BRANCH & BOUND

(für Optimierungsprobleme)

a) Einführung, Grobstruktur

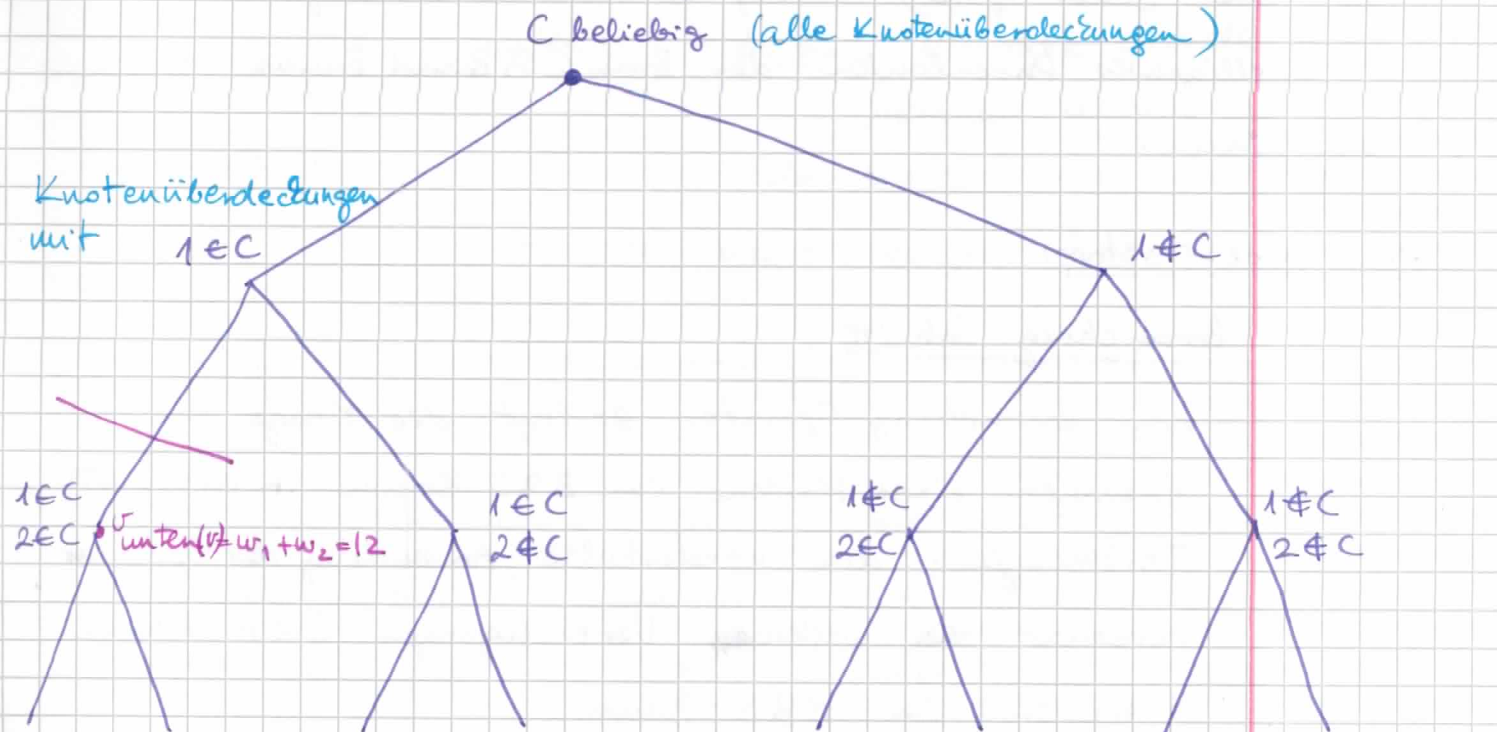
Bei schwierigen Problemen ist der Lösungsraum groß (es gibt mindestens exponentiell-viele mögliche Lösungen).

Theoretisch könnte man eine vollständige (erschöpfende Suche) Suche (die alle Lösungen betrachtet und eine optimale auswählt) gemäß eines „Suchbaums" (hier später: Branch & Bound Baum) durchführen, wie im folgenden einfachen (theoretischen!) Beispiel:

(Dummes) Beispiel 1. VERTEX COVER

Sei $G(V,E)$ ein Graph mit $V = \{1, 2, 3, \dots, n\}$ und Knotengewichten $w_1, w_2, \dots w_n$

Finde eine Knotenüberdeckung $C \subseteq V$ mit minimalem Gewicht.



Zweige, wo sich keine Knotenüberdeckungen mehr befinden kann, können abgebrochen werden (z.B. hier wenn $\{1,2\} \in E$)

Tiefe des Baumes: n

Laufzeit $\Omega(2^n)$ im Worst-Case, wenn alle möglichen Lösungen geprüft werden!

(Warum? Schließlich entspricht jedes Blatt einer Lösung C. Die Anzahl der Blätter kann für die meisten Graphen $G(V, E)$, $\Omega(2^n)$ werden (d.h. die Zweige nicht früher abgeschnitten).)

Was ist nun Branch & Bound?

Angenommen, eine Heuristik fand am Anfang eine Knotenüberdeckung mit Gewicht 10. Seien außerdem $w_1 = 7$ und $w_2 = 5$ die Gewichte der Knoten 1 bzw. 2.

In diesem Fall braucht der Algorithmus im linkesten Teilbaum gar nicht weiterzusuchen, dort werden alle C mindestens Gewicht $7+5=12$ haben!

Ein Branch & Bound Algorithmus (Heuristik) versucht eine optimale (oder gute) Lösung durch eine Intelligente Suche — intelligentes Durchlaufen des Branch & Bound Baums — zu finden.

Die Grobstruktur:

Im Branching Schnitt

- ein Branching Operator zerlegt die Menge von Lösungen eines Knotens im B&B-Baum in disjunkte Teilmengen; die wiederholte Anwendung des Operators erzeugt weitere Verzweigungen (d.h. Kanten u. Knoten) im B&B-Baum.

Die Wurzel entspricht der Menge aller Lösungen. Sei v ein Knoten mit Lösungsmenge $L(v)$ und Kindern $v_1, v_2 \ldots v_i$ im Baum. Dann ist $L(v_1), L(v_2), \ldots, L(v_i)$ eine Zerlegung (Partitionierung) von $L(v)$.

