Grundlagen der Künstlichen Intelligenz

13. Aussagenlogik: Erfüllbarkeitsalgorithmen

Malte Helmert

Universität Basel

26. April 2013

Motivation

Motivation für Aussagenlogik

- Aussagenlogik erlaubt Repräsentation von Wissen und Schlussfolgerungen auf Grundlage dieses Wissens
- viele Anwendungsprobleme direkt kodierbar, z. B.:
  - Constraint-Satisfaction-Probleme aller Art
  - Schaltkreisentwurf und -verifikation
- viele Probleme verwenden Logik als einen Bestandteil, z. B.:
  - Handlungsplanung
  - General Game Playing
  - Beschreibungslogik-Anfragen (Semantic Web)

Aussagenlogik: algorithmische Fragestellungen

wesentliche Fragestellungen:

- Schlussfolgern ($\Theta \models \varphi$?):
  Folgt aus Formeln Θ die Formel φ logisch?
- Äquivalenz ($\varphi \equiv \psi$):
  Sind Formeln φ und ψ logisch äquivalent?
- Erfüllbarkeit (SAT):
  Ist Formel φ erfüllbar? Falls ja, finde eine erfüllende Belegung.

Das Erfüllbarkeitsproblem

Das Erfüllbarkeitsproblem (SAT)

Gegeben:

aussagenlogische Formel in konjunktiver Normalform (KNF)

Üblicherweise repräsentiert als Paar $\langle V, \Delta \rangle$:

- V Menge von Aussagevariablen (Propositionen)
- Δ Menge von Klauseln über V
  (Klausel = Menge von Literalen v bzw. $\neg v$ mit $v \in V$)

Gesucht:

- erfüllende Belegung der Formel (Modell)
- oder Beweis, dass keine erfüllende Belegung existiert

SAT ist ein berühmtes NP-vollständiges Problem
(Cook 1971; Levin 1973).

Relevanz von SAT

- Unter SAT versteht man oft auch das Erfüllbarkeitsproblem für allgemeine Logikformeln (statt Einschränkung auf KNF).
- Allgemeines SAT ist auf den KNF-Fall zurückführbar (Aufwand für Umformung ist $O(n)$)
- Alle zuvor genannten Logikprobleme sind auf SAT zurückführbar (Aufwand für Umformung ist $O(n)$)

$\rightsquigarrow$ SAT-Algorithmen sehr wichtig und sehr intensiv erforscht

dieses Kapitel: SAT-Algorithmen

Systematische Suche: DPLL

SAT vs. CSP

SAT kann als Constraint-Satisfaction-Problem aufgefasst werden:

- CSP-Variablen = Aussagevariablen
- Wertebereiche = $\{\mathbf{F}, \mathbf{T}\}$
- Constraints = Klauseln

Allerdings haben wir hier oft Constraints,
die mehr als zwei Variablen betreffen.

Wegen Verwandtschaft alle CSP-Ideen auf SAT anwendbar:

- Suche
- Inferenz
- Variablen- und Werteordnungen

Der DPLL-Algorithmus

Der DPLL-Algorithmus (Davis/Putnam/Logemann/Loveland) entspricht Backtracking mit Inferenz bei CSPs.

- rekursiver Aufruf DPLL(Δ, I)
  für Klauselmenge Δ und partielle Belegung I
- Ergebnis ist erfüllende Belegung, die I erweitert;
  unsatisfiable, wenn keine solche Belegung existiert
- oberster Aufruf als DPLL($\Delta, \emptyset$)

Inferenz in DPLL:

- simplify: nachdem der Variablen v der Wert d zugewiesen wird, werden alle Klauseln vereinfacht, die über v sprechen
  $\rightsquigarrow$ entspricht Forward Checking (für mehrstellige Constraints)
- Unit Propagation: Variablen, die in Klauseln ohne weitere Variablen (Einheitsklauseln) auftreten, werden sofort belegt (entspricht minimum remaining values-Variablenordnung)

Der DPLL-Algorithmus: Pseudo-Code

function DPLL(Δ, I):

if $\square \in \Delta$: [Es gibt eine leere Klausel $\rightsquigarrow$ unerfüllbar]
    return unsatisfiable
else if $\Delta = \emptyset$: [keine Klauseln übrig $\rightsquigarrow$ Belegung I erfüllt die Formel]
    return I
else if there exists a unit clause $\{v\}$ or $\{\neg v\}$ in Δ: [Unit Propagation]
    Let v be such a variable, d the truth value that satisfies the clause.
    $\Delta' :=$ simplify(Δ, v, d)
    return DPLL($V, \Delta', I \cup \{v \mapsto d\}$)
else: [Splitting Rule]
    Select some variable v which occurs in Δ.
    for each $d \in \{\mathbf{F}, \mathbf{T}\}$ in some order:
        $\Delta' :=$ simplify(Δ, v, d)
        $I' :=$ DPLL($V, \Delta', I \cup \{v \mapsto d\}$)
        if $I' \neq$ **unsatisfiable**
            return I'
    return unsatisfiable

