4 Probabilistische Algorithmen

In diesem Kapitel werden probabilistische Algorithmen betrachtet, also solche, die zufällige Entscheidungen treffen. Typischerweise sind solche Algorithmen unvollständig, d.h., sie finden für erfüllbare Formeln mit hoher Wahrscheinlichkeit eine erfullende Bewertung, können aber nicht definitiv die Unerfüllbarkeit einer Formel feststellen.

Eine Klasse solcher Algorithmen verwendet eine lokale Verbesserungsstrategie. Diese Algorithmen arbeiten nach dem folgenden Schema:

wähle eine totale Bewertung α
repeat
    if $\alpha \models F$ then return α
    wähle Variable x
    $\alpha := \alpha$ with $[x \leftarrow 1 - \alpha(x)]$

Dies wird solange iteriert, bis entweder eine erfüllende Bewertung $\alpha \models F$ gefunden wird, oder bis hinreichend lange gesucht wurde, dass man mit einiger Sicherheit auf die Unerfüllbarkeit von F geschlossen werden kann.

Verschiedene Algorithmen unterscheiden sich darin, wie einerseits die Anfangsbewertungen α gewählt werden, und wie die Variable x, an der die Bewertung bei der Suche geändert wird, ausgewählt wird. Beide Auswahlen können deterministisch oder zufällig getroffen werden.

Werden die Stellen x, an der die Bewertung geändert wird, systematisch ausgewählt, aber die Anfangsbewertungen zufällig, so führt dies zu einem einfachen probabilistischen Algorithmus, dem *Hammingkugel*-Algorithmus. Eine derandomisierte Version davon, bei der auch die Anfangsbewertungen deterministisch ausgewählt werden, führt zum derzeit asymptotisch besten deterministischen Algorithmus für k-SAT: er erreicht bei 3-SAT die Komplexität $O(1.5^n)$, und kann durch geschickte Organisation der Suche so verbessert werden, dass er Ein e Laufzeit von $O(1.473^n)$ erreicht.

Werden dagegen beide Auswahlen zufällig getroffen, führt dies zum Irrfahrt-Algorithmus (*random walk*). Dieser ist mit einer Komplexität von $O((4/3)^n)$ der derzeit asymptotisch beste probabilistische Algorithmus für 3-SAT. Eine noch leicht verbesserte Version erreicht die Laufzeit $O(1.3302^n)$.

Schließlich betrachten wir noch einen probabilistischen Algorithmus, der der Vorgehensweise bei DLL ähnlicher ist als diese Algorithmen.

4.1 Hammingkugel-Algorithmus

Für zwei Bewertungen α, β ist die Hamming-Distanz

$$d(\alpha, \beta) := \left|\{x; \alpha(x) \neq \beta(x)\}\right| ,$$

die Hamming-Kugel vom Radius r um α ist

$$H(\alpha, r) := \{\, \beta \,;\, d(\alpha, \beta) \leq r \,\} .$$

Offenbar ist $|H(\alpha, r)| =: V(r) = \sum_{i=0}^{r} \binom{n}{r}$ unabhängig von α. Sei $\rho := r/n$ der normierte Radius. Es ist eine aus der Informationstheorie bekannte Tatsache, dass für $0 < \rho \leq 1/2$ gilt

$$\frac{1}{\sqrt{8n\rho(1-\rho)}} 2^{h(\rho)n} \leq V(r) \leq 2^{h(\rho)n} ,$$

wobei $h(\rho) := -\rho \log \rho - (1-\rho) \log(1-\rho)$ die Entropiefunktion ist. Ist also ρ konstant, so ist bis auf einen polynomiellen Faktor $V(r) \approx 2^{h(\rho)n}$.

Der folgende Algorithmus durchsucht die Hammingkugel $H(\alpha, r)$ nach einer erfüllenden Bewertung.

```
suche(α, r)
  if α ⊨ F
     then return α
  if r = 0
     then return NIL
  wähle C = a_1 ∨ ... ∨ a_k mit Cα = 0
  for i := 1 to k do
     α_i := α with [a_i ← 1]
     β := suche(α_i, r − 1)
     if β ≠ NIL
        then return β
  return NIL
```

Der Zeitaufwand ist offenbar k^r, da ein k-fach verzweigender Suchbaum der Tiefe r aufgespannt wird. Dies ist unter Umständen weniger als das Volumen $V(r)$. So ist z.B. für $k = 3$ und $r = n/2$ das Volumen $V(r) \geq 2^{n-1}$, aber $k^r = \sqrt{3}^n \geq 1.7321^n$.

Damit ergibt sich ein einfacher deterministischer Algorithmus für 3-SAT, der für die Wertzuweisungen α_0, die alle Variablen mit 0 bewertet, und α_1, die alle mit 1 bewertet, nacheinander suche($\alpha_0, n/2$) und suche($\alpha_0, n/2$) aufruft. Die Komplexität ist $O(\sqrt{3}^n) = O(1.7321^n)$.

Eine bessere Komplexität erreicht man, wenn man einen kleineren Radius $r = \rho n$ für ein $\rho < \frac{1}{2}$ wählt, und dafür mehrere Startpunkte zufällig wählt. Damit erhält man die folgende probabilistische Variante des Hammingkugel-Algorithmus:

```
for i := 1 to w do
      wähle α_i zufällig
      β := suche(α_i, ρn)
      if β ≠ NIL
          then return β
  return NIL
```

Dabei werden die Anfangsbewertungen α_i gleichverteilt und unabhängig aus den 2^n möglichen Bewertungen der n Variablen gezogen.

Die Komplexität ist offenbar $w \cdot k^{\rho n}$. Es gilt nun, die Parameter w und ρ so zu wählen, dass die Fehlerwahrscheinlichkeit vernachlässigbar klein wird, und dabei die Laufzeit minimiert wird.

Der einzige mögliche Fehler ist, dass die Eingabeformel erfüllbar ist, aber der Algorithmus keine erfüllende Bewertung findet.

Sei also $\alpha \models F$. Ein Aufruf von suche$(\alpha_i, \rho n)$ findet α, falls $\alpha_i \in H(\alpha, r)$ ist, was mit Wahrscheinlichkeit $V(\rho n)2^{-n}$ geschieht. Diese Wahrscheinlichkeit ist ungefähr $2^{-(1-h(\rho))n}$.

