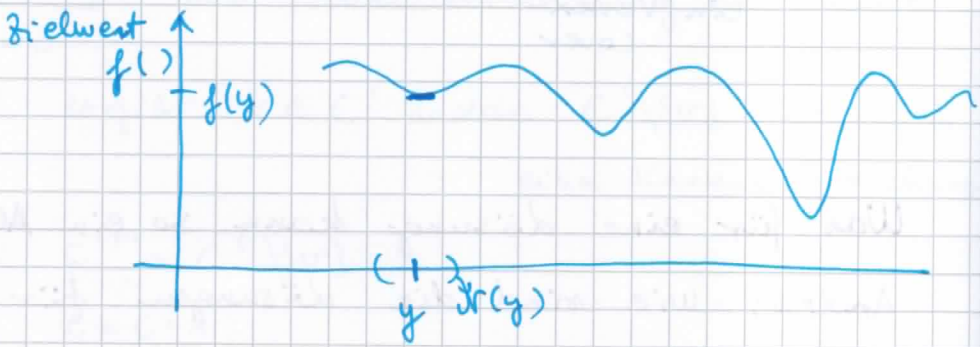
Eine solche Lösung heißt lokal optimal (weil es keine bessere in ihrer Umgebung (Nachbarschaft) gibt).

Eine lokal optimale Lösung kann, je nach Problem, viel schlechter sein als eine (global) optimale Lösung.

Abstrakte Darstellung einer lokal optimalen Lösung eines Minimierungsproblems:



Definition: Lokale Suche (allgemein)

Sei ein Minimierungsproblem zu lösen, wobei zu jeder Lösung y eine Umgebung $N(y)$ benachbarter Lösungen definiert ist.

Stiefe lokale Suche

→ sei y^0 eine Lösung für Instanz x

$i=0$

→ WHILE y^i nicht lokal optimal DO

- bestimme ein $y \in N(y^i)$ so dass

$$f(y) < f(y^i)$$

- setze $y^{i+1} = y$ und $i := i+1$

(Eine Lösung y ist lokal optimal, falls es in $N(y)$ keine Lösung mit kleinerem Zielwert gibt.)

Die Definition ist analog für Maximierungsprobleme.

Wie viele Verbesserungsschritte werden gebraucht bis (wenigstens) eine lokal optimale Lösung gefunden wird?

Im VERTEX COVER Beispiel → maximal n-1 Verbesserungen (wir hatten eigentlich einen Greedy Algorithmus)

bei den meisten (echten) Algorithmen mit lokaler Suche reichen polynomiell viele Schritte allgemein nicht aus, um in einem lokalen Optimum anzukommen!

c.) Unterschiede zu Greedy Algorithmen

(Lokale Suchverfahren sind ähnlich zu Greedy Algorithmen da die Lösung in jedem Schritt lokal verändert wird)

Unterschiede sind:

→ In einem Greedy Verfahren wird eine Lösung aus Teillösungen iterativ aufgebaut (es wird meistens nicht mit einer kompletten Lösung angefangen).

→ es wird immer eine beste Nachbarlösung ausgewählt

→ Tendenziell: die Anzahl der Schritte entspricht der Anzahl der Elemente in der Eingabe oder in der Lösung ⇒ schneller Algorithmus

→ In einer **Lokalen Suche** hat man in jedem Schritt eine komplette Lösung

→ die Suche wird nicht unbedingt mit dem besten, sondem nur mit einem besseren Nachbarn fortgesetzt.

→ Da die Anzahl aller Lösungen exponentiell ist, wird nicht garantiert dass man „schnell" ein lokales Optimum findet.

(Falls die Anzahl der möglichen Werte von f klein ist (wie beim VERTEX COVER), ist die strikte lokale Suche schnell fertig; sonst nicht unbedingt (wie bei k-MEDIAN))

→ wegen der großen Anzahl möglicher Lösungen wird eine Anfangslösung häufig durch Heuristiken oder randomisiert ausgewählt.

Beispiele für Nachbarschaften

→ Wir hatten die Nachbarschaften im k-MEDIAN Problem so gewählt:

Eine Menge $M \subset K$ von Zentren sei mit einer anderen Menge $M' \subset K$ von Zentren benachbart, wenn

$$M' = (M \setminus \{w\}) \cup \{u\}$$

d.h. irgendein Zentrum w wird durch einen Punkt $u \in K$ ersetzt.

→ Im VERTEX COVER Beispiel waren C und C' benachbart genau dann wenn $C' = C \cup \{v\}$ oder $C' = C \setminus \{v\}$ für irgendein $v \in V$

Definition: k-Flip Nachbarschaft. Wenn jede Lösung y einem n-dimensionalen Vektor über $\{0,1\}$ entspricht (also $\forall y \in \{0,1\}^n$), dann ist die k-Flip Nachbarschaft einer Lösung y

$$N_k(y) = \{ y' \in \{0,1\}^n \mid y' \text{ Lösung, und Hamming Distanz}(y,y') \leq k \}$$

Im Fall von k-MEDIAN bzw. VERTEX COVER:
Für welche k sind die oben definierten Nachbarschaften k-Flip Nachbarschaften und warum?

Im Folgenden untersuchen wir noch diese Fragen:

d.) Wie gut approximiert ein lokales Optimum das globale Optimum?

e.) Können wir zumindest ein lokales Optimum effizient berechnen, bzw. wann/warum sind wir ziemlich sicher, dass dies (für ein gegebenes Problem) nicht geht?

f.) Wenn ein lokales Optimum nicht effizient berechenbar ist, was ist zu tun? Kann man approximieren, und in welchem Sinne?

g.) Kann es hilfreich sein, während einer lokalen Suche manchmal Verschlechterungen zuzulassen?

