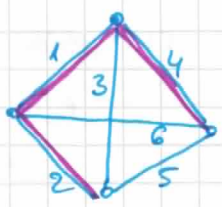
4. Minimaler Spannbaum

Eingabe: ein Graph $G(V, E)$ und Kantengewichtung w_e für jede Kante $e \in E$

Ausgabe: ein Spannbaum von G mit minimalem Gesamtgewicht seiner Kanten

Definition: Ein Spannbaum von $G(V, E)$ ist ein Baum $T(V, E')$ mit derselben Knotenmenge V und $E' \subseteq E$. (Ein Baum ist ein kreisfreier zusammenhängender Graph.)
(Wie viele Kanten enthält E'?)

minimaler Spannbaum:



Kruskal's Algorithmus (Greedy)

- sortiere die Kanten nach aufsteigendem Gewicht
- setze $E' = \emptyset$
- WHILE $E \neq \emptyset$ DO
  - wähle $e \in E$ mit minimalem w_e
    $E := E \setminus \{e\}$
  - falls $E' \cup \{e\}$ keinen Kreis enthält
    $E' := E' \cup \{e\}$
    sonst verwerfe e

Theorem: Kruskal's Algorithmus findet einen minimalen Spannbaum.

Beweis:

Sei $|V| = n$. Jeder Baum auf n Knoten hat $n-1$ Kanten.

- Sei $T(V, E')$ der Spannbaum ausgegeben von Kruskal's Algorithmus, und $T^*(V, E^*)$ ein minimaler Spannbaum mit maximaler Anzahl gemeinsamer Kanten mit T ($|E' \cap E^*|$ sei größtmöglich). Wenn $E' = E^*$, dann ist $T = T^*$ somit ist T minimal und wir sind fertig.
- Wenn $E' \neq E^*$, dann gibt es mindestens eine Kante in E' die in E^* nicht drin ist (da beide $n-1$ Kanten haben)

Minimaler Spannbaum

Thm: Der Algorithmus von Kruskal ist optimal für minimale Spannbäume.

Sei $G(V, E)$ der Eingabegraph, $|V| = n$

Beweis durch Induktion über die Anzahl der Knoten n

Basisfall: $n = 1$ Knoten (trivial)

{ $n = 2$ Knoten ohne Mehrfachkanten  trivial

mit Mehrfachkanten  auch trivial, eine ~~die~~ leichteste Kante ist optimaler Spannbaum

Induktionsschritt: Angenommen, der Kruskal-Algorithmus ist optimal für $n-1$ Knoten, dann ist er auch optimal für n Knoten:

Teil 1. Sei e eine Kante mit minimalem Gewicht, We dann gibt es einen minimalen Spannbaum, der diese Kante e enthält.

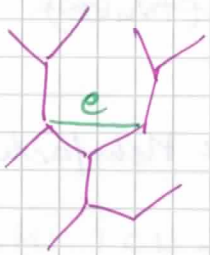
(d.h. der Alg darf mit dieser Kante anfangen, der Kruskal-Algorithmus fängt gut an)

G14.

Sei $T_{OPT}(V, E_{OPT})$ ein minimaler Spannbaum

→ falls $e \in E_{OPT}$, gibt es einen optimalen Spannbaum mit dieser Kante und wir sind fertig mit Teil 1.

→ falls $e \notin E_{OPT}$, verwenden wir ein Austausch-Argument:

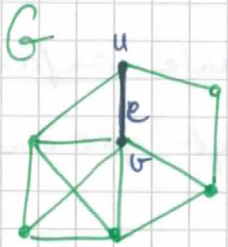


- nimm e zu T_{OPT}
- ein Kreis entsteht
- wir lassen eine beliebige andere Kante vom Kreis weg (wir tauschen sie gegen e)
- das Gewicht wächst nicht, weil w_e minimal.

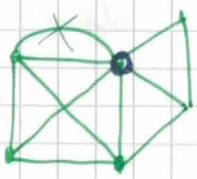
↙ wir haben einen minimalen Spannbaum der e enthält.