Die Wahrscheinlichkeit, dass α bei w unabhängigen Wiederholungen nicht gefunden wird, ist also höchstens

$$\left(1 - 2^{-(1-h(\rho))n}\right)^w \;\leq\; e^{-w \cdot 2^{-(1-h(\rho))n}} .$$

Also werden $w = c \cdot 2^{-(1-h(\rho))n}$ Wiederholungen gebraucht, um die Fehlerwahrscheinlichkeit unter e^{-c} zu drücken. Daher ist die Laufzeit, bis auf einen polynomiellen Faktor

$$\left(2^{-(1-h(\rho))n} \cdot k^{\rho}\right)^n = \left(2\rho^{\rho}(1-\rho)^{(1-\rho)}k^{\rho}\right)^n .$$

Indem man die Basis nach ρ differenziert und das Ergebnis nach ρ auflöst, erhält man, dass der Ausdruck für $\rho = \frac{1}{k+1}$ minimal wird. Durch einsetzen und elementare Vereinfachungen erhält man dann die Basis $\frac{2k}{k+1}$. Die Komplexität des probabilistischen Algorithmus ist also $\left(\frac{2k}{k+1}\right)^n$. Diese Basis ist für kleine Werte von k:

k	3	4	5	6	7	8
$\frac{2k}{k+1}$	1.5	1.6	1.66667	1.71429	1.75	1.77778

Derandomisierung mit Überdeckungscodes

Bei der probabilistischen Version des Hammingkugel-Algorithmus sind die einzigen zufälligen Entscheidungen, die getroffen werden, die Wahl der Anfangsbewertungen α_i. Um einen deterministischen Algorithmus zu gewinnen, wird der Begriff des Überdeckungscodes verwendet.

Definition. *Eine Menge* C *von Bewertungen ist ein* Überdeckungscode *vom Radius* r*, falls für jede Bewertung* α *gilt* $\alpha \in H(\gamma, r)$ *für ein* $\gamma \in C$.

So ist zum Beispiel die oben betrachtete Menge $\{\alpha_0, \alpha_1\}$ aus den beiden konstanten Bewertungen ein Überdeckungscode vom Radius $n/2$.

Hat man einen Überdeckungscode C vom Radius r zur Verfügung, so hat man den folgenden deterministischen Algorithmus für k-SAT, mit Zeitkomplexität $|C| \cdot k^r$:

```
for each γ ∈ C do
    β := suche(γ, r)
    if β ≠ NIL
        then return β
return NIL
```

Um also einen deterministischen Algorithmus zu erhalten, brauchen wir ein Verfahren, um deterministisch einen Überdeckungscode zu konstruieren.

Das Problem, einen möglichst kleinen solchen Code zu konstruieren, ist ein Spezialfall des bekannten NP-schweren Optimierungsproblems SET COVER:

Gegeben:	Eine Menge S, und eine Familie $F \subseteq 2^S$ von Teilmengen von S
Gesucht:	Eine Teilfamilie $C \subseteq F$, die S überdeckt, d.h. $S = \bigcup C$.
Ziel:	Minimiere $|C|$.

In unserem Fall ist S die Menge 2^n möglichen Bewertungen, und F die Menge der Hammingkugeln vom Radius r.

Der folgende Greedy-Algorithmus löst das Problem Set Cover: Ein Element $s \in S$ heißt unüberdeckt, falls $s \notin \bigcup C$ ist.

Am Anfang wird $C = \emptyset$ gesetzt. Solange es noch unüberdeckte Elemente in S gibt, wähle ein $c \in F$, dass maximal viele unüberdeckte Elemente enthält, und füge es zi C hinzu.

Falls am Ende der Schleife $\bigcup C = S$ ist, ist C eine Lösung, andernfalls gibt es keine, d.h. $S \setminus \bigcup F \neq \emptyset$.

Es ist eine bekannte Tatsache, dass dieser Greedy-Algorithmus ein $\log|S|$-Approximationsverfahren für Set Cover ist, d.h. er liefert stets eine Lösung C mit $|C| \leq (\log|S|) \cdot |C^*|$, wobei C^* die optimale Lösung, also eine kleinstmögliche Überdeckung, ist.

In unserem Fall ist $\log|S| = O(n)$, also liefert der Greedy-Algorithmus einen Überdeckungscode, der nur um einen Faktor n größer ist als der minimale. Um die Größe, und damit die Komplexität des obigen deterministischen Verfahrens, abschätzen zu können, brauchen wir noch eine obere Schranke an die Größe eines minimalen Überdeckungscodes.

Diese liefert das folgende Lemma. Aus Kardinalitätsgründen muss jeder Überdeckungscode C die Größe $|C| \geq 2^n/V(r)$ haben. Man kann beweisen, dass immer ein Code existiert, der nur um einen Faktor n größer ist.

Lemma 25. *Für alle n und $r = \rho n$ mit $0 < \rho \leq 1/2$ gibt es einen Überdeckungscode vom Radius r der Größe*

$$s := n \cdot \sqrt{8n\rho(1-\rho)} \cdot 2^{(1-h(\rho))n} .$$

Beweis. Die Aussage wird mit der probabilistischen Methode bewiesen: es wird gezeigt, dass eine zufällig gewählte Menge von s Bewertungen mit nicht verschwindender Wahrscheinlichkeit ein Überdeckungscode ist. Daher muss es insbesondere einen solchen geben.

Sei also β eine feste Bewertung, und α eine zufällig gewählte. Dann ist $\beta \in H(\alpha, r)$ mit Wahrscheinlichkeit $V(r)/2^n$.

