10.21

Satz

Sei F beliebige aussagenlogische Formel. Wir können zu F eine 3-KNF G konstruieren mit:

- F erfüllbar $\Leftrightarrow$ G erfüllbar
  (F, G erfüllbarkeitsäquivalent)
- Die Konstruktion läßt sich in Zeit $O(|F|)$ implementieren.
  (Insbesondere $|G| = O(|F|)$) $\square$

Beachte: Durch Ausmultiplizieren läßt sich jedes F in ein äquivalentes F in KNF transformieren. Aber

- Exponentielle Vergrößerung möglich.
- Keine 3-KNF.

10.30

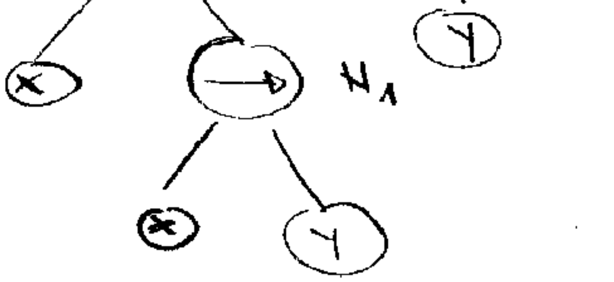
Beweis des Satzes:

Am Beispiel wird eigentlich alles klar. Der Trick besteht in der Einführung neuer Variablen!

$$F = (x \wedge (x \to y)) \vee \neg y$$

Schreiben F als Baum: Variablen = Blätter, Operationen = innere Knoten.

$F =$



H_1, H_2, H_3, H_4 : neue Variablen. Für jeden inneren Knoten eine neue Variable.

(10.31)

Idee: H_i = Wert an den Knoten, bei gegebener Belegung der alten Variablen x, y. Das drückt F' aus:

$$F' = (H_1 \leftrightarrow (x \to y)) \wedge (H_2 \leftrightarrow (x \wedge H_1))$$
$$\wedge (H_3 \leftrightarrow \neg y) \wedge (H_4 \leftrightarrow (H_3 \vee H_2))$$

Es ist F' immer erfüllbar. Brauchen nur die H_i von unten nach oben zu setzen. Aber

Variable der Wurzel

$$F'' = F' \wedge H_4$$

erfordert, daß alle H_i richtig und $H_4 = 1$ ist. Es gilt:

Ist $a(F) = 1$ so können wir eine Belegung b konstruieren, in der die $b(N_i)$ richtig stehen, so daß $b(F'') = 1$ ist. Ist andererseits $b(F'') = 1$, so $b(N_4) = 1$ und eine kleine Überlegung zeigt uns, daß die $b(x)$, $b(y)$ eine erfüllende Belegung von F sind. $\square$

3-KNF ist gemäß Prinzip auf Seite 10.13 zu bekommen. Anwendung auf die Äquivalenzen. $\square$

Frage: Warum kann man so keine 2-KNF bekommen?

Davis Putnam basiert auf dem Prinzip

F erfüllbar $\Leftrightarrow$ $F_{|x=1}$ oder $F_{|x=0}$ erfüllbar.

(Verzweigen nach dem Wert von x)

Haben wir eine 3-KNF, etwa

$$F = \ldots \wedge (x \vee \neg y \vee z) \wedge \ldots$$

dann

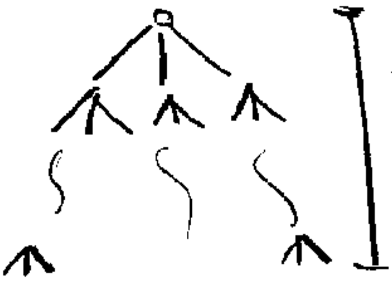
F erfüllbar

$\Leftrightarrow$ $F_{|x=1}$ oder $F_{|y=0}$ oder $F_{|z=1}$

erfüllbar

(Verzweigen nach der Klausel).

Ein backtracking Algorithmus nach diesem Muster führt zu einer Aufrufstruktur der Art



Tiefe m

$\Rightarrow$ 3^m Blätter (?!)

