

SAT-Solving und Anwendungen

Probabilistische Algorithmen für SAT

Prof. Dr. Wolfgang Küchlin
Dipl. Inform. Christoph Zengler

Universität Tübingen

05. Juni 2012



SAT als Optimierungsproblem

SAT kann auch als Optimierungsproblem gesehen werden:

- Minimiere die Anzahl der unerfüllten Klauseln

Grundlegende Idee

- Rate eine Zufallsbelegung aller Variablen (vollständige Belegung)
- „Flippe“ wiederholt die Belegung einer Variablen um die Anzahl der unerfüllten Klauseln zu minimieren.

Beispiel

$$F := \{ \{a, b, \neg c\}, \{ \neg a, b \}, \{ \neg b, c \} \}$$

- Rate Belegung: $[a \mapsto \top, b \mapsto \perp, c \mapsto \top]$ (F nicht erfüllt)
- „Flippe“ Variable a auf \perp (F weiterhin nicht erfüllt)
- „Flippe“ Variable b auf \top (F ist jetzt erfüllt)

Eigentliche Optimierung: Welche Variablen werden geflippt

Lokale Suche vs. DPLL

Kriterium	DPLL	Stochastische (Lokale) Suche
Methode	Erweiterung partieller Variablenbelegungen	Optimierung totaler Variablenbelegungen
Vereinfachung	Unit-Propagation, Pure-Literal-Deletion	-
vollständig	ja	nein
Stärken	strukturierte (unerfüllbare) Instanzen (z.B. Verifikation)	erfüllbare Instanzen mit vielen Lösungen (z.B. Planungsprobleme)

Begrifflichkeiten

Betrachte Flip, der Belegung α in Belegung α' überführt

Breakcount

Anzahl Klauseln, die in α erfüllt, aber in α' unerfüllt sind

Makecount

Anzahl Klauseln, die in α unerfüllt, aber in α' erfüllt sind

Diffscore

Anzahl der unerfüllten Klauseln in α minus Anzahl unerfüllte Klauseln in α'

- Alle drei Werte werden für jede Variable nach jedem Flip aktualisiert

Beispiel für Variablenflips

Beispiel (Variablenflips)

$$F := \{\{a, b, \neg c\}, \{\neg a, b\}, \{\neg b, c\}, \{\neg b, \neg d\}, \{a, c, d\}\}$$

Initiale Belegung: $\alpha = [a \mapsto \perp, b \mapsto \top, c \mapsto \perp, d \mapsto \top]$

- Flip von a : $\alpha' = [a \mapsto \top, b \mapsto \top, c \mapsto \perp, d \mapsto \top]$
 $\{\{a, b, \neg c\}, \{\neg a, b\}, \{\neg b, c\}, \{\neg b, \neg d\}, \{a, c, d\}\}$
 Breakcount: 0 / Makecount: 0 / Diffscore : 0
- Flip von b : $\alpha' = [a \mapsto \perp, b \mapsto \perp, c \mapsto \perp, d \mapsto \top]$
 $\{\{a, b, \neg c\}, \{\neg a, b\}, \{\neg b, c\}, \{\neg b, \neg d\}, \{a, c, d\}\}$
 Breakcount: 0 / Makecount: 2 / Diffscore : 2
- Flip von c : $\alpha' = [a \mapsto \perp, b \mapsto \top, c \mapsto \top, d \mapsto \top]$
 $\{\{a, b, \neg c\}, \{\neg a, b\}, \{\neg b, c\}, \{\neg b, \neg d\}, \{a, c, d\}\}$
 Breakcount: 0 / Makecount: 1 / Diffscore : 1
- Flip von d : $\alpha' = [a \mapsto \perp, b \mapsto \top, c \mapsto \perp, d \mapsto \perp]$
 $\{\{a, b, \neg c\}, \{\neg a, b\}, \{\neg b, c\}, \{\neg b, \neg d\}, \{a, c, d\}\}$
 Breakcount: 1 / Makecount: 1 / Diffscore: 0

GSAT Algorithmus

Algorithmus

Algorithm 1: GSAT

```
for  $i = 0$  to  $MAX\_TRIES$  do
   $\alpha$  = random assignment to all variables;
  for  $j = 0$  to  $MAX\_FLIPS$  do
    if  $\alpha$  satisfies all clauses then
      return true
     $x$  = variable that produces the highest diffscore;
    flip  $x$ ;
  return Nothing
```

- MAX_TRIES und MAX_FLIPS müssen so gewählt werden, dass die Fehlerwahrscheinlichkeit sehr gering wird