Teil 2. Unter allen Spannbäumen, die e enthalten, gibt Kruskal einen minimalen aus (d.h. der Alg. darf mit Kruskal weitermachen...)

wir erstellen eine einfachere Instanz (um die Induktionsannahme für $n-1$ Knoten zu benutzen)



identifiziere die Endknoten von e (kontrahiere e)



wir lassen Mehrfachkanten mit größeren gewichten weg

ein minimaler Spannbaum in G "entspricht" ein minimaler Spannbaum
der e enthält ⟷ in G'

↕ laut Induktionsannahme ist es optimal...

in G mit Kruskal weiterzumachen ⟷ in G' Kruskal durchzuführen

entspricht

C.) Heuristiken für das metrische TSP

(Problem des Handlungsreisenden)

TRAVELING-SALESMAN PROBLEM (TSP)

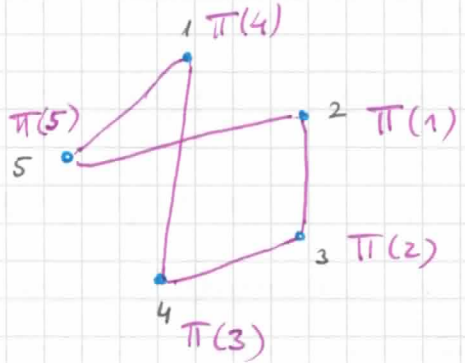
Eingabe: n Orte $\{1, 2, 3, \ldots, n\}$ und ein Distanzwert $d(i,j) \geq$ zwischen je zwei Orten (so dass $d(i,i) = 0$ und $d(i,j) = d(j,i)$ $\forall i,j$)

Ausgabe: Eine Rundreise minimaler Länge, also eine Permutation $\pi: \{1, 2 \ldots n\} \longrightarrow \{1, 2, \ldots n\}$ der Orte so dass

$$\sum_{i=1}^{n-1} d\left(\pi(i), \pi(i+1)\right) + d\left(\pi(n), \pi(1)\right)$$

minimiert wird

Ein Handlungsreisender soll alle Orte aufsuchen, und schließlich nach Hause (zum Startpunkt) zurückkehren, so dass die Länge der gesamten Reise minimiert wird.
($d(i,j)$ kann auch als Reisekosten von i nach j aufgefasst werden)



Die Rundreise ist also ~~eindeutig bestimmt durch die Permutation~~ $(\pi(1), \pi(2), \pi(3), \ldots, \pi(n))$ und wieder $\pi(1)$
(wobei jeder Ort genau einmal vorkommt (~~weil~~ weil π eine Permutation ist).