LS 14.

d.) Wie gut approximiert eine lokal Optimale Lösung?

Es hängt vom Problem ab.

Wir haben gesehen: für VERTEX-COVER -> gar nicht

für k-MEDIAN -> ein lokales Optimum ist 5-approximativ

Im Folgenden analysieren wir den Approximationsfaktor eines lokalen Maximum im Fall eines Maximierungsproblems:

max-Leaf-SPANNING TREE (= Connected-DOMINATING-SET)

(Anwendungen in Broadkasting)

Eingabe: ein (ungerichteter, ungewichteter) Graph

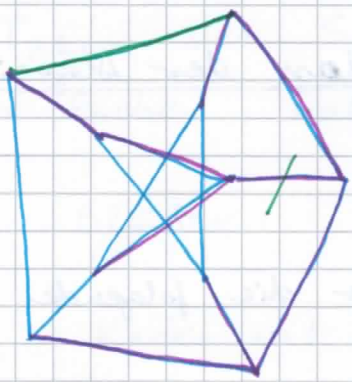
$$G(V, E)$$

Ausgabe: ein Spannbaum mit grösstmöglicher Blattzahl

Die Lösungen sind also Spannbäume (= Kantenmengen). Wann sollte ein Spannbaum eine Nachbarlösung eines anderen Spannbaums sein? -> Wir definieren jetzt Nachbarschaften gemäß 2-Flip Nachbarschaft.

Definition: Für einen gegebenen Eingabegraphen $G(V, E)$ seien zwei Spannbäume $T'(V, E')$ und $T''(V, E'')$ benachbarte Lösungen falls $E'' = (E' \setminus \{e\}) \cup \{f\}$ für zwei Kanten e und f.

Höchstens wie viele Nachbarn hat ein Spannbaum?



die Hinzunahme einer beliebigen Kante schliesst einen Kreis; entfernen wir eine beliebige andere Kante des Kreises, erhalten wir einen benachbarten Spannbaum.

das ergibt $O(m \cdot n)$ Nachbarn für jeden Spannbaum.

Lokale Suche für max-Leaf-SPANNING-TREE

- sei $G(V,E)$ der Eingabegraph, und $T(V,E')$ ein beliebiger Spannbaum
- WHILE es einen Nachbar-Spannbaum $T' \in \mathcal{N}(T)$ mit mehr Blättern als in T gibt, DO
  - ersetze T durch T'
- gib T aus

Wieviele Verbesserungsschritte sind möglich? $(n-3)$

Die Laufzeit ist in diesem Fall also polynomiell:

Jeder Baum hat $O(m \cdot n)$ Nachbarn zu inspizieren, und die WHILE-Schleife wird maximal $n-3$ mal durchlaufen $\longrightarrow$ Laufzeit $O(m \cdot n^2)$

Ein lokal optimaler Spannbaum ist 5-approximativ:

Theorem: Ein lokal optimaler Spannbaum hat mindestens $\frac{1}{5}$-mal so viele Blätter wie ein optimaler Spannbaum.

LS 16.

Beweis:

Hier zeigen wir nur, dass der Algorithmus 1/2-approximativ ist:

Für den Beweis brauchen wir die folgende Behauptung:

Behauptung: Jeder Baum hat mehr Blätter als Knoten mit Grad ≥ 3.

Beweis der Behauptung: Sei ein Baum T fixiert. und bezeichne k_d die Anzahl seiner Knoten mit Grad d

Die Anzahl aller Knoten ist

$$n = k_1 + k_2 + k_3 + \dots + k_{d_{max}} = k_1 + k_2 + \sum_{d \geq 3} k_d$$

$$2 \cdot \Big| \quad 2n = 2k_1 + 2k_2 + 2\sum_{d \geq 3} k_d \qquad (*)$$

Die Summe aller Knotengrade ist

$$2(n-1) = k_1 + 2 \cdot k_2 + 3 \cdot k_3 + 4 \cdot k_4 + \dots \geq k_1 + 2k_2 + 3\sum_{d \geq 3} k_d$$

$$2n \geq k_1 + 2 \cdot k_2 + 3 \cdot \sum_{d \geq 3} k_d + 2 \qquad (**)$$

aus $(*)$ und $(**)$ folgt:

$$2k_1 + 2k_2 + 2\sum_{d \geq 3} k_d \geq k_1 + 2k_2 + 3\sum_{d \geq 3} k_d + 2$$

$$k_1 > \sum_{d \geq 3} k_d$$

□ Behauptung.

→ Also, T_{ALG}, der Spannbaum gefunden mit lokaler Suche, hat n Knoten, davon sind

k_1 Blätter

k_2 Knoten mit Grad 2

$\leq k_1$ Knoten mit Grad ≥ 3

(Knotengrad bezieht sich auf den Baum T_{ALG})

→ Wieviele Knoten in den einzelnen Gruppen sind Blätter in T_{OPT}, d.h. im Spannbaum mit den meisten Blättern?

Wir zeigen, dass unter den k_2 Knoten mit Grad 2 in T_{ALG}, höchstens $8k_1$ sind Blätter in T_{OPT}.

