UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fakultät für Elektrotechnik, Informatik und Mathematik
Institut für Informatik

Ein erschöpfende Suche vermeidender Algorithmus für allgemeines k-SAT

Studienarbeit

von
Jaroslaw Klose

vorgelegt bei
Prof. Dr. Johannes Blömer

Betreuer
Prof. Dr. Johannes Blömer

4. Juli 2007

Abstrakt

In dieser Studienarbeit wird ein deterministischer Suchalgorithmus für k-SAT beschrieben. Dieser Algorithmus basiert auf lokaler Suche und wurde 2002 in *A deterministic* $(2-2/(k+1))^n$ *algorithm for* k-SAT *based on local search* [1] vorgestellt. Die hier angegebenen Laufzeiten sind abgesehen von einem polynomiellen Faktor zu verstehen. Die 1999 von Schöning bewiesene [5] obere worst-case Laufzeitschranke $(2-2/k)^n$ ist die beste, derzeit bekannte, für randomisierte k-SAT Algorithmen. Der hier vorgestellte deterministische Algorithmus benötigt Zeit $(2-2/(k+1))^n$ und basiert auf Schönings Algorithmus. Im Wesentlichen benutzt der Algorithmus anstatt der randomisierten Wahl einer initialen Belegung, wie bei Schönings Algorithmus, Überdeckungscodes.
Darüber hinaus wird in [1] eine verbesserte lokale Suche für 3-SAT vorgestellt. Damit lässt sich die obere worst-case Laufzeitschranke 1.481^n für 3-SAT erreichen. In [1] wird darauf hingewiesen, dass diese verbesserte lokale Suche auch für allgemeines $k \geq 3$ möglich ist. In dieser Studienarbeit erfolgt die Verallgemeinerung für beliebiges $k \geq 3$. Es wird außerdem gezeigt, dass mit der Verbesserung eine Beschleunigung für alle $k \geq 3$ erreicht werden kann.

Erklärung

Ich versichere, dass ich die beiliegende Studienarbeit ohne Hilfe Dritter und ohne anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Bielefeld, den 4. Juli 2007

Jaroslaw Klose

Danksagung

Diese Studienarbeit ist im Rahmen meines Informatikstudiums an der Universität Paderborn entstanden. Ich möchte mich hiermit bei Prof. Dr. Johannes Blömer für die gute Betreuung bedanken. Außerdem möchte ich Martin Niemeier, Christian Ikenmeyer und Grischa Ende für das Korrekturlesen und die zahlreichen, sehr hilfreichen Kommentare danken.

Inhaltsverzeichnis

1 Einleitung

SAT (SATISFIABILITY PROBLEM) für boolesche Formeln kann offensichtlich, abgesehen von einem polynomiellen Faktor, in Zeit 2^n gelöst werden, wobei n die Anzahl der Variablen in der gegebenen Formel ist. Hierbei werden alle 2^n möglichen Belegungen ausprobiert.
Da die Laufzeiten in dieser Einleitung immer einen polynomiellen Vorfaktor haben, sind sie immer abgesehen von einem polynomiellen Faktor zu verstehen.
Durch lokale Suche können allerdings bessere, wenn auch weiterhin exponentielle Ergebnisse erreicht werden. In dieser Studienarbeit wird ein Algorithmus vorgestellt, der erstmals in [1] veröffentlicht wurde. Mit diesem deterministischen Algorithmus kann SAT in Zeit $(2-2/(k+1))^n$ gelöst werden. Lokale Suche ist eine bekannte heuristische Suchmethode für SAT. Papadimitriou zeigte 1991 [6], dass 2-SAT durch lokale Suche in polynomieller Zeit gelöst werden kann. Hierzu wird eine randomisierte Prozedur zur lokalen Suche benutzt. 1999 zeigt Schöning [5] die obere worst-case Laufzeitschranke $(2-2/k)^n$ für randomisierte k-SAT Algorithmen. Der hierfür benutzte Algorithmus ähnelt stark dem Algorithmus von Papadimitriou.
Der hier vorgestellte, deterministische Algorithmus basiert auf Schönings Algorithmus. Im Wesentlichen benutzt der Algorithmus anstatt der randomisierten Wahl einer initialen Belegung, wie bei Schönings Algorithmus, Überdeckungscodes.
In [1] wird zusätzlich eine verbesserte lokale Suche für 3-SAT vorgestellt. Damit lässt sich die obere worst-case Laufzeitschranke 1.481^n für 3-SAT erreichen, indem eine modifizierte Prozedur zur lokalen Suche benutzt wird. In [1] wird darauf hingewiesen, dass diese verbesserte lokale Suche auch für allgemeines $k \geq 3$ möglich ist. In Abschnitt 5 erfolgt die Verallgemeinerung für beliebiges $k \geq 3$. Es wird außerdem gezeigt, dass mit der Verbesserung eine Beschleunigung für alle $k \geq 3$ erreicht werden kann.
Im nächsten Unterabschnitt führen wir zunächst die meisten, in dieser Arbeit benötigten, Definitionen ein, bevor wir das grundlegende Vorgehen des Algorithmus von Schöning erläutern und anschließend auf die nötigen Anpassungen eingehen, um eine deterministische Version zu erhalten.

1.1 Notation und Begriffe

An dieser Stelle werden für die Studienarbeit wichtige Notationen und Begriffe eingeführt.
Als erstes definieren wir Formeln in k-KNF und Belegungen.

1.1 Definition (Literal)**.** Ein *Literal* l ist eine boolesche Variable oder deren Negation. Die Negation eines Literals wird mit $\bar{l}$ bezeichnet.

Bei dieser Definition ist eins zu beachten: $\bar{l}$ besagt nicht etwa, dass wir ein negatives Literal meinen, sondern es bezeichnet die Negation eines Literals. Die Negation kann somit positiv oder negativ sein. Die Negation einer Variablen v wird hingegen mit $\neg v$ bezeichnet.

1.2 Definition (Klausel)**.** Eine *Klausel* ist eine Disjunktion von Literalen. Dabei ist die Länge einer Klausel die Anzahl der verschiedenen Literale in der Klausel.

Falls ein Literal mehrfach in einer Klausel vorkommt, wird die Aussage der Formel durch das Entfernen der Mehrfachvorkommen dieses Literals nicht verändert.

1.3 Definition (k-KNF)**.** Sei F eine boolesche Formel und $k \in \mathbb{N}$. F ist genau dann in k-KNF (Konjunktiver Normalform), wenn die Formel ausschließlich aus Konjunktionen von Klauseln der Länge höchstens k besteht.

Es werden ausschließlich boolesche Formeln in k-KNF betrachtet. Hierbei ist $k \geq 3$ eine Konstante. An allen Stellen dieser Arbeit wird $k \geq 3$ vorausgesetzt, auch wenn es nicht immer nochmals explizit dasteht.
Es treten auch triviale Formeln auf, die wie folgt definiert sind:

1.4 Definition (Triviale Formel)**.** Sei F eine boolesche Formel in k-KNF.
F ist eine *triviale Formel* genau dann, wenn einer der beiden Fälle eintritt:

- F ist die leere Formel, diese ist immer wahr.
- F enthält die leere Klausel, solche Formeln sind immer falsch.

Nun führen wir noch Belegungen und eine Notation dazu ein.

1.5 Definition (Belegung)**.** Eine *Belegung* ist eine Abbildung, die jede boolesche Variable auf den Wert 0 oder 1 abbildet, hierbei steht 0 für *falsch* und 1 für *wahr*.

Solche Belegungen werden durch binäre Worte identifiziert, da sie als 0,1-Folgen darstellbar sind.
Wir benötigen eine Notation, mit der das Setzen von Literale einer Formel leicht beschrieben werden kann, da dies an vielen Stellen in der Studienarbeit benötigt wird.

1.6 Definition ($F_{|l=1}$)**.** $F_{|l=1}$ bezeichnet die Formel, die aus F entsteht, wenn das Literal l auf den Wert 1 gesetzt wird. Hierbei werden alle Klauseln, die l enthalten, aus F entfernt. Aus allen anderen Klauseln wird das Literal $\bar{l}$ entfernt. Analog bezeichnet $F_{|l=1,m=1}$ die Formel, die aus $F_{|l=1}$ entsteht, wenn das Literal m auf den Wert 1 gesetzt wird.

Obwohl beim Setzen eines Literals auch die Belegung in gewisser Weise verändert wird, kommt Definition 1.6 ohne Veränderung einer Belegung aus. Dies hat folgenden Grund: Beim Bilden von $F_{|l=1}$ werden alle Vorkommen von l und $\bar{l}$ aus der Formel F eliminiert. Daher hat der Wert für l aus der Belegung a keinen Einfluss mehr auf $F_{|l=1}$. Somit ist es nicht nötig, a anzupassen, wenn $F_{|l=1}$ gebildet wird. Zur besseren Lesbarkeit wird trotzdem nicht auf die Anpassung von a verzichtet. Daher benötigen wir auch eine analoge Schreibweise für Belegungen.

1.7 Definition ($a_{|l=1}$)**.** Sei a eine Belegung einer Formel F und sei l ein Literal aus F. $a_{|l=1}$ bezeichnet die Belegung, die aus a entsteht, wenn der Wert von l auf 1 gesetzt wird.

Das heißt wir setzen den Wert des Literals l in der Belegung a und überschreiben damit den ursprünglichen Wert. Ist l ein positives Literal, so wird in $a_{|l=1}$ die zugehörige boolesche Variable von l mit 1 belegt, ansonsten mit 0.
Nun kommen wir zu einigen Definitionen, die wir für die im erwähnten Überdeckungscodes benötigen.

1.8 Definition (Hamming-Raum)**.** Die Menge aller binären Worte der Länge n, $\{0,1\}^n$, ist der *Hamming-Raum*. Er wird mit H_n bezeichnet.

1.9 Definition (Hamming-Abstand)**.** Die Anzahl der Positionen, an denen sich zwei binäre Worte $u, v \in H_n$ unterscheiden, ist der *Hamming-Abstand* zwischen u und v. Er wird mit $d(u,v)$ bezeichnet.

1.10 Definition (Kugel mit Radius r um ein Wort a)**.** Die *Kugel mit Radius r um ein Wort a* ist die Menge aller Worte, deren Hamming-Abstand zu a höchstens r ist.

1.11 Definition ($V(n,r)$)**.** $V(n,r)$ ist die Anzahl der Worte aus H_n, die in einer Kugel vom Radius r um ein festes Wort a gelegen sind.

Somit bezeichnet $V(n,r)$ das Volumen der Kugel vom Radius r um ein Wort a. Das Volumen ist nicht von a abhängig, sondern nur von r und n, wie wir später sehen werden.
Nun können wir Überdeckungscodes einführen.

1.12 Definition (Code der Länge n)**.** Ein *Code der Länge n* ist eine nicht leere Teilmenge von H_n. Er wird mit $\mathcal{C}$ bezeichnet. Das heißt:

$$\mathcal{C} \subseteq H_n$$

1.13 Definition (Überdeckungsradius eines Codes)**.** Der *Überdeckungsradius r eines Codes* $\mathcal{C}$ ist definiert durch:

$$r = \max_{u \in H_n} \min_{v \in \mathcal{C}} d(u,v)$$

Der Überdeckungsradius eines Codes $\mathcal{C}$ ist somit der maximale Hamming-Abstand eines Wortes $u \in H_n$ zu seinem nächstgelegenen Wort $v \in \mathcal{C}$.

1.14 Definition (Normalisierter Überdeckungsradius eines Codes)**.** Der *normalisierte Überdeckungsradius* ρ *eines Codes* ist definiert durch $\rho = r/n$, wobei r der Überdeckungsradius von $\mathcal{C}$ ist.

1.15 Definition (Überdeckungscode vom Radius r)**.** Sei $\mathcal{C}$ ein Code. $\mathcal{C}$ ist ein *Überdeckungscode vom Radius r* genau dann, wenn der Überdeckungsradius von $\mathcal{C}$ kleiner oder gleich r ist.

Das heißt wir haben einen Überdeckungscode vom Radius r, wenn jedes Wort aus H_n zu mindestens einer Kugel mit dem Radius höchstens r um ein Codewort aus dem Überdeckungscode gehört. Mit anderen Worten: H_n wird vollständig mit Kugeln vom Radius höchstens r überdeckt, wobei diese Kugeln durch Worte aus $\mathcal{C}$ erzeugt werden.
Die in dieser Arbeit betrachteten Laufzeitschranken für Algorithmen sind überwiegend bezüglich eines polynomiellen Vorfaktors formuliert, hierfür dient uns die $\text{poly}_{k,d}(n)$ Schreibweise. Betrachten wir beispielsweise das Polynom $p_{k,d}(n) = 3k \cdot n^{2k} - 3d \cdot n - 7k^d$ mit den Konstanten k und d in der Variablen n. Dann sagen wir, dass $p_{k,d}(n) = \text{poly}_{k,d}(n)$ gilt. Das heißt $p_{k,d}(n)$ ist ein Polynom in der Variablen n, welches zusätzlich von den Konstanten k und d abhängt. Somit handelt es sich bei $\text{poly}_{k,d}(n)$ um eine Klasse von Polynomen, in der in diesem Fall $p_{k,d}(n)$ liegt. Analog zur O-Notation wird allerdings trotzdem ein Gleichheitszeichen verwendet. Die Bezeichnung „die Laufzeit ist $\text{poly}_{k,d}(n)$“ bedeutet ebenfalls analog, dass ein Polynom $q_{k,d}(n)$ existiert, so dass die Laufzeit durch dieses Polynom beschränkt ist.

1.2 Schönings Algorithmus

In diesem Unterabschnitt wollen wir kurz die Vorgehensweise von Schönings Algorithmus erläutern, um zu sehen, an welcher Stelle Anpassungen nötig sind, um einen deterministischen Algorithmus zu erhalten.
Schönings Algorithmus besteht aus der Kernprozedur VERSUCHE. Sie ist die Prozedur für die lokale Suche. VERSUCHE erhält als Eingabe eine Formel F in k-KNF, und gibt *wahr* zurück, wenn eine erfüllende Belegung gefunden wurde, ansonsten *falsch*. Wir schauen uns den Pseudocode an, um das Vorgehen zu verstehen:

Prozedur VERSUCHE(F).

1. Wähle zufällig, gleichverteilt eine initiale Belegung $a \in H_n$
2. Wiederhole $3n$ mal:
   a) Gib *wahr* zurück, wenn F von a erfüllt wird.
   b) Wähle eine beliebige Klausel C, die nicht von a erfüllt wird.
   c) Wähle zufällig, gleichverteilt ein Literal $l \in C$.
   d) Setze $a := a_{l=1}$
3. Gib falsch zurück.

Wir werden an dieser Stelle nicht die Korrektheit der Prozedur zeigen, das kann in [5] nachgelesen werden. Wir sind hingegen interessiert an den randomisierten Komponenten dieser lokalen Suche und wie sie in den Gesamtalgorithmus eingebunden sind. Dieser startet $30 \cdot \left(\frac{2(k-1)}{k}\right)^n$ mal die Prozedur VERSUCHE(F). Wenn einer der Aufrufe *wahr* liefert, ist die Formel erfüllbar,

ansonsten wurde keine erfüllende Belegung gefunden. Somit wählt Schönings Algorithmus maximal exponentiell viele initiale Belegungen. Auf diesen Belegungen wird eine randomisierte Prozedur zur lokalen Suche ausgeführt. Daher ergeben sich im Wesentlichen zwei randomisierte Komponenten in Schönings Algorithmus:

1. Die Auswahl zufälliger, initialer Belegungen.
2. Lokale Suche, gestartet bei diesen Belegungen.

1.3 Anpassungen an Schönings Algorithmus

Der hier vorgestellte deterministische k-SAT Algorithmus ist als derandomisierte Version des kurz umrissenen Algorithmus von Schöning zu verstehen. Wie oben erwähnt, enthält Schönings Algorithmus im Wesentlichen zwei randomisierte Komponenten. Diese Komponenten müssen jeweils derandomisiert werden. Bei der Wahl von initialen Belegungen wird der Raum aller 2^n möglichen Belegungen bei n Variablen mit Kugeln vom Radius r überdeckt. Die Mittelpunkte dieser Kugeln ersetzen dabei die randomisiert gewählten, initialen Belegungen aus Schönings Algorithmus.
Es werden in Abschnitt 3 zwei Algorithmen zur Erzeugung einer guten Überdeckung zu einem gegebenen Radius r vorgestellt. Der erste Algorithmus erreicht eine bis auf einen polynomiellen Faktor minimale Kardinalität der Überdeckung, allerdings benötigt er dafür exponentiell viel Platz. Der zweite Algorithmus konstruiert eine Überdeckung die exponentiell etwas größer ist, dafür benötigt der Algorithmus aber auch nur polynomiell viel Platz.
In jeder Kugel der Überdeckung wird eine deterministische Version der lokalen Suche ausgeführt. Hiermit kann überprüft werden, ob sich in der jeweiligen Kugel eine erfüllende Belegung befindet. Da mit den Kugeln alle möglichen 2^n Belegungen überdeckt werden, wird eine erfüllende Belegung in einer Kugel gefunden, wenn es eine solche gibt.
Um die Gesamtlaufzeit zu minimieren, kann der Wert für r optimal gewählt werden. Wir erhalten die Laufzeit $\text{poly}_k(n) \cdot (2-2/(k+1))^n$, falls $r = n/(k+1)$ gewählt wird. Anschließend wird in Abschnitt 5 gezeigt, wie diese Schranke noch weiter verringert werden kann. Hierzu wird eine modifizierte Version der lokalen Suche benutzt. Außerdem wird gezeigt, dass eine Verbesserung für alle $k \geq 3$ möglich ist. Durch diese Modifikation an der lokalen Suche kann für $k = 3$ die Laufzeitschranke $\text{poly}(n) \cdot 1.481^n$, anstatt $\text{poly}(n) \cdot 1.5^n$ erreicht werden.

