Wettlauf um den schnellsten SAT-Algorithmus

Uwe Schöning

Universität Ulm
Abteilung Theoretische Informatik

schoenin@informatik.uni-ulm.de

Das 3-SAT-Problem:

$$F = (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee u \vee v) \wedge (\overline{y} \vee \overline{u}) \wedge (\overline{z} \vee v)$$

$\vee$ = oder

$\wedge$ = und

$\overline{\ }$ = nicht

Gibt es eine *erfüllende Belegung*?

Dies ist eine Wertzuweisung an die Variablen u, v, x, y, z mit Wahrheitswerten $\in \{0, 1\}$, so dass die Formel den Wert 1 erhält.

Wahrheitstafel aufstellen? Eine Wahrheitstafel hat bei einer Formel mit n Variablen insgesamt 2^n Zeilen...

Das 3-SAT-Problem ist NP-vollständig.

Es sind keine *effizienten* Algorithmen bekannt.

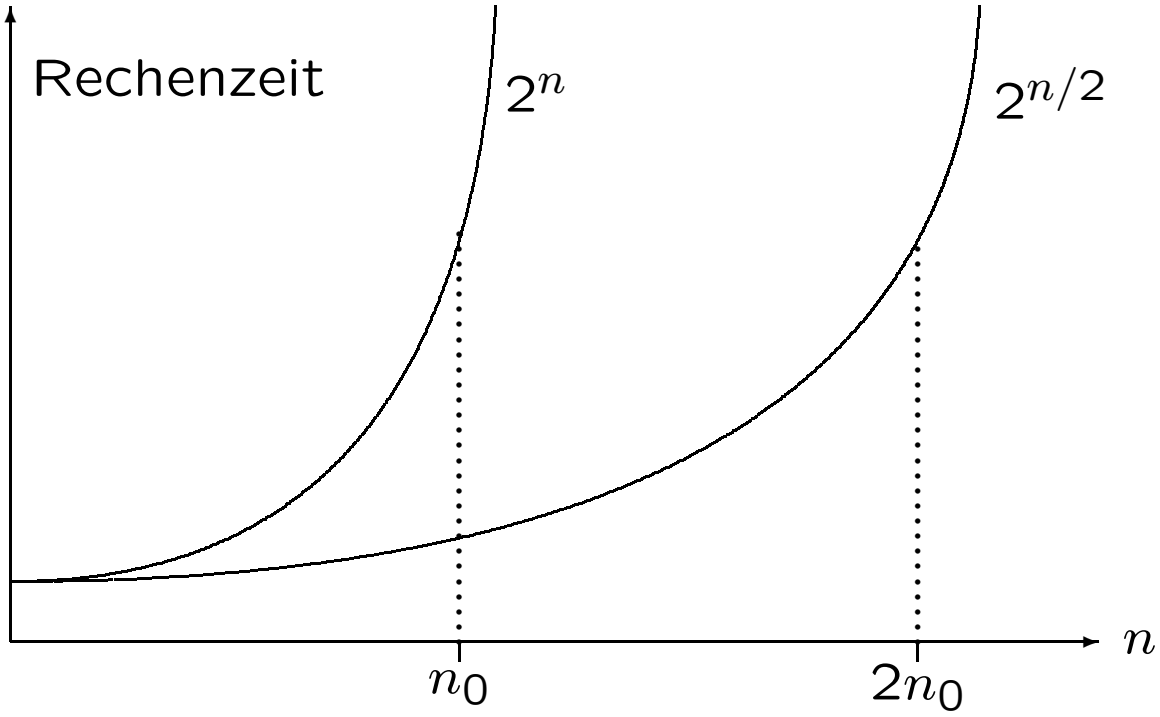
Effizienter Algorithmus = Algorithmus, dessen Laufzeit mit der *Eingabelänge* n (hier: n = Anzahl der Variablen) nur polynomial anwächst.

polynomial begrenzt sind: n, $n \log n$, n^2, n^5.

nicht polynomial begrenzt: n^n, 2^n, $n!$, 2^{2^n}.

Verbesserung von 2^n bis auf $1.414^n = 2^{n/2}$:

Es können Formeln mit doppelt so vielen Variablen in derselben Zeit gerechnet werden.



Zum Vergleich: Mit einem 10-mal schnelleren Rechner verschiebt sich n_0 nur zu $n_0 + \log_2 10$.

