10. Kombinatorische Suche und Rekursionsgleichungen

Bisher meistens Probleme in polynomieller Zeit (und damit auch Platz). Obwohl es prinzipiell exponentiell viele mögliche Wege von $u \to v$ gibt, gelingt es mit Dijkstra oder Floyd-Warshall, einen kürzesten Weg systematisch aufzubauen, ohne alles zu durchsuchen. Das liegt daran, daß man die richtige Wahl lokal erkennen kann. Bei Ford-Fulkerson haben wir prinzipiell unendlich viele Flüsse, trotzdem können wir einen maximalen systematisch aufbauen, ohne blind durchzuprobieren. Dadurch polynomiale Zeit.

Jetzt: Probleme bei denen man die Lösung nicht mehr zielgerichtet aufbauen kann, sondern im wesentlichen exponentiell viele Lösungskandidaten durchsuchen muß. (Das bezeichnet man als kombinatorische Suche.

Einfachst: Aussagenlogische Probleme.

Aussagenlogische Formeln, wie zum Beispiel

$$x \wedge y \vee \neg\left(\left(u \wedge \neg(v \wedge \neg x)\right) \to y\right),$$

$$x \vee y, \quad x \wedge y, \quad (x \vee y) \wedge (x \vee \neg y).$$

Also wir haben eine Menge von aussagenlogischen Variablen zur Verfügung, wie $x, y, v, u, \ldots$. Formeln werden mittels der üblichen aussagenlogischen Operationen

$\wedge$ (und), $\vee$ (oder, lat. vel)

$\neg$ oder $^-$ (nicht), $\to$ (Implikation)

$\leftrightarrow$ (Äquivalenz)

aufgebaut. Variablen stehen für die Wahrheitswerte 1 (= wahr) und 0 (= falsch). Die Implikation hat folgende Bedeutung:

x	y	$x \to y$
0	0	1
0	1	1
1	1	1
1	0	0

Nur dann falsch, wenn aus Wahrem Falsches folgen soll

Die Äquivalenz $x \leftrightarrow y$ ist genau dann wahr, wenn x und y beide den gleichen Wahrheitswert haben, also beide gleich 0 oder gleich 1 sind.

Damit ist $x \leftrightarrow y$ gleichbedeutend zu $(x \to y) \land (y \to x)$.

Ein Beispiel verdeutlicht die Relevanz der Aussagenlogik: Worin besteht das Geheimnis Ihres langen Lebens? Folgender Diätplan wird eingehalten:

- Falls kein Bier beim Essen, dann wird in jedem Falle Fisch gegessen.
- Falls aber Fisch und Bier, dann keinesfalls Eis.
- Falls Eis oder auch kein Bier, dann kein Fisch.

(10.6)

Was wird denn nun gegessen?

Dazu Aussagenlogik.

B = Bier beim Essen

F = Fisch

E = Eis.

Aussagenlogisch werden obige Aussagen nun so:

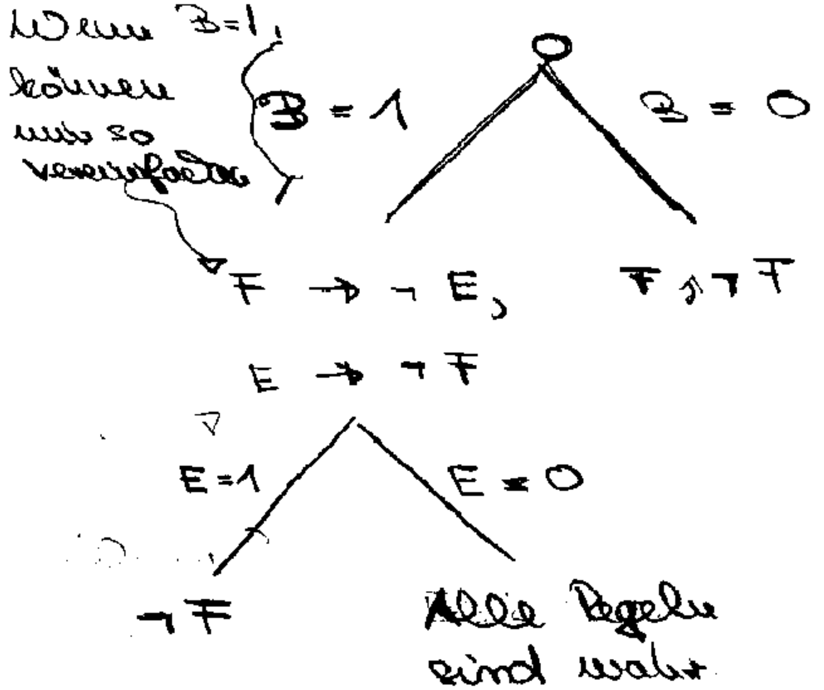
$$\neg B \rightarrow F$$

$$F \wedge B \rightarrow \neg E$$

$$E \vee \neg B \rightarrow \neg F$$

Aussagenlogik erlaubt die direkte Darstellung von Wissen (Wissensrepräsentation – ein eigenes Fach).

Wir wollen nun feststellen, was verzehrt wird:



Also: Immer Bier und falls Eis dann kein Fisch.

$$B \wedge (E \to \neg F).$$

Das ist gleichbedeutend zu

$$B \wedge (\neg E \vee \neg F)$$

und gleichbedeutend zu

$$B \wedge (\neg(E \wedge F)),$$

Immer Bier; Eis und Fisch nicht zusammen.

Die Syntax der aussagenlogischen Formeln sollte soweit klar sein. Die Semantik (Bedeutung) kann erst dann erklärt werden, wenn die Variablen einen Wahrheitswert haben, d.h. wir haben eine Abbildung

$$a: \text{Variablen} \to \{0,1\},$$

d.h. eine Belegung der Variablen mit Wahrheitswerten gegeben.

Dann ist $a(F)$ = Wahrheitswert von F bei Belegung a.

Ist $a(x) = a(y) = a(z) = 1$, dann

$$a(\neg x \vee \neg y) = 0,\ a(\neg x \vee \neg y \vee z) = 1$$

$$a(\neg x \wedge (y \vee z)) = 0.$$

Für eine Formel F der Art

$$F = G \wedge \neg G$$

gilt $a(F) = 0$ für jedes a,

für $F = G \vee \neg G$ ist $a(F) = 1$ für jedes a.

Einige Bezeichnungen:

- F ist erfüllbar gdw. es gibt eine Belegung a mit $a(F) = 1$.

- F ist unerfüllbar (widersprüchlich)

  gdw. für alle a ist $a(F) = 0$.

- F ist tautologisch

  gdw. für alle a ist $a(F) = 1$.

Beachte: Ist F nicht tautologisch, so heißt das im allgemeinen nicht, daß F unerfüllbar ist.

<u>Algorithmus</u> (Erfüllbarkeitsproblem)

Eingabe: F

1. Erzeuge hintereinander alle Belegungen a (0–0, 0–1, 0–10, ..) Ermittle $a(F)$. Ist $a(F) = 1$, return „F erfüllbar durch a".
2. return „F unerfüllbar".

Laufzeit: Bei n Variablen

$\mathcal{O}(2^n \cdot |F|)$, = #Bits (#Zeichen)

wobei $|F|$ = Größe von F ist.

Dabei muß F in einer geeigneten Datenstruktur vorliegen. Wenn F erfüllbar ist kann die Zeit wesentlich geringer sein! Reiner worst-case. Verbesserung durch backtracking: Schrittweises Einsetzen der Belegung und Vereinfachen von F (Davis-Putnam Prozedur, siehe unten).

Die Semantik einer Formel F mit n Variablen läßt sich auch verstehen als eine Funktion

$$F : \{0,1\}^n \to \{0,1\}.$$

Frage: Wieviele derartiger Funktionen gibt es ?

Jede derartige Funktion läßt sich in konjunktiver Normalform (KNF) darstellen. Auch in disjunktiver Normalform.