1.4 Organisation

Nun eine kleine Übersicht zur Organisation der Studienarbeit:
In Abschnitt 2 wird die deterministische Prozedur zur lokalen Suche vorgestellt. Die Konstruktion der Überdeckungscodes, die im Algorithmus verwendet werden, wird in Abschnitt 3 gezeigt. Abschnitt 4 enthält den eigentlichen Algorithmus, der in dieser Arbeit als Hauptalgorithmus bezeichnet wird, und dessen Analyse. Die Technik zum Modifizieren der lokalen Suche wird in Abschnitt 5 vorgestellt.

2 Lokale Suche

In diesem Abschnitt wird die Prozedur zur lokalen Suche aus unserem Algorithmus vorstellt. Dazu betrachten wir erst die Prozedur und deren Analyse. Anschließend können wir bereits einen ersten einfachen k-SAT Algorithmus angeben. Diese Prozedur und auch der einfache k-SAT Algorithmus wurden in [1] vorgestellt.

2.1 Prozedur SUCHE(F,a,r)

Wir werden nun die Prozedur zur lokalen Suche definieren. Anschließend zeigen wir die Korrektheit und Laufzeit dieser Prozedur.
Wir nehmen zunächst an, dass wir eine initiale Belegung $a \in H_n$ haben. Wie wir eine solche Belegung erhalten, wird in Abschnitt 3 diskutiert.
Als Parameter erhält die Prozedur eine Formel F, eine Belegung a und einen Radius r. Sie gibt *wahr* zurück, wenn F eine erfüllende Belegung innerhalb der Kugel vom Radius r um a hat. Andernfalls gibt die Prozedur *falsch* zurück.
Die Prozedur SUCHE(F,a,r) führt rekursiv eine Tiefensuche durch. Sie dient als Komponente für die lokale Suche des Algorithmus.
In der Prozedur wird an Klauseln *verzweigt*. Das bedeutet in diesem Zusammenhang, dass die Prozedur rekursiv für alle Literale aus einer falschen Klausel aufgerufen wird. Dabei wird das entsprechende Literal auf 1 gesetzt. Das heißt es wird SUCHE$(F_{|l_i=1}, a_{|l_i=1}, r-1)$ für alle l_i aus der falschen Klausel C ausgeführt. Wenn mindestens einer dieser Aufrufe *wahr* zurück gibt, wird *wahr* zurückgegeben, ansonsten *falsch*.
Nun zur Prozedur:

Prozedur SUCHE(F,a,r).

1. Wenn alle Klauseln von F wahr sind unter a, gib *wahr* zurück.
2. Wenn $r \leq 0$ ist, gib *falsch* zurück.
3. Falls F die leere Klausel enthält, gib *falsch* zurück.
4. Wähle (anhand einer deterministischen Regel) eine falsche Klausel C, an der verzweigt wird.

Die Effizienz dieser Prozedur beruht auf einer einfachen Beobachtung: Angenommen wir wollen überprüfen, ob eine erfüllende Belegung in einer Kugel vom Radius r um a existiert. Dann ist es nicht nötig alle $V(n,r)$ Belegungen innerhalb der Kugel zu betrachten. Es müssen nur Belegungen betrachtet werden, die falsche Klauseln wahr machen. Rekursive Aufrufe für wahre Klauseln sind nicht notwendig. Dies ist der Grund dafür, dass nur an falschen Klauseln verzweigt wird. Das Lemma 2.1 fasst diese Beobachtung zusammen.

2.1 Lemma. *Sei F eine Formel und a eine Belegung, so dass F falsch ist unter a. Sei C eine beliebige Klausel aus F, die falsch ist unter a und sei $r \geq 1$. Dann sind folgende Aussagen äquivalent:*

1. *F hat eine erfüllende Belegung in der Kugel vom Radius r um a.*
2. *Es existiert ein erfüllende Belegung b, so dass ein Literal l aus C in b den Wert 1 hat und der Hamming-Abstand zwischen b und $a_{|l=1}$ höchstens $r-1$ ist.*
3. *Es existiert ein Literal l in C, so dass $F_{|l=1}$ eine erfüllende Belegung innerhalb der Kugel mit dem Radius $r-1$ um a hat.*

Beweis. Zunächst zeigen wir die Äquivalenz von 1 und 2.
Wir nehmen an, dass F eine erfüllende Belegung b in der Kugel vom Radius r um a hat. Daher muss diese Belegung b auch die falsche Klausel C erfüllen. Dazu muss es ein Literal l in C geben, welches in b den Wert 1 hat. Da b in der Kugel vom Radius r um a liegt, ist der Hamming-Abstand zwischen b und a höchstens r. $a_{|l=1}$ ist die Belegung, in der l ebenfalls den Wert 1 hat, somit ist der Abstand zwischen b und $a_{|l=1}$ um 1 geringer als zwischen b und a, insgesamt also höchstens $r-1$.
Nun nehmen wir an, dass eine erfüllende Belegung b existiert, so dass ein Literal l aus C in b den Wert 1 hat und der Hamming-Abstand zwischen b und $a_{|l=1}$ höchstens $r-1$ ist. Somit ist der Hamming-Abstand von b und a höchstens r und F hat eine erfüllende Belegung in der Kugel vom Radius r um a, nämlich b.
Jetzt zeigen wir die Äquivalenz von 2 und 3.
Wir nehmen an, dass F unter $a_{|l=1}$ eine erfüllende Belegung innerhalb des Hamming-Abstands $r-1$ hat. Somit hat auch $F_{|l=1}$ unter a eine erfüllende Belegung innerhalb des Hamming-Abstands $r-1$, da ebenfalls l auf den Wert 1 gesetzt wird.
Nun habe $F_{|l=1}$ unter a eine erfüllende Belegung innerhalb des Hamming-Abstands $r-1$ von a. Somit hat auch F unter $a_{|l=1}$ eine Erfüllende Belegung innerhalb des Hamming-Abstands $r-1$, da der Hamming-Abstand zwischen $F_{|l=1}$ unter a und F unter $a_{|l=1}$ 0 ist. □

Nach Lemma 2.1 sind rekursive Aufrufe für bereits erfüllte Klauseln nicht nötig. Es genügt, Literale aus falschen Klauseln zu verändern. Das Lemma liefert allerdings nur eine Existenzaussage. Welches Literal für eine erfüllende Belegung verändert werden muss ist unbekannt. Somit müssen in der Prozedur beim Verzweigen rekursive Aufrufe für jedes Literal aus einer falschen Klausel erfolgen. Man beachte auch, dass es nicht von Bedeutung ist, welche Klausel man wählt. Mit Lemma 2.1 können wir nun leicht die Korrektheit der Prozedur per Induktion zeigen.

2.2 Lemma. *Die Prozedur* SUCHE(F,a,r) *gibt wahr zurück genau dann, wenn F eine erfüllende Belegung in der Kugel um a mit dem Radius r hat.*

Beweis. SUCHE(F,a,r) liefert nur *wahr*, wenn eine erfüllende Belegung innerhalb der Kugel um a vom Radius r gefunden wurde, ansonsten wird immer *falsch* zurückgegeben. Daher reicht

es zu zeigen: SUCHE(F,a,r) findet eine erfüllende Belegung in der Kugel mit dem Radius r um a, wenn F eine erfüllende Belegung in dieser Kugel hat, um die Äquivalenz des Lemmas zu erhalten. Wir zeigen dies per Induktion nach r.
Zunächst betrachten wir für den Induktionsanfang den Fall $r = 0$. Dann erfüllt die Belegung a bereits die Formel F, da die Kugel um a vom Radius 0 nur a selbst enthält. SUCHE(F,a,r) gibt also *wahr* zurück.
Nun zum Induktionsschritt von $r-1$ nach r. Wenn SUCHE(F,a,r) aufgerufen wird, verzweigt die Prozedur an einer falschen Klausel. Wenn eine erfüllende Belegung um a vom Radius r existiert, dann existiert nach Lemma 2.1 auch eine in der Kugel um $a_{|l=1}$ vom Radius $r-1$ für ein Literal l aus der gewählten Klausel. Es erfolgen rekursive Aufrufe für alle Literale aus der Klausel, insbesondere also auch für das Literal l. Da in dieser Kugel eine erfüllende Belegung existiert, wird sie von SUCHE$(F|l=1,a|l=1,r-1)$ nach Induktionsvoraussetzung gefunden. Somit gilt das Lemma für alle $r \geq 0$. □

Wir untersuchen nun die Laufzeit der Prozedur.

2.3 Lemma. SUCHE(F,a,r) *benötigt Zeit* $\text{poly}_k(n) \cdot k^r$.

Beweis. Die Rekursionstiefe von SUCHE(F,a,r) ist höchstens r. Wenn die Eingabeformel F in k-KNF ist, ist die Anzahl der rekursiven Aufrufe in Zeile 4 der Prozedur durch k beschränkt. Daher hat der Rekursionsbaum höchstens k^r Blätter. Die Anzahl der übrigen Knoten ergibt sich aus:

$$\sum_{i=0}^{r-1} k^i$$

Das es sich um eine Partialsumme der geometrischen Reihe handelt, erhalten wir:

$$\sum_{i=0}^{r-1} k^i = \frac{k^{(r-1)+1}-1}{k-1} = \frac{k^r-1}{k-1} < \frac{1}{k-1} \cdot k^r.$$

Daher ist die Anzahl aller Knoten im Baum polynomiell abhängig von der Anzahl der Blätter. Pro Knoten beziehungsweise Blatt wird nur polynomieller Aufwand in Abhängigkeit von k benötigt. Insgesamt erhalten wir für die Laufzeit:

$$\text{poly}_k(n) \cdot \left(k^r + \frac{1}{k-1} \cdot k^r \right) = \text{poly}_k(n) \cdot \left(1 + \frac{1}{k-1} \right) \cdot k^r = \text{poly}_k(n) \cdot k^r.$$

Somit gilt die Behauptung. □

Obwohl wir den exponentiellen Faktor k^r für die Anzahl an Aufrufen erhalten, kann dieser viel geringer sein, als das Volumen der Kugel vom Radius r um a. Dies werden wir an einem Beispiel für $k = 3$ im nächsten Unterabschnitt sehen. Daher müssen nicht alle Belegungen in der Kugel vom Radius r um a betrachtet werden, um eine Aussage über das Vorhandensein von erfüllenden Belegungen in der Kugel zu erhalten.

2.2 Ein einfacher Algorithmus für 3-SAT

In diesem Unterabschnitt leiten wir einen einfachen Algorithmus für 3-SAT her.
Für $r = n/2$ gilt: $V(n, n/2) \geq 2^{n-1}$, da höchstens die Hälfte aller Worte überdeckt wird. Für $k = 3$ gilt für die Laufzeit $\text{poly}_k(n) \cdot k^r$ nach 2.3:
$\text{poly}(n) \cdot 3^r = \text{poly}(n) \cdot 3^{n/2} = \text{poly}(n) \cdot \left(\sqrt{3}\right)^n = \text{poly}(n) \cdot (1.7320\ldots)^n < \text{poly}(n) \cdot 1.733^n$.
Diese Beobachtung liefert uns bereits einen ersten, sehr einfachen, deterministischen 3-SAT Algorithmus:

Einfacher 3-SAT Algorithmus

1. Setze $a_0 := (\underbrace{0,0,\ldots,0}_{n \text{ mal}})$ und $a_1 := (\underbrace{1,1,\ldots,1}_{n \text{ mal}})$.
2. Starte $\text{SUCHE}(F, a_0, n/2)$ und $\text{SUCHE}(F, a_1, n/2)$. Falls einer der Aufrufe *wahr* liefert, gib *wahr* zurück, ansonsten *falsch*.

Da jedes Wort aus H_n einen Hamming-Abstand $\leq n/2$ zu a_0 oder a_1 hat, bilden a_0 und a_1 einen Überdeckungscode, dessen Radius durch $n/2$ beschränkt ist. $\{a_0, a_1\}$ ist ein Überdeckungscode vom Radius $n/2$, somit ist der Algorithmus nach Lemma 2.2 korrekt, da $\text{SUCHE}(F, a, r)$ korrekt ist. Die Laufzeit ist durch $\text{poly}(n) \cdot 1.733^n$ beschränkt.
Durch eine so einfache Wahl eines Überdeckungscodes kann bereits ein gutes Ergebnis für $k = 3$ erreicht werden. Für größere k liefert dieser Algorithmus keine Beschleunigung.
Wir werden in Abschnitt 4 sehen, dass vor allem eine geschickte Wahl des Radius r zu einer Beschleunigung führt. Dazu müssen wir allerdings in der Lage sein, zu einem beliebigen r und n, einen Überdeckungscode zu konstruieren. Mit diesem Problem setzt sich der nächste Abschnitt auseinander.

3 Konstruktion von Überdeckungscodes

In diesem Abschnitt beschäftigen wir uns mit der Konstruktion von Überdeckungscode zu gegebenem n und r. Diese Konstruktion wurde in [1] vorgestellt, zum Teil mit Ergebnissen aus anderer Literatur, die in dieser Arbeit auch als solche gekennzeichnet sind.
Zunächst schätzen wir $V(n,r)$ ab, um leichter argumentieren zu können. Anschließend zeigen wir die Existenz von Überdeckungscodes zu gegebenem n und r, und stellen zwei Algorithmen zum Konstruieren solcher Codes vor.

3.1 Abschätzung für das Volumen einer Kugel vom Radius r

In diesem Unterabschnitt werden wir uns überlegen, wie viele Belegungen sich in einer Kugel vom Radius r befinden.

3.1 Lemma. *Für das Volumen einer Kugel vom Radius r $V(n,r)$ gilt:*

$$V(n,r) = \sum_{i=0}^{r} \binom{n}{i}.$$

Beweis. Der Radius der Kugel ist r, somit kann sich der Wahrheitswert eines Wortes in der Kugel an höchstens r Stellen verändern. Man kann sich das wie Ziehen ohne Zurücklegen von bis zu r Elementen aus einer Menge von n Elementen vorstellen. Die Anzahl der Möglichkeiten zum Ziehen von genau i Elementen ist $\binom{n}{i}$. Insgesamt gibt es also $\sum_{i=0}^{r} \binom{n}{i}$ Möglichkeiten. Somit gilt das Lemma. □

Wie wir in dem Lemma gesehen haben, ist das Volumen der Kugel unabhängig vom konkreten Wort, um das die Kugel gebildet wird. Es hängt lediglich von der Anzahl an Position ab, an denen sich die Worte in der Kugel von einem bestimmten Wort unterscheiden dürfen. Somit haben alle Kugeln vom gleichen Radius in H_n auch das gleiche Volumen.
Die obige Formel für das Volumen lässt sich bei unseren Abschätzungen nur schwer handhaben. Daher werden wir eine für unsere Zwecke einfachere Abschätzung benutzen. Diese liefert uns einen Wert für $V(n,r)$, der um höchstens einen polynomiellen Faktor vom tatsächlichen Wert abweicht. Bevor wir zur eigentlichen Abschätzung des Volumens kommen, wird noch eine Definition benötigt:

3.2 Definition (Binäre Entropiefunktion $h(\rho)$)**.** Sei $0 < \rho < 1$, dann ist die *binäre Entropiefunktion* definiert durch:

$$h(\rho) = -\rho \log_2(\rho) - (1-\rho)\log_2(1-\rho)$$

Nun können wir das Lemma zur Abschätzung formulieren.

3.3 Lemma. *Sei* $0 < \rho \leq 1/2$ *und* $h(\rho)$ *die binäre Entropiefunktion. Dann gilt:*

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

Beweis. Beweis siehe [2] Korollar 9, Seite 310. □

Diese Abschätzung zeigt, dass sich $V(n,r)$ und $2^{h(\rho)n}$, für ein konstantes ρ mit $0 < \rho \leq 1/2$, um höchstens einen polynomiellen Faktor unterscheiden.

3.2 Existenz eines Überdeckungscodes zu gegebenem Radius

In diesem Abschnitt zeigen wir die Existenz eines Überdeckungscodes zu einem gegebenem Radius $r = \rho n$, mit einer Größe von $poly(n) \cdot 2^{(1-h(\rho))n}$.
Als Erstes werden wir eine untere Schranke für die Größe von Überdeckungscodes herleiten. Dazu zunächst ein vorbereitendes Lemma.

3.4 Lemma. *Für jeden Überdeckungscode* $\mathcal{C}$ *mit Überdeckungsradius r gilt:*

$$|\mathcal{C}| \geq \frac{2^n}{V(n,r)}$$

Beweis. Da jedes Wort aus H_n zu einer Kugel vom Radius r um ein Wort aus $\mathcal{C}$ gehört, müssen in den erzeugten Kugeln mindestens so viele Worte wie in H_n enthalten sein. Dies gilt, da $\mathcal{C}$ ein Überdeckungscode ist. Es gibt insgesamt $|\mathcal{C}|$ Kugeln mit je $V(n,r)$ Worten. H_n enthält 2^n Worte. Insgesamt erhalten wir:

$$|\mathcal{C}| \cdot V(n,r) \geq 2^n.$$

Umformen liefert die Behauptung. □

Mit diesem Lemma und der oberen Schranke aus Lemma 3.3, erhält man leicht die folgende Abschätzung:

3.5 Lemma. *Sei* $\mathcal{C}$ *ein Überdeckungscode mit normalisiertem Überdeckungsradius* ρ*, wobei* $0 < \rho < 1/2$*. Dann gilt:*

$$|\mathcal{C}| \geq 2^{(1-h(\rho))n}$$

Beweis.

$$|\mathcal{C}| \underset{3.4}{\geq} \frac{2^n}{V(n,r)} \underset{3.3}{\geq} \frac{2^n}{2^{h(\rho)n}} = 2^{(1-h(\rho))n}$$

□

Diese untere Schranke wird auch als *sphärische Überdeckungsschranke* [3] bezeichnet.
Das folgende Lemma, bekannt aus der Codierungstheorie ([3] Satz 12.1.2, S. 320), zeigt die Existenz von Überdeckungscodes, deren Größe die Abschätzung aus Lemma 3.4 bis auf einen Faktor n erreicht.