Der DPLL-Algorithmus: simplify

function simplify(Δ, v, d)

Let ℓ be the literal on v that is satisfied by $v \mapsto d$.
Let $\bar{\ell}$ be the complementary (opposite) literal to ℓ.
$\Delta' := \{C \mid C \in \Delta \text{ s.t. } \ell \notin C\}$
$\Delta'' := \{C \setminus \{\bar{\ell}\} \mid C \in \Delta'\}$
return Δ''

Beispiel (1)

$$\Delta = \{\{X, Y, \neg Z\}, \{\neg X, \neg Y\}, \{Z\}, \{X, \neg Y\}\}$$

Beispiel (1)

$$\Delta = \{\{X, Y, \neg Z\}, \{\neg X, \neg Y\}, \{Z\}, \{X, \neg Y\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$

Beispiel (1)

$$\Delta = \{\{X, Y, \neg Z\}, \{\neg X, \neg Y\}, \{Z\}, \{X, \neg Y\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{X, Y\}, \{\neg X, \neg Y\}, \{X, \neg Y\}\}$

Beispiel (1)

$$\Delta = \{\{X, Y, \neg Z\}, \{\neg X, \neg Y\}, \{Z\}, \{X, \neg Y\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{X, Y\}, \{\neg X, \neg Y\}, \{X, \neg Y\}\}$
2. Splitting Rule:

Beispiel (1)

$$\Delta = \{\{X, Y, \neg Z\}, \{\neg X, \neg Y\}, \{Z\}, \{X, \neg Y\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{X, Y\}, \{\neg X, \neg Y\}, \{X, \neg Y\}\}$
2. Splitting Rule:

2a. $X \mapsto \mathbf{F}$
$\{\{Y\}, \{\neg Y\}\}$

Beispiel (1)

$$\Delta = \{\{X, Y, \neg Z\}, \{\neg X, \neg Y\}, \{Z\}, \{X, \neg Y\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{X, Y\}, \{\neg X, \neg Y\}, \{X, \neg Y\}\}$
2. Splitting Rule:

2a. $X \mapsto \mathbf{F}$
$\{\{Y\}, \{\neg Y\}\}$

3a. Unit Propagation: $Y \mapsto \mathbf{T}$
$\{\square\}$

Beispiel (1)

$$\Delta = \{\{X, Y, \neg Z\}, \{\neg X, \neg Y\}, \{Z\}, \{X, \neg Y\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{X, Y\}, \{\neg X, \neg Y\}, \{X, \neg Y\}\}$
2. Splitting Rule:

2a. $X \mapsto \mathbf{F}$
$\{\{Y\}, \{\neg Y\}\}$

3a. Unit Propagation: $Y \mapsto \mathbf{T}$
$\{\square\}$

2b. $X \mapsto \mathbf{T}$
$\{\{\neg Y\}\}$

Beispiel (1)

$$\Delta = \{\{X, Y, \neg Z\}, \{\neg X, \neg Y\}, \{Z\}, \{X, \neg Y\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{X, Y\}, \{\neg X, \neg Y\}, \{X, \neg Y\}\}$
2. Splitting Rule:

2a. $X \mapsto \mathbf{F}$
$\{\{Y\}, \{\neg Y\}\}$

3a. Unit Propagation: $Y \mapsto \mathbf{T}$
$\{\square\}$

2b. $X \mapsto \mathbf{T}$
$\{\{\neg Y\}\}$

3b. Unit Propagation: $Y \mapsto \mathbf{F}$
$\{\}$

Beispiel (1)

$$\Delta = \{\{X, Y, \neg Z\}, \{\neg X, \neg Y\}, \{Z\}, \{X, \neg Y\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{X, Y\}, \{\neg X, \neg Y\}, \{X, \neg Y\}\}$
2. Splitting Rule:

2a. $X \mapsto \mathbf{F}$
$\{\{Y\}, \{\neg Y\}\}$

3a. Unit Propagation: $Y \mapsto \mathbf{T}$
$\{\square\}$

2b. $X \mapsto \mathbf{T}$
$\{\{\neg Y\}\}$

3b. Unit Propagation: $Y \mapsto \mathbf{F}$
$\{\}$

Beispiel (2)

$$\Delta = \{\{W, \neg X, \neg Y, \neg Z\}, \{X, \neg Z\}, \{Y, \neg Z\}, \{Z\}\}$$