Konjunktive Normalform ist:

Literal Literal

$$\underbrace{(x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4)}_{\text{Klausel}} \wedge (\neg x_2 \vee x_3 \vee x_2)$$

$$\wedge (\neg x_1 \vee x_5 \vee \dots \vee x_m) \wedge \dots$$

Disjunktive Normalform ist:

$$(x_1 \wedge x_2 \wedge \neg x_3 \wedge x_4) \vee (x_2 \wedge x_3 \wedge \neg x_4)$$

$$\wedge (\neg x_1 \wedge x_5 \wedge \dots \wedge x_m) \wedge \dots$$

(10.13)

Im Prinzip reichen DNF oder KNF aus. Das heißt: ist

$$F : \{0,1\}^n \to \{0,1\}$$

eine Boolsche Funktion, so läßt sich F als KNF oder auch DNF darstellen.

KNF: Wir geben alle $(b_1, \ldots, b_n) \in \{0,1\}^n$ für die $F(b_1, \ldots, b_n) = 0$ ist, durch. Wir schreiben Klauseln nach folgendem Prinzip:

$$F(0 \ldots 0) = 0 \leadsto x_1 \vee x_2 \vee \ldots \vee x_n$$

$$F(110 \ldots 0) = 0 \leadsto \neg x_1 \vee \neg x_2 \vee \ldots \vee x_n$$

Die Konjunktion dieser Klauseln gibt die Formel, die F darstellt.

10.14

DNF: Alle $(b_1, \dots, b_n) \in \{0,1\}^n$ bei die $F(b_1, \dots, b_n) = 1$. Klauseln analog oben:

$$F(0,0,\dots,0) = 1 \rightsquigarrow \neg x_1 \wedge \neg x_2 \wedge \dots \wedge \neg x_n$$

$$F(1,1,0\dots0) = 1 \rightsquigarrow x_1 \wedge x_2 \wedge \neg x_2 \wedge \dots \wedge \neg x_n$$

$\vdots$

Beachte: Erfüllbarkeitsproblem bei KNF $\Leftrightarrow$ Jede(!) Klausel muß ein wahres Literal haben.

Erfüllbarkeitsproblem bei DNF $\Leftrightarrow$ Es gibt eine(!) Klausel, die wahr gemacht werden kann.

Erfüllbarkeitsproblem bei DNF leicht: Gibt es Klausel, die nicht x und $\neg x$ enthält.

Schwierig bei DNF: Gibt es Belegung, so dß 0 rauskommt.

Das ist nun wieder leicht bei KNF. (Wieso?!).

Eine weitere Einschränkung ist die k-KNF, k-DNF, $k = 1, 2, 3, \ldots$: Klauselgröße (= # Literale) $\leq k$ pro Klausel.

1-KNF: $x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge -$

2-KNF: $(x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_1 \vee x_1) \wedge (x_1 \vee \neg x_1)$

$x_1 \vee \neg x_1$ – tautologische Klausel, immer wahr, eigentlich unnötig.

3-KNF: $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_3) \wedge x_1 \wedge \ldots$

Eine unerfüllbare 1-KNF ist

$x_1 \wedge \neg x_1$

Eine unerfüllbare 2-KNF

$(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$.

Interessant ist, daß sich unerfüllbare Formeln auch durch geeignete Darstellung mathematischer Aussagen ergeben können.
Betrachten wir den Satz:
Ist $f: \{1,..,n\} \to \{1,..,n\}$ injektive Abbildung, dann auch surjektiv.
Dazu nehmen wir n^2 viele Variablen. Diese stellen wir uns so vor:

$x_{1,1}, x_{1,2}, x_{1,3}, \ldots, x_{1,m}$

$x_{2,1}, \ldots$

$\vdots$

$x_{n,m}$

Jede Belegung a der Variablen entspricht einer Menge von Paaren M:

$a(x_{i,j}) = 1 \Leftrightarrow (i,j) \in M$.

Eine Abbildung ist eine spezielle Menge von Paaren. Wir bekommen eine widersprüchliche Formel nach folgendem Prinzip:

① a stellt eine Abbildung dar

$\wedge$

② a ist injektive Abbildung

$\wedge$

③ a ist nicht surjektive Abbildung.

