12.4 Traveling Salesman Problem

Definition 12.3(TSP, Problem des Handlungsreisenden): *Wir betrachten einen gerichteten, gewichteten Graphen $G = (V, E)$ mit $K : V^2 \to \mathbb{R}$. $K(\{u, v\})$ sind die Kosten der Kante, ∞ falls $\{u, v\} \notin E$.*

Wir suchen eine Rundreise *(einfacher Kreis) mit folgenden Kriterien:*

- *Die Rundreise enthält alle Knoten und*
- *die Summe der Kosten der betretenen Kanten ist* minimal.

Der Graph ist durch eine Distanzmatrix gegeben. ($\infty \Leftrightarrow$ keine Kante). Etwa $M = \big(M(u, v)\big)$, $1 \le u, v \le 4$.

$$M = \begin{array}{c} \begin{matrix} 1 \le u, v \le 4 \end{matrix} \\ \begin{pmatrix} \infty & 10 & 15 & 20 \\ 5 & \infty & 9 & 10 \\ 6 & 13 & \infty & 12 \\ 8 & 8 & 9 & \infty \end{pmatrix} \\ \begin{matrix} 1 & 2 & 3 & 4 \end{matrix} \end{array} \quad \begin{array}{c} \text{Zeile} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \quad \text{Spalte}$$

Die Kosten der Rundreise $R = (1, 2, 3, 4, 1)$ sind

$$K(R) = M(1, 2) + M(2, 3) + M(3, 4) + M(4, 1) = 10 + 9 + 12 + 8 = 39$$

Für $R' = (1, 2, 4, 3, 1)$ ist $K(R') = 35$ Ist das minimal?

Ein erster Versuch. Wir halten Knoten 1 fest und zählen alle $(n - 1)!$ Permutationen der Knoten $\{2, \ldots, n\}$ auf. Für jede, der sich so ergebenden möglichen Rundreise, ermitteln wir die Kosten.

Laufzeit:

$$\begin{aligned} \Omega((n-1)! \cdot n) = \Omega(n!) &\ge \left(\frac{n}{2}\right)^{\frac{n}{2}} \\ &= 2^{((\log n)-1)\cdot\frac{n}{2}} \\ &= 2^{\frac{(\log n)-1}{2}\cdot n} \\ &= 2^{\Omega(\log n)\cdot n} \\ &\gg 2^n \end{aligned}$$

Beachte:

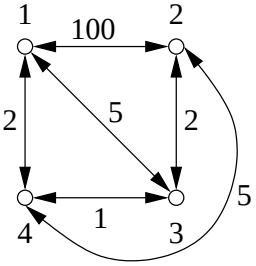
$$n^n = 2^{(\log n)\cdot n}, \quad n! \ge 2^{\frac{(\log n)-1}{2}\cdot n}, \quad n! \ll n^n$$

Zweiter Versuch. Wir versuchen es mit dem Greedy-Prinzip.

Gehe zum jeweils nächsten Knoten, der noch nicht vorkommt (analog Prim). Im Beispiel mit Startknoten 1 bekommen wir $R = (1, 2, 3, 4, 1)$, $K(R) = 39$, was nicht optimal ist. Wählen wir hingegen 4 als Startknoten, ergibt sich $R' = (4, 2, 1, 3, 4)$, $K(R') = 35$ optimal.

Aber: Das muss nicht immer klappen!

Beispiel 12.3: *Betrachten wir einmal den folgenden Graphen. Mit Greedy geht es zwingend in die Irre.*

$G =$ 

mit greedy gewählte Rundreisen:

$R_2 = (2, 3, 4, 1, 2)$ *Kosten* $= 105$
$R_3 = (3, 4, 1, 2, 3)$ *Kosten* $= 105$
$R_4 = (4, 3, 2, 1, 4)$ *Kosten* $= 105$
$R_1 = (1, 4, 3, 2, 1)$ *Kosten* $= 105$

Immer ist $1 \overset{100}{\longleftrightarrow} 2$ *dabei. Optimal ist* $R = (1, 4, 2, 3, 1)$ *mit Kosten von* 14. *Hier ist* $4 \to 2$ *nicht greedy, sondern mit Voraussicht gewählt. Tatsächlich sind für das TSP nur Exponentialzeitalgorithmen bekannt.*

Weiter führt unser Backtracking. Wir verzweigen nach dem Vorkommen einer Kante in der Rundreise. Das ist korrekt, denn: Entweder eine optimale Rundreise enthält die Kante (v, w) oder eben nicht.

Gehen wir auf unser Beispiel von Seite 196 Wir haben die aktuelle Matrix

$$M = \begin{pmatrix} \infty & 10 & 15 & 20 \\ 5 & \infty & 9 & 10 \\ 6 & 13 & \infty & 12 \\ 8 & 8 & 9 & \infty \end{pmatrix}$$

Wählen wir jetzt zum Beispiel die Kante $2 \to 3$, dann gilt:

- Keine Kante $u \to 3$, $u \neq 2$ muss noch betrachtet werden,
- keine Kante $2 \to v$, $v \neq 3$ muss noch betrachtet werden und
- die Kante $3 \to 2$ kann auch nicht mehr vorkommen.