Ist also C eine Menge von s unabhängig zufällig gewählten Bewertungen, so ist die Wahrscheinlichkeit, dass β von C nicht überdeckt wird, höchstens

$$(1 - \frac{V(r)}{2^n})^s \leq e^{-s\frac{V(r)}{2^n}} \leq e^{-s\frac{2^{(h(\rho)-1)n}}{\sqrt{8n\rho(1-\rho)}}} \leq e^{-n} .$$

Somit ist die Wahrscheinlichkeit, dass irgendeine Bewertung nicht überdeckt wird, höchstens $2^n \cdot e^{-}n < 1$, also ist C mit positiver Wahrscheinlichkeit ein Überdeckungscode. □

Da es stets einen Überdeckungscode dieser Größe s gibt, ist insbesondere der optimale Code C^* von der Größe $|C^*| \leq s$, und damit liefert der Greedy-Algorithmus einen Überdeckungscode der Größe

$$O(n) \cdot s = O(n^{5/2}) \cdot 2^{(1-h(\rho))n} .$$

Allerdings ist die Laufzeit des Greedy-Algorithmus für unsere Zwecke zu groß: selbst im günstigsten Fall braucht er mindestens $2^{(1-h(\rho))n}$ Auswahlen, und bei jeder Auswahl muss die Zahl der unüberdeckten Bewertungen für alle 2^n Hammingkugeln aktualisiert werden, die Laufzeit ist also selbst bei günstigster Implementierung größer als 2^n. Andererseits läuft der Greedy-Algorithmus aber auch, selbst in der einfachsten möpglichen Implementierung, in Zeit 2^{3n}.

Die Lösung besteht in der folgenden einfachen Beobachtung: ist C ein Überdeckungscode vom Radius r für n Variablen, so gewinnt man daraus einen ebensolchen für dn Variablen, indem man die Variablen in d Blöcke mit je n Variablen partitioniert, und alle Bewertungen betrachtet, in denen die Variablen in den Blöcken unabhängig mit allen Bewertungen aus C bewertet werden. Dieser Code C^d hat die Größe $|C|^d$ und den Radius $r \cdot d$.

Es wird also der Greedy-Algorithmus benutzt, um einen Überdeckungscode C für $n/6$ Variablen von Radius $\rho n/6$ zu finden. Dies braucht die Zeit $2^{3n/6} = \sqrt{2}^n$. Dann benutzt man den Code C^6, der die Größe

$$\left[O\left(\left(\frac{n}{6}\right)^{5/2}\right) \cdot 2^{(1-h(\rho))\frac{n}{6}}\right]^6 = O(n^{15}) \cdot 2^{(1-h(\rho))n}$$

hat. Die Gesamtlaufzeit des deterministischen Algorithmus setzt sich also zusammen aus der Zeit $\sqrt{2}^n$ für den Greedy-Algorithmus zum Auffinden des Überdeckungscodes C, plus der Zeit $\left(2^{1-h(\rho)}k^\rho\right)^n$ zum Durchsuchen der Hammingkugeln vom Radius ρn um die Bewertungen im Code C^6. Der deterministische Algorithmus hat also im wesentlichen die gleiche Zeitkomplexität wie der probabilistische Hammingkugel-Algorithmus, nämlich $\left(\frac{2k}{k+1}\right)^n$ für k-SAT.

Der Nachteil dieser Methode ist, dass der Greedy-Algorithmus zur Konstruktion des Überdeckungscodes exponentiell viel Speicherplatz braucht. Es gibt eine zweite Methode, die ohne zusätzlichen Speicherbedarf auskommt, aber dafür eine etwas schlechtere Zeitkomplexität liefert.

Für eine gegebenes $\delta > 0$ wähle ein kleinstmögliches b so dass folgende Ungleichung gilt:

$$b\sqrt{8b\rho(1-\rho)}2^{(1-h(\rho))b} \leq 2^{(1-h(\rho)+\delta)b}$$

Dann existiert nach Lemma 25 ein Überdeckungscode C_b für b Variablen vom Radius ρb der Größe $2^{(1-h(\rho)+\delta)b}$. Dieser endliche Code, den man etwa durch einmalige exhaustive Suche findet, wird im Algorithmus festverdrahtet. Man gewinnt daraus den Code $C_b^{n/b}$ für n Variablen vom Radius ρn, dieser hat die Größe $2^{(1-h(\rho)+\delta)n}$.

Es sei nun ein $\epsilon > 0$ vorgegeben. Dann setzt man $\delta := \log\left(\frac{k+1}{2k}\epsilon + 1\right)$, und benutzt den Code C_b für das zu diesem δ gehörige b wie oben. Die Laufzeit des deterministischen Algorithmus berechnet sich nun wie oben als $\left(2^{1-h(\rho)+\delta} \cdot k^\rho\right)^n$, und mit der Setzung $\rho := 1/(k+1)$ erhält man, dass sie

Basis sich errechnet zu

$$\begin{aligned}
\left(2^{1-h(\rho)+\delta} \cdot k^{\rho}\right) &= \left(2^{1+\frac{1}{k+1}\log(\frac{1}{k+1})+\frac{k}{k+1}\log(\frac{k}{k+1})+\delta} \cdot k^{\frac{1}{k+1}}\right) \\
&= \left(2^{1-\frac{1}{k+1}\log(k+1)+\frac{k}{k+1}\log k-\frac{k}{k+1}\log(k+1)+\delta} \cdot 2^{\frac{1}{k+1}\log k}\right) \\
&= 2^{1+\log k-\log(k+1)+\delta} = 2^{1+\log(\frac{k}{k+1})+\delta} \\
&= 2\frac{k}{k+1}\left(\frac{k+1}{2k}\epsilon+1\right) = \frac{2k}{k+1}+\epsilon
\end{aligned}$$

Also erhalten wir als Komplexität den Wert $\left(\frac{2k}{k+1}+\epsilon\right)^n$, für beliebig kleines $\epsilon > 0$.

Für k-SAT mit $k \geq 4$ ist dies der beste bekannte deterministische Algorithmus. Für 3-SAT ist die Komplexität $(1.5+\epsilon)^n$, die von dem oben erwähnten Algorithmus von Schiermeyer unterboten wird. Es ist aber möglich, die Prozedur suche(α, r) im Falle von 3-SAT geschickter zu implementieren, so dass sie statt 3^r nur eine Laufzeit von 2.848^r benötigt. Durch Verwendung dieser verbesserten Prozedur erreicht der deterministische Hammingkugel-Algorithmus die Komplexität 1.481^n, und ist damit auch der beste bekannte deterministische Algorithmus für 3-SAT.