Aber es gilt auch:

F erfüllbar

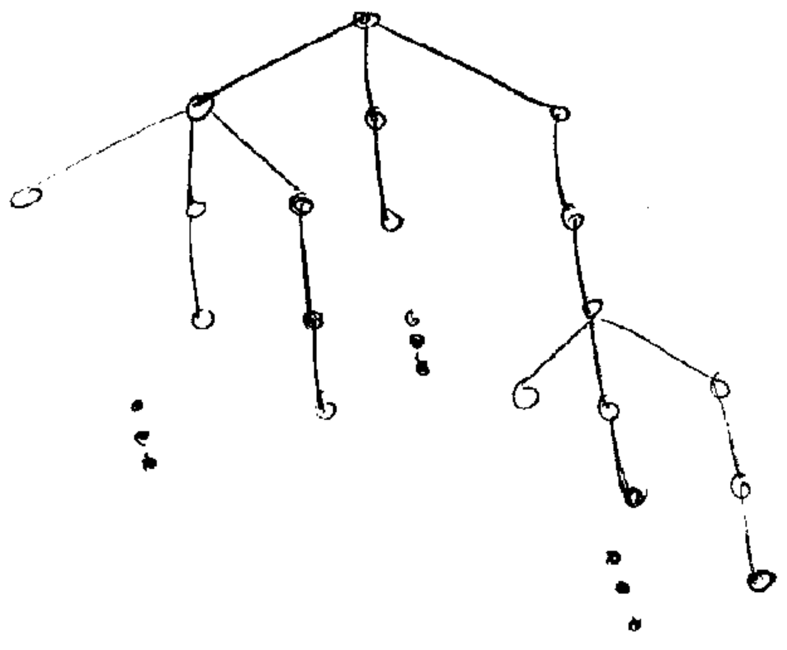
$\Leftrightarrow$

$F|_{x=1}$ erfüllbar

oder $F|_{x=0,y=0}$ erfüllbar

oder $F|_{x=0,y=1,z=1}$ erfüllbar.

Damit Aufrufstruktur im Backtracking:



10.35

Bei wieder

$T(n) :=$ maximale # Blätter bei n Variablen

dann gilt

$$T(n) \leq T(n-1) + T(n-2) + T(n-3)$$

für $n \geq n_0$, n_0 eine Konstante

$$T(n) \leq d = O(1)$$

für $n < n_0$. Was ist das? Lösung

$$T(n) = T(n-1) + T(n-2) + T(n-3).$$

Ansatz: $T(n) = c \cdot \alpha^n$ für alle n groß genug. Dann

$$c\alpha^n = c\alpha^{n-1} + c\alpha^{n-2} + c\alpha^{n-3}$$

also $\alpha^3 = \alpha^2 + \alpha + 1$.

(10.36)

Wir zeigen: Für $\alpha > 0$ mit

$$\alpha^3 = \alpha^2 + \alpha + 1$$

gilt $T(n) = O(\alpha^n)$.

Ist $n < n_0$, dann

ist $T(n) \le d \cdot \alpha^n$

für geeignete Konstante d, da $\alpha > 0$!

Sei $n \ge n_0$. Dann

$$T(n) = T(n-1) + T(n-2) + T(n-3)$$

Ind.-Vor. $\alpha > 0$.

$$\le d\,\alpha^{n-1} + d \cdot \alpha^{n-2} + d \cdot \alpha^{n-3}$$

$$= d\left(\alpha^{n-3}\overbrace{\left(\alpha^2 + \alpha + 1\right)}^{= \alpha^3 \text{ nach Wahl von } \alpha}\right)$$

$$= d\,\alpha^n, = O(\alpha^n).$$

Genauso $T(n) = \Omega(\alpha^n)$. Da

die angegebene Argumentation für jedes(!) $\alpha > 0$ mit $\alpha^3 = \alpha^2 + \alpha + 1$ gilt, darf es nur ein solches α geben! Es gibt ein solches $\alpha < 1{,}8393$.

Dann ergibt sich, daß die Laufzeit $O(|F| \cdot 1{,}8393^n)$ ist, da die inneren Knoten durch den konstanten Faktor mit erfaßt werden können. (10)

Frage: Wie genau geht das?

Bevor wir weiterverbessern, diskutieren wir, was derartige Verbesserungen bringen.

Zunächst ist der exponentielle Teil ausschlaggebend (zumindest für die Asymptotik, (n groß) der Theorie):

Es ist für $c > 1$ konstant und $\varepsilon > 0$ konstant (klein) und k konstant (groß)

$$n^k \cdot c^n \leq c^{(1+\varepsilon)n},$$

sofern n groß genug, denn

$$c^{\varepsilon n} \geq c^{\log_c n \cdot k} = n^k$$

für $\varepsilon > 0$ konstant und n groß genug (früher schon gezeigt).

