Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Diplomarbeit

# Schöning's k-SAT-Algorithmus

Matthias Kaufmann

Korrigierte Fassung vom 17. Oktober 2002

Institut für Theoretische Informatik
ETH Zürich

Diplomprofessor: Emo Welzl
Betreuer: Uli Wagner

## Korrektur

Ich danke den Herren A. Kulikov, E. Hirsch sowie Herrn U. Schöning für die Hinweise zur Berichtigung des Fehlers im Kapitel 4.2. Die dort gemachte Gleichverteilungsannahme wurde in dieser überarbeiteten Version der Diplomarbeit weggelassen.

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

## 1.1 Erfüllbarkeitsprobleme

In der formalen Logik wird untersucht, wie Aussagen miteinander verknüpft und wie daraus Schlüsse gezogen werden können. Dazu werden logische Formeln verwendet, die aus Verknüpfungen von Wahrheitswerten mittels "oder", "und" oder "nicht" entstehen. Eine solche Formel lautet zum Beispiel "((A und B) oder C) und (nicht(B) oder C)" und aus ihr kann "C" geschlossen werden. Häufig ist für eine solche Formel aber bloss von Interesse, ob es Wahrheitswerte für ihre Variablen gibt, welche die Formel als ganzes wahr machen. Dieses allgemeine Problem wird als "Erfüllbarkeit aussagenlogischer Formeln" oder auch einfach nur als "SAT-Problem" bezeichnet.
In der theoretischen Informatik spielt das SAT-Problem als Prototyp der NP-vollständigen Probleme eine grosse Rolle. Bis heute gibt es für diese Probleme nur Lösungsalgorithmen mit einem exponentiellen Worst-Case-Aufwand, und es wird vermutet, dass keine polynomiellen Verfahren existieren. Jede ihrer Lösungen kann aber durch Raten gefunden und dann mit einer Turingmaschine in Polynomialzeit verifiziert werden. Andererseits hat Cook [1] bewiesen, dass der Ablauf einer Turingmaschine auch durch eine SAT-Formel simuliert werden kann, die bloss polynomiell gross in der Länge der Arbeitszeit ist, die die Turingmaschine für die Verifikation einer Lösung braucht. Somit können über das Lösen des allgemeinen SAT-Problems auch alle anderen NP-vollständigen Probleme gelöst werden: Mit der SAT-Formel findet man heraus, ob die Turingmaschine in der Lage ist, eine Lösung zu erraten, welche alle Bedingungen erfüllt - womit die Formel

genau die Frage beantwortet, ob es überhaupt eine Lösung für das Problem gibt. Die Konsequenz davon ist, dass das Lösen des allgemeinen SAT-Problems in Polynomialzeit auch polynomielle Verfahren für alle anderen NP-vollständigen Probleme zur Folge hätte.

In der Praxis taucht das SAT-Problem in der künstlichen Intelligenz, bei vielen Problemen auf dem Gebiet des Scheduling und in Planungsalgorithmen auf. In der Elektrotechnik können mittels Erfüllbarkeitsberechnungen auf SAT-Formeln elektronische Schaltungen vereinfacht und überprüft werden.

Aufgrund seiner theoretischen wie auch praktischen Bedeutung möchte man das SAT-Problem trotz seiner Schwierigkeit mit einem möglichst geringen exponentiellen Aufwand lösen. Es gibt einige Verfahren, welche dies in weniger als $O(2^n)$ Schritten erledigen können (siehe z.B. [3], [4]). Unter ihnen war bis vor kurzem [8] der von Uwe Schöning entwickelte Algorithmus [2] das Verfahren mit der beweisbar besten Laufzeit, das gleichzeitig durch die simple Lösungsidee überzeugt.

## 1.2 Definition k-SAT-Problem

Für theoretische wie praktische Zwecke genügt es, sich auf SAT-Probleme in konjunktiver Normalform (KNF) zu beschränken - die k-SAT-Formeln. Eine k-SAT-Formel besteht aus $m$ Klauseln, welche untereinander durch Konjunktion (“und”, entspricht “$\wedge$”) verbunden sind. Eine Klausel besteht aus genau $k$ (nicht notwendigerweise verschiedenen) Literalen, welche ihrerseits durch Disjunktion (“oder”, entspricht “$\vee$”) verbunden sind. Ein Literal schlussendlich ist eine von $n$ Variablen, die negiert sein kann (“nicht”, entspricht “$^-$”).
Gefragt ist, ob jeder der Variablen einer der beiden Wahrheitswerte $wahr$, $falsch$ zugewiesen werden kann, so dass die Formel als ganzes $wahr$ ist. Die folgende Formel ist ein Beispiel für $k = 3$, $n = 4$, $m = 5$:

$$(\bar{a} \vee \bar{b} \vee c) \wedge (a \vee c \vee d) \wedge (b \vee \bar{c} \vee \bar{d}) \wedge (\bar{b} \vee \bar{c} \vee \bar{d}) \wedge (a \vee b \vee \bar{d})$$

Sie kann erfüllt werden, wenn die Variablen mit $a = wahr$, $b = falsch$, $c = falsch$ und $d = wahr$ belegt werden. Gibt es nur eine einzige solche

erfüllende Belegung, dann wird die Formel einfach erfüllbar genannt.
Die folgende 2-SAT-Formel ist nicht erfüllbar:

$$(a \vee b) \wedge (\bar{a} \vee b) \wedge (a \vee \bar{b}) \wedge (\bar{a} \vee \bar{b})$$

Aus den ersten beiden Klauseln folgt, das $b$ erfüllt sein muss, und aus den beiden anderen, dass auch $\bar{b}$ erfüllt sein muss, was nicht möglich ist.

## 1.3 Algorithmen

Es gibt verschiedene Algorithmen, die das k-SAT-Problem lösen. Ein grosser Teil von ihnen gehört zu den vollständigen Verfahren, welche in jedem Fall die richtige Antwort (erfüllbar/nicht erfüllbar) liefern. Zum Beispiel fallen die Backtracking-Verfahren mit ihren verschiedenen Heuristiken (z.B. Variablenreihenfolge) in diese Kategorie.
Zur Kategorie der unvollständigen Verfahren gehören die meisten Local-Search-SAT-Algorithmen. Diese arbeiten den Lösungsraum nicht systematisch ab, sondern versuchen, mit hoher Wahrscheinlichkeit eine erfüllende Belegung zu finden. Wenn sie diese nicht finden, dann gibt es mit einer gewissen (wenn auch kleinen) Wahrscheinlichkeit trotzdem eine erfüllende Belegung und der Algorithmus liefert das falsche Resultat. In diese Kategorie gehören die meisten probabilistischen Algorithmen, so zum Beispiel GSAT [5].

## 1.4 Inhalt

In den folgenden Kapiteln wird nun der Schöningalgorithmus und seine Analyse vorgestellt und versucht, die ihm zu Grunde liegende Lösungsidee auf andere NP-vollständige Probleme zu übertragen. Mittels Tests wird aufgezeigt, wie die Erfüllbarkeit des Random-k-SAT-Problems mit den Problemparametern ändert. Des weiteren werden Methoden zur Generierung einfach erfüllbarer k-SAT-Probleme vorgestellt und eine neue Methode vorgeschlagen, die für den Schöningalgorithmus schwierig zu lösende Formeln generiert. Schlussendlich wird die leicht verbesserte Version des Schöningalgorithmus und seine Analyse behandelt.

# Kapitel 2

# Schöning's Algorithmus

## 2.1 Der Algorithmus

---

*try*($F$: $k$-SAT-Formel mit $n$ Variablen):
Wähle eine zufällige Belegung $a \in \{0,1\}^n$.
Wiederhole $3n$ mal:
    Falls $F$ durch $a$ erfüllt wird,
        beende und liefere 1 zurück,
    sonst
        wähle eine unter $a$ nicht erfüllte Klausel $C$ in $F$,
        wähle zufällig eine Variable $v$ in $C$,
        kehre in $a$ die Belegung für $v$ um.
Liefere 0 zurück.

---

Wie aus der *try*-Prozedur ersichtlich ist, handelt es sich bei Schöning's Algorithmus um einen probabilistischen Local-Search-Algorithmus, welcher durch das zufällige "Verbessern" nicht erfüllter Klauseln versucht, zu einer erfüllenden Belegung zu gelangen.
*try* allein ist aber noch nicht der ganze Algorithmus: Die Wahrscheinlichkeit, dass ein einziger *try*-Aufruf zufällig gerade die erfüllende Belegung findet, ist vor allem bei grossen $n$ sehr klein. Deshalb muss *try* in exponentiell vielen voneinander unabhängigen Versuchen auf eine SAT-Formel angewendet werden. Erst wenn in keinem dieser Versuche eine erfüllende Belegung gefunden wurde, kann mit einer gewissen Fehlerwahrscheinlichkeit garantiert

werden, dass die Formel nicht erfüllbar ist.

Wie wahrscheinlich das Finden einer erfüllenden Belegung ist und wie die Anzahl benötigter *try*-Aufrufe davon abhängt, wird in den beiden folgenden Analysen aufgezeigt. Der Vorteil von Schöning's Analyse ist, dass sie sich sehr einfach auf eine Verallgemeinerung des Problems (siehe 2.4) übertragen lässt, während die Analyse von Welzl viel genauer ist und deshalb zum Beispiel für die Implementierung des Algorithmus benötigt wird.

## 2.2 Schöning's Analyse

Gesucht ist die Wahrscheinlichkeit $p$, dass $try$ eine erfüllende Belegung findet, falls eine solche existiert:

$$p = \mathbf{Pr}[\,try(F) = 1 \mid F \text{ erfüllbar}]$$

Schöning betrachtet die Anzahl boolscher Variablen, in der sich die aktuelle Belegung $a$ von der (unbekannten) erfüllenden Belegung $a^*$ unterscheidet. Diese Anzahl $j$, welche der Hammingdistanz $d(a, a^*)$ entspricht, ist gleichzeitig ein Zustand, wie weit $try$ noch von der erfüllenden Belegung entfernt ist. Sei $X_t$ eine Zufallsvariable, die diesen Zustand bei der $t$-ten Iteration hält. Falls die SAT-Formel erfüllbar ist, dann gibt es in einer nicht erfüllten Klausel mindestens eine Variable, die unter $a$ falsch belegt ist. Wenn also Schöning's Algorithmus in einer solchen Klausel mit $k$ Literalen den Wert einer zufälligen Variable umkehrt, dann wird es mit Wahrscheinlichkeit mindestens $\frac{1}{k}$ eine solche falsche Variable sein, die damit verbessert wird. In diesem Fall wird also $d(a, a^*)$ um eins abnehmen, weshalb

$$\mathbf{Pr}[X_{t+1} = j - 1 | X_t = j] \geq \frac{1}{k}.$$

Umgekehrt wird mit einer Wahrscheinlichkeit von höchstens $\frac{k-1}{k}$ eine schon richtig belegte Variable unnötig korrigiert, was einer Vergrösserung der Hammingdistanz beziehungsweise einem Übergang in den Zustand $j + 1$ entspricht. Bei den Verbesserungsschritten handelt es sich also um einen Random Walk, bei welchem mit Wahrscheinlichkeit $\frac{1}{k}$ in die richtige und mit Wahrscheinlichkeit $\frac{k-1}{k}$ in die falsche Richtung gegangen wird.

