UNIVERSITÄT
LEIPZIG

Algorithmen und Datenstrukturen II

Vorlesung t10

Leipzig, 04.06.2024

Peter F. Stadler & Thomas Gatter & Ronny Lorenz

RUCKSACK UND GREEDY

Ganzzahlige und relaxierte Probleme

- Wichtige Unterscheidung zwischen ganzzahligen und reellzahligen *(“kontinuierlichen”)* Optimierungsproblemen.
- Einschränkung auf ganze Zahlen macht viele Probleme qualitativ schwerer lösbar (*Beispiel hier:* 0/1-Rucksack).
- Das einem ursprünglich ganzzahligen Problem zugeordnete kontinuierliche Problem heißt *Relaxierung* oder *relaxiertes Problem*.
- Relaxierung wird oft betrachet, weil sie eine untere Schranke an die Werte der Kostenfunktion bzw. eine obere Schranke an die Gewinnfunktion liefert.

Zur Erinnerung 0/1-Rucksackproblem I

- Objekte $\{1, 2, \ldots, n\}$ *Zur Erinnerung: Folien VL 09, ab Slide 7*
- mit Volumina $t_1, t_2, \ldots, t_n$ und Werten $p_1, p_2, \ldots, p_n$
- Rucksack hat Gesamtvolumen c.

Optimierungsproblem: *finde einen* $0/1$*-Vektor* $a_1, ..., a_n \in \{0, 1\}$ *mit*

$$\sum_{i=1}^{n} a_i t_i \leq c \quad \textit{sodass} \quad f(a) = \sum_{i=1}^{n} a_i p_i \to \max$$

Zur Erinnerung 0/1-Rucksackproblem II

Beispiel

i	1	2	3	4	5
p_i	8	7	2	3	8
t_i	3	4	1	2	5

$c = 10$

Optimale Lösung: $f = 20$ ***mit*** $\{1, 2, 3, 4\}$

Greedy und das 0/1-Rucksackproblem

– Kanonischer Greedy für 0/1-Rucksack erreicht Wert 19,
Objekte $\{1, 4, 5\}$, also $x_1 = x_4 = x_5 = 1$, $x_2 = x_3 = 0$.

***Warum** war das nochmal so?*

– Mengensysteme erlaubter Objektkombinationen sind i.A. *keine Matroide*. Das sieht man z.B. daran, daß nicht-erweiterbare Objektmengen nicht die gleiche Kardinalität haben.
(siehe Lösung auf der vorigen Seite und diese hier)

– Daher liefert der kanonische Greedy-Algorithmus nicht immer die optimale Lösung.

Fraktionales Rucksackproblem I

Fraktionales Rucksackproblem

Eigentlich wie das **0/1-Rucksackproblem**, aber $x_i \in [0, 1]$.

→ *Beliebige Bruchteile eines Objekts* dürfen eingepackt werden.

Fraktionales Rucksackproblem II

Greedy-Ansatz zum Finden der optimalen Lösung

1. Berechne von jedem Objekt i den *Nutzen*, also das Verhältnis aus Gewinn und Volumen $p_i/t_i =: u_i$.
2. Sortiere Objekte absteigend nach Nutzen (spezifischer Gewinn): $u_1 \geq u_2 \geq \cdots \geq u_n$.
3. Finde maximales i, so dass die Objekte $\{1, \ldots, i\}$ vollständig in den Rucksack passen, $\sum_{j=1}^{i} t_j \leq c$.
4. Setze $x_j = 1$ für $j \leq i$, $x_j = 0$ sonst.
5. Falls $i \neq n$, setze

$$x_{i+1} = \left(c - \sum_{j=1}^{i} t_j \right) / t_{i+1}$$

Beispiel

i	1	2	3	4	5
p_i	8	7	2	3	8
t_i	3	4	1	2	5
u_i	$\frac{8}{3}$	$\frac{7}{4}$	2	$\frac{3}{2}$	$\frac{8}{5}$

$c = 10$

$u_1 > u_3 > u_2 > u_5 > u_4$

$t_1 = 3 \leq 10$ ✓ $\quad t_1 + t_3 = 4 \leq 10$ ✓ $\quad t_1 + t_3 + t_2 = 8 \leq 10$ ✓

$t_1 + t_3 + t_2 + t_5 = 13 > 10$ ↯

$t_1 + t_3 + t_2 + \frac{2}{5} t_5 = 10.$

– Greedy für fraktionalen Rucksack erreicht Wert 20.2
  $x_1 = x_2 = x_3 = 1$, $x_5 = \frac{2}{5}$, $x_4 = 0$.