Haben wir jetzt zwei Algorithmen

A_1 Zeit 2^n

A_2 Zeit $2^{\frac{1}{2}n} = (\sqrt{2})^n = (1{,}4\ldots)^n$

für dasselbe Problem. Betrachten wir eine vorgegebene Zeit x (fest). Mit A_1 können wir alle Eingaben der Größe n mit

$$2^n \leq x, \text{ d.h. } n \leq \log_2 x$$

in Zeit x sicher bearbeiten. Mit A_2 dagegen alle mit

$$2^{\frac{1}{2}n} \leq x, \text{ d.h. } \tfrac{1}{2}n \leq \log_2 x$$

$$\text{d.h. } n \leq 2 \cdot \log_2 x,$$

also Vergrößerung um konstanten Faktor! Lassen wir dagegen A_1 auf einem neuen Rechner laufen, auf dem jede Instruktion 10-mal so schnell abläuft, so

$\frac{1}{10} \cdot 2^n \leq x$ d.h. $n \leq \log_2 10x$

$= \log_2 x + \log_2 10$

nur die Addition einer Konstanten.

Fazit: Bei exponentiellen Algorithmen schlägt eine Vergrößerung der Rechengeschwindigkeit weniger durch als eine Verbesserung der Konstanten c in c^n der Laufzeit.

Aufgabe: Führen Sie die Betrachtung des schnelleren Rechners für polynomiale Laufzeiten n^k durch.

(10.41)

Haben wir das Literal x pur in F (d.h. $\neg x$ ist nicht dabei), so reicht es $F|x=1$ zu betrachten.

Analoges gilt, wenn wir mehrere Variablen gleichzeitig ~~setzen~~ setzen.

Betrachten wir die Variablen $x_1, \dots, x_k$ und Wahrheitswerte $b_1, \dots, b_k$ für diese Variablen und gilt

$$H := F|_{x_1=b_1, x_2=b_2, \dots, x_k=b_k} \subseteq F$$

d.h. jede Klausel C von H ist schon in F, d.h. wenn das Setzen von einem $x_i = b_i$

verkleinert

10.42

einen Klausel ändert, gibt es ein $x_j, \neg x_j$ in dieser Klausel, das durch $x_j = b_j$ wahr wird, so gilt

$$F \text{ erfüllbar} \iff H \text{ erfüllbar}.$$

wie beim Korrektheitsbeweis der pure literal rule. Kein backtracking möglich!

Wir sagen, die Belegung $x_1 = b_1, \dots, x_k = b_k$ ist autark für F. Etwa

$$F = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge G,$$

G ohne $x_3, x_1, \neg x_2$

dann

$$F|_{x_1 = 0,\, x_2 = 0,\, x_3 = 0} \subseteq F,$$

also $x_1 = 0, x_2 = 0, x_3 = 0$ autark für F. (In G fallen höchstens Klauseln weg.)

Algorithmus (Monien, Speckenmeyer)

MoSp(F) {

1. if F offensichtlich wahr; return „erf";
2. if F offensichtlich unerfüllbar; return „unerfüllbar";

3. Wähle eine kleinste Klausel C = $l_1 \vee \dots \vee l_s$ in F;
// l_i: Literal, x, oder $\neg x$
4. Betrachte die Belegungen
$l_1 = 1$; $l_1 = 0, l_2 = 1$; ...
$l_1 = 0, l_2 = 0, \dots, l_s = 1$
5. if eine der Belegungen autark in F; {
b := eine autarke Belegung;
return MoSp(F|b) }

6. Teste: MoSp(F_i) für $i = 1, \ldots, s$, und F_i jeweils durch Setzen einer der obigen Belegungen. return „erfüllbar", wenn ein Aufruf „erfüllbar" gibt, return „unerfüllbar" sonst.

// Hier ist
// keine der
// Belegungen
// erfüllt

Wir beschränken uns bei der Analyse der Laufzeit auf den 3-KNF Fall. Sei wieder

$T(n)$ = # Blätter bei kleinster Klausel mit 3 Literalen

$T'(n)$ = " ≤ 2 Literalen.