WalkSAT

Variation von GSAT, Grundalgorithmus bleibt gleich, aber die zu flippende Variable wird anders ausgewählt

Variablenauswahl bei WalkSAT

- Selektiere zufällig eine unerfüllte Klausel C
- Wenn Variable mit **breakcount 0** in C existiert, flippe diese um C zu erfüllen.
- Ansonsten:
 - mit Wahrscheinlichkeit p : wähle Zufallsvariable aus C
 - mit Wahrscheinlichkeit $1 - p$: wähle Variable mit **minimalem breakcount** in C
- WalkSAT läuft meistens viel schneller als GSAT

Hammingkugel-Algorithmus

Idee: Mache die innere Schleife von GSAT deterministisch

- Systematisches Prüfen aller Belegungen in der „Nachbarschaft“ von α

Hammingdistanz zweier Belegungen

- Zwei Belegungen α, β
- Hammingdistanz: $d(\alpha, \beta) := |\{x \mid \alpha(x) \neq \beta(x)\}|$
- Anzahl der Variablen mit unterschiedlicher Belegung in α und β

Beispiel (Hammingdistanz)

$\alpha = (0, 1, 1, 0, 0), \beta = (1, 0, 1, 0, 1), \gamma = (1, 1, 1, 0, 0)$

- $d(\alpha, \beta) = 3, d(\alpha, \gamma) = 1, d(\beta, \gamma) = 2$

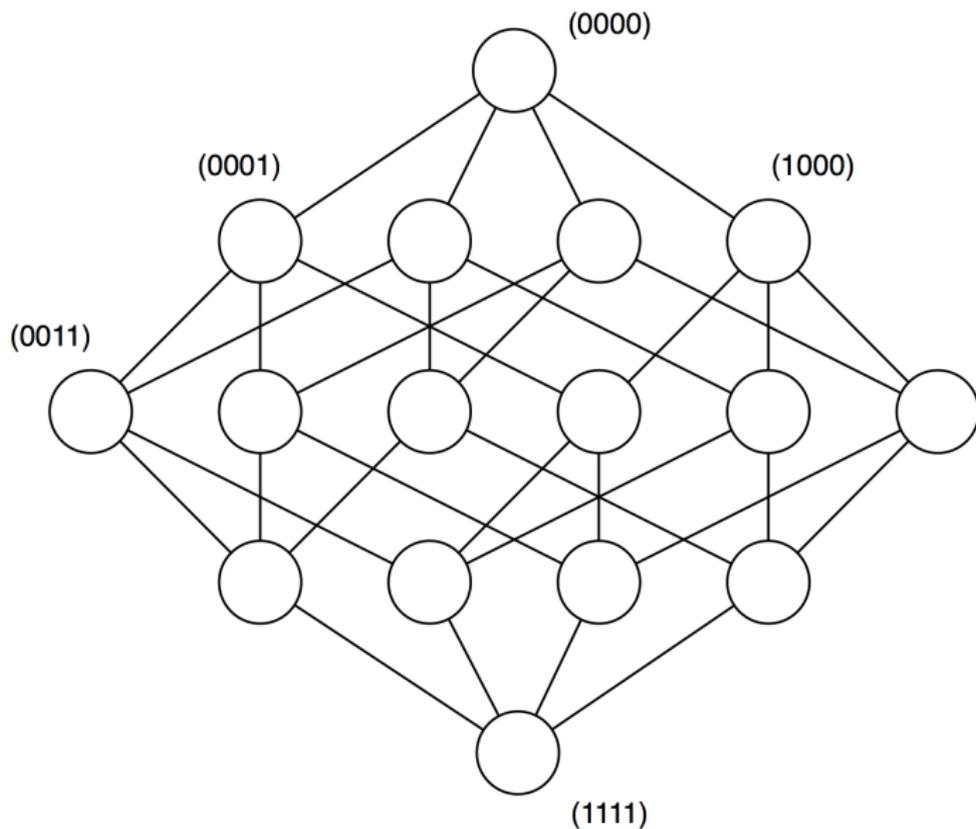
Hammingkugel

Hammingkugel vom Radius r um α :

