Institut für Theoretische Informatik
Leibniz Universität Hannover

SAT-Algorithmen

Bachelorarbeit

Bastian Saß

Betreuer	Dr. Arne Meier
Erstprüfer	Prof. Dr. Heribert Vollmer
Zweitprüfer	Dr. Arne Meier

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum Unterschrift

Inhaltsverzeichnis

1 Einleitung

SAT-Algorithmen dienen der Berechnung des Erfüllbarkeitsproblems für aussagenlogische Formeln, des SAT-Problems (von englisch *satisfiability*). Das SAT-Problem ist ein Entscheidungsproblem, ein entsprechender Algorithmus gibt aus, ob eine aussagenlogische Formel mindestens eine erfüllende Belegung besitzt.

Das SAT-Problem gehört zur Komplexitätsklasse $\mathcal{NP}$. Dass es NP-vollständig ist, zeigte Cook mit dem nach ihm benannten Satz im Jahre 1971. Solange die Frage ungeklärt ist, ob $\mathcal{P} \neq \mathcal{NP}$ oder $\mathcal{P} = \mathcal{NP}$ gilt, ist unbekannt, ob es einen effizienten Weg gibt, das SAT-Problem zu lösen, ob also ein Algorithmus existiert, der in polynomieller Zeit arbeitet und das Problem entscheidet.

Ein trivialer Ansatz besteht im Aufstellen einer Wahrheitstabelle für eine vorliegende Formel, aus der sich dann ablesen lässt, ob eine erfüllende Belegung existiert. Der zeitliche Aufwand dafür liegt im exponentiellen Bereich, da es für n Variablen in der Formel 2^n Belegungen für die Formel gibt.

Mit verschiedenen Methoden ist es mittlerweile gelungen, zumindest die Basis der exponentiellen Laufzeit zu verkleinern, beispielsweise auf $O(1{,}308^n)$ für einen Algorithmus, der auf dem in dieser Arbeit vorgestellten RW-ALGORITHMUS basiert.

Seit über zehn Jahren finden jährlich Wettbewerbe wie SAT Competition, SAT Challenge oder SAT Race statt, bei denen Verfahren zur Lösung des SAT-Problems in Hinblick auf Geschwindigkeit und Korrektheit verglichen werden.

In dieser Arbeit werden mehrere probabilistische Algorithmen aus dem Bereich der lokalen Suche betrachtet sowie zum Vergleich ein trivialer Algorithmus. Die Algorithmen werden dabei unter Beachtung von Regeln und Rahmenbedingungen aktueller SAT-Wettbewerbe implementiert. Ziel ist dabei, ein Programm zu erstellen, das grundsätzlich zur Teilnahme an einem Wettbewerb eingereicht werden könnte.

1 Einleitung

Das Prinzip der lokalen Suche basiert darauf, den Hamming-Abstand zwischen einer Startbelegung und einer erfüllenden Belegung zu minimieren. Dazu gibt es unterschiedliche Varianten, die sich u.a. darin unterscheiden, ob Startbelegungen und die eigentliche lokale Suche jeweils deterministisch oder probabilistisch gebildet bzw. durchgeführt werden. Darüber hinaus können weitere Heuristiken den Ablauf eines Algorithmus beeinflussen.

Dabei werden Eigenschaften der Klauselstruktur der in KNF vorliegenden Formeln ausgenutzt. Im Rahmen dieser Arbeit werden die Algorithmen für Formeln in 3-KNF implementiert. Die Ergebnisse lassen sich allerdings auf alle aussagenlogischen Formeln übertragen, da sich jede Formel mit lediglich polynomiellen Aufwand in eine äquivalente Formel in 3-KNF umformen lässt.

Abschließend werden mit den implementierten Algorithmen Laufzeitvergleiche durchgeführt, wobei getrennt für erfüllbare und unerfüllbare Formeln interpolierte Laufzeiten ermittelt und mit den theoretischen Laufzeiten verglichen werden.

2 Grundlagen und Notationen

Die folgenden Grundbegriffe und Definitionen wurden soweit nicht anders angegeben [Mei13] entnommen.

Aussagenlogische Formeln

Im Rahmen dieser Arbeit werden aussagenlogische Formeln betrachtet, die in KNF, insbesondere in 3-KNF, vorliegen. Eine Formel F liegt in KNF, also in *konjunktiver Normalform*, vor, wenn sie aus einer Konjunktion von Klauseln besteht. Eine Klausel besteht aus einer Disjunktion von Literalen. Im Falle einer Formel in 3-KNF enthält jede Klausel dabei höchstens drei Literale. Literale sind negierte oder nicht-negierte Variablen.

Jede aussagenlogische Formel ist äquivalent zu einer aussagenlogischen Formel in KNF bzw. 3-KNF.

Jeder Variablen kann über eine Belegung α einer der Wahrheitsheite *wahr* bzw. 1 oder *falsch* bzw. 0 zugeordnet werden. Ein Literal wird von α erfüllt, wenn entweder einer enthaltenen nicht-negierten Variablen der Wahrheitsheit 1 zugeordnet wird oder einer enthaltenen negierten Variablen der Wahrheitsheit 0 zugeordnet wird. Eine Klausel wird von α erfüllt, wenn mindestens eines der enthaltenen Literale erfüllt wird. Eine Formel wird von α erfüllt, wenn alle Klauseln erfüllt werden.

O-Symbole

Seien $f, g : \mathbb{N} \to \mathbb{N}$. Dann ist: $f = \mathrm{O}(g)$ (oder: $f(n) = \mathrm{O}(g(n))$), falls es $c, n_0 \in \mathbb{N}$ gibt, sodass für alle $n \geq n_0$ gilt: $f(n) \leq c \cdot g(n)$.

Hamming-Abstand

In [Pir07, Abschnitt 3.4] folgendermaßen definiert:

Bei eineindeutigen Codes müssen sich die zu unterschiedlichen Wörtern zugeordneten Codewörter an mindestens einer Stelle unterscheiden. Die Hamming-Distanz (oder der Hamming-Abstand) zweier Codewörter gibt an, an wieviel Stellen sich zwei Codewörter unterscheiden. Die Ermittlung der Hamming-Distanz zweier Codewörter kann durch einen stellenweisen (bitweisen) Vergleich dieser Codewörter erfolgen. Hierzu wird eine Elementarfunktion benötigt, die bei Gleichheit den Wert 0 und im anderen Falle den Wert 1 liefert. Diese Funktion wird als Exklusiv-Oder bezeichnet und diese erhält das Symbol $\oplus$. Die Ermittlung der Hamming-Distanz d zweier Codewörter Y_i und Y_j erfolgt mittels der Funktion

$$d(Y_i, Y_j) = \sum_{k=1}^{m} (y_{k,i} \oplus y_{k,j})$$

mit

$$Y_i = y_{1,i} y_{2,i} ... y_{m,i}$$

$$Y_j = y_{1,j} y_{2,j} ... y_{m,j}$$

Dabei können Y_i, Y_j als die Bit-Folgen zweier Belegungen für eine aussagenlogische Formel F angesehen werden.

Beispiel-Formel F_{Bsp}

$F_{Bsp} = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$

Tabelle 2.1: Belegungen für die Formel F_{Bsp}

$\alpha(x_1)$	$\alpha(x_2)$	$\alpha(x_3)$	$\alpha(F_{Bsp})$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

3 Algorithmen

In diesem Kapital werden die im Rahmen dieser Arbeit implementierten Algorithmen vorgestellt. Dazu wird die Arbeitsweise jedes Algorithmus erklärt und ein Beispiel angegeben.

Neben drei Algorithmen aus dem Bereich der lokalen Suche wurde zum Vergleich zunächst auch ein trivialer Lösungsansatz implementiert.

3.1 Ein Brute Force-Algorithmus (BF-ALGORITHMUS)

Der BF-ALGORITHMUS ist ein trivialer Ansatz zur Lösung des SAT-Problems. Es werden solange mögliche Belegungen überprüft, bis eine erfüllende gefunden wird, oder der Algorithmus wird nach der Überprüfung aller möglichen Belegungen beendet.

Beschreibung

Die Anzahl der möglichen Belegungen α einer aussagenlogischen Formel F mit n Variablen ist 2^n. Die Belegungen werden nacheinander darauf überprüft, ob sie die Formel erfüllen. Trifft dies zu, gibt der Algorithmus *erfüllbar* aus. Wurden alle möglichen Belegungen eingesetzt, ohne dass eine erfüllende darunter war, ist die Ausgabe *unerfüllbar*.