3.6 Lemma. *Für jedes $n \geq 1$ und $r \geq 0$ existiert ein Überdeckungscode $\mathcal{C}$ der Länge n, Überdeckungsradius höchstens r und Größe höchstens*

$$\left\lceil \frac{n \cdot 2^n}{V(n,r)} \right\rceil .$$

Beweis. Die Existenz von $\mathcal{C}$ wird durch ein probabilistisches Argument gezeigt.
Zunächst wählen wir $n \cdot 2^n / V(n,r)$ Worte aus H_n zufällig, gleichverteilt, unabhängig mit Zurücklegen. Nun zeigen wir, dass diese Worte mit einer echt positiven Wahrscheinlichkeit einen Überdeckungscode vom Radius höchstens r ergeben.
Sei a ein festes Wort aus H_n. Mit der Wahrscheinlichkeit $V(n,r)/2^n$ gehört es zu einer Kugel vom Radius r um ein zufällig, gleichverteilt, unabhängig gewähltes Wort b aus H_n. Somit ist die Wahrscheinlichkeit, dass es nicht zu dieser Kugel gehört $1 - V(n,r)/2^n$. Wir haben $n \cdot 2^n / V(n,r)$ solche Worte zufällig, gleichverteilt und unabhängig gewählt. Die Wahrscheinlichkeit, dass a zu keiner der Kugeln vom Radius r um die zufällig gewählten Worte gehört, ergibt sich daher als Produkt der Wahrscheinlichkeiten $1 - V(n,r)/2^n$, da die Ereignisse unabhängig sind. Insgesamt erhalten wir:

$$\begin{aligned}
& (1 - V(n,r)/2^n)^{n \cdot 2^n / V(n,r)} \\
= \quad & \left((1 - V(n,r)/2^n)^{2^n / V(n,r)} \right)^n \\
= \quad & ((1 - 1/x)^x)^n \,, \text{für } x = \frac{2^n}{V(n,r)} \\
\leq \quad & \left(e^{-\frac{1}{x} \cdot x} \right)^n \,, \text{ da } (1+x) \leq e^x \\
\leq \quad & e^{-n}
\end{aligned}$$

Somit ist die Wahrscheinlichkeit, dass ein beliebiges Wort nicht überdeckt wird höchstens e^{-n}. Insgesamt haben wir 2^n Worte. Die Wahrscheinlichkeit, dass mindestens eines dieser Worte nicht überdeckt wird, ergibt sich als Summe der Wahrscheinlichkeit für jedes Wort:

$$\sum_{i=1}^{2^n} e^{-n} = 2^n \cdot e^{-n}$$

Somit ist die Wahrscheinlichkeit, dass mindestens ein Wort nicht überdeckt wird, höchstens $2^n \cdot e^{-n}$. Daher bilden die gewählten Worte einen Überdeckungscode vom Radius höchstens r mit Wahrscheinlichkeit mindestens $1 - 2^n \cdot e^{-n}$. Dies ist jedoch echt größer als 0 für alle $n \geq 1$. Somit existiert ein Überdeckungscode zu beliebigem $n \geq 1$ und $r \geq 0$. □

Wir können jetzt leicht ein Korollar mit Hilfe von Lemma 3.3 herleiten.

3.7 Korollar. *Sei* $0 < \rho < 1/2$ *und sei* $\beta(n) = \sqrt{8n\rho(1-\rho)}$. *Für jedes n existiert ein Überdeckungscode* $\mathcal{C}$ *der Länge n, Radius höchstens* $\rho n = r$ *und Größe höchstens* $n\beta(n) \cdot 2^{(1-h(\rho))n}$.

Beweis. Aus Lemma 3.6 und der Schranke aus Lemma 3.3 folgt:

$$\begin{aligned} & \frac{n \cdot 2^n}{V(n,r)} \\ \underset{3.3}{\leq} & \frac{n \cdot 2^n}{\frac{1}{\beta(n)} \cdot 2^{h(\rho)n}} \\ = & n2^n\beta(n) \cdot 2^{-h(\rho)n} \\ = & n\beta(n) \cdot 2^{(1-h(\rho))n} \end{aligned}$$

□

Das Korollar liefert die Existenz eine Überdeckungscodes von fast minimaler Größe und einen einfachen, randomisierten Algorithmus zum Konstruieren eines solchen Codes. Der Algorithmus wählt zufällig, gleichverteilt die $\left\lceil \frac{n \cdot 2^n}{V(n,r)} \right\rceil$ Worte des Überdeckungscodes.

Nun werden zwei deterministische Algorithmen beschrieben. Der erste generiert einen Überdeckungscode, der die sphärische Überdeckungsschranke bis auf einen polynomiellen Faktor erreicht. Allerdings benötigt der Algorithmus exponentiellen Platz. Der zweite Algorithmus konstruiert in polynomiellem Platz einen Überdeckungscode, der „beinahe“ die sphärische Überdeckungsschranke erreicht. Hierbei heißt „beinahe“ bis auf einen Faktor von $2^{\varepsilon n}$, für ein beliebiges $\varepsilon > 0$. Obwohl der Algorithmus leicht aus früheren Ergebnissen in der Literatur abgeleitet werden kann, scheint die Benutzung des Algorithmus zur Erzeugung von Überdeckungscodes neu zu sein [1].

3.3 Erster Algorithmus

Wir können die Konstruktion eines Überdeckungscodes als einen Fall des SET COVER Problems betrachten: H_n soll mit möglichst wenigen Kugeln vom Radius ρn überdeckt werden. Nun schauen wir uns einen Greedy-Algorithmus [4] (Kapitel 3.2 Seite 99) an, der die benötigten Kugeln bis auf den Faktor $n+1$ approximiert [4]. Zunächst führen wir eine vereinfachende Notation ein: Es sei $\mathcal{C}$ ein Code und $a \in H_n$, dann bezeichnet $u(a, \mathcal{C}, r)$ die Anzahl an Worten in der Kugel vom Radius r um a, die von $\mathcal{C}$ noch nicht überdeckt werden. Der Algorithmus ist wie folgt definiert:

Prozedur ÜBERDECKE(n, ρ).

1. $\mathcal{C} = \emptyset$
2. Solange $\mathcal{C}$ kein Überdeckungscode vom Radius ρn ist, tue:

a) Wähle ein $a \in H_n$, so dass $u(a, \mathcal{C}, \rho n) = \max_{b \in H_n} u(b, \mathcal{C}, \rho n)$.

b) Setze $\mathcal{C} = \mathcal{C} \cup \{a\}$.

3. Gib $\mathcal{C}$ aus.

Somit wählt der Algorithmus in jedem Schritt eine Kugel um ein Wort a. Diese Kugel überdeckt möglichst viele bisher noch nicht überdeckte Worte.
Grobe Abschätzung der Laufzeit: Mit jeder Kugel assoziieren wir die Anzahl der bis jetzt noch nicht überdeckter Worte, die sie enthält. In einer Iteration wählen wir eine Kugel, für die diese Anzahl maximal ist und aktualisieren die Anzahlen für alle anderen Kugeln. Pro Kugel müssen höchstens 2^n Worte aktualisiert werden, somit ergibt sich für jede Iteration die Laufzeit $\text{poly}(n) \cdot 2^{2n}$. Es gibt höchstens 2^n Iterationen, da jede Kugel mindestens ein Wort enthält. Insgesamt erhält man also die Laufzeit $\text{poly}(n) \cdot 2^{3n}$. Nach [4] Satz 4.1 handelt es sich um einen $(1+n)$-Approximationsalgorithmus. Da die Größe des Überdeckungscodes nach Korollar 3.7 höchstens $n\beta(n) \cdot 2^{(1-h(\rho))n}$ ist, liefert der Algorithmus einen Überdeckungscode mit Größe höchstens $(n+1)n\beta(n) \cdot 2^{(1-h(\rho))n}$. Wir fassen das Ergebnis im folgenden Lemma zusammen:

3.8 Lemma. *Sei $n \geq 1$, $0 < \rho < 1/2$ und $\beta(n) = \sqrt{8n\rho(1-\rho)}$. Dann kann ein Überdeckungscode der Länge n vom Radius höchstens ρn und Größe höchstens $(n+1)n\beta(n) \cdot 2^{(1-h(\rho))n}$ in Zeit $\text{poly}(n) \cdot 2^{3n}$ konstruiert werden.*

Diese grobe Abschätzung liefert unglücklicherweise eine viel zu große Schranke für die Laufzeit. Im folgenden Lemma wird gezeigt, wie eine Verbesserung der Laufzeit erreicht werden kann, indem der zu konstruierende Überdeckungscode verkleinert wird.

3.9 Lemma. *Sei $d \geq 2$ ein Teiler von $n \geq 1$ und $0 < \rho < 1/2$. Dann kann ein Überdeckungscode der Länge n, Radius höchstens ρn und Größe höchstens $\text{poly}_d(n) \cdot 2^{(1-h(\rho))n}$ konstruiert werden. Hierbei wird bezüglich n exponentieller Platz und Zeit $\text{poly}_d(n) \cdot \left(2^{3n/d} + 2^{(1-h(\rho))n}\right)$ benötigt.*

Beweis. Zunächst werden die n Bits in den Worten aus H_n in d Blöcke der Länge n/d partitioniert. Wir benutzen Lemma 3.8, um einen Überdeckungscode $\mathcal{C}'$ vom Radius höchstens $\rho n/d$ für $H_{n/d}$ zu konstruieren. Dann definieren wir $\mathcal{C}$ als direkte Summe von d Worten aus $\mathcal{C}'$. Das heißt die Menge aller Konkatenationen von d Worten aus $\mathcal{C}'$.
Wir müssen noch zeigen, dass $\mathcal{C}$ ein Überdeckungscode vom Radius höchstens ρn ist:
Jedes $a \in H_n$ kann in d Blöcke $a_1, \ldots, a_d$, jeweils mit der Länge n/d, unterteilt werden. Wir identifizieren ein Wort $w_i \in \mathcal{C}'$ innerhalb des Abstands $\rho n/d$ von jedem Block a_i. Die Konkatenation $w_1 w_2 \ldots w_d$ ist dann ein Wort aus $\mathcal{C}$ innerhalb des Abstands ρn von a. Somit gehört jedes Wort aus H_n zu einer Kugel mit dem Radius ρn um ein Codewort. Daher ist $\mathcal{C}$ ein Überdeckungscode. Die Größe von $\mathcal{C}$ ist höchstens $((n+1)n\beta(n) \cdot 2^{(1-h(\rho))n/d})^d = \text{poly}_d(n) \cdot 2^{(1-h(\rho))n}$. Die benötigte Zeit zur Konstruktion von $\mathcal{C}'$ ist $\text{poly}_d(n) \cdot 2^{3n/d}$ nach Lemma 3.8.
Nun zum Platzbedarf: Für die Konstruktion von $\mathcal{C}'$ wird die Anzahl an noch nicht überdeckten Worten für jede mögliche Kugel gespeichert. Es gibt anfänglich $2^{n/d}$ solche Kugeln, für jedes

Wort eine. Da d fest gewählt wird, benötigt der vorgestellt Greedy-Algorithmus exponentiell viel Platz. □

Da der letzte Algorithmus exponentiell viel Platz benötigt, betrachten wir noch einen zweiten Algorithmus, der seine Blockgröße in Abhängigkeit von n wählt. Dieser benötigt dann nur polynomiell viel Platz. Der dabei entstehende Überdeckungscode wird dabei jedoch etwas größer.

3.4 Zweiter Algorithmus

In Lemma 3.9 besteht jedes Wort aus einer konstanten Anzahl von d Blöcken der Länge n/d. In dem folgenden Lemma besteht ein Codewort aus einer linearen Anzahl n/b von Blöcken konstanter Länge b. Da die Blöcke eine konstante Länge haben, benötigen wir nur polynomiell viel Platz bezüglich der Eingabelänge.

3.10 Lemma. *Sei $\delta > 0$ und $0 < \rho < 1/2$. Es gibt eine Konstante $b = b(\delta, \rho)$, so dass für jedes durch b teilbare n, ein Überdeckungscode der Länge n, Radius höchstens ρn und Größe höchstens $2^{(1-h(\rho)+\delta)n}$, in polynomiellem Platz konstruiert werden kann.*

Beweis. Zunächst erhält man mit Korollar 3.7, dass ein Code $\mathcal{C}' = \mathcal{C}'(\delta, \rho)$ mit Länge $b = b(\delta, \rho)$ existiert, mit Radius höchstens ρb und Größe höchstens $b\beta(b) \cdot 2^{(1-h(\rho))b}$. Da b von δ und ρ abhängt, kann b groß genug gewählt werden, sodass $b\beta(b) \cdot 2^{(1-h(\rho))b} \leq 2^{(1-h(\rho)+\delta)b}$ gilt. Das heißt der polynomielle Vorfaktor kann bei entsprechender Wahl von b vernachlässigt werden. Somit existiert ein Code mit der Größe höchstens $2^{(1-h(\rho)+\delta)b}$. Analog zum vorhergehenden Lemma bildet die direkte Summe von n/b Instanzen von $\mathcal{C}'$ den benötigten Code. Bei n/b Instanzen ergibt sich für die Größe insgesamt: $\left(2^{(1-h(\rho)+\delta)b}\right)^{n/b} = 2^{(1-h(\rho)+\delta)n}$. □

4 Hauptalgorithmus und seine Analyse

In diesem Abschnitt definieren wir den eigentlichen Algorithmus. Der Algorithmus und die Analyse wurden in [1] vorgestellt, und werden in diesem Abschnitt ausgearbeitet. In dieser Arbeit ist die Berechnung für die geschickte Wahl der im nächsten Unterabschnitt beschriebenen Parameter d und ρ ergänzt.

4.1 Der Hauptalgorithmus

Der Hauptalgorithmus erhält als Eingabe eine Formel F in k-KNF mit n Variablen, einen normalisierten Überdeckungsradius ρ mit $0 < \rho < 1/2$, und eine Zahl $d \geq 0$. Der Hauptalgorithmus liefert *wahr*, wenn F erfüllbar ist, ansonsten *falsch*.
Zur Vereinfachung nehmen wir an, dass n durch d geteilt werden kann. Es müssen höchstens $d-1$ Variablen hinzugefügt werden, um die Teilbarkeit zu erreichen. Für ein festes d sind es daher konstant viel Variablen, die hinzugefügt werden müssen. Durch das Hinzufügen von konstant vielen Variablen wird die Laufzeit um höchstens einen konstanten Faktor erhöht, somit können gegebenenfalls Variablen eingefügt werden, bis die Länge der entstehenden Formel von d geteilt wird. Wir werden in Unterabschnitt 4.3 sehen, dass $d = 6$ eine gute und konstante Wahl ist. Daher wirkt sich diese Vereinfachung höchstens konstant auf die später durch die Wahl von $d = 6$ erreichte Laufzeit aus, auch wenn an dieser Stelle diese Vereinfachung eine Erhöhung der Laufzeit bewirkt, die vom Parameter d abhängt. Aus diesem Grund werden wir die Laufzeit bei der Analyse für festes d betrachten.
Nun zum eigentlichen Algorithmus:

Hauptalgorithmus LÖSE(F, ρ, d).

1. Benutze die Konstruktion aus dem Beweis von Lemma 3.9 mit dem Parameter d, um einen Überdeckungscode $\mathcal{C}$ mit Länge n und Radius höchstens ρn zu erzeugen.

2. Für jedes Codewort $c \in \mathcal{C}$, führe SUCHE$(F, c, \rho n)$ aus. Gib *wahr* zurück, wenn mindestens eine aufgerufene Prozedur *wahr* zurück liefert. Ansonsten gib *falsch* zurück.

Zunächst zeigen wir die Korrektheit und Laufzeit:

4.1 Lemma. *Der Hauptalgorithmus* LÖSE(F, ρ, d) *löst* k-SAT *für jeden Parameter* d *und* $0 < \rho < 1/2$. *Die Laufzeit von* LÖSE(F, ρ, d) *für festes* d *ist*

$$T_1(n, \rho, d) + B(n, \rho, d) \cdot T_2(n, \rho), \tag{1}$$

wobei:

$$\begin{aligned} T_1(n, \rho, d) &:= \mathrm{poly}_d(n) \cdot \left(2^{3n/d} + 2^{(1-h(\rho))n}\right) \\ B(n, \rho, d) &:= \mathrm{poly}_d(n) \cdot 2^{(1-h(\rho))n} \\ T_2(n, \rho) &:= \mathrm{poly}_k(n) \cdot k^{\rho n} \end{aligned}$$

Beweis. In Lemma 2.2 wurde die Korrektheit von SUCHE(F,a,r) gezeigt. Durch die Nutzung des Überdeckungscodes $\mathcal{C}$ werden alle Worte aus H_n betrachtet. Daher wird *wahr* genau dann zurückgegeben, wenn F erfüllbar ist.
Nun leiten wir noch die Laufzeit für den Hauptalgorithmus her.
Der Hauptalgorithmus erhält als Eingabe eine Formel F in k-KNF mit n Variablen. Der Algorithmus überdeckt H_n mit Kugeln vom Radius ρn mit Hilfe der Konstruktion aus dem Beweis von Lemma 3.9. Die Überdeckung besteht aus $B(n,\rho,d) = \text{poly}_d(n) \cdot 2^{(1-h(\rho))n}$ Kugeln, die Konstruktion dieser Kugeln dauert $T_1(n,\rho,d) = \text{poly}_d(n) \cdot \left(2^{3n/d} + 2^{(1-h(\rho))n}\right)$. In jeder Kugel führen wir die Prozedur zur lokalen Suche durch, dies dauert $T_2(n,\rho) = \text{poly}_k(n) \cdot k^{\rho n}$ pro Kugel. Daher ist die Gesamtlaufzeit

$$T_1(n,\rho,d) + B(n,\rho,d) \cdot T_2(n,\rho).$$

□

Wir werden in den beiden nächsten Teilabschnitten sehen, dass wir durch geschickte Wahl von ρ die Laufzeit minimieren können und dass bei geschickter Wahl von d die Konstruktion der Kugeln bei der Laufzeit vernachlässigt werden kann. Dann können wir die Laufzeit mit dieser Wahl in Unterabschnitt 4.4 berechnen.