BRANCH AND BOUND

Branch and Bound

Bisher behandelte Strategie für Optimierungsprobleme:

- **Greedy-Verfahren**

Was tun, wenn dieses nicht anwendbar ist?

- Naiver Ansatz (“Brute Force”): vollständige Aufzählung aller möglichen Lösungen.
- **Branch and Bound**: Identifiziere möglichst große Teilmengen des Lösungsraums, die keine optimale Lösung enthalten können, und überspringe diese beim Aufzählen.

Beispiel für Branch and Bound

Aufzählen von Lösungen

$x_1 \dots x_5$	Ges.Wert	Ges.Vol.
00000	0	0
10000	8	3
01000	7	4
11000	15	7
001**	≤ 13	
...		

0/1 - Rucksack

$c = 10$

i	1	2	3	4	5
p_i	8	7	2	3	8
t_i	3	4	1	2	5

Keine Lösung x mit $x_1 = x_2 = 0$ und $x_3 = 1$ (egal ob sie die Gewichtbeschränkung erfüllt oder nicht) kann wertvoller als $8+3+2 = 13$ sein. Diese Lösungen brauchen daher nicht aufgezählt werden, denn sie sind alle schlechter als der bisherige Maximalwert 15.

Grundidee (Minimierungsproblem)

1. **Strukturiere den Lösungsraum so, dass er einen Baum darstellt.**
   (Die Wurzel entspricht der leeren Lösung, die zu jeder beliebigen Lösung erweitert werden kann.)
2. **Sobald ein neuer Knoten erzeugt wurde, berechne für diesen Teilbaum die untere Schranke *b* (bound).**
   (Keine Lösung, die sich oberhalb des betreffenden Knotens befindet, kann einen besseren Wert als *b* haben)
3. **Vergleiche die bounds aller bisherigen Baumblätter und expandiere dasjenige mit dem kleinsten bound.**
   (Wenn ein Ast so weit entwickelt worden ist, dass das entsprechende Blatt eine vollständige und im Vergleich mit allen anderen bounds minimale Lösung darstellt, dann ist diese Lösung optimal.)

Optimierungsproblem und Schranke

Gegeben:

- Menge erlaubter Lösungen X
- Kostenfunktion $f : X \to \mathbb{R}$

Gesucht:

- $y \in X$ so dass $f(y) \leq f(x)$ für alle $x \in X$
  y = globales *Minimum* der Kostenfunktion.

Definition

Sei $g : \mathcal{P}(X) \to \mathbb{R}$. Die Funktion g heißt *untere Schranke* von f, wenn für alle $A \subseteq X$ gilt:

$$g(A) \leq \min_{x \in A} f(x) \quad \text{(Abschätzung für Mengen von Lösungen)}$$

und für alle $x \in X$ gilt:

$$g(\{x\}) = f(x) \quad \text{(Exakt für einzelne Elemente)}$$

Split-Operation I

Wir betrachten Teilmengen $A \subseteq X$ in Form von Masken $m \in (0, 1, *)^n$ (n entspricht der Dimension von Lösungen in X), die sich wie folgt beschreiben lassen:

$$A = \{x \in X : \forall i : m_i \neq * \Rightarrow x_i = m_i\}$$

Die Maske schreibt also für eine Position i den Wert von x_i vor, wenn $m_i = 0$ oder $m_i = 1$. Ansonsten ist der Wert dort beliebig ($m_i = *$).

Split(A)=(B, C) mit $B = \{x \in A : x_k = 0\}$, $C = \{x \in A : x_k = 1\}$
$k = \min\{i : \exists x, y \in A : x_i \neq y_i\}$

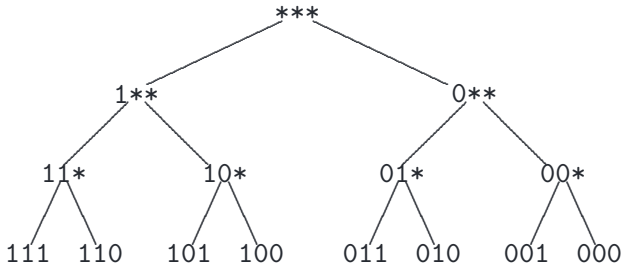
Split-Operation II

Beispiel

Split(`010***`) = (`0100**`, `0101**`)

Achtung: Momentan haben wir eine 0/1 Maske. Prinzipiell sind auch nicht-binäre Masken möglich mit mehr als zwei Split-Möglichkeiten.

