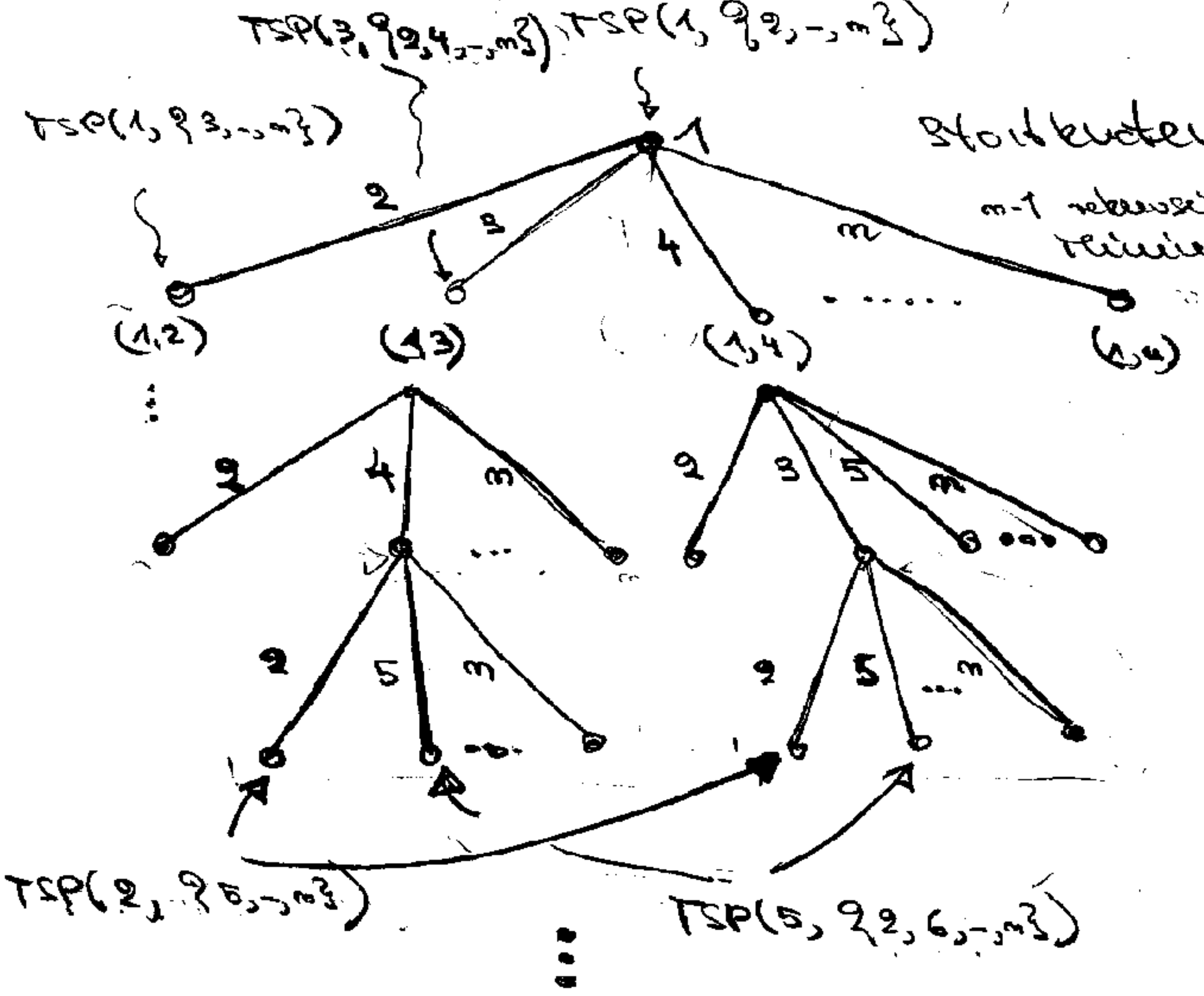
(10.83)

Das backtracking mit Verzweigen nach dem Vorkommen einer Kante ist praktisch wichtig und führt ja auch zum branch-and-bound. Jedoch, viel Bemerkbares ist dort bisher nicht herausgekommen.