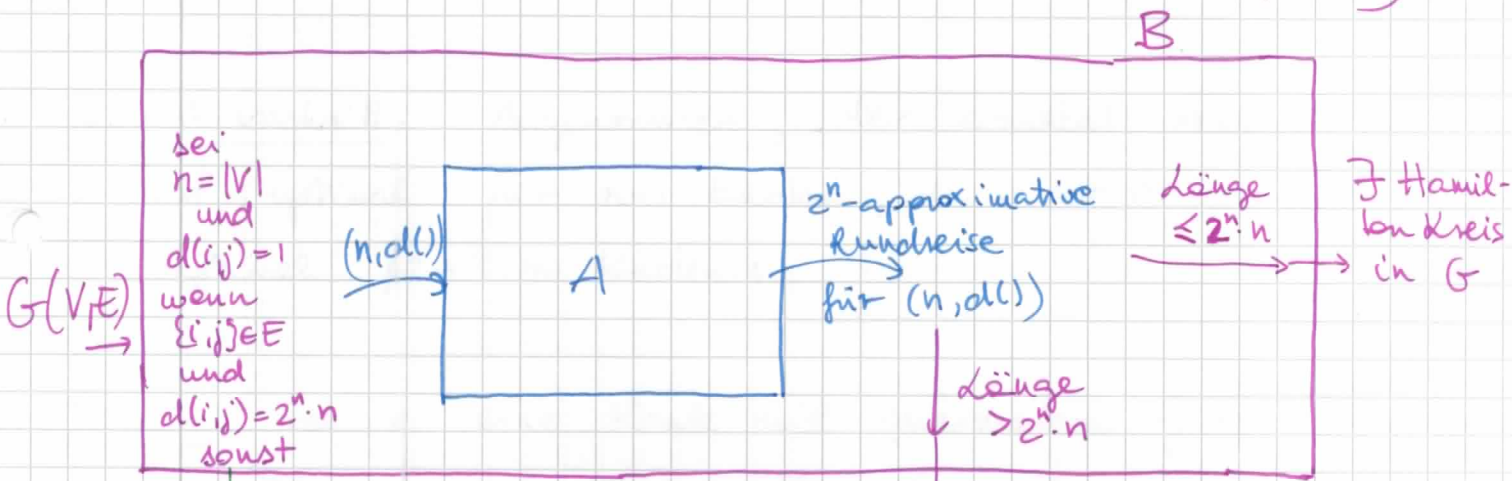
Travelling-Salesman-Problem

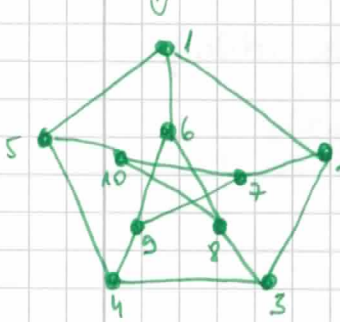
Thm: Das allgemeine TSP ist polynomiell nicht approximierbar um irgendeinen vernünftigen Faktor, zB. um Faktor $\alpha(n) = O(2^n)$.

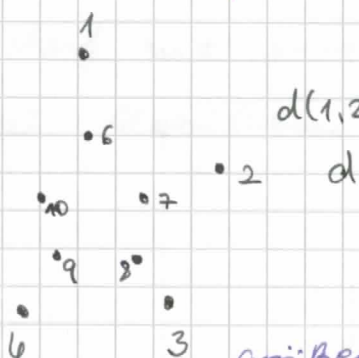
Wir zeigen den Beweis für $\alpha(n) = 2^n$

→ Nehmen wir das Gegenteil an, dass es einen polynomiellen Algorithmus A gibt, der für jede Instanz $I = (n, d())$ von TSP, eine $\leq 2^n \cdot OPT(I)$ lange Rundreise ausgibt.

→ Wir zeigen, dass in diesem Fall ein Algorithmus B das HAMILTONSCHER KREIS Problem in polynomieller Laufzeit entscheiden kann (Angenommen $P \neq NP$, ist das ein Widerspruch, weil HAMILTONSCHER KREIS NP-vollständig ist)







$d(1,2) = 1$

$d(1,3) = 10 \cdot 2^{10}$

usw.