(1)
$$\left.\begin{array}{l}(x_{1,1} \vee x_{1,2} \vee \dots \vee x_{1,m}) \\ \wedge (x_{2,1} \vee x_{2,2} \vee \dots \vee x_{2,m}) \\ \wedge \\ \vdots \\ \wedge (x_{m,1} \vee x_{m,2} \vee \dots \vee x_{m,m})\end{array}\right\} \text{Jedem Element aus } 1,\dots,m \text{ ist eines zugeordnet, } m \text{ Klauseln}$$

$$\underbrace{(\neg x_{1,1} \vee \neg x_{1,2}) \wedge (\neg x_{1,1} \vee \neg x_{1,3}) \wedge \dots \wedge (\neg x_{1,m-1} \vee \neg x_{1,m})}_{\text{1 ist höchstens 1 Element zugeordnet, } \binom{m}{2} \text{ Klauseln}}$$

- Ebenso für $2,\dots,m$.

Damit haben wir, daß a eine Abbildung ist.

$$\underbrace{(\neg x_{1,1} \vee \neg x_{2,1}) \wedge (\neg x_{1,1} \vee x_{3,1}) \wedge (\neg x_{1,1} \vee \neg x_{4,1}) \wedge \dots \wedge (\neg x_{m-1,1} \vee \neg x_{m,1})}_{\text{1 wird höchstens von einem Element getroffen.}}$$

$\vdots$

Ebenso für $2,\dots,m$.

Haben jetzt: a injektive Abbildung

$$\bigwedge\Big((\neg x_{1,1} \wedge \neg x_{2,1} \wedge \neg x_{3,1} \wedge \dots \wedge \neg x_{m,1}) \vee (\neg x_{1,2} \wedge \neg x_{2,2} \quad \dots \quad) \vee \dots \vee (\neg x_{1,m} \wedge \quad \dots \wedge \neg x_{m,m}) \Big)$$

a nicht surjektiv

Im Falle des Erfüllbarkeitsproblems für KNF läßt sich der backtracking Algorithmus etwas verbessern. Dazu eine Bezeichnung:

$F_{|x=1}$ = x auf 1 setzen und F vereinfachen

= Klauseln mit x löschen (da oder);

in Klauseln mit $\neg x$ das $\neg x$ löschen

(da es falsch ist)

Analog $F_{|x=0}$

Es gilt:

F erfüllbar $\Leftrightarrow$

$F_{|x=1}$ oder $F_{|x=0}$ erfüllbar.

Der backtracking Algorithmus für KNF führt zur Davis Putnam Prozedur, die so aussieht:

Algorithmus (Davis-Putnam)

Eingabe: F in KNF, Variablen $x_1, \ldots, x_m$

array $a[1, \ldots, m]$ of boolean // Für die Belegung

DP (F) {

1. if F offensichtlich wahr // Leere Formel
   (1 return „F erfüllbar" // ohne Klauseln

2. if F offensichtlich unerfüllbar
   return „unerfüllbar" // Enthält
   // leere Klausel
   // oder Klauseln
   // x und $\neg x$

3. Wähle eine Variable x von F
   gemäß einer Heuristik
   Wähle (b_1, b_2) mit $b_1 = 1$, $b_2 = 0$
   oder $b_1 = 0$, $b_2 = 1$

4. $H := F|_{x=b_1}$; $a[x] := b_1$;

if DP(H) = „H erfüllbar"

return „F erfüllbar"

5. $H := F|_{x=b_2}$; $a[x] := b_2$; // Nur,

// wenn in 4. „unerfüllbar"

return DP(H)

}

Ist DP(F) = „F erfüllbar",

so ist in a die gefundene

erfüllende Belegung. $\square$

Korrektheit: Induktion über die

Variablen in F, wobei das

a noch einzubauen ist.

Algorithmus (Pure literal rule, unit clause rule)

In die einfache Davis Putnam Prozedur werden noch folgende Heuristiken in 3. eingebaut:

- Pure literal rule — x ist ein „pures Literal"

if es gibt ein x in F, so daß $\neg x$ nicht in F {

$H := F|_{x=1}$; $a[x] := 1$; return DP(H); }

if $\neg x$ in F aber x nicht in F {

$H := F|_{x=0}$; $a[x]=0$; return DP(H); }

// Hier keine backtracking.

- Unit clause rule

if es gibt eine Einerklausel (x) in F {

$H := F|_{x=1}$; $a[x] := 1$; return DP(H); }

if $(\neg x)$ in F {

$H := F|_{x=0}$; a[x] := 0; return DP(H);}

// Kein backtracking.