Verzweigung für $n = 3$



Branch and Bound

Grundalgorithmus I

$b \leftarrow +\infty$, INIT(S), PUSH(S,X) ;
while ***not*** *EMPTY(S)* **do**
  A=POP(S);
  if $g(A) < b$ **then**
    if $|A| == 1$ **then**
      $b \leftarrow g(A)$
    else
      (B, C)=Split(A);
      if $B \neq \emptyset$ **then** PUSH(S,B) **end**
      if $C \neq \emptyset$ **then** PUSH(S,C) **end**

Anwendung auf das Rucksack-Problem

Da Branch and Bound *minimiert* (per Konvention), verwenden wir die Kostenfunktion mit umgedrehtem Vorzeichen:

$$f(x) = -\sum_{i=1}^{n} x_i p_i$$

Um Branch and Bound anwenden zu können, müssen wir festlegen:

1. Die Funktion Split (Mengenaufteilung)
2. untere Schranke g auf Teilmengen von X.
   Zur Erinnerung: $g(A) = f(\{x\})$ wenn $A = \{x\}$ aus nur einem Element besteht

Anwendung auf das Rucksack-Problem II

Die Funktion Split kann mit minimaler Änderung übernommen werden:

– Wird das Gesamtvolumen in C überschritten, erzeugt Split(A)=(B, $\emptyset$).

Berechnung von g ist ein Kompromiss aus zwei Forderungen:

– Untere Schranke $g(A)$ soll möglichst nah am wahren Kostenminimum auf $A \subseteq X$ liegen

– Zeitaufwand soll möglichst gering sein, insbesondere deutlich geringer als die Berechnung durch Aufzählen von A.

Bestimmung einer unteren Schranke g I

Schranke für Teilmenge mit Maske $m = (m_1, m_2, \ldots, m_n)$:

- Falls A genau ein Element x enthält [1], sei die Schranke $g(\{x\}) = f(x)$.
- Sonst:
  1. Berechne Kosten γ und verbrauchtes Volumen v auf der festgelegten Objektmenge, also auf $I = \{i : m_i \neq *\}$.

$$\gamma = -\sum_{i \in I} m_i p_i \ , \qquad v = \sum_{i \in I} m_i t_i$$

  2. Finde mit Greedy die minimalen Kosten β für das fraktionale Rucksackproblem auf der verbleibenden, nichtfestgelegten Objektmenge für Kapazität $c' = c - v$.

Dann ist $\gamma + \beta$ untere Schranke an Kosten auf der Teilmenge A.

[1] Die Maske kann dennoch $m_i = *$ enthalten. Mehr dazu auf der nächsten Folie.

Bestimmung einer unteren Schranke g II

Wir können bestimmte Lösungen ausschließen und stoppen deren Enumerierung direkt, auch wenn noch *-Elemente $S = \{i : m_i = *\}$ in der Maske für ein A verbleiben. Wir behandeln eine Lösungsmenge A als ob sie nur genau ein Element x enthällt, falls:

- kein *-Objekt mehr vollständig angefügt werden kann, also $t_i > c' \; \forall i \in S$
- alle *-Objekt angefügt werden können, also $\sum_{i \in I} t_i \leq c'$

Beide Fälle lassen sich einfach bei der Berechnung von g testen.