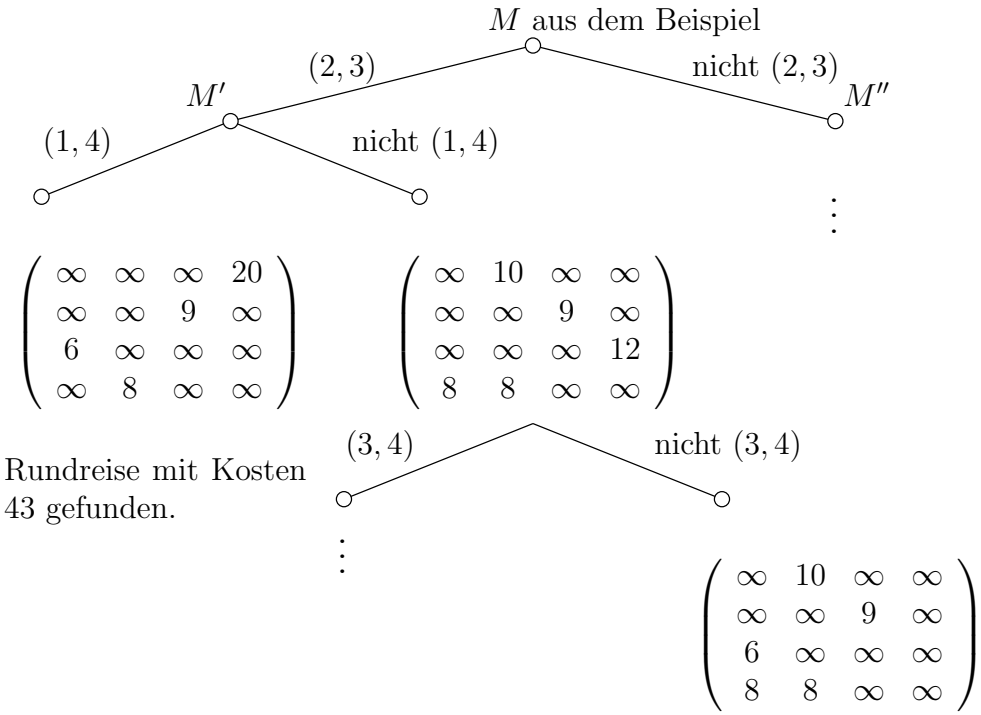
Das heißt, wir suchen eine optimale Reise in M'. Alle nicht mehr benötigten Kanten stehen auf ∞.

$$M' = \begin{pmatrix} \infty & 10 & \infty & 20 \\ \infty & \infty & 9 & \infty \\ 6 & \infty & \infty & 12 \\ 8 & 8 & \infty & \infty \end{pmatrix}$$

Ist dagegen $2 \to 3$ nicht gewählt, dann sieht M'' so aus. Nur $M(2,3)$ wird zu ∞.

$$M'' = \begin{pmatrix} \infty & 10 & 15 & 20 \\ 5 & \infty & \infty & 10 \\ 6 & 13 & \infty & 12 \\ 8 & 8 & 9 & \infty \end{pmatrix}$$

Für das Beispiel ergibt sich dann folgender Bachtrackingbaum.



$$\begin{pmatrix} \infty & \infty & \infty & 20 \\ \infty & \infty & 9 & \infty \\ 6 & \infty & \infty & \infty \\ \infty & 8 & \infty & \infty \end{pmatrix} \qquad \begin{pmatrix} \infty & 10 & \infty & \infty \\ \infty & \infty & 9 & \infty \\ \infty & \infty & \infty & 12 \\ 8 & 8 & \infty & \infty \end{pmatrix}$$

$$\begin{pmatrix} \infty & 10 & \infty & \infty \\ \infty & \infty & 9 & \infty \\ 6 & \infty & \infty & \infty \\ 8 & 8 & \infty & \infty \end{pmatrix}$$

Bemerkung: Wir haben eine Rundreise gefunden, wenn in *jeder Zeile* und in *jeder Spalte* noch *genau ein* Wert $\neq \infty$ steht und die n verbleibenden Kanten *einen einzigen* Kreis bilden. Falls in einer Zeile oder einer Spalte *nur noch* ∞ stehen, dann ist offensichtlich keine Rundreise mehr möglich.

Algorithmus 29: Backtracking für TSP

Input : Distanzmatrix M
Output : Minimale Rundreise in M, dargestellt als modifikation von M.

if *M stellt Rundreise dar* **then return** $(M, K(M))$;
if *M hat keine Rundreise* $< \infty$ **then**
  return (M, ∞); `/* z.B. Zeile/Spalte voller ∞ */`
end
Wähle eine Kante (u, v), $u \neq v$ mit $M(u, v) \neq \infty$, wobei in der Zeile von u oder der Spalte von v mindestens ein Wert $\neq \infty$ ist;
$M' = M$ modifiziert, so dass $u \to v$ gewählt; `/* (vgl. Seite 197) */`
$M'' = M$ modifiziert, so dass $M(u, v) = \infty$;
$(M_1, K_1) = \text{TSP}(M')$;
$(M_2, K_2) = \text{TSP}(M'')$;
if $K_1 \leq K_2$ **then** `/* Kleinere Rundreise zurückgeben. */`
  return (M_1, K_1);
else
  return (M_2, K_2);
end

Korrektheit: Induktion über die #Einträge in M, die den Wert ∞ haben. Solche Einträge kommen in jedem Schritt dazu.

Die Laufzeit ist zunächst einmal $O(2^{n(n-1)})$, $2^{n(n-1)} = 2^{\Omega(n^2)} \gg n^n$.

12.4.1 Branch-and-Bound

Um die Laufzeit von Backtracking-Algorithmen zu verbessern, versucht man im Backtrackingbaum ganze Zweige, die für die Lösung nicht mehr relevant sind, abzuschneiden. Das führt in der Praxis oft zu einer Beschleunigung. Im worst-case kann man aber nicht viel sagen.

Wir betrachten weiterhin das Problem des Handlungstreisenden. Welche Zweige im obigen Backtracking-Algorithmus können wir gefahrlos abschneiden?

