12 Kombinatorische Suche

Bisher haben wir meistens Probleme in polynomieller Zeit (und damit auch Platz) betrachtet. Obwohl es exponentiell viele mögliche Wege von $u \circ\!\!\longrightarrow\!\!\circ\, v$ 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 Fluß systematisch aufbauen, ohne blind durchzuprobieren. Dadurch erreichen wir polynomiale Zeit.

Jetzt betrachten wir Probleme, bei denen man die Lösung nicht mehr zielgerichtet aufbauen kann, sondern im Wesentlichen exponentiell viele Lösungskandidaten durchsuchen muss. Das bezeichnet man auch als *kombinatorische Suche*.

12.1 Aussagenlogische Probleme

Aussagenlogische Probleme, wie zum Beispiel

$$x \wedge y \vee \Big(\neg\big(u \wedge \neg(v \wedge \neg x)\big) \to y\Big), \quad x \vee y, \quad x \wedge y, \quad (x \vee y) \wedge (x \vee \neg y).$$

Also wir haben eine Menge von aussagenlogischen Variablen zur Verfügung, wie x, y, v, u. Formeln werden mittels der üblichen aussagenlogischen Operationen $\wedge$ (und), $\vee$ (oder, lat. vel), $\neg, \bar{}$ (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	0	0
1	1	1

Das heißt, das Ergebnis ist 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 $(x \to y) \wedge (y \to x)$.

Ein Beispiel verdeutlicht die Relevanz der Aussagenlogik:

Beispiel 12.1: Frage: *Worin besteht das Geheimnis Ihres langen Lebens?*

Antwort: *Folgender Diätplan wird eingehalten:*

- *Falls es kein Bier zum Essen gibt, dann wird in jedem Fall Fisch gegessen.*
- *Falls aber Fisch, gibt es auch Bier, dann aber keinesfalls Eis.*
- *Falls Eis oder auch kein Bier, dann gibts auch keinen Fisch.*

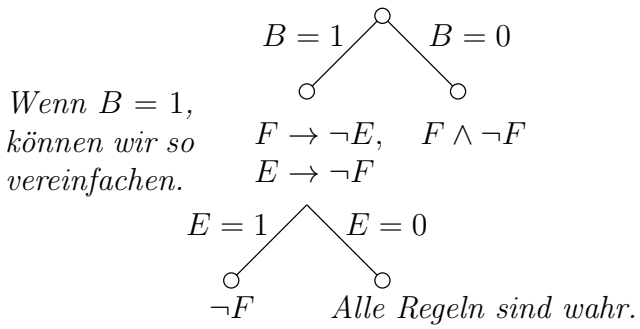
Wir wählen die folgenden aussagenlogischen Variablen:

$$\begin{aligned} B &= \textit{Bier beim Essen} \\ F &= \textit{Fisch} \\ E &= \textit{Eis} \end{aligned}$$

Ausagenlogisch werden obige Aussagen nun zu:

$$\begin{aligned} \neg B &\rightarrow F \\ F \wedge B &\rightarrow \neg E \\ E \vee \neg B &\rightarrow \neg F \end{aligned}$$

Die 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.

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

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. Das heißt wir haben eine Abbildung

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

Das heißt eine Belegung der Variablen mit Wahrheitswerten ist gegeben. Dann ist $a(F) =$ Wahrheitswert von F bei Belegung a.

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

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

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.

Definition 12.1: *Wir bezeichnen F als*

- erfüllbar*, genau dann, wenn es eine Belgung a mit $a(F) = 1$ gibt,*
- unerfüllbar *(widersprüchlich), genau dann, wenn für alle a $a(F) = 0$ ist,*
- tautologisch*, genau dann, wenn für alle a $a(F) = 1$ ist.*

Beachte: Ist F nicht tautologisch, so heißt das im Allgemeinen *nicht*, dass F unerfüllbar ist.

12.2 Aussagenlogisches Erfüllbarkeitsproblem

Wir betrachten zunächst einen einfachen Algorithmus zur Lösung des aussagenlogischen Erfüllbarkeitsproblems.

Algorithmus 25: Erfüllbarkeit

Input : Eine aussagenlogische Formel F.
Output : „erfüllbar“ falls F erfüllbar ist, sonst „unerfüllbar“.

```
1 foreach Belegung a ∈ {0,1}^n do        /* a(0...0),...,a(1...1) */
2 |  if a(F) = 1 then
3 |  |  return „erfüllbar durch a“;
4 |  end
5 end
6 return „unerfüllbar“;
```

Laufzeit: Bei n Variablen $O(2^n \cdot |F|)$, wobei $|F|$ = Größe von F ist.

Dabei muss F in einer geeigneten Datenstruktur vorliegen. Wenn F erfüllbar ist, kann die Zeit wesentlich geringer sein! Wir betrachten hier den reinen worst-case-Fall.