→ Dann hat T_{OPT} insgesamt maximal

$$k_1 + 8k_1 + k_1 = 10k_1$$

Blätter in allen drei Sorten von Knoten, und der 10-Approximation wird Bewiesen

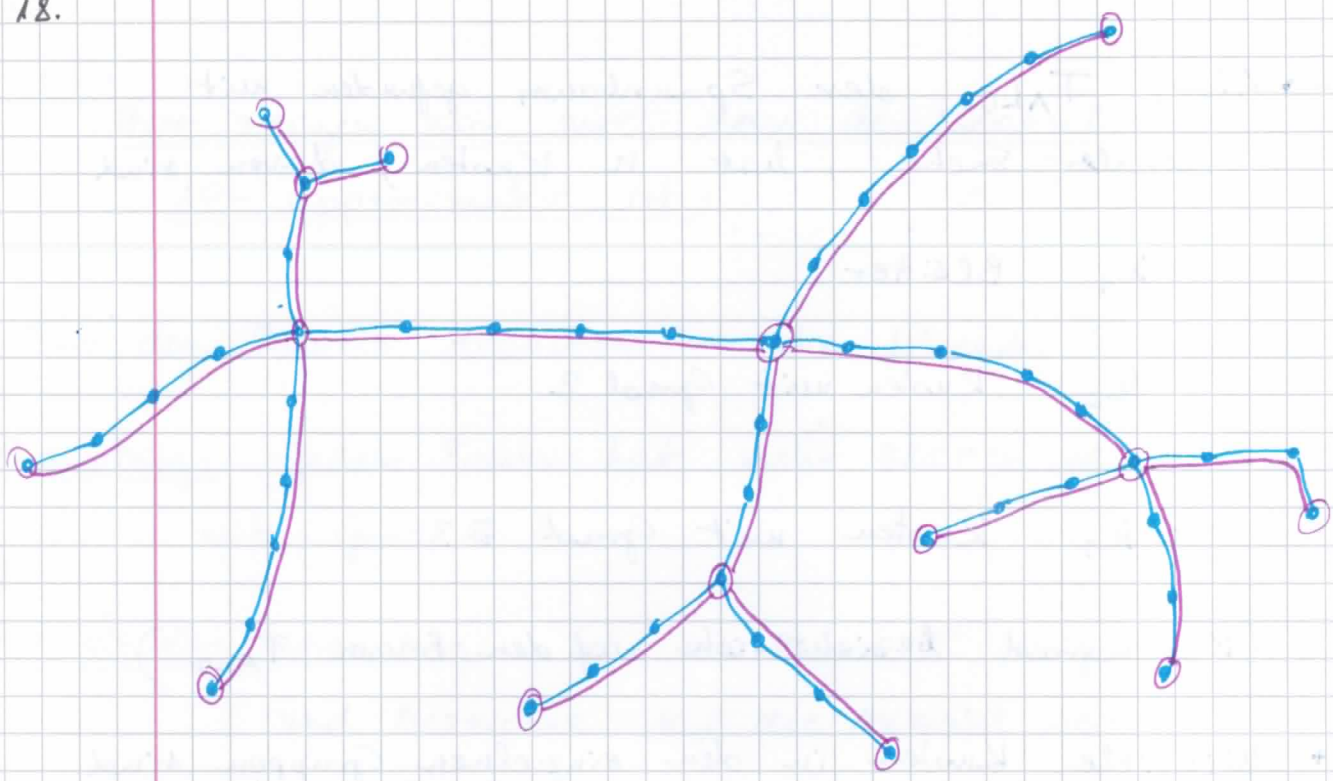
Behauptung: T_{OPT} hat höchstens $8k_1$ Blätter, die in T_{ALG} Knoten mit Grad 2 sind.

Beweis der Behauptung:

Wir betrachten die maximalen 2-Wege in T_{ALG}.

Definition: Ein maximaler 2-Weg durchläuft nur Knoten von Grad 2 als innere Knoten, und endet in einem Knoten mit höherem Grad, oder in einem Blatt.

LS 18.



Höchstens wieviele 2-Wege gibt es?

Wir stellen uns jeden 2-Weg als eine lange Kante vor; wir erhalten somit einen Baum mit höchstens $2k_1$ Knoten (die Blätter und die Knoten mit Grad ≥ 3)

$\Rightarrow$ es gibt maximal $2k_1 - 1$ 2-Wege

Teilbehauptung: Jeder 2-Weg enthält maximal 4 innere Knoten, die in T_{OPT} Blätter sind.

Korollar: T_{ALG} hat maximal $2k_1 \cdot 4$ Knoten mit Grad 2 die in T_{OPT} Blätter sind.

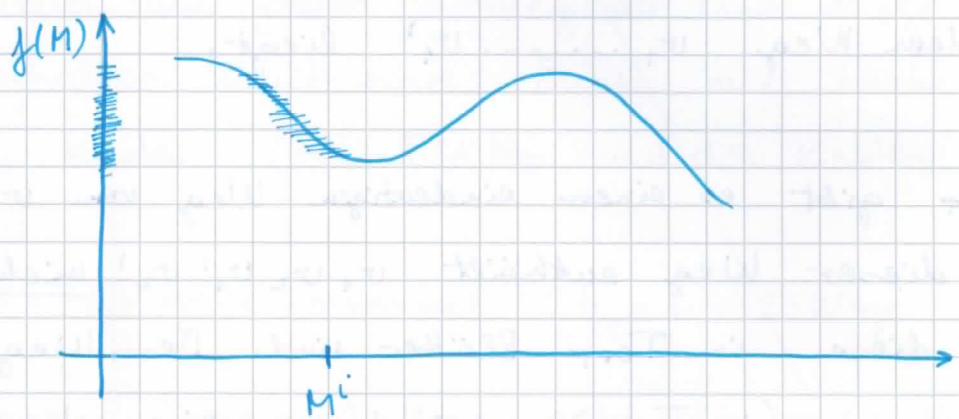
□

LS 20.

e.) Komplexität der Berechnung von lokalen Optima

→ Wir haben gesehen, dass eine beliebige lokal minimale Lösung für das k-MEDIAN Problem 5-approximativ ist, aber: wir hatten überhaupt keine Laufzeit-Garantie für die lokale Suche für k-MEDIAN.

Die Anzahl der möglichen verschiedenen Distanz-Summen (also Zielwerte im k-MEDIAN) kann exponentiell sein, so kann es vorkommen, dass eine lokale Suche exponentiell-viele Verbesserungsschritte braucht um in einem lokalen Minimum anzukommen.