(10.45)

Für $T(n)$ gilt:

$T(n) \leq T(n-1)$

(autarke Belegung gefunden, mindestens 1 Variable weniger)

$T(n) \leq T'(n-1) + T'(n-2) + T'(n-3)$

(Keine autarke Belegung gefunden, dann aber Klauseln der Größe ≤ 2, Alg. nimmt immer kleinste Klausel)

Ebenso bekommen wir

$T'(n) \leq T(n-1)$

$T'(n) \leq T'(n-1) + T'(n-2)$

(Haben ≤ 2'er Klausel ohne autarke Belegung)

(10.46)

Zwecks Verschwindens des ≤ - Zeichens neue $T(n)$, $T'(n)$ ≥ die alten $T(n)$, $T'(n)$.

$T(n) = d$ für $n \leq n_0$, d Konstante

$T(n) = \max\{T(n-1), T'(n-1) + T'(n-2) + T'(n-3)\}$

für $n > n_0$

$T'(n) = d$ für $n \leq n_0$, d Konstante

$T'(n) = \max\{T(n-1), T'(n-1) + T'(n-2)\}$

Zunächst müssen wir das Maximum loswerden. Dazu zeigen wir daß

für

$S(n) = d$ für $n \leq n_0$

$S(n) = S'(n-1) + S'(n-2) + S'(n-3)$

für $n > n_0$

$S'(n) = d$ für $n \leq n_0$

$S'(n) = S'(n-1) + S'(n-2)$, für $n > n_0$

gilt, daß

$$T(n) \le S(n), \quad T'(n) \le S'(n).$$

Ind.-Anf. ✓

Ind.-Schluß

$T(n)$

$=$

$\max\{T(n-1), T'(n-1) + T'(n-2) + T'(n-3)\}$

$\overset{\text{Ind.-Vor}}{\le}$

$\max\{S(n-1), S'(n-1) + S'(n-2) + S'(n-3)\}$

$\overset{\text{Monotonie}}{\le}$

$S'(n-1) + S'(n-2) + S'(n-3)$

$=$

$S(n)$.

$$T'(n)$$

$$=$$

$$\mathrm{Max}\{T(n-1),\ T'(n-1)+T'(n-2)\}$$

$$\overset{\text{Ind - Vor}}{\leq}$$

$$\mathrm{Max}\{S(n-1),\ S'(n-1)+S'(n-2)\}$$

$$=$$

$$\mathrm{Max}\{\overbrace{S'(n-2)+S'(n-3)}^{=S'(n-1)}+S'(n-4),\ S'(n-1)+S'(n-2)\}$$