Sei der Anfangszustand $X_0 = j$, das heisst $j$ Variablen sind falsch belegt. Unter der Annahme, dass genau $i$ Schritte in die falsche (die Hammingdistanz vergrössernde) Richtung gemacht werden, müssen aber auch genau $i+j$ Schritte in die richtige Richtung gemacht werden, wenn die erfüllende Belegung $a^*$ erreicht werden soll (total $j+2i$ Schritte). Die Anzahl Möglichkeiten, auf diese Weise zu $a^*$ zu gelangen, sei $m(i,j)$. Offensichtlich gilt

$$m(i,j) = \begin{cases} 1 & \text{für } i = 0 \\ m(i-1,2) & \text{für } i > 0 \ \text{ und } j = 1 \\ m(i-1,j+1) + m(i,j-1) & \text{für } i > 0 \ \text{ und } j > 1 \end{cases}$$

Mittels vollständiger Induktion über $2i+j$ kann gezeigt werden, dass

$$m(i,j) = \binom{2i+j}{i} \cdot \frac{j}{2i+j}.$$

Falls die Anzahl falsch gesetzter Variablen $j$ beträgt, dann gilt für die Wahrscheinlichkeit $q_j$, dass man mit höchstens $i$ Schritten (für $i \leq j$) in die falsche Richtung trotzdem noch zur erfüllenden Belegung gelangt,

$$\begin{aligned} q_j &\geq \sum_{i=0}^{j} m(i,j) \cdot \left(\frac{k-1}{k}\right)^i \cdot \left(\frac{1}{k}\right)^{i+j} \\ &= \sum_{i=0}^{j} \binom{j+2i}{i} \cdot \frac{j}{j+2i} \cdot \left(\frac{k-1}{k}\right)^i \cdot \left(\frac{1}{k}\right)^{i+j} \\ &\geq \frac{1}{3} \cdot \sum_{i=0}^{j} \binom{j+2i}{i} \cdot \left(\frac{k-1}{k}\right)^i \cdot \left(\frac{1}{k}\right)^{i+j}. \end{aligned}$$

Gemäss [11] gilt

$$\sum_{i=0}^{\alpha n} \binom{n}{i} \asymp 2^{h(\alpha)\cdot n} = \left(\left(\frac{1}{\alpha}\right)^{\alpha} \cdot \left(\frac{1}{1-\alpha}\right)^{1-\alpha}\right)^n,$$

das heisst, alle diese Funktionen sind bis auf einen polynomiellen Faktor gleich (“$\asymp$”), wenn $h()$ die binäre Entropiefunktion und $\alpha \leq \frac{1}{2}$ fest ist. Andererseits ist

$$\sum_{i=0}^{\alpha n} \binom{n}{i} \asymp \binom{n}{\alpha n}$$

(es gibt nur polynomiell viele Summanden auf der linken Seite), woraus

$$\binom{n}{\alpha n} \asymp \left(\left(\frac{1}{\alpha}\right)^{\alpha} \cdot \left(\frac{1}{1-\alpha}\right)^{1-\alpha}\right)^n$$

folgt. Da beide Funktionen symmetrisch bezüglich $\alpha = \frac{1}{2}$ sind und dort auch ihr jeweiliges Maximum haben, gilt die letzte Beziehung sogar für $\alpha \in [0,1]$. Sie lässt sich allgemeiner schreiben als

$$\binom{bj}{cj} \asymp \left(\left(\frac{b}{c}\right)^c \cdot \left(\frac{b}{b-c}\right)^{b-c}\right)^j \quad \text{für } c \leq b.$$

Wird $i$ durch $\alpha j$ ersetzt mit $\alpha \in [0,1]$, dann ist

$$\binom{j+2i}{i} = \binom{(1+2\alpha)j}{\alpha j} \asymp \left(\left(\frac{1+2\alpha}{\alpha}\right)^\alpha \cdot \left(\frac{1+2\alpha}{1+\alpha}\right)^{1+\alpha}\right)^j.$$

Für $q_j$ gilt darum

$$\begin{aligned} q_j &\geq \frac{1}{3} \cdot \sum_{i=0}^{j} \binom{j+2i}{i} \cdot \left(\frac{k-1}{k}\right)^i \cdot \left(\frac{1}{k}\right)^{i+j} \\ &\asymp \left(\left(\frac{1+2\alpha}{\alpha}\right)^\alpha \cdot \left(\frac{1+2\alpha}{1+\alpha}\right)^{1+\alpha} \cdot \left(\frac{k-1}{k}\right)^\alpha \cdot \left(\frac{1}{k}\right)^{1+\alpha}\right)^j. \end{aligned}$$

Dieser Ausdruck wird maximal für $\alpha = \frac{1}{k-2}$ und damit ist

$$q_j \asymp \left(\frac{1}{k-1}\right)^j.$$

Bei der zufälligen Belegung der Variablen am Anfang von $try$ ist die Anzahl $j$ der falsch gewählten Werte binomialverteilt:

$$\mathbf{Pr}[X_0 = j] = \frac{\binom{n}{j}}{2^n}$$

Die Erfolgswahrscheinlichkeit von $try$ ist deshalb

$$p = \sum_{j=0}^{n} \binom{n}{j} \cdot 2^{-n} \cdot q_j \asymp 2^{-n} \cdot \sum_{j=0}^{n} \binom{n}{j} \cdot \left(\frac{1}{k-1}\right)^j = 2^{-n} \cdot \left(1 + \frac{1}{k-1}\right)^n.$$

**Satz**: Bis auf einen polynomiellen Faktor ist die Wahrscheinlichkeit, dass $try$ die erfüllende Belegung findet, mindestens $\left(\frac{1}{2} \cdot \left(\frac{k}{k-1}\right)\right)^n$.

Dabei wurden $i$ Schritte in die falsche und $i+j$ Schritte in die richtige Richtung gemacht ($i \leq j \leq n$), weshalb die totale Anzahl Schritte nicht grösser als $3n$ ist.

Mit $p$ kann man nun die Anzahl *try*-Aufrufe $t$ berechnen, die mindestens gemacht werden müssen, damit die Fehlerwahrscheinlichkeit $\varepsilon$ garantiert werden kann:

$$(1-p)^t \leq \varepsilon \;\Rightarrow\; t \geq \frac{\log(\varepsilon)}{\log(1-p)},$$

die Fehlerwahrscheinlichkeit nimmt also bei einer Verdoppelung der *try*-Aufrufe quadratisch ab.

Bei der ganzen Analyse wurde angenommen, dass es nur eine einzige erfüllende Belegung $a^*$ gibt. Falls es mehrere gibt, dann kann $p$ nur grösser werden. Der Algorithmus wird also bei mehrfach erfüllbaren Formeln noch besser arbeiten, als mit dieser Worst-Case-Analyse berechnet wurde.

## 2.3 Welzl's Analyse

Wie aus Schöning's Analyse ersichtlich ist, wurde nur der exponentielle Teil von $p$ berechnet. Die Analyse von Welzl liefert zusätzlich eine saubere Abschätzung für den noch fehlenden polynomiellen Faktor. Wiederum wird die Erfolgswahrscheinlichkeit $p$ für *try* berechnet, falls es eine erfüllende Belegung gibt.

Für die verallgemeinerte Binomialreihe $\mathcal{B}_2()$ gilt:

$$\begin{aligned}
\mathcal{B}_2(z) &= \sum_{i=0}^{\infty} \binom{2i+r}{i} \cdot \frac{r}{2i+r} \cdot z^i \\
&= \left( \sum_{i=0}^{\infty} \binom{2i+1}{i} \cdot \frac{z^i}{2i+1} \right)^r \\
&= \left( \frac{1-\sqrt{1-4z}}{2z} \right)^r
\end{aligned}$$

Der Übersichtlichkeit wegen wird $\frac{1}{k}$ durch $q = \frac{1}{k} < \frac{1}{2}$ ersetzt. $X_t$ ist wiederum die Hammingdistanz zu der erfüllenden Belegung bei der Iteration $t$. Sei $N_j$

die Zufallsvariable, die die Anzahl Iterationen zählt, bis der Zustand $X_{N_j} = 0$ zum ersten Mal erreicht wird, wenn sich der Algorithmus im Zustand $j$ befindet. Dann ist $N_j$ endlich mit Wahrscheinlichkeit

$$\begin{aligned}
\mathbf{Pr}[N_j < \infty] &= \sum_{i=0}^{\infty} \binom{2i+j}{i} \cdot \frac{j}{2i+j} \cdot (1-q)^i \cdot q^{i+j} \\
&= q^j \cdot \sum_{i=0}^{\infty} \binom{2i+j}{i} \cdot \frac{j}{2i+j} \cdot ((1-q)q)^i \\
&= q^j \cdot (\mathcal{B}_2((1-q)q))^j \\
&= q^j \cdot \left( \frac{1-\sqrt{1-4\,(1-q)\,q}}{2\,(1-q)\,q} \right)^j \\
&= \left( \frac{q}{1-q} \right)^j .
\end{aligned}$$

Die Anzahl Schritte, die im Zustand $j$ bei $i$ Schritten in die falsche Richtung gemacht werden, beträgt $s(i,j) = 2i + j$. Die erwartete Anzahl Schritte bis zum Zustand 0, falls dieser erreicht werden kann, ist dann

$$\begin{aligned}
\mathbf{E}[N_j | N_j < \infty] &= \frac{1}{\mathbf{Pr}[N_j < \infty]} \cdot \sum_{i=0}^{\infty} s(i,j) \cdot \mathbf{Pr}[s(i,j) < \infty] \\
&= \frac{1}{\mathbf{Pr}[N_j < \infty]} \cdot \sum_{i=0}^{\infty} (2i+j) \cdot \binom{2i+j}{i} \cdot \frac{j}{2i+j} \cdot (1-q)^i \cdot q^{i+j} \\
&= \left( \frac{1-q}{q} \right)^j \cdot j \cdot q^j \cdot \sum_{i=0}^{\infty} \binom{2i+j}{i} \cdot ((1-q)\,q)^i \\
&= j\,(1-q)^j \cdot \frac{\left(\frac{1}{1-q}\right)^j}{1-2q} \\
&= \frac{j}{1-2q}.
\end{aligned}$$

Weil $\mathbf{Pr}[X_0 = j] = \frac{1}{2^n} \cdot \binom{n}{j}$ ist, gilt für die Zufallsvariable $N$ (Anzahl Schritte bis zum Zustand 0)

$$\mathbf{Pr}[N < \infty] = \sum_{j=0}^{n} \binom{n}{j} \cdot 2^{-n} \cdot \mathbf{Pr}[N_j < \infty]$$

$$
\begin{aligned}
&= 2^{-n} \cdot \sum_{j=0}^{n} \binom{n}{j} \cdot \left(\frac{q}{1-q}\right)^j \\
&= 2^{-n} \cdot \left(1 + \frac{q}{1-q}\right)^n \\
&= \left(\frac{1}{2\,(1-q)}\right)^n .
\end{aligned}
$$

Falls der Random Walk je in den Zustand 0 gelangt, dann ist $N$ durchschnittlich

$$
\begin{aligned}
\mathbf{E}[N|N<\infty] &= \sum_{i=0}^{\infty} i \cdot \mathbf{Pr}[N=i|N<\infty] \\
&= \sum_{i=0}^{\infty} i \cdot \sum_{j=0}^{n} \binom{n}{j} \cdot 2^{-n} \cdot \mathbf{Pr}[N_j=i|N<\infty] \\
&= \frac{2^{-n}}{\mathbf{Pr}[N<\infty]} \cdot \sum_{j=0}^{n} \binom{n}{j} \cdot \sum_{i=0}^{\infty} i \cdot \mathbf{Pr}[N_j=i] \\
&= \frac{2^{-n}}{\mathbf{Pr}[N<\infty]} \cdot \sum_{j=0}^{n} \binom{n}{j} \cdot \mathbf{E}[N_j|N_j<\infty] \cdot \mathbf{Pr}[N_j<\infty] \\
&= (1-q)^n \cdot \sum_{j=0}^{n} \binom{n}{j} \cdot \frac{j}{1-2q} \cdot \left(\frac{q}{1-q}\right)^j \\
&= (1-q)^n \cdot \sum_{j=0}^{n} \left(\binom{n-1}{j-1} + \binom{n-1}{j}\right) \cdot \frac{j}{1-2q} \cdot \left(\frac{q}{1-q}\right)^j \\
&= \frac{(1-q)^n}{1-2q} \cdot \sum_{j=0}^{n} \binom{n-1}{j-1} \cdot \left(1 + \frac{n-j}{j}\right) \cdot j \cdot \left(\frac{q}{1-q}\right)^j \\
&= \frac{n\,(1-q)^n}{1-2q} \cdot \sum_{j=0}^{n} \binom{n-1}{j-1} \cdot \left(\frac{q}{1-q}\right)^j \\
&= \frac{n\,(1-q)^n}{1-2q} \cdot \left(1 + \frac{q}{1-q}\right)^{n-1} \cdot \frac{q}{1-q} \\
&= \frac{n\,(1-q)^n}{1-2q} \cdot \frac{1}{(1-q)^n} \cdot q \\
&= \frac{nq}{1-2q}.
\end{aligned}
$$

Für $\mu = \mathbf{E}[N|N < \infty]$ gilt die Markovungleichung

$$\mathbf{Pr}[N > \lambda\mu|N < \infty] < \frac{1}{\lambda}$$

beziehungsweise $\mathbf{Pr}[N \leq \lambda\mu|N < \infty] \geq 1 - \frac{1}{\lambda}$. Wegen $N < \infty$ ist

$$\mathbf{Pr}[N \leq \lambda\mu|N < \infty] = \frac{\mathbf{Pr}[N \leq \lambda\mu]}{\mathbf{Pr}[N < \infty]} \geq 1 - \frac{1}{\lambda},$$

woraus

$$\mathbf{Pr}[N \leq \lambda\mu] \geq \left(1 - \frac{1}{\lambda}\right) \cdot \mathbf{Pr}[N < \infty] = \left(1 - \frac{1}{\lambda}\right) \cdot \left(\frac{1}{2(1-q)}\right)^n$$

folgt. Für $k \geq 3$, $q = \frac{1}{k}$ und $\lambda = 3(k-2)$ wird daraus

$$\mathbf{Pr}[N \leq 3n] \geq \left(1 - \frac{1}{3(k-2)}\right) \cdot \left(\frac{1}{2(1-q)}\right)^n,$$

weshalb

$$p \geq \frac{2}{3} \cdot \left(\frac{k}{2(k-1)}\right)^n \quad \forall k.$$

Dies führt zum folgenden

**Satz**: Schöning's Algorithmus löst probabilistisch jedes $k$-SAT-Problem mit $n$ Variablen in

$$O(\left(\frac{2(k-1)}{k}\right)^n).$$

## 2.4 Verallgemeinerung auf CSP

Wie aus der *try*-Prozedur ersichtlich ist, werden die Eigenschaften und die Bedeutung von SAT-Formeln kaum in die Lösungsidee einbezogen. Deshalb lässt sich der Algorithmus auch auf CSP's, die allgemeinere Form von Erfüllbarkeitsproblemen, anwenden.

