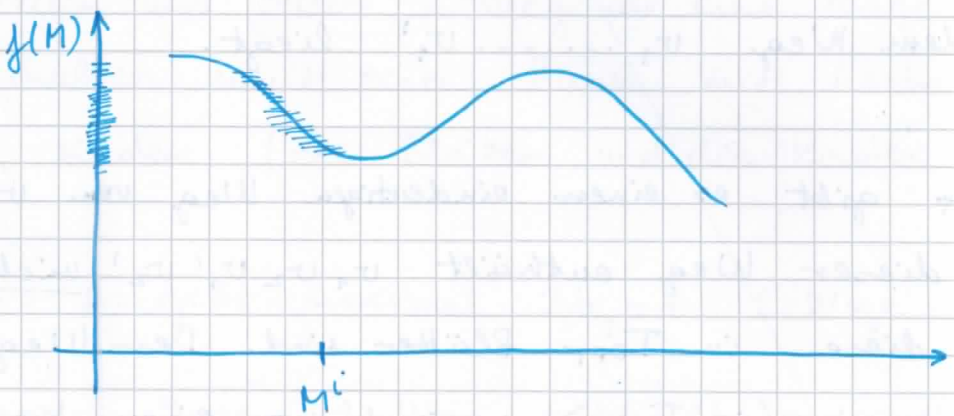
LS 20.

e.) Komplexität der Berechnung von lokalen Optima

→ Wir haben gesehen, dass eine beliebigen 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 ≠ 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!

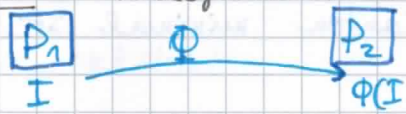
→ Wie könnte man jetzt schwierige Probleme in der Klasse PLS definieren? Wie hat man dies im Fall der Klasse NP gemacht?

→ 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 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 die Lösung $\Psi(I, y)$ 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.

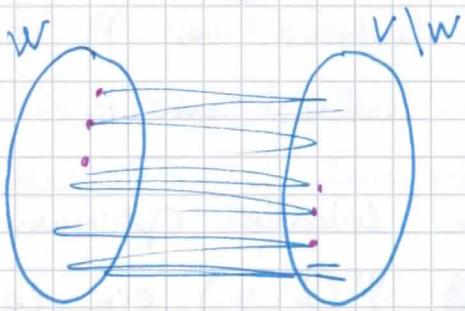
(Dass dies der Fall ist, ist ebenso sehr unwahrscheinlich wie $P = NP$. Es würde z.B. auch unter Anderen, $NP = co\text{-}NP$ implizieren ($P = NP$ aber nicht))

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?

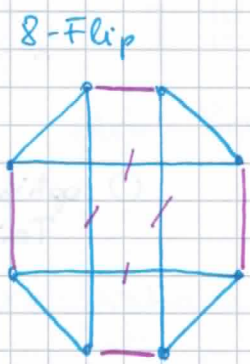
Die 2k-Flip Nachbarschaft ersetzt k Knoten aus W durch k Knoten aus $V \setminus W$.

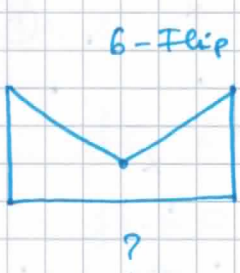
Thm: Mit der 2-Flip Nachbarschaft ist das entsprechende lokale Suchproblem für min-BALANCED-CUT PLS-vollständig.

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

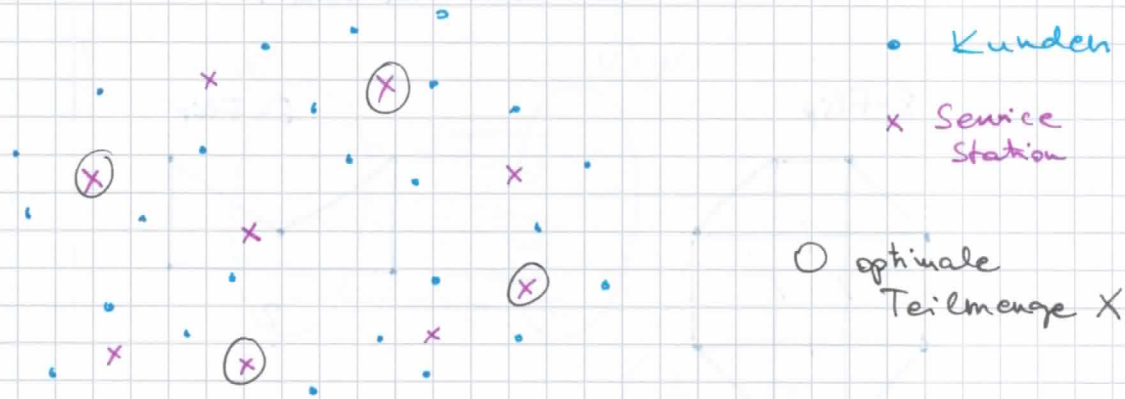
Beispiel: FACILITY LOCATION

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.