Eine Verbesserung kann durch Backtracking erreicht werden: Schrittweises Einsetzen der Belegung und Vereinfachen von F (*Davis-Putnam-Prozedur*, siehe später).

Die Semantik einer Formel F mit n Variablen lässt sich auch verstehen als eine Funktion $F : \{0,1\}^n \to \{0,1\}$.

Wieviele derartige Funktionen gibt es? Jede derartige Funktion lässt sich in *konjunktiver Normalform (KNF)* oder auch in *disjunktiver Normalform (DNF)* darstellen.

Eine Formel in *konjunktiver Normalform ist zum Beispiel*

$$F_{\text{KNF}} = (x_1 \vee \neg x_2 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee x_2),$$

und eine Formel in *disjunktiver Normalform ist*

$$F_{\text{DNF}} = (x_1 \wedge x_2 \wedge \neg x_3 \wedge x_4) \vee (x_2 \wedge x_3 \wedge \neg x_4) \vee (\neg x_1 \wedge x_5 \wedge \ldots \wedge x_n).$$

Im Prinzip reichen DNF oder KNF aus. Das heißt, ist $F : \{0,1\}^n \to \{0,1\}$ eine boolesche Funktion, so lässt sich F als KNF oder auch DNF darstellen.

KNF: Wir gehen 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:

$$\begin{aligned} F(0\ldots0) = 0 &\quad\to\quad x_1 \vee x_2 \vee \cdots \vee x_n \\ F(110\ldots0) = 0 &\quad\to\quad \neg x_1 \vee \neg x_2 \vee x_3 \vee \ldots \vee x_n \\ &\quad\vdots \end{aligned}$$

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

DNF: Alle $(b_1, \ldots, b_n) \in \{0,1\}^n$ für die $F(b_1, \ldots, b_n) = 1$ Klauseln analog zu oben:

$$\begin{aligned} F(0\ldots0) = 1 &\quad\to\quad \neg x_1 \wedge \neg x_2 \wedge \ldots \wedge \neg x_n \\ F(110\ldots0) = 1 &\quad\to\quad x_1 \wedge x_2 \wedge \neg x_3 \wedge \ldots \wedge \neg x_n \\ &\quad\vdots \end{aligned}$$

Die *Disjunktion* dieser Klauseln gibt eine Formel, die F darstellt.

Beachte:

- Erfüllbarkeitsproblem bei KNF $\iff$ *Jede* Klausel muss ein wahres Literal haben.
- Erfüllbarkeitsproblem bei DNF $\iff$ Es gibt *mindestens eine* Klausel, die wahr gemacht werden kann.

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

Schwierig bei DNF ist die folgende Frage: Gibt es eine Belegung, so dass 0 rauskommt? Das ist nun wieder leicht bei KNF. (Wieso?)

Eine weitere Einschränkung ist die k-KNF bzw. k-DNF, $k = 1, 2, 3, \ldots$. Hier ist die Klauselgröße auf $\leq k$ Literale pro Klausel beschränkt.

Beispiel 12.2:

$$\begin{array}{ll} 1\text{-}KNF: & x_1 \wedge \neg x_2 \wedge \neg x_3 \wedge \ldots \\ 2\text{-}KNF: & (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_1 \vee x_4) \wedge (x_1 \vee \neg x_1) \wedge \ldots \\ 3\text{-}KNF: & (x_1 \vee x_2 \vee x_3) \wedge (\neg x_2 \vee x_3) \wedge x_1 \wedge \ldots \end{array}$$

Die Klausel $(x_1 \vee \neg x_1)$ *oben ist eine tautologische Klausel. Solche Klauseln sind immer wahr und bei KNF eigentlich unnötig.*

Eine unerfüllbare 1*-KNF ist*

$$(x_1) \wedge (\neg x_1),$$

eine unerfüllbare 2*-KNF ist*

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

Interessant ist, dass sich unerfüllbare Formeln auch durch geeignete Darstellung mathematischer Aussagen ergeben können. Betrachten wir den Satz:

> Ist $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$ eine injektive Abbildung, dann ist diese Abbildung auch surjektiv.

Dazu nehmen wir n^2 viele Variablen. Diese stellen wir uns so vor:

$$\begin{array}{cccccc} x_{1,1}, & x_{1,2}, & x_{1,3}, & \ldots & x_{1,n} \\ x_{2,1}, & x_{2,2}, & \ldots & & x_{2,n} \\ \vdots & & & & \vdots \\ x_{n,1}, & & \ldots & & x_{n,n} \end{array}$$

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

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

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

1) a stellt eine Abbildung dar **und**

2) a ist eine injektive Abbildung **und**

3) a ist *keine* surjektive Abbildung. (Im Widerspruch zum Satz oben!)