→ Es gibt tatsächlich Probleme, für die die effiziente Berechnung eines lokalen Optimums allgemein (sehr wahrscheinlich) unmöglich ist.

→ Wie kann man sowas beweisen, dass gar kein effizienter Algorithmus (für ein gegebenes Problem) existieren kann? Solche Aussagen sind schwierig zu beweisen. (so schwierig wie $P \neq NP$ zu beweisen!)

→ Ähnlich zu Problemen der Problemklasse NP, ~~zeigen wir~~ zeigt man nun, dass es unter den „lokalen Suchproblemen" einige schwierigste Probleme gibt. Zunächst braucht man aber eine präzisere Definition für die Klasse der lokalen Suchprobleme. Es handelt sich ja um bestimmte Optimierungsprobleme:

(Komplexitätstheorie:)

Definition: Ein polynomielles Suchproblem

(Opt, f, L, A, B) ist ein NP-Optimierungsproblem mit einer Nachbarschaftsrelation über allen Lösungen einer beliebigen Probleminstanz, mit den zusätzlichen Eigenschaften:

- es gibt einen polynomiellen Algorithmus A, der für jede Instanz I eine Anfangslösung y_0 berechnet;
- es gibt einen polynomiellen Algorithmus B, der für jede Instanz I und jede Lösung y entscheidet, ob y ein lokales Optimum ist; falls nicht, dann bestimmt B eine Nachbarlösung mit besserem Zielwert, also eine $y' \in N_I(y)$ so dass

$$f(y') < f(y) \quad \text{falls } opt = min$$

bzw. $$f(y') > f(y) \quad \text{für } opt = max$$

Def: PLS ⊂ NPO bezeichnet die Klasse aller polynomiellen Suchprobleme.
(↑ 'Polynomial local search')

LS 22.

Was verlangt diese Definition von dem Suchproblem?

→ Dass eine Anfangslösung y und jeweils (für jede y) ggf. eine bessere Nachbarlösung y' effizient berechnet werden kann.

Die zweite ist offensichtlich erfüllt, falls zB. jede Nachbarschaft nur polynomiell viele Lösungen enthält (wie in unseren Beispielen).

Beachte, dass ohne diese natürlichen Forderungen eine effiziente lokale Suche ab ovo nicht vorstellbar wäre. (zumindest nicht mit Laufzeitgarantie)

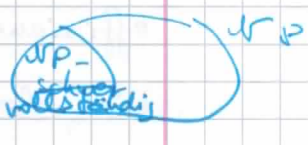
→ der bekannte Algorithmus mit lokaler Suche, lässt sich dann so zusammenfassen (allgemein):

der Standardalgorithmus für ein polynomielles Suchproblem:

- bezeichne I die Eingabe
- berechne $y = A(I)$
- WHILE y kein lokales Optimum, $(B(I,y) \neq \emptyset)$ DO
  setze $y := B(I,y)$

Beachte: Da A() und B() effizient berechenbar sind, entscheidet die Anzahl der Schleifendurchläufe (Verbesserungsschritte) die Laufzeit!

→ Wie könnte man jetzt schwierige Probleme in der Klasse PLS definieren? Wie hat man dies im Fall der Klasse NP gemacht?



→ Man könnte zeigen, dass alle Probleme in PLS, auf das gegebene (schwierige) Problem reduziert werden können, so dass falls dieses effizient (lokal) optimierbar sein sollte, dann alle Probleme in PLS effizient (lokal) optimierbar wären.

→ Was bedeutet aber in diesem Fall „reduzieren"? Wir haben keine JA/NEIN Instanzen, sondern wir suchen lokal optimale Lösungen. Das bekannte Konzept der Reduktion ~~reduktion~~ eignet sich nicht gänzlich, wir brauchen ein neues:

Definition: Seien P_1 und P_2 polynomielle Suchprobleme.

P_1 ist PLS-~~reduzierbar~~ auf P_2 ($P_1 \leq_{PLS} P_2$) wenn es polynomielle Transformationen Φ und Ψ gibt, so dass

- wenn I eine Instanz von P_1 ist, dann ist $\Phi(I)$ eine Instanz von P_2

- wenn y (die Lösung) ein lokales Optimum für P_2 ist, dann ist $\Psi(I, y)$ (die Lösung) ein lokales Optimum für P_1

Definition: Ein Problem $P \in PLS$ ist PLS-vollständig wenn $Q \leq_{PLS} P$ für jedes $Q \in PLS$ gilt. (d.h. alle poly. Suchprobleme auf P reduzierbar sind).

$\Longrightarrow$ Sei P ein PLS-vollständiges Suchproblem.

Wenn ein lokales Optimum für jede Instanz von P effizient berechnet werden könnte, dann könnte ein lokales Optimum für ein beliebiges polynomielles Suchproblem effizient berechnet werden.

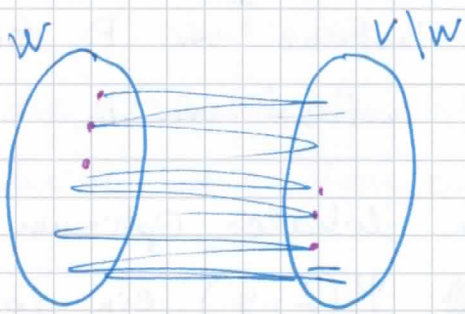
(Dass dies der Fall ist, ist ebenso sehr unwahrscheinlich wie $P = NP$. Es würde zB. auch unter Anderem, $NP = co\text{-}NP$ implizieren ($P = NP$ aber nicht))

Beispiele für PLS-vollständige Probleme

1. minimum-BALANCED-CUT

Eingabe: ein ungerichteter Graph $G(V, E)$ mit Kantengewichtung $w: E \to \mathbb{R}_{\geq 0}$

Ausgabe: eine Knotenmenge $W \subset V$, $|W| = \frac{|V|}{2}$ so dass das Gesamtgewicht der kreuzenden Kanten minimal ist



(minimiere $f(W) = \sum_{\substack{e \text{ führt} \\ \text{zwischen } W \text{ und} \\ V \setminus W}} w_e$) (Anwendungen in VLSI-Entwurf)

Wie kann die Nachbarschaft $N(W)$ definiert werden?