Branch and Bound für 0/1-Rucksack

Beispiel

b	g(TOP(S))	Stack S
$+\infty$	-20.2	*****
$+\infty$	-20.2	1****, 0****
$+\infty$	-20.2	11***, 10***, 0****
$+\infty$	-20.2	111**, 110**, 10***, 0****
-20.0	-20.0	1111*, 1110*, 110**, 10***, 0****
-20.0	-17.0	1110*, 110**, 10***, 0****
-20.0	-19.5	110**, 10***, 0****
-20.0	-19.8	10***, 0****
-20.0	-17.0	0****

Resultat: $\min_{x \in X} f(x) = b_{\text{final}} = -20.0$

$b \leftarrow +\infty$, INIT(S), PUSH(S,X) ;
while ***not*** *EMPTY(S)* **do**
  A=POP(S);
  if $g(A) < b$ **then**
    if $|A| = 1$ **then**
      $b \leftarrow g(A)$
    else
      (B, C)=Split(A);
      if $B \neq \emptyset$ **then**
        PUSH(S,B)
      if $C \neq \emptyset$ **then**
        PUSH(S,C)

Branch and Bound für 0/1-Rucksack

Beispiel

b	g(TOP(S))	Stack S
$+\infty$	-20.2	*****
$+\infty$	-20.2	1****, 0****
$+\infty$	-20.2	11***, 10***, 0****
$+\infty$	-20.2	111**, 110**, 10***, 0****
-20.0	-20.0	1111*, 1110*, 110**, 10***, 0****
-20.0	-17.0	1110*, 110**, 10***, 0****
-20.0	-19.5	110**, 10***, 0****
-20.0	-19.8	10***, 0****
-20.0	-17.0	0****

c=10

i	1	2	3	4	5
p_i	8	7	2	3	8
t_i	3	4	1	2	5
u_i	$\frac{8}{3}$	$\frac{7}{4}$	2	$\frac{3}{2}$	$\frac{8}{5}$

Resultat: $\min_{x \in X} f(x) = b_{\text{final}} = -20.0$

Traveling Salesman Problem I

TSP

- **Gegeben:** n Städte, paarweise Distanzen:
  d_{ij} Entfernung von Stadt i nach Stadt j.
- **Gesucht:** Rundreise mit minimaler Länge durch alle Städte, also Permutation $\pi : (1, ..., n) \to (1, ..., n)$, für die $c(\pi)$ minimal wird.

$$c(\pi) = \left[\sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} \right] + d_{\pi(n)\pi(1)}$$

- NP-hart (d.h. exponentiell, ausser P=NP)
- **Naiver Algorithmus:** Alle $(n-1)!$ Reihenfolgen betrachten.

Branch and Bound für TSP

- **Operiere auf Kantenmengen** (statt auf Permutationen der Städte).
  Zulässige Kantenmengen X: maximal je eine eingehende und eine ausgehende Kante pro Stadt, keine Zyklen kürzer als n (Anzahl Städte).
- **Split-Operation für Kantenmengen *fast* wie für Objektmengen bei Rucksack:**
  Für jede teilweise spezifizierte Tour wähle eine der verbleibenden möglichen Kanten. Der Verzweigungsbaum ist also *nicht* binär wie beim Rucksack sondern hat $(n - 1)$ Kinder an der Wurzel, von denen jedes $(n - 2)$ Kinder hat, die wiederum $(n - 3)$ Kinder haben usw.
- **Untere Schranke:** Kosten **aller** bisher gewählten Kanten + Kosten der billigsten (eingehenden und ausgehenden) Kanten der unverbundenen Städte.

Branch and Bound

Grundalgorithmus II

$b \leftarrow +\infty$;
S = {X};
while ***not*** *EMPTY(S)* **do**
  $A = \arg\min_X \{g(X) | X \in S\}$;
  if $g(A) < b$ **then**
    if $|A| == 1$ **then**
      $b \leftarrow g(A)$
    else
      $(X_1, \ldots, X_n)$=Split(A);
      $S = S \cup \{X_1, \ldots, X_n\}$;

Branch and Bound für TSP - zunächst symmetrisch

Abschätzung der Minimalen Distanz der noch unverbundenen Städte enspricht der Summe der **Zeilen**- bzw. Spalten-**Minima**, für das Beispiel hier:

$$5 + 4 + 6 + 4 + 7 = 26$$

	1	2	3	4	5
1	-	5	13	28	17
2	5	-	9	4	14
3	13	9	-	6	7
4	28	4	6	-	11
5	17	14	7	11	-

Kosten für Verzweigung (bound) sind Kosten für den bisher gewählten Weg plus minimale Kosten für die Summe der Zeilen-/Spalten-Minima in der Distanzmatrix ohne diesen Weg.