Algorithmus 1 BF-ALGORITHMUS

Eingabe: Eine aussagenlogische Formel F in 3-KNF
Ausgabe: *erfüllbar*, falls es eine erfüllende Belegung für F gibt, sonst *unerfüllbar*

```
for i = 1 to 2^n do
    Setze α auf nächste Belegung α_i
    if Fα = 1 then
        return erfüllbar
    end if
end for
return unerfüllbar
```

Beispiel

Der BF-ALGORITHMUS wird auf die Formel F_{Bsp} aus Abschnitt 2 angewendet:

1. α wird auf $\alpha_1 = \{0, 0, 0\}$ gesetzt. $F_{Bsp}\alpha = 0$.
2. α wird auf $\alpha_2 = \{0, 0, 1\}$ gesetzt. $F_{Bsp}\alpha = 0$.
3. α wird auf $\alpha_3 = \{0, 1, 0\}$ gesetzt. $F_{Bsp}\alpha = 1$.
4. Ausgabe *erfüllbar*. Der Algorithmus wird beendet.

3.2 Lokale Suche und Hamming-Kugeln

Schöning und Torán beschreiben in [Sch11, Kapitel 5] die lokale Suche als ein grundsätzlich sehr einfaches Verfahren, das in Kombination mit fortgeschrittenen Heuristiken erstaunlich effizient sein kann.

Im Rahmen dieser Arbeit wurden drei probabilistische Algorithmen implementiert, die nach dem Prinzip der lokalen Suche arbeiten.

Das Verfahren basiert darauf, dass eine initiale Belegung α für die untersuchte Formel F gewählt wird. Bei den implementierten Algorithmen erfolgt dies (pseudo-)zufällig. Dass die Belegung α bereits F erfüllt, ist unwahrscheinlich.

Soweit die Belegung die Formel nicht erfüllt, existieren unerfüllte Klauseln. Unter Anwendung des Zufalls oder einer bestimmten Strategie werden nun die Wahrheitswerte von Variablen in diesen unerfüllten Klauseln geändert, um Maße wie eben die Anzahl der unerfüllten Klauseln oder den Hamming-Abstand zu einer erfüllenden Belegung α_0 zu minimieren.

Findet sich eine erfüllende Belegung, haben beide Maße den Wert 0. Es gibt allerdings keine gute Korrelation zwischen der Anzahl der unerfüllten Klauseln und dem Hamming-Abstand. Während sich die Anzahl der unerfüllten Klauseln direkt abzählen lässt, kann über den Hamming-Abstand nur indirekt etwas ausgesagt werden.

Bei Schöning und Torán wird auch die Frage diskutiert, inwieweit probabilistische Algorithmen, die auf der lokalen Suche basieren, als *vollständig* angesehen werden können. Deterministische Algorithmen heißen *vollständig*, da sie nach dem Ablauf entweder eine korrekte Ausgabe *erfüllbar* oder *unerfüllbar* liefern.

Bei einem probabilistischen Algorithmus kann nach einem erfolglosen Ablauf nicht absolut unterschieden werden, ob die untersuchte Formel tatsächlich unerfüllbar ist oder ob doch mindestens eine erfüllende Belegung existiert, diese mit der angewandten Heuristik aber nicht gefunden werden konnte. Allgemein wird ein solcher Algorithmus als *unvollständig* bezeichnet.

Schöning und Torán vertreten hier allerdings die Ansicht, dass auch ein probabilistischer Algorithmus als vollständig angesehen werden kann, wenn eine theoretische Analyse existiert, die quasi als Teil des Algorithmus selbst Auskunft darüber gibt, mit welcher Wahrscheinlichkeit erfüllende Belegungen übersehen werden.

Für die in den folgenden Abschnitten vorgestellten Algorithmen LS-ALGORITHMUS und RW-ALGORITHMUS geben Schöning und Torán solche theoretischen Analysen an. Für den vom LS-ALGORITHMUS abgeleiteten LF-ALGORITHMUS gibt es streng genommen keine separate theoretische Analyse, allerdings erfolgt bei der Analyse des MOSER-SCHEDER-ALGORITHMUS, der im Rahmen dieser Arbeit nicht implementiert wurde, eine Betrachtung eines gleichartigen Vorgehens. Von daher wird der LF-ALGORITHMUS hier auch als vollständig nach der Sichtweise von Schöning und Torán angesehen.

Die Frage der Voll- bzw. Unvollständigkeit eines Algorithmus besitzt bei der Teilnahme an SAT-Wettbewerben auch eine praktische Bedeutung, worauf in Abschnitt 4.1 näher eingegangen wird.

3.2.1 Deterministische lokale Suche mit zufälliger Anfangsbelegung (LS-ALGORITHMUS)

Der LS-ALGORITHMUS durchsucht deterministisch eine bestimmte Anzahl von Hamming-Kugeln um unterschiedliche Startbelegungen herum. Die theoretische Worst-Case-Laufzeit liegt in $O(1{,}5^n)$.

Beschreibung

Seien $p = \delta \cdot n$ der Hamming-Radius und t die Anzahl unabhängiger Zufallsauswahlen einer Startbelegung α. Zur Bestimmung der Werte von p und t siehe Abschnitt 4.4.

Die deterministische Prozedur localSearch überprüft, ob es zu einer gegebenen Formel F und einer ebenfalls gegebenen Startbelegung α eine erfüllende Belegung α_0 gibt, die maximal den Hamming-Abstand $d(\alpha, \alpha_0)$ von $p = \delta \cdot n$ mit $\delta \leq \frac{1}{2}$ hat.

Vorausgesetzt, eine erfüllende Belegung α_0 existiert: Wenn eine Belegung α die Formel F nicht erfüllt, so gibt es mindestens eine Klausel $K \in F$, die von α nicht erfüllt wird. F liegt in 3-KNF vor, damit enthält K drei Literale. Die Belegungen dieser drei Literale werden nacheinander geändert, so dass die Klausel danach jeweils durch die neue Belegung erfüllt wird. In mindestens einem dieser Fälle ergibt sich so eine Verkleinerung des Hamming-Abstands zu α_0 um den Wert 1.

Formal lässt sich eine Hamming-Kugel H um eine Belegung α mit Radius p als Menge aller möglichen Belegungungen fassen, für die

$$H_p(\alpha) = \{\alpha' | d(\alpha, \alpha') \leq p\}$$

gilt.

Beispiel

Der LS-ALGORITHMUS wird auf die Formel F_{Bsp} aus Abschnitt 2 angewendet:

1. Innerhalb der Schleife wird eine erste zufällige Belegung $\alpha = \{0, 0, 0\}$ gewählt.
2. Die Prozedur localSearch(F_{Bsp}, α, 1) wird aufgerufen.
3. $F_{Bsp}\alpha = 0$ und $p > 0$.
4. Es wird die Klausel $K = \{x_1, x_2, x_3\}$ mit $K\alpha = 0$ gewählt.
5. Die Prozedur localSearch(F_{Bsp}, $\alpha[x_1 = 1] = \{1, 0, 0\}$, 0) wird aufgerufen.
6. $F_{Bsp}\alpha[x_1 = 1] = 1$, die Prozedur gibt eine logische Eins zurück, die rekursiv an den Algorithmus zurückgegeben wird.
7. Ausgabe *erfüllbar*. Der Algorithmus wird beendet.

Algorithmus 2 LS-ALGORITHMUS

Eingabe: Eine aussagenlogische Formel F in 3-KNF
Ausgabe: *erfüllbar*, falls es eine erfüllende Belegung für F gibt, sonst *unerfüllbar*

```
for i = 1 to t do
    Wähle eine zufällige Belegung α
    if localSearch(F, α, p) then
        return erfüllbar
    end if
end for
return unerfüllbar
```

Prozedur 3 localSearch

```
if Fα = 1 then
  return 1
end if
if p = 0 then
  return 0
end if
Sei K ∈ F eine Klausel mit Kα = 0, K = {u_1, u_2, u_3}
for i = 1 to 3 do
  if localSearch(F, α[u_i = 1], p − 1) then
    return 1
  end if
end for
return 0
```

3.2.2 Deterministische lokale Suche mit zufälliger Anfangsbelegung ohne Back-Flips (LF-ALGORITHMUS)

Der LF-ALGORITHMUS basiert auf dem LS-ALGORITHMUS (siehe Unterabschnitt 3.2.1). Im Gegensatz zu diesem werden dabei Bit-Flips innerhalb der Rekursion „eingefroren“. Somit wird ausgeschlossen, dass die Belegung einer bereits „geflippten“ Variable in einer tieferen Rekursionsstufe wieder verändert wird.

Beschreibung

Der Ablauf ist zunächst gleich dem des LS-ALGORITHMUS. Die Prozedur localSearch wird durch eine erweiterte Prozedur localSearchAndFreeze ersetzt. Innerhalb der Suche in einer Hamming-Kugel merkt sich der Algorithmus, für welche Variablen der Wahrheitswert in der Belegung bereits einmal geändert wurde. Trifft der Algorithmus in der Prozedur localSearchAndFreeze dann in einer nichterfüllten Klausel auf ein Literal, das eine Variable enthält, deren Belegung schon einmal geändert wurde, so wird dieses Literal übersprungen und die Rekursion an dieser Stelle nicht fortgesetzt; ein Back-Flip der Variablen wird also verhindert.