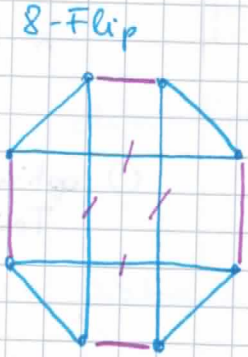
Die 2k-Flip Nachbarschaft ersetzt k Knoten aus W durch k Knoten aus $V \setminus W$.

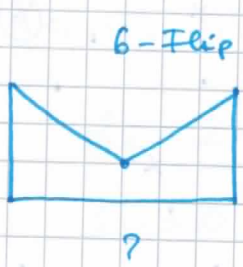
Thm: Mit der 2-Flip Nachbarschaft ist das entsprechende lokale Suchproblem für min-BALANCED-CUT PLS-vollständig.

2. Die besten Algorithmen für TSP in der Praxis sind Algorithmen mit lokaler Suche (kombiniert mit Branch and Bound bzw. Branch and Cut, siehe später).

→ die Lösungen sind die Kantenmengen, die einer Rundreise entsprechen

→ die lokale Suche fängt mit irgendeiner Rundreise an (gefunden mit Hilfe einer Heuristik), dann werden k Kanten aus der Rundreise entfernt und durch andere k Kanten ersetzt, so dass eine bessere Rundreise entsteht.





Thm: Es gibt eine (große) k so dass TSP mit der 2k-Flip Nachbarschaft PLS-vollständig ist.

f.) Die Approximierung von lokalen Optima

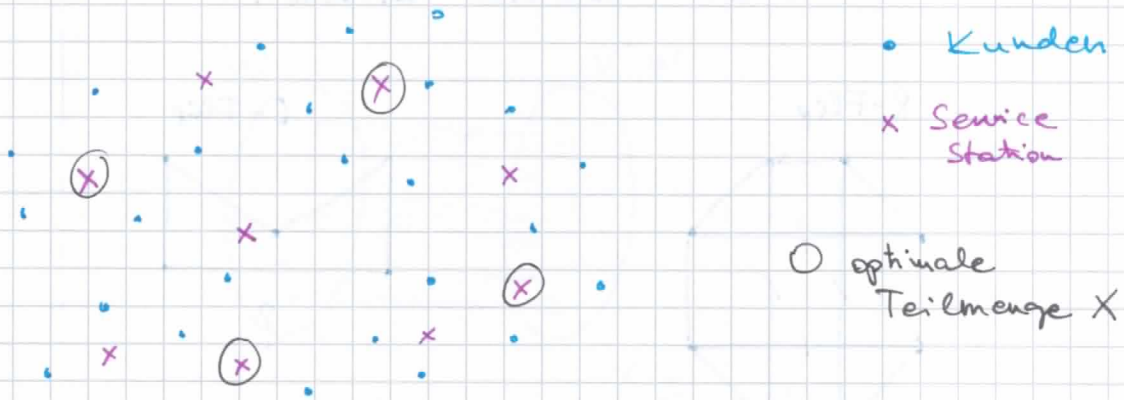
Beispiel: FACILITY LOCATION

Eingabe:
- eine Menge K (Kunden)
- eine Menge S (mögliche Service Stationen)
- eine Metrik $d()$ auf $K \cup S$ (Distanzwerte)
- für jede $s \in S$ die Betriebskosten f_s

Definition: Sei $X \subseteq S$ eine Teilmenge der Service Stationen, und $k \in K$ ein Kunde. Die Anschlusskosten des Kunden k an X sind $dist(k, X) = \min_{s \in X} d(k, s)$.
(durch die Distanz des k von X definiert)

Ausgabe: Eine Teilmenge $X \subseteq S$ von Service-Stationen so dass die Summe aller Anschluss- und Betriebskosten minimal ist.

$$\left(\text{minimiere } C(X) = \sum_{k \in K} dist(k, X) + \sum_{s \in X} f_s\right)$$



Lokale Suche für FACILITY LOCATION

→ sei $X^0 \subseteq S$ eine beliebige Menge von Service Stationen

→ WHILE das Entfernen, Hinzufügen oder Ersetzung einer Service Station zu einer Verbesserung führt, führe eine beliebige solche Operation aus (sei X^i die aktuelle Lösung nach i Runden)

→ gib die lokal optimale Lösung X aus

Theorem: Die lokale Suche ist 3-approximativ für FACILITY-LOCATION, d.h. wenn X^* eine minimale Lösung ist, und X eine lokal minimale Lösung ist, dann

$$C(X) \leq 3 \cdot C(X^*)$$

und dieser Approximationsfaktor ist scharf.