Zu 1):

$$\left.\begin{array}{l}(x_{1,1} \vee x_{1,2} \vee \ldots \vee x_{1,n}) \\ \wedge \\ (x_{2,1} \vee x_{2,2} \vee \ldots \vee x_{2,n}) \\ \wedge \\ \vdots \\ \wedge \\ (x_{n,1} \vee x_{n,2} \vee \ldots \vee x_{n,n})\end{array}\right\} \text{Jedem Element aus } \{1,\ldots,n\} \text{ ist mindestens ein anderes zugeordnet, } n \text{ Klauseln.}$$

Das Element 1 auf der linken Seite ist höchstens einem Element zugeordnet.

$$(\neg x_{1,1} \vee \neg x_{1,2}) \wedge (\neg x_{1,1} \vee \neg x_{1,3}) \wedge \ldots \wedge (\neg x_{1,2} \vee \neg x_{1,3}) \wedge \ldots \wedge (\neg x_{1,n-1} \vee \neg x_{1,n})$$

Das ergibt $\binom{n}{2}$ Klauseln. (Sobald zum Beispiel $f : 1 \to 2$ und $f : 1 \to 3$ gilt, ist dieser Teil der Formel falsch!)

Analog haben wir auch Klauseln für $2,\ldots,n$. Damit haben wir gezeigt, dass a eine Abbildung ist.

Zu 2): Die 1 auf der rechten Seite wird höchstens von einem Element getroffen. Ebenso für $2,\ldots,n$.

$$(\neg x_{1,1} \vee \neg x_{2,1}) \wedge (\neg x_{1,1} \vee \neg x_{3,1}) \wedge (\neg x_{1,1} \vee \neg x_{4,1}) \wedge \ldots \wedge (\neg x_{1,1} \vee \neg x_{n,1})$$

Haben jetzt: a ist eine injektive Abbildung.

Zu 3):

$$\left.\begin{array}{l}\wedge \\ \Big((\neg x_{1,1} \wedge \neg x_{2,1} \wedge \neg x_{3,1} \wedge \ldots \wedge \neg x_{n,1}) \\ \vee \\ (\neg x_{1,2} \wedge \neg x_{2,2} \wedge \ldots \wedge \neg x_{n,2}) \\ \vdots \\ \vee \\ (\neg x_{1,n} \wedge \ldots \wedge \neg x_{n,n})\Big)\end{array}\right\} \begin{array}{l}a \text{ nicht surjektiv. Das heißt: 1 wir nicht getroffen oder 2 wird nicht getroffen} \\ \text{oder 3 usw.}\end{array}$$

Die so zusammengesetzte Formel ist unerfüllbar. Der zugehörige Beweis ist sehr lang und soll hier keine weitere Rolle spielen.

Kommen wir zurück zum Erfüllbarkeitsproblem bei KNF-Formeln.

12.2.1 Davis-Putnam-Prozedur

Im Falle des Erfüllungsproblems für KNF lässt sich der Backtracking-Algorithmus etwas verbessern. Dazu eine Bezeichnung:

$$F_{|x=1} \Rightarrow x \text{ auf 1 setzen und } F \text{ vereinfachen.}$$

Das führt dazu, dass Klauseln mit x gelöscht werden, da sie wahr sind. In Klauseln, in denen $\neg x$ vorkommt, kann das $\neg x$ entfernt werden, da es falsch ist und die Klausel nicht wahr machen kann. Analog für $F_{|x=0}$.

Es gilt:

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

Der Backtracking-Algorithmus für KNF führt zur *Davis-Putman-Prozedur*, die so aussieht:

Algorithmus 26: Davis-Putnam-Prozedur DP(F)

Input : Formel F in KNF mit Variablen $x_1, \ldots x_n$
Output : „erfüllbar“ mit erfüllender Belegung in a, falls F erfüllbar ist, sonst „unerüllbar“.
Data : Array $a[1, \ldots, n]$ für die Belegung

```
/* F = ∅, leere Formel ohne Klausel */
1  if F ist offensichtlich wahr then return „erfüllbar“;
/* {} ∈ F, enthält leere Klausel oder die Klauseln x und ¬x. */
2  if F ist offensichtlich falsch then return „unerfüllbar“;
3  Wähle eine Variable x von F gemäß einer Heuristik;
4  Wähle (b1, b2) = (1, 0) oder (b1, b2) = (0, 1);
5  H = F|x=b1;
6  a[x] = b1;
7  if DP(H)==„erfüllbar“ then
8  |  return „erfüllbar“;
9  else
10 |  H = F|x=b2;
11 |  a[x] = b2;
12 |  return DP(H);
13 end
```

Liefert DP(F) „erfüllbar“, so ist in a eine erfüllende Belegung gespeichert.