Beipiel

Der LF-ALGORITHMUS wird auf die Formel F_{Bsp} aus Abschnitt 2 angewendet:

1. Innerhalb der Schleife wird eine erste zufällige Belegung $\alpha = \{1, 1, 0\}$ gewählt.
2. Die Prozedur localSearchAndFreeze(F_{Bsp}, α, 1) wird aufgerufen.
3. $F_{Bsp}\alpha = 0$ und $p > 0$.
4. Es wird die Klausel $K = \{\neg x_1, \neg x_2, x_3\}$ mit $K\alpha = 0$ gewählt.
5. Die Prozedur localSearchAndFreeze($F_{Bsp}\{\neg x_1 = 1\}$, $\alpha[\neg x_1 = 1] = \{0, 1, 0\}$, 0) wird aufgerufen.
6. $F_{Bsp}\{\neg x_1 = 1\}\alpha[\neg x_1 = 1] = 1$, die Prozedur gibt eine logische Eins zurück, die rekursiv an den Algorithmus zurückgegeben wird.
7. Ausgabe *erfüllbar*. Der Algorithmus wird beendet.

Algorithmus 4 LF-ALGORITHMUS

Eingabe: Eine aussagenlogische Formel F in 3-KNF
Ausgabe: *erfüllbar*, falls es eine erfüllende Belegung für F gibt, sonst *unerfüllbar*

```
for i = 1 to t do
  Wähle eine zufällige Belegung α
  if localSearchAndFreeze(F, α, p) then
    return erfüllbar
  end if
end for
return unerfüllbar
```

Prozedur 5 localSearchAndFreeze

```
if Fα = 1 then
  return 1
end if
if p = 0 then
  return 0
end if
Sei K ∈ F eine Klausel mit Kα = 0, K = {u_1, u_2, u_3}
for i = 1 to 3 do
  Falls die Belegung der Variablen im Literal u_i innerhalb der aktuell durchsuchten
  Hamming-Kugel bereits verändert wurde, überspringe diesen Schleifendurchlauf und
  prüfe ggf. das nächste Literal.
  if localSearchAndFreeze(F{u_i = 1}, α[u_i = 1], p − 1) then
    return 1
  end if
end for
return 0
```

3.2.3 Ein Random Walk-Algorithmus (RW-ALGORITHMUS)

Der RW-ALGORITHMUS ersetzt die deterministische lokale Suche, wie sie beim LS-ALGORITHMUS durchgeführt wird, durch eine stochastische Suche. Die theoretische Worst-Case-Laufzeit liegt in $O((\frac{4}{3})^n)$.

Beschreibung

Seien $p = n$ der Hamming-Radius und t die Anzahl unabhängiger Zufallsauswahlen einer Startbelegung α. Zur Bestimmung des Wertes von t siehe Abschnitt 4.6.

Angenommen, es existiert eine erfüllende Belegung α_0: Dann gibt es in jeder Klausel mindestens ein Literal, das durch diese Belegung den Wahrheitswert 1 annimmt. Betrachtet wird nun in jeder durch die initiale Belegung nicht erfüllten Klausel ein solches Literal. Der Algorithmus wählt zufällig eines der drei Literale aus und ändert seine Belegung. Die Wahrscheinlichkeit, dass die Belegung des zuvor betrachteten Literals geändert wird, liegt bei $\frac{1}{3}$. Geschieht dies, verringert sich der Hamming-Abstand zwischen der aktuellen Belegung α und der erfüllenden Belegung α_0 um 1.

Algorithmus 6 RW-ALGORITHMUS

Eingabe: Eine aussagenlogische Formel F in 3-KNF
Ausgabe: *erfüllbar*, falls es eine erfüllende Belegung für F gibt, sonst *unerfüllbar*

```
for i = 1 to t do
  Wähle eine zufällige Anfangsbelegung α
  for j = 1 to n do
    if Fα = 1 then
      return erfüllbar
    end if
    Wähle ein K ∈ F mit Kα = 0, K = {u_1, u_2, u_3}
    Wähle zufällig l ∈ {1, 2, 3}
    α ← α[u_l = 1]
  end for
end for
return unerfüllbar
```

Beipiel

Der RW-ALGORITHMUS wird auf die Formel F_{Bsp} aus Abschnitt 2 angewendet:

1. Innerhalb der äußeren Schleife wird eine erste zufällige Belegung $\alpha = \{0, 1, 1\}$ gewählt.
2. Beginn des ersten Durchlaufs der inneren Schleife.
3. $F_{Bsp}\alpha = 0$.
4. Es wird die Klausel $K = \{x_1, \neg x_2, \neg x_3\}$ mit $K\alpha = 0$ gewählt.
5. Es wird zufällig $l = 2$ gewählt.
6. α wird auf $\alpha[u_l = \neg x_2 = 1] = \{0, 0, 1\}$ gesetzt.
7. Beginn des zweiten Durchlaufs der inneren Schleife.
8. $F_{Bsp}\alpha = 0$.
9. Es wird die Klausel $K = \{x_1, x_2, \neg x_3\}$ mit $K\alpha = 0$ gewählt.
10. Es wird zufällig $l = 1$ gewählt.
11. α wird auf $\alpha[u_l = x_1 = 1] = \{1, 0, 1\}$ gesetzt.
12. Beginn des dritten Durchlaufs der inneren Schleife.

13. $F_{Bsp}\alpha = 1$.

14. Ausgabe *erfüllbar*. Der Algorithmus wird beendet.

4 Implementierung

In diesem Kapital wird die Implementierung des Programms LSSOLVE vorgestellt. Zunächst werden dazu eine Reihe von Regeln und Rahmenbedingungen aktueller SAT-Wettbewerbe dargestellt, die Beachtung gefunden haben. Danach wird erläutert, wie Hauptprogramm und Datenmodell in Java Version 6 implementiert wurden und wie weitere Algorithmen hinzugefügt werden können. Abschließend erfolgen Hinweise zur Implementierung der einzelnen Algorithmen.

4.1 SAT-Wettbewerbe

Die Implementierung von LSSOLVE orientiert sich in unterschiedlichen Aspekten an den Vorgaben und Empfehlungen von SAT-Wettbewerben wie der SAT Competition 2011 [SATa], dem SAT Challenge 2012 [SATc] oder der SAT Competition 2013 [SATd].

Reproduzierbarkeit

Die Regeln der SAT Competition 2011 [SATb] schreiben in Abschnitt 6.2 vor, dass zur Initialisierung von Pseudozufallszahlengeneratoren ein vorgegebener Wert verwendet werden muss. Dies soll dazu dienen, falls nötig, Testläufe unter den selben Bedingungen wiederholen zu können.

Falsche Antworten

Nach den Regeln der SAT Competition 2013 [SATe] werden Programme im Wettbewerb disqualifiziert, wenn sie für eine Formel *erfüllbar* zusammen mit einer nicht-erfüllenden Belegung ausgeben oder wenn sie für eine eigentlich erfüllbare Formel *unerfüllbar* ausgeben.

Insbesondere letzterer Aspekt stellt im Bezug auf die im Rahmen dieser Arbeit untersuchten Algorithmen zur lokalen Suche eine Herausforderung dar. Zwar lässt sich die Wahrscheinlichkeit minimieren, dass eine erfüllende Belegung übersehen und damit eine Formel fälschlicherweise als unerfüllbar erkannt wird, ausschließen lässt sich die Möglichkeit aber nicht. Dieser Punkt wird bei den Hinweisen auf die Implementierungen der einzelnen Algorithmen jeweils noch einmal aufgegriffen werden.

Programmaufruf

Zur Teilnahme an den SAT-Wettbewerben muss eine Signatur zum Aufruf des Programms angegeben werden, die später mit Werten aus der Testumgebung gefüllt wird. Für das im Rahmen dieser Arbeit implementierte Programm LSSOLVE würde der Aufruf mit Platzhaltern so lauten:
`java -server -jar DIR/LSSolve.jar -solver RW -randomseed RANDOMSEED -benchname BENCHNAME`
Bei der Option `-solver` könnten dabei auch andere implementierte Algorithmen gewählt werden.