Bei Optimierungsproblemen sind das die Zweige, die mit Sicherheit keine bessere Lösung als die bereits gefundenen Lösungen liefen *können*. Das Problem ist nur, wie soll man die erkennen?

Dazu definiert man eine *untere* bzw. *obere* Schranke für die ab dem betrachteten Punkt noch zu erwartenden Lösungen. Je nachdem, ob es sich um ein *Maximierungs*- oder ein *Minimierungs*-Problem handelt.

Für das TSP wählen wir also eine *untere Schranke* $S(M)$. Zu einer Matrix M – das ist ein Knoten im Backtrackingbaum – ist $S(M)$ *kleiner oder gleich* der Kosten *jeder* Rundreise, die in M möglist ist. Damit gilt auch $S(M) \leq$ Kosten einer minimalen Rundreise.

Wenn wir ein solches $S(M)$ haben, können wir das folgende Prinzip während des Backtrackings anwenden.

> Haben wir eine Rundreise der Kosten K gefunden, dann brauchen wir keine Auswertung (Expansion) der Kosten an Knoten mit $S(M) \geq K$ mehr durchführen.

Zunächst einmal zwei „offensichtliche“ Schranken.

$$S_1(M) = \text{ minimale Kosten einer Rundreise unter } M$$

Das ist die *beste*, das heißt größte Schranke, aber nicht polynomial zu ermitteln. Dazu müssten wir schließlich das Problem selbst lösen.

$$S_2(M) = \text{ Summe der Kosten der bei } M \text{ gewählten Kanten}$$

Ist eine untere Schranke bei $M(u, v) \geq 0$. Das können wir aber hier im Unterschied zu den kürzesten Wegen annehmen. (Wieso?)

Ist M die Wurzel des Backtrackingbaumes, dann ist im Allgemeinen $S_2(M) = 0$, sofern nicht die Zeile $(\infty\, \infty\, \infty\, m\, \infty\, \infty)$ oder analoges für eine Spalte auftritt. S_2 ist auch einfach zu berechnen, das ist schonmal ein guter Anfang.

Die folgenden Schranken sind etwas aufwendiger zu berechnen, liefern dafür aber auch bessere Ergebnisse.

$$S_3(M) = \frac{1}{2} \cdot \sum_v \min\left\{M(u,v) + M(v,w) \mid u,w \in V\right\}$$

Das ist jedesmal das Minimum, um durch den Knoten v zu gehen. ($u \overset{\min}{\to} v \overset{\min}{\to} w$) Wieso ist $S_3(M)$ eine untere Schranke?

Sei $R = (1, v_1, \ldots, v_{n-1}, 1)$ eine Rundreise unter M. Dann gilt:

$$\begin{aligned}
K(R) &= M(1,v_1) + M(v_1,v_2) + \cdots + M(v_{n-1},1) \\
&\quad \text{(In der Summe alles verdoppelt.)} \\
&= \frac{1}{2}\Big(M(1,v_1) + M(1,v_1) + M(v_1,v_2) + M(v_1,v_2) + \cdots + \\
&\qquad + M(v_{n-1},1) + M(v_{n-1},1)\Big) \\
&\quad \text{(Shift um 1 nach rechts.)} \\
&= \frac{1}{2}\Big(M(v_{n-1},\underbrace{1}_{v=1}) + M(\underbrace{1}_{v=1},v_1) + M(1,\underbrace{v_1}_{v=v_1}) + M(\underbrace{v_1}_{v=v_1},v_2) + \\
&\qquad + M(v_1,\underbrace{v_2}_{v=v_2}) + M(\underbrace{v_2}_{v=v_2},v_3) + \ldots \\
&\qquad + M(v_{n-2},\underbrace{v_{n-1}}_{v=v_{n-1}}) + M(\underbrace{v_{n-1}}_{v=v_{n-1}},1)\Big) \\
&\geq \frac{1}{2} \cdot \sum_v \min\left\{M(u,v) + M(v,w) \mid u,w \in V\right\}
\end{aligned}$$

Also ist $S_3(M)$ eine korrekte untere Schranke für eine Rundreise in M.

Es ist $S_3(M) = \infty \Longleftrightarrow$ Eine Zeile oder Spalte voller ∞. Ebenso für $S_2(M)$.

Betrachten wir M von Seite 196 Wir ermitteln $S_2(M)$.

$$\begin{aligned}
v = 1 &\Rightarrow 2 \xrightarrow{5} 1 \xrightarrow{10} 2 && \text{Das ist in einer Rundreise nicht möglich,} \\
& && \text{bei der Schranke aber egal.} \\
v = 2 &\Rightarrow 4 \xrightarrow{8} 2 \xrightarrow{5} 1 \\
v = 3 &\Rightarrow 2 \xrightarrow{9} 3 \xrightarrow{6} 1 && \text{oder} \quad 4 \xrightarrow{9} 3 \xrightarrow{6} 1 \\
v = 4 &\Rightarrow 2 \xrightarrow{10} 4 \xrightarrow{8} 1 && \text{oder} \quad 2 \xrightarrow{10} 4 \xrightarrow{8} 2
\end{aligned}$$

Also $S_3(M) = \frac{1}{2} \cdot 61 = 30,5$. Wegen ganzzahliger Kosten sogar $S_3(M) = 31$. Somit gilt, dass $K(R) \geq 31$ für jede Rundreise R.