Korrektheit: Induktiv über die #Variablen in F, wobei das a noch einzubauen ist.

12.2.2 Heuristiken

In die einfache Davis-Putman-Prozedur wurden noch die folgednen Heuristiken eingebaut. Man wählt in den Schritten sieben und acht, die nächste Variable geschickter aus.

pure literal rule: Wenn es in F ein Literal x gibt, so dass $\neg x$ *nicht* in F vorkommt, dann setze $x = 1$ und mache ohne Backtracking weiter. Analog für $\neg x$.

unit clause rule: Wenn es in F eine Einerklausel (x) gibt, dann setze $x = 1$ und mache ohne Backtracking weiter. Analog für $\neg x$.

Erst danach geht die normale Davis-Putman-Algorithmus weiter.

Zur *Korrektheit:*

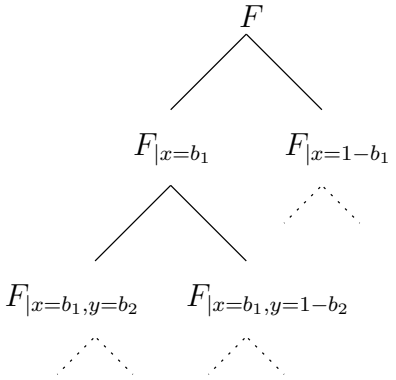
pure literal: Es ist $F_{|x=1} \subseteq F$, das heißt jede Klausel in $F_{|x=1}$ tritt auch in F auf. Damit gilt:

$$\begin{aligned} F_{|x=1} \text{ erfüllbar} &\Rightarrow F \text{ erfüllbar und} \\ F_{|x=1} \text{ unerfüllbar} &\Rightarrow F \text{ unerfüllbar (wegen } F_{|x=1} \subseteq F). \end{aligned}$$

Also ist $F_{|x=1}$ erfüllbar $\Longleftrightarrow$ F erfüllbar. Backtracking ist hier nicht nötig.

unit clause: $F_{|x=0}$ ist offensichtlich unerfüllbar, wegen leerer Klausel. Also ist kein backtracking nötig.

Laufzeit: Prozedurbaum



$$\begin{aligned} b = 0 &\Leftrightarrow 1 - b = 1 \\ b = 1 &\Leftrightarrow 1 - b = 0 \end{aligned}$$

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

$$\begin{aligned} T(1) &= 2 \\ T(n) &\leq T(n-1) + T(n-1) \quad \text{für } n \gneq 1 \end{aligned}$$

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

$$\begin{aligned} T(1) &= 2 \\ T(n) &= T(n-1) + T(n-1) \quad \text{für } n \gneq 1 \end{aligned}$$

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

Eine allgemeine Methode läuft so: Machen wir den Ansatz (das heißt eine Annahme), dass

$$T(n) = \alpha^n \text{ für ein } \alpha \geq 1$$

ist. Dann muss für $n > 1$

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

sein. Also teilen wir durch α^{n-1} und erhalten

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

Diese Annahme muss durch Induktion verifiziert werden, da der Ansatz nicht unbedingt stimmen muss.

Damit haben wir die Laufzeit $O(2^n \cdot |F|)$. $|F|$ ist die Zeit fürs Einsetzen.

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

Bei k-KNFs für ein festes k lässt sich der Davis-Putman-Ansatz jedoch verbessern. Interessant ist, dass in gewissem Sinne $k = 3$ reicht, wenn man sich auf die Erfüllbarkeit von Formeln beschränkt.

12.2.3 Erfüllbarkeitsäquivalente 3-KNF

Satz 12.1: *Sei F eine beliebige aussagenlogische Formel. Wir können zu F eine 3-KNF G konstruieren mit:*

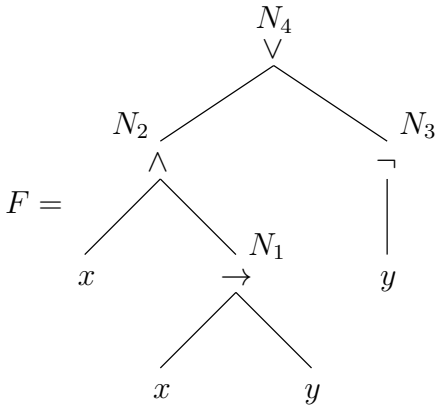
- *F erfüllbar $\Longleftrightarrow$ G erfüllbar. Man sagt: F und G sind erfüllbarkeitsäquivalent.*
- *Die Konstruktion lässt sich in der Zeit $O(|F|)$ implementieren. Insbesondere gilt $|G| = O(|F|)$.*

Beachte: Durch „Ausmultiplizieren“ lässt sich jedes F in ein *äquivalentes* G in KNF transformieren. Aber exponentielle Vergrößerung ist möglich und es entsteht keine 3-KNF.