4.2 Zufallsirrfahrt

Ein anderer Ansatz besteht darin, nicht nur die Anfangsbewertungen zufälig zu wählen, sondern auch die Verbesserungsschritte. Die innere Schleife führt also eine zufällige Irrfahrt (random walk) im Raum der möglichen Bewertungen durch. Der folgende Algorithmus setzt diese Idee um:

```
repeat w times
    wähle α zufällig
    repeat 3n times
        if α ⊨ F
            then return α
        wähle C = a_1 ∨ ... ∨ a_k mit Cα = 0
        wähle i zufällig aus {1, ..., k}
        α := α with [a_i ← 1]
return NIL
```

Dabei werden die Anfangsbewertungen α wiededrum gleichverteilt und unabhängig gewählt.

Nun ist die Anzahl der Wiederholungen w so zu wählen, dass die Fehlerwahrscheinlichkeit klein wird. Da der Algorithmus nur dann einen Fehler machen, wenn F erfüllbar ist, sei wieder α^* eine fest gewählte Bewertung mit $\alpha^* \models F$.

Wir berechnen die Wahrscheinlichkeit $p(n)$ dafür, dass ein Durchlauf der inneren Schleife α^* findet. Dann ergibt sich die Fehlerwahrscheinlichkeit als $(1-p(n))^w \le e^{-wp(n)}$, und somit reichen $w = c\frac{1}{p(n)}$ Wiederholungen aus, damit die Fehlerwahrscheinlichkeit höchstens e^{-c} beträgt.

Da $\alpha^* \models F$, gibt es in jeder Klausel $C = (a_1 \vee \ldots \vee a_k)$ in F mindestens ein Literal a_i mit $\alpha^*(a_i) = 1$. Für jede Klausel C wählen wir nun ein festes Literal a_C mit $\alpha^*(a_C) = 1$.

Zur Analyse der Erfolgswahrscheinlichkeit $p(n)$ eines Schleifendurchlaufs wird nun eine Zufallsvariable Z definiert. Bei der Wahl der Anfangsbewertung α ist Z die Hamming-Distanz zwischen α und α^*, also

$$Z := d(\alpha, \alpha^*) \ .$$

Wird in einem Durchlauf der Schleife das Literal a_i aus der Klausel C gewählt, so wird gesetzt

$$Z := \begin{cases} Z-1 & \text{falls } a_i = a_C \\ Z+1 & \text{sonst.} \end{cases}$$

Wird $a_i = a_C$ gewählt, so verringert sich die Distanz $d(\alpha, \alpha^*)$ um 1, da $\alpha^*(a_C) = 1$ ist. Wird dagegen ein Literal $a_i \neq a_C$ gewählt, so wird die Distanz $d(\alpha, \alpha^*)$ entweder um 1 kleiner (falls $\alpha^*(a_i) = 1$ ist) oder um 1 größer (falls $\alpha^*(a_i) = 0$ ist). Es gilt also stets $Z \ge d(\alpha, \alpha^*)$.

Daher ist die Wahrscheinlichkeit, dass α^* gefunden wird, mindestens so gross wie die Wahrscheinlichkeit, dass $Z = 0$ erreicht wird. Diese Wahrscheinlichkeit ist aber wesentlich leichter auszurechnen, als direkt die Wahrscheinlichkeit zu berechnen, dass $\alpha = \alpha^*$ wird, da sich die Zufallsvariable Z sehr einfach verhält.

Man kann sich Z als den Zustand in einem probabilistischen Automaten (einer sog. Markov-Kette), wie in Abbildung 3 dargestellt, vorstellen. Der Automat startet in einem zufällig zwischen $Z = 0$ und $Z = n$ gewählten Anfangszustand, und geht dann vom einem Zustand Z stets mit Wahrscheinlichkeit $\frac{1}{k}$ in den Zustand $Z-1$ und mit Wahrscheinlichkeit $1-\frac{1}{k}$ in den Zustand $Z+1$ über.

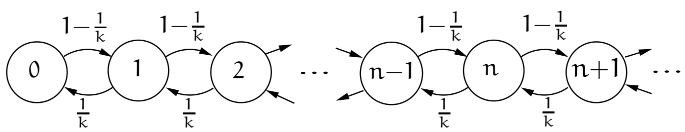

Abbildung 3: Die Markov-Kette zum Irrfahrt-Algorithmus

Der Wahrscheinlichkeitsverteilung des Anfangszustand lässt sich nun leicht ausrechnen, da α gleichverteilt gewählt wird, und am Anfang $Z = d(\alpha, \alpha^*)$ ist. Es gibt $\binom{n}{j}$ Bewertungen, die von α^* die Distanz j haben, und jede davon wird mit Wahrscheinlichkeit 2^{-n} gewählt. Der Automat startet daher mit Wahrscheinlichkeit $2^{-n}\binom{n}{j}$ im Zustand $Z = j$.

Es bezeichne nun p_j die Wahrscheinlichkeit, dass der Zustand $Z = 0$ ausgehend von Anfangszustand $Z = j$ erreicht wird. Mit Hilfe der Theorie der Markov-Ketten errechnet man den Wert $p_j = \left(\frac{1}{k-1}\right)^j$. Daher errechnet sich die Wahrscheinlichkeit, dass $Z = 0$ erreicht wird, zu

$$\begin{aligned}
&\sum_{j=0}^{n} 2^{-n}\binom{n}{j} p_j \\
&= 2^{-n}\sum_{j=0}^{n}\binom{n}{j}\left(\tfrac{1}{k-1}\right)^j \\
&= 2^{-n}\left(1 + \tfrac{1}{k-1}\right)^n \\
&= \left(\frac{k}{2(k-1)}\right)^n
\end{aligned}$$

Damit ist auch die Erfolgswahrscheinlichkeit eines Schleifendurchlaufs mindestens $\left(\frac{k}{2(k-1)}\right)^n$, und wir erhalten als obere Schranke an die nötige Zahl von Wiederholungen, und damit die Laufzeit des Algorithmus $O\left(\left(\frac{2(k-1)}{k}\right)^n\right)$. Für $k = 3$ ist dies $(4/3)^n$, und für größere Werte von k erhalten wir die folgenden Basen:

k	3	4	5	6	7	8
$\frac{2(k-1)}{k}$	1.33334	1.5	1.6	1.66667	1.71429	1.75

Verbesserung durch unabhängige Klauseln

Für 3-SAT läßt sich die Komplexität des Irrfahrt-Algorithmus noch leicht verbessern.