Teilmatrizen nach 1. Kantenwahl I

- Das TSP schliesst einen Kreis über alle Knoten.
  $\rightarrow$ daraus folgt, dass der Startknoten beliebig gewählt werden kann
- Da alle Wege symmetrisch sind ($x_{ij} = x_{ji}$), können die Kanten zu allen Zielen ungleich des Startpunkts gewählt werden.
- Für das Beispiel haben 4 Wahlmöglichkeiten für die 1.Kante mit Startknoten 1.
  Wahlmöglichkeiten: $\{1,2\}, \{1,3\}, \{1,4\}, \{1,5\}$
- Im folgenden sind die 4 Wahlmöglichkeiten angegeben. Zusätzlich zu jeder Wahl die Restmatrix mit den noch zu wählenden Kanten.

Teilmatrizen nach 1. Kantenwahl II

- **Achtung**: die Zeilen- und Spaltenannotationen sind nun nicht mehr symmetrisch. Die Wahl von *i* und *j*, zum Beispiel $x_{1,2}$, also $\{1, 2\}$ als 1. Kante, ändert die Möglichkeiten zur Fortsetzung des Rundwegens. Im symmetrischen Beispiel wählen wir *i* als Startknoten, *j* als Endknoten. Damit ist es bei $x_{1,2}$ nicht mehr möglich *von* Knoten 1 aus eine Kante zu wählen. Ebenso kann 2 kein Endknoten einer weiteren Kante werden.
  $\rightarrow$ Dies wird durch die Entfernung der Zeile “i” und Spalte “j” erreicht.

Beispiel symmetrisch 1

$x_{1,2} = 5$	1	3	4	5
2	-	9	**4**	14
3	13	-	**6**	7
4	28	**6**	-	11
5	17	**7**	11	-

Summe der Zeilen-Minima

$4 + 6 + 6 + 7 = 23$

+ Kosten von $x_{1,2} = 5$ macht 28

$x_{1,3} = 13$	1	2	4	5
2	5	-	**4**	14
3	-	9	**6**	7
4	28	**4**	-	11
5	17	14	**11**	-

Summe der Zeilen-Minima

$4 + 6 + 4 + 11 = 25$

+ Kosten von $x_{1,3} = 13$ macht 38

Beispiel symmetrisch 1

$x_{1,4} = 28$	1	2	3	5
2	**5**	-	9	14
3	13	9	-	**7**
4	-	**4**	6	11
5	17	14	**7**	-

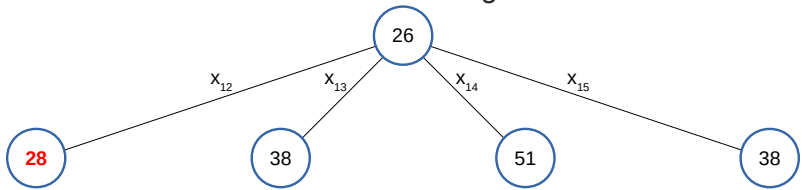
Summe der Zeilen-Minima
$5 + 7 + 4 + 7 = 23$
+ Kosten von $x_{1,4} = 28$ macht 51

$x_{1,5} = 17$	1	2	3	4
2	5	-	9	**4**
3	13	9	-	**6**
4	28	**4**	6	-
5	-	14	**7**	11

Summe der Zeilen-Minima
$4 + 6 + 4 + 7 = 21$
+ Kosten von $x_{1,5} = 17$ macht 38

Baumstruktur für 1. Schritt

Die folgende Baumstruktur visualisiert die 4 möglichen Schritte mit Startstadt 1.



- **Zur Erinnerung:** die Wahl der Startstadt ist frei, und “1” wurde o.b.d.A. gewählt.
- Die Berechnung der Summe der Minima über den noch verbleibenden Wahlmöglichkeiten liefert *nur* eine untere Schranke, die tatsächlichen TSP-Kosten liegen möglicherweise höher. Die Minima liefern *nicht* unbedingt eine Tour!
- **Achtung:** Das momentane Minimum von 28 allein reicht *nicht* um alle anderen Teilbäume ausschliessen zu können. Nur eine vollständige Tour ($|A| == 1$) kann einen Bound geben.