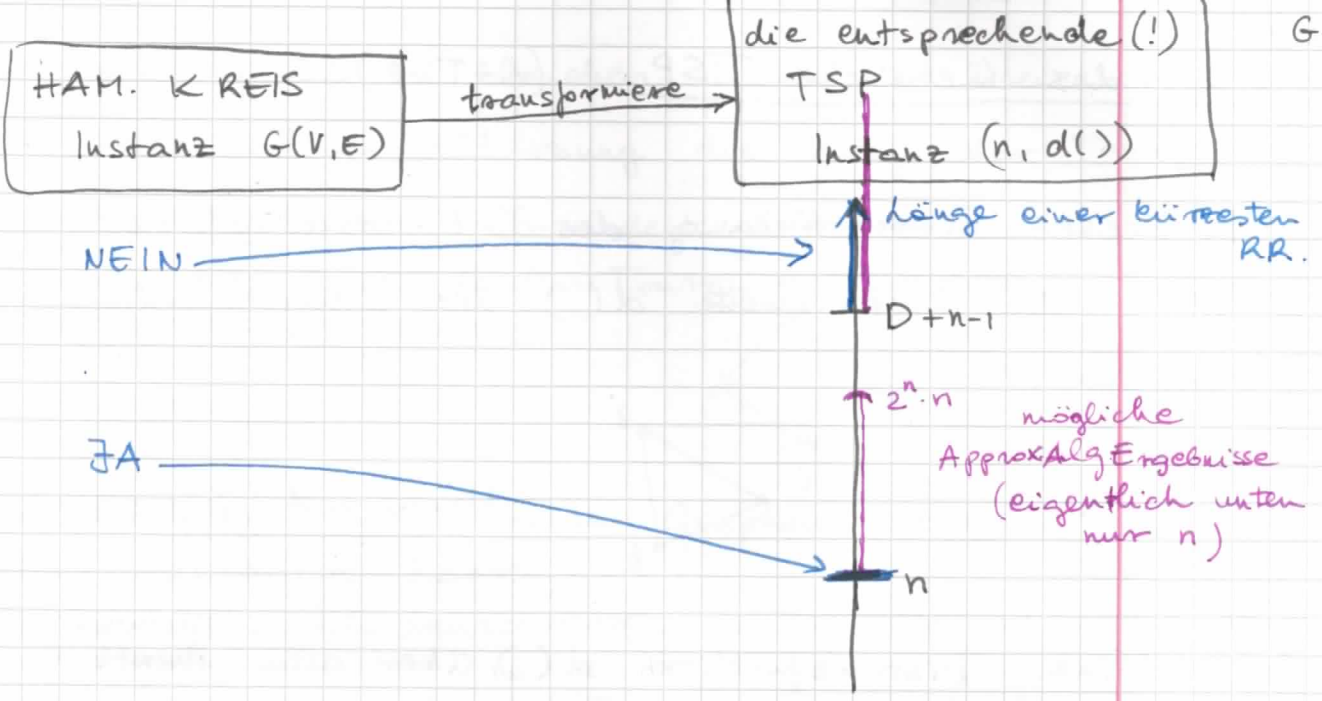
Jede Rundreise ist n-lang oder größer als $2^n \cdot n$ lang!

§ 18.

es gibt Hamilton Kreis in $G(V,E)$ $\Leftrightarrow$ $\exists$ Rundreise mit Länge n für $(n, d(\,))$ $\Leftrightarrow$ A muss eine RR. mit Länge $\leq 2^n \cdot n$ ausgeben (eine n-lange RR. eigentlich)

es gibt keinen Hamilton Kreis in $G(V,E)$ $\Leftrightarrow$ jede Rundreise für $(n, d(\,))$ hat Länge $> 2^n \cdot n$ $\Leftrightarrow$ A gibt eine solche aus

Wir haben HAMILTONSCHER-KREIS auf die 2^n-Approximierung von TSP reduziert.



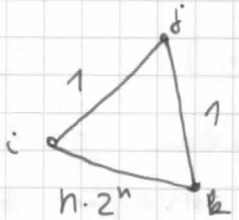
in der Menge der möglichen Zielwerte haben wir eine Lücke geschafft

Angenommen, A ist ein effizienter 2^n-approximativer Algorithmus für TSP. Wir zeigen, dass A das HAM. KREIS effizient entscheidet.

Setzen wir $D = 2^n \cdot n + 1$, (dann gilt $D+n-1 = n2^n + n > n \cdot 2^n$).
$\leq n \cdot 2^n$

- wenn A eine Rundreise der Länge $< D+n-1$ ausgibt, (für so definierte TSP-Instanzen) kann das nur eine Rundreise der Länge n sein $\Rightarrow \exists$ Hamiltonscher Kreis in G
- wenn A eine Rundreise der Länge $> n \cdot 2^n$ ausgibt,
  $\Rightarrow$ es gibt keine Rundreise der Länge n, weil A 2^n-approximativ
  $\Rightarrow \nexists$ Hamiltonscher Kreis in G □

Das Problem mit dem allgemeinen TSP ist, dass sowas vorkommen kann:



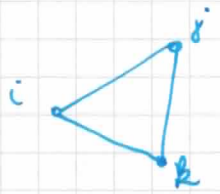
Als Nächstes, beschränken wir uns (d.h. die möglichen Instanzen) auf ein realistischeres Modell ...

wir wollen solche Dreiecke ausschließen in der Instanz (n,d())

G 20.