Beweis. Am Beispiel wird eigentlich alles klar. Der Trick besteht in der Einfürhung neuer Variablen.

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

Schreiben F als Baum: Variablen $\to$ Blätter, Operatoren $\to$ innere Knoten.



N_1, N_2, N_3, N_4 sind die neuen Variablen. Für jeden inneren Knoten eine.

Die Idee ist, dass $N_i \Leftrightarrow$ Wert an den Knoten, bei gegebener Belegung der alten Variablen x und y.

Das drückt F' aus:

$$F' = \big(N_1 \Leftrightarrow (x \to y)\big) \wedge \big(N_2 \Leftrightarrow (x \wedge N_1)\big) \wedge \big(N_3 \Leftrightarrow \neg y\big) \wedge \big(N_4 \Leftrightarrow (N_3 \vee N_2)\big)$$

Die Formel F' ist immer erfüllbar. Wir brauchen nur die N_i von unten nach oben passend zu setzen. Aber

$$F'' = F' \wedge N_4, \quad N_4 \text{ Variable der Wurzel,}$$

erfordert, dass alle N_i richtig und $N_4 = 1$ 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, dass die $b(x)$, $b(y)$ eine erfüllende Belegug von F sind.

Eine 3-KNF zu F'' ist gemäß Prinzip auf Seite 180 zu bekommen, indem man es auf die einzelnen Äquivalenzen anwendet. □

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

Davis-Putman basiert auf dem Prinzip: Verzweigen nach dem Wert von x.

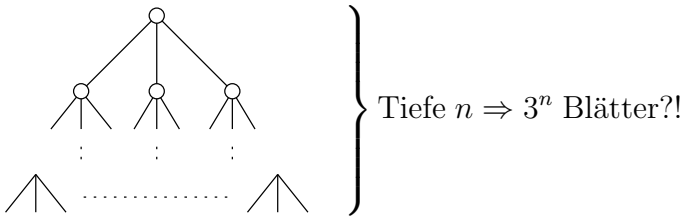
$$F \text{ erfüllbar } \iff F_{|x=1} \text{ oder } F_{|x=0} \text{ erfüllbar.}$$

Haben wir eine 3-KNF, etwa $F = \ldots \wedge (x \vee \neg y \vee z) \wedge \ldots$, dann ist

$$F \text{ erfüllbar} \iff F_{|x=1} \text{ oder } F_{|y=0} \text{ oder } F_{|z=1} \text{ erfüllbar.}$$

Wir können bei k-KNF Formeln auch nach den Klauseln verzweigen.

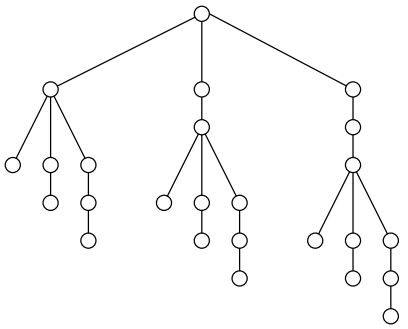
Ein Backtracking-Algorithmus nach diesem Muster führt zu einer Aufrufstruktur der Art:



Aber es gilt auch: (für die obige Beispielklausel)

F erfüllbar $\iff F_{|x=1}$ erfüllbar oder $F_{|x=0,y=0}$ erfüllbar oder $F_{|x=0,y=1,z=1}$ erfüllbar.

Damit ergibt sich die folgende Aufrufstruktur im Backtracking:



Sei wieder $T(n)$ = maximale #Blätter bei n Variablen, dann gilt:

$$\begin{aligned} T(n) &\leq d = O(1) \quad \text{für } n < n_0 \\ T(n) &\leq T(n-1) + T(n-2) + T(n-3) \quad \text{für } n \geq n_0,\ n_0 \text{ eine Konstante} \end{aligned}$$

Was ergibt das? Wir lösen $T(n) = T(n-1) + T(n-2) + T(n-3)$.

Ansatz:

$$T(n) = c \cdot \alpha^n \quad \text{für alle } n \text{ groß genug.}$$

Dann

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

also

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

Wir zeigen: Für $\alpha > 0$ mit $\alpha^3 = \alpha^2 + \alpha + 1$ gilt $T(n) = O(\alpha^n)$. Ist $n < n_0$, dann $T(n) \leq d \cdot \alpha^n$ für eine geeignete Konstante d, da $\alpha > 0$. Sei $n \geq n_0$.