4.2 Günstige Wahl des Überdeckungsradius

An dieser Stelle minimieren wir den exponentiellen Anteil des Produktes aus Formel (1), indem wir ρ, den normierten Überdeckungsradius, möglichst geschickt wählen. Hierzu zeigen wir das folgende Lemma:

4.2 Lemma. *Der exponentielle Anteil des Produktes $B(n,\rho,d) \cdot T_2(n,\rho)$ hat ein globales Minimum für $\rho = \frac{1}{k+1}$.*

Beweis. Zunächst bestimmen wir den exponentiellen Anteil, indem wir die Definition für reelle Exponenten

$$a^b := e^{b \cdot \ln(a)} \tag{2}$$

benutzen:

$$\begin{aligned}
& B(n,\rho,d) \cdot T_2(n,\rho) \\
= \;& \text{poly}_d(n) \cdot 2^{(1-h(\rho))n} \cdot \text{poly}_k(n) \cdot k^{\rho n} \\
= \;& \text{poly}_{d,k}(n) \cdot 2^{(1-h(\rho))n} \cdot k^{\rho n} \\
\underset{(2)}{=} \;& \text{poly}_{d,k}(n) \cdot e^{(1-h(\rho))n \cdot \ln(2)} \cdot e^{\rho n \cdot \ln(k)} \\
= \;& \text{poly}_{d,k}(n) \cdot e^{(1-h(\rho))n \cdot \ln(2) + \rho n \cdot \ln(k)}
\end{aligned}$$

Nun wird der exponentielle Anteil minimiert.

$$\begin{aligned} & (1-h(\rho))n\cdot\ln(2)+\rho n\cdot\ln(k) \\ = \;& n\cdot\ln(2)\cdot\left(1-h(\rho)+\rho\cdot\frac{\ln(k)}{\ln(2)}\right) \\ = \;& n\cdot\ln(2)\cdot(1-h(\rho)+\rho\cdot\log_2(k)) \end{aligned}$$

Der Vorfaktor hängt nicht von ρ ab, somit muss nur der Ausdruck in der Klammer minimiert werden. Hierzu bilden wir die Ableitung und suchen ein lokales Minimum für $0<\rho\leq 1/2$. Zunächst setzen wir Definition 3.2 von $h(\rho)$ ein.

$$\begin{aligned} & (1-h(\rho)+\rho\cdot\log_2(k) \\ = \;& (1+\rho\log_2(\rho)+(1-\rho)\log_2(1-\rho)+\rho\cdot\log_2(k) \\ = \;& \frac{1}{\ln(2)}\cdot(\ln(2)+\rho\ln(\rho)+(1-\rho)\ln(1-\rho)+\rho\cdot\ln(k)) \end{aligned}$$

Nun bilden wir die Ableitung des Ausdrucks in der Klammer, der Vorfaktor hängt wie oben nicht von ρ ab.

$$\begin{aligned} & (\ln(2)+\rho\ln(\rho)+(1-\rho)\ln(1-\rho)+\rho\cdot\ln(k))\,\frac{\partial}{\partial\rho} \\ = \;& \ln(\rho)+\frac{\rho}{\rho}-\ln(1-\rho)-\frac{1-\rho}{1-\rho}+\ln(k) \\ = \;& \ln(\rho)-\ln(1-\rho)+\ln(k) \\ = \;& \ln\left(\frac{\rho}{1-\rho}\right)+\ln(k) \end{aligned}$$

Eine notwendige Bedingung für ein lokales Extremum ist eine Nullstelle in der ersten Ableitung. Daher suchen wir eine Nullstelle:

$$\begin{aligned} & \ln\left(\frac{\rho}{1-\rho}\right)+\ln(k)=0 \\ \Leftrightarrow \quad & \ln\left(\frac{\rho}{1-\rho}\right)=\ln\left(\frac{1}{k}\right) \end{aligned}$$

Anwenden der Exponentialfunktion liefert:

$$\begin{aligned} & \frac{\rho}{1-\rho}=\frac{1}{k} \\ \Leftrightarrow \;& k\rho+\rho=1 \\ \Leftrightarrow \;& \rho=\frac{1}{k+1} \end{aligned}$$

Eine hinreichende Bedingung für ein lokales Minimum ist eine Nullstelle in der ersten Ableitung und ein Funktionswert größer 0 in der zweiten Ableitung. Dies muss noch überprüft werden:

$$\begin{aligned} & (\ln(2)+\rho\ln(\rho)+(1-\rho)\ln(1-\rho)+\rho\cdot\ln(k))\,\frac{\partial^2}{\partial\rho^2} \\ = \;& (\ln(\rho)-\ln(1-\rho)+\ln(k))\,\frac{\partial}{\partial\rho} \\ = \;& \frac{1}{\rho}+\frac{1}{1-\rho} \end{aligned}$$

Somit ist die zweite Ableitung für alle $0<\rho$ größer als 0. Daher hat die Funktion keine Wendestellen (der Logarithmus ist nur im Positiven definiert). Somit kann es keine weiteren Extrema geben. Auch die Randwerte der Funktion sind größer als das von uns gefundene lokale Minimum. Somit erhalten wir ein globales Minimum für $\rho=\frac{1}{k+1}$. □

Nun können wir das Produkt aus (1) berechnen, indem wir ρ auf $\frac{1}{k+1}$ setzen.
Zunächst stellen wir $T_2(n,\rho)=\text{poly}_k(n)\cdot k^{\rho n}$ als Zweierpotenz dar.
Mit $\log_2(k)=\frac{\ln(k)}{\ln(2)}$ erhält man: $\ln(k)\underset{(*)}{=}\log_2(k)\cdot\ln(2)$. Einsetzen ergibt:

$$\begin{aligned} & \text{poly}_k(n)\cdot k^{\rho n} \\ \underset{(2)}{=} \;& \text{poly}_k(n)\cdot e^{\rho n\cdot\ln(k)} \\ \underset{(*)}{=} \;& \text{poly}_k(n)\cdot e^{\rho n\cdot\log_2(k)\cdot\ln(2)} \\ \underset{(2)}{=} \;& \text{poly}_k(n)\cdot 2^{\rho n\cdot\log_2(k)} \end{aligned}$$

Nun kann das Produkt berechnet werden:

$$\begin{aligned} & B(n,\rho,d)\cdot T_2(n,\rho) \\ = \;& \text{poly}_d(n)\cdot 2^{(1-h(\rho))n}\cdot\text{poly}_k(n)\cdot k^{\rho n} \\ = \;& \text{poly}_{k,d}(n)\cdot 2^{(1-h(\rho))n}\cdot 2^{\rho n\cdot\log_2(k)} \\ = \;& \text{poly}_{k,d}(n)\cdot 2^{(1-h(\rho))n+\rho n\cdot\log_2(k)} \\ \underset{3.2}{=} \;& \text{poly}_{k,d}(n)\cdot 2^{n\left(1+\frac{1}{k+1}\log_2\frac{1}{k+1}+\frac{k}{k+1}\log_2\frac{k}{k+1}+\frac{1}{k+1}\log_2(k)\right)} \\ = \;& \text{poly}_{k,d}(n)\cdot 2^{n\left(1-\frac{1}{k+1}\log_2(k+1)+\frac{k}{k+1}\log_2(k)-\frac{k}{k+1}\log_2(k+1)+\frac{1}{k+1}\log_2(k)\right)} \\ = \;& \text{poly}_{k,d}(n)\cdot 2^{n\left(1+\log_2\frac{k}{k+1}\right)} \\ = \;& \text{poly}_{k,d}(n)\cdot\left(2\cdot\frac{k}{k+1}\right)^n \\ = \;& \text{poly}_{k,d}(n)\cdot\left(2-\frac{2}{k+1}\right)^n \end{aligned}$$

Somit erhalten wir insgesamt:

4.3 Lemma. *Für* $\rho = \frac{1}{k+1}$ *gilt:*

$$B(n,\rho,d) \cdot T_2(n,\rho) = \text{poly}_{k,d}(n) \cdot \left(2 - \frac{2}{k+1}\right)^n$$

4.3 Günstige Wahl von d

Der erste Summand von (1) kann durch geschickte Wahl von d kleiner gemacht werden als der zweite Summand, wenn man von einem polynomiellen Faktor absieht.
Der zweite Summand $2^{(1-h(\rho))n}$ von $T_1(n,\rho,d)$ kommt im Produkt $B(n,\rho,d) \cdot T_2(n,\rho)$ als Faktor vor. Daher reicht es zu zeigen, dass der erste Summand $2^{3n/d}$ kleiner ist als $\text{poly}_d(n) \cdot (B(n,\rho,d) \cdot T_2(n,\rho))$, da $k^{\rho n}$ bei der Wahl von $\rho = \frac{1}{k+1}$ für alle k größer als 1 ist und für $k \mapsto \infty$ gegen 1 geht. Somit muss d so gewählt werden, dass $2^{3n/d} \leq \left(2 - \frac{2}{k+1}\right)^n$ gilt, nach 4.3. $\left(2 - \frac{2}{k+1}\right)^n$ ist am kleinsten für $k = 3$. Somit kann der Fall $k = 3$ betrachtet werden, damit Gültigkeit für alle $k \geq 3$ erreicht wird.

$$\begin{aligned} 2^{3n/d} &\leq \left(2 - \frac{2}{k+1}\right)^n \\ \left(2^{3/d}\right)^n &\leq \left(2 - \frac{2}{3+1}\right)^n \\ 2^{3/d} &\leq 1.5 \end{aligned}$$

Da $2^{3/d}$ monoton fallend ist, für $d \to \infty$, interessiert uns das erste $d > 0$, für das die Ungleichung erfüllt ist. Für $d = 5$ gilt $2^{3/5} = 1.515\ldots$. Ab $d = 6$ erhalten wir die gewünschte Eigenschaft, da $2^{3/6} = 1.414\ldots$. Somit gilt das folgende Lemma:

4.4 Lemma. *Für* $\rho = \frac{1}{k+1}$ *und für alle* $d \in \mathbb{N}$ *mit* $d \geq 6$*, gilt:*

$$\text{poly}_{k,d}(n) \cdot (T_1(n,\rho,d) + B(n,\rho,d) \cdot T_2(n,\rho)) = \text{poly}_{k,d}(n) \cdot B(n,\rho,d) \cdot T_2(n,\rho)$$

4.4 Laufzeitanalyse des Hauptalgorithmus

In diesem Unterabschnitt können wir nun die Laufzeit mit der Wahl von $d = 6$ und $\rho = \frac{1}{k+1}$ bestimmen.

4.5 Satz. *Sei* $d = 6$ *und* $\rho = \frac{1}{k+1}$*, dann löst der Hauptalgorithmus* LÖSE(F,ρ,d) k-SAT *in Zeit* $\text{poly}_k(n) \cdot (2 - 2/(k+1))^n$*, wobei n die Anzahl der Variablen in der Eingabeformel ist.*

Beweis. Da in Lemma 4.1 die Korrektheit bereits gezeigt wurde, reicht es hier die Laufzeit entsprechend der Parameter zu betrachten.

Wir berechnen die Laufzeit nach der Formel (1). Der polynomielle Faktor hängt nicht mehr von d ab, da $d = 6$ gesetzt wurde:

$$\begin{aligned} & T_1(n,\rho,d) + B(n,\rho,d) \cdot T_2(n,\rho) \\ \underset{4.4}{=} & \operatorname{poly}_k(n) \cdot B(n, \frac{1}{k+1}, 6) \cdot T_2(n, \frac{1}{k+1}) \\ \underset{4.3}{=} & \operatorname{poly}_k(n) \cdot \left(2 - \frac{2}{k+1}\right)^n \end{aligned}$$

□

Der Hauptalgorithmus benötigt für die Konstruktion des Überdeckungscode nach dem Beweis von Lemma 3.9 exponentiellen Platz. Der Algorithmus kann aber mit Lemma 3.10 angepasst werden. Dann ist der Platzbedarf polynomiell.

4.6 Satz. *Sei* $\rho = \frac{1}{k+1}$*, dann kann der Hauptalgorithmus für jedes* $\varepsilon > 0$ *und jedes* $k \geq 3$ *angepasst werden, so dass er Zeit* $\operatorname{poly}_k(n) \cdot (2 - 2/(k+1) + \varepsilon)^n$ *und polynomiellen Platz benötigt.*

Beweis. Wir modifizieren den Hauptalgorithmus wie folgt: Anstatt Lemma 3.9 benutzen wir Lemma 3.10 mit $\delta := \log_2(\varepsilon(k+1)/(2k)+1)$. Wie oben nehmen wir an, dass n durch $b(\delta,\rho)$ teilbar ist. Dann kann die Gesamtlaufzeit wie folgt abgeschätzt werden:

$$\begin{aligned} & \operatorname{poly}_k(n) \cdot B'(n,\rho) \cdot T_2(n,\rho) \\ = & \operatorname{poly}_k(n) \cdot 2^{(1-h(\rho)+\delta)n} \cdot k^{\rho n} \\ = & \operatorname{poly}_k(n) \cdot 2^{n\left(1 + \frac{1}{k+1}\log_2 \frac{1}{k+1} + \frac{k}{k+1}\log_2 \frac{k}{k+1} + \frac{1}{k+1}\log_2(k) + \delta\right)} \\ \underset{\text{Def. } \delta}{=} & \operatorname{poly}_k(n) \cdot \left(\frac{2k}{k+1} \cdot \left(\frac{\varepsilon(k+1)}{2k} + 1\right)\right)^n \\ = & \operatorname{poly}_k(n) \cdot \left(2 - \frac{2}{k+1} + \varepsilon\right)^n \end{aligned}$$

□

In diesem Abschnitt wurde der Hauptalgorithmus vorgestellt. Dieser Algorithmus löst k-SAT in Zeit $\operatorname{poly}_k(n) \cdot (2 - 2/(k+1))^n$. Hierzu wird die lokale Suche aus Abschnitt 2 verwendet. Außerdem werden die Konstruktionen für Überdeckungscodes aus Abschnitt 3 benötigt. Im nächsten Abschnitt werden wir sehen, wie eine noch weiter verbesserte Laufzeit erreicht werden kann, indem wir eine modifizierte Version der Prozedur zur lokalen Suche benutzen.

5 Verbesserte lokale Suche

Die Prozedur SUCHE(F,a,r) aus dem Abschnitt 2 verwendet eine beliebige falsche Klausel zum Verzweigen. Die Zeitkomplexität dieser Prozedur ist $\text{poly}_k(n)\cdot k^r$. Eine sorgfältigere Wahl einer Klausel zum Verzweigen kann diese Schranke verbessern, und somit auch die Gesamtlaufzeit für den Hauptalgorithmus. Durch die sorgfältigere Wahl kann die Laufzeit für 3-KNF dann durch $\text{poly}(n)\cdot 2.848^r$, anstatt durch $\text{poly}(n)\cdot 3^r$, beschränkt werden. Damit erreichen wir für den Hauptalgorithmus mit verbesserter lokaler Suche die Laufzeitschranke $\text{poly}(n)\cdot 1.481^r$ für 3-SAT.
In [1] wird diese Verbesserung für den Fall $k=3$ gezeigt. Außerdem wird darauf hingewiesen, dass eine Verallgemeinerung für alle $k\geq 3$ möglich ist. Für den Fall $k=3$ werden die Beweise zum Teil durch eine vollständige Betrachtung aller möglichen Fälle gelöst. Dies wäre für allgemeines k äußerst aufwendig.
In dieser Arbeit wird die Verbesserung aus [1] verallgemeinert für beliebiges $k\geq 3$. Unter anderem wird die wichtige Definition der (3)-Beschränktheit aus [1] angepasst. Anhand dieser Definition werden dann grundlegende Eigenschaften hergeleitet, um die Verbesserung für $k\geq 3$ zu beweisen.
Das wichtigste Lemma beim Beweis in [1] wird durch ein etwas schwächere Aussage ersetzt, die ohne vollständige Fallunterscheidung auskommt. Auch die Prozedur zur verbesserten Lokalen Suche wird für $k\geq 3$ verallgemeinert. Außerdem wird noch gezeigt, dass eine Verbesserung für alle $k\geq 3$ erreicht wird.

5.1 (g)-Beschränktheit

In diesem Unterabschnitt führen wir die wichtigste Definition für unsere neue lokale Suche ein. Zunächst benötigen wir jedoch eine Notation, die es uns leicht ermöglicht die Anzahl an nicht erfüllten und erfüllten Literalen einer Klausel abzulesen.

5.1 Definition. Sei F eine Formel und a eine Belegung. Sei C eine Klausel in F. C ist eine

$$1^p0^m - \textit{Klausel} \text{ bezüglich } a,$$

wenn C genau p erfüllte Literale und genau m nicht erfüllte Literale unter a hat.

Zum Beispiel besteht eine $1^30^7 - \textit{Klausel}$ aus drei erfüllten und sieben nicht erfüllten Literalen. Falls $p=0$ gilt, werden diese Klauseln als 0^m-Klauseln bezeichnet. Entsprechendes gilt für $m=0$. Da wir Klauseln immer bezüglich einer Belegung a betrachten und aus dem Kontext ersichtlich ist, um welche Belegung es sich jeweils handelt, wird sie meistens nicht explizit nochmals dahinter geschrieben, auch wenn die Definition nur bezüglich einer Belegung sinnvoll ist.
Nun fehlt uns noch eine vorbereitende Definition.

5.2 Definition ((g)-falsch)**.** Sei F eine Formel in k-KNF, a eine Belegung und $g \in \mathbb{N}_0$.
F ist *(g)-falsch* unter a genau dann, wenn keine erfüllende Belegung mit Hamming-Abstand höchstens g von a für F existiert.