2. das metrische TSP (Δ-TSP)

In dieser Einschränkung des TSP wird gefordert, dass die Distanzwerte $d(\,)$ die Dreiecksungleichung erfüllen:



$$d(i,k) \leq d(i,j) + d(j,k)$$

Def: Eine Distanzfunktion $d(\,)$ über alle Element-Paare einer Menge heisst eine Metrik, falls

1. $d(i,j) = 0 \iff i = j$
2. $d(i,j) = d(j,i)$ Symmetrie
3. $d(i,k) \leq d(i,j) + d(j,k)$ Dreiecksungleichung o. Transitivität

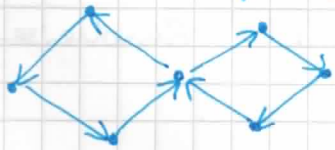
(Metrische Distanzen nicht mit den Euklidschen Distanzen verwechseln! Euklidische Distanzen entsprechen einer Metrik, aber es gibt viele andere Metriken, zB. Hamming-Distanz, oder $d(i,j) = 1 \iff i \neq j$ sind auch Metriken.)

Das metrische TSP kann man um konstanten Faktor effizient approximieren.

Heuristiken

Die ersten beiden Heuristiken suchen eine sog. Euler-Tour über alle Orte. Hierfür brauchen sie als Erstes einen Eulerschen Graphen über alle Orte (zusammenhängender Graph mit geradem Grad für alle Knoten)
Die Euler-Tour kann man anschliekend kürzen, um jeden

Ort nur einmal zu besuchen. Dank der Dreiecksungleichung, wird jede solche Abkürzung die Länge der Tour tatsächlich kürzen (zumindest nicht verlängern)
(Achtung! Nur kurze Euler-Touren mit möglichst wenigen Kanten sind brauchbar)



Zur Erinnerung:

Eine Euler-Tour in einem Graphen ist eine Tour über adjazente Kanten,
(Folge von jeweils benachbarten Knoten, die im Startknoten endet), die jede Kante genau einmal durchläuft,
(die Knoten dürfen mehrmals besucht werden).

Theorem (Euler): Ein zusammenhängender, ungerichteter Graph $G(V,E)$ enthält eine Euler-Tour

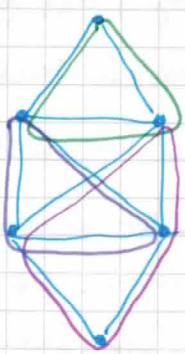
$\Updownarrow$

alle Knoten haben geraden Grad

Beweis:

$\Downarrow$ diese Richtung ist trivial, wenn die Tour in einen Knoten reinläuft, muss über eine neue Kante wieder rauslaufen

$\Uparrow$ Wir laufen von Kante zur Kante, solange es geht. Jedes mal, wenn es nicht weitergeht, haben wir einen Kreis (von Kanten) geschlossen. Da der Graph zusammenhängend ist, können all diese 'Kreise' (Touren) letztendlich in eine einzige Tour geschmolzt werden.
(wenn unser Kreis noch nicht jede Kante durchläuft, kann er verlängert werden (iteratio) um einen weiteren adjazenten Kreis)



G22. Betrachte die Orte im TSP Problem als eine Menge von Knoten.

Die nächsten beiden Heuristiken erstellen einen zusammenhängenden Graphen mit geradem Knotengrad für jeden Knoten und berechnen eine Euler-Tour mit Abkürzungen auf diesem Graphen, als Rundreise über die Orte.

i.) Die Spannbaum-Heuristik

- Betrachte den vollständigen Graphen über alle Orte als Knoten, und mit den Distanzen $d(i,j)$ als Kantengewichten / Längen K_n
- Bestimme einen minimalen Spannbaum in diesem vollständigen Graphen mit Kruskal's Algorithmus T

(Der Spannbaum spannt zwar alle Orte, es gibt noch aber Knoten mit ungeradem Grad ...)