Beispiel 12.4: *Ist* $2 \to 3$ *bei* M *gewählt, dann*

$$M' = \begin{pmatrix} \infty & 10 & \infty & 20 \\ \infty & \infty & 9 & \infty \\ 6 & \infty & \infty & 12 \\ 8 & 8 & \infty & \infty \end{pmatrix}$$

und

$$\begin{array}{lcl} v = 1 & \Rightarrow & 3 \xrightarrow{6} 1 \xrightarrow{10} 2 \\ v = 2 & \Rightarrow & 4 \xrightarrow{8} 2 \xrightarrow{9} 3 \\ v = 3 & \Rightarrow & 2 \xrightarrow{9} 3 \xrightarrow{6} 1 \\ v = 4 & \Rightarrow & 3 \xrightarrow{12} 4 \xrightarrow{8} 2 \quad \textit{oder} \quad 3 \xrightarrow{12} 4 \xrightarrow{8} 1. \end{array}$$

$S_3(M') = \frac{1}{2}(16 + 17 + 15 + 20) = 34$, *wogegen* $S_2(M') = 9$.

Kommen wir nun zur letzten Schranke. Betrachten wir eine allgemeine Matrix $M = \big(M(u,v)\big)_{1 \le u,v \le n}$. Alle Einträge $M(u,v) \ge 0$.

Ist R eine Rundreise zu M, dann ergeben sich die Kosten von R als

$$K(R) = z_1 + z_2 + \cdots + z_n,$$

wobei z_i einen geeigneten Wert aus Zeile i der Matrix darstellt. Alternativ ist

$$K(R) = s_1 + s_2 + \cdots + s_n,$$

wobei s_j einen geeigneten Wert aus Spalte j darstellt. Dann gilt

$$\begin{array}{rcl} S'_4(M) & = & \min\big\{\overbrace{M(1,1), M(1,2), \ldots, M(1,n)}^{\text{Zeile 1}}\big\} + \\ & & \min\big\{\overbrace{M(2,1), M(2,2), \ldots, M(2,n)}^{\text{Zeile 2}}\big\} + \\ & & \cdots + \\ & & \min\big\{\overbrace{M(n,1), M(n,2), \ldots, M(n,n)}^{\text{Zeile } n}\big\} \end{array}$$

nimmt aus jeder Zeile den kleinsten Wert. $S'_4(M)$ ist eine korrekte Schranke.

Wenn jetzt noch nicht alle Spalten vorkommen, können wir noch besser werden. Und zwar so:

Sei also für $1 \le u \le n \quad M_u = \min\big\{M(u,1), M(u,2), \ldots, M(u,n)\big\}$ der kleinste Wert der Zeile u. Wir setzen

$$\widehat{M} = \begin{pmatrix} M(1,1) - M_1 & \ldots & M(1,n) - M_1 \\ M(2,1) - M_2 & \ldots & M(2,n) - M_2 \\ \vdots & \ddots & \vdots \\ M(n,1) - M_n & \ldots & M(n,n) - M_n \end{pmatrix}$$

Alles ≥ 0. Pro Zeile ist mindestens ein Eintrag 0.

Es gilt für jede Rundreise R zu M,

$$K_M(R) = K_{\widehat{M}}(R) + S_4'(M) \geq S_4'(M).$$

mit K_M = Kosten in M und $K_{\widehat{M}}$ = Kosten in $\widehat{M}$, in $\widehat{M}$ alles ≥ 0.

Ist in $\widehat{M}$ auch in jeder Spalte eine 0, so $S_4'(M)$ = die minimalen Kosten einer Rundreise. Ist das nicht der Fall, iterieren wir den Schritt mit den Spalten von $\widehat{M}_u$.

Ist also $S_u = \min\{\overbrace{\widehat{M}(1,u), \widehat{M}(2,u), \ldots, \widehat{M}(n,u)}^{\text{Spalte } u \text{ von } \widehat{M}}\}$, dann

$$\widehat{\widehat{M}} = \begin{pmatrix} \widehat{M}(1,1) - S_1 & \ldots & \widehat{M}(1,n) - S_n \\ \widehat{M}(2,1) - S_1 & \ldots & \widehat{M}(2,n) - S_n \\ \vdots & \ddots & \vdots \\ \widehat{M}(n,1) - S_1 & \ldots & \widehat{M}(n,n) - S_n \end{pmatrix}$$

Alles ≥ 0. Pro Spalte eine 0, pro Zeile eine 0.

Für jede Rundreise R durch $\widehat{M}$ gilt

$$K_{\widehat{M}}(R) = K_{\widehat{\widehat{M}}}(R) + S_1 + \cdots + S_n.$$

Also haben wir insgesamt:

$$\begin{aligned} K_M(R) &= K_{\widehat{M}}(R) + M_1 + \cdots + M_n \\ &= K_{\widehat{\widehat{M}}}(R) + \overbrace{S_1 + \cdots + S_n}^{\text{aus } \widehat{M}} + \underbrace{M_1 + \cdots + M_n}_{\text{aus } M} \\ &\quad (\text{In } \widehat{\widehat{M}} \text{ alles} \geq 0) \\ &\geq M_1 + \cdots + M_n + S_1 + \cdots + S_n. \end{aligned}$$

Wir definieren jetzt offiziell:

$$S_4(M) = M_1 + \cdots + M_n + S_1 + \ldots S_n$$

Aufgabe: $S_1(M) \geq S_4(M) \geq S_3(M) \geq S_2(M)$. $S_4(M) \geq S_3(M)$ ist nicht ganz so offensichtlich.