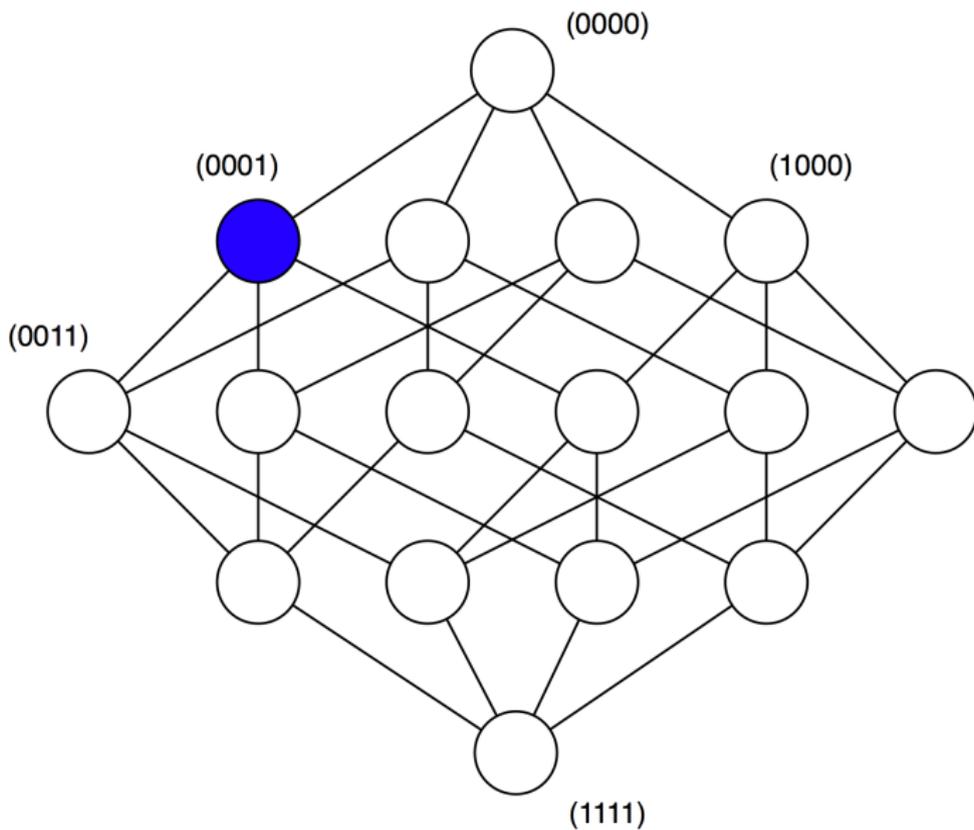
$$H(\alpha, r) := \{\alpha' \mid d(\alpha, \alpha') \leq r\}$$