Dann:

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + T(n-3) \\ &\leq d \cdot \alpha^{n-1} + d \cdot \alpha^{n-2} + d \cdot \alpha^{n-3} \\ &= d(\alpha^{n-3} \cdot \underbrace{(\alpha^2 + \alpha + 1)}_{=\alpha^3 \text{ nach Wahl von } \alpha}) \\ &= d \cdot \alpha^n = O(\alpha^n). \end{aligned}$$

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.

Frage: Wie genau geht das?

Bevor wir weiter verbessern, diskutieren wir, was derartige Verbesserungen bringen. Zunächst ist der exponentielle Teil maßgeblich (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 ist. Denn

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

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

Haben wir jetzt zwei Algorithmen

- A_1 Zeit 2^n
- A_2 Zeit $2^{n/2} = \left(\sqrt{2}\right)^n = (1.4\ldots)^n$

für dasselbe Problem. Betrachten wir eine vorgegebene Zeit x (fest). Mit A_1 können wir alle Eingaben der Größe n mit $2^n \leq x$, das heißt $n \leq \log_2 x$ in Zeit x sicher bearbeiten.

Mit A_2 dagegen alle mit $2^{n/2} \leq x$, das heißt $n \leq 2 \cdot \log_2 x$, also Vergrößerung um einen konstanten Faktor! Lassen wir dagegen A_1 auf einem neuen Rechner laufen, auf dem jede Instruktion 10-mal so schnell abläuft, so $\frac{1}{10} \cdot 2^n \leq x$, das heißt

$$n \leq log_2 10x = \log_2 x + log_2 10$$

nur die Addition einer Konstanten.

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

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

12.2.4 Monien-Speckenmeyer

Haben wir das Literal x *pure* in F (das heißt $\neg x$ ist nicht dabei), so reicht es, $F_{|x=1}$ zu betrachten. Analoges gilt, wenn wir mehrere Variablen gleichzeitig ersetzen.

Betrachten wir die Variablen $x_1, \ldots, x_k$ und Wahrheitswerte $b_1, \ldots, b_k$ für diese Variablen. Falls nun gilt

$$H = F_{|x_1=b_1, x_2=b_2, \ldots, x_k=b_k} \subseteq F,$$

das heißt jede Klausel C von H ist schon in F enthalten, dann gilt

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

Oder anders: Für jede Setzung $x_i = b_i$, die eine Klausel verkleinert, gibt es ein $x_j = b_j$, das diese Klausel wahr macht. Das ist wie beim Korrektheitsbeweis der *pure literal rule*. In diesem Fall ist kein Backtracking nötig.

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

$$F = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge G, \quad G \text{ ohne } x_1, x_2, x_3.$$

Dann $F_{|x_1=0, x_2=0, x_3=0} \subseteq F$, also ist $x_1 = 0, x_2 = 0, x_3 = 0$ autark für F. In G fallen höchstens Klauseln weg.

Algorithmus 27: MoSP(F)

Input : F in KNF.
Output : „erfüllbar“ / „unerfüllbar“

```
1  if F offensichtlich wahr then return „erfüllbar“;
2  if F offensichtlich falsch then return „unerfüllbar“;
3  C = (l_1 ∨ l_2 ∨ ··· ∨ l_s), eine kleinste Klausel in F;      /* l_i Literal */
   /* Betrachte die Belegungen */
4  b_1 = (l_1 = 1);                                          /* nur l_1 gesetzt */
5  b_2 = (l_1 = 0, l_2 = 1);                                 /* nur l_1 und l_2 gesetzt */
6  ...;
7  b_s = (l_1 = 0, l_2 = 0, ..., l_s = 1);
8  if eine der Belegungen autark in F then
9  |  b = eine autarke Belegung;
10 |  return MoSp(F_|b);
11 else
   |  /* Hier ist keine der Belegungen autark. */
12 |  foreach b_i do
13 |  |  if MoSp(F_|b_i) == „erfüllbar“ then return „erfüllbar“;
14 |  end
   |  /* Mit keiner der Belegungen erfüllbar. */
15 |  return „unerfüllbar“;
16 end
```

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

$$\begin{aligned} T(n) &= \#\text{Blätter, wenn in der kleinsten Klausel 3 Literale sind, und} \\ T'(n) &= \#\text{Blätter bei kleinster Klausel mit} \leq 2 \text{ Literalen.} \end{aligned}$$

Für $T(n)$ gilt:

$T(n) \leq T(n-1)$ Autarke Belegung gefunden, minestens 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, der Algorithmus nimmt immer die kleinste Klausel.

Ebenso bekommen wir

$T'(n) \leq T(n-1)$ wenn autarke Belegung.
$T'(n) \leq T'(n-1) + T'(n-2)$ Haben $\leq$ 2er Klausel ohne autarke Belegung.