Wir fangen mit einer anderen Art des backtracking an: Verzweigen nach dem nächsten Knoten, zu dem die Rundreise gehen soll.

Der Prozeduraufruf bauen hat dann folgende Struktur:



Startknoten

n-1 rekursive Aufrufe, Minimum nehmen.

Tiefe n-1

$(n-1)(n-2)\ldots 1 = (n-1)!$

Blätter. Zeit so

$\Omega((n-1)!)$.

Aufrufe der Art : $TSP(k, \underbrace{\{i_1,..,i_e\}}_{\subseteq \{2,..,m\}})$.

10.8?

Wieviele verschiedene rekursive Aufrufe TSP(k, S) gibt es prinzipiell?

- Wähle k : n Möglichkeiten.
- Wähle S : $\leq 2^{n-1}$ Möglichkeiten.
  1 nie dabei!

$$2^{n-1} \cdot n = 2^{n-1+\log_2 n} = 2^{O(n)} \ll (n-1)!$$

wogegen

$$(n-1)! = 2^{\overbrace{\Omega(\log n)\cdot n}^{\gg n}}.$$

Wir tabellieren die Ergebnisse der rekursiven Aufrufe, in der Reihenfolge, wobei

TSP(k, S) = kürzeste Route

$$k \longrightarrow S \longrightarrow 1, \quad k \neq 1, \; 1 \notin S:$$

10.86

Zunächst $S = \emptyset$:

$TSP(1, \emptyset) = 0$

$TSP(2, \emptyset) = M(2,1)$ // $M = (M(u,v))$

// Eingabematrix.

$\vdots$

$TSP(n, \emptyset) = M(n,1)$

Dann $|S| = 1$.

$TSP(2, \{3\}) = M(2,3) + TSP(3, S \setminus \{3\})$

$\vdots$

$TSP(2, \{n\}) = M(2,n) + TSP(n, S \setminus \{n\})$

$\vdots$

Dann $|S| = 2$

$TSP(2, \{3,4\}) =$

$= \min \{ M(2,3) + TSP(3, \{4\}),$

$M(2,4) + TSP(4, \{3\}) \}$

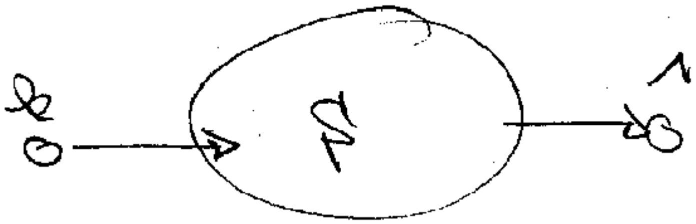
⋮

Allgemein

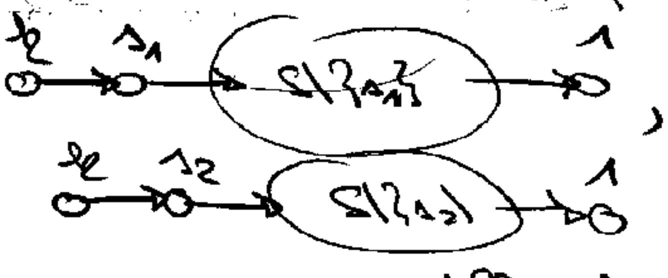
$\mathrm{TSP}(k, S)$

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

Also: Minimum über den Einstiegspunkt in S:



= Minimum von



: Alle Elemente aus S probieren.