Eingaben im DIMACS-Format

DIMACS ist ein Format zur Speicherung von aussagenlogischen Formeln in KNF in Textdateien. Dafür gelten diese Regeln:

- Am Anfang der Datei können Kommentare stehen, die Zeilen beginnen mit dem Zeichen `c`.
- Nach möglichen Kommentaren folgt eine Zeile mit dem Inhalt `p <nbvar> <nbclauses>`. `<nbvar>` ist dabei eine ganze positive Zahl, die angibt, wieviele Variablen in der Formel enthalten sind. Jede Variable zwischen 1 und `<nbvar>` kommt dabei in der Formel mindestens einmal vor. Der ganzzahlige positive Wert von `<nbclauses>` gibt an, aus wievielen Klauseln die Formel besteht.
- Anschließend folgen *nbclauses* viele Zeilen für die einzelnen Klauseln. Eine Klausel wird dabei durch eine Reihe von Zahlen repräsentiert, die zwischen $-nbvar$ und $nbvar$ liegen und nicht Null sind. Die einzelnen Zahlen stehen für die Literale. Ist

eine Zahl positiv enthält das Literal die entsprechende Variable, ist eine Zahl negativ entsprechend die negierte Variable. Beendet wird jede Zeile mit dem Zeichen `0`.

Ausgaben des Programms

In den Regeln der SAT Competition 2011 werden folgende Vorgaben für drei mögliche Arten von Ausgaben gemacht:

- Zu jeder Zeit während der Ausführung des Programms kann eine Kommentarzeile mit beliebigem Inhalt ausgegeben werden. Eine solche Kommentarzeile wird durch das Zeichen `c` gefolgt von einem Leerzeichen eingeleitet.
- Das Programm muss genau eine Ergebniszeile ausgeben, die `s SATISFIABLE`, `s UNSATISFIABLE` oder `s UNKNOWN` sein kann. Sobald eine Formel als erfüllbar erkannt wurde, erfolgt die Ausgabe von `s SATISFIABLE`. Steht fest, dass die Formel nicht erfüllbar ist, wird `s UNSATISFIABLE` ausgegeben. Ist keine Aussage möglich, bleibt die Ausgabe der Zeile `s UNKNOWN`.
- Sofern als Ergebniszeile `s SATISFIABLE` ausgegeben wurde, muss mindestens eine Zeile folgen, in der die gefundene erfüllende Belegung ausgegeben wird. Eine solche Belegungszeile wird durch das Zeichen `v` gefolgt von einem Leerzeichen eingeleitet. Die Aufteilung der Ausgabe der erfüllenden Belegung auf mehrere Belegungszeilen ist dabei möglich. Ist eine Variable in der erfüllenden Belegung mit wahr belegt, erfolgt die Ausgabe ihres Bezeichners, ist sie mit falsch belegt, wird dieser negiert. Beendet wird die letzte Belegungszeile mit dem Zeichen `0`.

4.2 Hauptprogramm und Datenmodell

Optionen beim Aufruf

Die folgenden Optionen können bzw. müssen LSSOLVE beim Aufruf übergeben werden:

-help Gibt die verfügbaren Optionen aus und beendet das Programm.

-version Gibt die aktuelle Versionsnummer aus und beendet das Programm.

-benchname <file> Übergibt eine Datei im DIMACS-Format, Angabe obligatorisch.

-solver <XX> Angabe des zu verwendenen Algorithmus, Angabe obligatorisch.

-randomseed <unsignedint> Angabe eines Initialisierungswertes für den Pseudozufallszahlengenerator im Bereich von 1 bis 4.294.967.295, Angabe optional.

-constant <unsignedint> Angabe einer Konstante im Bereich von 0 bis 2.147.483.647, Angabe optional. Die Bedeutung hängt vom ausgewählten Algorithmus ab, der diesen Wert auch ignorieren oder weiter einschränken kann.

Zur Verarbeitung der beim Programmaufruf übergebenen Optionen findet die Apache Commons CLI-Bibliothek Verwendung [Com].

Hauptprogramm und Hilfsklasse

Das Hauptprogramm stellt den implementierten Algorithmen folgende Daten über statische Variablen zur Verfügung:

randomSeed Ein Wert vom Typ long, mit dem alle verwendeten Pseudozufallsgeneratoren initialisiert werden sollen. Soweit dem Programm kein anderer Wert übergeben wurde, ist der Standardwert 17323698L.

constantParam Ein Wert vom Typ int, der als Konstante an den ausgewählten Algorithmus übergeben werden kann.

formula Ein Objekt vom Typ Formula, das die aussagenlogische Formel enthält.

nbvar Ein Wert vom Typ int, der die Anzahl der Variablen in der aussagenlogischen Formel enthält.

nbclauses Ein Wert vom Typ int, der die Anzahl der Klauseln in der aussagenlogischen Formel enthält.

ccnf Ein Wert vom Typ int, der für die in k-KNF vorliegende aussagenlogischen Formel den Wert von k enthält und auf den Wert 3 festgelegt ist.

Desweiteren findet im Hauptprogramm die Auswertung der beim Aufruf übergebenen Optionen statt.

In der Hilfsklasse Util steht z.B. die Methode zur Berechnung der Laufzeit zur Verfügung.

Datenmodell

Das Datenmodell von LSSOLVE besteht aus den drei Klassen Formula, Clause und Assignment.

Ein Objekt der Klasse Formula dient als Container für die Klauselmenge. Objekte vom Typ Clause werden in einem Array gespeichert. Die Klasse Formula besitzt zudem eine Methode zum Laden und Parsen von Textdateien im DIMACS-Format.

Objekte vom Typ Clause repräsentieren die Klauseln, aus denen eine Formel besteht. Die Literale einer Klausel werden in einer HashMap gespeichert. Dabei dient der Bezeichner der Variable, eine ganze Zahl, als Schlüssel. Als Wert wird mittels eines Boolean-Objekts jeweils gespeichert, ob das Literal die Variable negiert oder nicht-negiert enthält.

Objekte vom Typ Assignment speichern Belegungen in einem Array des primitiven Typs boolean. Der Index des Arrays steht dabei mit den Bezeichern der Variablen in der zugehörigen aussagenlogischen Formel in Beziehung. Assignment besitzt zudem einen Pseudozufallszahlengenerator und kann über diesen Belegungen generieren. Mit verschiedenen Methoden kann überprüft werden, ob eine Belegung eine Formel erfüllt. Dabei können bei Bedarf eine unerfüllte Klausel oder deren Index innerhalb des Arrays im Formula-Objekt zurückgegeben werden.

Hinzufügen weiterer Algorithmen

LSSOLVE können weitere Algorithmen hinzugefügt werden. Für einen neuen Algorithmus ist ein Kürzel bestehend aus Klein- und Großbuchstaben sowie Ziffern zu wählen. Im Folgenden wird das Kürzel ABC als Beispiel verwendet.

Für den neuen Algorithmus ist eine eigene Klasse anzulegen, die mit dem gewählten Kürzel benannt wird. Im Beispiel also SolverABC.java.

Im Hauptprogramm LSSolve.java ist in der if-Kaskade bei der Auswertung der solver-Option ein Fall für den neuen ABC-Algorithmus hinzuzufügen.

Innerhalb der Klasse SolverABC.java ist eine statische Methode solver() anzulegen. Diese bzw. weitere optionale Methoden innerhalb der neuen Klasse beenden den Programmablauf mittels der exit-Methode. Zuvor erfolgt eine der drei möglichen Ausgaben `s SATISFIABLE`, `s UNSATISFIABLE` oder `s UNKNOWN` sowie ggf. die Ausgabe einer erfüllenden Belegung.

Beschränkungen

Wenngleich LSSOLVE in der Lage ist, beliebige gültige Formeln im DIMACS-Format einzulesen und zu verarbeiten, sind die Algorithmen an verschiedenen Stellen auf Formeln in 3-KNF abgestimmt. Für eine Anwendung des Programms auf andere Formeln als solche in 3-KNF lassen sich sinnvolle Resultate daher nicht voraussagen.

4.3 BF-ALGORITHMUS

Für den BF-ALGORITHMUS gibt es keine besonderen Hinweise zur Implementierung. Beispielhaft für alle implementierten Algorithmen kann erwähnt werden, dass zur Berechnung der Anzahl der Schleifendurchläufe bzw. Hamming-Kugeln Objekte vom Typ BigInteger verwendet wurden.

4.4 LS-ALGORITHMUS

Berechnung des Radius p

Schöning und Torán zeigen in [Sch11], dass der optimale Wert für δ für die Berechnung des Radius der zu durchsuchenden Hamming-Kugeln $p = \delta \cdot n$ bei $\frac{1}{4}$ liegt, mit n der Anzahl der Variablen in einer aussagenlogischen Formel F.

Soweit die Multiplikation von δ und n kein ganzzahliges Ergebnis liefert, wird der Wert von p auf die nächste ganze Zahl aufgerundet.

Anzahl der Hamming-Kugeln t

Schöning und Torán zeigen in [Sch11] über einen Beweis mittels Zufallsvariablen, dass die Wahrscheinlichkeit, dass der LS-ALGORITHMUS eine erfüllende Belegung übersieht, bei e^{-c} für eine frei wählbare Konstante c liegt, wenn die Anzahl t der zu durchsuchenden Hamming-Kugeln über

$$t = \frac{c \cdot 2^n}{\sum_{i=1}^{\delta n} \binom{n}{i}}$$

berechnet wird.