Ein diskretes CSP (constraint satisfaction problem) besteht aus $n$ Variablen $x_1, ..., x_n$, die jeweils $d$ verschiedene Werte annehmen können: $x_i \in$

$\{c_{i1}, c_{i2}, ..., c_{id}\}$. Weiter enthält es eine Menge von $m$ Restriktionen $\{C_1, ..., C_m\}$, welche als Funktionen, Algorithmen oder Klauseln festlegen, ob zwischen den Variablen eine gewünschte Beziehung erfüllt ist oder nicht. Eine solche Restriktion $C_j$ sei erfüllt, falls $C_j(x_1, ..., x_n) = 1$.
Die Frage ist nun, ob es eine Belegung der $x_i$'s gibt, welche für alle $j$ zu $C_j(x_1, ..., x_n) = 1$ führt.
Häufig betrifft eine Restriktion nicht alle Variablen des CSP's, sondern nur einen kleinen Teil davon. Sei $l$ die maximale Anzahl Variablen, die eine Restriktion des CSP's betrifft. Wenn eine Bedingung nicht erfüllt ist, wählt Schöning's Algorithmus bei CSP's eine beliebige Variable aus und ersetzt ihre Belegung durch eine der $d-1$ anderen möglichen Belegungen.
Die Rolle von $k$ in der Analyse des $k$-SAT-Problems nimmt bei CSP's deshalb $l(d-1)$ ein, da pro Iteration höchstens eine aus $l$ Variablen mit jeweils einem der anderen $d-1$ Werte belegt werden kann.
Die Wahrscheinlichkeit, bei einer zufälligen Belegung genau $j$ von $n$ Variablen mit einem falschen aus jeweils $d$ möglichen Werten zu belegen, beträgt

$$\mathbf{Pr}[j \text{ von } n \text{ Variablen werden falsch gewählt}] = \binom{n}{j} \cdot \left(\frac{1}{d}\right)^{n-j} \cdot \left(\frac{d-1}{d}\right)^{j}.$$

Die Analyse des SAT-Problems kann damit übernommen werden wie folgt:

$$\begin{aligned}
p &\geq \sum_{j=0}^{n} \binom{n}{j} \left(\frac{1}{d}\right)^{n-j} \cdot \left(\frac{d-1}{d}\right)^{j} \cdot q_j \\
&= \sum_{j=0}^{n} \binom{n}{j} \left(\frac{1}{d}\right)^{n-j} \cdot \left(\frac{d-1}{d}\right)^{j} \cdot \left(\frac{1}{k-1}\right)^{j} \\
&= d^{-n} \sum_{j=0}^{n} \binom{n}{j} \cdot \left(\frac{d-1}{k-1}\right)^{j} \\
&= d^{-n} \left(1 + \frac{d-1}{k-1}\right)^{n} \quad \text{für } k = l(d-1).
\end{aligned}$$

Wenn aber $d > 2$ ist, gibt es nicht nur Zustandsänderungen von $j$ nach $j+1$ bzw. $j-1$, sondern man kann trotz des "Verbesserungsschrittes" im Zustand $j$ bleiben, wenn etwa eine falsch belegte Variable mit einem anderen falschen Wert belegt wird. Der Random Walk wird deshalb vom Zustand $j$ in die Zustände $j-1$, $j$, $j+1$ mit Wahrscheinlichkeiten $\frac{1}{l(d-1)}$, $\frac{d-2}{l(d-1)}$ bzw. $\frac{l-1}{l}$

übergehen. Zählt man Umbelegungen der Variablen, die den Zustand nicht ändern, mit demjenigen Übergang zu einem Schritt zusammen, der ihn dann schliesslich doch noch ändert (etwa $j \to j \to \ldots \to j \to j-1$ zu $j \to j-1$), dann ist die Wahrscheinlichkeit einer solchen Folge

$$\frac{1}{l(d-1)} \cdot \sum_{k=0}^{\infty} \left(\frac{d-2}{l(d-1)}\right)^k = \frac{1}{l(d-1)} \cdot \frac{1}{1-\frac{d-2}{l(d-1)}} = \frac{1}{(l-1)(d-1)+1}.$$

Die vormalige Erfolgswahrscheinlichkeit von $\frac{1}{k}$ bzw. $\frac{1}{l(d-1)}$ kann dann also ersetzt werden durch $\frac{1}{(l-1)(d-1)+1}$:

$$p \geq d^{-n}\left(1+\frac{d-1}{k-1}\right)^n = d^{-n} \cdot \left(1+\frac{1}{l-1}\right)^n \text{ für } d > 2.$$

Der Algorithmus kann aber nur während einer beschränkten Anzahl (z.B. $c$) Zwischenschritten in demselben Zustand verharren, weshalb die obige Wahrscheinlichkeit nicht exakt, sondern nur beliebig genau erreicht werden kann. Deshalb lässt sich dieser polynomielle Mehraufwand ($3cn$ statt $3n$ Iterationen) zusammenfassen durch Einfügen eines beliebig kleinen Mehraufwandes $\varepsilon$ in die Basis der Exponentialfunktion $d\,(1-1/l))^n$, was zu folgendem Satz führt:

**Satz**: CSP's mit $d > 2$ können für jedes $\varepsilon > 0$ vom Schöningalgorithmus probabilistisch gelöst werden in

$$O(\left(d\left(1-\frac{1}{l}\right)+\varepsilon\right)^n).$$

# Kapitel 3

# Das Threshold-Phänomen

Nicht jede Instanz des k-SAT-Problems ist erfüllbar. Für den Schöningalgorithmus aber führen k-SAT-Formeln, die nicht erfüllbar sind, immer zu einer maximalen Laufzeit. Aus diesem und anderen Gründen möchte man möglichst früh wissen, ob eine k-SAT-Formel lösbar ist. Natürlich kann auf diese Frage keine allgemeine und immer richtige Antwort gegeben werden, weil dazu das SAT-Problem selber gelöst werden müsste. Für die im folgenden diskutierte SAT-Problemklasse kann aber die Erfüllbarkeit schon vor dem Lösen recht genau bestimmt werden.

## 3.1 Random-k-SAT

Eine k-SAT-Formel mit $n$ Variablen und $m$ Klauseln kann erzeugt werden wie folgt: Jeweils $k$ Variablen werden zufällig (ohne Zurücklegen) aus $n$ Variablen gezogen. Mit Wahrscheinlichkeit $1/2$ wird danach jede dieser $k$ Variablen negiert. Werden auf diese Weise $m$ Klauseln generiert, entsteht eine Random-k-SAT-Formel, deren Erfüllbarkeit im folgenden untersucht werden soll.

## 3.2 Klauseln/Variablen-Verhältnis

Jede Klausel einer k-SAT-Formel schränkt die Anzahl möglicher Lösungen für diese Formel ein. Hat eine Formel nur sehr wenige Klauseln, dann schränken diese die Lösungsmenge folglich nur schwach ein und es gibt viele erfüllende

Belegungen. Wenn es andererseits viele Klauseln gibt, werden im schlimmsten Fall alle Belegungen verhindert und die Formel wird unerfüllbar. Wird in Random-k-SAT die Anzahl Klauseln im Verhältnis zu den vorhandenen Variablen erhöht, muss also die Wahrscheinlichkeit, dass die Formel trotzdem noch erfüllt werden kann, kontinuierlich abnehmen. Der folgende Versuch zeigt diese Eigenschaft von Random-k-SAT.

**Versuch**: Es werden Random-3-SAT-Formeln mit einer konstanten Anzahl ($n = 15$) Variablen und verschiedener Anzahl Klauseln ($m$) generiert, für jedes Klauseln/Variablen-Verhältnis $c = \frac{m}{n}$ jeweils 1000 Instanzen. Gemessen wird der Anteil erfüllbarer Instanzen in Abhängigkeit von $c$.
**Ergebnis**: Für $n = 15$ ist der Anteil der erfüllbaren Formeln bei $c < 3.0$ fast 100%. Im Bereich $3.0 \leq c \leq 7.0$ nimmt die Erfüllbarkeit kontinuierlich ab, wobei für $c \approx 4.8$ etwa die Hälfte der Formeln nicht mehr erfüllbar ist. Ab $c > 7.0$ wird der Anteil der erfüllbaren Formeln vernachlässigbar klein (Abb. 3.1).


Abbildung 3.1: Anteil erfüllbarer Random-3-SAT-Formeln in Abhängigkeit des Klausel/Variablen-Verhältnisses $c$ bei $n = 15$ Variablen

## 3.3 Änderung der Problemgrösse

Es ist offensichtlich, dass der Anteil erfüllbarer Instanzen auch bei Formeln mit einer grösseren Variablenzahl abnimmt, falls das Klausel/Variablen-Verhältnis grösser wird. Wie aber ändert sich diese "Übergangsregion", falls die Anzahl Variablen erhöht wird? Um diese Frage zu beantworten, wurde der folgende Versuch gemacht:

**Versuch**: Für $n = 15$, $n = 20$ und $n = 25$ Variablen wird jeweils der Anteil erfüllbarer Random-3-SAT-Formeln in Abhängigkeit des Klausel/Variablen-Verhältnisses $c$ gemessen, wobei für jedes $n$ und $c$ jeweils 1000 Testinstanzen generiert werden.

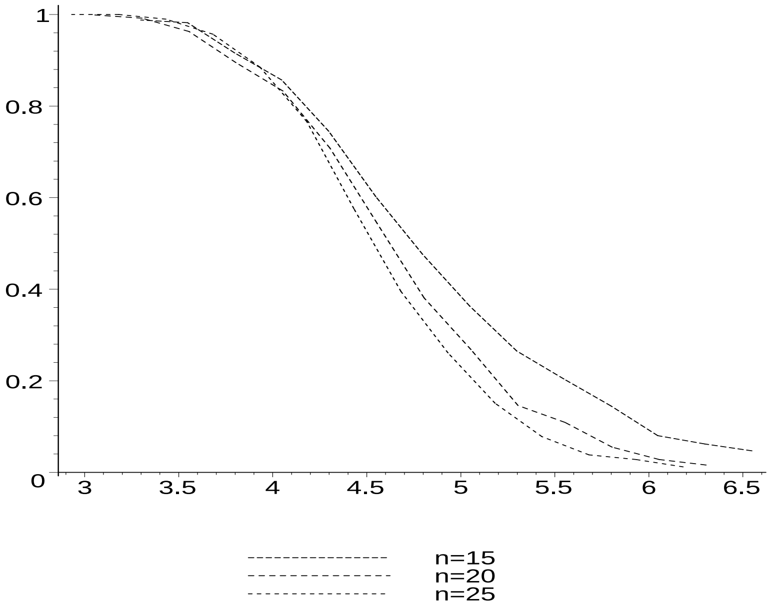



Abbildung 3.2: Veränderung der Übergangsregion bei grösseren Random-3-SAT-Problemen

**Ergebnis**: Für alle drei Problemgrössen gibt es die Übergangsregion, in der der Anteil erfüllbarer Formeln stark abnimmt. Für Formeln mit einer grösseren Variablenzahl wird diese Region schmaler, das heisst, der Übergang von erfüllbaren zu nicht erfüllbaren Formeln wird bei zunehmender Problemgrösse ($n$) abrupter. Gleichzeitig scheint sich die Übergangsregion leicht in die Richtung kleinerer $c$'s zu bewegen, das Klausel/Variablen-Verhältnis, bei welchem etwa 50% der Formeln erfüllbar sind, wird also etwas kleiner für grösser werdende $n$ (Abb. 3.2).

## 3.4 Genauere Resultate

Die Frage ist nun, wo dasjenige Klausel/Variablen-Verhältnis $c^*$ liegt, bei welchem genau 50% der Random-k-SAT-Formeln erfüllbar sind. Wie der vorherige Versuch nahelegt, hängt dieser Threshold-Wert $c^*$ von der Anzahl Variablen $n$ ab. Weiter ist er abhängig von der Anzahl Literale ($k$) pro Klausel, weshalb $c^* = c(n, k)$.
Im Fall $k = 3$ ist $c^*$ für grosse $n$ nach oben beschränkt durch 5.191. Eine beliebige Belegung der Variablen erfüllt jede Klausel mit einer Wahrscheinlichkeit von $\frac{7}{8}$, weshalb bei $m$ $(= cn)$ Klauseln die Formel mit Wahrscheinlichkeit $(\frac{7}{8})^{cn}$ erfüllt werden kann. Die erwartete Anzahl erfüllender Belegungen ist dann aber $2^n(\frac{7}{8})^{cn}$, und für $n \to \infty$ wird diese 0, wenn $c > \log_{\frac{8}{7}} 2 = 5.191$ ist (siehe [6]).
Genauere Analysen in [12] zeigen, dass für $k = 3$ der Thresholdwert $c^*$ im Bereich von $3.003 < c^* < 4.598$ liegen muss für grosse $n$. In [7] wurde für Random-3-SAT experimentell $c^* \approx 4.258 + 58.26n^{-5/3}$ ermittelt.