Algorithmus (TSP mit dynamischem Programmieren

Datenstruktur

array $TSP[1,\dots,n, \overbrace{0\dots0}^{n-1 \text{ Bits}}, \dots, \overbrace{1\dots1}^{n-1 \text{ Bits}}]$

of integer

1. for $k=2$ to n $TSP(k, 0\dots0) := M(k,1)$

2. for $i=2$ to $n-2$ {

 for all $S \subseteq \{2,\dots,n\}, |S|=i$ {

 for all $k \in \{2,\dots,n\} \setminus S$ {

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

 }

 } }

Ergebnis ist

$$TSP(1, \{2,\dots,n-1\}) = \min \{ M(1,s) + TSP(s, \{2,\dots,n\} \setminus \{s\}) \}$$

$\square$

10.2

Laufzeit: $O(n \cdot 2^n)$ Einträge im array TSP. Pro Eintrag das Minimum ermitteln, Zeit $O(n)$. Also $O(n^2 \cdot 2^n)$

Es ist n^2.

$$n^2 \cdot 2^n = 2^{n + 2\log n} \leq 2^{n(1+\varepsilon)}$$

$$= 2^{(1+\varepsilon)\cdot n} = \left(2(1+\varepsilon')\right)^n \ll (n-1)!$$

für $\varepsilon, \varepsilon' > 0$ geeignet.

$2^{\varepsilon} \to 1$ für $\varepsilon \to 0$, da $2^0 = 1$!

Verwandt mit dem Problem des Handlungsreisenden ist das Problem des Hamilton'schen Kreises.

Definition (Hamilton'scher Kreis)

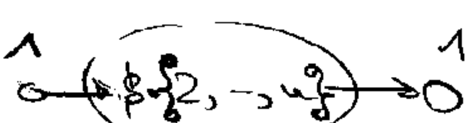
Sei $G=(V,E)$ ein ungerichteter Graph. Ein Hamilton-Kreis ist ein einfacher Kreis der Art

$$(1, v_1, \ldots, v_{n-1}, 1).$$

(Also ein Kreis in dem alle Knoten (genau einmal) auftreten.) □

Mit dynamischem Programmieren

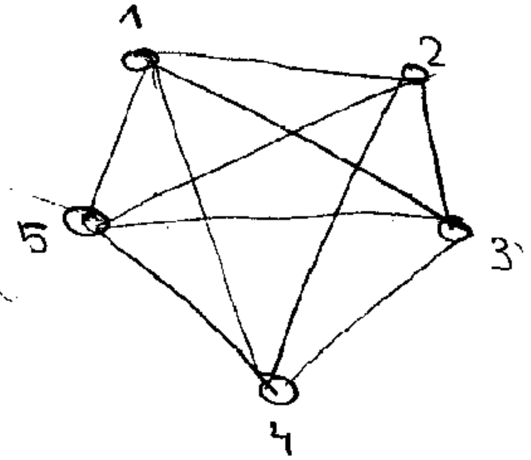
$H(k, S)$ = für 

$H(1, S)$ für 

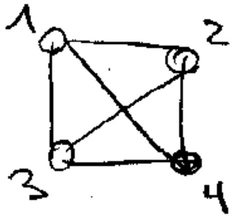
in Zeit $O(n^2 \cdot 2^n)$ lösbar. Besser als $\Omega((n-1)!)$ durch einfaches Backtracking.

Definition (Eulerscher Kreis)

Sei $G = (V, E)$ ein ungerichteter Graph. Ein Eulerscher Kreis ist ein geschlossener Weg in dem jede Kante genau einmal vorkommt. □



Dann $(1,5,4,3,2;1,4,2,5,3,1)$ sollte Eulerscher Kreis sein



$(1,3,4,3,2 \begin{smallmatrix} 1 \\ 4 \end{smallmatrix})$ ↯

$(1,4,2,1,3,4-1)$ ↯

Scheint nicht zu gehen. Dynamisches Programmieren ↯ $O(m \cdot m \cdot 2^m)$, $m=|E|$

$|V|=n$, $|E|=m$ sollte klappen.

Es geht aber in polynomialer Zeit und wir haben wieder:

- Hamilton Kreis poly. Zeit nicht bekannt.
- Eulerscher Kreis poly. Zeit.

Dazu der

grd(v) = # direkte Nachbarn

Satz

Sei G ohne Knoten vom Grad 0.