Beispiel 12.5: *In unserem Eingangsbeispiel von Seite 196 ist*

$$M = \begin{pmatrix} \infty & 10 & 15 & 20 \\ 5 & \infty & 9 & 10 \\ 6 & 13 & \infty & 12 \\ 8 & 8 & 9 & \infty \end{pmatrix} \quad \begin{matrix} M_1 = 10 \\ M_2 = 5 \\ M_3 = 6 \\ M_4 = 8 \end{matrix}$$

$$\widehat{M} = \begin{pmatrix} \infty & 0 & 5 & 10 \\ 0 & \infty & 4 & 5 \\ 0 & 7 & \infty & 6 \\ 0 & 0 & 1 & \infty \end{pmatrix} \quad \begin{matrix} S_1 = 0 \\ S_2 = 0 \\ S_3 = 1 \\ S_4 = 5 \end{matrix}$$

$$\widehat{\widehat{M}} = \begin{pmatrix} \infty & 0 & 4 & 5 \\ 0 & \infty & 3 & 0 \\ 0 & 7 & \infty & 1 \\ 0 & 0 & 0 & \infty \end{pmatrix} \quad S_4(M) = 35 \; > \; S_3(M) = 31$$

Die Schranken lassen sich für jede Durchlaufreihenfolge des Backtracking-Baumes zum Herausschneiden weiterer Teile verwenden.

Zusammenfassend noch einmal das prinzipielle Vorgehen.

Algorithmus 30: Offizielles branch-and-bound

1. Die Front des aktuellen Teils des Backtrackingbaumes steht im Heap, geordnet nach $S(M)$ $\quad (S(M) =$ die verwendete Schranke);
2. Immer an dem M mit $S(M)$ minimal weitermachen;
3. Minimale gefundene Rundreise merken;
4. Anhalten, wenn minimales $S(M)$ im Heap $\geq$ Kosten der minimalen bisher gefundenen Rundreise;

Beispiel 12.6: *Zum Abschluß noch das Beispiel von Seite 196 mit branch-and-bound und der Schranke S_4.*

Hier noch eine etwas andere Darstellung. Wir geben nur noch die Suchfront Q an. In Q sind alle Matrizen, die noch eine Rundreise enthalten können. Den Test, ob wir eine Rundreise haben, machen wir beim Herausnehmen der M aus Q.

1. *Am Anfang:*

$$M = \begin{pmatrix} \infty & 10 & 15 & 20 \\ 5 & \infty & 9 & 10 \\ 6 & 13 & \infty & 12 \\ 8 & 8 & 9 & \infty \end{pmatrix} \quad S_4(M) = 35$$

$$Q = \big[(M, 35)\big]$$

2. Kante $2 \to 1$*:*

$$M_{(2,1)} = \begin{pmatrix} \infty & \infty & 15 & 20 \\ 5 & \infty & \infty & \infty \\ \infty & 13 & \infty & 12 \\ \infty & 8 & 9 & \infty \end{pmatrix} \quad S_4(M_{(2,1)}) = 40$$

$$M_{\neg(2,1)} = \begin{pmatrix} \infty & 10 & 15 & 20 \\ \infty & \infty & 9 & 10 \\ 6 & 13 & \infty & 12 \\ 8 & 8 & 9 & \infty \end{pmatrix} \quad \begin{array}{l} S_4(M_{\neg(2,1)}) = 34 < S_4(M) = 35 \\ \textit{Wir nehmen weiterhin } 35 \textit{ als Schranke,} \\ \textit{da } M_{\neg(2,1)} \textit{ aus } M \textit{ entstanden ist.} \end{array}$$

$$Q = \big[(M_{\neg(2,1)}, 35), (M_{(2,1)}, 40)\big]$$

3. Kante $(3 \to 1)$*:*

$$M_{\neg(2,1),(3,1)} = \begin{pmatrix} \infty & 10 & \infty & 20 \\ \infty & \infty & 9 & 10 \\ 6 & \infty & \infty & \infty \\ \infty & 8 & 9 & \infty \end{pmatrix} \quad \begin{array}{l} S_4(M_{\neg(2,1),(3,1)}) = 34 = 35 \\ \textit{(siehe oben)} \end{array}$$

$$M_{\neg(2,1),\neg(3,1)} = \begin{pmatrix} \infty & 10 & 15 & 20 \\ \infty & \infty & 9 & 10 \\ \infty & 13 & \infty & 12 \\ 8 & 8 & 9 & \infty \end{pmatrix} \quad S_4(M_{\neg(2,1),\neg(3,1)}) = 39$$

$$Q = \big[(M_{\neg(2,1),(3,1)}, 35), (M_{\neg(2,1),\neg(3,1)}, 39), (M_{(2,1)}, 40)\big]$$

4. Kante $(4 \to 2)$*:*

$$M_{\neg(2,1),(3,1),(4,2)} = \begin{pmatrix} \infty & \infty & \infty & 20 \\ \infty & \infty & 9 & \infty \\ 6 & \infty & \infty & \infty \\ \infty & 8 & \infty & \infty \end{pmatrix} \quad S_4(M_{\neg(2,1),(3,1),(4,2)}) = 43$$

$$M_{\neg(2,1),(3,1),\neg(4,2)} = \begin{pmatrix} \infty & 10 & \infty & 20 \\ \infty & \infty & 9 & 10 \\ 6 & \infty & \infty & \infty \\ \infty & \infty & 9 & \infty \end{pmatrix} \quad S_4(M_{\neg(2,1),(3,1),\neg(4,2)}) = 35$$

$$Q = \big[(M_{\neg(2,1),(3,1),\neg(4,2)}, 35), (M_{\neg(2,1),\neg(3,1)}, 39), (M_{(2,1)}, 40), (M_{\neg(2,1),(3,1),(4,2)}, 43)\big]$$