(z.B. die Menge

$\{ C \text{ Knotenüberdeckung} \mid 1 \in C, 2 \notin C \}$

kann zerlegt werden in die Lösungsmengen

$\{ C \text{ Knotenüberdeckung} \mid \{1,3\} \subset C, 2 \notin C \}$

und $\{ C \text{ Knotenüberdeckung} \mid 1 \in C, 2,3 \notin C \}$)

Bounding Schritt: (sei $\mathcal{B}$ der B&B-Baum)

- angenommen, für jeden Knoten $v \in \mathcal{B}$ gibt es eine (effizient berechenbare) untere Schranke unten(v) so dass

$$\text{unten}(v) \leq f(y)$$

für jede Lösung in $y \in L(v)$ gilt (f ist die Zielfunktion).

→ v kann verworfen werden, wenn

$\text{unten}(v) \geq f(y_0)$ für eine aktuelle beste bekannte Lösung y_0.

(das Blatt v wird deaktiviert)

Wie in unserem Beispiel mit

$$10 = f(y_0) \leq \text{unten}(v) = w_1 + w_2 = 12$$

($f(y_0)$ ist eine aktuelle globale obere Schranke für das Optimum, die unten(v) sind lokale untere Schranken für die Teilbäume (mit Wurzel v jeweils))

BB4.

Branch & Bound Algorithmus (für Minimierungsprobleme oBdA.)

→ anfänglich besteht der B&B Baum $\mathcal{B}$ nur aus der <u>aktivierten</u> Wurzel

→ eine Lösung y_0 wird mit Hilfe einer Heuristik (o. Approximationsalgorithmus) berechnet

→ WHILE es gibt aktivierte Blätter, DO

- wähle das erfolgversprechendste aktivierte Blatt v in $\mathcal{B}$
- wende den Branching-Operator auf v an, um die Kinder $v_1\ v_2 \dots v_k$ zu erhalten
- FOR $i = 1$ TO k DO (berechne unten(v_i)) ~~und eine Lösung $y_i \in L(v_i)$~~

  IF unten$(v_i) \geq f(y_0)$ deaktiviere v_i

  (ggf. berechne eine Lösung $y_i \in L(v_i)$

  IF ~~[illegible]~~ $f(y_i) < f(y_0)$ setze $y_0 = y_i$

  und deaktiviere ggf. Blätter)

→ gib y_0 als Lösung aus

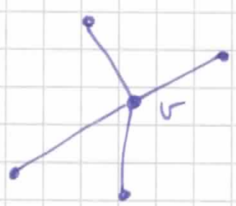
- Die Anfangslösung y_0, bzw y_i soll möglichst nah am Optimum (von $L(v_i)$) liegen, damit schlechte andere Zweige (Knoten) frühzeitig deaktiviert werden
- unten(v_i) soll möglichst nah am Optimum in $L(v_i)$ liegen damit v_i ggf. frühzeitig deaktiviert wird
- Was ist das „erfolgversprechendste Blatt" von $\mathcal{B}$?
  - best-fit-search: wähle Knoten mit der niedrigsten unteren Schranke
  - Tiefensuche: schont den Speicherplatzverbrauch

<u>Bemerkung</u>: Der B&B Baum $\mathcal{B}$ entspricht auch etwa dem Rekursionsbaum eines rekursiven Algorithmus (nicht polynomiell), wie im Fall der Dynamischen Programmierung. In der Dynamischen Programmierung führen die Überlappungen der Teilprobleme zu einem effizienten Algorithmus. Bei B&B hilft die frühzeitige Erkennung schlechter Lösungen, bzw. das nicht jeder Zweig durchlaufen werden soll.

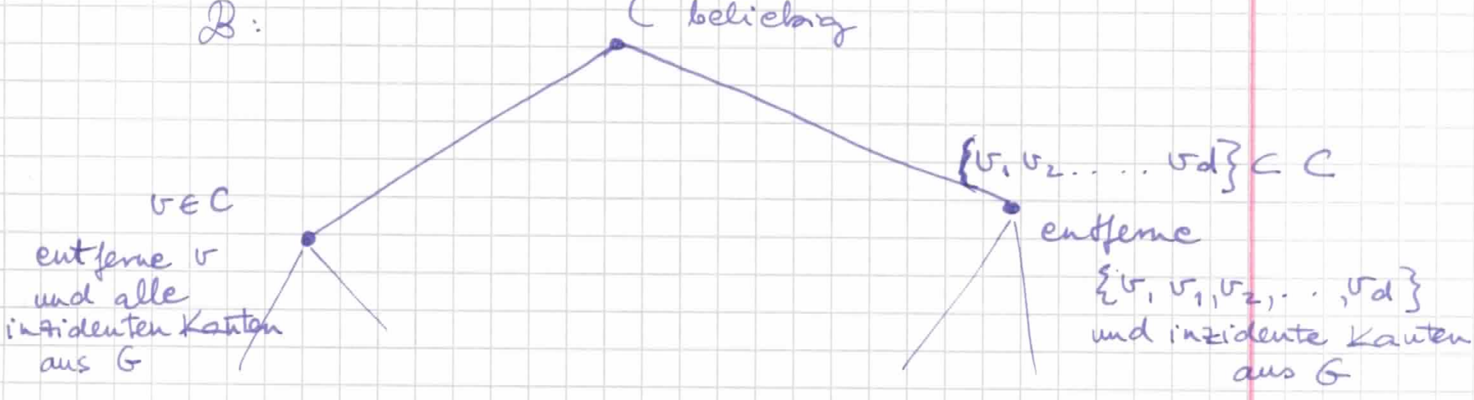
Lösungsmengen ⟷ Teilprobleme

<u>Beispiel 2.</u> Ein besserer Branching-Operator für VERTEX COVER im ungewichteten Fall

- wähle einen Knoten $v \in V$ mit $\overset{d}{\overset{\|}{\deg(v)}} \geq 3$ und Nachbarn $v_1 v_2 \dots v_d$



entweder $v \in C$ oder $\{v_1 v_2 \dots v_d\} \subset C$



- falls nur Knoten mit Grad ≤ 2 gibt, finde eine minimale Knotenüberdeckung

→ (ohne Bounding Schnitte) hat dieser Algorithmus Worst-Case Laufzeit $O(1{,}38^n)$ (vielleicht auch mit ...)