Alle Belegungen α' , die einen Hammingabstand kleinergleich r von α haben

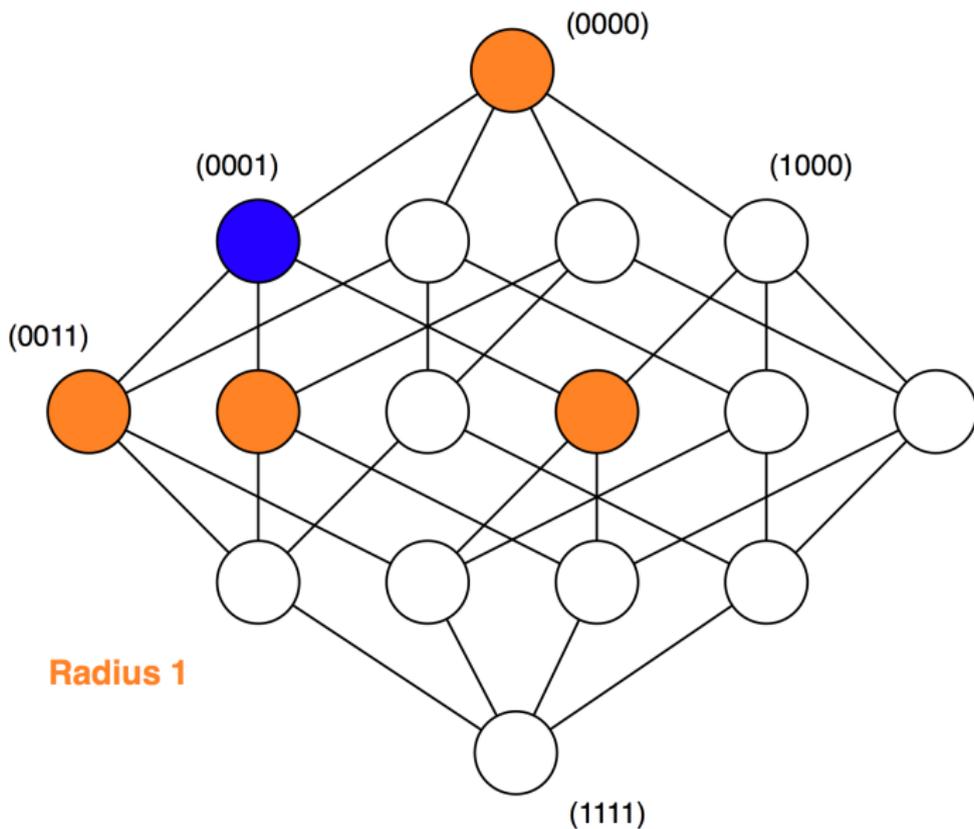
Hammingkugeln



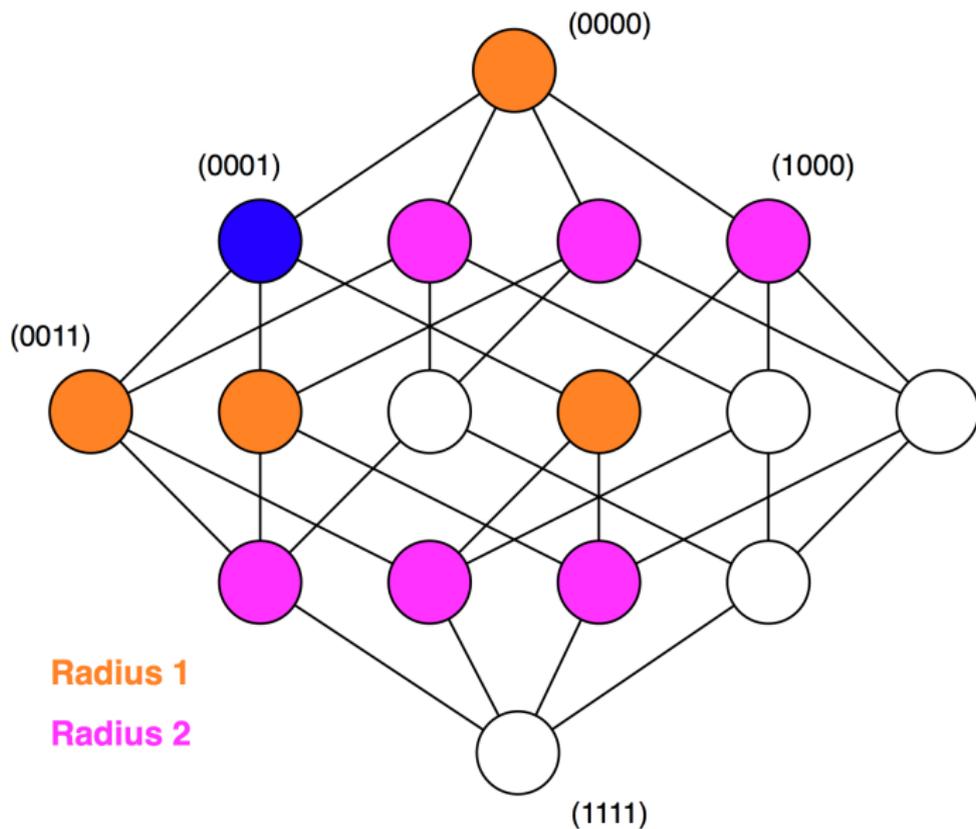
Hammingkugeln



Hammingkugeln



Hammingkugeln



Suchen in einer Hammingkugel

Algorithmus

Methode zum Durchsuchen einer Hammingkugel nach einer erfüllenden Belegung:

Algorithm 2: $\text{test}(\alpha, r)$

if α is a satisfying assignment **then**

└ **return** α

if $r = 0$ **then**

└ **return** *Nothing*

choose unsatisfied clause $C = (x_1 \vee \dots \vee x_n)$;

for $i = 1$ **to** n **do**

┌ $\alpha' = \alpha$ with flipped x_i ;

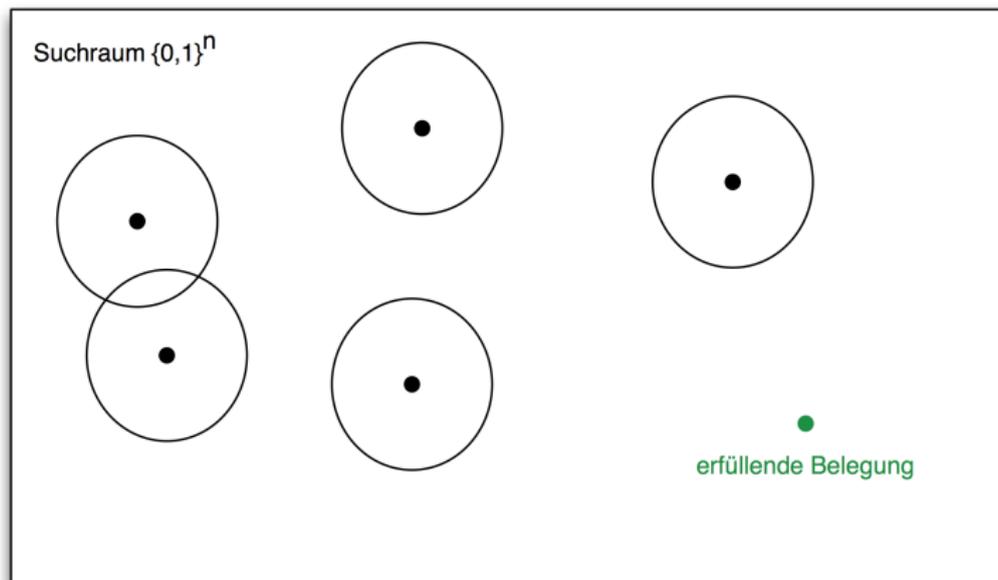
┌ $\beta = \text{test}(\alpha', r - 1)$;