G hat Eulerschen Kreis

$\Leftrightarrow$

- G zusammenhängend und
- grd(v) gerade für alle $v \in V$.

Beweis:

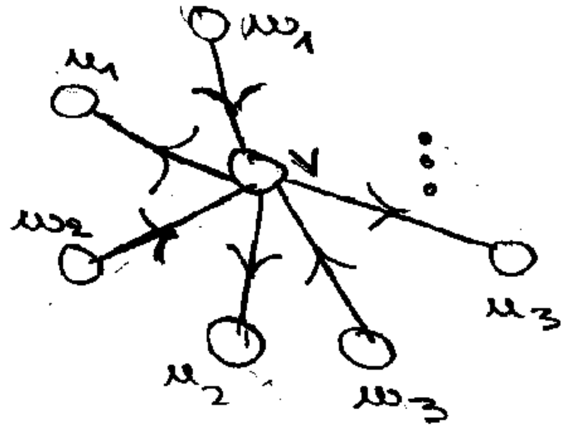
„$\Rightarrow$" Sei also $G = (V, E)$, $|V| = n$, $|E| = m$.

Sei

$$\mathcal{R} = (v_1, v_2, v_3, \ldots, v_m, v_1)$$

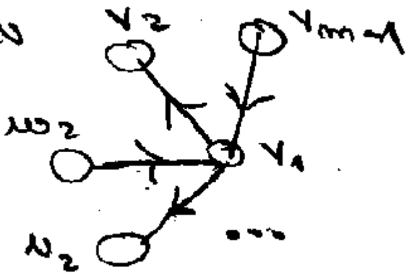
ein Eulerscher Kreis von G. Betrachten wir einen beliebigen Knoten $v \neq v_1$ der etwa k-mal auf $\mathcal{R}$ vorkommt, dann haben wir eine Situation

wie



alle u_i, w_i verschieden, also $\operatorname{grad}(v) = 2k$.

Für $v = v_1$ haben wir



und $\operatorname{grad}(v_1)$ ist gerade.

„⇐“ Das folgende ist eine Schlüssel-beobachtung für Graphen, die zshg. 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:

$$W = v_1 - v_2 - v_3 - v_4 - v_5 - v_6 \ldots v_1$$

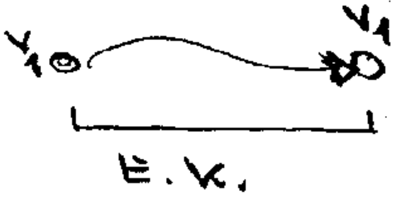
Ist etwa $v_3 = v_5 = u$, so hat u mindestens 3, also ≥ 4 Nachbarn. Wir können also von v_5 wieder durch eine neue Kante weggehen. Erst, wenn wir bei v_1 gelandet sind ist das nicht mehr sicher, und wir machen dort Schluß.

Nehmen wir nun W aus G heraus, haben alle Knoten wieder gerade Grad und wir können mit den Startknoten $v_1, v_2, \ldots$ so weitermachen und am Ende alles zu einem E.K. 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 E.K

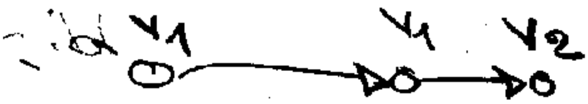
2. Nach Ind.-Vor haben wir E.K. auf dem Stück, das jetzt von v_1 erreichbar ist. Schreibe

diesen E.k. hin. Wir bekommen



3. Erweitere den E.k. auf

W ze[illegible] [illegible] von v_1. Das



Mache von v_2 dasselbe wie bei v_1:

So geht es weiter bis zum Ende von W. $\square$

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

(10.27)

Eine weitere Möglichkeit die kombinatorische Suche zu gestalten, ist das Prinzip der lokalen Suche.

Beim aussagenlogischen Erfüllbarkeitsproblem für KNF gestaltet es sich etwa folgendermaßen:

<u>Algorithmus</u> (Lokale Suche bei KNF)

Eingabe: F in KNF.