BB6.

6.) Branch & Bound für RUCKSACK

Eingabe: n Objekte mit Gewichten $g_1\, g_2 \dots g_n$
und Werten $w_1\, w_2 \dots w_n$
eine Gewichtsschranke G

Ausgabe: Finde eine Auswahl von Objekten $I \subset \{1, 2, \dots, n\}$ mit maximalem Gesamtwert, so dass die Gewichtsschranke nicht überstiegen wird

Überlegungen:

- Welche Lösungsmengen (bzw. Teilprobleme) entsprechen den Knoten des B&B Baums?
  - (J, i) bezeichne alle Bepackungen so dass von den ersten i Objekten genau die Menge $J \subseteq \{1, 2, 3, \dots, i\}$ eingepackt wird

    (es gibt 2^{n-i} solche Teilmengen von $\{1, 2, \dots n\}$)

    (Beachte: hier werden auch in jedem Knoten manche Objekte erzwungen, manche andere verboten)
- Wir haben hier ein Maximierungsproblem: in jedem Knoten von $\mathcal{B}$ werden wir eine obere Schranke oben(v) berechnen; höchstens diesen Wert erreichen Bepackungen in $L(v)$.
- Wie finden wir eine obere Schranke oben(v) für die Werte im Teilbaum von $v = (J, i)$?
- Wie finden wir eine gute Anfangslösung y_0 ?

– Wie finden wir eine gute Lösung in $L(v) = L(\mathcal{Y}, i)$?

Exkurs: Die Bestimmung einer oberen Schranke $oben(v)$

der Einfachheit halber sei $oben(v)$ gesucht für das ganze Problem $v = (\emptyset, 0)$

Wenn wir für RUCKSACK einen greedy Algorithmus entwerfen sollten, in welcher Reihenfolge würden wir die Objekte betrachten / in den Rucksack packen?

→ nach absteigendem Wert? (NEIN, siehe zB. ($w_i \sim 1$ aber $g_1 = G$))

→ nach aufsteigendem Gewicht? (NEIN, siehe ($g_i \sim 1$ aber $w_1 = 0$))

→ Idee: sortieren wir die Objekte nach ihrem „Wert pro Kilo" $\frac{w_i}{g_i}$ und packen wir sie in absteigender Reihenfolge in den Rucksack:

$$\frac{w_1}{g_1} \geq \frac{w_2}{g_2} \geq \frac{w_3}{g_3} \geq \dots$$

Welchen Approximationsfaktor erreichen wir?

(∞ Approximation, schon mit $n=2$ Objekten; Warum?)

Fraktionale RUCKSACK: beliebige Bruchteile der Objekte dürfen eingepackt werden

LP Formulierung (maximiere $\sum_i x_i w_i$

so dass $\sum x_i g_i \leq G$

$0 \leq x_i \leq 1$)

um eine ~~untere~~ obere Schranke zu bestimmen, relaxieren wir das Problem P (wir lassen auch nicht-Lösungen als Lösungen zu), so dass das relaxierte Problem P' effizient optimierbar ist. Das Maximum für P' kann nicht ~~größer~~ kleiner sein als für P, und ist somit eine obere Schranke.

BB 8.

Greedy Algorithmus für das fraktionale RUCKSACK

- sortiere die Objekte nach absteigendem Wert pro Kilo

$$\frac{w_1}{g_1} \geq \frac{w_2}{g_2} \geq \dots$$

- wenn $\sum_{i=1}^{k} g_i \leq G < \sum_{i=1}^{k+1} g_i$ dann

packe die Objekte $1, 2, \dots, k$ ein und noch ein Bruchteil x_{k+1} so dass

$$g_1 + g_2 + \dots + g_k + x_{k+1} \cdot g_{k+1} = G$$

Behauptung 1. Diese Lösung ist optimal für das fraktionale RUCKSACK.

(Wenn eine Lösung nicht die Form

$$(1, 1, 1, \dots, 1, x, 0, 0, \dots, 0)$$ hat, dann

gibt es ein $x_{k'} < 1$ und $x_{k'+1} > 0$:

dann kann vom Objekt $k+1$ y Gewicht ausgeladen und vom Objekt k y Gewicht eingepackt werden ohne den Gesamtwert zu reduzieren.)

Behauptung 2: Der Maximumwert einer ganzzahligen Bepackung ist nicht größer als der Maximumwert einer fraktionalen Bepackung.

$$W_{max} \leq W_{max}^{frac}$$

(die beste ganzzahlige Bepackung ist auch eine fraktionale Bepackung, also höchstens so gut wie die beste fraktionale Bepackung)

$\Rightarrow$ die optimale fraktionale Lösung bestimmt also eine obere Schranke $\sum_{i=1}^{k} w_i + x_{k+1} w_{k+1} = W_{OPT}^{frac}$ für das Optimum von RUCKSACK.

Wir fassen zusammen (die Antworte auf unsere Fragen)

- jeder Knoten v von $\mathcal{B}$ entspricht einem Paar $v = (J, i)$ für $1 \le i \le n$ und $J \subseteq \{1, 2, \ldots, i\}$

  $L(v)$ ist die Menge aller Bepackungen, die von den ersten i Objekten genau die Teilmenge J enthalten

- für $v = (J, i)$ erzeugt der Branching-Operator die Kinder

$$v = (J, i)$$

$$v_1 = (J \cup \{i+1\}, i+1) \qquad v_2 = (J, i+1)$$

- um gute approximative Lösungen zu finden, wenden wir den approximativen dynamischen Programmier-Algorithmus an (PTAS mit Skalieren und Runden)

  $\boxed{\text{auf } \{i+1, i+2, \ldots, n\} \text{ und } G - \sum_{k \in J} g_k}$

  $G - \sum_{k \in J} g_k = G'$

- der (fraktionale) greedy Algorithmus bestimmt die obere Schranke $oben(v)$ für alle Lösungen in $L(v)$

$$\boxed{oben(v) = \sum_{k \in J} w_k + W^{frac,\,opt}_{\{i+1, \ldots, n\},\, G'}}$$

- als erfolgversprechendste Lösung (J, i) wird eine bislang wertvollste Bepackung gewählt.