5.3 Bemerkung. Für $g = 0$ erhalten wir, dass keine erfüllende Belegung mit Hamming-Abstand höchstens 0 von a existiert. Somit bedeutet F ist (0)-falsch unter a nichts anderes als F ist falsch unter a.

5.4 Bemerkung. Wenn g fest ist, können wir in Zeit $\text{poly}_k(n)$ für gegebenes F und a überprüfen, ob F (g)-falsch ist unter a. Dazu kann die Prozedur $\text{SUCHE}(F,a,r)$ aus Abschnitt 2 benutzt werden. (g)-falsch unter a bedeutet, dass sich innerhalb der Kugel vom Radius g um a keine erfüllende Belegung befindet. Diese kann, nach Lemma 2.2, mit der Prozedur $\text{SUCHE}(F,a,r)$ für $r = g$ entschieden werden. Hierbei ist die Laufzeit, nach Lemma 2.3, durch $\text{poly}_k(n) \cdot k^g$ beschränkt. Für festes g kann somit in Zeit $\text{poly}_k(n)$ überprüft werden, ob F (g)-falsch ist unter a.

Beim Modifizieren des Verzweigens machen wir uns zu Nutze, dass nicht jeder rekursive Aufruf ausgeführt werden muss. Hierzu die Idee:
Wenn wir an einer 0^k-Klausel $l_1 \vee l_2 \vee \ldots \vee l_k$ verzweigen, so dass F die 1^1-Klausel $\overline{l_i}$ für ein $i \in \{1,2,\ldots,k\}$ enthält, dann führen wir $\text{SUCHE}(F_{|l_i=1}, a_{|l_i=1}, r-1)$ nicht aus, da die 1^1-Klausel $\overline{l_i}$ zur leeren Klausel wird in $F_{|l_i=1}$ und dieser Aufruf somit immer *falsch* liefert, nach Definition 1.4.
Diese Beobachtung fassen wir im folgenden Lemma zusammen:

5.5 Lemma. *Sei F eine Formel, falsch unter a. Wenn F eine 0^k-Klausel $l_1 \vee l_2 \vee \ldots \vee l_k$ und eine 1^1-Klausel $\overline{l_i}$ enthält, wobei $i \in \{1,2,\ldots,k\}$, dann gilt die folgende Äquivalenz: F ist erfüllbar innerhalb des Hamming-Abstands $\leq r$ von a genau dann, wenn ein $j \in \{1,2,\ldots,k\}$ existiert, so dass $j \neq i$ und $F_{|l_j=1}$ erfüllbar ist innerhalb des Hamming-Abstands $\leq r-1$ von $a_{|l_j=1}$.*

Beweis. Zunächst nehmen wir an, dass F erfüllbar ist innerhalb eines Hamming-Abstands $\leq r$ von a.
Da F erfüllbar ist, muss auch die Klausel $l_1 \vee l_2 \vee \ldots \vee l_k$ erfüllt werden. Da $l_1 \vee l_2 \vee \ldots \vee l_k$ eine 0^k-Klausel ist, muss ein Literal auf 1 gesetzt werden. Das Literal l_i kann nicht auf 1 gesetzt werden, da sonst die 1^1-Klausel $\overline{l_i}$ aus F zur leeren Klausel wird in $F_{|l_i=1}$ und F nicht erfüllbar wäre. Daher muss ein $j \neq i$ existieren, so dass $F_{|l_j=1}$ erfüllbar ist. Da dieses l_j auf 1 gesetzt wird, ist der Hamming-Abstand höchstens $\leq r-1$ von $a_{|l_j=1}$ zu einer erfüllenden Belegung.
Nun existiere ein $j \in \{1,2,\ldots,k\}$, so dass $j \neq i$ und $F_{|l_j=1}$ erfüllt ist, innerhalb des Hamming-Abstands $\leq r-1$ von $a_{|l_j=1}$.
Da nur ein Literal verändert wurde, ist F erfüllbar innerhalb des Hamming-Abstandes r von a, da dieses eine Literal auf 1 gesetzt wird um $F_{|l_j=1}$ zu erhalten. □

Nun betrachten wir bestimmte Typen von Formeln, die für die neue Version der Prozedur $\text{SUCHE}(F,a,r)$ benutzt werden. Die Definition beruht auf der Beobachtung für Lemma 5.5. Zunächst definieren wir die Typen von Formeln und anschließend zeigen wir, welche besonderen

Eigenschaften daraus entstehen. Anhand dieser Eigenschaften ergibt sich die neue Prozedur SUCHE(F,a,r) ganz intuitiv.

5.6 Definition ((g)-beschränkt)**.** Sei F eine nicht triviale Formel in k-KNF und a eine Belegung. F sei $(g-1)$-falsch unter a.
Falls $g=1$: F ist *(1)-beschränkt* bezüglich a genau dann, wenn einer der beiden Fälle gilt:

- F besitzt eine 0^h-Klausel bezüglich a für ein h, mit $1 \leq h < k$.
- F besitzt eine 0^k-Klausel bezüglich a, in der ein Literal l existiert, so dass $\overline{l}$ eine 1^1-Klausel von F ist.

Falls $1 < g < k$: F ist *(g)-beschränkt* bezüglich a genau dann, wenn einer der beiden Fälle gilt:

- F ist (h)-beschränkt bezüglich a für ein h, mit $1 \leq h < g$.
- F besitzt eine 0^k-Klausel bezüglich a, in der ein Literal l existiert, so dass die Formel $F_{|l=1}$ $(g-1)$-beschränkt ist bezüglich $a_{|l=1}$.

Falls $g=k$: F ist *(k)-beschränkt* bezüglich a genau dann, wenn einer der beiden Fälle gilt:

- F ist (h)-beschränkt bezüglich a für ein h, mit $1 \leq h < k$.
- F besitzt eine 0^k-Klausel bezüglich a, so dass $F_{|l=1}$ $(k-1)$-beschränkt ist bezüglich $a_{|l=1}$ für alle Literale l in dieser Klausel.

Diese Definition werden wir uns mit vier Beispielen für $k=3$ verdeutlichen:

1 Beispiel ((1)-Beschränktheit (Erster Fall))**.** Sei F eine Formel in 3-KNF und sei $C := (l_1 \vee l_2)$ eine Klausel aus F. Sei a eine Belegung, die l_1 und l_2 auf 0 setzt. Dann ist F falsch unter a, da C nicht erfüllt ist. Außerdem enthält F dann eine 0^2-Klausel bezüglich a, nämlich C. Beides zusammen liefert, dass F (1)-beschränkt bezüglich a ist.

2 Beispiel ((1)-Beschränktheit (Zweiter Fall))**.** Die zu betrachtende Formel in 3-KNF sei $F := (l_1 \vee l_2 \vee l_3) \wedge (\neg l_1)$. Sei a eine Belegung, die l_1, l_2 und l_3 auf 0 setzt. Dann hat F eine 0^3-Klausel und eine 1^1-Klausel bezüglich a. Das Literal l_1 kommt auch in der 1^1-Klausel als $\overline{l_1}$ vor. Da F auch falsch ist unter a, ist F somit (1)-beschränkt bezüglich a.

3 Beispiel ((2)-Beschränktheit (Zweiter Fall))**.** Sei $F := (l_1 \vee l_2 \vee l_3) \wedge (\neg l_1 \vee m_1 \vee m_2) \wedge (h_1 \vee h_2 \vee h_3)$ eine Formel in 3-KNF. Sei a eine Belegung, die alle Variablen auf 0 setzt. Dann enthält F die 0^3-Klausel $l_1 \vee l_2 \vee l_3$, die $1^1 0^2$-Klausel $\neg l_1 \vee m_1 \vee m_2$ und die 0^3-Klausel $h_1 \vee h_2 \vee h_3$ bezüglich a. F ist (1)-falsch unter a, da zwei Klauseln, die keine gemeinsamen Literale haben, jeweils nicht erfüllt sind. Es würde somit nicht reichen, ein Literal auf 1 zu setzen, um eine erfüllte Formel zu erhalten.
Wir bilden $F_{|l_1=1}$. Dabei werden alle Klauseln in denen l_1 vorkommt entfernt und aus den übrigen Klauseln wird das Literal $\overline{l_1}$ entfernt. Wir erhalten $F_{|l_1=1} = (m_1 \vee m_2) \wedge (h_1 \vee h_2 \vee h_3)$. $F_{|l_1=1}$

enthält dann eine 0^2-Klausel und eine 0^3-Klausel bezüglich $a_{|l_1=1}$. Somit ist $F_{|l_1=1}$ falsch unter $a_{|l_1=1}$. Da $F_{|l_1=1}$ eine 0^2-Klausel bezüglich $a_{|l_1=1}$ enthält, ist $F_{|l_1=1}$ (1)-beschränkt. Das heißt wir haben das Literal l_1, so dass $F_{|l_1=1}$ (1)-beschränkt ist bezüglich $a_{|l_1=1}$. Daher ist F (2)-beschränkt bezüglich a, denn in der Definition vom zweiten Fall von (g)-beschränkt wird gerade die Existenz eines solchen Literals verlangt.

Für den Fall der (3)-Beschränktheit haben wir einen Allquantor in der Definition, da $g = k$ gilt. Somit wird die Formel aus dem Beispiel etwas umfangreicher:

4 Beispiel ((3)-Beschränktheit (Zweiter Fall))**.** Sei $F := (m_1 \vee m_2 \vee m_3) \wedge (l_1 \vee l_2 \vee l_3) \wedge (\neg m_1 \vee \neg l_1 \vee s_1) \wedge (\neg m_2 \vee \neg l_2 \vee s_2) \wedge (\neg m_3 \vee \neg l_3 \vee s_3) \wedge (h_1 \vee h_2 \vee h_3)$ eine Formel in 3-KNF. Sei a eine Belegung, die alle Variablen auf 0 setzt. Dann enthält F die 0^3-Klauseln $m_1 \vee m_2 \vee m_3$ und $l_1 \vee l_2 \vee l_3$, die drei 1^20^1-Klauseln $\neg m_i \vee \neg l_i \vee s_i$ für $i = 1,2,3$ und die 0^3-Klausel $h_1 \vee h_2 \vee h_3$ bezüglich a. F ist (2)-falsch unter a, da drei Klauseln, die keine gemeinsamen Literale haben, jeweils nicht erfüllt sind. Es würde somit nicht reichen, zwei Literale auf 1 zu setzen, um eine erfüllte Formel zu erhalten.
Wir bilden $F_{|m_1=1}$ und erhalten $F_{|m_1=1} = (l_1 \vee l_2 \vee l_3) \wedge (\neg l_1 \vee s_1) \wedge (\neg m_2 \vee \neg l_2 \vee s_2) \wedge (\neg m_3 \vee \neg l_3 \vee s_3) \wedge (h_1 \vee h_2 \vee h_3)$. $F_{|m_1=1}$ ist analog zum letzten Beispiel (1)-falsch unter $a_{|m_1=1}$ und $F_{|m_1=1,l_1=1}$ ist (1)-beschränkt bezüglich $a_{|m_1=1,l_1=1}$. Da $F_{|m_1=1,l_1=1}$ die 0^1-Klausel s_1 bezüglich $a_{|m_1=1,l_1=1}$ enthält und falsch ist unter $a_{|m_1=1,l_1=1}$.
Daher ist $F_{|m_1=1}$ (2)-beschränkt bezüglich $a_{|m_1=1}$. Analog sind auch $F_{|m_2=1}$ und $F_{|m_3=1}$ (2)-beschränkt. Somit ist $F_{|m_i=1}$ (2)-beschränkt bezüglich $a_{|m_i=1}$, wobei $i = 1,2,3$ und die Literale m_i aus einer 0^k-Klausel bezüglich a stammen. Daher ist F (3)-beschränkt bezüglich a.

Im nächsten Unterabschnitt werden wir Definition 5.6 benutzen um zu zeigen, dass nicht (k)-beschränkte Formeln immer erfüllbar sind.

5.2 Nicht (k)-beschränkte Formeln sind immer erfüllbar

Im Folgenden werden wir zeigen, dass nicht (k)-beschränkte Formeln immer erfüllbar sind. Wir werden daher nur (g)-beschränkte Formeln in der neuen Prozedur SUCHE(F,a,r) betrachten müssen. Zuerst allerdings überlegen wir uns einige Eigenschaften, die wir anhand der Definition 5.6 herleiten können. Zunächst ein einfaches Lemma über 0^k-Klauseln.

5.7 Lemma. *Sei F eine nicht triviale Formel in k-KNF, falsch unter einer Belegung a und nicht (1)-beschränkt bezüglich a, dann enthält F eine 0^k-Klausel bezüglich a und keine 0^h-Klausel bezüglich a, mit $1 \leq h < k$.*

Beweis. Nach Definition 1.4 ist eine Formel trivial, wenn sie die leere Formel ist oder eine leere Klausel enthält.
Da F falsch ist unter a, existiert mindestens eine nicht erfüllte 0^h-Klausel mit $1 \leq h \leq k$, da F nicht trivial ist. Falls $h < k$ gilt, folgt nach Definition 5.6, dass F dann (1)-beschränkt ist. Dies ist ein Widerspruch zur Voraussetzung, daher gilt $h = k$. □

5.8 Bemerkung. Jede bezüglich einer Belegung a nicht (m)-beschränkte, unter a falsche Formel, ist insbesondere nicht (1)-beschränkt bezüglich a, für $1 \leq m \leq k$. Daher enthält jede solche Formel eine 0^k-Klausel nach Lemma 5.7. Man beachte, dass wir Gültigkeit auch für $m = k$ haben.

Wir müssen bei der Definition 5.6 verlangen, dass F $(g-1)$-falsch ist, damit beim Verzweigen keine trivialen Formeln gebildet werden. Wir nutzen die nächsten Lemmata um diese Eigenschaft zu zeigen.

5.9 Lemma. *Sei F eine nicht triviale Formel in k-KNF, (g)-falsch unter einer Belegung a und nicht (g)-beschränkt bezüglich a. Hierbei ist $1 \leq g \leq k$. Dann ist $F_{|l_j=1}$ nicht trivial für beliebige Literale l_i, aus einer beliebigen, unter a falschen, Klausel von F.*

Beweis. Wir können Lemma 5.7 anwenden und erhalten, dass eine falsche Klausel immer eine 0^k-Klausel bezüglich a ist. Außerdem existiert immer eine solche Klausel. Sei ohne Beschränkung der Allgemeinheit $l \vee m_1 \vee \ldots m_{k-1}$ diese Klausel und l ein beliebiges Literal aus der Klausel (gegebenenfalls muss umsortiert werden). Die leere Klausel kann in $F_{|l=1}$ nur generiert werden, wenn F die Klausel $\bar{l}$ enthält. Dies ist nicht der Fall, falls F nicht (1)-beschränkt ist bezüglich a, was nach Voraussetzung gilt.
$F_{|l=1}$ ist auch nicht die leere Formel, da F (g)-falsch ist unter a. Denn eine leere Formel ist immer wahr. F und $F_{|l=1}$ haben den Hamming-Abstand 1, somit kann $F_{|l=1}$ nicht die leere Formel sein. Dies wäre ein Widerspruch zur Voraussetzung, dass F (g)-falsch ist unter a. □

Beim Verzweigen an nicht (k)-beschränkten Formel gelten bestimmte Eigenschaften, wie das folgende Lemma zeigt.

5.10 Lemma. *Sei F eine nicht triviale Formel in k-KNF, nicht (k)-beschränkt bezüglich einer Belegung a und $(k-1)$-falsch. Dann existiert ein Literal l aus einer 0^k-Klausel bezüglich a, so dass $F_{|l=1}$ nicht $(k-1)$-beschränkt bezüglich $a_{|l=1}$ und nicht trivial ist.*

Beweis. Sei $l_1 \vee \ldots \vee l_k$ eine beliebige 0^k Klausel bezüglich a aus F. Diese existiert nach Lemma 5.7. Wir nehmen an, dass $F_{|l_i=1}$ $(k-1)$-beschränkt ist für alle $i = 1, \ldots, k$. $F_{|l_i=1}$ ist nicht trivial nach Lemma 5.9 und $(k-2)$-falsch. Dann ist F nach Definition 5.6 (k)-beschränkt. Das ist ein Widerspruch zur Voraussetzung. Somit erhalten wir, dass ein $i \in \{1, \ldots, k\}$ existiert, so dass $F_{|l_i=1}$ nicht $(k-1)$-beschränkt ist bezüglich $a_{|l=1}$. Wir setzen $l = l_i$ und das Lemma gilt. □

Der Beweis des letzten Lemmas lässt vermuten, dass auch ein Zusammenhang zwischen nicht (g)-beschränkten Formeln $(1 < g < k)$ und den Formel nach dem Verzweigen besteht. Allerdings enthält die Definition von (g)-beschränkten Formeln einen Existenzquantor. Daher sind beim Verzweigen alle entstehenden Formeln nicht $(g-1)$-beschränkt. Diesen Zusammenhang formulieren und beweisen wir im nächsten Lemma.

5.11 Lemma. *Sei F eine nicht triviale Formel in k-KNF, nicht (g)-beschränkt $(1 < g < k)$ bezüglich einer Belegung a und $(g-1)$-falsch. Dann ist $F_{|l_i=1}$ nicht trivial und nicht $(g-1)$-beschränkt bezüglich $a_{|l_i=1}$ für alle l_i aus einer beliebigen unter a falschen Klausel von F.*

Beweis. Sei $l_1 \vee \ldots \vee l_k$ eine beliebige 0^k Klausel aus F. Diese existiert nach Lemma 5.7. Außerdem gibt es keine 0^h Klausel mit $1 \leq h < k$. Sei l_i beliebig, wobei $i \in \{1,\ldots,k\}$. Wir nehmen an, dass $F_{|l_i=1}$ $(g-1)$-beschränkt ist. $F_{|l_i=1}$ ist $(g-2)$-falsch und nicht trivial nach Lemma 5.9. Wir erhalten mit Definition 5.6, dass F (g)-beschränkt ist. Das ist ein Widerspruch zur Voraussetzung und somit gilt das Lemma für jedes l_i. □

5.12 Korollar. *Sei F eine nicht triviale Formel in k-KNF, nicht (g)-beschränkt bezüglich einer Belegung a $(1 < g < k)$ und $(g-1)$-falsch. Dann ist $F_{|l_i=1}$ nicht (h)-beschränkt bezüglich $a_{|l_i=1}$ für alle l_i aus einer beliebigen unter a falschen Klausel von F, wobei $1 \leq h < g$ ist.*

Beweis. Da jede nicht (g)-beschränkte Formel nach Definition auch nicht (h)-beschränkt ist für $h = 1,\ldots,g-1$, kann für jedes h das Lemma 5.11 angewandt werden und wir erhalten die gewünschte Aussage. □

Mit Hilfe des letzten Korollars können wir die Aussage aus Lemma 5.9 verallgemeinern.