Definition. *Eine Menge $G \subseteq F$ von Klauseln in F heißt* unabhängig, *wenn je zwei Klauseln C, D in F variablendisjunkt sind, also $V(C) \cap V(D) = \emptyset$ ist.*

Ist G unabhängig, so ist $|V(G)| = 3|G|$.

Der Algorithmus berechnet zunächst in Greedy-Manier eine maximale unabhängige Teilmenge G in F wie folgt:

$G := \emptyset$

while es gibt C in F mit $V(C) \cap V(G) = \emptyset$
    $G := G \cup \{C\}$

Nun wird je nach der Größe von G unterschiedlich verfahren:

Da G maximal unabhängig ist, enthält jede der übrigen Klauseln in $F \setminus G$ mindestens eine der Variablen in $V(G)$ enthalten. Wäre nämlich C eine Klausel, für die dies nicht der Fall ist, so könnte G um diese Klausel erweitert werden.

Ist nun α eine Bewertung der Variablen in $V(G)$, so ist $F\alpha$ eine 2-KNF-Formel, für die das Erfüllbarkeitsproblem in polynomieller Zeit gelöst werden kann. Es gibt insgesamt $7^{|G|}$ Bewertungen, die die Klauseln in G erfüllen.

Ist nun G klein, nämlich $|G| \leq \delta n$ für einen noch zu bestimmenden Parameter $0 < \delta \leq \frac{1}{3}$, so wird für jede der $7^{|G|}$ Bewertungen $\alpha \models G$ das Erfüllbarkeitsproblem für die 2-SAT Instanz $F\alpha$ gelöst. Die Zeitkomplexität ist damit im wesentlichen $7^{\delta n}$.

Ist dagegen $|G| > \delta n$, so wird der Irrfahrt-Algorithmus durchgeführt, wobei allerdings die Anfangsbewertungen nicht gleichverteilt, sondern gemäß der Verteilung $A_G(p_1, p_2, p_3)$ gewählt werden.

Für Wahrscheinlichkeiten $p_1, p_2, p_3 > 0$ mit $3p_1 + 3p_2 + p_3 = 1$ ist die Verteilung $V_G(p_1, p_2, p_3)$ von Bewertungen definiert wie folgt:

Zunächst werden die Variablen in $V(G)$ bewertert, indem für jede Klausel $C = (a_1 \vee a_2 \vee a_3)$ in G Werte $\epsilon_1, \epsilon_2, \epsilon_3$ mit Wahrscheinlichkeit nach der Tabelle

ϵ_1	ϵ_2	ϵ_3		ϵ_1	ϵ_2	ϵ_3	
0	0	0	0	0	1	1	p_2
0	0	1	p_1	1	0	1	p_2
0	1	0	p_1	1	1	0	p_2
1	0	0	p_1	1	1	1	p_3

gewählt werden, und dann die Bewertung

$$[a_1 \leftarrow \epsilon_1, a_1 \leftarrow \epsilon_2, a_1 \leftarrow \epsilon_3]$$

gesetzt wird. Anschliessend wird für jede Variable x in $V(F) \setminus V(G)$ ein Wert $\epsilon \in \{0, 1\}$ zufällig gewählt und $[x \leftarrow \epsilon]$ gesetzt.

Zur Abschätzung der Erfolgswahrscheinlichkeit erinnern wir und, dass die Wahrscheinlichkeit, von einer Anfangsbewertung α aus die erfüllende Bewertung α^* zu erreichen, mindestens

$$(\frac{1}{k-1})^{d(\alpha,\alpha^*)}$$

ist. Für $k = 3$ ist dies $\left(\frac{1}{2}\right)^{d(\alpha,\alpha^*)}$.

Es bezeichne $p(\alpha)$ die Wahrscheinlichkeit, dass α als Anfangsbewertung gewählt wird. Dann ist die Erfolgswahrscheinlichkeit

$$p(n) \geq \sum_{\alpha} p(\alpha)\left(\frac{1}{2}\right)^{d(\alpha,\alpha^*)} ,$$

und der rechte Term ist gerade der Erwartungswert $E[\left(\frac{1}{2}\right)^{d(\alpha,\alpha^*)}]$ der Zufallsvariable $\left(\frac{1}{2}\right)^{d(\alpha,\alpha^*)}$.

Wir betrachten nun folgende Zufallsvariablen: X_i sei die Indikatorvariable des Ereignisses, dass $\alpha(x_i) \neq \alpha^*(x_i)$ ist, also

$$X_i = \begin{cases} 1 & \text{falls } \alpha(x_i) \neq \alpha^*(x_i) \\ 0 & \text{sonst.} \end{cases}$$

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

$$E\left[\left(\frac{1}{2}\right)^{d(\alpha,\alpha^*)}\right] = E\left[\left(\frac{1}{2}\right)^{X_1+\ldots+X_n}\right] .$$

Bei gleichverteilter Wahl von α sind nun die Variablen X_i unabhängig, daher können wir ausrechnen

$$E\left[\left(\frac{1}{2}\right)^{X_1+\ldots+X_n}\right] = \prod_{i=1}^{n} E\left[\left(\frac{1}{2}\right)^{X_i}\right] ,$$

und der Erwartungswert $E[(\frac{1}{2})^{X_i}]$ ist $(\frac{1}{2})^0 \cdot \frac{1}{2} + (\frac{1}{2})^1 \cdot \frac{1}{2} = \frac{3}{4}$, woraus wir wie oben $p(n) \geq (\frac{3}{4})^n$ erhalten.

Im Fall der Wahl von α gemäß $V_G(p_1, p_2, p_3)$ sind die Zufallsvariable X_i nicht mehr unabhängig, aber Abhängigkeit besteht nur zwischen den Variablen innerhalb einer Klausel in G.