1. Wähle eine Belegung $a = (a_1, \ldots, a_n)$ der Variablen.

2. If $a(F) = 1$ return „$a(F) = 1$"

3. Wähle Klausel C von F mit $a(C) = 0$

10.12.

4. Ändere a so, daß $a(C) = 1$ ist,
indem ein (oder mehrere) Werte
von a geändert werden.

5. Mache bei 1. weiter. $\square$

Wie Horn-Erfüllbarkeit erkennen?

Eine Modifikation: 2 lokale
Suchen, von

$$a = (0, \dots, 0) \text{ und } b = (1, \dots, 1).$$

Nun gilt: Jede Belegung
unterscheidet sich auf $\leq \lceil \frac{n}{2} \rceil$
Positionen von a oder von b.

$\leq \lceil \frac{n}{2} \rceil$ Einsen $\qquad \geq \lceil \frac{n}{2} \rceil$ Einsen

(10.88)

__Bezeichnung:__ Für 2 Belegungen

$a = (a_1, \dots, a_m)$, $b = (b_1, \dots, b_m)$ ist

$$D(a,b) = |\{i \mid 1 \le i \le m, a_i \ne b_i\}|,$$

also die # Positionen, auf denen a und b verschieden sind, die Distanz von a und b. □

Für F und Belegung a gilt:

Es gibt b mit $b(F) = 1$ und $D(a,b) \le k$

$\iff$

- $a(F) = 1$ oder
- Für jedes $C = l_1 \vee \dots \vee l_m \in F$ mit $a(C) = 0$ gibt es ein l_j so daß für $a'(l_j) = 0$ und a' sonst wie a gilt: $D(b, a) \le k-1$.

Mit diesem Prinzip rekursiv

für 3-KNF F:

a, k

$a(l_1 \vee l_2 \vee l_3) = 0, \; l_1 \vee l_2 \vee l_3 \in F$

$l_1 = 1$ $\qquad$ $l_2 = 1$ $\qquad$ $l_3 = 1$

$a|_{l_1=1}, k-1$ $\qquad$ $a|_{l_2=1}, k-1$ $\qquad$ $a|_{l_3=1}, k-1$

$a(l_1' \vee \neg l_2 \vee l_3') = 0$

$a|_{l_2=1, l_1'=1}$ $\qquad$ $a|_{l_2=1, \neg l_2=1}$ $\qquad$ $a|_{l_2=1, l_3'=1}$

$k-2$ $\qquad$ $k-2$ $\qquad$ $k-2$

Es kann hin und her gehen

$\vdots$

Tiefe k

Zeit $O(n \cdot 3^k)$

10.101

Einmal rekursiv mit

$$F,\ a=(0-0),\ \left\lceil \frac{n}{2} \right\rceil$$

noch einmal mit

$$F,\ b=(1,-,1),\ \left\lceil \frac{n}{2} \right\rceil$$ Wegen 3-KNF!

Dann Zeit $O(n \cdot 3^{\frac{n}{2}}) = O(\underbrace{1{,}7}_{<2}\ldots^{n})$!

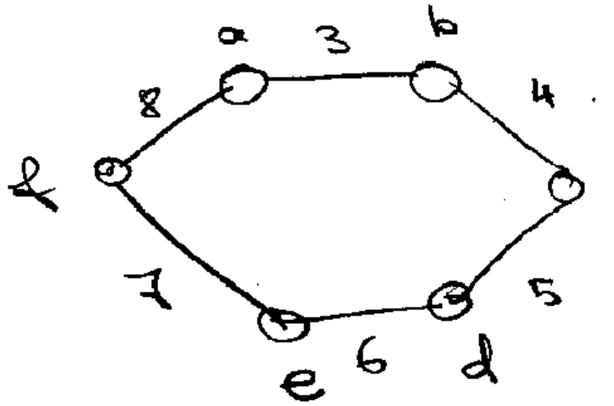
Auf die Art bis zu $O(1{,}5^n)$.