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



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 $|K| = n$ $K = \{k_1 k_2 \dots k_n\}$

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

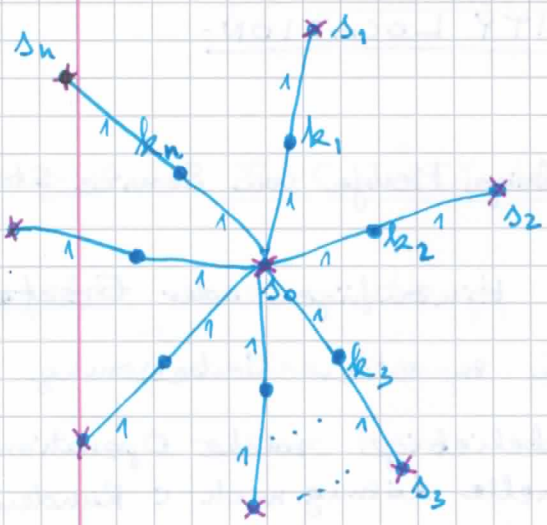
die Betriebskosten: $f_{s_0} = 2n-2$

$f_{s_i} = 0$ 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

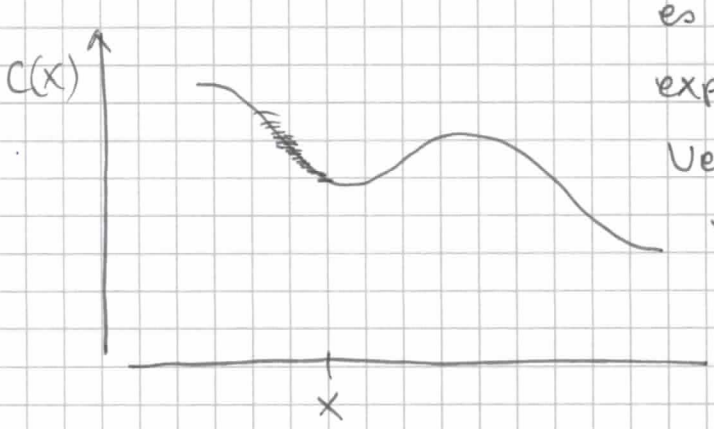
$X' = \{s_i\}$ → Anschluss $3(n-1)+1 = 3n-2$

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

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

⇒ Kein Nachbar hat kleineren Zielwert

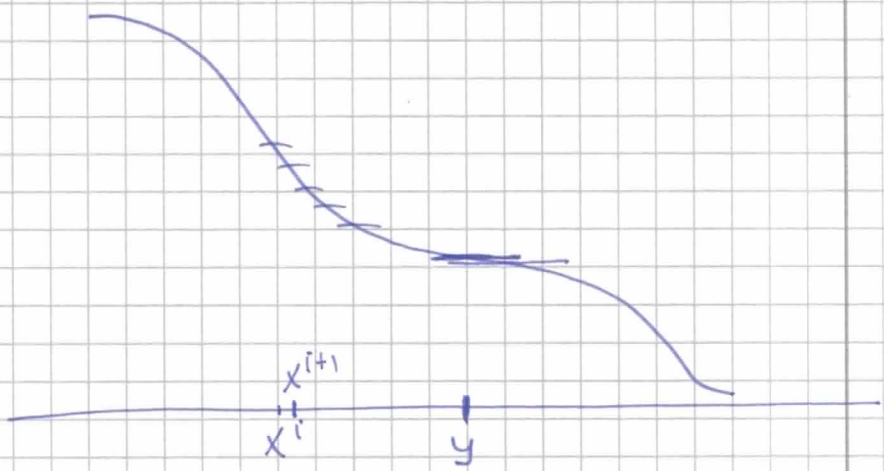
Problem: Die lokale Suche für FACILITY LOCATION ist nicht effizient!