Beispiel (2)

$$\Delta = \{\{W, \neg X, \neg Y, \neg Z\}, \{X, \neg Z\}, \{Y, \neg Z\}, \{Z\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$

Beispiel (2)

$$\Delta = \{\{W, \neg X, \neg Y, \neg Z\}, \{X, \neg Z\}, \{Y, \neg Z\}, \{Z\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{W, \neg X, \neg Y\}, \{X\}, \{Y\}\}$

Beispiel (2)

$$\Delta = \{\{W, \neg X, \neg Y, \neg Z\}, \{X, \neg Z\}, \{Y, \neg Z\}, \{Z\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{W, \neg X, \neg Y\}, \{X\}, \{Y\}\}$
2. Unit Propagation: $X \mapsto \mathbf{T}$
   $\{\{W, \neg Y\}, \{Y\}\}$

Beispiel (2)

$$\Delta = \{\{W, \neg X, \neg Y, \neg Z\}, \{X, \neg Z\}, \{Y, \neg Z\}, \{Z\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{W, \neg X, \neg Y\}, \{X\}, \{Y\}\}$
2. Unit Propagation: $X \mapsto \mathbf{T}$
   $\{\{W, \neg Y\}, \{Y\}\}$
3. Unit Propagation: $Y \mapsto \mathbf{T}$
   $\{\{W\}\}$

Beispiel (2)

$$\Delta = \{\{W, \neg X, \neg Y, \neg Z\}, \{X, \neg Z\}, \{Y, \neg Z\}, \{Z\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{W, \neg X, \neg Y\}, \{X\}, \{Y\}\}$
2. Unit Propagation: $X \mapsto \mathbf{T}$
   $\{\{W, \neg Y\}, \{Y\}\}$
3. Unit Propagation: $Y \mapsto \mathbf{T}$
   $\{\{W\}\}$
4. Unit Propagation: $W \mapsto \mathbf{T}$
   $\{\}$

Beispiel (2)

$$\Delta = \{\{W, \neg X, \neg Y, \neg Z\}, \{X, \neg Z\}, \{Y, \neg Z\}, \{Z\}\}$$

1. Unit Propagation: $Z \mapsto \mathbf{T}$
   $\{\{W, \neg X, \neg Y\}, \{X\}, \{Y\}\}$
2. Unit Propagation: $X \mapsto \mathbf{T}$
   $\{\{W, \neg Y\}, \{Y\}\}$
3. Unit Propagation: $Y \mapsto \mathbf{T}$
   $\{\{W\}\}$
4. Unit Propagation: $W \mapsto \mathbf{T}$
   $\{\}$

Eigenschaften von DPLL

- DPLL ist korrekt und vollständig
- DPLL erzeugt ein Modell, falls eines existiert
  - Manche Variablen werden evtl. in der Lösung I nicht belegt; deren Werte können dann beliebig gewählt werden.
- Zeitaufwand im Allgemeinen exponentiell

$\rightsquigarrow$ gute Variablenordnungen in der Praxis wichtig; ebenso zusätzliche Inferenzmethoden, v.a. clause learning

- beste bekannte SAT-Algorithmen basieren auf DPLL

Hornformeln

wichtiger Spezialfall: Hornformeln

Definition (Hornformel)

Eine Hornklausel ist eine Klausel
mit maximal einem positivem Literal, also von der Form

$$\neg x_1 \vee \cdots \vee \neg x_n \vee y \text{ oder } \neg x_1 \vee \cdots \vee \neg x_n$$

(Der Fall $n = 0$ ist erlaubt.)

Eine Hornformel ist eine aussagenlogische Formel
in konjunktiver Normalform, die nur aus Hornklauseln besteht.

$\leadsto$ Grundlage von Logikprogrammierung (z.B. PROLOG)
und deduktiven Datenbanken

DPLL auf Hornformeln

Satz (DPLL auf Hornformeln)

Wenn die Eingabeformel φ eine Hornformel ist, dann ist der Zeitaufwand von DPLL polynomiell in der Länge von φ.

Beweis.