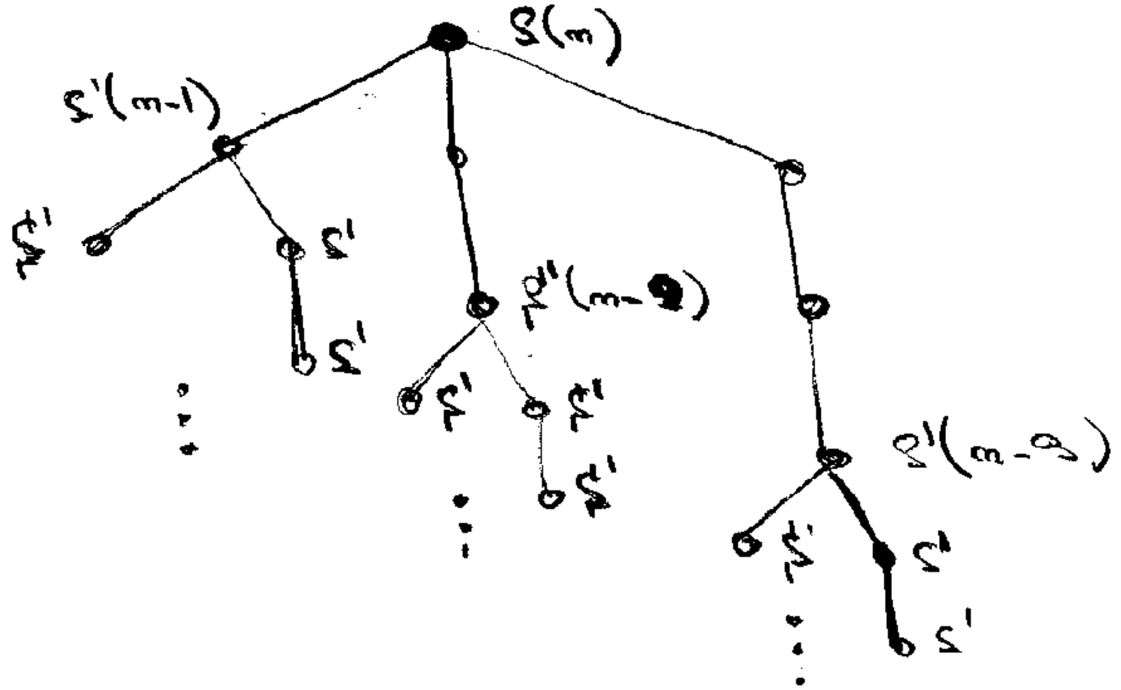
$$\overset{\text{Houdini}}{=}$$

$$S'(n-1)+S'(n-2)$$

$$=$$

$$S'(n)$$

Der Rekursionsbaum für $S(n)$ hat nun folgende Struktur:



Ansatz $\quad S'(n) = \alpha^n$

$$\alpha^n = \alpha^{n-1} + \alpha^{n-2}$$

$$\Rightarrow \quad \alpha^2 = \alpha + 1$$

Ist $\alpha > 0$ Lösung der Gleichung, dann $\quad S'(n) = O(\alpha^n)$

(10.50)

Ind.-Anf: $n \le n_0$ ✓ da $\alpha > 0$.

Ind.-Schluß:

$$f'(n+1) = f'(n) + f'(n-1)$$

Ind.-Vor

$$\le c\cdot\alpha^n + c\cdot\alpha^{n-1}$$

$$= c\alpha^{n-1}(\alpha + 1)$$

Wahl von α

$$= c\,\alpha^{n-1}\,\alpha^2$$

$$= c\cdot\alpha^{n+1} = O(\alpha^{n+1}).$$

Dann auch $f(n) = O(\alpha^n)$

und Laufzeit von Mon. Sp $O(\alpha^n\cdot|F|)$

Für F in 3-KNF ist $|F| \le (2n)^3$

also Zeit $O(\alpha^{(1+\varepsilon)n})$ für n groß genug.

Es ist $\alpha < 1{,}681$.

(10.51)

Die Rekursionsgleichung von $S'(n)$ ist wichtig.

<u>Definition</u>

Die Zahlen $(F_n)_{n \geq 0}$ mit

$$F_0 = 1, \; F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \text{ für alle } n \geq 2$$

heißen Fibonacci Zahlen. $\square$

Es ist $F_n = \Theta(1{,}681^n)$. $\blacksquare$

Verwandt mit dem Erfüllbarkeitsproblem ist das Max-Sat- und Max-k-Sat Problem. Dort hat man als Eingabe eine Formel $F = C_1 \wedge \dots \wedge C_m$ in KNF und sucht eine Belegung a, so daß

$$|\{ i \mid a(C_i) = 1 \}| \quad \text{maximal}$$

ist. Für das Max-Ek-Sat Problem, d.h. jede Klausel besteht genau aus k verschiedenen Literalen hat man folgenden Zusammenhang.

Satz

Sei

$m = \# F = C_1 \wedge \dots \wedge C_m$

Formel gemäß Max-Ek-Sat. ($C_i = C_j$ ist zugelassen.) Es gibt Belegung a, so daß

$$|\{ j \mid a(C_j) = 1 \}| \geq \left(1 - \frac{1}{2^k}\right) \cdot m$$

<u>Beweis:</u> Eine feste Klausel $C \in \{C_1, \ldots, C_m\}$ wird von wievielen Belegungen falsch gemacht?

$$\leq 2^{n-k}, \quad n = \#\,\text{Variablen},$$

denn die k Literale der Klausel müssen alle auf 0 stehen.

Also haben wir folgende Situation vorliegen:

Bedeutet $a_i(C_1) = 1$

$$\begin{array}{lccccc}
 & a_1 & a_2 & \cdots & a_{2^n} & \\
C_1 & * & * & * & * & \geq 2^n - 2^{n-k} \\
C_2 & \times & \times & \times & \times & = 2^n\left(1 - \frac{1}{2^k}\right) \\
\vdots & \times & \times & \times & \times & *\text{'e pro Zeile.} \\
 & & \vdots & & & \\
C_m & \times & \times & \times\ \times & \cdots\ \times &
\end{array}$$

$\#$ [illegible] $\sum_{i=1}^{m} |\ [illegible]\ (C_i) = 1\ |$

Zeilenweise Summation ergibt

$m \cdot 2^n \cdot \left(1 - \frac{1}{2^k}\right)$

$\leq \sum_{j=1}^{m} \left| \{ a_i \mid a_i(C_j) = 1 \} \right|$

$=$ # *'e insgesamt.