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)

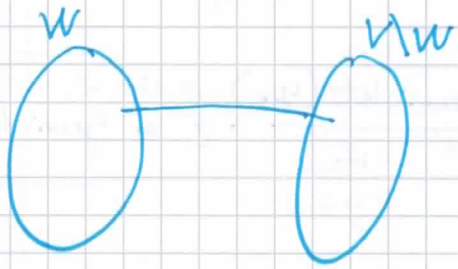
$$O\left(\frac{|S|}{\varepsilon} \cdot \ln \frac{C(x^0)}{C(x^*)}\right)$$

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

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

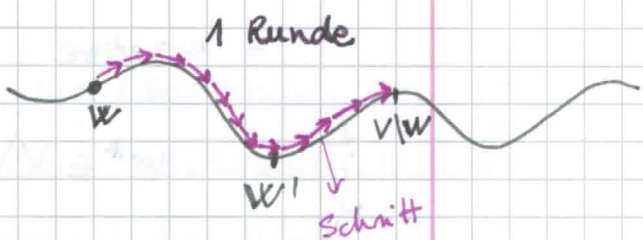
→ Wir betrachten die 2-Flip Nachbarschaft, also wenn zwei Knoten zwischen W und $V \setminus W$ getauscht werden, erhalten wir eine mit W benachbarte Lösung.

→ 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)

    ← ein Schritt
  - friere w und w^* ein

    (sie werden während FOR nicht mehr geändert)

    ← eine Runde
- 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)

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

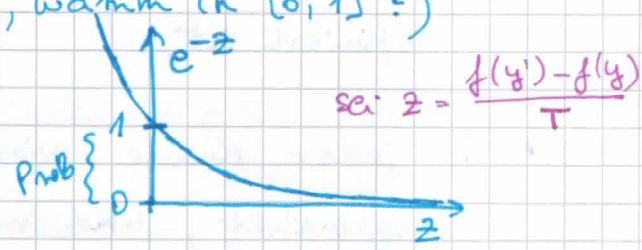
LS 40.

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, wann in $(0,1]$?)



sei $z = \frac{f(y')-f(y)}{T}$

→ 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

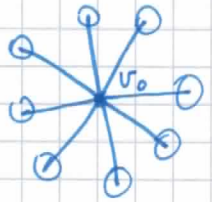
$e^{\frac{f(y')-f(y)}{T}} \approx 1$

⇒ Prob ≈ 1 (groß)

wenn T klein ⇒ Prob klein)

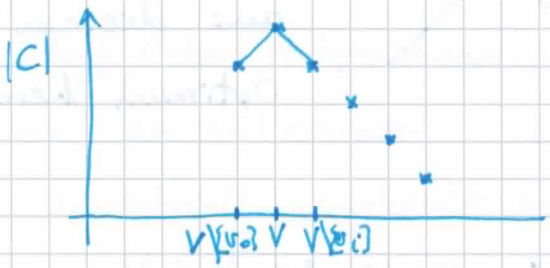
Beispiel: Metropolis Algorithmus auf dem Stemgraph für VERTEX COVER mit Anfangslösung V

Wäre zufällig das Zentrum als Erstes entfernt



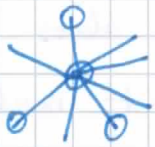
$y = V \setminus \{v_0\}$

→ 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



Wie soll die Temperatur T gesetzt werden?

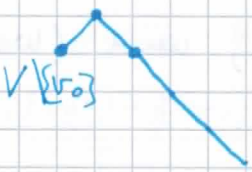
(falls $T >> f(y') - f(y) = 1$

dann die Akzeptanzwahrscheinlichkeit ist

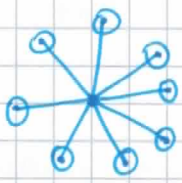
$$Prob = e^{-\frac{1}{T}} \approx 1$$

und falls die aktuelle Lösung wenige Satelliten enthält, wird wahrscheinlicher eine schlechtere Nachbar mit mehr Satelliten ausgewählt und weil Prob ≈ 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

Das langsame Abkühlen eines flüssigen Stoffes wird damit simuliert: Wenn T hoch ist, dann hat der Stoff / der Algorithmus viel Energie um lokale Minima zu entkommen.
Wenn zu schnell abgekühlt wird, dann bleibt der Stoff in einem lokal minimalen Energiezustand gefangen.
Bei Stoffen bedeutet dies Fehler in der Kristallgitterstruktur.
Ein globales Energieminimum bedeutet reine Kristallgitterstruktur.