Die lokale Suche basiert auf einem Distanzbegriff zwischen möglichen Lösungen. Bei Belegungen ist dieser klar gegeben. Wie bei Rundreisen bzw. das TSP ? Wir betrachten

10.102

hier nur der Falle, daß der
zu Grunde liegende Graph unge-
richtet ist. !!

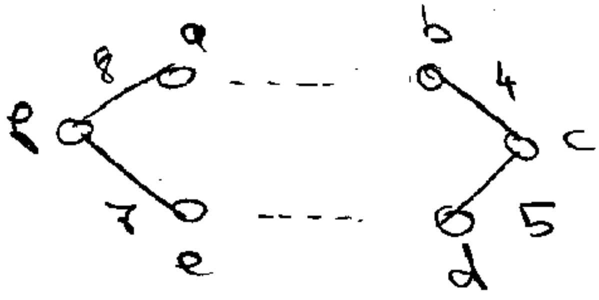
Haben wir also eine Rundreise
wie



Kosten 33

gegeben. Durch Ändern einer
Kante bekommen wir keine
neue Rundreise hin. An zwei
benachbarten Kanten können
wir auch nichts ändern. Löschen
wir einmal 2 nicht benachbarte

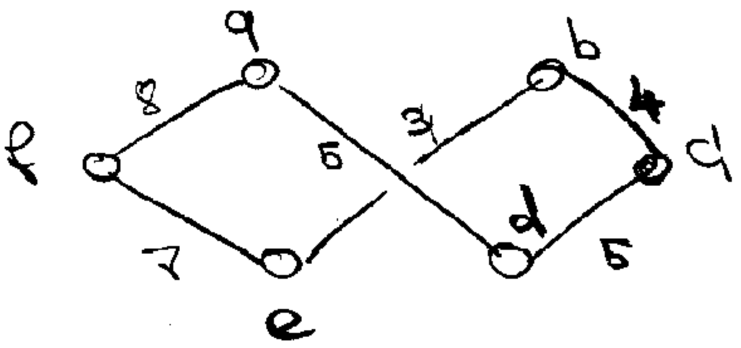
Kanten: $a - b$, $e - d$.



Wie können wir eine neue Rundreise zusammensetzen?

$d - e$ (und $a - b$) gibt ~~die~~ Alte.

Also $d - a$ (und $b - e$) : Das gibt



Kosten 32.

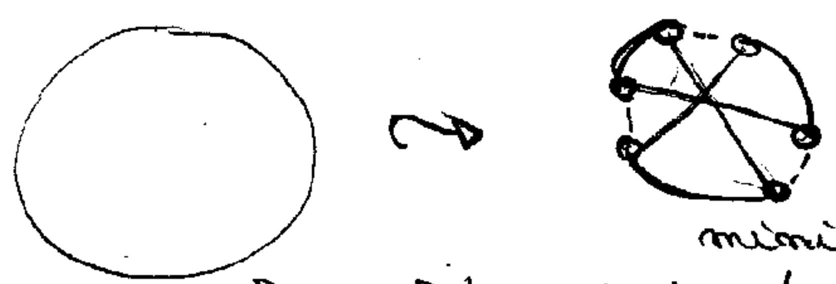
10.107

Man kann in $O(n^2)$ Schritten testen, ob es so zu einer Verbesserung kommt. Ein solcher Verbesserungsschritt wird dann gemacht.
Aber es gilt (leider) nicht:
Keine Verbesserung möglich

$\Leftrightarrow$

minimale Rundreise gefunden.