Die Formel eignet sich in dieser Form noch nicht zur Implementierung, zunächst muss die Summe abgeschätzt und dann der resultierende Bruch gekürzt werden.

Als Abschätzung der Summe nach oben geben Schöning und Torán

$$\sum_{i=1}^{\delta n} \binom{n}{i} \leq 2^{h(\delta) \cdot n}$$

an und zeigen letztlich, dass

$$\sum_{i=1}^{\delta n} \binom{n}{i} \overset{poly}{=} 2^{h(\delta) \cdot n}$$

gilt, wobei das Zeichen $\overset{poly}{=}$ bedeutet, dass unter Nichtbeachtung polynomieller Faktoren das exponentielle Verhalten der beiden Funktionen gleich ist.

$h(\delta) = -\delta \cdot \log_2(\delta) - (1-\delta) \cdot \log_2(1-\delta)$ ist dabei die binäre Entropiefunktion nach Shannon.

Für $\delta = \frac{1}{4}$ ergibt sich $h(\frac{1}{4}) \approx 0,811278$, somit $2^{h(\frac{1}{4})\cdot n} \approx 2^{0,811278 \cdot n}$. Eingesetzt in die Formel für t erhalten wir

$$t \overset{poly}{\approx} \frac{c \cdot 2^n}{2^{0,811278 \cdot n}} = c \cdot 2^{0,188722 \cdot n} \approx c \cdot 1,39754^n$$

Diese Abschätzung der Summe nach oben ist allerdings nicht ausreichend dafür, dass der implementierte Algorithmus für eine frei gewählte Konstante c eine erfüllende Belegung nur mit einer Wahrscheinlichkeit e^{-c} übersieht.

Für die Abschätzung der Summe nach unten geben Schöning und Torán zunächst

$$\sum_{i=1}^{\delta n} \binom{n}{i} \geq \frac{1}{n+1} \cdot 2^{h(\delta)\cdot n}$$

und dann mit Hilfe der Stirling'schen Formel

$$\sum_{i=1}^{\delta n} \binom{n}{i} \geq \frac{1}{\sqrt{8n\delta(1-\delta)}} \cdot 2^{h(\delta)\cdot n}$$

an, was für $\delta = \frac{1}{4}$

$$\frac{1}{\sqrt{\frac{3}{2} \cdot n}} \cdot 2^{h(\frac{1}{4})\cdot n}$$

ergibt.

Als mögliche polynomielle Faktoren zur Berechnung von t ergeben sich somit $\sqrt{\frac{3}{2} \cdot n}$ und $(n+1)$.

Um die praktische Eignung der polynomiellen Faktoren zu überprüfen, wird mit einer bestimmten Menge an erfüllbaren Instanzen ein Wert für die Konstante c gesucht, so dass

der Algorithmus für alle Instanzen eine erfüllende Belegung findet und bis zu diesem Wert von c der Anteil der übersehenen erfüllenden Belegungen unterhalb von e^{-c} liegt. Verwendung finden dabei die in Kapitel 5 beschriebenen erfüllbaren Formeln mit bis zu 20 Variablen. Der Pseudozufallszahlengenerator wird jeweils mit dem Standardwert initialisiert (siehe dazu Abschnitt 4.1).

Durch diesen Test zeigt sich, dass sowohl $\sqrt{\frac{3}{2} \cdot n}$ als auch $(n+1)$ zu kleine polynomielle Faktoren sind, und eine großzügigere Abschätzung der Summe nach unten notwendig ist.

Der polynomielle Faktor n^2 erweist sich als zu groß, $(n \cdot \ln n)$ sowie $(n \cdot \sqrt{n})$ sind wiederum zu klein. Weitere Versuche mit den Faktoren $(0{,}5 \cdot (n^2 + n \cdot \sqrt{n}))$ und $(0{,}7 \cdot (n^2 + n \cdot \sqrt{n}))$ führen letztlich zu einem polynomiellen Faktor von $(0{,}6 \cdot (n^2 + n \cdot \sqrt{n}))$, so dass für einen Wert von $c = 7$ für alle getesteten Formeln eine erfüllende Belegung gefunden wird.

Für die Testläufe im Rahmen der Laufzeitvergleiche (siehe Abschnitt 5.2) wurde also die Formel

$$t = 7 \cdot (0{,}6 \cdot (n^2 + n \cdot \sqrt{n})) \cdot 1{,}39754^n$$

zur Berechnung der Anzahl der Hamming-Kugeln t verwendet.

Weitere Untersuchungen zur Anzahl der Hamming-Kugeln

Bei der vorhergehenden Suche nach einem geeigneten polynomiellen Faktor und einer Konstanten zur Berechnung der Anzahl der zu durchsuchenden Hamming-Kugeln wurde der Pseudozufallszahlengenerator jeweils mit dem innerhalb von LSSOLVE vorgesehenen Standardwert initialisiert. Will man auch für schwerer zu lösende Formeln (siehe [Sch11, Abschnitt 5.7]) oder ungünstigere Initialisierungswerte für den Pseudozufallszahlengenerator möglichst erreichen, dass der Algorithmus keine erfüllende Belegung übersieht und für eine eigentlich erfüllbare Formel *unerfüllbar* ausgibt, ist ggf. ein größerer polynomieller Faktor oder ein größerer Wert für die Konstante c zu wählen.

Der zuvor beschriebene Versuch ist fünfzehnmal wiederholt worden, wobei jeweils ein anderer, zufällig gewählter Wert zur Initialisierung des Zufallszahlengenerators verwendet

wurde. Hierbei ergab sich, dass erst für einen Wert von $c = 14$ keine erfüllende Belegung mehr übersehen wurde. Der Anteil der übersehenen Belegungen lag teilweise ein bis zwei Größenordnungen über dem Wert von e^{-c}.

Dies lässt darauf schließen, dass der polynomielle Faktor größer als $(0{,}6 \cdot (n^2 + n \cdot \sqrt{n}))$ gewählt werden sollte.

4.5 LF-ALGORITHMUS

Zur Implementierung des LF-ALGORITHMUS gelten grundsätzlich die gleichen Anmerkungen wie beim LS-ALGORITHMUS (siehe Abschnitt 4.4), was die Berechnung des Radius und die Anzahl der zu durchsuchenden Hamming-Kugeln anbetrifft. Für die Konstante c lässt sich ein Wert von 6 für den polynomiellen Faktor $(0{,}6 \cdot (n^2 + n \cdot \sqrt{n}))$ ermitteln. Zur Vergleichbarkeit der Laufzeiten wurde der für den LS-ALGORITHMUS ermittelte Wert von $c = 7$ verwendet.

Für die Testläufe im Rahmen der Laufzeitvergleiche (siehe Abschnitt 5.3) wurde also ebenfalls die Formel

$$t = 7 \cdot (0{,}6 \cdot (n^2 + n \cdot \sqrt{n})) \cdot 1{,}39754^n$$

zur Berechnung der Anzahl der Hamming-Kugeln t verwendet.

Bei den Testreihen mit unterschiedlichen Initialisierungen des Pseudozufallszahlengenerators erweisen sich die Ergebnisse beim LF-ALGORITHMUS als deutlich stabiler als beim LS-ALGORITHMUS. Der Wert für die Konstante c steigt lediglich auf 8, bis dahin liegt der Anteil der übersehenen erfüllenden Belegungen unter dem Wert von e^{-c}.

4.6 RW-ALGORITHMUS

Anzahl der Hamming-Kugeln t

Schöning und Torán zeigen in [Sch11, S. 105] wieder über einen Beweis mit Wahrscheinlichkeitsvariablen, dass für eine Anzahl an Wiederholungsläufen des Algorithmus bzw. zu durchsuchenden Hamming-Kugeln von $t = c \cdot (\frac{4}{3})^n$ die Wahrscheinlichkeit bei e^{-c} liegt, dass eine erfüllende Belegung übersehen wird.

Innerhalb des Beweises wird wieder mit dem Zeichen $\overset{poly}{=}$ operiert, und damit polynomielle Faktoren bei einer gleichen Entwicklung im exponentiellen Verhalten ignoriert. Bei einer Untersuchung analog zu den in Abschnitt 4.4 beschriebenen Tests zeigt sich, dass bereits die Wahl von $c = 6$ ausreicht, für alle Formeln eine erfüllende Belegung zu finden. Ein zusätzlicher polynomieller Faktor ist nicht notwendig.

Für die Testläufe im Rahmen der Laufzeitvergleiche (siehe Abschnitt 5.4) wurde also die Formel

$$t = 6 \cdot \left(\frac{4}{3}\right)^n$$

zur Berechnung der Anzahl der Hamming-Kugeln t verwendet.