LINEARE PROGRAMMIERUNG

(Buch über LP: Bertsimas-Tsitsiklis; siehe auch Kap. 9.4–9.9 in Moore-Mertens)

a.) Definition, kanonische Form

lineare Programmierung ist eine der wichtigsten Methoden der Optimierung, und zentral für Approximationsalgorithmen für schwierige Probleme (S.a. Shmoys, Vazirani)

Beispiel: Minimiere:

$$2x_1 - x_2 + 4 \cdot x_3$$ (eine lineare Zielfunktion)

so dass

$$x_1 + x_2 + x_4 \leq 2$$
$$3x_2 - x_3 = 5$$
$$x_3 + x_4 \geqslant 3$$
$$x_1 \geqslant 0$$
$$x_3 \leq 0$$

(lineare Nebenbedingungen – Gleichungen o. Ungleichungen; es kommen keine höheren Potenzen der Variablen vor) → oder andere Funktionen

x_1 x_2 x_3 x_4 sind Variablen. Eine Belegung der Variablen wird gesucht, die unter allen Lösungen die die Nebenbedingungen erfüllen, die Zielfunktion minimiert.

(manche Bedingungen betreffen das Vorzeichen von Variablen
$x_1 \geqslant 0$
$x_3 \leq 0$
diese werden meistens separat behandelt)

Wir haben einen Vektor von Variablen $x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$

Der lineare Term in den (Un)gleichungen kann als inneres Produkt mit einem Vektor (von Koeffizienten / Konstanten) aufgefasst werden:

$$a_1^T = (a_{11}\ a_{12}\ a_{13}\ a_{14}) = (1, 1, 0, 1)$$
$$a_2^T = (a_{21}\ a_{22}\ a_{23}\ a_{24}) = (0, 3, -1, 0)$$

Sei $b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 3 \end{bmatrix}$

Die erste Bedingung kann dann als $a_1^T \cdot x \leq b_1$ geschrieben werden

$$a_1^T \cdot x = (1, 1, 0, 1) \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = x_1 + x_2 + x_4$$

Die zweite als $a_2^T \cdot x = b_2$, usw.

(Das innere Produkt wird sofort als Matrix-Produkt aufgefasst. Wir müssen also Vektoren von ihrem jeweiligen transponierten Vektor unterscheiden.)

für $c = \begin{bmatrix} 2 \\ -1 \\ 4 \\ 0 \end{bmatrix}$: das Ziel ist $c^T \cdot x$ zu minimieren.

LP3.

Im allgemeinen betrachten wir Optimierungsprobleme der Form:

Minimiere / Maximiere $c^T \cdot x$ sodass

$$a_i^T \cdot x \geq b_i \quad \text{für } i \in M_1$$
$$a_i^T \cdot x = b_i \quad i \in M_2$$
$$a_i^T \cdot x \leq b_i \quad i \in M_3$$
$$x_j \geq 0 \quad j \in N_1$$
$$x_j \leq 0 \quad j \in N_2$$

wobei $x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ $b \in \mathbb{R}^m$ $c, a_i \in \mathbb{R}^n$ (oder $\mathbb{Q}^m$, $\mathbb{Q}^n$)

M_1, M_2, M_3 ist eine Partition von $\{1, 2, \ldots, m\}$ und $N_1, N_2 \subseteq \{1, 2, \ldots, n\}$

Die kanonische Form: o.B.d.A. haben wir nur Bedingungen $a_i^T \cdot x \geq b_i$ für $i = 1, 2, \ldots, m$

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \geq \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \iff \sum_{j=1}^{n} a_{ij} x_j \geq b_i \text{ für } i = 1, 2 \ldots m$$

Kurz: $A \cdot x \geq b$ für eine Matrix A und Vektor b

Definition: lineares Programm in kanonischer Form:

minimiere $c^T \cdot x$ so dass $Ax \geq b$
$(x \geq 0)$

wobei $x^T = (x_1 x_2 \ldots x_n)$ $c \in \mathbb{Q}^n$ $A \in \mathbb{Q}^{m \times n}$ $b \in \mathbb{Q}^m$

Beobachtung: Jedes lineare Programm (LP) hat ein äquivalentes LP in kanonischer Form.

(d.h. die kanonische Form ist allgemein genug)

Warum? Wie können wir die Folgenden umformulieren?