Heuristische Wahl der Kante

- Wir setzen die Suche immer am kleinsten Blattknoten fort, also hier mit $x_{1,2}$.
- Diese Festlegung ist eine ***Heuristik***.
  Allerdings eine Heuristik welche nur der Wahl der Enumeration, keine Heuristik in Bezug auf die Tour mit minimalen Kosten.
- Wir verlieren hier nichts, sondern hoffen nur, dass die lokal beste Wahl uns frühzeitig zu guten Touren führt mit deren Hilfe wir noch nicht komplett evaluierte Teilbäume entfernen können.
- Wir dürfen dabei nur Kanten wählen, die nicht zu bereits besuchten Städten führen!
- Wir wählen also greedy (lokal) die günstigste Kante und wiederholen nun den Algorithmus mit der Wahl der 2. Kante.

Beispiel: Wahl der 2. Kante

$x_{2,3} = 9$	1	4	5
3	13	**6**	7
4	28	-	**11**
5	17	**11**	-

Summe der Zeilen-Minima
$6 + 11 + 11 = 28$
+ Kosten von $x_{2,3} = 9$
+ Kosten von $x_{1,2} = 5$ macht 42

$x_{2,4} = 4$	1	3	5
3	13	-	**7**
4	28	**6**	11
5	17	**7**	-

Summe der Zeilen-Minima
$7 + 6 + 7 = 20$
+ Kosten von $x_{2,4} = 4$
+ Kosten von $x_{1,2} = 5$ macht 29

Beispiel: Wahl der 2. Kante

$x_{2,5} = 14$	1	3	4
3	13	-	**6**
4	28	**6**	-
5	17	**7**	11

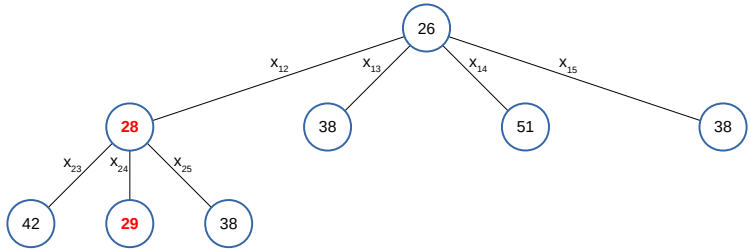
Summe der Zeilen-Minima

$6 + 6 + 7 = 19$

+ Kosten von $x_{2,5} = 14$

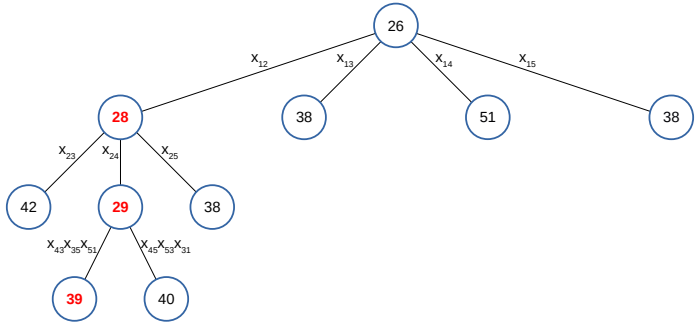
+ Kosten von $x_{1,2} = 5$ macht 38

Auswahl der 2. Kante



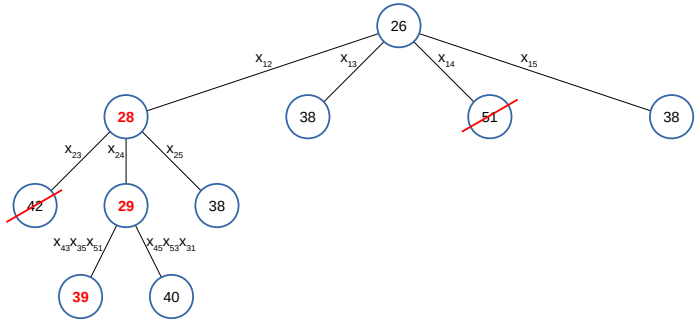
- Heuristisch / Greedy ist die Wahl $x_{2,4}$ lokal optimal.
- Der bisherige Teil der Tour ist damit $(1, 2, 4)$, zwei Knoten fehlen noch: $\{3, 5\}$.
- Wir zeichnen nun wieder den Branch-and-Bound Baum nach dieser Wahl.
- In jedem inneren Knoten stehen die Kosten der Kantenwahl (z.B $x_{1,4} + x_{2,4}$) plus der minimalen restlichen Kosten nach Heuristik!