BB10.

Branch & Bound für RUCKSACK

→ anfänglich besteht $\mathcal{B}$ aus der aktivierten Wurzel $v_0 = (\emptyset, 0)$

und $\text{oben}(v_0) = W^{frac}_{OPT}$

→ eine Lösung I_0 mit Dynamischer Programmierung (PTAS)

wird berechnet mit Wert $W(I_0)$

→ WHILE es aktivierte Blätter in $\mathcal{B}$ gibt, DO

- wähle das Blatt $v = (\mathcal{J}, i)$ mit $\sum_{k \in \mathcal{J}} w_k$ maximal

(und i minimal)

- seien $v_1 = (\mathcal{J} \cup \{i+1\}, i+1)$ und $v_2 = (\mathcal{J}, i+1)$

$\mathcal{J}_1 = \mathcal{J} \cup \{i+1\}$ $\quad \mathcal{J}_2 = \mathcal{J}$

- $\text{oben}(v_1) = \sum_{k \in \mathcal{J}} w_k + w_{i+1} + W^{frac, OPT}_{\{i+2, \dots, n\}, G_{neu}}$

- IF $\text{oben}(v_1) \leq W(I_0)$ deaktiviere v_1

ELSE sei I' Ergebnis des PTAS auf

$\{i+2, \dots, n\}$ und $G - \sum_{k \in \mathcal{J}_1} g_k$

und $I_1 = \mathcal{J} \cup \{i+1\} \cup I'$

IF $W(I_1) > W(I_0)$

sei $I_0 := I_1$

- $\text{oben}(v_2) = \sum_{k \in \mathcal{J}} w_k + W^{frac, OPT}_{\{i+2, \dots, n\}}$

- IF $\text{oben}(v_2) \leq W(I_0)$ - - - -

analog

→ gib $I_0 = (\mathcal{J}, n)$ mit $\sum_{k \in \mathcal{J}} w_k$ max aus

c.) BRANCH & BOUND für Δ-TSP (metrisches TSP)

Eingabe: n Städte $N = \{1, 2, 3, \ldots, n\}$

und Distanzen $d(i,j)$ zwischen je zwei Städten i, j

die einer Metrik entsprechen

Ausgabe: eine kürzeste Rundreise

(Sei $K_n(N, E)$ der vollständige ungerichtete Graph über N)

Überlegungen:

→ Was seien die Knoten des B&B Baums?

Wir legen Teillösungen fest, (und schränken somit den Lösungsraum ein), wie üblich:

- die Knoten v des Baums werden durch Paare (S, T) beschrieben: $S \subseteq E$ ist eine Menge erzwungener Kanten, die müssen Teil der Rundreise sein

  $T \subset E$ ist eine Menge verbotener Kanten
- das Ausgangsproblem ist $(\emptyset, \emptyset)$ (Wurzel)

Vom Branching-Operator werden natürlich jeweils weitere Kanten entweder erzwungen oder verboten (siehe später).

→ Wie findet man gute Anfangslösung y_0?

Wir kennen gute Heuristiken für Δ-TSP, wie Christofides' Algorithmus, Farthest-Insertion, oder Nearest-Neighbor Heuristik.

Wir haben hier ein Minimierungsproblem, und für ein B&B Algorithmus brauchen wir gute untere Schranke unten(v) für die Teillösungen $v = (S, T)$

→ Die Bestimmung einer guten unteren Schranke unten(v)

Der Einfachheit halber wird unten(v) für die Wurzel $v = (\emptyset, \emptyset)$ (d.h. für das unbeschränkte Problem) gesucht

Untere Schranke mit 1-Bäumen

Wir relaxieren das Δ-TSP Problem, und erhalten ein effizient lösbares (optimierbares) Problem, so dass alle Rundreisen, aber auch andere Teilgraphen Objekte Lösungen dieses relaxierten Problems sind:
Wir beobachten, dass jede Rundreise aus einem Spannbaum und noch einer Kante besteht. Wir werden über solche Objekte optimieren. Präziser: Jede Rundreise besteht aus einem Spannbaum über die Städte $\{2, 3, 4, \ldots, n\}$ und zwei Kanten inzident mit der Stadt 1.
Einen solchen Teilgraphen nennen wir 1-Baum (OneTree), und wir optimieren über alle 1-Bäume. (Beachte, dass ein 1-Baum ein Spanngraph mit genau einem Kreis (über Stadt 1) ist.)

Definition: Ein 1-Baum besteht aus einem Spannbaum über der Städte $\{2, 3, 4, \ldots, n\}$ und zwei mit 1 inzidenten Kanten.

Beobachtung 1. Bezeichne $\mathrm{OneT}(d)$ die minimale Länge über 1-Bäumen, und $\mathrm{TSP}(d)$ die minimale Länge über Rundreisen für die Distanzfunktion $d()$.
Dann gilt $\mathrm{OneT}(d) \leq \mathrm{TSP}(d)$

Warum? Da eine minimale Rundreise selbst ein 1-Baum ist, kann sie nicht kürzer sein als $\mathrm{OneT}(d)$ die minimale 1-Baum Länge.

Somit ist OneT(d) eine untere Schranke!

Beobachtung 2: OneT(d) die minimale 1-Baum-Länge kann effizient berechnet werden.

(Warum? Mit dem Greedy Algorithmus von Kruskal berechnen wir einen minimalen Spannbaum über $\{2, 3, 4, \dots, n\}$; dann verbinden wir Stadt 1 mit diesem Spannbaum über seine beiden kürzesten inzidenten Kanten. Für beide Teile des 1-Baums haben wir jetzt eine minimale Lösung (für den Spannbaum über $N \setminus \{1\}$ und für die zwei Verbindungskanten), die gemeinsam einen 1-Baum ergeben. — besser geht's nicht!)

Somit ist OneT(d) eine schnell berechenbare untere Schranke!