Es sei also $G = \{C_1, \ldots, C_{\delta n}\}$ mit $V(C_{i+1}) = \{x_{3i+1}, x_{3i+1}, x_{3i+1}\}$ für $i = 0, \ldots, \delta n - 1$. Wir definieren die Zufallsvariable $S_i = X_{3i+1} + X_{3i+1} + X_{3i+1}$, dann sind diese Variablen S_i wiederum unabhängig, und wir erhalten

$$E\left[\left(\frac{1}{2}\right)^{X_1+\ldots+X_n}\right] = \prod_{i=1}^{\delta n} E\left[\left(\frac{1}{2}\right)^{S_i}\right] \cdot \prod_{j=3\delta n+1}^{n} E\left[\left(\frac{1}{2}\right)^{X_j}\right] .$$

Der rechte Faktor berechnet sich wie oben zu $(\frac{3}{4})^{1-3dn}$.

Zur Berechnung des linken Faktors brauchen wir die Erwartungswerte der Variablen $(\frac{1}{2})^{S_i}$. Diese hängen aber davon ab, wieviele der Literale in C_i von α^* erfüllt werden.

Falls α^* genau ein Literal a in C_i mit 1 bewertet, so hat S_j den Wert 0, wenn α auf C_i mit α^* übereinstimmt, also nur a mit 1 und die anderen Literale in C_i mit 0 bewertet. Dies passiert mit Wahrscheinlichkeit p_1.

Weiter hat S_i den Wert 1, wenn α das Literal a und ein weiteres mit 1 bewertet, dafür gibt es 2 Möglichkeiten, die jeweils mit Wahrscheinlichkeit p_2 eintreten.

S_i hat den Wert 2, wenn α alle drei Literale in C_i mit 1 bewertet, was mit Wahrscheinlichkeit p_3 eintritt, oder wenn α das Literal a mit 0 und ein anderes mit 1 bewertet, was mit Wahrscheinlichkeit $2p_1$ eintritt.

Schließlich hat S_i den Wert 3, wenn a von α mit 0 und die zwei anderen Literale mit 1 bewertet werden, was mit Wahrcheinlichkeit p_2 eintritt.

Insgesamt errechnet sich der Erwartungswert in diesem Fall also als

$$E\left[\left(\tfrac{1}{2}\right)^{S_i}\right] = \left(\tfrac{1}{2}\right)^0 p_1 + \left(\tfrac{1}{2}\right)^1 2p_2 + \left(\tfrac{1}{2}\right)^2 (2p_1 + p_3) + \left(\tfrac{1}{2}\right)^3 p_2 = \tfrac{3}{2}p_1 + \tfrac{9}{8}p_2 + \tfrac{1}{4}p_3 .$$

Im Fall, dass α^* genau zwei Literale in C_i mit 1 bewertet, errechnet man auf ähnliche Weise

$$E\left[\left(\tfrac{1}{2}\right)^{S_i}\right] = \left(\tfrac{1}{2}\right)^0 p_2 + \left(\tfrac{1}{2}\right)^1 (2p_1 + p_3) + \left(\tfrac{1}{2}\right)^2 2p_2 + \left(\tfrac{1}{2}\right)^3 p_1 = \tfrac{9}{8}p_1 + \tfrac{3}{2}p_2 + \tfrac{1}{2}p_3 .$$

Im letzten Fall, wo α^* alle drei Literale in C_i mit 1 bewertet, errechnet man schließlich

$$E\left[\left(\tfrac{1}{2}\right)^{S_i}\right] = \left(\tfrac{1}{2}\right)^0 p_3 + \left(\tfrac{1}{2}\right)^1 3p_2 + \left(\tfrac{1}{2}\right)^2 3p_1 = \tfrac{3}{4}p_1 + \tfrac{3}{2}p_2 + p_3 .$$

Für $j = 1, 2, 3$ sei also m_j die Anzahl der Klauseln in G, in denen α^* genau j Literale erfüllt. Dann ist also der linke Faktor des obigen Produktes gleich

$$\left(\tfrac{3}{2}p_1 + \tfrac{9}{8}p_2 + \tfrac{1}{4}p_3\right)^{m_1} \cdot \left(\tfrac{9}{8}p_1 + \tfrac{3}{2}p_2 + \tfrac{1}{2}p_3\right)^{m_2} \cdot \left(\tfrac{3}{4}p_1 + \tfrac{3}{2}p_2 + p_3\right)^{m_3} .$$

und durch Lösen des linearen Gleichungssystems aus der Gleichheit der drei Basen zusammen mit der Einschränkung $3p_1 + 3p_2 + p_3 = 1$ erhält man die Werte

$$p_1 = \tfrac{4}{21}, \qquad p_2 = \tfrac{2}{21}, \qquad p_3 = \tfrac{1}{7} \quad ,$$

für die sich die Basis als $frac37$ ergibt. Damit ist die Erfolgswahrscheinlichkeit mindestens

$$p(n) \geq \left(\tfrac{3}{7}\right)^{m_1+m_2+m_3} \cdot \left(\tfrac{3}{4}\right)^{1-3\delta n} = \left(\tfrac{3}{7}\right)^{\delta n} \cdot \left(\tfrac{3}{4}\right)^{1-3\delta n} ,$$

und damit die nötige Anzahl von Wiederholungen und somit auch die Laufzeit des Algorithmus

$$\left(\left(\tfrac{4}{3}\right)^{1-3\delta} \cdot \left(\tfrac{7}{3}\right)^{\delta}\right)^n .$$

Man beobachtet nun, dass dieser Wert in δ monoton fallend ist, während die Laufzeit $7^{\delta n}$ im anderen Zweig monoton in δ ansteigt. Der minimale Wert der Laufzeit ergibt sich also an der Schnitstelle, wo

$$7^{\delta} = \left(\frac{4}{3}\right)^{1-3\delta} \cdot \left(\frac{7}{3}\right)^{\delta}$$

ist. Dieser Wert von δ ist ungefähr 0.14666, und der Wert 7^{δ} an dieser Stelle ist ungefähr 1.33026, also erhalten wir für den Algorithmus insgesamt eine Laufzeit von $O(1.33026^n)$.

Es gibt noch eine verbesserte Version des Algorithmus, die eine Laufzeit von $O(1.32793^n)$ erreicht, diese wird aber aus Platzgründen hier nicht behandelt.