Im folgenden Schritt passiert etwas mehr!



- Zuerst wird die nächste Kante $x_{4,3}$, oder alternativ $x_{4,5}$ gewählt.
- Damit existiert allerdings nur noch jeweils ein freier Knoten: 5, bzw. 3!
- Damit ist auch die letzte freie Kante klar bestimmt. Es muss $x_{3,5}$, bzw. $x_{5,3}$ folgen.
- Hinzu kommt noch die Kante, die die Rundtour abschließt, also $x_{5,1}$ bzw. $x_{3,1}$.
- Wir erhalten also 2 vollständige Touren mit Tourkosten 39 bzw. 40.

Einschränkung des Suchraums:



- **Tree Pruning:** Alle Teilbäume deren heuristische minimale Kosten $>$ 39 sind brauchen nun nicht mehr angesehen werden.
- Wähle nun einen Blattknoten mit minimalen Kosten noch nicht vollständig evaluiert ist und nicht durch einen Bound verboten ist.
- Sobald es keine solche Wahl mehr gibt muss eines der evaluierten Blätter die optimalen Kosten zur optimalen Tour enthalten.

Branch and Bound für TSP - asymmetrisch

Beispiel für untere Schranke bei TSP

	1	2	3	4	5
1	-	5	13	8	17
2	7	-	9	4	14
3	12	10	-	6	7
4	8	4	9	-	11
5	15	14	8	12	-

Betrachte Teilmenge von Lösungen, die Kanten $(1,2)$ und $(2,3)$ enthalten.

- *Kosten der vorhandenen Kanten:*
  $\gamma = d_{12} + d_{23}$
- *minimale Kosten für ausgehende Kanten:*
  $\beta_{\text{aus}} = d_{34} + d_{41} + d_{54}$
- *minimale Kosten für eingehende Kanten:*
  $\beta_{\text{ein}} = d_{41} + d_{34} + d_{35}$
- *Wert der unteren Schranke*
  $\gamma + \max\{\beta_{\text{aus}}, \beta_{\text{ein}}\}$

Beispiel asymmetrisch

	1	2	**3**	**4**	**5**
1	-	5	13	8	17
2	7	-	9	4	14
3	12	10	-	6	7
4	8	4	9	-	11
5	15	14	8	12	-

Relexation im Allgemeinen I

Relaxation des Optimierungsproblems ist ein recht allgemeines Prinzip zur Konstruktion der Schrankenfunktion g. Der Übergang von diskreten zu kontinuierlichen Variablen ist nur eine von mehreren Möglichkeiten.

Allgemeine Idee: Erweiterung des Suchraums X auf eine grössere Menge Y und Erweiterung der Kostenfunktion f zu $\tilde{f}$, die dann auf ganz Y definiert ist. Das Optimierungsproblem $(Y, \tilde{f})$ soll dann einfach zu lösen sein.

Formal:

- $X \subset Y$
- $f(x) = \tilde{f}(x)$ für all $x \in X$.

Relexation im Allgemeinen II

Das Beispiel TSP in diesem Licht: $y \in Y_1$ sei eine Menge von Paaren (i, j), sodass jede Stadt genau einmal Endpunkt einer Kante ist. Analog: Y_2 besteht aus den Kantenmengen für die jede Stadt genau einmal Ausgangspunkt ist. In beiden Fällen sei $\tilde{f}(Y) = \sum_{(i,j)\in Y} d_{ij}$. Minimierung von $\tilde{f}$ ist trivial: Wähle die kürzeste einlaufenden (Y_1) bzw. auslaufende (Y_2) Kante an jedem Knoten.

Alle TSP Touren sind erlaubte Mengen vom Type Y_1 and Y_2: Wenn die Kantenmenge y so gewählt wird, dass sie einen Hamiltonschen Kreis bildet, hat man eine TSP Tour.

Sowohl Y_1 (einlaufende Kanten) als auch Y_2 auslaufende Kanten erzeugen eine untere Schranke, wenn an jeder Stadt die kürzeste Kante gewählt wird. Man kann also die grössere von beiden wählen.