# Kapitel 4

# Schöning's Algorithmus für andere Probleme

Weil die Lösungsidee des Schöningalgorithmus so allgemein ist, stellt sich die Frage, ob bei gewissen Problemen durch ein speziell darauf angepasstes Schöningverfahren nicht noch bessere Resultate erreicht werden könnten. Natürlich müssen dabei nur diskrete CSP's (siehe 2.4) in Betracht gezogen werden, weil genau diese Probleme aus denselben Problemelementen bestehen wie SAT-Formeln.
Im folgenden werden mehrere solche Probleme vorgestellt und untersucht, welche Vereinfachungen und Verbesserungen sich für den Schöningalgorithmus und seine Laufzeit ergeben.

## 4.1 Not-All-Equal-k-SAT

**Definition Not-All-Equal-k-SAT (NAE-k-SAT)**
**Instanz**: SAT-Formel in konjunktiver Normalform (KNF) mit $k$ Literalen pro Klausel.
**Frage**: Existiert eine erfüllende Belegung, so dass in jeder Klausel mindestens ein Literal $falsch$ und mindestens ein Literal $wahr$ ist?

**Beispiel**
Instanz: $(\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee x_4) \wedge (x_1 \vee x_2 \vee \bar{x}_4)$
Erfüllende Belegung: $x_1 = wahr$, $x_2 = wahr$, $x_3 = falsch$, $x_4 = wahr$

Nicht erfüllende Belegung: $x_1 = falsch$, $x_2 = wahr$, $x_3 = wahr$, $x_4 = wahr$

**Bemerkung**
Falls eine Klausel die NAE-Bedingung nicht einhält, kann die Belegung eines zufälligen Literals invertiert werden, und die Klausel wird damit sicher erfüllt. Es ist also keine Anpassung von Schöning's Algorithmus notwendig.

**Analyse**
Wie aus der Definition des NAE-k-SAT-Problems abgeleitet werden kann, ist für jede erfüllende Belegung $a^*$ auch deren "Komplement" $\bar{a}^*$ eine zulässige Lösung; eine NAE-SAT-Instanz hat also immer null oder eine gerade Anzahl erfüllender Belegungen. Jeweils zwei solcher Belegungen $a^*$, $\bar{a}^*$ liegen auf zwei maximal voneinander entfernten "Ecken" des Hyperwürfels, den die Belegungen $a$ der Formel bilden. Dies bedeutet andererseits, dass bei einer erfüllbaren NAE-k-SAT-Formel keine Belegung $a$ weiter als $n/2$ von einer der beiden gesuchten Belegungen entfernt sein kann, also ist $d(a, a^*) \leq n/2$. Für die Analyse von *try* ergibt sich deshalb

$$p \asymp 2^{-\frac{n}{2}} \cdot \sum_{j=0}^{n/2} \binom{n/2}{j} \cdot \left(\frac{1}{k-1}\right)^j = \left(\frac{k}{2(k-1)}\right)^{\frac{1}{2}n},$$

und die Laufzeit von Schöning's Algorithmus für NAE-k-SAT wird

$$O(p(n) \cdot \left(\sqrt{\frac{2(k-1)}{k}}\right)^n).$$

Im Fall $k = 3$ entspricht dies $O(p(n) \cdot 1.155^n)$.

## 4.2 One-in-k-SAT

**Definition One-in-k-SAT (1-in-k-SAT)**
**Instanz**: SAT-Formel in konjunktiver Normalform (KNF) mit $k$ Literalen pro Klausel.
**Frage**: Existiert eine Belegung, so dass in jeder Klausel genau ein Literal den Wert *wahr* hat?

**Beispiel**
Formel: $(\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_2 \vee \bar{x}_3 \vee x_4) \wedge (\bar{x}_1 \vee x_3 \vee \bar{x}_4)$
Erfüllende Belegung: $x_1 = wahr$, $x_2 = wahr$, $x_3 = falsch$, $x_4 = falsch$
Nicht erfüllende Belegung: $x_1 = falsch$, $x_2 = wahr$, $x_3 = wahr$, $x_4 = wahr$

**Adaptierter Schöning-Algorithmus**
Für eine nicht erfüllte Klausel muss abhängig von der Anzahl erfüllter Literale $(L)$ wie folgt vorgegangen werden:
**1. Fall**: $L = 0$
Es muss eine zufällige Variable umbelegt werden.
**2. Fall**: $2 \leq L \leq k$
Weil genau ein Literal *wahr* sein sollte, müssen unter den Variablen der erfüllten Literale mindestens $L-1$ falsch belegte sein. Flippe deshalb bis auf eine zufällige Ausnahme alle Variablenbelegungen der erfüllten Literale.

**Analyse**
Sei $v_E$ die Variable desjenigen (eindeutigen) Literals, welches unter $a^*$ in dieser Klausel *wahr* ist.
Im ersten Fall beträgt die Wahrscheinlichkeit, die Hammingdistanz um eins zu verbessern, genau $\frac{1}{k}$.
Für den zweiten Fall müssen weitere Unterscheidungen gemacht werden. Bei $L = 2$ wird mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ eine zu korrigierende Variable von zwei Variablen geflippt. Für $L = 3$ wird sich die Hammingdistanz mit Wahrscheinlichkeit $\frac{1}{3}$ um mindestens 2 verbessern. Verschlechtern kann sie sich nicht, da vor dem (mehrfachen) Flip mindestens 2 dieser Literale falsch waren und nachher höchstens 2 falsch sind. Für $L \geq 4$ schlussendlich hat eine Klauselkorrektur nach obigem Schema sogar *immer* eine Verbesserung zur Folge, da mindestens drei Literale falsch waren und nach dem Flip höchstens zwei Variablen falsch sein können. Zumindest im allerersten Schritt des *try*-Calls beträgt die Verbesserungswahrscheinlichkeit also

$$\begin{aligned}
\mathbf{Pr}[X_1 \leq j-1 | X_0 = j] &= \mathbf{Pr}[L=0] \cdot \mathbf{Pr}[v_E \text{ wird geflippt} | L=0] + \\
&\quad \mathbf{Pr}[L=2] \cdot \mathbf{Pr}[\text{nicht } v_E \text{ wird geflippt} | L=2] + \\
&\quad \mathbf{Pr}[L=3] \cdot \mathbf{Pr}[v_E \text{ wird nicht geflippt} | L=3] + \\
&\quad \mathbf{Pr}[L \geq 4] \\
&\geq \frac{1}{2^k - k} \cdot \frac{1}{k} + \frac{\binom{k}{2}}{2^k - k} \cdot \frac{1}{2} + \frac{\binom{k}{3}}{2^k - k} \cdot \frac{1}{3} + \frac{\sum_{i=4}^{k} \binom{k}{i}}{2^k - k}
\end{aligned}$$

$$\geq \frac{13}{30} \quad \forall k.$$

Danach aber werden die Wahrscheinlichkeiten für $L = 0$, $L = 2$ bzw. $L = 3$ nicht mehr dieselben sein wie im ersten Schritt, und ohne genauere Analyse der Verteilung dieser Fälle gilt

$$\mathbf{Pr}[X_{t+1} \leq j - 1 | X_t = j] > \frac{1}{3}.$$

**Bemerkung**
Die effektive Anzahl der Variablenflips pro try-Call beträgt mit diesem Algorithmus nicht $3n$, sondern höchstens $3n(k-1)$.

## 4.3 k-XOR-SAT

**Definition k-XOR-SAT**
**Instanz**: SAT-Formel, deren Klauseln aus genau k exklusiv veroderten Literalen bestehen.
**Frage**: Existiert eine die Formel erfüllende Belegung?

**Beispiel**
Formel: $(\bar{x}_1 \oplus x_2 \oplus x_3) \wedge (\bar{x}_2 \oplus \bar{x}_3 \oplus x_4) \wedge (x_1 \oplus x_3 \oplus \bar{x}_4)$
Erfüllende Belegung: $x_1 = falsch$, $x_2 = wahr$, $x_3 = wahr$, $x_4 = wahr$
Nicht erfüllende Belegung: $x_1 = falsch$, $x_2 = wahr$, $x_3 = wahr$, $x_4 = falsch$

**Bemerkung**
Hier ist keine Anpassung von Schönings Algorithmus notwendig. Es ist allerdings nicht sinnvoll, Schöning's Algorithmus auf k-XOR-SAT-Probleme anzuwenden, weil für XOR-SAT-Probleme ein polynomieller Entscheidungsalgorithmus existiert: jede Klausel (z.B. $(x_1 \oplus \bar{x}_2 \oplus x_3)$) muss in eine Gleichung $(x_1 + (1 - x_2) + x_3 \equiv 1 \ (mod\ 2))$ umgewandelt werden, wonach das entstehende lineare Gleichungssystem mit dem Gaussalgorithmus in $O(n^3)$ gelöst oder als nicht lösbar erkannt werden kann.

## 4.4 Färbungen mit niedriger Diskrepanz

**Definition Diskrepanz-Problem**
**Instanz**: Grundmenge $V = 1..n$, Teilmengen $S_j \subset V$, $j = 1..m$, $a_j$, $b_j \in \mathbb{Z}$ mit $-|S_j| \leq a_j \leq b_j \leq |S_j|$
**Frage**: Können die Elemente $v \in V$ mit einem Wert $f(v) \in \{-1, 1\}$ belegt werden, so dass gilt:

$$a_j \leq discr(S_j) \leq b_j \quad \forall j,$$

wo

$$discr(S_j) = \sum_{v \in S_j} f(v)\ ?$$

**Bemerkung**: Für $b_j = a_j = 0$ und $|S_j|$ gerade $\forall j$ erhält man eine gleichmässige Färbung der Teilmengen $S_j$ mit den beiden "Farben" $+1/-1$, falls eine solche existiert.

**Adaptierter Schöning-Algorithmus**
Falls eine Menge die Diskrepanzbedingung nicht erfüllt, wird zuerst festgestellt, ob sie die obere $(b_j)$ oder die untere $(a_j)$ Diskrepanzschranke nicht einhält. Im ersten Fall wird unter denjenigen Elementen, die $+1$ sind, ein zufälliges auf $-1$ gesetzt, im zweiten Fall umgekehrt.

**Bemerkung**
Da die Teilmengen $S_j$ den Klauseln im k-SAT-Problem entsprechen und die "Färbung" mit $-1/+1$ der Belegung mit Wahrheitswerten, gilt

$$k := \max_j \{|S_j|\}.$$

Gemäss Schöning's Analyse beträgt die Laufzeit dann

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

Schöning's Algorithmus ist also für dieses Problem umso weniger geeignet, je grösser die Teilmengen $S_j$ sind, da die Laufzeit dann gleich schlecht wie die der trivialen Enumerations-Lösung wird.

## 4.5 k-Colorability

**Definition k-Colorability-Problem**
**Instanz**: Ungerichteter, ungewichteter Graph $G = (V, E)$, $k \in \mathbb{N}$ mit $k \geq 3$
**Frage**: Existiert eine Färbung $f(v) \in \{1..k\}$ der Knoten $v \in V$, so dass für jede Kante $e = (i, j) \in E$ gilt: $f(i) \neq f(j)$?
**Bemerkung**: 2-Colorability ist in Polynomialzeit lösbar.

**Adaptierter Schöning-Algorithmus**
Wähle bei einer Kante, deren beide Knoten dieselbe Farbe haben, einen Knoten zufällig aus und belege ihn zufällig mit einer der $k-1$ übrigen Farben.

**Bemerkung**
Die Analyse von Schöning für allgemeine CSP-Probleme lässt sich eins zu eins auf dieses Problem übertragen mit $d := k$ (Anzahl mögliche Farben eines Knotens) und $l := 2$ (die Färbungsbedingung betrifft jeweils 2 Knoten), woraus sich die Komplexität $O(p(n) \cdot (k/2)^n)$ ergibt.

## 4.6 Bin Packing Problem

**Definition Bin Packing Problem (BPP)**
**Instanz**: Eine Menge $V = \{1, 2, ..., n\}$ von Gegenständen $i$ der Grösse $g_i$, eine Menge $B = \{1, 2, ..., k\}$ von Behältern $j$ der Grösse $b_j$.
**Frage**: Gibt es eine Zuweisung $f : V \to B$ der Gegenstände zu den Behältern, so dass gilt:

$$\sum_{i:\, f(i)=j} g_i \leq b_j \quad \forall j \ ?$$

**Bemerkung**: Für $k = 2$ existiert ein polynomieller Algorithmus.

**Adaptierter Schöning-Algorithmus**
Wähle aus einem Behälter, der überfüllt ist, einen zufälligen Gegenstand und lege ihn zufällig in einen der übrigen $k - 1$ Behälter.

**Bemerkung**
In Schöning's Analyse gilt für dieses Problem $d := k$; $l$ ist die Anzahl der Gegenstände in einem Behälter, welche die Füllbedingung einzuhalten

haben. Im schlimmsten Fall kommen durch den Algorithmus kurzfristig alle Gegenstände in einen einzigen überfüllten Behälter. In diesem Fall wäre $l = n$, und Schöning's Analyse ergäbe

$$\left(k\frac{(n-1)}{n}\right)^n \approx k^n,$$

was nicht besser ist als vollständige Enumeration.