Spaltenweise Summation

$\sum_{i=1}^{2^n} \left| \{ j \mid a_i(C_j) = 1 \} \right|$

$=$ # *'e insgesamt.

$\geq 2^n \cdot \left(m \cdot \left(1 - \frac{1}{2^k}\right)\right)$

Da wir 2^n Summanden haben, muß es ein a_i geben mit

$$\left| \{ j \mid a_i(C_j) = 1 \} \right| \geq m \cdot \left(1 - \frac{1}{2^k}\right). \qquad \square$$

Wie findet man eine Belegung a, so daß stets $|\{j \mid a(C_j) = 1\}|$ maximal?

Letzteres durch Ausprobieren, sofern $k \geq 2$, also Zeit $O(|F| \cdot 2^n)$. (Für $k = 2$ gibt es in $O(c^n)$ für ein $c < 2$)

Wir haben 2 ungleiche Brüder:

2-Sat polynomielle Zeit

Max-2-Sat nur exponentielle Algorithmen vermutet.

Vergleiche bei Graphen mit Kantenlängen ≥ 0:

kürzester Weg polynomial

Längster kreisfreier Weg nur exponentiell bekannt.

auch 2-färbbar (poly) 3-färbbar (nur expo. bekannt)

10.56

Ein ähnliches Phänomen betrifft das Problem des maximalen Schnitts: Für einen Graphen $G=(V,E)$ ist ein Paar (S_1, S_2) mit $S_1 \dot\cup S_2 = V$ ein Schnitt ($S_1 \cap S_2 = \emptyset$, $S_1 \cup S_2 = V$).
Eine Kante $\{v, w\}$, $v \neq w$, liegt im Schnitt (S_1, S_2) gdw.
$v \in S_1$, $w \in S_2$ (oder umgekehrt).

Es gilt: G 2-färbbar $\Leftrightarrow$ es gibt Schnitt, so daß jede Kante in diesem Schnitt liegt.

Beim Problem des maximalen Schnitts geht es darum einen Schnitt (S_1, S_2) zu finden, so daß

$|\{e \in E \mid e \text{ liegt im } (P_1, P_2)\}|$ maximal

ist.

Wie beim Max-Ek-Satz gilt:

<u>Satz</u>

Zu $G = (V, E)$ gibt es Schnitt (P_1, P_2), so daß

$$|\{e \in E \mid e \text{ in } (P_1, P_2)\}| \geq |E|/2 .$$

Beweis: Übungsaufgabe. $\square$

Algorithmus (Findet Schnitt in dem $\geq |E|/2$ Kanten liegen)

Eingabe: $G=(V,E)$, $V=\{1,\ldots,n\}$.

1. for $v=1$ to n {
2. if Knoten v hat mehr direkte Nachbarn in S_2 als S_1 {

   $S_1 = S_1 \cup \{v\}$; break;

   }

3. $S_2 = S_2 \cup \{v\}$ $\square$

Laufzeit: $O(n^2)$, S_1, S_2 als boolesches array.

Wieviele Kanten liegen im Schnitt (S_1, S_2)? Invariante:

Nach dem j-ten Lauf gilt:
Von den Kanten $\{v,w\} \in E$
mit $v, w \in \tilde{V}_{1,j} \cup \tilde{V}_{2,j}$
liegt mindestens die Hälfte
im Schnitt $(\tilde{V}_{1,j}, \tilde{V}_{2,j})$.
Daraus folgt die Korrektheit.
So findet man aber nicht
immer einen Schnitt mit einer
maximalen Anzahl Kanten.

Frage: Beispiel dafür.

Das Problem minimaler Schnitt
dagegen fragt nach einem Schnitt
(V_1, V_2), so daß $|\{e \in E \mid e \text{ in } (V_1, V_2)\}|$
minimal ist. Dieses Problem

löst Ford-Fulkerson in
polynomialer Zeit.

Frage: Wie ist Übungsaufgabe.

Für den maximalen Schnitt
ist ein solcher Algorithmus
nicht bekannt. Wir haben also
wieder:

Minimaler Schnitt polynomiell
Maximaler Schnitt nur exponentiell
bekannt.

Die eben betrachteten Probleme sind
kombinatorische Optimierungsprobleme.
Deren bekanntestes ist das Problem
des Handlungsreisenden (TSP - travelling
salesperson)