Deterministische Algorithmen für das 3-SAT-Problem:

2^n	Wahrheitstafel, backtracking
1.618^n	Monien, Speckenmeyer (1985)
1.505^n	Kullmann (1999)
1.481^n	Dantsin, Goerdt, Hirsch, Schöning (2000)

Probabilistische Algorithmen für das 3-SAT-Problem:

1.587^n	Paturi, Pudlák, Zane (1997)
1.5^n	Schöning (1998)
1.447^n	Paturi, Pudlák, Saks, Zane (1998)
1.3333^n	Schöning (1999)
1.3302^n	Hofmeister, Schöning, Schuler, Watanabe (2001)
1.3298^n	Baumer, Schöning, Schuler (2002)

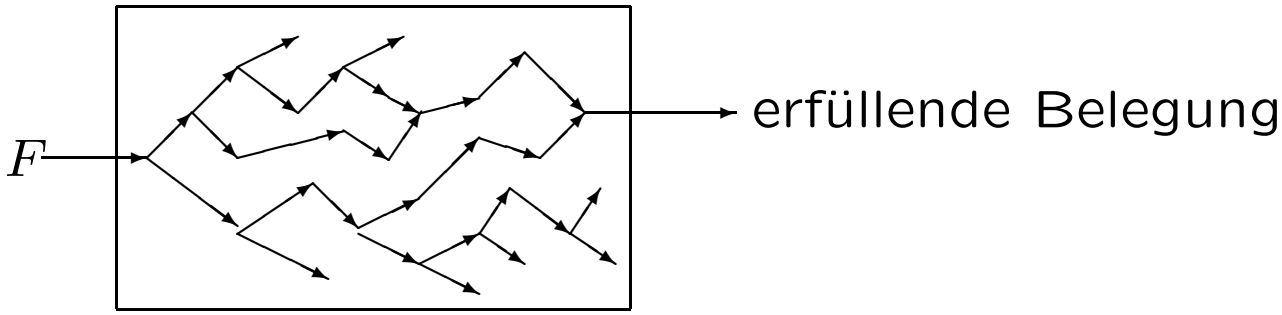
Ein Zahlenbeispiel

Sei $n = 80$. Ein ,,Rechenschritt'' dauere 10^{-8} Sekunden.

Wahrheitstafel-Methode hat Laufzeit 2^n :
benötigt $2^{80} \cdot 10^{-8}$ Sekunden = 400 Mio Jahre

Beste Methode hat Laufzeit $20 \cdot 1.33^n$:
benötigt $20 \cdot 1.33^{80} \cdot 10^{-8}$ Sekunden
= 27 Minuten.

Probabilistische Algorithmen:



Falls Wahrscheinlichkeit für das Finden einer erfüllenden Belegung $p = p(n)$ beträgt, so muss man einen solchen Algorithmus im Mittel $1/p$ mal wiederholen, bis man eine erfüllende Belegung findet.

Falls man nach $20/p$ Versuchen immer noch keine erfüllende Belegung gefunden hat, so beträgt die Wahrscheinlichkeit, dass doch eine erfüllende Belegung existiert, höchstens $(1-p)^{20/p} \leq e^{-20} = 0.000000002$.

Monien-Speckenmeyer:

PROC test(F: Formel): BOOL;
{Liefert TRUE, genau dann wenn F erfüllbar ist}
IF (F enthält keine Variablen mehr) THEN
    RETURN (Wert von F)
Sei $K = (x \vee y)$ eine Teilformel in F
IF test($F|_{x=1}$) THEN RETURN TRUE
    ELSE RETURN test($F|_{x=0,y=1}$)

Rekursionsgleichung für die Anzahl $T(n)$ der rekursiven Aufrufe:

$$\begin{aligned} T(0) &= 1 \\ T(1) &= 1 \\ T(n) &= T(n-1) + T(n-2) \quad \text{für } n > 1 \end{aligned}$$

Dies ist die so genannte Fibonacci-Folge.

Lösung: $T(n) \approx (\frac{1+\sqrt{5}}{2})^n = 1.618^n$.

Local Search von einer gegebenen Belegung a aus:

PROC test(a:Belegung,m:CARDINAL):BOOL;
{Liefert TRUE, genau dann wenn es eine
erfüllende Belegung a^* mit $d(a, a^*) \leq m$ gibt.
Hierbei ist $d(...)$ die Hamming-Distanz.}
IF $F(a) = 1$ THEN RETURN TRUE;
IF $m = 0$ THEN RETURN FALSE;
{Sei $K = (x \vee y \vee z)$ eine Teilformel
in F mit $K(a) = 0$}
IF test(a_x , $m - 1$) THEN RETURN TRUE;
IF test(a_y , $m - 1$) THEN RETURN TRUE;
IF test(a_z , $m - 1$) THEN RETURN TRUE;
RETURN FALSE;

Rekursionsgleichung für die Anzahl $T(m)$ der rekursiven Aufrufe:

$$\begin{aligned} T(0) &= 1 \\ T(m) &= 3T(m-1) \quad \text{für } n > 1 \end{aligned}$$

Lösung: $T(m) = 3^m$.