Zwecks Verschwinden des $\leq$-Zeichens neue $T(n), T'(n) \leq$ die alten $T(n), T'(n)$.

$$\begin{aligned} T(n) &= d \quad \text{für } n \leq n_0,\ d \text{ Konstante} \\ T(n) &= \max\{T(n-1),\ T'(n-1) + T'(n-2) + T'(n-3)\} \quad \text{für } n > n_0 \\ T'(n) &= d \quad \text{für } n \leq n_0,\ d \text{ Konstante} \\ T'(n) &= \max\{T(n-1),\ T'(n-1) + T'(n-2)\} \quad \text{für } n > n_0. \end{aligned}$$

Zunächst müssen wir das Maximum loswerden. Dazu zeigen wir, dass für

$$\begin{aligned} S(n) &= d \quad \text{für } n \leq n_0, \\ S(n) &= S'(n-1) + S'(n-2) + S'(n-3) \quad \text{für } n > n_0, \\ S'(n) &= d \quad \text{für } n \leq n_0, \\ S'(n) &= S'(n-1) + S'(n-2) \quad \text{für } n > n_0 \end{aligned}$$

gilt, dass $T(n) \leq S(n)$, $T'(n) \leq S'(n)$.

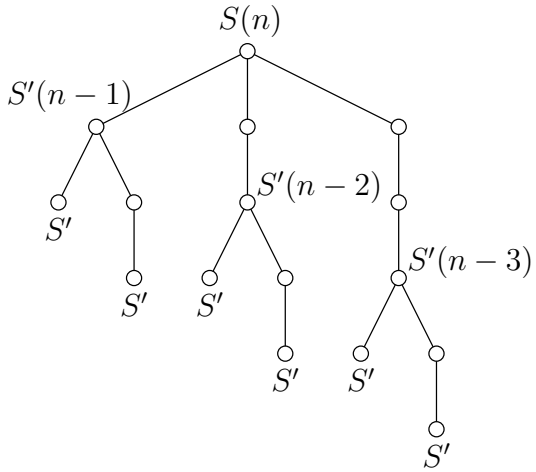
Induktionsanfang: ✓

Induktionsschluss:

$$\begin{aligned} T(n) &= \max\{T'(n-1) + T'(n-2) + T'(n-3)\} \\ &\quad \text{(nach Induktionsvoraussetzung)} \\ &\leq \max\{S(n-1),\ S'(n-1) + S'(n-2) + S'(n-3)\} \\ &\quad \text{(Monotonie)} \\ &\leq S'(n-1) + S'(n-2) + S'(n-3)\} = S(n). \end{aligned}$$

$$\begin{aligned} T'(n) &= \max\{T(n-1),\ T'(n-1) + T'(n-2)\} \\ &\quad \text{(nach Induktionsvoraussetzung)} \\ &\leq \max\{S(n-1),\ S'(n-1) + S'(n-2)\} \\ &= \max\{\overbrace{S'(n-2) + S'(n-3)}^{=S'(n-1)} + S'(n-4),\ S'(n-1) + S'(n-2)\} \\ &\quad \text{(Monotonie)} \\ &= S'(n-1) + S'(n-2) = S'(n). \end{aligned}$$

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



Ansatz: $S'(n) = \alpha^n$, $\alpha^n = \alpha^{n-1} + \alpha^{n-2} \Longrightarrow \alpha^2 = \alpha + 1$.

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

Induktionsanfanfang: $n \leq n_0$ ✓, da $\alpha > 0$.

Induktionsschluss:

$$\begin{aligned} S'(n+1) &= S'(n) + S'(n-1) \\ &\quad \text{(nach Induktionsvoraussetzung)} \\ &\leq c \cdot \alpha^n + c \cdot \alpha^{n-1} \\ &= c \cdot \alpha^{n-1}(\alpha + 1) \\ &\quad \text{(nach Wahl von } \alpha) \\ &= c \cdot \alpha^{n-1} \cdot \alpha^2 \\ &= c \cdot \alpha^{n+1} = O(\alpha^{n+1}). \end{aligned}$$

Dann ist auch $S(n) = O(\alpha^n)$ und die Laufzeit von *Monien-Speckenmayer* ist $O(\alpha^n \cdot |F|)$. Für F in 3-KNF ist $|F| \leq (2n)^3$, also Zeit $O(\alpha^{(1+\varepsilon)n})$ für n groß genug. Es ist $\alpha < 1.681$.

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

Definition 12.2: *Die Zahlen* $(F_n)_{n\geq 0}$ *mit*

$$\begin{aligned} F_0 &= 1, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \end{aligned}$$

für alle $n \geq 2$ *heißen Fibonacci-Zahlen. Es ist* $F_n = O(1.681^n)$.