## 4.7 Set Cover

**Definition Set-Cover-Problem**
**Instanz**: Menge $V = \{1, 2, ..., n\}$, Teilmengen $S_j$, $j = 1..w$, mit $S_j \subset V$, $k \in \mathbb{N}$ mit $k < w$.
**Frage**: Gibt es Indizes $l_i \in \{1, 2, ..., w\}$, $i = 1..k$, so dass $S_{l_1} \cup S_{l_2} \cup ... \cup S_{l_k} = V$ (können mit höchstens $k$ Teilmengen alle Elemente von $V$ abgedeckt werden)?

**Adaptierter Schöning-Algorithmus**
Wähle zufällig $k$ Teilmengen $S_j$ aus (ohne Zurücklegen). Wähle für ein $i$, das in keiner dieser gewählten Teilmengen enthalten ist, eine beliebige Teilmenge ab und ersetze sie durch eine zufällige, die $i$ enthält.

**Bemerkungen**
Sei $M := \max_i\{|\{S_j | i \in S_j\}|\} \leq w - 1$ die maximale Anzahl von Teilmengen, in denen ein gleiches Element vorkommt (der Fall $M = w$ ist uninteressant, weil er auf ein kleineres Problem reduzierbar ist). Dann gilt $l := k$ (für jedes Element muss die Bedingung erfüllt sein, dass es in mindestens einer der gewählten $k$ Teilmengen ist) und $d := M$ (falls eine Teilmengenkombination ein Element $i$ nicht enthält, dann kann aus denjenigen Teilmengen gewählt werden, in denen $i$ vorkommt). Dies ergibt

$$\left((w-1)\frac{k-1}{k}\right)^k,$$

weshalb dieser Ansatz auch nicht wesentlich besser ist als Enumeration.

**Andere Möglichkeit**
Transformiere das Problem in ein SAT-Problem: Seien $s_j \in \{0,1\}$ Indikatorvariablen für die Wahl der Teilmengen $S_j$ ins Set Cover. Für jedes $i$, bilde eine Restriktion vom Typ

$$\sum_{j:i \in S_j} s_j \geq 1.$$

Nun muss noch gewährleistet sein, dass nicht mehr als $k$ Teilmengen gewählt werden. Deshalb muss auch noch die Restriktion $s_1 + s_2 + ... + s_w \leq k$ eingehalten werden.

**Bemerkung**
$d$ ist offensichtlich 2, und die maximale Klauselgrösse ist $w$. Deshalb wird die Laufzeit bis auf einen polynomiellen Faktor

$$\left(2\frac{(w-1)}{w}\right)^w \approx 2^w,$$

was es auch nicht bringt.

# Kapitel 5

# Schwierige SAT-Probleme für Schöning's Algorithmus

Im allgemeinen ist für Local-Search-Algorithmen die Wahrscheinlichkeit, eine zulässige Lösung zu finden, umso grösser, je mehr Lösungen es gibt. Insbesondere sollte also auch Schöning's Ansatz bei mehrfach erfüllbaren Formeln schnell ein Resultat finden. Dies bedeutet umgekehrt, dass SAT-Probleme, welche nur eine einzige erfüllende Belegung haben, für Schöning im allgemeinen schwierig zu lösen sind. Falls überhaupt keine erfüllende Belegung existiert, dann liefert Schöning's Algorithmus natürlich immer die richtige Antwort ("nicht erfüllbar") mit maximalem Aufwand, weshalb dieser Fall nicht interessiert.
Es gibt mehrere Methoden, einfach erfüllbare SAT-Probleme zu generieren. Im folgenden Kapitel werden stellvertretend zwei dieser Methoden aufgezeigt und ihre Vor- und Nachteile diskutiert. Anschliessend wird untersucht, ob es für Schöning Unterschiede im Schwierigkeitsgrad solcher einfach erfüllbaren SAT-Probleme gibt.

## 5.1 Festlegen der erfüllenden Belegung in Random-3-SAT

Man kann eine eine erfüllbare 3-SAT-Formel in $n$ Variablen generieren, indem man zuerst eine erfüllende Belegung festlegt und danach eine Formel zufällig aus genau denjenigen Klauseln bildet, welche nicht im Widerspruch zur

gewählten Belegung stehen.
Der Nachteil dieses Ansatzes ist, dass man keine Sicherheit über die Anzahl erfüllender Belegungen hat. Es wird also unter Umständen starke Schwankungen in der "Schwierigkeit" solcher Probleme geben.
Der Vorteil dieser Methode ist, dass auch sehr einfach Instanzen mit mehreren erfüllenden Belegungen erzeugt werden können. Es dürfen dann einfach nur solche Klauseln in die Formel aufgenommen werden, die kompatibel zu allen gewählten Lösungen sind.

## 5.2 Faktorisieren eines Primzahlenproduktes

In der Digitaltechnik sind elektronische Schaltkreise entwickelt worden, welche arithmetische Operationen wie zum Beispiel die Addition oder den Vergleich zweier ganzer Zahlen ausführen. Alle diese Schaltkreise können mittels einfachster logischer Grundbausteine (AND-, OR-, NOT-Gatter) zusammengebaut werden. Für die Multiplikation zweier ganzer Zahlen kann zum Beispiel der in Abb. 5.1 gezeigte 4-Bit-Schaltkreis verwendet werden. Dabei ist der Input links und oben die binäre Repräsentation der Zahlen $A$

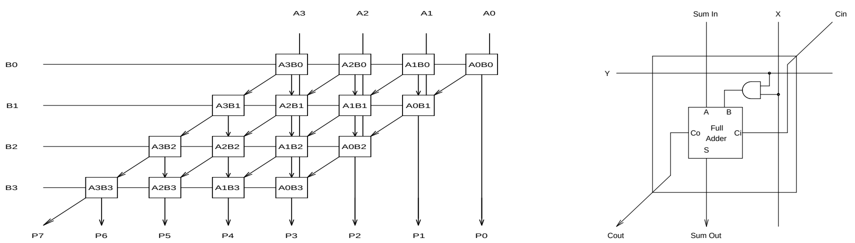



Abbildung 5.1: Elektronischer Schaltkreis für die Multiplikation

und $B$, während unten das gesuchte Produkt $P = A \cdot B$ als Output geliefert wird.
Die logischen Gatter können nun aufgefasst werden als Bedingungen, welche zwischen den Eingaben $A$, $B$ und der Ausgabe $P$ erfüllt sein müssen, und die nichtoperativen Verbindungen zwischen zwei Gattern können als temporärer Zustand verstanden werden. Werden sämtliche solchen Verbindungen als boolsche Variablen modelliert und die logischen Gatter als Klauseln, welche die Gatterfunktion zwischen den einzelnen temporären Zuständen festlegen,

dann kann sowohl der Input als auch der Output der Schaltung festgelegt und über die Erfüllbarkeit der entstandenen logischen Formel geprüft werden, ob die Schaltung korrekt arbeitet: Ist die Formel erfüllt, dann ist der Output wirklich das Produkt der beiden Inputs, sonst nicht. Es ist offensichtlich, dass für bestimmte Outputs (z.B. 24) verschiedene Inputkombinationen akzeptiert werden (z.B. (4, 6) oder (3, 8)). Wählt man hingegen den Output als das Produkt zweier Primzahlen $p_1$ und $p_2$, dann gibt es genau zwei den Schaltkreis erfüllende Inputs, nämlich ($p_1$, $p_2$) und ($p_2$, $p_1$). Erweitert man den Schaltkreis noch um eine zusätzliche Vergleichsoperation, welche 1 Bit Outputinformation zur Frage liefert, ob $A \leq B$ war, dann kann umgekehrt durch Festlegen ebendieser Information garantiert werden, dass es eine einzige Inputkombination gibt, welche sämtliche Bedingungen erfüllt. Codiert man also eine solche Multiplier-Schaltung in eine SAT-Formel und legt anschliessend den Output als ein Produkt von zwei Primzahlen fest, dann erhält man ein SAT-Formel, welche genau eine erfüllende Belegung hat. In [10] sind mehrere Instanzen dieses Problemtyps publiziert worden.
Die Nachteile dieses Ansatzes sind offensichtlich: Nach der Codierung hat man durch die Wahl des Outputs zwar beinahe beliebig viele Instanzen derselben Problemgrösse zur Verfügung, doch kann die Anzahl Variablen nur schlecht kontrolliert werden, wenn man die Inputgrösse ($= n$) des Multipliers um ein Bit ändert, weil sich dann die totale Anzahl benötigter Variablen aufgrund der Komplexität des Schaltkreises ($O(n^2)$) sprunghaft ändern wird. Ebenfalls unschön ist, dass gewisse Bits des Inputs (bzw. die entsprechenden Variablen in der SAT-Formel) einfacher festgelegt werden können als andere, was dem zu testenden Algorithmus ungewollt Vorteile verschaffen könnte. In [6] wird jedoch vermutet, dass auf diese Weise erzeugte SAT-Instanzen für alle SAT-Entscheidungsalgorithmen schwierig zu erfüllen sind, und zwar unabhängig vom Lösungsansatz.

## 5.3 Eigenschaften von schwierigen Problemen

Eine für Schöning schwierig zu lösende Klasse von SAT-Problemen hat also idealerweise die folgenden Eigenschaften:

- Jede Instanz muss zwingend erfüllbar sein.

- Eine obere Schranke für die Anzahl erfüllender Belegungen sollte festlegbar oder festgelegt sein.
- Die Problemgrösse (Anzahl Variablen) sollte frei wählbar bleiben.
- Alle Instanzen gleicher Grösse sollten etwa gleich schwierig zu lösen sein (Testaufwand wird kleiner und Analyse vermutlich einfacher).
- Alle Bedingungen/Klauseln einer Instanz sollten "gleich schwierig" zu erfüllen sein (sonst sagt die Problemgrösse eventuell nichts mehr über den Schwierigkeitsgrad der Instanz aus und der Algorithmus könnte Vorteile aus der Struktur des Problemes ziehen).
- Das Problem muss in irgendeiner Weise vom Zufall abhängig sein (es sollte möglichst wenig strukturiert sein).

Im folgenden wird eine neue Klasse von SAT-Problemen vorgestellt, welche die meisten der obigen Eigenschaften besitzt.

## 5.4 Circular-3-XOR-SAT

**Definition Circular-3-XOR-SAT-Problem**
**Instanz**: 3-XOR-SAT-Formel $\phi$ in $n$ Variablen $V = \{1, 2, ..., n\}$ und $n$ Klauseln $C_i$, $i = 0..n-1$, wo jede Klausel $C_i$ die Form $(l_{i0} \oplus l_{i1} \oplus l_{i2})$ hat mit $l_{ij} \in \{x_{(i+j)\, mod\, n\ +\ 1},\ \bar{x}_{(i+j)\, mod\, n\ +\ 1}\}$.
**Frage**: Gibt es eine erfüllende Belegung für $\phi$?
**Beispiel**: $(x_1 \oplus \bar{x}_2 \oplus x_3) \wedge (x_2 \oplus \bar{x}_3 \oplus \bar{x}_4) \wedge (x_3 \oplus x_4 \oplus x_5) \wedge (\bar{x}_4 \oplus x_5 \oplus \bar{x}_6) \wedge (\bar{x}_5 \oplus \bar{x}_6 \oplus \bar{x}_7) \wedge (x_6 \oplus x_7 \oplus x_1) \wedge (\bar{x}_7 \oplus x_1 \oplus x_2)$

Im folgenden wird bewiesen, unter welchen Umständen eine solche Formel genau eine erfüllende Belegung hat.

**Definition**: Zwei Klauseln $C_1$, $C_2$ sind benachbart, wenn $C_1$ und $C_2$ Literale in genau zwei gleichen Variablen haben.
**Beispiel**: $(x_4 \oplus x_5 \oplus x_6)$ und $(\bar{x}_5 \oplus x_6 \oplus x_7)$ sind benachbart in $x_5$, $x_6$.

**Definition**: Zwei Klauseln sind nebeneinander liegend, wenn ihre sechs Variablen alle verschieden sind und sie eine lückenlose Reihenfolge bilden.

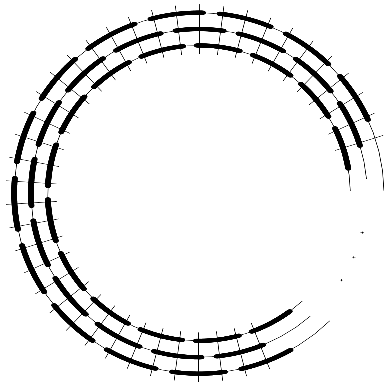


Abbildung 5.2: Aufbau einer Circular-3-XOR-SAT-Formel

**Beispiel**: $(\bar{x}_4 \oplus x_5 \oplus \bar{x}_6)$ und $(x_7 \oplus x_8 \oplus \bar{x}_9)$ sind nebeneinander liegend, ebenso $(x_6 \oplus \bar{x}_7 \oplus x_1)$ und $(\bar{x}_2 \oplus x_3 \oplus x_4)$ in einer Formel mit sieben Variablen.