Bemerkung: **Die Festlegung** der Stadt 1 in der Definition von 1-Bäumen macht die Definition asymmetrisch. Wir könnten versuchen OneT(d) für andere fixierte Städte auch zu berechnen, und die höchste solche untere Schranke nehmen. Mit der folgenden Methode erhält man jedoch eine bessere Schranke, und für diese Methode darf man die Stadt 1 in der Definition von OneT(d) ohne Beschränkung der Allgemeinheit festlegen.

Die Held-Karp Schranke

Wir verfeinern die untere Schranke OneT(d) wie folgt:

- wir führen ein Parameter π_i für jede Stadt $i \in N$ ein (π_i wird auch Strafe oder Preis für den „Besuch" dieser Stadt genannt)

Sei $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ der Vektor dieser Parameter.

- wir modifizieren alle Distanzwerte $d()$ und erhalten eine neue Metrik d^π (neue Distanzen)

$$d^\pi(i,j) = d(i,j) + \pi_i + \pi_j.$$

- Für die Metrik d^π gilt natürlich auch, dass ein kürzester 1-Baum nicht länger ist als eine kürzeste Rundreise

$$\mathrm{OneT}(d^\pi) \leq \mathrm{TSP}(d^\pi) \qquad (1)$$

- weiterhin gilt

$$\mathrm{TSP}(d^\pi) = \mathrm{TSP}(d) + 2\sum_{i=1}^{n} \pi_i. \qquad (2)$$

Warum? Diese Gleichung gilt für die Länge jeder konkreten <u>Rundreise</u> in den Metriken d bzw. d^π, weil jeder Knoten i mit genau zwei inzidenten Kanten teilnimmt und so genau $2\pi_i$ plus Länge verursacht in d^π im Vergleich zu d. Dies gilt <u>nicht</u> für <u>1-Bäume</u>! Da die Länge jeder <u>Rundreise</u> um den gleichen Wert $2\sum \pi_i$ verschoben ist, ist eine kürzeste Rundreise in d auch kürzeste in d^π, und für ihre Länge $\mathrm{TSP}(d)$ bzw. $\mathrm{TSP}(d^\pi)$ gilt die Gleichung (2).

Aus (1) und (2) folgt

$$\mathrm{OneT}(d^\pi) - 2\sum_{i=1}^{n} \pi_i \leq \mathrm{TSP}(d)$$

Für beliebige π, $OneT(d^\pi) - 2\sum_{i=1}^{n} \pi_i$ ist also eine untere Schranke

für $TSP(d)$, und (für fixierten $\pi = (\pi_1 \pi_2 \dots \pi_n)$)
es ist leicht zu berechnen, weil der minimale
1-Baum leicht zu berechnen ist, mit Länge $OneT(d^\pi)$.

Eine gute untere Schranke ist hoch, also nehmen
wir das Maximum dieser Schranken über alle
Vektoren π

$$\max_{\pi} \left[OneT(d^\pi) - 2\sum_{i=1}^{n} \pi_i \right]$$

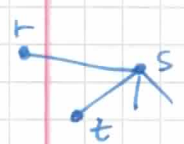
Diese Schranke heißt Held-Karp Schranke, und
ist effizient berechenbar. (Die Schwierigkeit dieser
Berechnung besteht in der Maximierung über alle π.)

Der Branching Operator für die Wurzel $(\emptyset, \emptyset)$:

Sei π^* der Vektor der die Held-Karp Schranke
(für $(\emptyset, \emptyset)$) bestimmt, und T^* ein minimaler
1-Baum für die Distanzen d^{π^*}.

→ Falls in T^* alle Knoten i Grad $= 2$ besitzen, dann
ist T^* eine minimale Rundreise, und wir sind fertig.

→ Sonst gibt es eine Stadt s mit Grad ≥ 3. Seien
$\{r, s\}$ und $\{t, s\}$ die längsten,
mit s inzidenten Kanten in T^*



[weil $w(\pi^*) \overset{\text{Länge}}{=} d^{\pi}(T^*) + 0 \leq TSP(d)$, siehe später]

Die Wurzel $v = (\emptyset, \emptyset)$ des B&B Baums hat dann 3 Kinder:

- in v_1 wird die Benutzung der Kante $\{r, s\}$ verboten
- in v_2 wird $\{r, s\}$ erzwungen aber $\{t, s\}$ verboten
- in v_3 werden $\{r, s\}$ und $\{t, s\}$ erzwungen, alle andere verboten

- das erfolgversprechendste Blatt wird stets mit Tiefensuche gewählt; (bessere Lösungen ergeben sich auch während der Tiefensuche)

- für ein aktiviertes Blatt $v = (S, T)$, um die Held-Karp Schranke zu berechnen, betrachtet man entsprechend 1-Bäume die alle Kanten aus S enthalten aber alle Kanten aus T verbieten.

Weitere Erklärung zur Held-Karp Schranke

Die Held-Karp Schranke ist $\max\limits_{\pi} w(\pi)$ für

$$\rightarrow\ w(\pi) = \mathrm{OneT}(d^{\pi}) - 2\sum_{i=1}^{n} \pi_i = \min_{T:\,1\text{-Baum}} \left[\text{Länge}_T(d^{\pi}) - 2\sum_{i=1}^{n} \pi_i \right] =$$

$$= \min_{T:\,1\text{-Baum}} \left[\text{Länge}_T(d) + \sum_{i=1}^{n} (\deg_i^T - 2)\,\pi_i \right]$$

wobei $\text{Länge}_T()$ ist die Länge des 1-Baums T in der gegebenen Metrik, und $\deg_i^T$ der Grad von Knoten (Stadt) i im 1-Baum T.

BB 17.

Beachte, dass falls T eine Rundreise ist $T = R$, dann ist der Ausdruck die Länge der Rundreise und unabhängig von Π.

$$\text{Länge}_R(d) + \sum_{i=1}^{n} (2-2) \cdot \Pi_i = \text{Länge}_R(d)$$

→ man kann Π mit einer Art „lokaler Suche" (Gradienten-Verfahren) Schritt für Schritt modifizieren um $w(\Pi)$ zu maximieren. Die Veränderung von Π beeinflusst den Wert für 1-Bäume die Rundreisen sind, nicht.

→ Für einzelne (!) 1-Bäume T, die keine Rundreisen sind, kann der Wert erhöht werden, indem Städte i mit $\deg_i^T \geq 3$ immer höhere Strafe Π_i zugeordnet wird.