5.13 Korollar. *Sei F eine nicht triviale Formel in k-KNF, (g)-falsch unter einer Belegung a und nicht (g)-beschränkt bezüglich a. Hierbei ist $1 \leq g < k$.*
Dann ist $F_{|l_1=1,l_2=1,\ldots,l_g=1}$ nicht trivial. l_1 ist ein beliebiges Literal aus einer unter a falschen Klausel von F. Für $1 < i \leq g$ ist l_i ein beliebiges Literal aus einer unter $a_{|l_1=1,\ldots,l_{i-1}=1}$ falschen Klausel von $F_{|l_1=1,\ldots,l_{i-1}=1}$.

Beweis. Durch Anwenden von Lemma 5.11 erhalten wir, dass $F_{|l_1=1}$ nicht $(g-1)$-beschränkt bezüglich $a_{|l_1=1}$ und nicht trivial ist. Dies lässt sich induktiv bis $F_{|l_1=1,l_2=1,\ldots,l_g=1}$ fortsetzen und wir erhalten die gewünschte Aussage. □

Nun formulieren wir eine Aussage über Klauseln, die Formeln (k)-beschränkt machen. Diese Formeln sind dabei jedoch nicht $(k-1)$-beschränkt.

5.14 Lemma. *Sei F eine nicht triviale Formel in k-KNF, (k)-falsch unter einer Belegung a, (k)-beschränkt bezüglich a und nicht $(k-1)$-beschränkt bezüglich a. Sei $F' = F_{|l_{1,j}=1,l_{2,j}=1,\ldots,l_{k-1,j}=1}$, hierbei steht der erste Index für den Verzweigungsschritt und der zweite für den Index des Literals, welches beim ersten Verzweigen auf 1 gesetzt wurde, in der Klausel an der zuerst verzweigt wurde.*
Dann gelten folgende Aussagen:

1. *F' ist nicht trivial und (1)-beschränkt, wobei die Literale $l_{1,j}$ paarweise verschieden sind für alle $j = 1,\ldots,k$.*
2. *F enthält bereits alle Klauseln, an denen beim Bilden von F' verzweigt wird.*
3. *F' ist immer aufgrund eines der beiden Fälle (1)-beschränkt:*
   a) *F' enthält eine 0^1-Klausel n_1, dann enthält F die $0^1 1^{k-1}$-Klausel $n_1 \vee \overline{l_{1,j}} \vee \ldots \vee \overline{l_{k-1,j}}$ für alle j.*

b) F' enthält die 1^1-Klausel $\overline{l_{k,j}}$ und eine 0^k-Klausel $l_{k-1,j} \vee m_{1,j} \vee \ldots \vee m_{k-1,j}$, dann enthält F die 1^k-Klauseln $\overline{l_{1,j}} \vee \ldots \vee \overline{l_{k-1,j}} \vee \overline{l_{k,j}}$ für alle j.

Beweis. Da F nicht $(k-1)$-beschränkt ist, kann nach Definition 5.6 nur ein Fall gelten: F enthält eine 0^k-Klausel $l_{1,1} \vee l_{1,2} \vee \ldots \vee l_{1,k}$ so dass $F_{|l_{1,j}=1}$ $(k-1)$-beschränkt ist für alle $j = 1, \ldots, k$. Die Literale $l_{1,j}$ sind paarweise verschieden für alle j, da sie aus einer Klausel stammen.
Nach Lemma 5.11 ist $F_{|l_{1,j}=1}$ nicht trivial und nicht $(k-2)$-beschränkt für alle j. Daher enthält $F_{|l_{1,j}=1}$ nach Definition 5.6 eine 0^k-Klausel. Ohne Beschränkung der Allgemeinheit (gegebenenfalls muss umsortiert werden) sei $l_{2,j}$ ein Literal aus dieser Klausel, so dass $F_{|l_{1,j}=1,l_{2,j}=1}$ $(k-2)$-beschränkt ist, dieses existiert nach Definition 5.6. Dies lässt sich wiederum mit Lemma 5.11 induktiv bis zur Formel $F' := F_{|l_{1,j}=1,l_{2,j}=1,\ldots,l_{k-1,j}=1}$ fortsetzen. Diese ist nicht trivial und (1)-beschränkt, somit gilt 1. Es wird immer an 0^k Klauseln verzweigt. Da F in k-KNF ist, müssen alle diese Klauseln bereits in F enthalten sein, somit gilt auch 2.
F' ist (1)-beschränkt, somit können 2 Fälle nach Definition 5.6 gelten. Zunächst nehmen wir an, dass F' eine 0^h-Klausel $n_1 \vee \ldots \vee n_h$ mit $h < k$ enthält. Diese Klausel muss beim Verzweigen entstanden sein, da F nicht (1)-beschränkt ist. Das heißt sie wird mit den Negationen von Literalen aufgefüllt, die beim Verzweigen auf 1 gesetzt wurden. Wir behaupten, dass $h = 1$ gilt und F die Klausel $n_1 \vee \overline{l_{1,j}} \vee \ldots \vee \overline{l_{k-1,j}}$ enthält und zeigen dies per Widerspruch. Somit nehmen wir an, dass $h > 1$ gilt. Dann kann die in F enthaltene, aus der 0^h Klausel entstehende Klausel höchstens $k-2$ Literale enthalten, die beim Verzweigen auf 1 gesetzt wurden. Damit wurde diese Klausel mindestens einmal beim Verzweigen nicht verändert. Somit kann dieses Verzweigen ausgelassen werden. Da nur an bereits in F enthaltenen Klauseln verzweigt wird, ist das möglich. Dann würden wir eine (1)-beschränkte Formeln bereits nach $k-2$ Verzweigungen erhalten und das macht F im Widerspruch zur Annahme $(k-1)$-beschränkt. Somit ist $h = 1$ und die Klausel $n_1 \vee \overline{l_{1,j}} \vee \ldots \vee \overline{l_{k-1,j}}$ muss in F enthalten sein. Denn wenn nicht alle $l_{i,j}$, die beim Verzweigen auf 1 gesetzt wurden, als $\overline{l_{i,j}}$ in der Klausel enthalten sind, erhalten wir analog nach höchstens $k-2$ Verzweigungen, dass F $(k-1)$-beschränkt ist. Somit ist diese Klausel in F vorhanden, wir erhalten den Fall 3a und das Lemma gilt für diesen Fall.
Es bleibt der Fall einer 1^1 Klausel $\overline{l_{k,j}}$ und einer 0^k-Klausel $l_{k,j} \vee m_{1,j} \vee \ldots \vee m_{k-1,j}$ in F' zu zeigen. Analog zum letzten Fall wird die Klausel $\overline{l_{k,j}}$ bei jeder vorhergehenden Formel um das Literal erweitert, das beim Verzweigen auf 1 gesetzt wurde. Dies geschieht $k-1$ mal, und wir erhalten eine 1^k Klausel $\overline{l_{1,j}} \vee \overline{l_{2,j}} \vee \ldots \vee \overline{l_{k,j}}$, die bereits in F enthalten ist. Somit erhalten wir den Fall 3b, und das Lemma gilt. □

Jetzt können wir den wichtigsten Satz für unsere neue Prozedur SUCHE(F, a, r) beweisen. Aufgrund des nächsten Satzes brauchen rekursive Aufrufe für nicht (k)-beschränkte Formeln nicht ausgeführt werden, da diese sich immer als erfüllbar erweisen, wenn F $(k-1)$-falsch ist.

5.15 Satz. *Sei F eine Formel in k-KNF, $(k-1)$-falsch und nicht (k)-beschränkt bezüglich einer Belegung a. Dann ist F erfüllbar.*

Beweis. Falls F nicht (k)-falsch ist, gilt der Satz, da dann F ohne Einschränkung erfüllbar ist.
Somit gehen wir davon aus, dass F (k)-falsch ist. Nach Lemma 5.10 existiert ein Literal l, so

dass $F_{|l=1}$ nicht $(k-1)$-beschränkt ist bezüglich a. Wir nehmen an, dass $F_{|l=1}$ (k)-beschränkt ist bezüglich a. Dann ist Lemma 5.14 anwendbar und $F_{|l=1}$ beinhaltet alle Klauseln, die $F_{|l=1}$ (k)-beschränkt machen. Da es sich um Klauseln mit k Literalen handelt, sind diese bereits in F vorhanden und machen F damit im Widerspruch zur Annahme (k)-beschränkt. Damit ist $F_{|l=1}$ ebenfalls nicht (k)-beschränkt und $(k-1)$-falsch.
Analog können wir zeigen, dass ein Literal l_2 existiert, so dass $F_{|l=1,l_2=1}$ nicht (k)-beschränkt und $(k-1)$-falsch ist. Hierbei enthält $F_{|l=1,l_2=1}$ echt weniger falsche Klauseln als $F_{|l=1}$. Wenn wir das Argument induktiv fortsetzen, gelangen wir nach endlich vielen Schritten zu einer Formel, die nicht mehr $(k-1)$-falsch ist. Diese ist dann durch die aktuelle Belegung erfüllt nach Definition 5.2. Somit ist auch F erfüllbar. □

5.3 Neue Prozedur SUCHE(F,a,r)

Nun können wir mit Hilfe der Aussagen des letzten Unterabschnitts die neue Version der Prozedur SUCHE(F,a,r) formulieren. Wie die ursprüngliche Version der Prozedur SUCHE(F,a,r) aus Abschnitt 2, erhält auch die neue Version als Eingabe eine Formel F in k-KNF mit n Variablen, eine initiale Belegung a und einen Radius r. Wenn F eine erfüllende Belegung innerhalb der Kugel vom Radius r um a besitzt, dann gibt die Prozedur *wahr* zurück.

Prozedur SUCHE(F,a,r).

1. Wenn F nicht $(k-1)$-falsch ist unter a, gib *wahr* zurück.
2. Wenn $r \leq k-1$ ist, dann gib *falsch* zurück.
3. Falls F die leere Klausel enthält, gib *falsch* zurück.
4. Für $m = 1,\ldots,k$: Falls F (m)-beschränkt ist bezüglich a, verzweige an einer falschen Klausel, die zur (m)-Beschränktheit beiträgt.
5. Gib *wahr* zurück.

Die 1. Zeile kann, nach Bemerkung 5.4, in Zeit $\text{poly}_k(n)$ realisiert werden. Man beachte, dass die letzte Zeile nur ausgeführt wird, wenn nicht in der 4. Zeile verzweigt wurde. Denn beim Verzweigen wird ja bereits ein Wert zurückgegeben, somit wird kein weiterer Code ausgeführt. Somit wird auch die Schleife in Zeile 4 nach dem Verzweigen nicht weiter ausgeführt.
Die Korrektheit der Prozedur ergibt sich relativ leicht aus den vorherigen Überlegungen. Da in der 1. Zeile unabhängig von r auf $(k-1)$-falsch geprüft wird, können für $r < k-1$ auch erfüllende Belegungen außerhalb der Kugel vom Radius r um a gefunden werden. Dies kann unabhängig von r auch in Zeile 5 passieren. Daher gilt, im Gegensatz zur Korrektheitsanalyse der alten Prozedur in Lemma 2.2, in diesem Lemma nur die Implikation und keine Äquivalenz. Was allerdings nichts an der Korrektheit ändert, da immer *falsch* zurückgegeben wird, wenn F keine erfüllende Belegung besitzt.

5.16 Lemma. SUCHE(F,a,r) *gibt wahr zurück, wenn F unter a eine erfüllende Belegung innerhalb der Kugel vom Radius r um a hat.*

Beweis. Wir zeigen die Behauptung per Induktion nach r. Für den Induktionsanfang sei zunächst $0 \leq r < k-1$. Wenn F nicht $(k-1)$-falsch ist unter a gibt SUCHE(F,a,r) *wahr* zurück. Dies ist genau dann der Fall, wenn eine erfüllende Belegung in der Kugel vom Radius r um a für F existiert. Somit gilt der Induktionsanfang.
Den Induktionsschritt führen wir von $r-1$ nach r. Sei $r > k-1$. In diesem Fall wird entweder an einer Klausel verzweigt, die belegt, dass F (i)-beschränkt, für $i = 1,\ldots,k$ ist, oder es wird in der letzten Zeile *wahr* zurück gegeben. Wenn in der letzten Zeile *wahr* zurückgegeben wurde, ist die Formel F nicht (k)-beschränkt und $(k-1)$-falsch. Dann kann Satz 5.15 angewandt werden und wir erhalten, dass F erfüllbar ist. Somit liefert der Algorithmus in diesem Fall insbesondere dann *wahr*, wenn F unter a eine erfüllende Belegung innerhalb der Kugel vom Radius r um a hat. Für diesen Fall gilt das Lemma.
Daher nehmen wir jetzt an, dass an einer Klausel verzweigt wird, die die (i)-Beschränktheit von F belegt. Dann wird SUCHE$(F_{|l_j=1}, a_{|l_j=1}, r-1)$ für alle Literale l_j aus dieser Klausel aufgerufen. Der Algorithmus gibt nur *wahr* zurück, wenn einer der Aufrufe *wahr* liefert. Dies passiert, wenn $F_{|l_j=1}$ unter $a_{|l_j=1}$ eine erfüllende Belegung innerhalb der Kugel vom Radius $r-1$ um $a_{|l_j=1}$ für mindestens ein j hat, nach Induktionsvoraussetzung. Dies ist jedoch äquivalent dazu, dass F unter a eine erfüllende Belegung innerhalb der Kugel vom Radius r um a hat, nach Lemma 2.1. Somit gilt das Lemma für alle $r \geq 0$. □

5.4 Anzahl der Blätter des Rekursionsbaumes $L(F,a,r)$

Es bezeichne $L(F,a,r)$ die Anzahl der Blätter des Rekursionsbaumes von SUCHE(F,a,r). In diesem Unterabschnitt zeigen wir zunächst einige vorbereitende Lemmata, die uns auch beim Verständnis des Verzweigens helfen. Zunächst beweisen wir ein Lemma, welches uns hilft zu verstehen, welche Auswirkungen es hat, immer an bestimmten Klauseln zu verzweigen. Wenn wir noch mal an die Prozedur denken, wird immer an Klauseln verzweigt, die die (g)-Beschränktheit der jeweils aktuellen Formel zeigen. Dabei ist vor allem zu beachten, dass immer zuerst geprüft wird, ob eine Formel $(g-1)$-beschränkt ist, bevor auf (g)-Beschränktheit geprüft wird. Somit sind die zu betrachtenden Formeln (g)-beschränkt, jedoch nicht $(g-1)$-beschränkt. Daher muss jeweils nur der zweite Fall aus der Definition 5.6 beachtet werden.
$L(F,a,r)$ hängt von F und a ab. Trotzdem wird beim nächsten Lemma $L(r)$ betrachtet. Denn wir werden $L(F,a,r)$ abschätzen, so dass diese Abschätzung für alle F und a gilt. Somit wird bei den Lemmata immer von worst-case F und a ausgegangen, ohne dass diese konkret bekannt sind. Das heißt es wird, falls nichts Näheres bekannt ist, von (k)-beschränkten Formeln ausgegangen. Da bei diesen die maximale Anzahl von $k-1$ Verzweigungen nötig ist, bis wir zu einer (1)-beschränkten Formel gelangen um einen Aufruf weglassen zu können. Daher erhalten wir Aussagen für $L(r)$, für die dann $L(F,a,r) \leq L(r)$ gilt für alle F und a.

5.17 Lemma. *Sei F eine nicht triviale Formel, a eine Belegung, sei F $(k-1)$-falsch unter a und sei $r \geq k$. Wenn* SUCHE(F,a,r) *an einer Klausel verzweigt, die belegt, dass F (g)-beschränkt ist, mit $1 \leq g < k$, dann gilt für $L(r)$:*

$$L(r) \leq (k-1) \cdot \sum_{i=1}^{g} L(r-i)$$

Beweis. Zunächst gilt $r \geq k$, daher kann immer $L(r-i)$ gebildet werden.
Wir zeigen das Lemma per Induktion nach g. Für den Induktionsanfang betrachten wir den Fall $g = 1$. Dann ist die Formel (1)-beschränkt. Nach Definition 5.6 gibt es dafür zwei mögliche Fälle. Zunächst betrachten wir denn Fall einer 0^h-Klausel, wobei $1 \leq h < k$. Dann gilt $h \leq k-1$. SUCHE(F,a,r) wird daher höchstens $k-1$ rekursive Aufrufe über diese Klausel machen.
Im anderen Fall haben wir eine 0^k-Klausel, die ein Literal l beinhaltet, so dass $\bar{l}$ eine 1^1-Klausel ist. Nach Lemma 5.5 benötigen wir keinen rekursiven Aufruf für das Literal l und somit genügen höchstens $k-1$ Aufrufe. Wir sehen diesen Zusammenhang in Abbildung 1. Dabei wird davon ausgegangen, dass diese $k-1$ Aufrufe jeweils zu (k)-beschränkten Formeln führen, denn für diese wird $L(r-1)$ am größten und somit gilt dann die Abschätzung auch für alle anderen Fälle. In beiden Fällen genügen insgesamt $k-1$ Aufrufe mit um 1 reduziertem Radius. Wir erhalten daher $L(r) \leq (k-1) \cdot L(r-1)$ und das Lemma gilt für $g = 1$. Somit ist der Induktionsanfang gezeigt.