**Behauptung**: Sei $\phi$ eine Circular-3-XOR-SAT-Formel mit $n$ Variablen, in der es nur positive Literale (nichtnegierte Variablen) gibt. Falls die Anzahl Variablen nicht durch drei teilbar ist ($n \neq 3k \quad \forall k \in \mathbb{N}$), dann gibt es genau eine einzige erfüllende Belegung: Alle Variablen müssen auf *wahr* gesetzt werden.
**Beweis**:
1. Fall: Seien alle Variablen *wahr* $\Rightarrow$ jede Klausel ist erfüllt $\Rightarrow$ $\phi$ ist erfüllt.
2. Fall: Seien alle Variablen $falsch$ $\Rightarrow$ jede Klausel ist nicht erfüllt $\Rightarrow$ $\phi$ ist nicht erfüllt.
3. Fall: Mindestens eine Variable ist *wahr* und mindestens eine ist $falsch$. $\Rightarrow$ Sei $C$ eine Klausel, die die auf $falsch$ gesetzte Variable enthält. Dann gilt: damit $C$ trotzdem erfüllt wird, muss genau eine der übrigen Variablen *wahr* und genau eine $falsch$ sein. $C$ enthält also genau eine erfüllte und genau zwei nichterfüllte Literale. Nun aber müssen beide Nachbarklauseln von $C$ ebenfalls mit genau einem Literal erfüllt werden (Beweis: Falls eine benachbarte Klausel beide nicht erfüllten Variablen mit $C$ teilt, muss die übrigbleibende Variable *wahr* sein, damit diese Klausel überhaupt erfüllt ist, und die Klausel ist somit ebenfalls mit genau einem Literal *wahr*; falls die benachbarte Klausel eine wahre und eine falsche Variable mit $C$ teilt und sie erfüllt werden soll, dann kann die übrigbleibende Variable nur $falsch$ sein, weshalb sie ebenfalls genau ein wahres Literal bzw. Variable enthält; in

beiden Fällen hat dieser Nachbar also genau ein erfülltes Literal, q.e.d). Mit vollständiger Induktion für den ganzen Kreis ergibt sich, dass jede Klausel genau ein erfülltes Literal hat. Wenn jede Klausel aber genau ein erfüllendes Literal hat, dann müssen genau $\frac{1}{3}$ der Variablen $wahr$ und $\frac{2}{3}$ der Variablen $falsch$ sein, da zwischen zwei erfüllten Variablen genau zwei nicht erfüllte stehen müssen (wären es weniger, wäre eine Klausel mit mehr als einem Literal erfüllt, wären es mehr, wäre eine Klausel überhaupt nicht erfüllt). Nun ist aber die Anzahl Variablen nach Voraussetzung $3k + b$ mit $b \in \{1, 2\}$, weshalb unmöglich $\frac{1}{3}$ der Variablen $wahr$ sein kann, ein Widerspruch.
Der einzige Fall, wo $\phi$ erfüllt ist, ist also Fall 1, und da es keine anderen möglichen Fälle gibt, folgt, dass die Belegung mit $wahr$ die alleinerfüllende ist.

Nun muss gezeigt werden, wie aus einer solch eingeschränkten Formel mit nur positiven Literalen eine beliebige Circular-3-XOR-SAT-Formel mit $3k+b$ Variablen erzeugt werden kann, die genau gleich viele erfüllende Belegungen hat. Dazu werden drei Operationen definiert, welche auf beliebige Circular-3-XOR-SAT-Instanzen angewendet werden können und dabei die Anzahl erfüllender Belegungen nicht ändern:

**Operation 1**: In einer Klausel werden zwei beliebige Literale negiert.

**Operation 2**: In drei Klauseln, welche zusammen dieselbe (eindeutige) Variable teilen, wird das Literal dieser Variablen jeweils negiert.

Eine Klausel, welche durch Operation 1 geändert wurde, hat genau dieselben (vier) erfüllenden Belegungen (Beweis durch Austesten aller möglichen Fälle).
Mit Operation 2 wird die erfüllende Belegung bis auf eine Ausnahme nicht geändert, denn nur die Belegung der gemeinsamen Variablen muss invertiert werden (die gemeinsame Variable kommt ja in keiner weiteren Klausel vor).

Aus Operation 1 und 2 kann eine weitere Operation zusammengesetzt werden:

**Operation 3**: In zwei nebeneinander liegenden Klauseln wird je ein beliebiges Literal negiert.

Diese Operation kann wie folgt aus der 1. und 2. Operation zusammengesetzt werden:
Seien die Klauseln oBdA $C_1 = (x_i \oplus x_{i+1} \oplus x_{i+2})$, $C_2 = (x_{i+1} \oplus x_{i+2} \oplus x_{i+3})$, $C_3 = (x_{i+2} \oplus x_{i+3} \oplus x_{i+4})$ und $C_4 = (x_{i+3} \oplus x_{i+4} \oplus x_{i+5})$. Nun lassen sich in den nebeneinander liegenden Klauseln $C_1$ und $C_4$ wie folgt die Variablen $x_{i+2}$ und $x_{i+3}$ negieren: Wende Operation 2 auf $C_1$, $C_2$ und $C_3$ an (gemeinsame Variable ist $x_{i+2}$), dann auf $C_2$, $C_3$ und $C_4$ (gemeinsame Variable ist $x_{i+3}$). Jetzt wird auf $C_2$ die Operation 1 angewendet (Variablen $x_{i+2}$, $x_{i+3}$), dann dieselbe Operation auch noch auf $C_3$ (auf dieselben Variabeln $x_{i+2}$, $x_{i+3}$). Nun ist die Formel dieselbe wie vorher, nur ist in $C_1$ $x_{i+2}$ und in $C_4$ $x_{i+3}$ negiert. Mittels einmaligem Anwenden von Operation 1 auf jede dieser beiden Klauseln können jetzt aber (wenn nötig) die Klauseln so verändert werden, wie dies durch Operation 3 alleine erreicht werden könnte.
Es ist offensichtlich, dass durch Operation 3 die Anzahl erfüllender Belegungen ebenfalls nicht geändert wird.

Betrachte nun die Struktur von Circular-3-XOR-SAT-Formeln mit $n = 3k+b$ Variablen (Abb. 5.3 und 5.4): In den Fällen $b = 1$, $b = 2$ bilden die

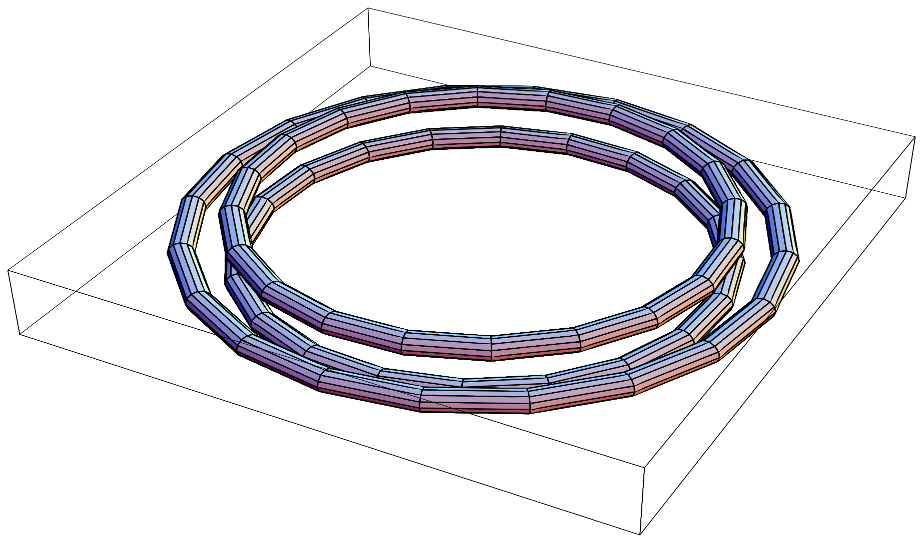


Abbildung 5.3: Struktur bei $3k$ Variablen

nebeneinander liegenden Klauseln der Formel die Struktur eines einzigen zusammenhängenden Ringes, während es bei $b = 0$ drei solche Ringe gibt.

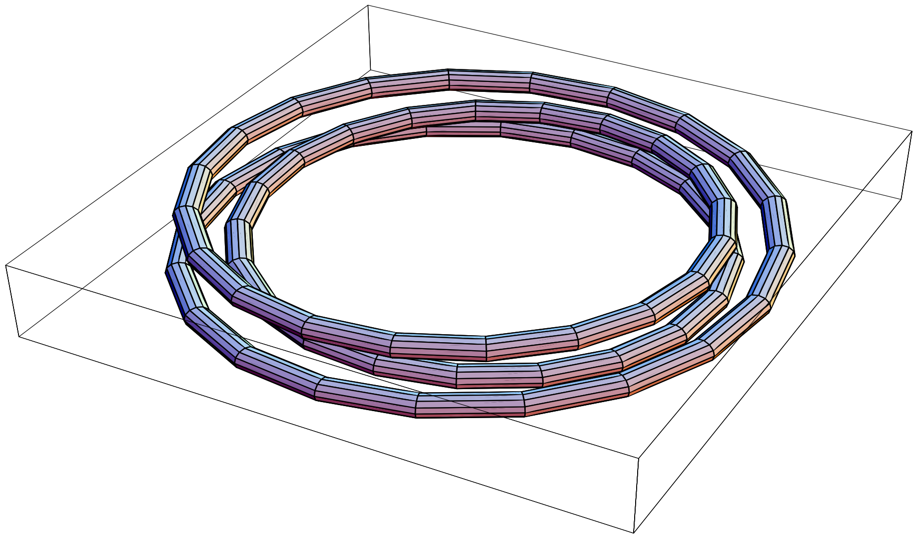


Abbildung 5.4: Struktur bei $3k + 1$ (bzw. $3k + 2$) Variablen

Für die Fälle $b = 1$, $b = 2$ darf also mittels mehrmaligem Anwenden von Operation 3 eine verallgemeinerte Operation $3^+$ eingeführt werden:

**Operation $3^+$**: Falls $n = 3k + b$ mit $b = 1$ oder $b = 2$, dann negiere in beliebigen zwei Klauseln ein beliebiges Literal.

Wird nun Operation 2 mit Operation $3^+$ kombiniert, indem Operation $3^+$ geeignet auf zwei durch Operation 2 veränderte Literale angewendet wird, dann bleibt nur noch ein einziges negiertes Literal übrig, weshalb die Kombination dieser beiden Operationen der folgenden allgemeinen Operation entspricht:

**Operation**: Falls $n \neq 3k$, negiere in einer beliebigen Klausel ein beliebiges Literal.

Dies bedeutet, dass in einer Circular-3-XOR-SAT-Formel mit $3k + 1$ oder $3k + 2$ Variablen ein beliebiges Literal negiert werden kann, ohne dass sich die Anzahl erfüllender Belegungen ändert. Wenn also bereits eine eindeutig erfüllbare Instanz existiert, kann diese Formel durch Literalnegation beliebig geändert werden und bleibt eindeutig erfüllbar.

Umgekehrt kann jede beliebige Circular-3-XOR-SAT-Formel mit $n \neq 3k$ Variablen zurückgeführt werden auf ein Circular-3-XOR-SAT-Problem mit nur positiven Variablen, welches, wie oben gezeigt, immer eindeutig erfüllbar ist. Ist also $b = 1$ oder $b = 2$, dann gibt es genau gleich viele erfüllende Belegungen in einer Circular-3-XOR-SAT-Formel mit nur positiven Literalen, wie es erfüllende Belegungen in einer allgemeinen Circular-3-XOR-Formel (mit derselben Anzahl Variablen) gibt. Daraus ergibt sich der folgende

**Satz**: Jede Circular-3-XOR-SAT-Formel mit $3k + 1$ oder $3k + 2$ Variablen hat genau eine erfüllende Belegung.

Eine 3-XOR-Klausel der Form $(x_1 \oplus x_2 \oplus x_3)$ ist äquivalent zum KNF-Ausdruck $(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3)$. Durch Anwenden dieser Regel lässt sich somit aus einer Circular-3-XOR-SAT-Formel eine entsprechende 3-SAT-Formel in KNF mit $n$ Variablen ($n \neq 3k$) und $4n$ Klauseln konstruieren, die jeweils genau eine erfüllende Belegung hat.

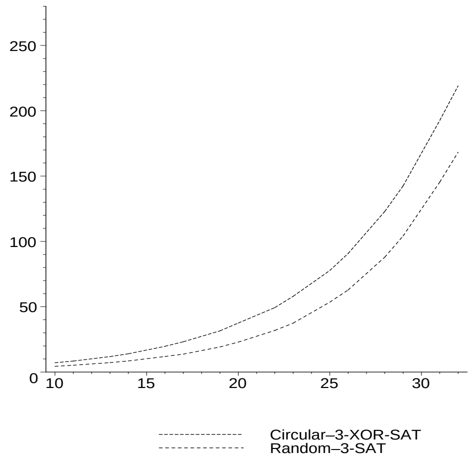
## 5.5 Versuch

Um herauszufinden, wie schwierig Circular-3-XOR-SAT im Vergleich zu anderen einfach erfüllbaren 3-SAT-Problemen für Schöning zu lösen ist, wurde der folgende Versuch gemacht.