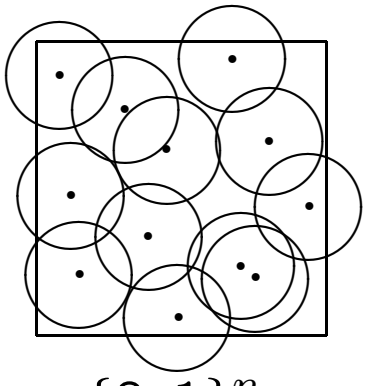
im Hauptprogramm:

Aufruf von test(0^n, $n/2$) und von test(1^n, $n/2$).

Rechenzeit in der Größenordnung
$3^{n/2} = 1.732^n$.

Verbesserung (probabilistischer Algorithmus):

```
FOR i := 1 TO t(n) DO
   Wähle eine zufällige Belegung a ∈ {0,1}^n;
   IF test(a, β·n) THEN RETURN TRUE;
RETURN FALSE
```



$\{0,1\}^n$

Laufzeit: $t(n) \cdot 3^{\beta n}$

Wie wählt man $t(n)$ und β, so dass die *Laufzeit*, aber auch die *Fehlerwahrscheinlichkeit*, minimiert wird?

Wahrscheinlichkeit, dass erfüllende Belegung a^* bei einem Versuch gefunden wird:

$$\frac{\sum_{i=0}^{\beta n} \binom{n}{i}}{2^n} \approx \frac{2^{h(\beta)n}}{2^n} = q^n$$

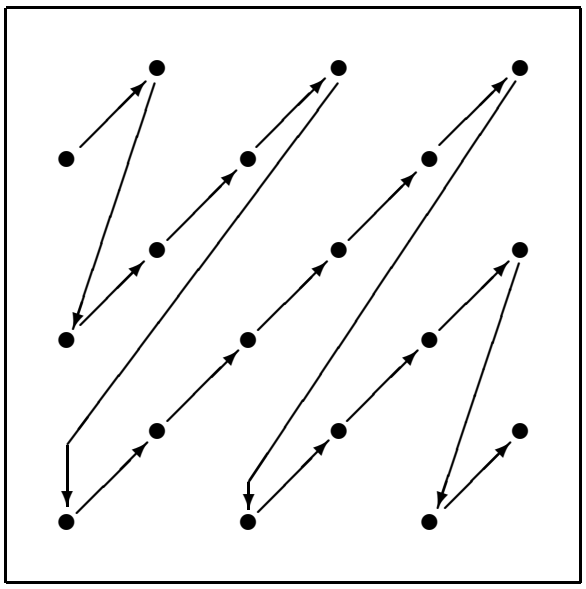
$\Rightarrow$ Notwendige Anzahl von Wiederholungen (Schleifendurchläufen) ist $t(n) = 20 \cdot (1/q)^n$. Damit ergibt sich als Laufzeit insgesamt:

$$20 \cdot (3^\beta/q)^n$$

Die optimale Wahl von β ergibt sich (durch Ableiten von $3^\beta/q$ nach β und auf Null setzen) als $\beta = 1/4$.

Dies ergibt eingesetzt eine Laufzeit in der Größenordnung 1.5^n.

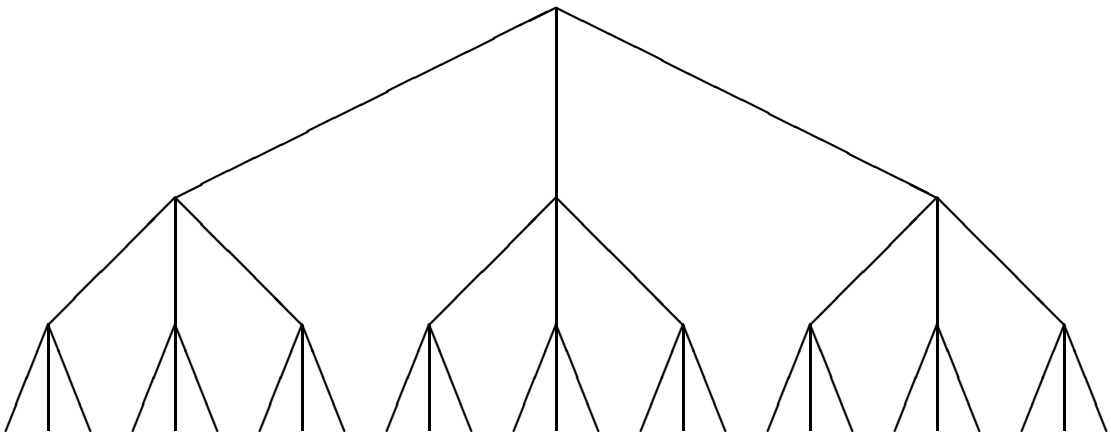
Dieser Algorithmus lässt sich *derandomisieren*, das heißt, zu einem deterministischen Algorithmus machen, indem man das zufällige Wählen der Anfangsbelegungen a ersetzt durch ein systematisches Durchlaufen von $a_1, a_2, \ldots, a_{t(n)}$, wobei diese 0-1-Wörter a_i einen *perfekten Überdeckungscode* bilden sollten.