- maximiere $c^T \cdot x$ $\Longleftrightarrow$ minimiere $(-c^T) \cdot x$

- Bedingungen der Form

$$a_i^T \cdot x \leq b_i \iff (-a_i^T) \cdot x \geq -b_i$$

$$a_i^T \cdot x = b_i \iff a_i^T \cdot x \geq b_i \text{ und } (-a_i^T) \cdot x \geq -b_i$$

(wir haben keine Absicht negative Koeffizienten zu meiden)

Beispiel: Die entsprechende Matrix für unser Beispiel in kanonischer Form ist:

$$A \cdot x = \begin{bmatrix} -1 & -1 & 0 & -1 \\ 0 & 3 & -1 & 0 \\ 0 & -3 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \geq \begin{bmatrix} -2 \\ 5 \\ -5 \\ 3 \\ 0 \\ 0 \end{bmatrix}$$

Anwendungsbeispiele:

Beispiel 1. Produktionsplanung (ein ökonomisches Problem)

Eine Firma herstellt n verschiedene Produkte aus m verschiedenen Rohstoffen. Eine Einheit vom Produkt j braucht a_{ij} vom i-ten Rohstoff, und erzeugt einen Erlös von c_j.

Von Rohstoff i steht eine Menge von b_i zur Verfügung.

Wir wollen den Gesamterlös maximieren (so dass der Rohstoff reicht)

Sei x_j die (gesuchte) zu produzierende Menge vom Produkt j.

Um den Erlös zu maximieren, brauchen wir

$$\sum_{j=1}^{n} c_j \cdot x_j \text{ zu maximieren.}$$

Damit für jeden Rohstoff i den Vorrat reicht, soll

$$\sum_{j=1}^{n} a_{ij} \cdot x_j \leq b_i \qquad \text{(für jedes } i \in \{1, 2, \dots m\}\text{)}$$

gelten.

Produkt j

$$\begin{matrix} \\ 1 \\ \text{Rohstoff } i \\ m \end{matrix} \begin{bmatrix} a_{1j} \\ a_{ij} \\ a_{mj} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \leq \begin{bmatrix} b_1 \\ b_i \\ b_m \end{bmatrix}$$

Die folgenden Beispiele sind klassische Optimierungsprobleme formuliert als lineare Programme:

Beispiel 2. VERTEX COVER

Sei $G(V,E)$ der Eingabegraph mit Knotengewichten w_v und $C \subseteq V$ eine (gesuchte) Knotenüberdeckung mit minimalem Gewicht. Sei $V = \{1, 2, \ldots n\}$.
Unsere Variablen ("decision variables") sind

$$x_1 \; x_2 \; \ldots \; x_n$$

Die beabsichtigte Interpretation: zu einer Lösung C

$$x_v = \begin{cases} 1 & \text{falls } v \in C \\ 0 & v \notin C \end{cases}$$

Wie können wir unsere Zielfunktion und unsere Bedingungen als ein LP formulieren? Wir werden es nicht schaffen Vertex Cover vollkommen als ein LP zu formulieren. Warum? Lineare Programme sind in Polynomialzeit lösbar, VERTEX COVER aber nicht!

($\Rightarrow$ Statt $x_v \in \{0,1\}$, können wir nur $0 \leq x_v \leq 1$ fordern.)

Was ist zu minimieren? Das Gewicht der Knotenüberdeckung:

$$\text{minimiere} \quad \sum_{v=1}^{n} x_v \cdot w_v$$

$c^T = (w_1 \; w_2 \; \ldots \; w_n)$ (für ungewichtete Graphen $c^T = (1, 1, 1, \ldots, 1)$)

Was sind die Nebenbedingungen? Die Kanten müssen überdeckt werden.

für jede Kante $\{u,v\} \in E$ ($|E|$ Ungleichungen)

$$x_u + x_v \geq 1$$

und $0 \leq x_v \leq 1$ ($2 \cdot |V|$ Ungleichungen)

Falls wir zusätzlich $x_v \in \{0,1\}$ fordern (was genau dem VC Problem entspricht), haben wir ein ganzzahliges Programm IP (Integer (linear) program) IP-s können aber nicht effizient gelöst werden.

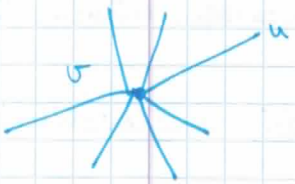
Die LP Variante mit $0 \le x_v \le 1$ ist eine Relaxierung des IP-s (mit $x_v \in \{0,1\}$).
Wir können das LP lösen, und eine 2-Approximation erhalten, siehe später.