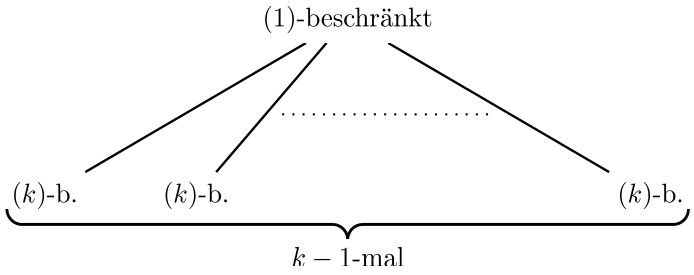

Abbildung 1: Verzweigen an einer Klausel, die die Formel (1)-beschränkt macht

Nun zum Induktionsschritt:
Die Formel ist (g)-beschränkt, jedoch nicht $(g-1)$-beschränkt, da sonst bereits bei $g-1$ verzweigt worden wäre. Nach Definition 5.6 kann dann nur der Fall einer 0^k-Klausel eintreten, die ein Literal l beinhaltet, so dass beim rekursiven Aufruf für dieses Literal eine $(g-1)$-beschränkte Formel entsteht. Somit haben wir $k-1$ Aufrufe mit dem Radius $r-1$, über diese ist nichts weiter bekannt. Wir gehen daher davon aus, dass es sich um (k)-beschränkte Formeln handelt. Außerdem haben wir noch einen Aufruf, auf den die Induktionsvoraussetzung angewandt werden kann. Dieser Zusammenhang ist in Abbildung 2 zu sehen. Insgesamt erhalten wir:

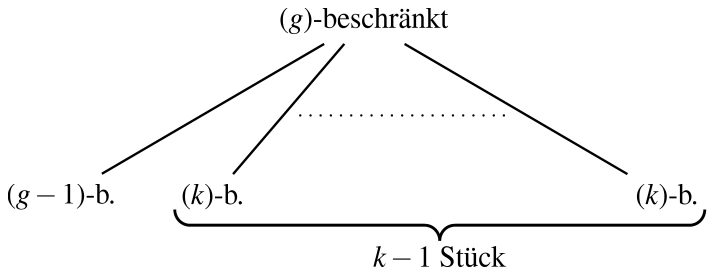

Abbildung 2: Verzweigen an einer Klausel, die die Formel (g)-beschränkt macht

$$\begin{aligned} L(r) &\underset{IV}{\leq} (k-1)\cdot L(r-1) + (k-1)\cdot \sum_{i=1}^{g-1} L((r-1)-i) \\ &= (k-1)\cdot L(r-1) + (k-1)\cdot \sum_{i=2}^{g} L(r-i) \\ &= (k-1)\cdot \sum_{i=1}^{g} L(r-i) \end{aligned}$$

Somit gilt die Aussage für alle g. □

Nun bleibt noch der Fall von (k)-beschränkten Formeln, dieser ergibt sich jedoch als leichtes Korollar.

5.18 Korollar. *Sei F eine nicht triviale Formel, a eine Belegung, sei F $(k-1)$-falsch unter a und sei $r \geq k$. Wenn* SUCHE(F,a,r) *an einer Klausel verzweigt, die belegt, dass F (k)-beschränkt ist, dann gilt für $L(r)$:*

$$L(r) \leq k\cdot(k-1)\cdot \sum_{i=2}^{k} L(r-i)$$

Beweis. Nach Definition 5.6 kann nur der Fall einer 0^k-Klausel auftreten, so dass beim Verzweigen an dieser Klausel alle Formeln in den rekursiven Aufrufen $(k-1)$-beschränkt sind. Es kann kein anderer Fall eintreten, da die Formel nicht $(k-1)$-beschränkt ist. Wir sehen diesen Zusammenhang in Abbildung 3. Jeder dieser Aufrufe ist durch $(k-1)\cdot \sum_{i=1}^{k-1} L((r-1)-i)$ beschränkt nach Lemma 5.17, da diese $(k-1)$-beschränkt sind nach Definition 5.6. Insgesamt ergibt sich:

$$\begin{aligned} L(r) &\leq k\cdot L(r-1) \\ &\underset{5.17}{\leq} k\cdot(k-1)\cdot \sum_{i=1}^{k-1} L((r-1)-i) \\ &= k\cdot(k-1)\cdot \sum_{i=2}^{k} L(r-i) \end{aligned}$$

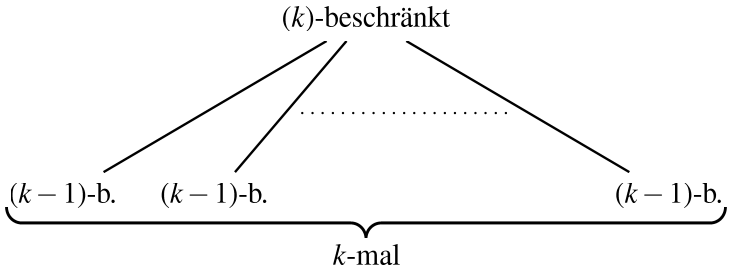

Abbildung 3: Verzweigen an einer Klausel, die die Formel (k)-beschränkt macht

und das Korollar gilt. □

Die Abbildung 4 dient dazu, sich noch einmal die Zusammenhänge in den letzten beiden Lemmata zu verdeutlichen. In der Abbildung werden für den Fall $k = 3$ die ersten Ebenen der Rekursion aufgezeigt. Somit wird auch der Zusammenhang zwischen (g)- und (k)-beschränkten Formeln klarer. Zusätzlich steht links neben den Knoten in jeder Ebene die Anzahl an Knoten in dieser Ebene. Mit Verbesserung steht der Wert oben und ohne Verbesserung unten. Im nächsten Unterabschnitt definieren wir die Funktion $H_k(r)$, die uns hilft $L(F,a,r)$ abzuschätzen.

5.5 Die Funktion $H_k(r)$

In diesem Unterabschnitt werden wir die rekursive Funktion H_k definieren. Sie hilft uns $L(F,a,r)$ abzuschätzen.

5.19 Definition (Funktion $H_k(r)$)**.** Sei $r \geq 0$. Dann ist $H_k(r)$ definiert durch:
$H_k(r) = k^r$ für $r \leq k-1$ und

$$H_k(r) = k \cdot (k-1) \cdot \sum_{i=2}^{k} H_k(r-i)$$

für $r \geq k$.

Nun zeigen wir einige Abschätzungen für die Funktion $H_k(r)$, die uns bereits für $L(r)$ bekannt sind.

5.20 Lemma. *Sei $r \geq 1$, dann gilt:*

$$(k-1) \cdot H_k(r-1) < H_k(r)$$

Beweis. Wir beweisen das Lemma per Induktion nach r. Für den Induktionsanfang sei zunächst $1 \leq r \leq k-1$.

$$(k-1) \cdot H_k(r-1) \underset{(k\geq 3)}{<} k \cdot H_k(r-1) \underset{5.19}{=} k \cdot k^{r-1} = k^r \underset{5.19}{=} H_k(r)$$

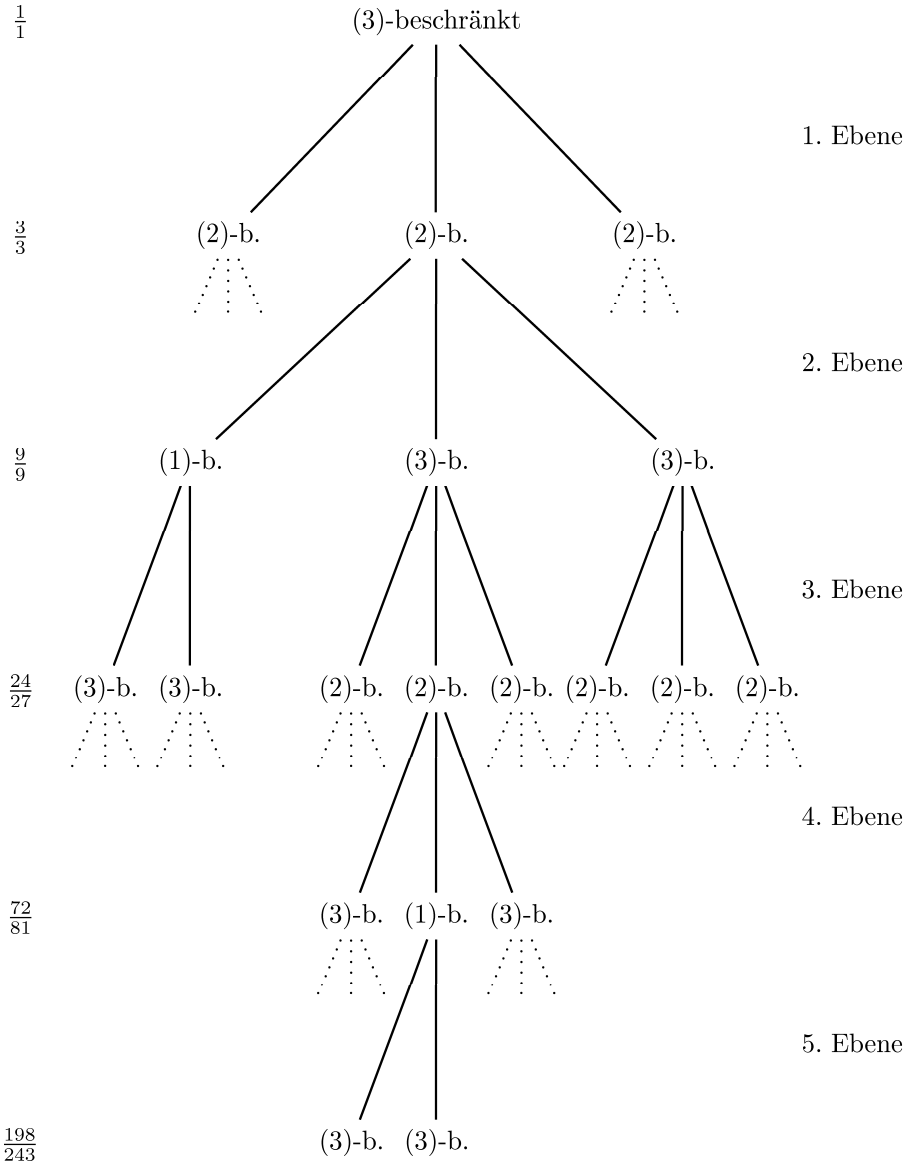

Abbildung 4: Verzweigen an einer Klausel, die die 3-KNF Formel (3)-beschränkt macht

Den Induktionsschritt von r nach $r+1$ führen wir unter der Annahme, dass die Aussage für alle $0 \leq i \leq r-1$ gilt. Sei $r > k-1$.

$$\begin{aligned}
& (k-1) \cdot H_k((r+1)-1) \\
\underset{5.19}{=} \; & (k-1) \cdot \left(k \cdot (k-1) \cdot \sum_{i=2}^{k} H_k(r-i) \right) \\
\underset{IV}{\leq} \; & (k-1) \cdot \left(k \cdot \sum_{i=2}^{k} H_k((r+1)-i) \right) \\
= \; & k \cdot (k-1) \cdot \sum_{i=2}^{k} H_k((r+1)-i) \\
\underset{5.19}{=} \; & H_k(r+1)
\end{aligned}$$

Somit gilt das Lemma für alle $r \geq 1$. □

Mit Hilfe des letzten Lemmas können wir leicht zwei Eigenschaften der Funktion $H_k(r)$ herleiten.

5.21 Korollar. *Die Funktion $H_k(r)$ ist streng monoton steigend und positiv.*

Beweis. Nach Lemma 5.20 ist $(k-1) \cdot H_k(r-1) < H_k(r)$, wobei $k \geq 3$ und $r \geq 1$ gilt. Daher ist $H_k(r)$ streng monoton steigend. Man beachte, dass zusätzlich $H_k(0) = 1 < k = H_k(1)$ gilt. Da $H_k(0) = 1$ gilt, ist somit $H_k(r)$ auch positiv für alle $r \geq 0$. □

Das letzte Lemma und das Korollar liefern uns eine noch stärkere Abschätzung, die wir nun formulieren können:

5.22 Korollar. *Sei $r > k-1$ und $1 \leq g < k-1$, dann gilt:*

$$(k-1) \cdot \sum_{i=1}^{g} H_k(r-i) < H_k(r)$$

Beweis. Die Summe $\sum_{i=1}^{g} H_k(r-i)$ wird maximal für $g = k-1$, da nach Korollar 5.21 $H_k(r')$ streng monoton steigend und positiv ist für alle $r' \geq 0$. Somit reicht es die Behauptung für

$g = k-1$ zu zeigen, um die Gültigkeit für alle $1 \leq g < k$ zu erhalten.

$$\begin{aligned}
& (k-1) \cdot \sum_{i=1}^{g} H_k(r-i) \\
\leq \quad & (k-1) \cdot \sum_{i=1}^{k-1} H_k(r-i) \\
\leq \quad & \sum_{i=1}^{k-1} (k-1) \cdot H_k(r-i) \\
\underset{5.20}{<} \quad & \sum_{i=1}^{k-1} H_k((r+1)-i) \\
= \quad & \sum_{i=2}^{k} H_k(r-i) \\
< \quad & k \cdot (k-1) \cdot \sum_{i=2}^{k} H_k(r-i) \\
\underset{5.19}{=} \quad & H_k(r)
\end{aligned}$$

□

Nun haben wir alles gezeigt, um die Abschätzung von $L(F,a,r)$ durch $H_k(r)$ vorzunehmen.

5.23 Korollar. *$L(F,a,r) \leq H_k(r)$ für alle $r \geq 0$.*

Beweis. Wir zeigen das Lemma per Induktion nach r. Für den Induktionsanfang sei zunächst $0 \leq r \leq k-1$. Dann hat der Rekursionsbaum höchstens k^r Blätter nach Lemma 2.3. Insgesamt erhalten wir: $L(F,a,r) \leq k^r \underset{(5.19)}{=} H_k(r)$ für $0 \leq r \leq k-1$. Somit gilt der Induktionsanfang.
Den Induktionsschritt von $r-1$ nach r führen wir unter der Annahme, dass die Aussage für alle $0 \leq i < r$ gilt. Sei außerdem $r \geq k$. Dann gibt es zwei Möglichkeiten: Entweder liefert SUCHE(F,a,r) direkt *wahr* und wir haben ein Blatt oder F ist (h)-beschränkt für $1 \leq h \leq k$ und es wird an einer falschen Klausel verzweigt, die das belegt. Wenn wir ein Blatt haben, gilt $L(F,a,r) = 1$. Dann gilt das Lemma, denn $H_k(r) \geq 1$ für alle r, nach Korollar 5.21.
Somit nehmen wir an, dass an einer falschen Klausel verzweigt wird. Diese Klausel belegt, dass F (h)-beschränkt ist für ein $h \in \{1,\ldots,k-1\}$. Dabei ist F nicht $(h-1)$-beschränkt. Da wir kein Blatt hatten, ist F $(k-1)$-falsch. Somit kann Lemma 5.17 angewandt werden und wir erhalten

für $L(r)$:

$$\begin{aligned} L(r) &\underset{(5.17)}{\leq} (k-1)\cdot\sum_{i=1}^{h} L(r-i) \\ &\underset{IV}{\leq} (k-1)\cdot\sum_{i=1}^{h} H_k(r-i) \\ &\underset{5.22}{<} H_k(r) \end{aligned}$$

Somit gilt das Korollar für $h \in \{1,\ldots,k-1\}$.
Es bleibt der Fall für $h = k$ zu betrachten. Dazu wenden wir Korollar 5.18 an und erhalten:

$$\begin{aligned} L(r) &\underset{5.18}{\leq} k\cdot(k-1)\cdot\sum_{i=2}^{k} L(r-i) \\ &\underset{IV}{\leq} k\cdot(k-1)\cdot\sum_{i=2}^{k} H_k(r-i) \\ &\underset{5.19}{=} H_k(r) \end{aligned}$$

Nun haben wir die Aussage für $L(r)$ gezeigt. In Lemma 5.17 und Korollar 5.18 wird vom worst-case Fall für F und a ausgegangen und wir erhalten automatisch die Gültigkeit für $L(F,a,r)$. Somit gilt das Korollar für alle $r \geq 0$. □

5.6 Abschätzung von $L(F,a,r)$

Wir können $H_k(r)$ aus dem letzten Unterabschnitt benutzen, um eine Abschätzung über die Anzahl der Blätter des Rekursionsbaumes und somit auch über die Laufzeit der neuen Prozedur SUCHE(F,a,r) vorzunehmen, nach Korollar 5.23. Hierzu vereinfachen wir das rekursive Berechnen von $H_k(r)$ auf das Suchen einer Nullstelle in einem Polynom $p_k(x)$ k-ten Grades. Dieses Polynom wird aus $H_k(r)$ abgeleitet. Wir werden es zunächst definieren.

5.24 Definition ($p_k(r)$)**.** Sei $k \geq 3$, dann ist das Polynom $p_k(x)$ über $\mathbb{R}$ definiert durch:

$$p_k(x) = x^k - \sum_{i=2}^{k} k\cdot(k-1)\cdot x^{k-i}$$

Jetzt zeigen wir, dass dieses Polynom eine reelle Nullstelle zwischen $k-1$ und k besitzt. Anschließend werden wir uns überlegen, dass es die einzige, positive, reelle Nullstelle ist.