Wir beweisen den Approximationsfaktor nicht, nur die untere Schranke:

Behauptung:
Es kann vorkommen, dass die obige lokale Suche eine $(3-\varepsilon)$-approximative Lösung resultiert, für beliebige $\varepsilon > 0$.
(i.e. dass ein lokal minimale Lösung $(3-\varepsilon)$-approximativ ist)

Beweis: sei $|K| = n \quad K = \{k_1 k_2 \dots k_n\}$

$$|S| = n+1 \quad S = \{s_0 s_1 \dots s_n\}$$

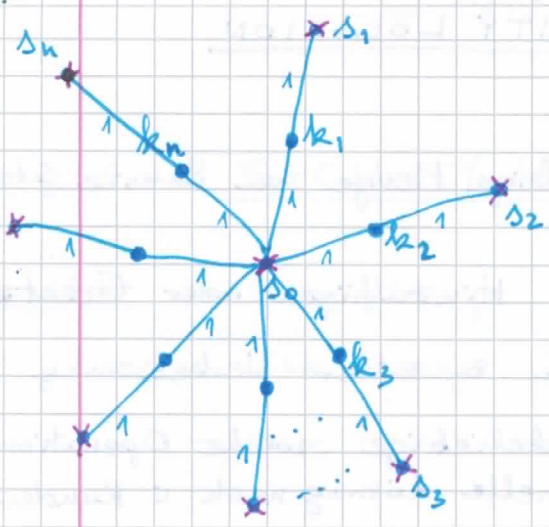
die Betriebskosten: $f_{s_0} = 2n-2$

$f_{s_i} = 0$ für $i \neq 0$

die Distanzwerte $d(\,)$ seien die Distanzen (kürzeste Weglänge) im folgenden Graphen:

(kürzeste Weglängen in einem Graphen entsprechen einer Metrik!)

LS 28.



$f_{s_0} = 2n-2$

$f_{s_i} = 0$

$d(k_i, s_i) = 1$

$d(k_i, s_0) = 1$

$d(k_i, s_j) = 3$ sonst

→ die optimale Lösung ist $X^* = \{s_1, s_2, \dots, s_n\}$ mit Kosten

$$C(X^*) = n + 0 = n$$

(n: Anschlusskosten, 0: Betriebskosten)

→ ein lokales Minimum ist $X = \{s_0\}$ mit Kosten

$$C(X) = n + 2n - 2 = 3n - 2$$

(n: Anschluss, $2n-2$: Betrieb)

Approxfaktor: $\frac{C(X)}{C(X^*)} = \frac{3n-2}{n} = 3 - \frac{2}{n}$ ist beliebig nah an 3 als $n \to \infty$

→ Warum ist $\{s_0\}$ lokal optimal?

die ~~möglichen~~ Nachbarlösungen X' sind: $C(X')$

$X' = \emptyset$ → ∞ Anschlusskosten

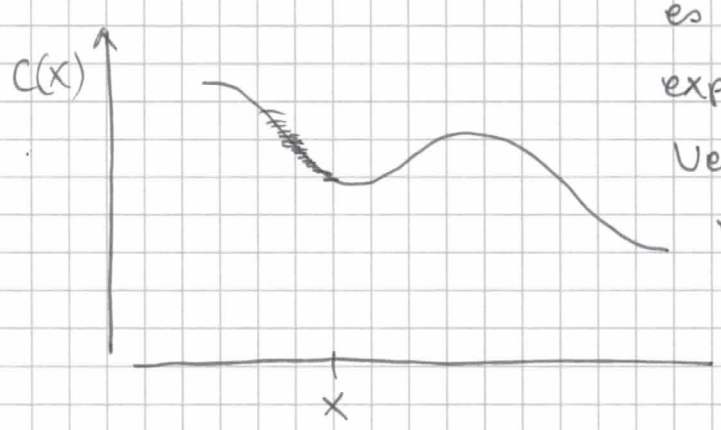
$X' = \{s_i\}$ → Anschluss $3(n-1)+1 = 3n-2$

$X' = \{s_0, s_i\}$ → $n + (2n-2) = 3n-2$

(n: Anschluss, $2n-2$: Betrieb)

⇒ kein Nachbar hat kleineren Zielwert

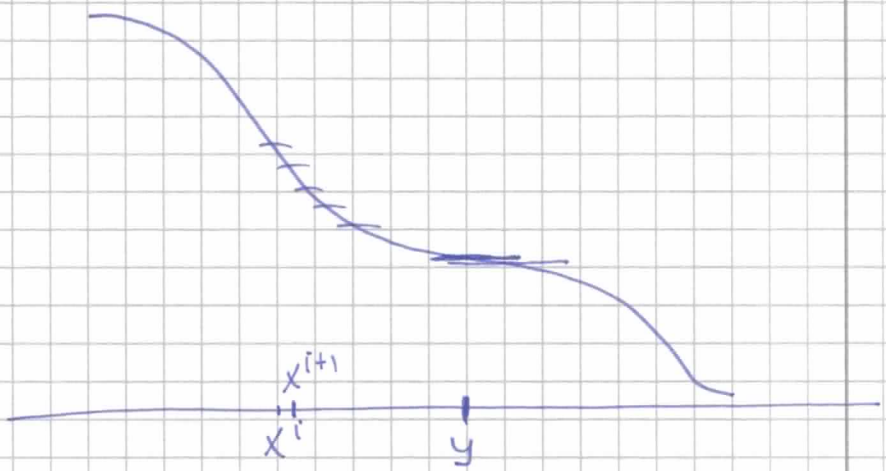
Problem: Die lokale Suche für FACILITY LOCATION ist nicht effizient!



es kann vorkommen:
exponentiell-viele kleine (WHILE-Runden) Verbesserungen führen zu exponentieller Laufzeit

→ Idee: Führen wir nur Verbesserungsschritte durch, die „groß genug" sind. (Wenn X^i keine Nachbarlösung hat mit „deutlich besserem" Zielwert, dann erklären wir X^i zu einem approximativen lokalen Optimum.)