$\{0,1\}^n$

Verbesserung auf 1.481^n**:**
[Dantsin,Goerdt,Hirsch,Schöning'2000]:

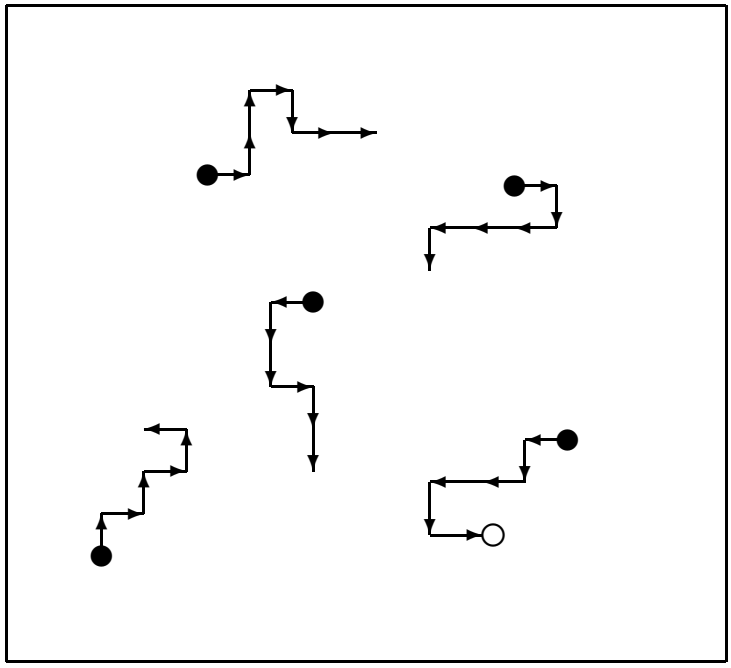
Betrachte den Rekursionsbaum bei 3 aufeinander folgenden Aufrufen von test:



Eine Fallanalyse zeigt, dass nur 24 dieser 27 Fälle wirklich auftreten bzw. betrachtet werden brauchen.

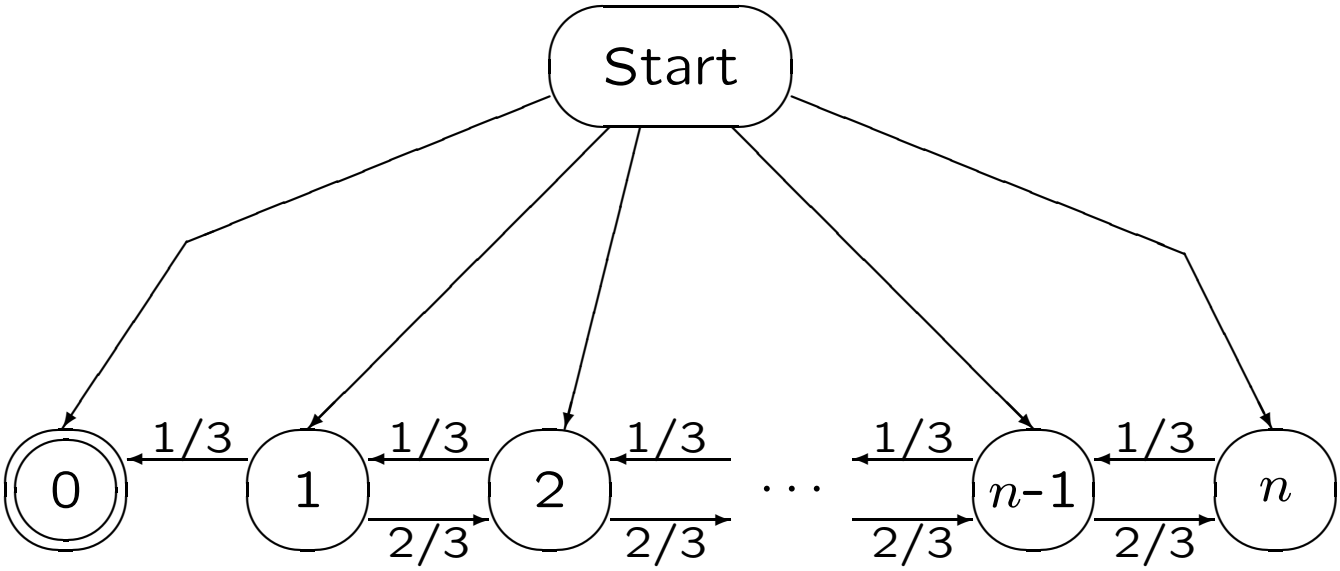
Probabilistische Variante: Ersetze die systematische Such-Prozedur test durch zufälliges Raten:

FOR $i := 1$ TO $t(n)$ DO
  Wähle eine zufällige Belegung $a \in \{0,1\}^n$;
  FOR $j := 1$ TO $3n$ DO
    IF $F(a) = 1$ THEN STOP;
    Wähle eine Teilformel K mit $K(a) = 0$ aus;
    {Sei $K = (x \vee y \vee z)$}
    Wähle zufällig $u \in \{x, y, z\}$ aus;
    $a := a_u$;



$\{0,1\}^n$

Modellvorstellung (Markov-Kette):



Zustandsnummer = Hammingabstand zwischen aktueller Belegung und der erfüllenden Belegung a^*.

Analyse:

Wahrscheinlichkeit, vom Zustand j aus (in $3j \leq 3n$ Schritten) den Zustand 0 zu erreichen, wobei j Schritte in die „falsche“ und $2j$ Schritte in die „richtige“ Richtung gehen, ist (mindestens)

$$\binom{3j}{j} \cdot (2/3)^j \cdot (1/3)^{2j}$$

$$\approx 2^{H(1/3)\cdot 3j} \cdot (2/3)^j \cdot (1/3)^{2j} = \boxed{(1/2)^j}$$

Sei

$$X_i = \begin{cases} 1, & \text{Übereinstimmung zwischen geratener und erfüllender Belegung im } i\text{-ten Bit} \\ 0, & \text{sonst} \end{cases}$$

Dann ist $d(a, a^*) = X_1 + \ldots + X_n$

und

$$\begin{aligned} & P(\text{ Erfolg }) = \sum_a P(a) \cdot (1/2)^{d(a,a^*)} \\ & = E[(1/2)^{X_1 + \cdots + X_n}] \\ & = E[\prod_i^n (1/2)^{X_i}] \\ & = \prod_i^n E[(1/2)^{X_i}] \\ & = E[(1/2)^{X_i}]^n \\ & = ((1/2) \cdot 1 + (1/2) \cdot (1/2))^n = (3/4)^n \end{aligned}$$

$\Rightarrow$ Komplexität $O((4/3)^n)$

Entscheidend ist $E[(1/2)^{\text{Hammingabstand}}]$.

Kann man Verteilung der Anfangsbelegungen so modifizieren, dass dieser Erwartungswert größer (und damit die Gesamt-Komplexität kleiner) wird ?

Beispiel: Wenn F die Klausel $(x \vee y \vee z)$ enthält so wird mit Wahrscheinlichkeit 1/8 auch die Anfangsbelegung $x = y = z = 0$ zugelassen. Kann man das verbessern?
Problem: Klauseln mit überlappenden Variablenmengen.

Ansatz: Finde eine maximale Teilmenge der Klauseln, die paarweise disjunkte Variablenmengen haben (per „Greedy“-Algorithmus).
Sei dies $K = \{K_1, \ldots, K_t\} \subseteq F$.

Entweder ist t „groß“$\Rightarrow$
Erzeuge zufällige Anfangsbelegung, wobei für die $3t$ Variablen, die in solchen Klauseln vorkommen, dies auf eine Weise geschieht, dass $x = y = z = 0$ vermieden wird. Dann Random Walk.

Oder t ist „klein“$\Rightarrow$
Dann kann man durch systematisches Belegen der betroffenen $3t$ Variablen die verbleibende Formel zu einer 2-KNF-Formel machen. Diese lässt sich dann in Polynomialzeit lösen.

$\rightarrow$ Komplexität $O(1.3302^n)$.

Weitere Verbesserung: (Zusätzlich) Paare von Klauseln betrachten, die sich in einer Variablen überlappen, aber unabhängig voneinander sind.

$\rightarrow$ Komplexität $O(1.3298^n)$.