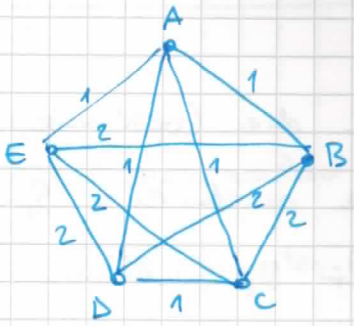
- verdoppele jede Kante im Spannbaum $\hat{T}$

(jetzt haben wir einen Spanngraphen mit geraden Knotengraden)

- nimm eine Euler-Tour in $\hat{T}$; z.B. ein Präorder Durchlauf (Tiefensuche) und dann der Rückweg zur Wurzel entspricht einer Euler-Tour.
- Mache Abkürzungen wo nötig, um jeden Ort nur einmal zu besuchen (beim ersten Mal).

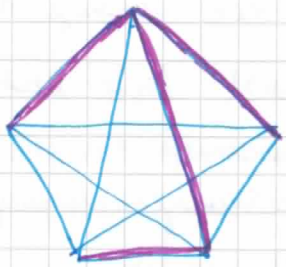
(Achtung: wenn der kürzeste Weg von j nach k entlang Ort i führt, und i schon besucht wurde früher, darf die Rundreise natürlich an i vorbeiführen. ZB. in der Eukl. Ebene, oder in einem Straßennetzwerk / Graphen.)

Beispiel:

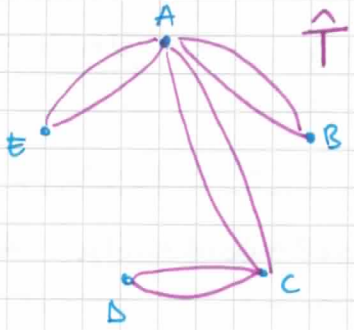


K_n

die Orte mit ihren Distanzwerten



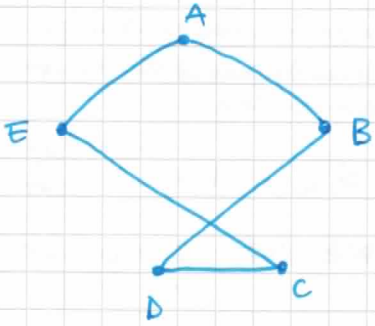
ein minimaler Spannbaum T



$\hat{T}$

verdoppelte Kanten und eine Euler-Tour:

A E A C D C A B A



die Rundreise = Euler-Tour mit Abkürzungen

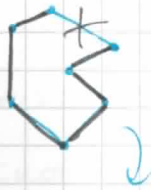
A E C D B (A)

Theorem: Die Spannbaum-Heuristik ist 2-approximativ.

Warum? Sei l_{SH} die Länge der Rundreise ausgegeben von der Spannbaum-Heuristik, und l_{OPT} die optimale Länge. Sei noch $l(T)$ die Gesamtlänge der Kanten im Spannbaum.

G 24.

Es gelten:



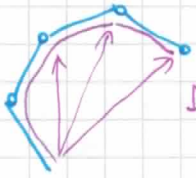
$l_{OPT} > l(T)$ weil die (optimale) Rundreise mit sogar minus eine Kante auch ein Spannbaum ist, und T ein minimaler Spannbaum ist.

$l_{SH} \leq 2 \cdot l(T)$ weil die Länge der Euler-Tour genau $2 l(T)$ ist, und l_{SH} hat noch Abkürzungen im Vgl. zur Euler-Tour.

(Beachte, dass ohne der Dreiecksungleichung eine „Abkürzung" zu einer Verlängerung führen könnte

(ohne Dreiecksungleichung wird „Vorbeifahren" auch nicht definiert)).

Mit Dreiecksungleichung kann aber eine Abkürzung nicht länger werden als vorher ohne Abkürzung:



man kann die Dreiecksungleichung iterativ oder durch Induktion anwenden)

Wir erhalten: $l_{SH} \leq 2 \cdot l(T) < 2 l_{OPT}$ also die Spannbaum-Heuristik ist 2-approximativ.