Beispiel 3. Das gewichtete MATCHING Problem

Eingabe: $G(V,E)$ mit Kantengewichtungen w_e

Ausgabe: Ein Matching $M \subseteq E$ mit maximum Gewicht
(keine zwei Kanten in M haben einen gemeinsamen Endpunkt)

- die Variablen x_e $e \in E$
  Beabsichtigte Bedeutung: $x_e = \begin{cases} 1 & \text{falls } e \in M \\ 0 & \text{sonst} \end{cases}$
- zu maximieren ist die Zielfunktion $\sum_{e \in E} x_e \cdot w_e$
- die Nebenbedingungen:
  (für eine LP-Relaxierung:) $0 \le x_e \le 1$
  und damit die Lösung ein Matching ist:
  keine adjazenten Kanten
  dürfen in M sein: für jeden Knoten $v \in V$ maximal eine Kante:



$$\sum_{\{u \mid \{v,u\} \in E\}} x_{\{v,u\}} \le 1$$

(|V| Bedingungen) ↗ für jede $v \in V$ eine

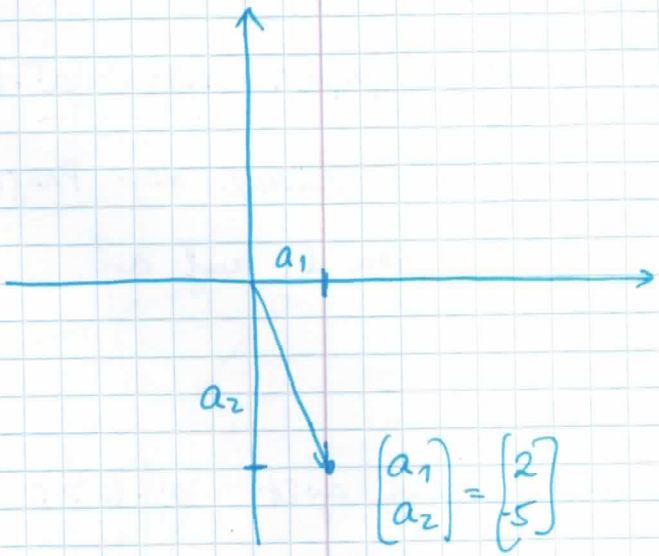
b.) Geometrische Grundlagen

i.) Geometrie in der Ebene $\mathbb{R}^2$ (Wiederholung)

– der Vektor

$\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$ $\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$ usw. $\right)$

bedeutet einen Punkt (bzw. einen Vektor mit diesem Endpunkt) in der Ebene

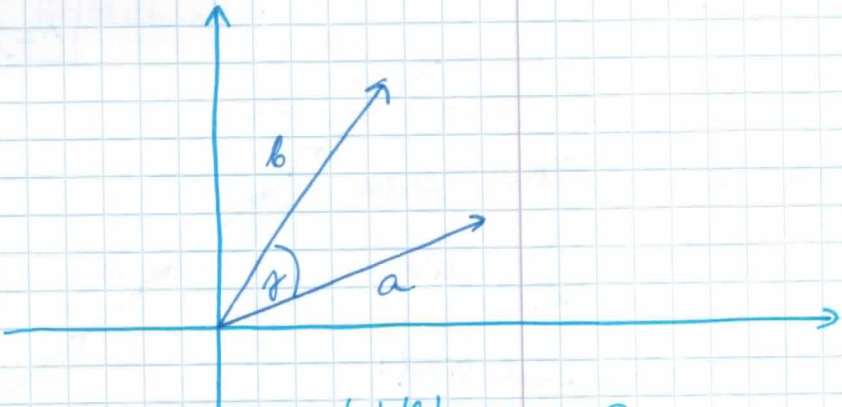


– das innere Produkt (Skalarprodukt) zweier Vektoren

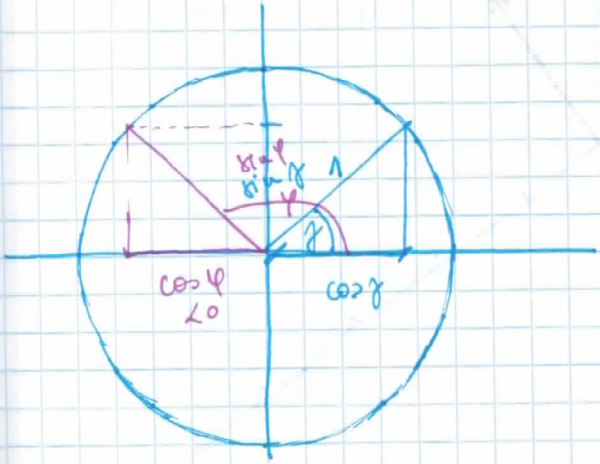
$$a = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad \text{und} \quad b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$a^T \cdot b = \begin{bmatrix} a_1 & a_2 \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = a_1 \cdot b_1 + a_2 \cdot b_2 = (b^T \cdot a)$$