→ Was bedeutet hier „groß genug"? Die Verbesserung messen wir relativ zum aktuellen Zielwert $C(X^i)$
- die Verbesserung soll $> \delta \cdot C(X^i)$ sein für eine gegebene δ
- äquivalent: $C(X^{i+1}) < (1-\delta) \cdot C(X^i)$



→ Wenn für eine Lösung Y keine Lösung mit hinreichend kleinerem Zielwert in seiner Nachbarschaft $N(Y)$ gibt, dann erklären wir Y zu einem approximativen lokalen Optimum:

Definition: Sei eine Instanz eines polynomiellen Suchproblems gegeben, und sei y eine Lösung mit Wert $f(y)$.
y ist ein *δ-approximatives lokales Minimum*, wenn

$$f(y') \geq (1-\delta) \cdot f(y)$$

für alle benachbarte Lösungen $y' \in N(y)$ gilt.
(Für Maximierungsprobleme analog.)

Achtung! y hat *nicht* unbedingt ein *lokales Minimum* in der Nachbarschaft $N(y)$ (mit Zielwert $\geq (1-\delta) f(y)$)
(Deshalb ist das Folgende nicht selbstverständlich)

Trotzdem gilt im Fall von FACILITY LOCATION Folgendes:

Theorem: Wenn im FACILITY LOCATION nur Verbesserungen akzeptiert werden, die den aktuellen Zielwert mindestens um Faktor $\left(1 - \frac{\varepsilon}{|S|}\right)$ reduzieren, (dann erhalten wir am Ende ein $\frac{\varepsilon}{|S|}$-approximatives lokales Minimum X),

und diese Lösung ist ein $(3+\varepsilon)$-approximatives globales Minimum. (folgt aus der hier nicht präsentierten Analyse der 3-Approximation)

Laufzeit: Da immer um mindestens um einen gegebenen Faktor verbessert wird, ist die Laufzeit logarithmisch im Zielwert, (und polynomiell in der Eingabelänge)

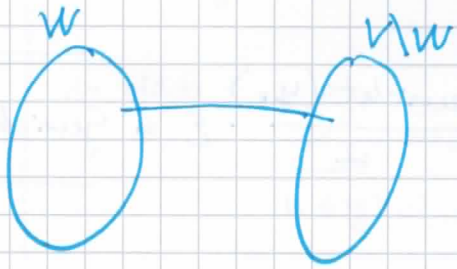
$$O\left(\frac{|S|}{\varepsilon} \cdot \ln \frac{C(x^0)}{C(x^*)}\right)$$

LS. 36.

g.) Verschlechterungen während der lokalen Suche

1. Der Kernighan-Lin Algorithmus für minimum-BALANCED-CUT („Einfrieren")

Zur Erinnerung: die Eingabe ist ein ungerichteter Graph $G(V, E)$ mit Kantengewichten, gesucht wird $W \subset V$ mit $|W| = \frac{|V|}{2}$ mit minimalem Gesamtgewicht kreuzender Kanten zwischen W und $V \setminus W$.



→ Der Zielwert einer Lösung W ist also

$$f(W) = \sum_{\substack{e \text{ führt} \\ \text{zwischen } W \\ \text{und } V \setminus W}} w_e$$

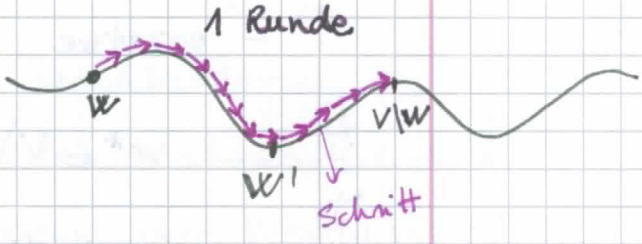
→ Wir betrachten die 2-Flip Nachbarschaft, also wenn zwei Knoten zwischen W und $V \setminus W$ getauscht werden, erhalten wir eine mit W benachbarte Lösung.

→ jede Lösung W entspricht einem Bitvektor $y \in \{0,1\}^n$ mit genau $\frac{n}{2}$ 1-Einträgen

$$y_i = 1 \iff v_i \in W$$

$$y_i = 0 \iff v_i \notin W$$

→ der Kernighan-Lin Algorithmus läuft in Runden
in jeder Runde wird ausgehend von der aktuellen Lösung W letztendlich eine bessere Lösung W' gefunden (so lange es geht), die aber mit W nicht direkt benachbart ist



→ die Beschreibung einer Runde:

- W' wird über mehreren Schritten gefunden, wobei die einzelnen Schritte Aufwärtsbewegungen zulassen
- jeder Schritt führt zu einer besten Nachbarlösung, auch wenn diese schlechter ist als die aktuelle Lösung
- in jedem Schritt werden die getauschten Knoten eingefroren: d.h. wir legen fest ob der Knoten $w \in W$ oder $w \notin W$, und ändern dies in dieser Runde nicht mehr
- das Einfrieren entspricht also dem Einfrieren von 0-1 Einträgen in dem Lösungsvektor y

(Warum soll man einfrieren?)

- am Ende der Runde wird die beste Lösung dieser Runde ausgewählt

LS 38.

sei W eine Anfangslösung $W = \frac{|V|}{2}$

REPEAT

- $W_1 = W$
- FOR $i = 1$ to $\frac{n}{2}$ DO
  - ersetze einen Knoten $w \in W_i$ durch $w^* \in V \setminus W_i$ so dass der Gewinn größtmöglich

    $Gewinn(w, w^*) = f(W_i) - f\left((W_i \setminus \{w\}) \cup \{w^*\}\right)$

    (auch bei negativem Gewinn)
  - friere w und w^* ein

    (sie werden während FOR nicht mehr geändert)
- sei W' die Lösung in $\{W_1, W_2, \ldots, W_{\frac{n}{2}}\}$ mit minimalem $f(W_i)$
- setze $W = W'$

UNTIL $W' = W_1$ → (es wurde keine bessere Lösung in der Runde gefunden)

eine Runde

ein Schritt

Diese lokale Suche in variabler Tiefe mit Einfrieren kann für andere Probleme verwendet werden wo der Lösungsraum Teil von $\{0,1\}^n$ ist (z.B. TSP mit k-Flip Nachbarschaft)

2. Der Metropolis Algorithmus

→ für ein Minimierungsproblem wo eine Nachbarschaft $N(y)$ für jede Lösung (einer gegebenen Instanz) definiert ist

→ In jeder Runde wird eine Nachbarlösung zufällig ausgewählt, und mit gewisser Wahrscheinlichkeit auch dann akzeptiert wenn ihr Wert schlechter (höher) ist.