Allgemein kann man auch 3 Kanten löschen und den Rest zusammenbauen



Rundreise
minimale

Findet auch nicht immer unbedingt eine

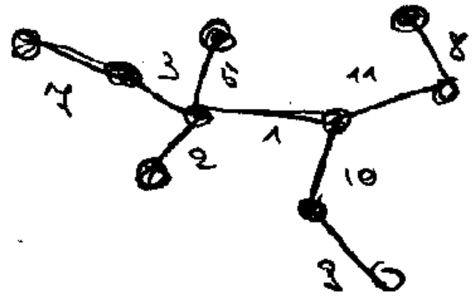
Bemerkung

- Für alle die Probleme, für die wir keine poly. Zeit Algorithmen angegeben haben, sind auch keine bekannt.
- Das bedeutet für diese Probleme kann das weitgehend blinde Ausprobieren von (expo.) vielen Lösungsmöglichkeiten nicht vermieden werden.
- Es ist kein Beweis bekannt, dß es nicht doch in poly. Zeit geht. (Wird aber nicht erwartet)
- Die hier betrachteten expo. Zeit Probleme sind ineinander übersetzbar. (Th Inf. II)

Die lokale Suche erlaubt es dagegen in poly. Zeit einen min. Sb zu finden. Erinnerung: Können min. Sb. in $O(|E| \log |V|)$ finden. (sogar $O(|E| \log^* |V|)$ wenn E sortiert nach den Kosten.)

Wir definieren zunächst den grundlegenden Transformations-schritt der lokalen Suche beim min. Sb.:

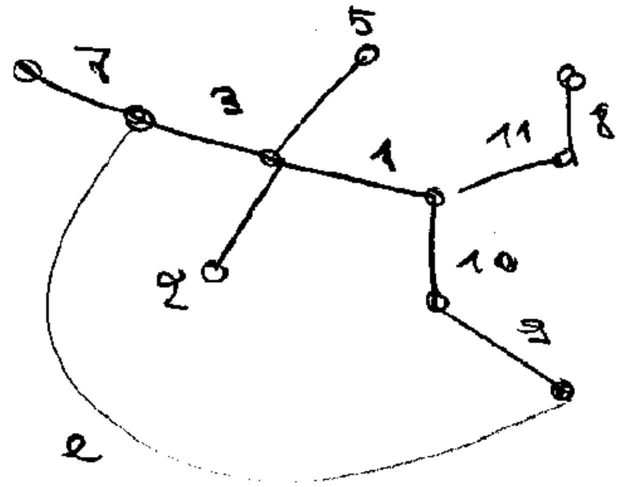
T



Spannbaum

(10.107)

Wähle eine Kante e nicht im Baum:



Diese Kante induziert genau einen einfachen Kreis mit dem Spannbaum.

Wir prüfen für jede Kante auf diesem Kreis, ob ihre Kosten $\gneq$ die Kosten von e sind.
Haben wir eine solche Kante löschen wir sie und bekommen einen SB mit geringeren Kosten.

(10.10)

Satz

Haben wir einen nicht-minimalen Spannbaum, so gibt es immer eine Kante e, mit der obere Tangentialebenen-Transformationsschritt zu einer Verbesserung führt.

Beweis

Sei also T ein Spannbaum, nicht minimal, sei S ein minimaler Spannbaum, dann $S \neq T$. Wir betrachten einmal die Kantenmenge

$$S \setminus T = \{e_1, \dots, e_k\}.$$

(10.108)

Falls der oben angegebene Transformationsschritt mit T und e_1, T und e_2, ..., T und e_k jedesmal zu keiner Verringerung der Kosten führt, transformieren wir so:

$T_1 := T \cup \{e_1\}$

$R_1 := T_1 \setminus \{$ Kante nicht aus f auf Kreis durch $e_1 \}$.
// keine Verbesserung

$T_2 := R_1 \cup \{e_2\}$ // nach Annahme

$R_1 := T_2 \setminus \{$ Kante nicht aus f auf Kreis durch $e_2 \}$
// keine Verbesserung,
// da e_1 nicht kleiner als
// alle Kanten auf
// dem vorherigen Kreis.

⋮

$T_k := R_{k-1} \cup \{e_k\}$

$R_k := T_k \setminus \{$ Kante nicht aus F auf Kreis durch $e_k\}$

// keine Verbesserung,
// da vorher keine Verbesserungen.