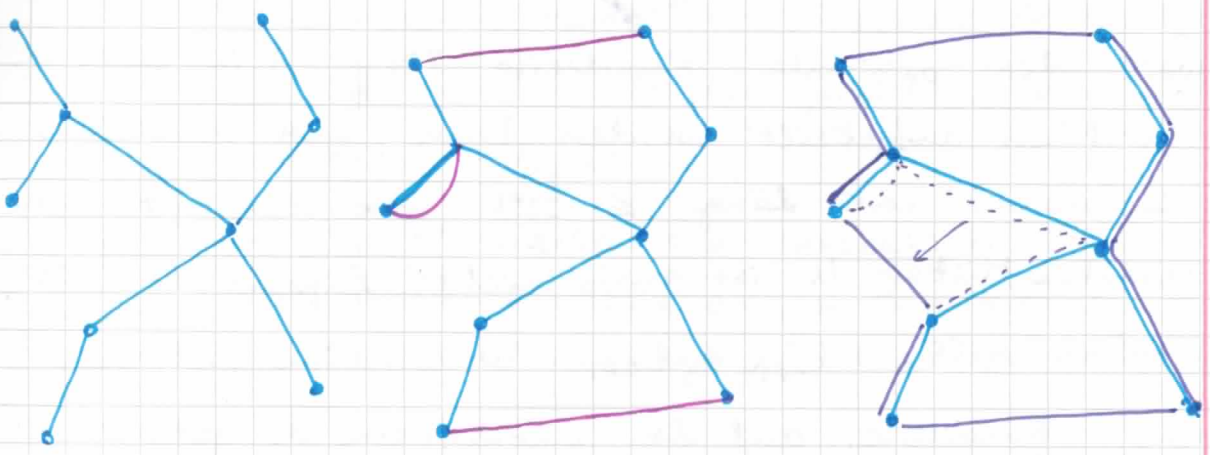
Der folgende Algorithmus definiert den Eulerschen Graphen (d.h. mit den geraden Knotengraden) raffinierter, und erreicht einen noch besseren Approximationsfaktor:

ii.) der Algorithmus von Christofides für Δ-TSP

- bestimme einen minimalen Spannbaum T in vollständigen Graphen K_n mit Kantenlängen $d(i,j)$

  (es gibt noch Knoten in T mit ungeradem Grad)

- Bezeichne U die Menge der Knoten mit ungeradem Grad in T (|U| ist gerade, weil die Summe aller Knotengrade immer gerade ist $2 \cdot |E| = 2 \cdot (n-1) \Rightarrow$ gerade)
- Berechne ein perfektes Matching mit minimalem Gesamtlänge seiner Kanten über die Knoten von U. Sei M dieses Matching.
- Nimm die Kanten in M zu den Baumkanten in T hinzu: jetzt hat jeder Ort geraden Grad. (Weil die Knoten in U jetzt +1 Grad haben von M.)
- Bestimme eine Euler-Tour in T ∪ M, und dann mache Abkürzungen um jeden Ort genau einmal zu besuchen.



T T ∪ M Rundreise

G 26.

Theorem: Der Algorithmus von Christofides ist $\frac{3}{2}$ - approximativ.

Beweis: Sei ℓ_{CH} die Länge der Rundreise ausgegeben von Christofides

Wir wissen:

$$\ell(T) < \ell_{OPT}$$

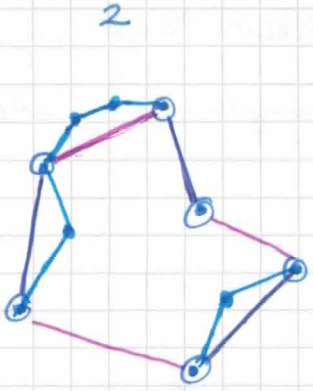
und $\ell_{CH} \leq \ell(T) + \ell(M)$

(wobei $\ell(M)$ ist die Gesamtlänge des optimalen Matchings über die Knotenmenge U)

Wir zeigen noch, dass $\ell(M) \leq \frac{\ell_{OPT}}{2}$

Sei ℓ^U_{OPT} die Länge einer optimalen Rundreise über die Menge $U \subseteq \{1, 2, \dots, n\}$ (der Knoten mit ungeradem Grad in T)

Dann ist $\ell(M) \leq \frac{\ell^U_{OPT}}{2}$



weil die optimale Rundreise über U in zwei Matchings aufgeteilt werden kann, und mindestens eins davon hat Länge $\leq \frac{\ell^U_{OPT}}{2}$, also ein optimales Matching über U ist auch nicht länger als $\frac{\ell^U_{OPT}}{2}$.