**Versuch**: Für verschiedene Problemgrössen ($n \neq 3k$) werden jeweils 10000 Circular-3-XOR-SAT-Instanzen generiert und mit Schöning's Algorithmus ermittelt, wieviele try-Calls durchschnittlich pro Instanz benötigt werden, bis die erfüllende Belegung gefunden wird. Für Vergleichszwecke werden auch jeweils 10000 erfüllbare Random-3-SAT-Instanzen (siehe 5.1) mit einem Klausel/Variablen-Verhältnis von $c = \frac{m}{n} = 30$ erzeugt und ebenfalls die durchschnittliche Anzahl try-Calls ermittelt. $c$ ist für diesen Zweck genügend gross, da für die Anzahl zusätzlich erfüllender Belegungen $N$ $\mathbf{E}[N] \leq 2^n \cdot (\frac{7}{8})^{30n}$ gilt und mit der Markovungleichung $\mathbf{Pr}[N \geq \lambda \cdot \mathbf{E}[N]] \leq \frac{1}{\lambda}$ folgt: $\mathbf{Pr}[N \geq 1] \leq \mathbf{E}[N] < 0.037^n$ (vergleiche auch 3.4).
**Ergebnis**: Zumindest im betrachteten Bereich ($n$=10..32) ist Circular-3-XOR-SAT schwieriger zu lösen als Random-3-SAT (Tab. 5.1, Abb. 5.5). Aber auch wenn für grösseres $n$ Random-3-SAT für den Schöningalgorithmus

| Variablen $n$ | Circular-3-XOR-SAT | Random-3-SAT | Verhältnis C3X/R3 |
|---|---|---|---|
| 10 | 7.082 | 4.393 | 1.612 |
| 11 | 8.389 | 5.192 | 1.616 |
| 13 | 11.796 | 7.212 | 1.636 |
| 14 | 13.976 | 8.491 | 1.646 |
| 16 | 19.661 | 11.874 | 1.656 |
| 17 | 23.189 | 13.675 | 1.695 |
| 19 | 31.539 | 19.216 | 1.641 |
| 20 | 37.486 | 22.914 | 1.636 |
| 22 | 49.301 | 31.888 | 1.546 |
| 23 | 57.950 | 37.434 | 1.548 |
| 25 | 77.694 | 53.495 | 1.452 |
| 26 | 90.764 | 62.819 | 1.445 |
| 28 | 122.978 | 87.971 | 1.398 |
| 29 | 142.468 | 104.180 | 1.368 |
| 31 | 192.621 | 145.426 | 1.325 |
| 32 | 219.069 | 168.270 | 1.302 |

Tabelle 5.1: Durchschnittliche Anzahl try-Calls (jeweils 10000 Instanzen)



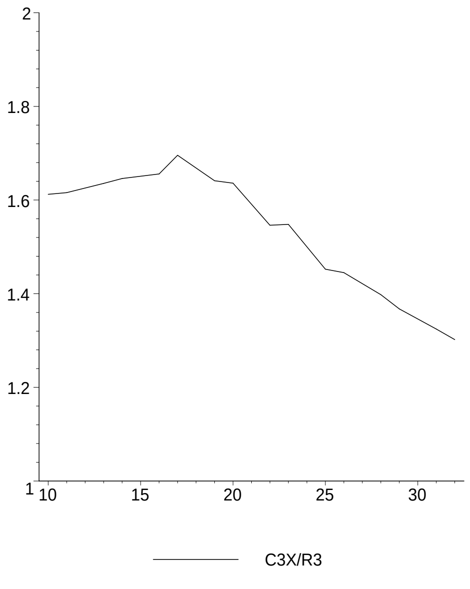




Abbildung 5.5: Durchschnittliche Anzahl benötigter try-Calls und Verhältnis bei verschiedenen Problemgrössen $n$

irgendwann schwieriger werden sollte (Abb. 5.5 rechts), so sind trotzdem nicht alle einfach erfüllbaren Probleme derselben Grösse $n$ gleich schwierig zu lösen, wie aufgrund der Tatsache, dass die Klauseln einer Formel und ihre Abhängigkeiten untereinander nicht wirklich in die Lösungsidee des Algorithmus einbezogen werden, angenommen werden könnte.

Für den Schöningalgorithmus gibt es also schwierigere und leichtere SAT-Probleme, die einfach erfüllbar sind.

# Kapitel 6

# Verbesserung des Schöningalgorithmus’

In [8] wurde eine neues Verfahren vorgestellt, welches basierend auf dem Schöningalgorithmus für $k = 3$ eine leicht bessere Laufzeit hat als der Schöningalgorithmus selber.

## 6.1 Idee

Eine maximale unabhängige Klauselmenge $C$ einer 3-SAT-Formel $F$ besteht aus Klauseln mit den folgenden Eigenschaften:

- Jede Variable von $F$ kommt in höchstens einer der Klauseln von $C$ vor.
- Es kann keine weitere Klausel von $F$ zu $C$ gehören, wenn jede Variable von $F$ in höchstens einer der Klauseln von $C$ enthalten sein soll.

Aus einer 3-SAT-Formel kann eine solche maximale unabhängige Klauselmenge offensichtlich in Polynomialzeit bestimmt werden.
Wenn die Variablen der Klauseln von $C$ belegt werden, dann haben alle übrigen Klauseln mindestens ein Literal, welches ebenfalls schon belegt ist. Das heisst, dass diese Klauseln zusammen mit den übrigen Variablen ein 2-SAT-Problem bilden, welches bekanntlich in Polynomialzeit gelöst werden kann.
Für eine erfüllende Belegung der Klauseln in $C$ kommen aber nicht alle acht,

sondern bloss sieben Belegungen in Frage, was bei der zufälligen Belegung der unabhängigen Klauseln berücksichtigt werden sollte. Wenn $\hat{m}$ die Anzahl Klauseln in der ermittelten maximalen unabhängigen Klauselmenge $C$ ist, dann gibt es deshalb zwei Fälle:

**1. Fall**: $\hat{m}$ ist eher klein
Dies bedeutet, dass es relativ viele noch unbelegte Variablen gibt, welche effizient durch das Lösen eines 2-SAT-Problems bestimmt werden können. Der Vorteil liegt hier also darin, dass bloss wenige Variablen (nämlich diejenigen der Klauseln von $C$) zufällig belegt und ausgetestet werden müssen.

**2. Fall**: $\hat{m}$ ist eher gross
In diesem Fall werden viele Variablen ($3\hat{m}$) durch die Belegung der unabhängigen Klauselmenge festgelegt, und man erspart sich einen gewissen Aufwand durch das direkte Belegen mit bloss 7 erfüllenden statt 8 möglichen Belegungen.

## 6.2 Algorithmus

Um die unabhängigen Klauseln sinnvoll zu belegen, benötigt man eine Prozedur, welche anfangs für jede Klausel eine der 7 möglichen Belegungen auswählt. Wie später ersichtlich sein wird, muss die Wahrscheinlichkeitsverteilung für die Anzahl erfüllter Literale bei dieser zufälligen Belegung wählbar bleiben ($p_1$, $p_2$, $p_3$).

$BelegeUnabh\ddot{a}ngigeKlauseln(p_1, p_2, p_3)$:
Prüfe, dass $3p_1 + 3p_2 + p_3 = 1$.
Für jede Klausel $C_i \in C$:
  Belege zufällig die Variablen von $C_i$ so, dass
  mit Wahrscheinlichkeit $3p_1$ genau 1,
  mit Wahrscheinlichkeit $3p_2$ genau 2 und
  mit Wahrscheinlichkeit $p_3$ genau 3 Literale erfüllt werden.

Unter der unbekannten erfüllenden Belegung $a^*$ seien $m_1$ die Anzahl einfach erfüllter Klauseln in $C$, $m_2$ die Anzahl zweifach erfüllter Klauseln in $C$ und $m_3$ die Anzahl dreifach erfüllter Klauseln in $C$. Offensichtlich ist $m_1 + m_2 + m_3 = \hat{m}$. Wählt man bei kleinem (bekanntem) $\hat{m}$ den Algorithmus

> *Alg*1($q_1$, $q_2$, $q_3$):
> BelegeUnabhängigeKlauseln($q_1$, $q_2$, $q_3$).
> Löse das übrigbleibende 2-SAT-Problem.

dann findet dieser eine erfüllende Belegung mit Wahrscheinlichkeit

$$\mathbf{Pr}[\text{Alg1 findet die erfüllende Belegung}] = q_1^{m_1} \cdot q_2^{m_2} \cdot q_3^{m_3},$$

weil der Algorithmus jeweils mit Wahrscheinlichkeit $q_i$ die richtige Belegung einer Klausel mit $i$ wahren Literalen wählt, dies aber für $m_i$ Klauseln richtig tun muss.
Obwohl die $m_i$ nicht bekannt sind, gilt für die Wahl von $q_1 = q_2 = q_3 = \frac{1}{7}$ für den Algorithmus Alg1

$$\mathbf{Pr}[\text{Alg1 findet die erfüllende Belegung}] = \left(\frac{1}{7}\right)^{m_1+m_2+m_3} = \left(\frac{1}{7}\right)^{\hat{m}}.$$

Im anderen Fall, wenn $\hat{m}$ gross ist, sollte der folgende Algorithmus gewählt werden.

> *Alg*2($q_1$, $q_2$, $q_3$):
> BelegeUnabhängigeKlauseln($q_1$, $q_2$, $q_3$).
> Belege alle noch übrigen Variablen zufällig mit 0 oder 1.
> Wiederhole $3n$ mal:
>     Falls die Belegung erfüllend ist, akzeptiere sie und brich ab.
>     Wähle aus einer nicht erfüllten zufälligen Klausel
>     eine zufällige Variable und belege sie um.
> Verwirf die Belegung.

Um Alg2 analysieren zu können, kann ein Ergebnis aus dem ursprünglichen Schöningalgorithmus wiederverwendet werden: Für $k = 3$ geht aus der

Schöninganalyse hervor, dass bis auf einen polynomiellen Faktor gilt

$$q_j \geq \left(\frac{1}{k-1}\right)^j = \left(\frac{1}{2}\right)^j = \left(\frac{1}{2}\right)^{d(a,a^*)}.$$

Deshalb ist

$$p_{try} \geq \frac{1}{p(n)} \cdot \mathbf{E}[\left(\frac{1}{2}\right)^{d(a,a^*)}].$$

$d(a, a^*)$ ist nach der speziellen Belegung der unabhängigen Klauseln durch Alg2 etwas besser (niedriger) als beim Schöningalgorithmus. Ansonsten kann die obige Formel eins zu eins für die Analyse von Alg2 verwendet werden. Dies wird im folgenden gemacht.

Seien oBdA $X_{i,i+1,i+2}$ für $i = 1, 4, 7, ..., 3\hat{m} - 2$ Zufallsvariablen, welche die falsch gesetzten Variablen einer unabhängigen Klausel des Typs $(x_i \vee x_{i+1} \vee x_{i+2})$ nach der zufälligen Belegung durch Alg2 zählen. Ebenso seien $X_i$ für $i = 3\hat{m} + 1, ..., n$ solche Zufallsvariablen für die übrigen Variablen $x_i$. OBdA gilt nach der zufälligen Belegung durch Alg2

$$d(a, a^*) = X_{1,2,3} + X_{4,5,6} + ... + X_{3\hat{m}-2,3\hat{m}-1,3\hat{m}} + X_{3\hat{m}+1} + ... + X_n$$

und für dessen Erfolgswahrscheinlichkeit deshalb

$$p \geq \frac{1}{p(n)} \cdot \mathbf{E}[\left(\frac{1}{2}\right)^{X_{1,2,3}+X_{4,5,6}+...+X_{3\hat{m}-2,3\hat{m}-1,3\hat{m}}+X_{3\hat{m}+1}+...+X_n}].$$

Weil die übrigen Variablen $x_i$ unabhängig voneinander gesetzt werden und für unabhängige Zufallvariablen $X$, $Y$ die Beziehung $\mathbf{E}[X \cdot Y] = \mathbf{E}[X] \cdot \mathbf{E}[Y]$ gilt, ist also

$$p \geq \frac{1}{p(n)} \cdot \mathbf{E}[\left(\frac{1}{2}\right)^{X_{1,2,3}+X_{4,5,6}+...+X_{3\hat{m}-2,3\hat{m}-1,3\hat{m}}}] \cdot \mathbf{E}[\left(\frac{1}{2}\right)^{X_i}]^{n-3\hat{m}}$$

für jedes beliebige $i \in \{3\hat{m} + 1, ..., n\}$.

$$\mathbf{E}[\left(\frac{1}{2}\right)^{X_i}] = \frac{1}{2} \cdot \left(\frac{1}{2}\right)^0 + \frac{1}{2} \cdot \left(\frac{1}{2}\right)^1 = \frac{3}{4},$$

und deswegen ist

$$p \geq \frac{1}{p(n)} \cdot \mathbf{E}[\left(\frac{1}{2}\right)^{X_{1,2,3}+X_{4,5,6}+...+X_{3\hat{m}-2,3\hat{m}-1,3\hat{m}}}] \cdot \left(\frac{3}{4}\right)^{n-3\hat{m}}.$$

Da die unabhängigen Klauseln nicht vollkommen zufällig belegt werden, muss $\mathbf{E}[(1/2)^{X_{1,2,3}}]$ gesondert berechnet werden. Betrachte die Fälle, wo die unbekannte erfüllende Belegung $a^*$ für eine unabhängige 3-SAT-Klausel $(x_1 \vee x_2 \vee x_3)$ entweder 1, 2 oder 3 erfüllte Literale vorsieht:

**Fall 1**: $a_1^* + a_2^* + a_3^* = 1$
Unter $a^*$ ist also genau ein Literal dieser Klausel *wahr*. Dann gilt

$$\begin{aligned}\mathbf{E}[\left(\frac{1}{2}\right)^{X_{1,2,3}}] &= p_1 \left(\frac{1}{2}\right)^0 + 2p_2 \left(\frac{1}{2}\right)^1 + (2p_1 + p_3) \left(\frac{1}{2}\right)^2 + p_2 \left(\frac{1}{2}\right)^3 \\ &= \frac{p_3}{4} + \frac{9p_2}{8} + \frac{3p_1}{2},\end{aligned}$$

weil mit einer einfachen Wahrscheinlichkeit von $p_1$ gerade die richtige Belegung gewählt wird (Hammingdistanz ist 0), es zwei Möglichkeiten gibt, mit einer zweifach erfüllenden Belegung höchstens ein Literal falsch zu belegen, für die Belegung mit zwei falschen Variablen es 2 Möglichkeiten der Wahrscheinlichkeit $p_1$ und eine der Wahrscheinlichkeit $p_3$ gibt, und weil bloss eine einzige Belegung mit Wahrscheinlichkeit $p_2$ gerade alle Variablen falsch belegt (entspricht Hammingdistanz 3).