Also es sind die Kosten von T minimal im Widerspruch zur Annahme. $\square$

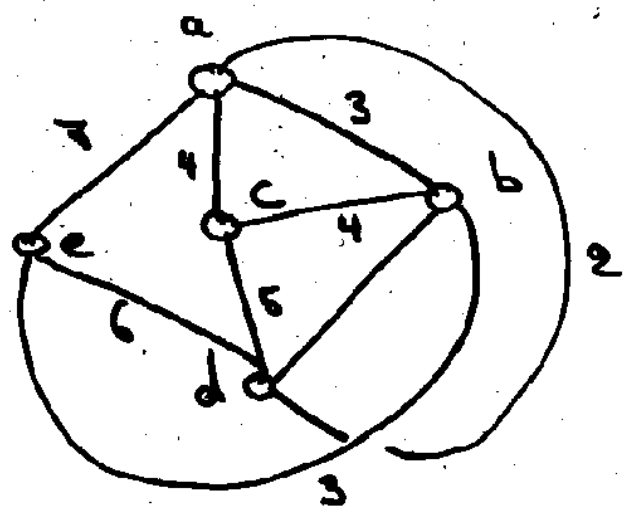
Laufzeit eines Algorithmus mit dem Transformationsschritt:

- Maximal $|E|^2$ Schritte.
- Pro Schritt $|E|$ Kanten ausprobieren
- Pro Kante $O(|V|+|E|)$

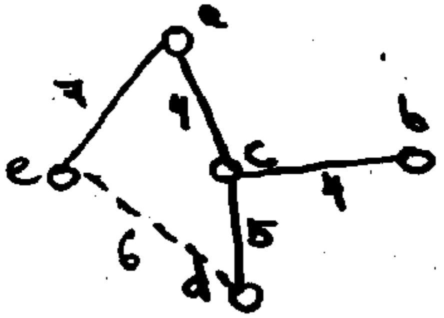
Insgesamt $O(|E|^4 + |V|)$

BEISPIEL

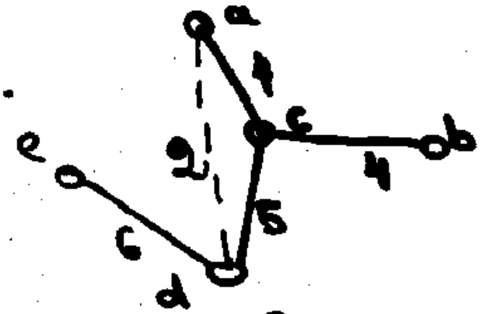
G



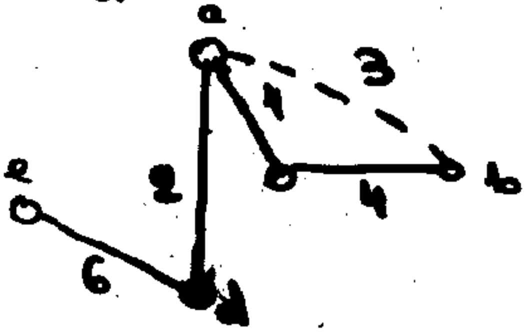
1.



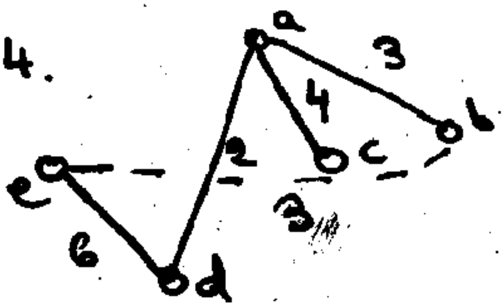
2.



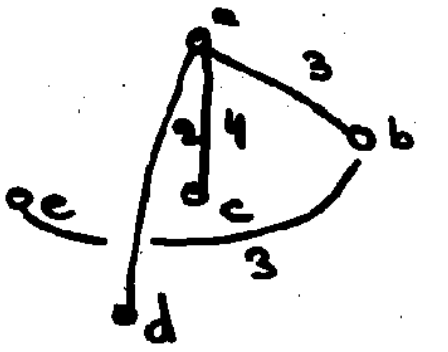
3.



4.



5.



KOSTEN 12