$M_{\neg(2,1),(3,1),(4,2)}$ *ist bereits eine Rundreise.* $(1 \to 4 \to 2 \to 3 \to 1)$ *Wir fügen sie trotzdem in den Heap ein. Wenn es noch Matrizen gibt, die bessere Rundreisen liefern können, werden die vorher bearbeitet.*

5. *Kante* $(2 \to 4)$*:*

$$M_{\neg(2,1),(3,1),\neg(4,2),(2,4)} = \begin{pmatrix} \infty & 10 & \infty & \infty \\ \infty & \infty & \infty & 10 \\ 6 & \infty & \infty & \infty \\ \infty & \infty & 9 & \infty \end{pmatrix} \quad S_4(M_{\neg(2,1),(3,1),\neg(4,2),(2,4)}) = 35$$

$$M_{\neg(2,1),(3,1),(4,2),\neg(2,4)} = \begin{pmatrix} \infty & 10 & \infty & 20 \\ \infty & \infty & 9 & \infty \\ 6 & \infty & \infty & \infty \\ \infty & \infty & 9 & \infty \end{pmatrix} \quad S_4(M_{\neg(2,1),(3,1),(4,2),\neg(2,4)}) = 44$$

$$Q = \big[(M_{\neg(2,1),(3,1),\neg(4,2),(2,4)}, 35), (M_{\neg(2,1),\neg(3,1)}, 39), (M_{(2,1)}, 40), (M_{\neg(2,1),\neg(3,1),(4,2)}, 43), (M_{\neg(2,1),(3,1),(4,2),\neg(2,4)}, 44)\big]$$

6. *Als nächstes ist* $M_{\neg(2,1),(3,1),\neg(4,2),(2,4)}$ *an der Reihe. Das ist eine fertige Rundreise mit Kosten* 35*. Alle anderen* unteren Schranken *sind größer. Die verbleibenden Matrizen im Heap können also keine bessere Rundreise mehr liefern.*

   Damit haben wir mit

$$(1 \to 2 \to 4 \to 3 \to 1)$$

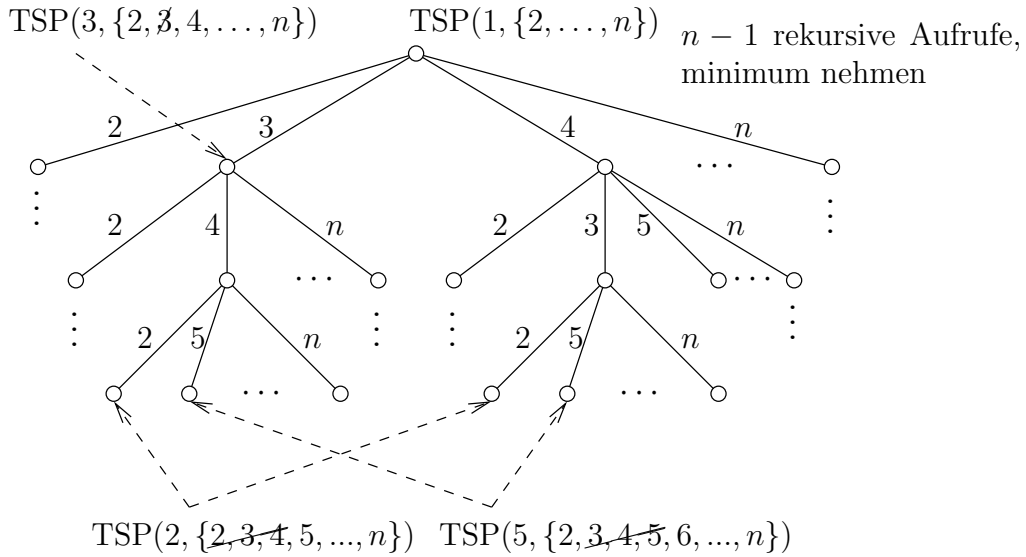
   eine minimale Rundreise im M *gefunden und sind fertig.*

Aufgabe: Zeigen Sie für M und M', wobei M' unter M im Backtrackingbaum hängt Baum, dass $S_2(M') \geq S_2(M), S_3(M') \geq S_3(M)$. (Für S_4 gilt das, wie oben gesehen, so nicht.)

12.4.2 TSP mit dynamischer Programmierung

Das Backtracking mit Verzweigen nach dem Vorkommen einer Kante ist praktisch wichtig und führt ja auch zum branch-and-bound. Jedoch, viel Beweisbares ist dort bisher nicht herausgekommen. Wir fangen mit einer anderen Art des Backtracking an.

Verzweigen nach dem nächsten Knoten, zu dem eine Rundreise gehen soll. Der Prozeduraufrufbaum hat dann folgende Struktur:

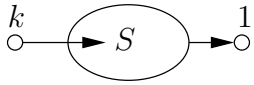


Die Tiefe des Baumes ist $n-1$, wir haben $(n-1)(n-2)\ldots 1 = (n-1)!$ Blätter. Wir bekommen Aufrufe der Art $\mathrm{TSP}(k, \underbrace{\{i_1, \ldots, i_l\}}_{\subseteq\{2,\ldots,n\}})$.

Wieviele verschiedene rekursive Aufrufe $\mathrm{TSP}(k, S)$ gibt es prinzipiell?

- Wähle k: n Möglichkeiten
- Wähle S: $\leq 2^{n-1}$ Möglichkeiten
- $2^{n-1} \cdot n = 2^{n-1+\log n} = 2^{O(n)}$, wogegen $(n-1)! = 2^{\overbrace{\Omega(\log n) \cdot n}^{\gg n}}$

Wir tabellieren wieder die Ergebnisse der rekursiven Aufrufe, wobei $TSP(k, S) =$ kürzeste Reise der Art:



$k \neq 1,\ 1, k \notin S$

Wir suchen einen Weg beginnend beim Knoten k, der über *alle* Knoten in S zum Knoten 1 führt.