→ Je höher der neue Wert, desto geringer die Akzeptanzwahrscheinlichkeit

- sei y eine Anfangslösung
- REPEAT „hinreichend oft"
  - wähle zufällig einen Nachbarn $y' \in N(y)$
  - IF $f(y') \leq f(y)$ $y := y'$
    ELSE $y := y'$ mit Wahrscheinlichkeit

$$\text{Prob} = e^{-\frac{(f(y') - f(y))}{T}}$$

T ist ein Parameter „Temperatur"

e ist die Euler-Zahl

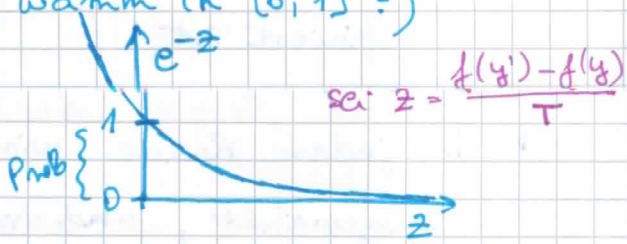
LS 40.

Die Akzeptanzwahrscheinlichkeit für schlechtere Nachbarn

Wir untersuchen diese Wahrscheinlichkeit

$$\text{Prob} = e^{-\frac{(f(y')-f(y))}{T}} = \frac{1}{e^{\frac{f(y')-f(y)}{T}}}$$

(wann groß, wann klein, wamm in $[0,1]$?)



→ sei $T > 0$ fixiert

wenn $f(y')-f(y)$ groß → viel schlechtere Nachbarlösung y'

$e^{\frac{f(y')-f(y)}{T}}$ auch groß

→ Prob klein

→ sei $f(y')$ fixiert

wenn T groß → $\frac{f(y')-f(y)}{T}$ klein

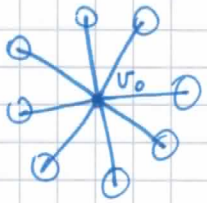
$e^{\frac{f(y')-f(y)}{T}} \approx 1$

⇒ Prob ≈ 1 (groß)

wenn T klein ⇒ Prob klein)

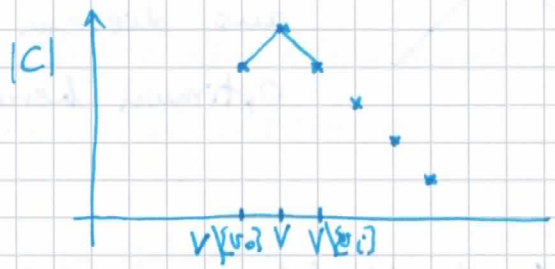
Beispiel: Metropolis Algorithmus auf dem Stemgraph für VERTEX COVER mit Anfangslösung V

Wäre zufällig das Zentrum als Erstes entfernt



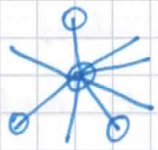
$y = V \setminus \{v_0\}$

→ irgendwann wird $y = V$ als zufälliger Nachbar wieder betrachtet, und (wenn T groß!) akzeptiert



danach werden wahrscheinlich andere Knoten (Satelliten) entfernt, und v_0 nie wieder entfernt (keine Knotenüberdeckung)

abhängig von T bleibt die Lösung bei ein Paar Satelliten



<u>Wie soll die Temperatur T gesetzt werden?</u>

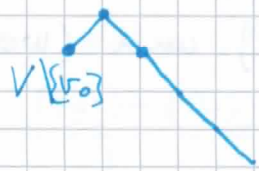
(falls $T >> f(y') - f(y) = 1$

dann die Akzeptanzwahrscheinlichkeit ist

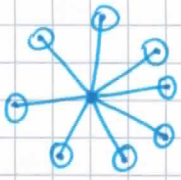
$$Prob = e^{-\frac{1}{T}} \approx 1$$

und falls die aktuelle Lösung wenige Satelliten enthält, wird wahrscheinlicher eine schlechtere Nachbar mit mehr Satelliten ausgewählt und weil $Prob \approx 1$, auch akzeptiert

=> wir sollten den Parameter T also klein stellen, damit in dieser Phase die schlechtere Lösungen mit hoher Wahrscheinlichkeit abgelehnt werden

→ dann aber 

wird es auch schwierig aus diesem lokalen Optimum herauszukommen



auch wenn $y = V$ als Nachbar ausprobiert wird, wegen zu kleiner T wird nicht akzeptiert!

Idee: Fangen wir mit hoher Temperatur an, und dann „kühlen wir T langsam aus": reduzieren wir T langsam, so dass schlechtere Lösungen nach einer Zeit nicht mehr akzeptiert werden.

Simulated Annealing (Simuliertes Ausglühen)

1. sei y eine Anfangslösung und T die Anfangstemperatur
2. REPEAT „hinreichend oft"
   - wähle zufällig einen Nachbarn $y' \in N(y)$
   - IF $f(y') \leq f(y)$ $y := y'$

     ELSE $y := y'$ mit

     $$\text{Prob} = e^{-\frac{(f(y') - f(y))}{T}}$$
3. wenn T noch nicht tief genug, wähle eine niedrigere T und GOTO 2