12.2.5 Max-SAT

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

$$\big|\{i \mid a(C_i) = 1\}\big|$$

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

Satz 12.2: *Sei* $F = C_1 \wedge \ldots \wedge C_m$ *eine Formel gemäß Max-Ek-SAT. (*$C_i = C_j$ *ist zugelassen.) Es gibt eine Belegung* a*, so dass*

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

Beweis. Eine feste Klausel $C \subseteq \{C_1, \ldots, C_m\}$ wird von wievielen Belegungen *falsch* gemacht?

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

denn die k Literale der Klausel müssen alle auf 0 stehen. Wir haben wir folgende Situation vorliegen:

	a_1	a_2	a_3	$\ldots$	a_{2^n}
C_1	$*$		$*$		$*$
C_2		$*$			
$\vdots$			$\vdots$		
C_m		$*$		$*$	$*$

$* \Leftrightarrow a_i(C_j) = 1$, das heißt Klausel C_j wird unter der Belegung a_i wahr.
Pro Zeile gibt es $\geq 2^n - 2^{n-k} = 2^n(1 - 1/2^k)$ viele Sternchen.

Zeilenweise Summation ergibt:

$$m \cdot 2^n \cdot \left(1 - \frac{1}{2^k}\right) \leq \sum_{j=1}^{m} \left|\{a_i \mid a_i(C_j) = 1\}\right| = \#\text{ der } * \text{ insgesamt.}$$

Spaltenweise Summation ergibt:

$$\begin{aligned} \sum_{i=1}^{2^n} \left|\{j \mid a_i(C_j) = 1\}\right| &= \#\text{ der } * \text{ insgesamt} \\ &\geq 2^n \cdot m \cdot \left(1 - \frac{1}{2^k}\right) \end{aligned}$$

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

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

□

Wie findet man eine Belegung, so dass $\left|\{j \mid a(C_j) = 1\}\right|$ maximal ist? Letztlich durch Ausprobieren, sofern $k \geq 2$, also Zeit $O(|F| \cdot 2^n)$. (Für $k = 2$ geht es in $O(c^n)$ für ein $c < 2$.)

Wir haben also zwei ungleiche Brüder:

1. 2-SAT: polynomiale Zeit
2. Max-2-SAT: nur exponentielle Algorithmen vermutet.

Vergleiche bei Graphen mit Kantenlängen ≥ 0: kürzester Weg polynomial, längster kreisfreier Weg nur exponentiell bekannt.

Ebenfalls *Zweifärbbarkeit* ist polynomiell, für 3-Färbbarkeit sind nur exponentielle Algorithmen bekannt.

12.3 Maximaler und minimaler Schnitt

Ein ähnliches Phäomen liefert das Problem des maximalen Schnittes:

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), genau dann, wenn $v \in S_1, w \in S_2$ oder umgekehrt.

Es gilt: G ist 2-färbbar $\iff$ Es gibt einen Schnitt, so dass jede Kante in diesem Schnitt liegt.

Beim Problem des *maximalen Schnittes* geht es darum, einen Schnitt (S_1, S_2) zu finden, so dass

$$\big|\{e \in E \mid e \text{ liegt in } (S_1, S_2)\}\big|$$

maximal ist. Wie beim Max-Ek-SAT gilt:

Satz 12.3: *Zu $G = (V, E)$ gibt es einen Schnitt (S_1, S_2) so, dass*

$$\big|\{e \in E \mid e \textit{ liegt in } (S_1, S_2)\}\big| \geq \frac{|E|}{2}.$$

Beweis: Übungsaufgabe.

Algorithmus 28: $|E|/2$-Schnitt

Input : $G = (V, E)$, $V = \{1, \ldots, n\}$
Output : Findet den Schnitt, in dem $\geq |E|/2$ Kanten liegen.

```
1 S_1 = ∅;
2 S_2 = ∅;
3 for v = 1 to n do
4 |   if v hat mehr direkte Nachbarn in S_2 als in S_1 then
5 |   |   S_1 = S_1 ∪ {v};
6 |   else
7 |   |   S_2 = S_2 ∪ {v};
8 |   end
9 end
```

Laufzeit: $O(n^2)$, S_1 und 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 $v, w \in S_{1,j} \cup S_{2,j}$ liegt mindestens die Hälfte im Schnitt $(S_{1,j}, S_{2,j})$. Dann 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 (S_1, S_2), so dass

$$\left|\{e \in E \mid e \text{ liegt in } (S_1, S_2)\}\right|$$

minimal ist. Dieses Problem löst *Ford-Fulkerson* in polynomialer Zeit.

Frage: Wie? Übungsaufgabe.

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

- Minimaler Schnitt: polynomial,
- Maximaler Schnitt: nur exponentiell bekannt.

Die eben betrachteten Probleme sind *kombinatorische Optimierungsprobleme*. Deren bekanntestes ist das Problem des Handlungsreisenden. (*TSP, Travelling Salesmen Problem*)