- Zunächst $S = \emptyset$: (Zwischen k und 1 liegen *keine* Knoten.)

$$\begin{aligned} TSP(2,\emptyset) &= M(2,1) \qquad M = \big(M(u,v)\big) \text{ Eingangsmatrix} \\ &\vdots \\ TSP(n,\emptyset) &= M(n,1) \end{aligned}$$

- Dann $|S| = 1$:

$$\begin{aligned} TSP(2,\{3\}) &= M(2,3) + TSP(3, \underbrace{S \setminus \{3\}}_{=\emptyset}) \\ &\vdots \\ TSP(2,\{n\}) &= M(2,n) + TSP(n, \underbrace{S \setminus \{n\}}_{=\emptyset}) \\ &\vdots \end{aligned}$$

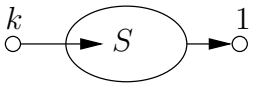
- Dann $|S| = 2$

$$\begin{aligned} TSP(2,\{3,4\}) &= \min\big\{M(2,3) + TSP(3,\{4\}),\ M(2,4) + TSP(4,\{3\})\big\} \\ TSP(2,\{3,5\}) &= \min\big\{M(2,3) + TSP(3,\{5\}),\ M(2,5) + TSP(5,\{3\})\big\} \\ &\vdots \end{aligned}$$

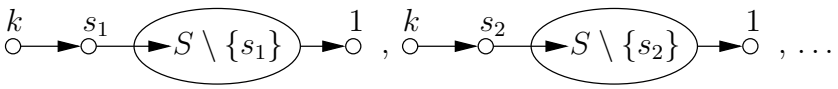
- Allgemein:

$$TSP(k,S) = \min\big\{M(k,s) + TSP(s, S \setminus \{s\}) \mid s \in S\big\}$$

In Worten: Wir suchen jeweils das *minimale Reststück* einer Rundreise. Das Reststück beginnt mit $k \to s$ und durchläuft alle Knoten der Restmenge S. Also bilden wir das Minimum über alle Einstiegspunkte $s \in S$.



ist das Minimum von



für alle Nachbarn $s_i \in S$ von k. Falls k keine Nachbarn in S hat, ∞. Dann gibt es keine Rundreise dieser Form.

Algorithmus 31: TSP mit dynamischer Programmierung

Input : Gerichteter Graph G dargestellt als Distanzmatrix M.
Output : Minimale Rundreise in G, falls eine existiert.
Data : 2-dimensionales Array TSP indiziert mit [1, . . . , n] für den Startpunkt und $[0 = \emptyset, \ldots, 2^{n-1} - 1 = \{2, \ldots, n\}]$, allen möglichem Mengen von Restknoten

```
1  for k = 1 to n do
2  |  TSP(k, ∅) = M(k, 1);
3  end
4  for i = 1 to n − 1 do
   |  /* Alle Teilmengen der Restknoten der Größe nach. */
5  |  foreach S ⊆ {2, . . . , n}, |S| = i do
   |  |  /* Alle Knoten, in denen wir vor betreten der Menge S
   |  |     sein können. */
6  |  |  foreach k ∉ S do
7  |  |  |  TSP(k, S) = min{M(k, s) + TSP(s, S \ {s}) | s ∈ S};
8  |  |  end
9  |  end
10 end
```

Das Ergebnis ist

$$TSP(1, \{2, \ldots, n\}) = \min\big\{M(1, s) + TSP(s, \{2, \ldots, n\} \setminus s) \mid s \in \{2, \ldots, n\}\big\}.$$

Laufzeit: $O(n \cdot 2^n)$ Einträge im Array TSP. Pro Eintrag das Minimum ermitteln, also Zeit $O(n)$. Ergibt insgesammt $O(n^2 \cdot 2^n)$.

Es ist

$$n^2 \cdot 2^n = 2^{n+2\log n} \leq 2^{n(1+\varepsilon)} = 2^{(1+\varepsilon)\cdot n} = \big(2(1+\varepsilon')\big)^n \ll (n-1)!$$

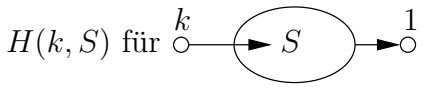
für $\varepsilon, \varepsilon' \geq 0$ geeignet. ($2^\varepsilon \to 1$ für $\varepsilon \to 0$, da $2^0 = 1$.)

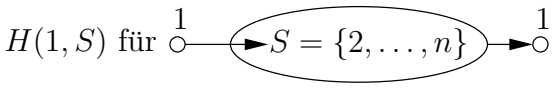
12.5 Hamilton-Kreis und Eulerscher Kreis

Verwandt mit dem Problem des Handlungsreisenden ist das Problem des Hamiltonschen Kreises.

Definition 12.4(Hamilton-Kreis): *Sei $G = (V, E)$ ein ungerichteter Graph. Ein Hamilton-Kreis ist ein einfacher Kreis der Art $(1, v_1, v_2, \ldots, v_{n-1}, 1)$. (Also ist dies ein Kreis, in dem alle Knoten genau einmal auftreten.)*

Mit dynamischem Programmieren

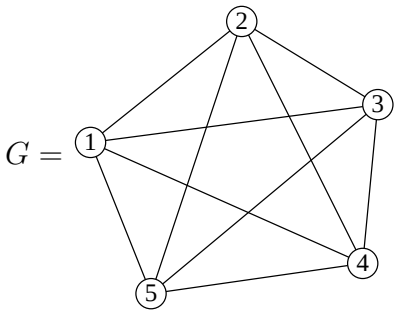




in Zeit $O(n^2 \cdot 2^n)$ lösbar. Besser als $\Omega((n-1)!)$ druch einfaches Backtracking. Geht genauso wie TSP mit Kosten überall $= 1$.