Weitere Untersuchungen zur Anzahl der Hamming-Kugeln

Wie beim LS- und beim LF-ALGORITHMUS wurden weitere Untersuchungen angestellt, wie sich der Algorithmus bei unterschiedlichen Werten zur Initialisierung des Pseudozufallszahlengenerators im Bezug darauf verhält, bis zur Wahl welchen Wertes für die Konstante c wieviele erfüllende Belegungen übersehen werden (siehe auch Abschnitt 4.4).

Bei diesen Untersuchungen ergibt sich, dass erst bei einer Wahl von $c = 11$ keine erfüllenden Belegungen mehr übersehen werden. Bis dahin liegt der Anteil der übersehenen Belegungen maximal in der Größenordnung von e^{-c}.

5 Laufzeitvergleich

In diesem Kapital werden die Ergebnisse der Laufzeitvergleiche der implementierten Algorithmen dargestellt. Zunächst wird dazu die Testumgebung, der Ablauf der Tests sowie die Auswertung der gewonnenen Daten erläutert. Danach folgen Übersichten für die einzelnen Algorithmen.

Umgebung für die Laufzeitvergleiche

Alle Testläufe zum Laufzeitvergleich wurden auf einem Computer vom Typ SGI Altix XE 250 (kurz XE) mit 48 Knoten (2 Intel Xeon Harpertown Prozessoren (E5472 mit 3,0 GHz) pro Knoten) unter SuSE Linux Enterprinse Server (SLES) Version 10 mit Java SE Runtime Environment (build 1.6.0_12-b04) ausgeführt.

Der XE-Cluster ist Teil des Systemcomputer-Systems von SGI (HLRN-II) im Komplex HICE (Hannover Altix ICE)[1] des Norddeutschen Verbundes für Hoch- und Höchstleistungsrechnen (HLRN). [HLR]

Verwendete Testdaten

Als Testdaten fanden eine Reihe Formeln im DIMACS-Format (siehe Abschnitt 4.1) Verwendung, die schon in der Arbeit von Meier zur Laufzeitanalyse implementierter Algorithmen benutzt wurden. [Mei05, Abschnitt 5.2]

Dabei handelt es sich um Formeln in 3-KNF, die entweder mindestens eine erfüllende Belegung besitzen (SAT-Instanzen) oder unerfüllbar sind (UNSAT-Instanzen). Daneben sind die Instanzen nach der Eingabelänge der Formel, also Anzahl der Variablen und Klauseln, geordnet.

1 Der Komplex HICE belegte im Juni 2013 Platz 420 auf der Top500-Liste. [TOP]

Tabelle 5.1 gibt eine Übersicht über die Anzahl der Instanzen aufgeteilt nach Eingabelänge und Erfüllbarkeit sowie Unerfüllbarkeit.

Tabelle 5.1: Anzahl der Instanzen nach Eingabelänge

Eingabelänge	Anzahl SAT-Instanzen	Anzahl UNSAT-Instanzen
5 Variablen, 28 Klauseln (5 V., 28 K.)	692	309
10 Variablen, 50 Klauseln (10 V., 50 K.)	485	479
15 Variablen, 70 Klauseln (15 V., 70 K.)	569	432
20 Variablen, 91 Klauseln (20 V., 91 K.)	1.000	-
50 Variablen, 218 Klauseln (50 V., 218 K.)	1.000	1.000
75 Variablen, 325 Klauseln (75 V., 325 K.)	100	-

Testläufe

Innerhalb eines Testlaufs wurde LSSOLVE mit einem Algorithmus für entweder alle erfüllbaren oder unerfüllbaren Formeln einer bestimmten Eingabelänge aufgerufen. Die Steuerung erfolgte jeweils über ein Shell-Skript, das an das Batch-System des HLRN übergeben wurde.

Die maximale Laufzeit für einen Batch-Job auf dem XE-System des HLRN-II beträgt einen Tag. Soweit innerhalb dieser Zeit nicht alle Instanzen des Testlaufs berechnet werden konnten, wurde ggf. ein neuer Batch-Jobs für die verbleibenden Formeln angelegt.

Generell wurden nicht für alle implementierten Algorithmen alle möglichen Testläufe absolviert. So wächst z.B. beim BF-ALGORITHMUS die Laufzeit derart an, dass schon bei der Berechnung der ersten erfüllbaren Instanz mit 50 Variablen der Batch-Job nach einem Tag ergebnislos beendet wurde.

Es wurden, soweit möglich, bei allen Testläufen zwei unterschiedlichen Zeiten gemessen: Zum einen die Zeit t_1, die durchschnittliche Zeit, die der jeweilige Algorithmus innerhalb des Programms läuft, zum anderen die Zeit t_2, die durchschnittliche Zeit, die das Programm zur Berechnung einer Instanz läuft.

Ermittlung der durchschnittlichen Zeit t_1

Die Laufzeit des Algorithmus zur Berechnung einer Instanz wird von LSSOLVE u.a. in Millisekunden als Kommentar ausgegeben. Die Berechnung erfolgt über die Differenz der Werte der Methode nanoTime() der Standardklasse java.lang.System vor und nach Ablauf des Algorithmus. Die Genauigkeit der Methode nanoTime() liegt in der Implementierung der Java-VM von Sun unter Linux bei einer Mikrosekunde (siehe dazu auch [Mei05, S. 36]).

Die durchschnittliche Zeit t_1 ist dann das arithmetische Mittel der einzelnen Laufzeiten eines Testlaufs.

Ermittlung der durchschnittlichen Zeit t_2

Die durchschnittliche Zeit, die LSSOLVE zur Berechnung einer Instanz während eines Testlaufs benötigt, wurde auf Basis der vom Batch-System des HLRN ermittelten Gesamtlaufzeit des Batch-Jobs berechnet.

Die Angabe der Laufzeit des gesamten Batch-Jobs erfolgt auf eine Sekunde genau. Zur Berechnung der durchschnittlichen Zeit t_2 wird die Laufzeit durch die Anzahl der berechneten Instanzen geteilt.

Soweit ein Testlauf auf mehrere Batch-Jobs aufgeteilt werden musste, da innerhalb der maximalen Laufzeit eines einzelnen Batch-Jobs von einem Tag nicht alle Instanzen berechnet werden konnnten, wurde auf die Angabe einer durchschnittlichen Zeit t_2 verzichtet.

Ermittlung der durchschnittlichen Zeit Δt

Falls sowohl eine durchschnittliche Zeit t_1 als auch t_2 für einen Testlauf ermittelt werden konnte, erfolgt die Angabe eines Wertes für die durchschnittlichen Zeit Δt mit $\Delta t = t_2 - t_1$. Der Wert gibt an, wie lange die Ausführung von LSSOLVE für die Berechnung einer Distanz, von der Laufzeit des Algorithmus abgesehen, durchschnittlich dauert.

Berechnung der interpolierten Laufzeit

Die Berechnung der interpolierten Laufzeiten für die einzelnen Algorithmen erfolgt auf Basis der durchschnittlichen Zeiten t_1. Dabei findet das in [Mei05, S. 38] vorgestellte Verfahren Anwendung:

Getrennt nach erfüllbaren und unerfüllbaren Instanzen werden Punktemengen (x, y) gebildet, wobei $x \in \{5, 10, 15, 20, 50, 75\}$ die Variablenanzahl und $y \in \mathbb{R}^+$ die durchschnittliche Zeit t_1 in Millisekunden repräsentiert.

Bekannt ist, dass die gesuchten Laufzeiten exponentiell ansteigen. Gesucht wird deshalb jeweils eine Exponentialfunktion g mit

$$g(x) = b \cdot e^{c \cdot x}$$

die für geeignete $b, c \in \mathbb{R}^+$ die Laufzeit beschreibt. Die Werte $f(x) := \ln(g(x))$ und $d := \ln(b)$ werden logarithmisiert und es ergibt sich eine linearisierte Gleichung

$$f(x) = \ln(g(x)) = \ln(b \cdot e^{c \cdot x}) = d + c \cdot x.$$

Über die Methode kleinster Quadrate lässt sich eine Ausgleichsgerade interpolieren, wodurch sich letztlich die gesuchten Parameter b und c für die Exponentialdarstellung ermitteln lassen.

Die so bestimmte Exponentialfunktion g kann direkt mit der in O-Notation angegebenen Laufzeitfunktion f verglichen werden, um den polynomiellen Faktor p in

$$g(x) = p(x) \cdot f(x)$$

zu bestimmen. Dieses Polynom wird mittels Taylorpolynomen approximiert.

5.1 BF-ALGORITHMUS

Tabelle 5.2: Laufzeiten des BF-ALGORITHMUS

Instanzen	SAT (t_1) ms	SAT (t_2) ms	SAT (Δt) ms	UNSAT (t_1) ms	UNSAT (t_2) ms	UNSAT (Δt) ms
5 V., 28 K.	0[1]	160,405	160,405	0[1]	233,010	233,010
10 V., 50 K.	5,610	206,186	200,576	13,184	212,944	199,760
15 V., 70 K.	35,842	235,501	199,659	59,421	256,944	197,523
20 V., 91 K.	313,726	525,000	211,274	-	-	-

[1] Laufzeit des Algorithmus in allen Fällen unter der messbaren Grenze. Zur Berechnung der interpolierten Laufzeit wurde ein Wert von 0,05 ms angenommen.