diese Zahl (Skalar) gleicht $a^T \cdot b = b^T \cdot a = |a| \cdot |b| \cdot \cos\gamma$



Zur Erinnerung: für $0 \leq \gamma \leq 180°$



deshalb $a^T \cdot b = |a| \cdot |b| \cdot \cos\gamma = 0$

$\Updownarrow$

$a = 0$ oder $b = 0$ oder $(\cos\gamma = 0 \Leftrightarrow \gamma = 90° (\pm k \cdot 1$

dann sagen wir: a und b sind **orthogonal**

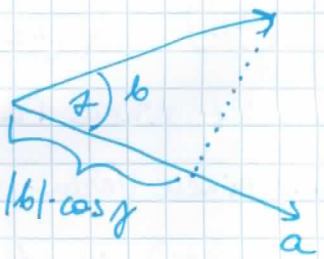
LP9.

(teste das Skalarprodukt von $a=\begin{bmatrix}3\\4\end{bmatrix}$ $b=\begin{bmatrix}3\\4\end{bmatrix}$

und von $a=\begin{bmatrix}3\\4\end{bmatrix}$ $b=\begin{bmatrix}-4\\3\end{bmatrix}$)

— für $|a|=1$ ist $a^T \cdot b = |b| \cdot \cos\gamma$

die Länge der Projektion von b auf a



— Wann gilt $a^T \cdot b > 0$? Falls $\cos\gamma > 0 \iff \gamma < 90°$

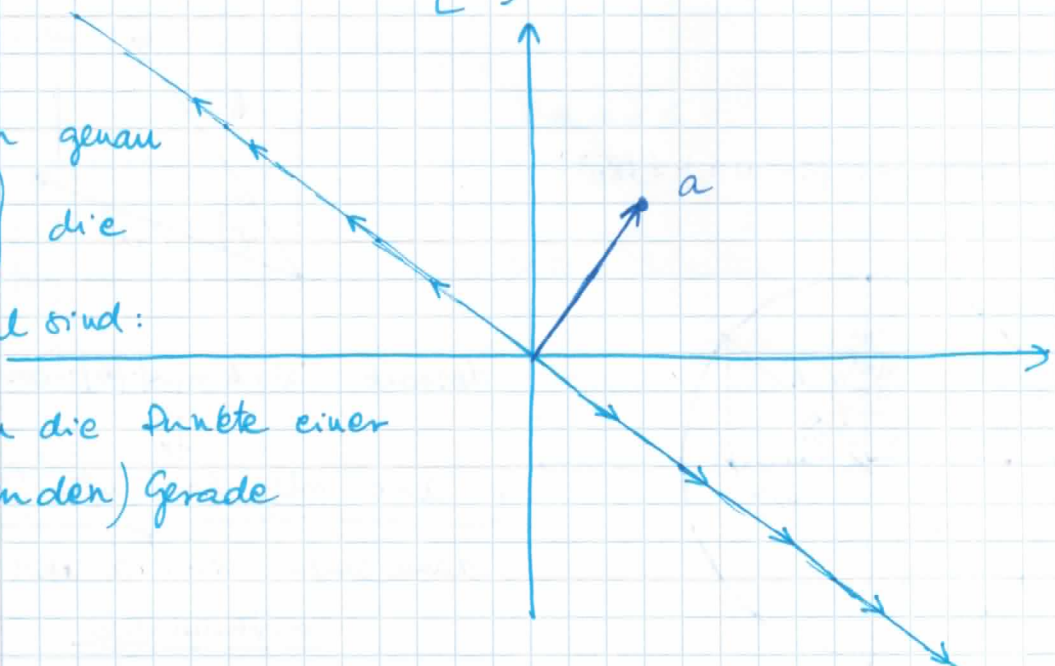
$a^T \cdot b < 0$ falls $\cos\gamma < 0 \iff 90° < \gamma \leq 180°$

— Sei $\begin{bmatrix}a_1\\a_2\end{bmatrix}$ fixiert (z.B. $\begin{bmatrix}a_1\\a_2\end{bmatrix}=\begin{bmatrix}3\\4\end{bmatrix}$)

Wo sind all die Punkte $\begin{bmatrix}x_1\\x_2\end{bmatrix}$ so dass $a^T \cdot x = 0$?