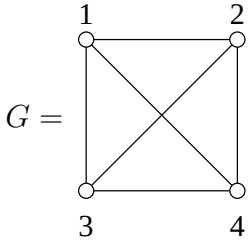
Definition 12.5(Eulerscher Kreis): *Sei $G = (V, E)$ ein ungerichteter Graph. Ein Eulerscher Kreis ist ein geschlossener Pfad, in dem jede Kante genau einmal vorkommt.*

Beispiel 12.7:



Dann sollte $(1, 5, 4, 3, 2, 1, 4, 2, 5, 3, 1)$ *ein Eulerscher Kreis in G sein.*

Beispiel 12.8: *Wie sieht das hier aus?*



$(1, 2, 3, 4, 1, 3, ?)$, $\{2, 4\}$ *fehlt*

$(1, 2, 4, 1, 3, 4, ?)$, $\{3, 4\}$ *fehlt*

Scheint nicht zu gehen. Es fehlt immer eine Kante, außerdem kein geschlossener Pfad möglich?!

Dynamisches Programmieren? $O(m \cdot n \cdot 2^m)$, $|V| = n$, $|E| = m$ sollte klappen. Es geht aber in polyninomialer Zeit und wir haben wieder den Gegensatz

- Hamilton-Kreis: polynomiale Zeit nicht bekannt,
- Eulerscher Kreis: polynomiale Zeit.

Wie findet man jetzt in Polynomialzeit heraus, ob ein Graph einen Eulerschen Kreis entält? Und wie findet man ihn?

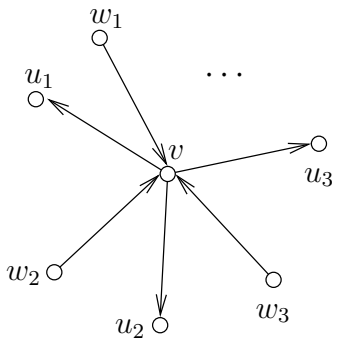
Der Beweis des folgenden Satzes liefert praktischerweise gleich einen Algorithmus dazu.

Satz 12.4: *Sei G ohne Knoten vom Grad 0 (*$\mathrm{Grad}(v)$ *= #direkter Nachbarn von v).*

G hat Eulerschen Kreis $\Longleftrightarrow$ G ist zusammenhängend und $Grad(v)$ ist gerade für alle $v \in V$.

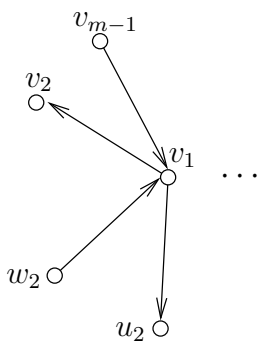
Beweis. „$\Rightarrow$“ Sei also $G = (V, E), |V| = n, |E| = m$. Sei $K = (v_1, v_2, v_3, \ldots, v_m = v_1)$ ein Eulerscher Kreis von G.

Betrachten wir einen beliebigen Knoten $v \neq v_1$ der etwa k-mal auf K vorkommt, dann haben wir eine Situation wie



alle u_i, w_i verschieden, also $Grad(v) = 2k$.

Für $v = v_1$ haben wir



und $Grad(v_1)$ ist gerade.

„⇐“ Das folgende ist eine Schlüsselbeobachtung für Graphen, die zusammenhängend sind und geraden Grad haben.

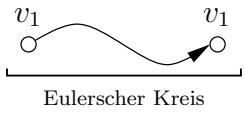
Wir beginnen einen Weg an einem beliebigen Knoten v_1 und benutzen jede vorkommende Kante nur einmal. Irgendwann landen wir wieder an v_1:

$$W = v_1 \to v_2 \to v_3 \to v_4 \to v_5 \to v_6 \to \cdots \to v_1$$

Ist etwa $v_3 = v_5 = u$, so hat u mindestens 3, also ≥ 4 Nachbarn. Wir können also von v_5 wieder durch eine neue Kante weggehen. Erst, wenn wir bei v_1 gelandet sind, ist das nicht mehr sicher, und wir machen dort Schluss.

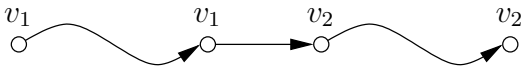
Nehmen wir nun W aus G heraus, haben alle Knoten wieder geraden Grad und wir können mit den Startknoten $v_1, v_2, \ldots$ so weitermachen und am Ende alles zu einem Eulerschen Kreis zusammensetzen. Konkret sieht das so aus:

1. Gehe einen Weg W wie oben und streiche die Kanten von W aus G. (W gehört nicht direkt zum Eulerschen Kreis)
2. Nach Induktionsvoraussetzung haben wir einen Eulerschen Kreis auf dem Stück, das jetzt noch von v_1 erreichbar ist. Schreibe diesen Eulerschen Kreis hin. Wir bekommen:



3. Erweitere den Eulerschen Kreis auf W zu

Mache von v_2 aus dasselbe wie bei v_1.



So geht es weiter bis zum Ende von W.

□

Aufgabe: Formulieren Sie mit dem Beweis oben einen (rekursiven) Algorithmus, der in $O(|V| + |E|)$ auf einen *Eulerschen Kreis* testet und im positiven Fall einen *Eulerschen Kreis* liefert.