Theoretische Laufzeit: $O(2^n)$

Interpolierte Laufzeit

für SAT-Instanzen: $0{,}0066843 \cdot 1{,}7538^n \in O(1{,}7538^n)$

für UNSAT-Instanzen: $0{,}002856 \cdot 2{,}0301^n \in O(2{,}0301^n)$

Tabelle 5.3: Polynomfaktoren für den BF-ALGORITHMUS

SAT-Instanzen	UNSAT-Instanzen
$-6{,}21305 \cdot 10^{-11} \cdot x^{10}$	$-1{,}09409 \cdot 10^{-10} \cdot x^{10}$
$-5{,}43598 \cdot 10^{-10} \cdot x^{9}$	$-8{,}69942 \cdot 10^{-10} \cdot x^{9}$
$-4{,}66318 \cdot 10^{-9} \cdot x^{8}$	$-6{,}69981 \cdot 10^{-9} \cdot x^{8}$
$-3{,}86425 \cdot 10^{-8} \cdot x^{7}$	$-4{,}91066 \cdot 10^{-8} \cdot x^{7}$
$-3{,}0279 \cdot 10^{-7} \cdot x^{6}$	$-3{,}3484 \cdot 10^{-7} \cdot x^{6}$
$-2{,}17977 \cdot 10^{-6} \cdot x^{5}$	$-2{,}06407 \cdot 10^{-6} \cdot x^{5}$
$-0{,}0000138835 \cdot x^{4}$	$-0{,}0000110931 \cdot x^{4}$
$-0{,}000074386 \cdot x^{3}$	$-0{,}0000495256 \cdot x^{3}$
$-0{,}000311525 \cdot x^{2}$	$-0{,}000171069 \cdot x^{2}$
$-0{,}000899546 \cdot x$	$-0{,}000404135 \cdot x$
$-0{,}00133472$	$-0{,}000487513$

5.2 LS-ALGORITHMUS

Tabelle 5.4: Laufzeiten des LS-ALGORITHMUS

Instanzen	SAT (t_1) ms	SAT (t_2) ms	SAT (Δt) ms	UNSAT (t_1) ms	UNSAT (t_2) ms	UNSAT (Δt) ms
5 V., 28 K.	0,065	197,977	197,912	21,502	213,592	192,090
10 V., 50 K.	4,843	202,062	197,219	126,735	338,205	211,470
15 V., 70 K.	16,924	216,169	199,245	1.123,282	1.351,852	228,570
20 V., 91 K.	109,76	348,000	238,240	-	-	-

Theoretische Laufzeit: $O(1{,}5^n)$

Interpolierte Laufzeit

für SAT-Instanzen: $0{,}013148 \cdot 1{,}6039^n \in O(1{,}6039^n)$

für UNSAT-Instanzen: $2{,}7456 \cdot 1{,}488^n \in O(1{,}488^n)$

Tabelle 5.5: Polynomfaktoren für den LS-ALGORITHMUS

SAT-Instanzen	UNSAT-Instanzen
$-5{,}46852 \cdot 10^{-11} \cdot x^{10}$	$0{,}0000880528 \cdot x^{10}$
$-4{,}97171 \cdot 10^{-10} \cdot x^{9}$	$-0{,}000223788 \cdot x^{9}$
$-4{,}46877 \cdot 10^{-9} \cdot x^{8}$	$+0{,}000568766 \cdot x^{8}$
$-3{,}92627 \cdot 10^{-8} \cdot x^{7}$	$-0{,}00144542 \cdot x^{7}$
$-3{,}31012 \cdot 10^{-7} \cdot x^{6}$	$+0{,}00367572 \cdot x^{6}$
$-2{,}60549 \cdot 10^{-6} \cdot x^{5}$	$-0{,}00930412 \cdot x^{5}$
$-0{,}0000184335 \cdot x^{4}$	$+0{,}0242172 \cdot x^{4}$
$-0{,}000111266 \cdot x^{3}$	$-0{,}0543688 \cdot x^{3}$
$-0{,}000531189 \cdot x^{2}$	$+0{,}210447 \cdot x^{2}$
$-0{,}00176499 \cdot x$	$+0{,}010696 \cdot x$
$-0{,}00303544$	$+2{,}71842$

5.3 LF-ALGORITHMUS

Tabelle 5.6: Laufzeiten des LF-ALGORITHMUS

Instanzen	SAT (t_1) ms	SAT (t_2) ms	SAT (Δt) ms	UNSAT (t_1) ms	UNSAT (t_2) ms	UNSAT (Δt) ms
5 V., 28 K.	0,090	192,197	192,107	19,773	200,647	180,874
10 V., 50 K.	4,700	160,405	155,705	87,355	319,415	232,060
15 V., 70 K.	17,926	228,471	210,545	286,606	490,741	204,135
20 V., 91 K.	60,019	267,000	206,981	-	-	-
50 V., 218 K.	-[1]	-	-	-	-	-

[1] Gewählte Wiederholungskonstante von $c = 7$ zu klein, um bei allen Instanzen eine erfüllende Belegung zu finden.

Theoretische Laufzeit: $O(1{,}5^n)$ (für den LS-ALGORITHMUS)

Interpolierte Laufzeit

für SAT-Instanzen: $0{,}025185 \cdot 1{,}5137^n \in O(1{,}5137^n)$

für UNSAT-Instanzen: $5{,}4557 \cdot 1{,}3066^n \in O(1{,}3066^n)$

Tabelle 5.7: Polynomfaktoren für den LF-ALGORITHMUS

SAT-Instanzen	UNSAT-Instanzen
$-9{,}40559 \cdot 10^{-11} \cdot x^{10}$	$5{,}57133 \cdot 10^{-9} \cdot x^{10}$
$-8{,}27934 \cdot 10^{-10} \cdot x^{9}$	$-3{,}53406 \cdot 10^{-8} \cdot x^{9}$
$-7{,}24787 \cdot 10^{-9} \cdot x^{8}$	$+2{,}24635 \cdot 10^{-7} \cdot x^{8}$
$-6{,}26362 \cdot 10^{-8} \cdot x^{7}$	$-1{,}41212 \cdot 10^{-6} \cdot x^{7}$
$-5{,}26882 \cdot 10^{-7} \cdot x^{6}$	$+9{,}35588 \cdot 10^{-6} \cdot x^{6}$
$-4{,}21219 \cdot 10^{-6} \cdot x^{5}$	$-0{,}0000490055 \cdot x^{5}$
$-0{,}0000308564 \cdot x^{4}$	$+0{,}000543391 \cdot x^{4}$
$-0{,}000196445 \cdot x^{3}$	$+0{,}000898427 \cdot x^{3}$
$-0{,}00100519 \cdot x^{2}$	$+0{,}0593156 \cdot x^{2}$
$-0{,}00362668 \cdot x$	$+0{,}353059 \cdot x$
$-0{,}00684096$	$+3{,}21548$

5.4 RW-ALGORITHMUS

Tabelle 5.8: Laufzeiten des RW-ALGORITHMUS

Instanzen	SAT (t_1) ms	SAT (t_2) ms	SAT (Δt) ms	UNSAT (t_1) ms	UNSAT (t_2) ms	UNSAT (Δt) ms
5 V., 28 K.	0,072	167,630	167,558	1,754	200,647	198,893
10 V., 50 K.	2,118	164,948	162,830	18,056	227,557	209,501
15 V., 70 K.	7,364	156,415	149,051	38,912	263,000	224,977
20 V., 91 K.	15,361	179,000	163,639	-	-	-
50 V., 218 K.	3.359,876	3.607,000	247,124	1.627.063,439[1]	-[2]	-
75 V., 325 K.	935.673,720	-[3]	-	-	-	-

[1] In fünf Batch-Jobs wurden 264 Instanzen berechnet.
[2] Keine genaue Angabe möglich, da Testlauf über fünf Batch-Jobs lief.
[3] Keine genaue Angabe möglich, da Testlauf über zwei Batch-Jobs lief.