→ Wenn durch die Änderung in Π ein anderer 1-Baum T' minimalen Wert hat, dann ändert man Π auch entsprechend gemäß diesem Baum T'

→ Dies hilft jedoch nicht weiter, wenn mit beliebiger Änderung von Π irgendein 1-Baum seinen Wert senken würde (oder eine Rundreise als 1-Baum mit minimalem Wert erreicht wurde → dann ergibt die Held-Karp Schranke die optimale Länge einer Rundreise

d.) Cutting Planes (Schnittebenen)

Untere Schranken für Minimierungsprobleme, bzw. obere Schranken für Maximierungsprobleme (gegeben in IP-Formulierung)

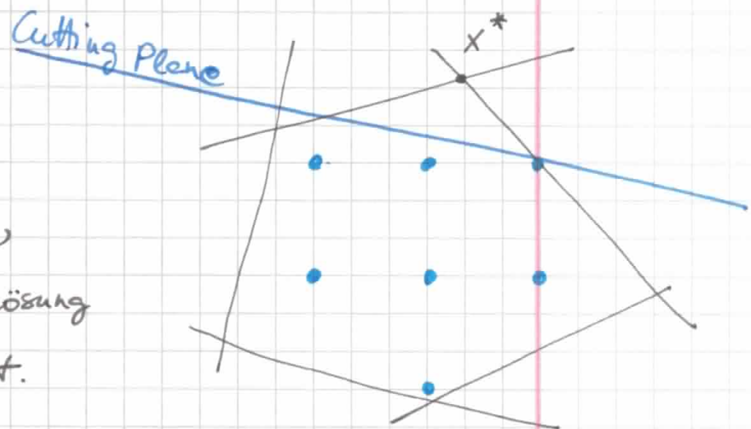
Grobstruktur

Aufgabe: Minimiere $c^T \cdot x$ so dass $A \cdot x = b$ | oder $Ax \geq b$

$x \geq 0$

x ganzzahlig

- löse die LP-Relaxierung, sei x^* eine optimale fraktionale Lösung
- falls x^* ganzzahlig, halt
- sonst füge eine neue Nebenbedingung (Ungleichung) ein, die von jeder ganzzahligen Lösung erfüllt wird, von x^* aber nicht.



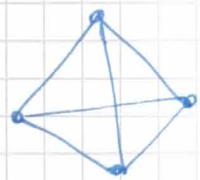
Diese Ungleichung heißt Schnittebene (Cutting Plane)
(sie bestimmt eine neue Hyperebene als Seitenfläche des Lösungspolyeders)

- (Schnittebenen werden ggf. so lange eingeführt, bis ein ganzzahliges Optimum berechnet wird.)

Cutting-Plane Methoden unterscheiden sich in der Wahl der Schnittebene, die x^* aus der Lösungsmenge ausschneidet.

Beispiel/ Illustration: max-INDEPENDENT SET auf dem vollständigen Graphen (ungewichtet)

Sei $V = \{1, 2, \ldots, n\}$



IP: maximiere $\sum_{i=1}^{n} x_i$

so dass $x_i + x_j \leq 1 \quad \forall i, j \; i \neq j$

$x_i \in \{0, 1\}$

- $x_i = \frac{1}{2} \; \forall i$ ist eine fraktionale Lösung mit Zielwert $\frac{n}{2}$
- die optimale ganzzahlige Lösung hat Wert 1

Wir werden die Schnittebenen

$$x_1 + x_2 \leq 1$$
$$x_1 + x_2 + x_3 \leq 1$$
$$x_1 + x_2 + x_3 + x_4 \leq 1$$
$$\vdots$$
$$x_1 + x_2 + x_3 + x_4 + \cdots + x_n \leq 1$$

Schnitt für Schnitt einfügen können, d.h. zeigen, mit Induktion, dass eine ganzzahlige Lösung sie erfüllen soll.

Die letzte $\sum_{i=1}^{n} x_i \leq 1$ hat eine ganzzahlige optimale Lösung $(1, 0, 0, \ldots, 0)$

Angenommen, die Ungleichung $\sum_{i=1}^{k-1} x_i \leq 1$ wurde eingefügt, wir zeigen, dass $\sum_{i=1}^{k} x_i \leq 1$ von einer ganzzahligen Lösung erfüllt werden soll:

Wir haben die Bedingungen

$$
\begin{array}{r|l}
1\cdot & x_1 \qquad\qquad + x_k \leq 1 \\
1\cdot & \quad x_2 \qquad\quad + x_k \leq 1 \\
 & \qquad \ddots \\
1\cdot & \qquad\qquad x_{k-1} + x_k \leq 1 \\
(k-2)\cdot & x_1 + x_2 + \dots + x_{k-1} \leq 1 \\
\hline
\end{array}
$$

Wir multiplizieren diese Ungleichungen durch $1, 1, 1 \dots$ bzw. $(k-2)$ und summieren sie.

wir erhalten:

$$(k-1)\cdot(x_1 + x_2 + \dots + x_{k-1} + x_k) \leq k-1 + k-2$$

$$x_1 + x_2 + \dots + x_{k-1} + x_k \leq 1 + \frac{k-2}{k-1}$$

Für ganzzahlige Lösungen soll also gelten

$$x_1 + x_2 + \dots + x_{k-1} + x_k \leq 1 \qquad \square$$

Das ist ein Spezialfall von Gomory-Chvátal Schnitten

Solving hard problems in practice

(aus: The Nature of Computation, Sec. 9.10)

TSP-Instanz: 42 Großstädte der USA
$N = \{1, 2, 3, \ldots, 42\}$

Wir versuchen obere bzw. untere Schranken für die Länge einer kürzesten Rundreise nah aneinander bringen

obere Schranke: die Länge einer bislang besten gefundenen Rundreise

untere Schranke: der optimale Wert einer relaxierten, effizient berechenbaren Version des TSP Problems

1. erste approximative Lösungen

→ man findet eine Rundreise mit der Greedy Nearest-Neighbor Heuristik, wobei in jedem Schritt die nächstmögliche noch unbesuchte Stadt besucht wird

→ man verbessert diese Tour mit lokaler Suche:

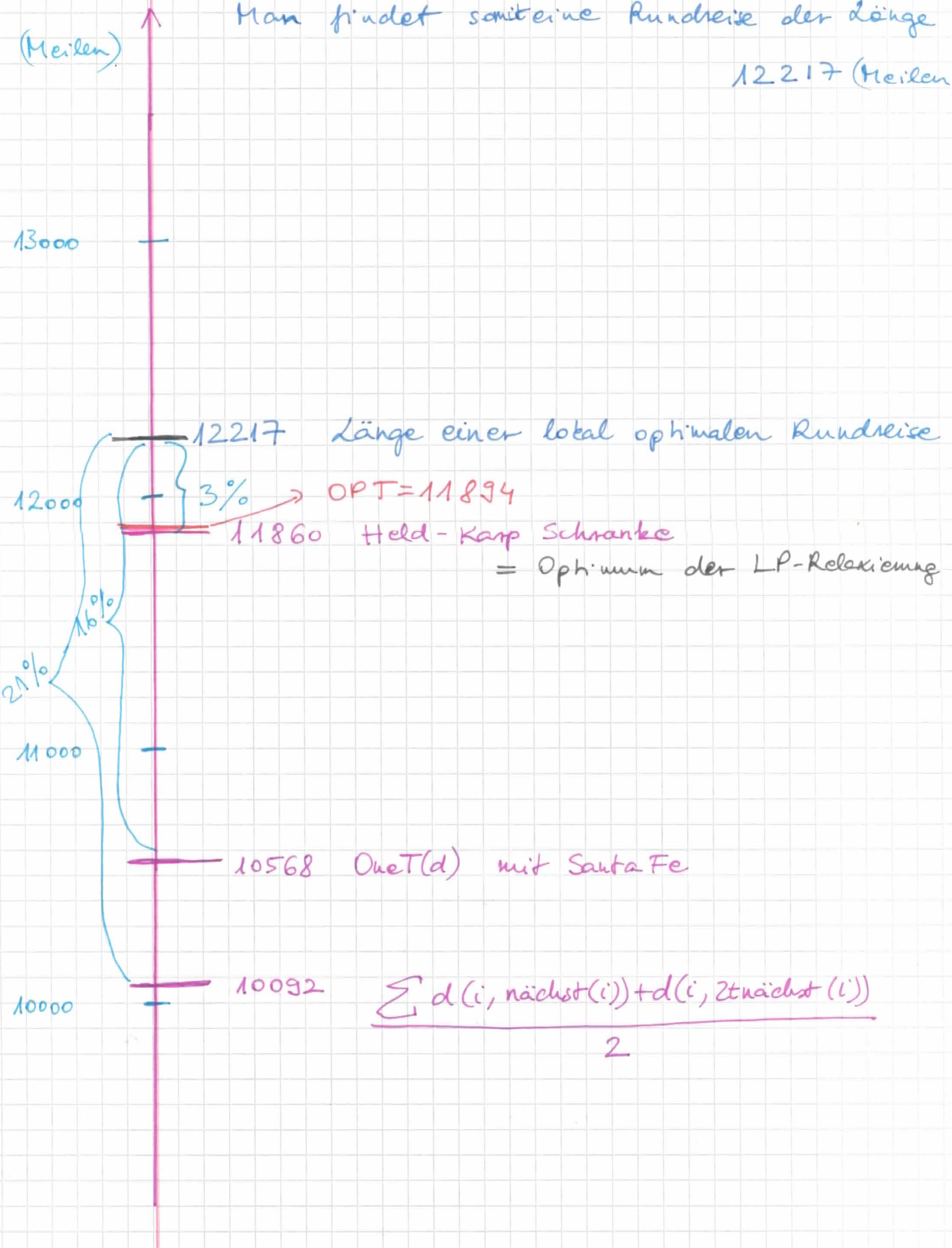
Vertex-Insertion ändert die Position einer Stadt in der Rundreise; es gibt n Positionen und $\approx n \cdot n = n^2$ mögliche Änderungen (so viele benachbarte Rundreisen zu einer gegebenen Rundreise)
jeweils die beste Nachbarlösung wird gewählt

2-opt nimmt 2 Kanten heraus, und fügt sie andersrum in die beiden Teilrundreisen ein.
Wieder gibt es jeweils $\approx n^2$ Nachbarlösungen.

BB 22.

Diese Schnitte werden iteriert, bis die Tour <u>lokal</u> optimal wird, d.h. alle benachbarten Touren sind länger.

Man findet somit eine Rundreise der Länge 12217 (Meilen)



2. Untere Schranken

→ Sei $\sigma_1 \sigma_2 \dots \sigma_n$ eine Permutation entsprechend einer optimalen Rundreise $\sum_{i=1}^{n} d(\sigma_i, \sigma_{i+1}) = OPT$

$$\sum_{i=1}^{n} \left(d(\sigma_i, \sigma_{i-1}) + d(\sigma_i, \sigma_{i+1}) \right) = 2\,OPT$$

$$\sum_{i=1}^{n} \text{Distanz von } i \text{ zu den beiden nächsten Nachbarn} \leq 2\,OPT$$

→ diese untere Schranke ergibt: 10092 Meilen

→ 1-Baum (mit Stadt 21 Santa Fe als Knoten 1)

minimale Länge: 10568

→ Held-Karp Schranke: 11860

3. LP-Relaxierung

Man möchte eine LP-Relaxierung des TSP lösen, um noch bessere untere Schranke zu erhalten

TSP hat (unter anderen) die folgende IP-Formulierung:

Sei x_e die Variable für Kante e im vollständigen Graphen über alle Städte in N, und d_e ihre Länge

$$\text{minimiere} \quad \sum_{e} d_e \cdot x_e$$

$$x_e \in \{0,1\}$$

$$\text{so dass} \quad \sum_{e \in \delta(i)} x_e = 2 \quad \text{für jede Stadt } i \in N \qquad (IP)$$