4.3 Algorithmus von Paturi, Pudlák und Zane

In diesem Abschnitt behandeln wir einen probabilistischen Algorithmus, der der Vorgehensweise bei DLL ähnlicher ist als die vorher betrachteten.

Wie bei einem DLL-Verfahren werden die Variablen der Reihe nach betrachtet und ihnen Werte zugewiesen. Dabei werden Variablen in Einerklauseln natürlich so bewertet, dass diese erfüllt sind, die übrigen Variablen werden zufällig bewertet. Die Reihenfolge, in der die Variablen betrachtet werden, ist durch eine Permutation $\pi \in S_n$ der Indizes $1, \dots, n$ vorgegeben, die ebenfalls zufällig gewählt wird.

Ist am Ende die Formel erfüllt, wird die so gefundene Bewertung audsgegeben, andernfalls wird sie verworfen und mit einer neuen Permutation von vorn begonnen.

Die Prozedur suche(π) betrachtet die Variablen in der durch π vorgegebenen Reihenfolge und weist ihnen einen zufälligen Wert zu, außer wenn ein Wert bereits durch Einerklauseln erzwungen ist.

suche(π)
  for $i := 1$ to n do
    if Einerklausel $x^{\epsilon}_{\pi(i)}$ in F
      then $\epsilon_i := \epsilon$
      else wähle ϵ_i zufällig aus $\{0, 1\}$
    $F := F[x_{\pi(i)} \leftarrow \epsilon_i]$
  if $\alpha \models F$
    then return α
    else return NIL

Diese Prozedur wird in einer Schleife eine bestimmte Anzahl w mal wiederholt.

  repeat w times

```
    wähle π ∈ S_n zufällig
    β := suche(π)
    if β ≠ NIL
        then return β
return NIL
```

Wie bei den probablilistischen Algorithmen im vorhergehenden Kapitel berechnen wir die Wahrscheinlichkeit $p(n)$ dafür, dass ein Schleifendurchlauf erfolgreich ist, also eine erfüllende Bewertung findet, im Fall dass F erfüllbar ist. Daraus ergibt sich, dass $w = O(1/p(n))$ Wiederholungen genügen, um die Fehlerwahrscheinlichkeit beliebig klein zu halten, und damit eine Laufzeit von $O(1/p(n))$.

Definition. *Für eine Menge A von Bewertungen und $\alpha \in A$ ist*

$$N_A(\alpha) := |H(\alpha, 1) \cap A| - 1$$

die Anzahl der Nachbarn von α in A.

Lemma 26. *Ist $A \neq \emptyset$, so gilt*

$$\sum_{\alpha \in A} 2^{-N_A(\alpha)} \geq 1 \ .$$

Beweis. Die Aussage wird durch Induktion nach n bewiesen. Der Induktionsanfang für $n = 0$ ist trivial.

Sei also $n > 0$, und gelte die Aussage für Mengen von Bewertungen von $n-1$ Variablen. Für eine Bewertung α der Variablen $x_1, \ldots, x_{n-1}$ und $i = 0, 1$ sei $\alpha i := \alpha \cup [x_n \leftarrow i]$.

Sei $A \neq \emptyset$ eine Menge von Bewertungen aller n Variablen. Für $i = 0, 1$ sei $A_i := \{\, \alpha \,;\, \alpha i \in A \,\}$. Für mindestens ein i muss $A_i \neq \emptyset$ sein.

Ist $A_0 = \emptyset$, so muss $A_1 \neq \emptyset$ sein. Also ist

$$\sum_{\alpha \in A} 2^{-N_A(\alpha)} \ = \ \sum_{\alpha \in A_1} 2^{-N_A(\alpha 1)} \ = \ \sum_{\alpha \in A_1} 2^{-N_{A_1}(\alpha)} \ \geq \ 1 \ ,$$

wobei die letzte Ungleichung nach der Induktionshypothese gilt. Die letzte Gleichheit gilt, da für $\alpha \in A_1$ der einzige Nachbar von $\alpha 1$, der nicht von der Form $\beta 1$ ist, gerade $\alpha 0$ ist. $A_0 = \emptyset$ ist aber $\alpha 0 \notin A$.

Der Fall $A_1 = \emptyset$ ist symmetrisch. Seien nun A_0 und A_1 beide nichtleer. Dann ist

$$\begin{aligned}
\sum_{\alpha\in A} 2^{-N_A(\alpha)} &= \sum_{\alpha\in A_0} 2^{-N_A(\alpha 0)} + \sum_{\alpha\in A_1} 2^{-N_A(\alpha 1)} \\
&\geq \sum_{\alpha\in A_0} 2^{-N_{A_0}(\alpha)-1} + \sum_{\alpha\in A_1} 2^{-N_{A_1}(\alpha)-1} \\
&\geq \frac{1}{2} + \frac{1}{2} = 1 \,.
\end{aligned}$$

Dabei gilt die vorletzte Ungleichung wiederum, da es für $i = 0, 1$ höchstens einen Nachbarn von αi gibt, der nicht von der Form β_i ist, nämlich $\alpha 0$ $\square$

Zur Berechnung der Erfolgswahrscheinlichkeit $p(n)$ sei $A := \{\, \alpha\, ;\ \alpha \models F \}$ die Menge der F erfüllenden Bewertungen, und sei $\alpha \in A$. Definiere

$$j(\alpha) := n - N_A(\alpha)$$

und für $i = 1, \dots, n$ sei $\alpha_i := \alpha$ with $[x_i \leftarrow 1 - \alpha(x)]$. Das heißt, es gibt $j(\alpha)$ Variablen x_i mit $\alpha_i \not\models F$, diese heißen die *kritischen* Variablen.

Für jede kritische Variable x_i gibt es eine *kritische* Klausel C_i in F, derart dass das einzige Literal a in C_i mit $\alpha(a) = 1$ entweder x_i oder $\neg x_i$ ist.