Eigenschaften:

1. Wenn Δ eine Hornformel ist, dann ist auch simplify(Δ, v, d) eine Hornformel. (Warum?)

   $\rightsquigarrow$ alle während der Suche von DPLL betrachteten Formeln sind Hornformeln, wenn die Eingabe es ist
2. Jede Hornformel ohne leere oder Einheitsklauseln ist erfüllbar:
   - alle solchen Klauseln enthalten mindestens zwei Literale
   - da Horn: mindestens eines davon negativ
   - Zuweisung **F** an alle Variablen erfüllt die Formel

DPLL auf Hornformeln (Fortsetzung)

Beweis (Fortsetzung).

3. Aus 2. folgt:
   - immer, wenn die Splitting Rule angewandt wird, ist die aktuelle Formel erfüllbar, und
   - immer, wenn dabei eine falsche Entscheidung getroffen wird, wird dies sofort (d. h. nur durch Unit-Propagation-Schritte und Herleiten einer leeren Klausel) erkannt.
4. Deshalb kann der erzeugte Suchbaum für n Variablen nur maximal n viele Knoten enthalten, in denen die Splitting Rule angewandt wird (und der Baum verzweigt).
5. Damit ist der Suchbaum nur polynomiell gross und folglich die Gesamtlaufzeit polynomiell.

□

Lokale Suche: GSAT und Walksat

Lokale Suche für SAT

- Neben systematischen gibt es auch erfolgreiche lokale Suchverfahren für SAT.
- Diese sind im Normalfall nicht vollständig und können insbesondere nicht die Unerfüllbarkeit einer Formel zeigen.
- Oft ist dies aber verschmerzbar, wenn man dafür für schwierigere Probleme erfüllende Belegungen finden kann.
- Insgesamt sind DPLL-basierte systematische Verfahren allerdings in den letzten Jahren erfolgreicher.

Lokale Suche für SAT: Ideen

Lokale Suchverfahren sind für SAT direkt anwendbar:

- Zustände: (vollständige) Belegungen
- Zielzustände: erfüllende Belegungen
- Suchnachbarschaft: ändere Belegung einer Variable
- Heuristiken: je nach Algorithmus;
  z.B. Anzahl unerfüllter Klauseln

GSAT (Greedy SAT): Pseudo-Code

Hilfsfunktionen:

- violated(Δ, I): Anzahl Klauseln in Δ, die I nicht erfüllt
- flip(I, v): Die Belegung, die aus I entsteht,
  wenn man die Belegung der Aussagevariable v ändert

function GSAT(Δ):

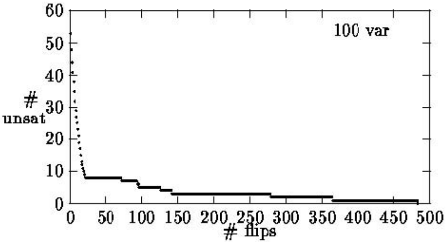
repeat *max-tries* **times**:
    $I :=$ a random truth assignment
    repeat *max-flips* **times**:
        if $I \models \Delta$:
            return I
        $V_{\text{greedy}} :=$ the set of variables v occurring in Δ
                for which violated(Δ, flip(I, v)) is minimal
        randomly select $v \in V_{\text{greedy}}$
        $I :=$ flip(I, v)
return no solution found

GSAT: Diskussion

GSAT hat übliche Merkmale von lokalen Suchverfahren:

- Hill-Climbing
- Zufall (allerdings relativ wenig!)
- Neustarts

empirisch wird viel Zeit auf Plateaus verbracht:



Walksat: Pseudo-Code

$\text{lost}(\Delta, I, v)$: #Klauseln in Δ, die I erfüllt, aber $\text{flip}(I, v)$ nicht

function Walksat(Δ):

repeat *max-tries* **times**:
  $I :=$ a random truth assignment
  repeat *max-flips* **times**:
    if $I \models \Delta$:
      return I
    $C :=$ randomly chosen unsatisfied clause in Δ
    if there is a variable v in C with $\text{lost}(\Delta, I, v) = 0$:
      $V_{\text{choices}} :=$ all such variables
    else with probability p_{noise}:
      $V_{\text{choices}} :=$ all variables occurring in C
    else:
      $V_{\text{choices}} :=$ variables v in C that minimize $\text{lost}(\Delta, I, v)$
    randomly select $v \in V_{\text{choices}}$
    $I := \text{flip}(I, v)$
return no solution found

Walksat vs. GSAT

Vergleich GSAT vs. Walksat:

- sehr viel mehr Zufall in Walksat
  durch zufällige Wahl der betrachteten Klausel
- auch „unintuitive" Schritte, die die Zahl der verletzten Klauseln erst mal erhöhen, sind bei Walksat meistens möglich

$\rightsquigarrow$ geringere Gefahr, in lokalen Minima stecken zu bleiben

Wie schwierig ist SAT?