**Fall 2**: $a_1^* + a_2^* + a_3^* = 2$
Hier gilt analog

$$\begin{aligned}\mathbf{E}[\left(\frac{1}{2}\right)^{X_{1,2,3}}] &= p_2 \left(\frac{1}{2}\right)^0 + (p_3 + 2p_1) \left(\frac{1}{2}\right)^1 + 2p_2 \left(\frac{1}{2}\right)^2 + p_1 \left(\frac{1}{2}\right)^3 \\ &= \frac{p_3}{2} + \frac{3p_2}{2} + \frac{9p_1}{8}.\end{aligned}$$

**Fall 3**: $a_1^* + a_2^* + a_3^* = 3$

$$\mathbf{E}[\left(\frac{1}{2}\right)^{X_{1,2,3}}] = p_3 \left(\frac{1}{2}\right)^0 + 3p_2 \left(\frac{1}{2}\right)^1 + 3p_1 \left(\frac{1}{2}\right)^2 = p_3 + \frac{3p_2}{2} + \frac{3p_1}{4}$$

Da nicht bekannt ist, wieviele erfüllte Literale jede einzelne der unabhängigen Klauseln unter der erfüllenden Belegung $a^*$ hat, konnten in der vorherigen Analyse nur bedingte Erwartungswerte für $(1/2)^{X_{1,2,3}}$ berechnet werden.

Weil aber $m_1 + m_2 + m_3 = \hat{m}$ ist und die einzelnen Klauseln unabhängig voneinander belegt werden, gilt für alle unabhängigen Klauseln zusammen

$$\mathbf{E}[\left(\frac{1}{2}\right)^{X_{1,2,3}+X_{4,5,6}+\ldots}] = \mathbf{E}[\left(\frac{1}{2}\right)^{X_{1,2,3}}] \cdot \mathbf{E}[\left(\frac{1}{2}\right)^{X_{4,5,6}}] \cdot \ldots$$
$$= \left(\frac{p_3}{4} + \frac{9p_2}{8} + \frac{3p_1}{2}\right)^{m_1} \cdot \left(\frac{p_3}{2} + \frac{3p_2}{2} + \frac{9p_1}{8}\right)^{m_2} \cdot \left(p_3 + \frac{3p_2}{2} + \frac{3p_1}{4}\right)^{m_3}.$$

Wird $p_1$, $p_2$ und $p_3$ auf 4/21, 2/21 bzw. 3/21 gesetzt, dann ergibt sich für den Algorithmus Alg2:

$$\begin{aligned} p &\geq \frac{1}{p(n)} \cdot \left(\frac{3}{4}\right)^{n-3\hat{m}} \cdot \left(\frac{3}{7}\right)^{m_1} \cdot \left(\frac{3}{7}\right)^{m_2} \cdot \left(\frac{3}{7}\right)^{m_3} \\ &= \frac{1}{p(n)} \cdot \left(\frac{3}{4}\right)^{n-3\hat{m}} \cdot \left(\frac{3}{7}\right)^{\hat{m}} \\ &= \frac{1}{p(n)} \cdot \left(\frac{3}{4}\right)^{n} \cdot \left(\frac{64}{63}\right)^{\hat{m}} \end{aligned}$$

Die beiden Algorithmen Alg1 und Alg2 kombiniert ergeben

---

$VerbesserterSchöningAlgorithmus$($F$: 3-SAT-Formel in KNF):
Finde eine maximale unabhängige Klauselmenge $C$ von $F$.
Wende auf $F$ (zusammen mit $C$) mehrmals an:
    Alg1(1/7, 1/7, 1/7),
    Alg2(4/21, 2/21, 3/21).

---

## 6.3 Laufzeitanalyse

Mit den gewählten Werten für $p_1$, $p_2$ und $p_3$ sind die Erfolgswahrscheinlichkeiten der beiden Algorithmen Alg1 und Alg2 also

$$\begin{aligned} \mathbf{Pr}[\text{Alg1 findet die erfüllende Belegung}] &= \frac{1}{p(n)} \cdot \left(\frac{1}{7}\right)^{\hat{m}}, \\ \mathbf{Pr}[\text{Alg2 findet die erfüllende Belegung}] &\geq \frac{1}{p(n)} \cdot \left(\frac{3}{4}\right)^{n} \cdot \left(\frac{64}{63}\right)^{\hat{m}}. \end{aligned}$$

Wenn
$$\frac{n}{\hat{m}} = \frac{\log(9/64)}{\log(3/4)} \approx 6.81884168$$
ist, sind beide Algorithmen bis auf einen polynomiellen Faktor gleich schnell. Weil bei kleinerem $\hat{m}$ Alg1 besser wird und bei grösserem Alg2, ist die Erfolgswahrscheinlichkeit der Algorithmenkombination aber mindestens so hoch wie die des besseren Algorithmus, weshalb
$$\left(\frac{1}{7}\right)^{\hat{m}} = \left(\frac{1}{7}\right)^{\frac{n}{6.81884}} = \left(\frac{1}{1.33026}\right)^{n}$$
zum Satz führt:

**Satz**: Der verbesserte Schöning-Algorithmus entscheidet 3-SAT probabilistisch in $O(p(n) \cdot 1.33026^n)$.

## 6.4 Weitere Verbesserung

Wüsste man, wie gross $m_1$, $m_2$ und $m_3$ sind, dann könnte der Algorithmus Alg1 verbessert werden dadurch, dass nicht alle Belegungen getestet werden müssten. Weil aber $\hat{m}$ bekannt und in jedem Fall $m_1 + m_2 + m_3 = \hat{m}$ ist, lassen sich durch polynomiell ($O(\hat{m}^2)$) viele Aufrufe sämtliche Kombinationen von $m_1$, $m_2$ und $m_3$ durchtesten und die Erfolgswahrscheinlichkeiten der beiden Algorithmen nochmals ein wenig verbessern, was zu einer Laufzeit von $O(p(n) \cdot 1.330193^n)$ führen würde.
Da der Sinn des verbesserten Schöningalgorithmus aber wohl eher darin liegt, zu zeigen, dass 3-SAT mit weniger als $O(p(n) \cdot (4/3)^n)$ Aufwand lösbar ist, werden die Details für die zusätzliche Verbesserung hier nicht weiter erläutert (siehe [8]).

# Anhang A

# Source-Code

## Implementierung des Schöningalgorithmus

Für verschiedene Tests wurde Schöning's Algorithmus implementiert und verwendet. Um ein möglichst effizientes Programm zu erhalten, wurde der Algorithmus in C geschrieben.
Das Programm verwendet mehrere Tabellen, um die Variablenflip-Operation effizient zu erledigen: Im ersten Array werden sämtliche Variablen sowohl in normaler als auch in negierter Form gehalten, damit die Negierungsoperation nicht zusätzlich implementiert werden muss. Beim Flippen der Variable müssen dann einfach beide Variablen umbelegt werden.
Im Klauselarray sind in linearer Form nur noch $k$ Pointer pro Klausel gespeichert, die auf die entsprechenden Variablen (negiert/nicht negiert) zeigen. Damit kann die Auswertung einer Klausel sehr effizient erledigt werden.
In einer weiteren Tabelle, der Variablen-nach-Klauseln-Datenstruktur, sind für jede Variable die Anzahl bzw. sämtliche sie enthaltende Klauseln (Pointer) gespeichert. Mit dieser Tabelle wird es möglich, nur solche Klauseln neu auszuwerten, welche wegen des letzten Variablenflips nicht mehr erfüllt bzw. neu erfüllt sein könnten.
In einer vierten Tabelle schlussendlich werden ebenfalls mit Pointern alle Klauseln gehalten, die nicht erfüllt sind. Damit wird die zufällige Auswahl einer unerfüllten Klausel erleichtert. Der Schöningalgorithmus schreibt zwar nicht vor, dass die zu korrigierende Klausel zufällig gewählt werden muss, doch ist diese Auswahlmethode sicher nicht falsch.

Beim Start des Programmes können die maximal zur Verfügung stehende Zeit, die Fehlerwahrscheinlichkeit sowie ein Initialwert für den Pseudozufallszahlengenerator (für reproduzierbare Resultate) festlegt werden.
Die eigentliche Problemeingabe für den Algorithmus ist ein .cnf-File, welche die zu lösende SAT-Formel im DIMACS-cnf-Format [9] enthält. Auch die Ausgabe erfolgt in ein File im cnf-Format.
Für Circular-3-XOR-SAT wurde ein von den übrigen Programmen und Tools losgelöstes C-Programm geschrieben, welches das auf dieses Problem optimierte Schöningverfahren verwendet.

## Tools

Für verschiedene Zwecke wurden Zusatzprogramme in Java geschrieben; dazu gehören zwei Problemgeneratoren (Random-k-SAT bzw. Circular-3-XOR-SAT), ein Lösungsverifikator sowie ein Interface zum Aufruf des C-Programms aus Java.

## Download

Der Source-Code kann im Internet unter http://n.ethz.ch/student/kaufmanm /download/schule/diplomarbeit/ heruntergeladen werden. Alle weiteren Details dazu stehen im File “readme.txt”.

# Literaturverzeichnis


[1] Stephen A. Cook, *The complexity of theorem proving procedures*, Proc. 3rd Annual ACM Symp. on Theory of Computing, ACM Press, 1971, pp. 151-158.

[2] Uwe Schöning, *A probabilistic algorithm for k-SAT and constraint satisfaction problems*, Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, IEEE Press, 1999, pp. 410-414.

[3] Burkhard Monien, Ewald Speckenmeyer, *Solving Satisfiability in less than $2^n$ steps*, Discrete Applied Mathematics, Vol. 10, 1985, pp. 287-295.

[4] Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon Kleinberg, Christos Papadimitriou, Prabhakar Raghavan, Uwe Schöning, *A deterministic $(2-\frac{2}{k+1})^n$ algorithm for k-SAT based on local search*, Proceedings of the twenty-seventh International Colloquium on Automata, Languages and Programming (ICALP2000), Genf, 2000.

[5] Bart Selman, Hector Levesque, David G. Mitchell, *A new method for solving hard satisfiability problems*, Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92), 1992, pp. 459-465.

[6] Stephen A. Cook, David G. Mitchell, *Finding hard instances of the satisfiability problem: A survey*, in Du, Gu, Pardalos, Satisfiability Problem: Theory and Applications, Dimacs Series in Discrete Mathematics and Theoretical Computer Science Vol. 35, American Mathematical Society, 1997, pp. 1-17.

[7] Jim M. Crawford, Larry D. Auton, *Experimental results on the crossover point in satisfiability problems*, Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI'93), 1993, pp. 21-27.

[8] Thomas Hofmeister, Uwe Schöning, Rainer Schuler, Osamu Watanabe, *A probabilistic 3-SAT algorithm further improved*, Proc. of the 19. Annual Symposium on Theoretical Aspects of Computer Science (STACS), 2002.

[9] *Satisfiability Suggested Format*, 1993, ftp://dimacs.rutgers.edu/pub/challenge/satisfiability/doc/satformat.dvi

[10] Osamu Watanabe, *SAT Instance Generation Page*, http://www.is.titech.ac.jp/~watanabe/gensat/

[11] Robert B. Ash, *Information Theory*, Dover, 1965.

[12] Andrei Z. Broder, Alan M. Frieze, Eli Upfal, *On the satisfiability and maximum satisfiability of random 3-CNF formulas*, Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, 1993, pp. 322-330.