Erst danach geht das normale 3. des Davis-Putnam Algorithmus weiter. $\square$

Korrektheit:

- Pure literal: Es ist

$$F|_{x=1} \subseteq F,$$

heißt jede Klausel in $F|_{x=1}$ tritt auch in F auf. Damit gilt:

$F|_{x=1}$ erfüllbar $\Rightarrow$ F erfüllbar

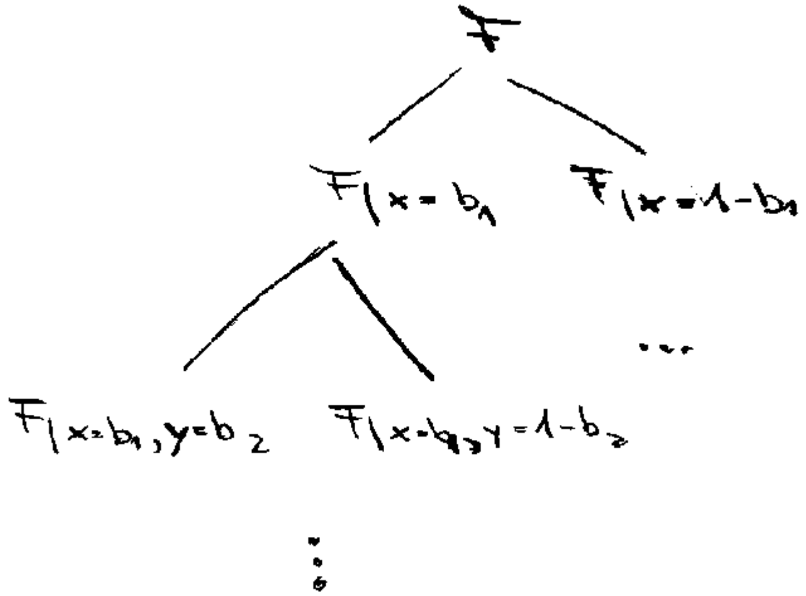
$F|_{x=1}$ unerfüllbar $\Rightarrow$ F unerfüllbar (wegen $F_{|x=1} \subseteq F$). Also

$F_{|x=1}$ erfüllbar $\Leftrightarrow$ F erfüllbar,

backtracking nicht nötig.

- Unit clause: $F_{|x=0}$ ist offensichtlich unerfüllbar, weg. leere Klausel, also kein backtracking nötig.

Laufzeit: Prozedurbaum



$b_1 = 1 \Leftrightarrow 1-b_1 = 0$

$b_1 = 0 \Leftrightarrow 1-b_1 = 1$

Sei

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

dann gilt:

$$T(n) \leq T(n-1) + T(n-1) \quad \text{für } n \gneq 1$$

$$T(1) = 2.$$

Dann das „neue $T(n)$" ($\geq$ „altes $T(n)$")

$$T(n) = T(n-1) + T(n-1), \quad \text{für } n \gneq 1$$

$$T(1) = 2$$

Hier sieht man direkt $T(n) = 2^n$.

Eine allgemeine Methode läuft so:

(10.27)

Machen wir den Ansatz (= eine Annahme), daß

$$T(n) = \alpha^n$$

für ein $\alpha \geq 1$ ist. Dann muß

für $n \geq 1$

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

sein. Also, teilen durch α^{n-1}

$$\alpha = 1 + 1 = 2.$$

Muß durch Induktion verifiziert werden, da der Ansatz nicht stimmen muß.

Zeit fürs Einsetzen

Damit Laufzeit

$$O\left(\left(2^n - 1 + 2^n\right) \cdot |F|\right) = O(2^n \cdot |F|)$$

10.28

Also: Obwohl backtracking ganze Stücke des Lösungsraums, also der Menge $\{0,1\}^m$ rausschneidet, können wir zunächst nichts Besseres als beim einfachen Durchgehen aller Belegungen zeigen.

Bei k-KNF's für festes k läßt sich der Davis Putnam Ansatz verbessern. Interessant ist, daß in gewissem Sinne k=3 reicht.

Konkret beim Erfüllbarkeitsproblem.