Wie schwierig ist SAT in der Praxis?

- SAT ist NP-vollständig

$\leadsto$ Algorithmen wie DPLL benötigen im schlechtesten Fall exponentielle Zeit

- Wie sieht es im Durchschnitt aus?
- hängt davon ab, über welche Probleminstanzen der Durchschnitt gebildet wird

SAT: polynomielle durchschnittliche Laufzeit

Gute Nachrichten (Goldberg 1979)

Konstruierte zufällige KNF-Formeln mit n Variablen und k Klauseln wie folgt:

In jeder Klausel taucht jede Variable

- mit Wahrscheinlichkeit $\frac{1}{3}$ positiv,
- mit Wahrscheinlichkeit $\frac{1}{3}$ negativ,
- mit Wahrscheinlichkeit $\frac{1}{3}$ gar nicht auf.

Dann ist die Laufzeit von DPLL polynomiell in n und k.

$\rightsquigarrow$ leider kein sehr realistisches Modell für praktisch interessante KNF-Formeln (fast alle Zufallsformeln erfüllbar)

Phasenübergänge

Wie finden wir interessante zufällige Probleme?

Vermutung von Cheeseman et al.:

Cheeseman et al., IJCAI 1991

Alle NP-vollständigen Probleme haben mindestens einen Grössenparameter, für den die schwierigen Probleminstanzen in der Nähe eines kritischen Werts für diesen Parameter liegen.

Dieser so genannte Phasenübergang trennt zwei Problemregionen, z. B. eine zu stark eingeschränkte (over-constrained) von einer zu schwach eingeschränkten (under-constrained).

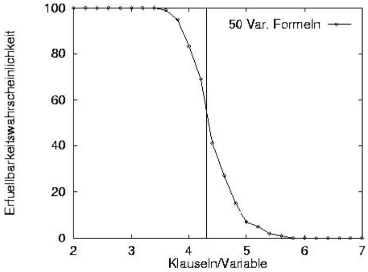
$\rightsquigarrow$ bestätigt z. B. für Graphfärbung, Hamilton-Pfade und SAT

Phasenübergänge für 3-SAT

Problemmodell von Mitchell et al., AAAI 1992

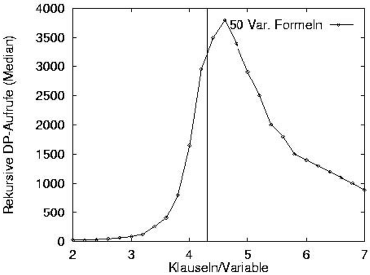
- feste Klausellänge 3
- wähle in jeder Klausel die Variablen zufällig
- Literale sind mit Wahrscheinlichkeit $\frac{1}{2}$ positiv bzw. negativ

kritischer Parameter: Anz. Klauseln geteilt durch Anz. Variablen
Phasenübergang bei Verhältnis von ca. 4.3



Phasenübergang bei DPLL

DPLL zeigt hohe Laufzeit in der Nähe des Phasenübergangs:



Phasenübergang: intuitive Erklärung

- Wenn es sehr viele Klauseln gibt,
  das Problem daher mit hoher Wahrscheinlichkeit unlösbar ist,
  wird das schnell durch Unit-Propagation nachgewiesen.
- Wenn es sehr wenige Klauseln gibt,
  gibt es sehr viele erfüllende Belegungen,
  und es ist leicht, eine zu finden.
- Nahe des Phasenübergangs gibt es viele „Fast-Lösungen“,
  die vom Suchalgorithmen verfolgt werden müssen.

Ausblick

Stand der Wissenschaft

- SAT-Forschung allgemein:
  $\rightsquigarrow$ `http://www.satlive.org/`
- SAT-Konferenzen seit 1996; seit 2000 jedes Jahr
  $\rightsquigarrow$ `http://www.satisfiability.org/`
- Wettbewerbe für SAT-Algorithmen seit 1992
  $\rightsquigarrow$ `http://www.satcompetition.org/`
  - grösste Instanzen haben mehr als 1'000'000 Literale
  - verschiedene Disziplinen (z. B. SAT vs. SAT+UNSAT; industrielle vs. zufällige Instanzen)

Weiterführende Themen

DPLL-basierte SAT-Algorithmen:

- effiziente Implementierungstechniken
- gute Variablenordnungen
- clause learning

lokale Suchalgorithmen:

- effiziente Implementierungstechniken
- adaptive Suchverfahren („schwierige“ Klauseln werden mit der Zeit erkannt und priorisiert)