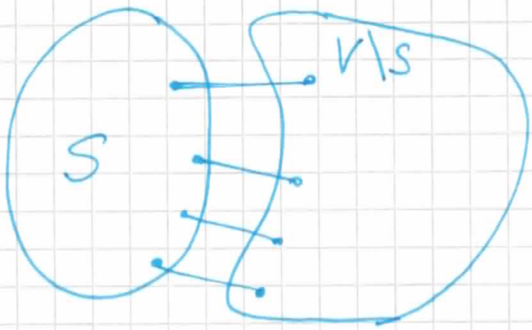
Schnitt-Bedingungen (Ungleichung) (cut-constraints) $\sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset \{1,2,3,\dots,42\}$

$\delta(i)$: Menge der mit i inzidenten Kanten

$\delta(S)$: Menge der kreuzenden Kanten in $(S, V \setminus S)$

BB24.

eine Rundreise ist <u>zweifach zusammenhängend</u>:



Für jeden Schnitt $(S, V\setminus S)$ sollen mindestens 2 kreuzende Kanten in der Rundreise sein

(LP) Relaxierung: $x_e \in [0,1]$ statt $x_e \in \{0,1\}$

+ sonst dieselben Bedingungen

[Beachte, dass hier „Schnitt" in einem anderen Sinn benutzt wird als bei den Schnittebenen, nämlich als Zweiteilung eines Graphen!]

Die Lösungsmenge dieses LP ist das sog. Cut-Polytop

<u>Aber!</u> dieses LP hat exponentiell-viele Bedingungen!

somit ist es nicht in polynomieller Zeit optimierbar mit LP-Solver (also nicht mal die LP-relaxierung!)

Wir machen etwas ähnliches wie bei der Cutting-Plane Methode:

→ das LP ohne Schnitt-Bedingungen wird gelöst:

$$\text{minimiere} \quad \sum_e d_e x_e$$

$$\text{s.d.} \quad \sum_{e \in \delta(i)} x_e = 2 \quad \forall i \in N$$

$$0 \leq x_e \leq 1$$

→ eine optimale Lösung (x_e^*) entspricht einem Graphen mit Kantengewichten x_e^*

→ ein minimaler Schnitt in diesem Graphen kann effizient berechnet werden (siehe Max-Flow-Min-Cut Theorem)

→ FALLS MinCut ≥ 2 ⇒ die Schnitt Bedingungen werden erfüllt (x_e^*) ist eine fraktionale Rundreise

→ SONST füge die Bedingung (Cut-Constraint o. Schnitt-Ungleichung als Schnittebene (!))

$$\sum_{e \in \delta(S_0)} x_e \geq 2$$

für den minimalen Schnitt $(S_0, V \setminus S_0)$ zu den LP-Bedingungen

→ in unserem Beispiel erhält man nach dem Einfügen von 6 Schnitt-Bedingungen eine minimale fraktionale Lösung von TSP (d.h. die alle Bedingungen von (LP) erfüllt)

ihre Länge ist 11860 – genau die Held-Karp-Schranke!

Warum?

Die Held-Karp Schranke (mit den Variablen $\pi_1\ \pi_2 \ldots \pi_n$)

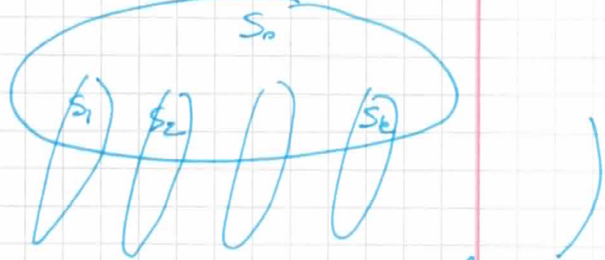
ist das duale der LP-Relaxierung!

(4.) Ende I: man sollte die untere oder die obere Schranke weiter verbessern.

Es gibt weitere lineare Bedingungen, die eine ganzzahlige Lösung erfüllen soll.

(Kamm-Ungleichungen

die Summe der Schnitte $k+1$ solcher Mengen soll $\geq 3k+1$ sein



Nach dem Einfügen von 2 Kamm-Ungleichungen (als Cutting Planes) wird die optimale Lösung des LP (in unserem Beispiel) eine ganzzahlige Rundreise also eine optimale Rundreise

5. Ende II.

Eine optimale Lösung mit nur Cutting planes braucht zu viel Kreativität oder ist sogar nicht möglich zu finden im Allgemeinen

→ Man kann die Cutting-Plane Methode mit Branch & Bound kombinieren

→ Die optimale Lösung vom (LP) enthält fraktionale Kanten mit $x_e = \frac{1}{2}$

Man fixiert eine dieser fraktionalen Kanten, und prüft separat zwei Möglichkeiten: $x_e = 1$ oder $x_e = 0$

das ist ein <u>Branching-Operator</u>

In beiden Fällen (Zweigen) hat man eine eingeschränkte TSP-Instanz, und kann für diese die LP-Relaxierung optimieren. (ohne Schnitt-Ungleichungen, oder mit manchen von ihnen)

In der konkreten Instanz erhält man die folgenden Ergebnisse:

→ Fall 1. $x_e = 1$ (für die konkret ausgewählte Kante)

ergibt die Optimierung des LP eine ganzzahlige Rundreise der Länge: 11 894

→ Fall 2. $x_e = 0$

das LP ergibt eine optimale Lösung, (die keine Rundreise ist), mit Wert 11 928

→ da jede weitere Bedingung, bzw. Branching-Operator die Lösungsmenge einschränkt, wird das Optimum in diesem Zweig mindestens 11 928

diese ist eine untere Schranke

⇒ diesen Zweig (Lösungen mit $x_e = 0$) braucht man nicht weiter zu prüfen.

⇒ In der Praxis funktioniert Branch & Bound oft, selbst wenn es im Worst-Case kein Polynomialzeit Algorithmus ist. Gute untere Schranken zu finden ist wesentlich!

→ Die Methode wird Branch and Cut genannt, falls gute untere Schranken in den Zweigen (Kinder-Knoten) mit Hilfe von Cutting Planes berechnet werden (d.h. für die beste ganzzahlige Lösung in diesem Teilbaum)

(TSP wurde mit Branch & Cut gelöst für alle 24978 Städte in Schweden, und für alle 15112 Städte in Deutschland.

Für 1904711 Orte der Welt hat man mit Branch and Cut eine Lösung mit höchstens 0,05 % Fehler)