Theoretische Laufzeit: $O(1{,}3333^n)$

Interpolierte Laufzeit

für SAT-Instanzen: $0{,}13117 \cdot 1{,}2348^n \in O(1{,}2348^n)$

für UNSAT-Instanzen: $0{,}53804 \cdot 1{,}3481^n \in O(1{,}3481^n)$

Tabelle 5.9: Polynomfaktoren für den RW-ALGORITHMUS

SAT-Instanzen	UNSAT-Instanzen
$-7{,}16579 \cdot 10^{-11} \cdot x^{10}$	$-0{,}00109037 \cdot x^{10}$
$-6{,}90148 \cdot 10^{-10} \cdot x^{9}$	$-0{,}00226252 \cdot x^{9}$
$-6{,}6458 \cdot 10^{-9} \cdot x^{8}$	$-0{,}0046947 \cdot x^{8}$
$-6{,}39492 \cdot 10^{-8} \cdot x^{7}$	$-0{,}00974146 \cdot x^{7}$
$-6{,}13642 \cdot 10^{-7} \cdot x^{6}$	$-0{,}0202134 \cdot x^{6}$
$-5{,}83433 \cdot 10^{-6} \cdot x^{5}$	$-0{,}0419408 \cdot x^{5}$
$-0{,}0000540313 \cdot x^{4}$	$-0{,}0869911 \cdot x^{4}$
$-0{,}000469138 \cdot x^{3}$	$-0{,}179908 \cdot x^{3}$
$-0{,}00354615 \cdot x^{2}$	$-0{,}365308 \cdot x^{2}$
$-0{,}0203232 \cdot x$	$-0{,}677654 \cdot x$
$-0{,}0645744$	$-0{,}868086$

5.5 Parallelisierung

Im Rahmen dieser Arbeit wurden die Algorithmen zur lokalen Suche so implementiert, dass sie sequentiell ablaufen. Die gebildeten Hamming-Kugeln werden nacheinander durchsucht. In diesem Abschnitt erfolgt ein kurzer Hinweis darauf, wie durch eine Parallelisierung kürzere Laufzeiten erreicht werden könnten.

Eine Parallelisierung von Algorithmen kann die tatsächliche Laufzeit (*wall-clock time*) verkürzen, indem die Fähigkeiten moderner Hardwarearchitekturen ausgenutzt werden. Dies ist für praktische Anwendungen interessant und spielt auch bei den jährlichen SAT-Wettbewerben zunehmend eine Rolle.

Die Regeln der SAT Competition 2011, die auch für die Wettbewerbe in den Jahren 2012 und 2013 gegolten haben, sehen die Möglichkeit vor, den Programmen mit der Umgebungsvariablen NBCORE die Anzahl der zur Verfügung stehenden *processing units* zu übergeben. Dabei hängt von der konkreten Umgebung ab, ob es sich bei den *processing units* um Prozessoren, Prozessorkerne oder virtuelle Prozessoren handelt.

Bei der SAT Competition 2013 wurden separate Wettbewerbe für Programme mit parallel arbeitenden Algorithmen ausgerichtet [SATf].

Die in dieser Arbeit untersuchten Algorithmen zur lokalen Suchen eignen sich gut für eine Parallelisierung. Die Suche innerhalb einer Hamming-Kugel beim LS-ALGORITHMUS ist z.B. unabhängig von der Suche nach einer erfüllenden Belegung in einer anderen. Es besteht lediglich eine Verbindung über den Pseudozufallszahlengenerator, von dem die Startbelegungen der einzelnen Durchläufe abhängen.

Die im Rahmen der SAT-Wettbewerbe gewünschte Reproduzierbarkeit der Ergebnisse und Laufzeiten sollte durch eine Parellelisierung nicht beeinträchtigt werden. Einzig die Ausgabe einer erfüllenden Belegung könnte zwischen mehreren Programmaufrufen trotz gleicher Initialisierung unterschiedlich ausfallen, wenn es denn mehr als eine erfüllende Belegung für die untersuchte Formel gäbe. Dies könnte dann geschehen, wenn mehrere Hamming-Kugeln, in denen sich jeweils mindestens eine erfüllende Belegung finden ließe, parallel durchsucht würden und der Algorithmus keinen direkten Einfluss besäße, wann genau in welcher Kugel gesucht würde.

6 Zusammenfassung

Mit LSSOLVE wurde ein Programm erstellt, das grundsätzlich für einen der jährlich stattfindenden SAT-Wettbewerbe eingereicht werden könnte. Der Aufruf des Programms, die Eingabe in Form von Instanzen im DIMACS-Format sowie die Ausgaben wurden nach den Regelen aktueller SAT-Wettbewerbe realisiert.

Tabelle 6.1: Zusammenfassung der Laufzeiten aller implementierten Algorithmen

		interpolierte Laufzeiten	
Algorithmus	theoretische Laufzeit	SAT-Instanzen	UNSAT-Instanzen
BF-ALGORITHMUS	$O(2^n)$	$O(1{,}7538^n)$	$O(2{,}0301^n)$
LS-ALGORITHMUS	$O(1{,}5^n)$	$O(1{,}6039^n)$	$O(1{,}488^n)$
LF-ALGORITHMUS	$O(1{,}5^n)$	$O(1{,}5137^n)$	$O(1{,}3066^n)$
RW-ALGORITHMUS	$O(1{,}3333^n)$	$O(1{,}2348^n)$	$O(1{,}3481^n)$

Die interpolierten Laufzeiten liegen allesamt in der Nähe der theoretischen Laufzeiten. Beim BF-ALGORITHMUS und RW-ALGORITHMUS liegen die interpolierten Laufzeiten für die SAT-Instanzen unter den theoretischen Laufzeiten, diejenigen für die UNSAT-Instanzen liegen knapp darüber, wobei jeweils die erste Nachkommastelle noch übereinstimmt.

Umgekehrt verhält es sich beim LS-ALGORITHMUS und LF-ALGORITHMUS. Für die SAT-Instanzen sind die interpolierten Laufzeiten etwas schlechter als die theoretische Laufzeit, bei den UNSAT-Instanzen sind sie besser, beim LF-ALGORITHMUS sogar deutlich.

Ein Problem ergab sich beim LF-ALGORITHMUS im Zusammenhang mit den erfüllbaren Formeln mit 50 Variablen während der Testläufe zum Laufzeitvergleich. Hier wurde bei gut 18 % der untersuchten Formeln die erfüllende Belegung übersehen. Dabei schien bei der Voruntersuchung zur Berechnung der Anzahl der zu durchsuchenden Hamming-Kugeln eigentlich ein geeigneter Faktor gefunden worden zu sein. Beim RW-ALGORITHMUS zeigten sich ähnliche Probleme nicht, bei keiner der Formeln mit 50 oder 75 Variablen wurde

eine erfüllende Belegung übersehen. Der LS-ALGORITHMUS wurde nur mit Formeln mit bis zu 20 Variablen getestet.

Nicht weiter ausgewertet wurden die durchschnittlichen Ausführungszeiten von LSSOLVE abzüglich der Laufzeiten der jeweiligen Algorithmen (Δt). Tendenziell steigt der Wert mit der Eingabelänge in den Variablen, allerdings sind die Ergebnisse nicht eindeutig. Teilweise sinken die Zeiten für die Formeln mit 10 oder 15 Variablen wieder, was für einen Einfluss der Testumgebung auf die Werte spricht, der bei den Formeln mit weniger Variablen stärker zum Tragen kommt.

Literaturverzeichnis

[Com] *Commons CLI*, http://commons.apache.org/proper/commons-cli/index.html, abgerufen am 21. September 2013

[HLR] *The SGI System - Hardware Overview*, https://www.hlrn.de/home/view/System/SgiHardware, abgerufen am 21. September 2013

[Mei05] Meier, Arne: *SAT-Algorithmen*, Universität Hannover, Bachelorarbeit, 2005

[Mei13] Meier, Arne: *Skript zur Vorlesung Komplexität von Algorithmen*, Universität Hannover, 2013

[Pir07] Pirsch, Peter: *Skript zur Vorlesung Grundlagen digitaler Systeme*, 2. Auflage, Universität Hannover, 2007

[SATa] *SAT Competition 2011*, http://satcompetition.org/2011/, abgerufen am 21. September 2013

[SATb] *Rules of the 2011 SAT Competition*, http://www.satcompetition.org/2011/rules.pdf, abgerufen am 21. September 2013

[SATc] *SAT Challenge 2012*, http://baldur.iti.kit.edu/SAT-Challenge-2012/, abgerufen am 21. September 2013

[SATd] *SAT Competition 2013*, http://satcompetition.org/2013/index.shtml, abgerufen am 21. September 2013

[SATe] *SAT Competition 2013*, http://satcompetition.org/2013/rules.shtml, abgerufen am 21. September 2013

[SATf] *SAT Competition 2013*, http://satcompetition.org/2013/description.shtml, abgerufen am 21. September 2013

[Sch11] Schöning, Uwe und Torán, Jacobo: *Das Erfüllbarkeitsproblem SAT: Algorithmen und Analysen*, Lehmanns Media, 2012

[TOP] *Top500 List - June 2013*, `http://www.top500.org/list/2013/06/?page=5`, abgerufen am 21. September 2013

Tabellenverzeichnis

Algorithmenverzeichnis