Für eine Permutation $\pi \in S_n$ sei

$$R(\pi, \alpha) = \Big\{\, x_i \text{ kritisch} \,;\ \pi^{-1}(i) \geq \pi^{-1}(j) \text{ für alle } x_j \in V(C_i) \Big\}$$

die Menge derjenigen kritischen Variablen x_i, die unter den Variablen in ihrer kritischen Klausel C_i in der Permutation π die letzte sind, und $r(\pi, \alpha) = |R(\pi, \alpha)|$.

Da die Permutation π gleichverteilt gewählt wird, ist jede der k Variablen in C_i mit gleicher Wahrscheinlichkeit $1/k$ die letzte in π. Jede der $j(\alpha)$ kritischen Variablen ist also mit Wahrscheinlichkeit $1/k$ in $R(\pi, \alpha)$, daher ist der erwartete Wert $E[r(\pi, \alpha)] = j(\alpha)/k$.

Mit jeder Variablen $x_i \in R(\pi, \alpha)$ erhöht sich die Wahrscheinlichkeit dafür, dass α bei suche(π) gefunden wird: wenn die anderen Variablen y in C_i zufällig den richtigen Wert $\alpha(y)$ erhalten, dann erhält auch x_i den Wert $\alpha(x_i)$, da zu dem Zeitpunkt, wo der Algorithmus x_i betrachtet, die Klausel C_i Einerklausel ist.

Daher ist die Wahrscheinlichkeit $p_{\alpha,\pi}$ dafür dass α gefunden wird, falls die Permutation π gewählt wurde, mindestens $p_{\alpha,\pi} \geq 2^{-n+r(\pi,\alpha)}$.

Also ist die Wahrscheinlichkeit p_α dafür, dass α gefunden wird, insgesamt

$$\begin{aligned} p_\alpha &= \sum_{\pi \in S_n} \frac{1}{n!} p_{\alpha,\pi} = \sum_{\pi \in S_n} \frac{1}{n!} 2^{-n+r(\pi,\alpha)} \\ &= 2^{-n} \sum_{\pi \in S_n} \frac{1}{n!} 2^{r(\pi,\alpha)} = 2^{-n} \cdot E[2^{r(\pi,\alpha)}] \,. \end{aligned}$$

Der Erwartungswert $E[2^{r(\pi,\alpha)}]$ ist aber nach der Jensen'schen Ungleichung mindesten $2^{E[r(\pi,\alpha)]}$, also ist

$$p_\alpha \geq 2^{-n+E[r(\pi,\alpha)]} \geq 2^{-n+j(\alpha)/k} \,.$$

Die Jensen'sche Ungleichung besagt, dass für eine Zufallsvariable X, die jeden der Werte $a_1, \ldots, a_n$ mit Wahrscheinlichkeit $1/n$ annimmt, gilt $E[2^X] \cdot 2^{E[x]}$. Dies rechnet man nach, indem man den Logarithmus zur Basis 2 betrachtet, also zeigt $\log E[2^X] \geq E[X]$. Es ist nämlich

$$\begin{aligned} \log E[2^X] &= \log\Big(\frac{1}{n}\sum_{i=1}^{n} 2^{a_i}\Big) \geq \log\Big(\big(\prod_{i=1}^{n} 2^{a_i}\big)^{1/n}\Big) \\ &= \frac{1}{n}\log\prod_{i=1}^{n} 2^{a_i} = \frac{1}{n}\sum_{i=1}^{n}\log 2_i^a = \frac{1}{n}\sum_{i=1}^{n} a_i = E[X] \end{aligned}$$

wobei die erste Ungleichung nach dem Satz vom arithmetischen und geometrischen Mittel gilt.

Die gesamte Erfolgswahrscheinlichkeit lässt sich nun beschränken durch

$$\begin{aligned} p(n) &= \sum_{\alpha \in A} p_\alpha \geq \sum_{\alpha \in A} 2^{-n+j(\alpha)/k} \\ &= \sum_{\alpha \in A} 2^{-n+n/k+(j(\alpha)-n)/k} = 2^{-n+n/k} \sum_{\alpha \in A} 2^{-N(\alpha)/k} \\ &\geq 2^{-n+n/k} \sum_{\alpha \in A} 2^{-N(\alpha)} \geq 2^{-n+n/k} \,. \end{aligned}$$

wobei die letzte Ungleichung nach Lemma 26 gilt.

Also ist $p(n) \geq 2^{-n+n/k}$, und somit erhalten wir als Schranke an die Laufzeit $O\big((2^{1-1/k})^n\big)$. Die Basis errechnet sich für kleine Werte von k wie folgt:

k	3	4	5	6	7	8
$2^{1-1/k}$	1.58741	1.68180	1.74111	1.78180	1.81145	1.83401

Dieser Algorithmus war zum Zeitpunkt seiner Veröffentlichung 1997 der erste, der eine bessere Komplexität für k-SAT mit $k \geq 4$ erreicht als der von Monien und Speckenmeyer. Ausserdem war dies der erste theoretisch analysierte probabilistische Algorithmus für k-SAT.

Eine verbesserte Form dieses Algorithmus wurde von Paturi, Pudlák, Saks und Zane etwas später angegeben. Der Algorithmus arbeitet genau wie der hier vorgestellte, aber es wird ein Präprozessor vorgeschaltet, der die Formel um alle Klauseln ergänzt, die aus den Eingabeklauseln durch Resolution erzeugt werden können, und deren Größe durch einen (von k und n abhängigen) Parameter s beschränkt ist. Die Analyse ist zu komplex, um hier präsentiert zu werden.

Dieser verbesserte Algorithmus ist derzeit der beste probabilistische Algorithmus für k-SAT mit $k \geq 5$, er erreicht die Komplexität $O(b_k^n)$ für die folgenden Werte der Basis b_k:

k	3	4	5	6	7	8
b_k	1.36406	1.49579	1.56943	1.63788	1.68753	1.72521

Der Algorithmus von Paturi, Pudlák, Saks und Zane läßt sich wiederum mit dem Irrfahrt-Algorithmus geschickt kombinieren, so dass sich für 3-SAT und 4-SAT noch eine Verbesserung ergibt. Dieser kombinierte Algorithmus ist der derzeit beste Algorithmus fuer diese Fälle, er erreicht für 3-SAT die Komplexität $O(1.324^n)$ und für 4-SAT $O(1.474^n)$.