5.25 Lemma. *$p_k(x)$ besitzt eine reelle Nullstelle im offenen Intervall $(k-1,k)$.*

Beweis. Da $p_k(x)$ als Polynom stetig ist, genügt es zu zeigen, dass $p_k(k-1) \neq 0$, $p_k(k) \neq 0$ und ein Vorzeichenwechsel zwischen $p_k(k-1)$ und $p_k(k)$ vorliegt. Dann würde nach dem Zwischenwertsatz folgen, dass ein $c \in (k-1,k)$ existiert mit $p_k(c) = 0$.
Zunächst berechnen wir $p_k(k-1)$:

$$\begin{aligned}
p_k(k-1) &= (k-1)^k - \sum_{i=2}^{k} k \cdot (k-1) \cdot (k-1)^{k-i} \\
&= (k-1)^k - \sum_{i=2}^{k} k \cdot (k-1)^{(k+1)-i} \\
&= (k-1)^k - \sum_{i=1}^{k-1} k \cdot (k-1)^{k-i} \\
&= (k-1)^k - \left(\sum_{i=1}^{k-1} (k-1) \cdot (k-1)^{k-i} + \sum_{i=1}^{k-1} (k-1)^{k-i} \right) \\
&= (k-1)^k - \sum_{i=0}^{k-2} (k-1)^{k-i} - \sum_{i=1}^{k-1} (k-1)^{k-i} \\
&= -\sum_{i=1}^{k-2} (k-1)^{k-i} - \sum_{i=1}^{k-1} (k-1)^{k-i} \\
&\underset{(k\geq 3)}{<} 0
\end{aligned}$$

Wir erhalten: $p_k(k-1) < 0$.
Nun wird $p_k(k)$ berechnet:

$$\begin{aligned}
p_k(k) &= k^k - \sum_{i=2}^{k} k \cdot (k-1) \cdot k^{k-i} \\
&= k^k - \sum_{i=1}^{k-1} (k-1) \cdot k^{k-i} \\
&= k^k - \left(\sum_{i=1}^{k-1} k \cdot k^{k-i} - \sum_{i=1}^{k-1} k^{k-i} \right) \\
&= k^k - \sum_{i=0}^{k-2} k^{k-i} + \sum_{i=1}^{k-1} k^{k-i} \\
&= \sum_{i=1}^{k-1} k^{k-i} - \sum_{i=1}^{k-2} k^{k-i} \\
&= k^{k-(k-1)} \\
&= k \\
&\underset{(k\geq 3)}{>} 0
\end{aligned}$$

Wir erhalten: $p_k(k) > 0$.

□

Nun überlegen wir uns, dass $p_k(x)$ genau eine positive, reelle Nullstelle besitzt.

5.26 Lemma. *Es existiert genau ein* $\alpha \geq 0$ *mit* $p_k(\alpha) = 0$.

Dieses Lemma ist eine direkte Folgerung aus der Vorzeichenregel von Descartes ([7] Satz 5.5, Seite 197). Wir werden uns trotzdem kurz überlegen, wieso dieser Spezialfall gilt. Auch der Beweis des Speziallfalls ist abgeleitet aus [7].

Beweis von Lemma 5.26. Die Existenz einer Nullstelle wurde bereits in Lemma 5.25 gezeigt, somit genügt es zu zeigen, dass es keine weiteren gibt.
Seien $g_k(x) = k^k$ und $h_k(x) = -\sum_{i=2}^{k} k \cdot (k-1) \cdot x^{k-i}$ Polynome über $\mathbb{R}$. Dann ist $g(x)$ streng monoton steigend und $h_k(x)$ streng monoton fallend, da sich die Vorzeichen der Koeffizienten nicht ändern. Außerdem gilt $g_k(x) + h_k(x) \xrightarrow[x \to +\infty]{} +\infty$, da der Grad von $g_k(x)$ größer ist als der von $h_k(x)$. Somit ist auch $g_k(x) + h_k(x) = p_k(x)$ streng monoton steigend für $x \geq 0$. Daher kann höchstens ein $\alpha > 0$ existieren mit $p_k(\alpha) = 0$. Nach Lemma 5.25 existiert genau ein solches α. □

Das nächste Lemma zeigt uns den Zusammenhang zwischen der Nullstelle α und $H_k(r)$. Es ermöglicht uns $H_k(r)$ mit Hilfe der Nullstelle α abzuschätzen.

5.27 Lemma. *Sei* $\alpha \geq 0$ *mit* $p_k(\alpha) = 0$. *Dann gilt* $H_k(r) < k^{k-1} \cdot \alpha^r$ *für alle* $r \geq 0$.

Beweis. Wir zeigen das Lemma per Induktion nach r. Für den Induktionsanfang betrachten wir die Fälle $0 \leq r < k$. Dann gilt:

$$H_k(r) \underset{5.19}{=} k^r \underset{(r<k)}{\leq} k^{k-1} \underset{(\alpha>1)}{<} k^{k-1} \cdot \alpha^r.$$

Da $p_k(\alpha) = 0$, gilt $0 = \alpha^k - \sum_{i=2}^{k} k \cdot (k-1) \cdot \alpha^{k-i}$ und wir erhalten $\alpha^k = \sum_{i=2}^{k} k \cdot (k-1) \cdot \alpha^{k-i}$ $(*)$.
Den Induktionsschritt von r nach $r+1$ führen wir unter der Annahme, dass die Aussage für alle

$0 \leq i < r$ gilt. Sei außerdem $r \geq k$.

$$\begin{aligned}
H_k(r+1) &= k \cdot (k-1) \cdot \sum_{i=2}^{k} H_k((r+1)-i) \\
&= k \cdot (k-1) \cdot \sum_{i=1}^{k-1} H_k(r-i) \\
&\underset{IV}{\leq} k \cdot (k-1) \cdot \sum_{i=1}^{k-1} k^{k-1} \cdot \alpha^{r-i} \\
&= k^{k-1} \cdot \alpha \cdot \sum_{i=1}^{k-1} k \cdot (k-1) \cdot \alpha^{(r-1)-i} \\
&= k^{k-1} \cdot \alpha \cdot \sum_{i=2}^{k} k \cdot (k-1) \cdot \alpha^{(r-k+k)-i} \\
&= k^{k-1} \cdot \alpha^{r-k+1} \cdot \sum_{i=2}^{k} k \cdot (k-1) \cdot \alpha^{k-i} \\
&\underset{(*)}{=} k^{k-1} \cdot \alpha^{r-k+1} \cdot \alpha^{k} \\
&= k^{k-1} \cdot \alpha^{r+1}
\end{aligned}$$

Somit gilt das Lemma. □

Jetzt können wir die Laufzeit für unseren Hauptalgorithmus mit der neuen Prozedur zur lokalen Suche SUCHE(F,a,r) in Abhängigkeit von α berechnen. Zur Erinnerung: Die Überdeckung bestand aus $B(n,\rho,d) = \text{poly}_d(n) \cdot \left(2^{(1-h(\rho))n}\right)$ Kugeln. Durch die Wahl von $d = 6$ kann nach Lemma 4.4 die Konstruktionszeit für die Kugeln vernachlässigt werden. Das Laufzeit von SUCHE(F,a,r) ist in jeder Kugel nach Lemma 5.27 durch $k^{k-1} \cdot \alpha^r$ beschränkt. Aufgrund von Lemma 4.2 setzen wir $r = n\rho = \frac{n}{k+1}$. Insgesamt ergibt sich für die Laufzeit:

$$\text{poly}_k(n) \cdot k^{k-1} \cdot \alpha^{\frac{n}{k+1}} \cdot 2^{(1-h(\rho))n} \underset{(k \text{ fest})}{=} \text{poly}_k(n) \cdot \left(\alpha^{\frac{1}{k+1}} \cdot 2^{(1-h(\rho))}\right)^n \tag{3}$$

5.28 Bemerkung. Wir haben in 5.25 gezeigt, dass $\alpha < k$ gilt. Somit ist diese Verbesserung eine echte Beschleunigung für alle $k \geq 3$ gegenüber der alten Laufzeit

$$\text{poly}_k(n) \cdot \left(k^{\frac{1}{k+1}} \cdot 2^{(1-h(\rho))}\right)^n \underset{4.5}{=} \text{poly}_k(n) \cdot \left(2 - \frac{2}{k+1}\right)^n.$$

Um eine Laufzeitschranke für ein konkretes k zu erhalten, müssen wir die Nullstelle α des Polynoms $p_k(x)$ berechnen. Dies werden wir zunächst für $k = 3$ und $k = 4$ tun, für $k \geq 5$ erhalten wir ein Polynom mindestens fünften Grades. Hierfür existiert keine allgemeine Lösungsformel (siehe [8] Korollar 14.44, Seite 292), daher wurden diese Werte numerisch ermittelt und anschließend bis $k = 10$ tabelliert.

5.7 Errechnete Laufzeit für *k=3*

Für $k = 3$ ist die kubische Gleichung $\alpha^3 - 6 \cdot \alpha - 6 = 0$ zu lösen. Die Gleichung hat die Form $x^3 + a_1 x^2 + a_0 = 0$. Wir berechnen die Nullstelle mit Hilfe von Cardanos Formel ([8] Kapitel 2.2 Seite 15). Die reelle Lösung ergibt sich aus:

$$x_1 = \sqrt[3]{-\frac{a_0}{2} + \sqrt{\left(\frac{a_0}{2}\right)^2 + \left(\frac{a_1}{3}\right)^3}} + \sqrt[3]{-\frac{a_0}{2} - \sqrt{\left(\frac{a_0}{2}\right)^2 + \left(\frac{a_1}{3}\right)^3}}$$

Für unseren Fall gilt somit:

$$\begin{aligned} \alpha &= \sqrt[3]{3 + \sqrt{(9-8)}} + \sqrt[3]{3 - \sqrt{(9-8)}} \\ &= \sqrt[3]{4} + \sqrt[3]{2} \end{aligned}$$

Wir erhalten, dass α zwischen 2.847 und 2.848 liegt.
Nun schätzen wir die Laufzeit durch Einsetzen von $\alpha = 2.848$ in die Formel (3) ab.

$$\begin{aligned} & \text{poly}(n) \cdot \left(2.848^{0.25} \cdot 2^{(1-h(0.25))}\right)^n \\ < \; & \text{poly}(n) \cdot \left(1.2991 \cdot 2^{(1-0.811)}\right)^n \\ < \; & \text{poly}(n) \cdot 1.481^n \end{aligned}$$

Insgesamt erhalten wir einen Algorithmus, der 3-SAT in Zeit $\text{poly}(n) \cdot 1.481^n$ löst.

5.8 Errechnete Laufzeit für *k=4*

Für $k = 4$ wird die Nullstelle mit Hilfe von Ferraris Formel ([8] Kapitel 3.2 Seite 22) berechnet. Für diesen Fall erhalten wir folgendes Polynom:

$$p_4(x) = x^4 - 12x^2 - 12x - 12$$

Wir haben ein Polynom der Form:

$$x^4 + a_2 x^2 + a_1 x + a_0$$

Zunächst wird eine Lösung der Gleichung $u^3 - 12u^2 + 48u - 18 = 0$ mit Cardanos Formel berechnet. Allerdings wird mit $a_1 = 48 - \frac{144}{3}$ und $a_0 = -18 + \frac{12}{3}48 + 2\left(\frac{-12}{3}\right)^3$ gerechnet um das quadratische Glied $-12u^2$ verschwinden zu lassen. Wir erhalten $u_1 = \sqrt[3]{-46} + 4$ als Lösung. Nun können wir die Lösung x_1 für $p_4(x)$ berechnen. Hierzu dient die Formel:

$$x = \varepsilon \sqrt{\frac{u_1}{2}} + \varepsilon' \sqrt{-\frac{u_1}{2} - \frac{a_2}{2} - \frac{\varepsilon a_1}{2\sqrt{2u_1}}} + \frac{a_3}{4},$$

nach Ferrari, wobei $\varepsilon, \varepsilon' \in \{-1, 1\}$. Da wir nur an der positiven, reellen Lösung interessiert sind, setzen wir $\varepsilon = \varepsilon' = 1$. Einsetzen der Variablen liefert:

$$x_1 = \sqrt{\frac{\sqrt[3]{-46}+4}{2}} + \sqrt{-\frac{\sqrt[3]{-46}+4}{2} + 6\frac{6}{\sqrt{2\sqrt[3]{-46}+4}}}$$

Insgesamt erhalten wir $x_1 = 3,972547463\ldots < 3.973$. Jetzt können wir die Laufzeit durch Einsetzen von $\alpha = 3.973$ in die Formel (3) bestimmen.

$$\begin{aligned} & \text{poly}(n) \cdot \left(3.973^{0.2} \cdot 2^{(1-h(0.2))}\right)^n \\ < \; & \text{poly}(n) \cdot \left(1.31773 \cdot 2^{(1-0.72192)}\right)^n \\ < \; & \text{poly}(n) \cdot 1.598^n \end{aligned}$$

Für diesen Fall erhalten wir, dass die abgeschätzte Beschleunigung mit verbesserter lokaler Suche nur gering ist. Denn mit der alten lokalen Suche hätten wir für $k = 4$ die Laufzeit

$$\text{poly}(n) \cdot \left(2 - \frac{2}{5}\right)^n = \text{poly}(n) \cdot 1.6^n$$

erhalten.

5.9 Numerisch ermittelte Laufzeiten für *k=3* bis *k=10*

An dieser Stelle werden wir uns noch die Werte bis $k = 10$ anschauen. In der Tabelle 1 wurden die Nullstellen bis $k = 10$ numerisch ermittelt. Außerdem sind Zwischenwerte für die Berechnung zu sehen. In der vorletzten Spalte sehen wir die erreichte Laufzeit. Auch hier muss ein wenig abgeschätzt werden, um Gültigkeit zu erreichen. Die letzte Spalte zeigt, zum Vergleich, die Laufzeit mit der alten Prozedur aus Unterabschnitt 2.1. Wie man sehen kann, nähern sich die Werte sehr schnell an. Für $k = 6$ und $k = 10$ ist aufgrund von Approximations- und Rundungsfehlern die Beschleunigung sogar negativ. Es handelt sich um Approximations- und Rundungsfehlern, da diese negative Beschleunigung bedeutet, dass $k^r < \alpha^r$ gilt. Dies wäre allerdings ein Widerspruch zu Lemma 5.25, dieses liefert $\alpha < k$. Nun überlegen wir uns, wieso wir so schnell gegen die Laufzeiten mit der alten lokalen Suche konvergieren.
Dieses Ergebnis ist nicht weiter verwunderlich, denn in unserem Rekursionsbaum werden in der k-ten Rekursionsstufe lediglich k Aufrufe weggelassen, wie man in Abbildung 4 für den Fall $k = 3$ sehen kann. Dieses Auslassen wird bei wachsendem k in der Rekursion nach unten in die k-te Ebene verschoben. Dies bedeutet, dass bei einem Suchradius $r = k$ von den ersten k^k Aufrufen lediglich mindestens k weggelassen werden. Die Beschleunigung wird bei wachsendem k schon für relativ kleine k sehr gering. Vor allem im Vergleich zur Potenz von k, die maßgeblich an der Anzahl der Blätter des Rekursionsbaumes beteiligt ist. Hierbei ist allerdings zu beachten, dass die Tabelle 1 nur eine obere Schranke für die Laufzeit angibt, allerdings ist eine ähnliche Tendenz bei der tatsächlichen Laufzeit zu erwarten. Also eine Konvergenz von $L(F, a, r)$ gegen k^r.

k	ρ	α	$\alpha^\rho \cdot 2^{1-h(\rho)}$	Laufzeit neu	Laufzeit alt
3	$\frac{1}{4}$	2.847322102	1.480539828	$\text{poly}(n) \cdot 1.481^n$	$\text{poly}(n) \cdot 1.5^n$
4	$\frac{1}{5}$	3.972547463	1.597797743	$\text{poly}(n) \cdot 1.598^n$	$\text{poly}(n) \cdot 1.6^n$
5	$\frac{1}{6}$	4.996432866	1.666468434	$\text{poly}(n) \cdot 1.6665^n$	$\text{poly}(n) \cdot 1.667^n$
6	$\frac{1}{7}$	5.999649157	1.714271394	$\text{poly}(n) \cdot 1.7143^n$	$\text{poly}(n) \cdot 1.714^n$
7	$\frac{1}{8}$	6.999972538	1.749999141	$\text{poly}(n) \cdot 1.75^n$	$\text{poly}(n) \cdot 1.75^n$
8	$\frac{1}{9}$	7.99999822	1.777777734	$\text{poly}(n) \cdot 1.778^n$	$\text{poly}(n) \cdot 1.778^n$
9	$\frac{1}{10}$	8.999999902	1.799999997	$\text{poly}(n) \cdot 1.8^n$	$\text{poly}(n) \cdot 1.8^n$
10	$\frac{1}{11}$	9.999999995	1.818181818	$\text{poly}(n) \cdot 1.8182^n$	$\text{poly}(n) \cdot 1.818^n$

Tabelle 1: Die Laufzeiten mit der neuen Prozedur SUCHE(F, a, r) im Vergleich zur alten

Literatur

[1] E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, U. Schöning; *A deterministic* $(2-2/(k+1))^n$ *algorithm for k-*SAT *based on local search*; Theoretical Computer Science 289 (2002) 69-83

[2] F.J. MacWilliams, N.J.A. Sloane; *The Theory of Error-Correcting Codes*; Amsterdam: North-Holland, (1997)

[3] G. Cohen, I. Honkala, S. Litsyn, A. Lobstein; *Covering Codes*; Mathematical Library, vol. 54, Elsevier, Amsterdam (1997)

[4] D.S. Hochbaum (Ed.); *Approximation Algorithms for NP-hard Problems*; PWS Publishing Company, MA, (1997)

[5] U. Schöning; *A probabilistic algorithm for k-*SAT *and constraint satisfaction problems*; Proc. 40th Annual IEEE Symp. on Foundations of Computer Science, FOCS'99, (1999) S. 410–414

[6] C.H. Papadimitriou; *On selecting a satisfying truth assignment*; Proc. 32nd Annual IEEE Symp. on Foundations of Computer Science, FOCS'91, (1991), S. 163–169

[7] M. Mignotte; *Mathematics for Computer Algebra*; Springer-Verlag New York Inc. (1992)

[8] J.-P. Tignol; *Galois' Theory of Algebraic Equations*; World Scientific Publishing Co. Pte. Ltd. Singapore (2001)