$a^T \cdot x = 0$ erfüllen genau die Vektoren $\begin{bmatrix}x_1\\x_2\end{bmatrix}$ die zu $\begin{bmatrix}a_1\\a_2\end{bmatrix}$ orthogonal sind:

die sind genau die Punkte einer (die 0 enthaltenden) Gerade



also: $a^T \cdot x = 0$ oder $a_1 x_1 + a_2 \cdot x_2 = 0$ ist die

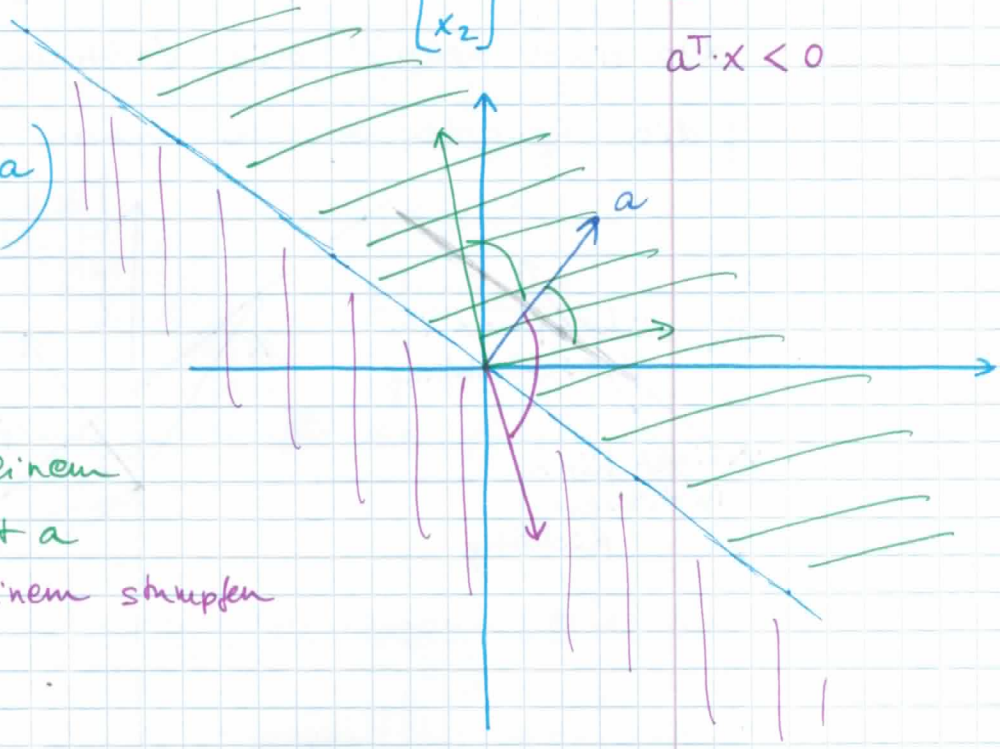
Gleichung einer zu $\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$ orthogonalen Gerade über 0.

(die Lösungen x zu $a^T \cdot x = 0$ bilden eine Gerade)

– Wo befinden sich die Punkte $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ mit $a^T \cdot x > 0$

$a^T \cdot x < 0$

($a^T \cdot x$ wächst in Richtung a und sinkt in Richtung $-a$)



die Vektoren mit einem spitzen Winkel mit a

die Vektoren mit einem stumpfen Winkel mit a

– die Punkte x mit $a^T \cdot x > 0$ (bzw. $a^T x < 0$) entsprechen einer offenen Halbebene

(die Lösungen x zu $a^T \cdot x \geq 0$ bilden eine Halbebene)

– die Punkte x mit $a^T \cdot x \geq 0$ ($bzw. a^T \cdot x \leq 0$) entsprechen einer abgeschlossenen Halbebene

---

– Insbesondere wo befinden sich die Punkte $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ mit $a^T \cdot x = 10$

Also, wo sind die Punkte mit

$$a_1 x_1 + a_2 x_2 = 10$$

z.B. $3x_1 + 4x_2 = 10$

Diese Punkte x formen auch eine Gerade, aber wo?