Weiterhin gilt $\ell^U_{OPT} \leq \ell_{OPT}$ weil eine optimale (ganze) Rundreise, auf die Knoten von U gekürzt werden kann. Somit gilt $\ell(M) \leq \frac{\ell^U_{OPT}}{2} \leq \frac{\ell_{OPT}}{2}$.

Schließlich: $\ell_{CH} \leq \ell(T) + \ell(M) \leq \ell_{OPT} + \frac{\ell_{OPT}}{2} = \frac{3}{2}\ell_{OPT}$

□

Laufzeit: $O(n^3)$ (Die Berechnung eines perfekten Matchings mit minimalem Gewicht braucht $\Theta(|V|^3) = O(n^3)$ Zeit und dies dominiert die Laufzeit; die anderen Schritte sind linear.) $\Rightarrow O(n)$

[Bemerkung: es wurde gezeigt (2011) dass mit einer geschickten Wahl des Spannbaums ein kleinerer Approximationsfaktor erreichbar ist, wenn die Metrik durch die Distanzen in einem Graphen definiert wird.]

Die folgenden beiden (greedy) Heuristiken haben schlechteren Approximationsfaktor im Worst-Case als Christofides.

iii) **Nearest-Insertion Heuristik** — 2-approximativ

- sei $V = \{1, 2, \ldots, n\}$
- sei $\{i, j\}$ ein Paar von Orten mit maximaler minimaler Distanz $d(i,j)$
- setze $V := V \setminus \{i, j\}$
- sei die (partielle) Rundreise $R = (i, j, i)$
- REPEAT
  - nimm einen Ort $k \in V$ mit weitestem einzelnem Abstand zu einem Ort in der aktuellen partiellen Rundreise R
  - sei $V := V \setminus \{k\}$
  - füge k zwischen zwei benachbarten Orten in der partiellen Rundreise ein, so dass dies den kleinsten Anstieg der Länge der Rundreise resultiert

  UNTIL $V = \emptyset$

iv) **Farthest-Insertion Heuristik** — $6.5 \leq$ Approx-Faktor $\leq \log n + 1$

In der Euklidschen Ebene $\mathbb{R}^2$ mit den gewöhnlichen (Euklidschen) Distanzen, fügt Nearest-Insertion und Farthest-Insertion Heuristic mit einer konvexen Hülle aller Orten als partieller Rundreise R an.

Durchschnittlicher Approx.-Faktor im Experiment mit je 1000 zufälligen Punkten $\mathbb{R}^2$

Christofides: $\alpha \sim 1.1$
Farthest-Insertion: 1.1
Nearest-Insertion: $\alpha \sim 1.25$
Spannbaum: $\alpha \sim 1.4$

3. Das Euklidsche TSP

Eingabe: n Punkte $V \subseteq \mathbb{R}^d$ mit den euklidschen Distanzen

$$d(a,b) = \sqrt{(a_1-b_1)^2+(a_2-b_2)^2+\ldots+(a_d-b_d)^2}$$

für jede $a = (a_1\ a_2 \ldots a_d) \in V$
$b = (b_1\ b_2 \ldots b_d)$

Ausgabe: eine kürzeste Rundreise die alle n Punkte genau einmal besucht.

(Bemerkung: 

(a, b, c) und zurück zum a zählt auch als Rundreise, b wird nicht nochmal "besucht" wenn es auf dem kürzesten Weg von c nach a liegt)

Theorem: Für das euklidsche TSP gibt es ein PTAS (Arora's PTAS), also dieser Spezialfall des TSP Problems ist beliebig nah approximierbar. (siehe später)