┌ **if** $\beta \neq \text{Nothing}$ **then**

└└ **return** β

return *Nothing*

Visualisierung des Algorithmus



- **Problem:** Passende Anzahl und Wahl der Anfangsbelegungen, so dass Wahrscheinlichkeit, dass keine Hammingkugel eine erfüllende Belegung überdeckt, vernachlässigbar wird.

Derandomisierung der Anfangsbelegungen

Idee: Wähle Anfangsbelegungen nicht zufällig, sondern nach bestimmtem Schema, um möglichst große Teile des Suchraums abzudecken

Verfahren: Überdeckungscode

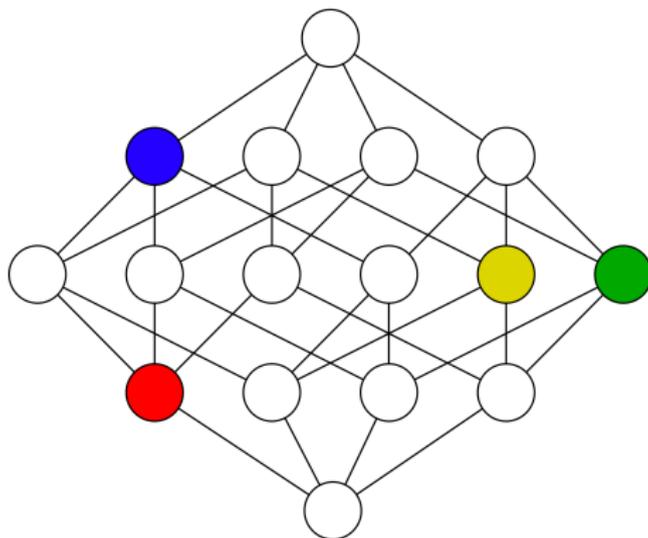
Überdeckungscode

Eine Menge B von Belegungen ist ein Überdeckungscode mit Radius r , falls für jede Belegung α gilt, dass es ein $\beta \in B$ gibt mit $\alpha \in H(\beta, r)$.

Folgerung: Wenn alle Belegungen $\beta \in B$ als Anfangsbelegungen getestet werden und jeweils eine Hammingkugel-Suche mit Radius r vollzogen wird, werden alle möglichen Belegungen getestet (möglicherweise sogar mehrmals wegen Überschneidungen).

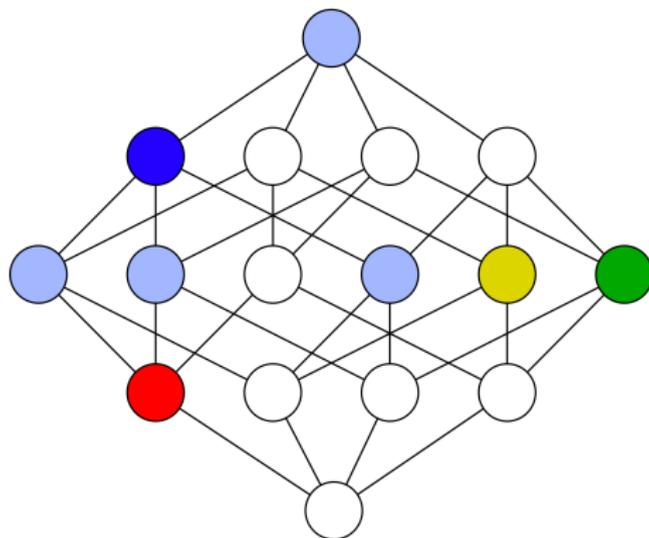
Überdeckungscode - Beispiel 1

Überdeckungscode mit Radius 1:



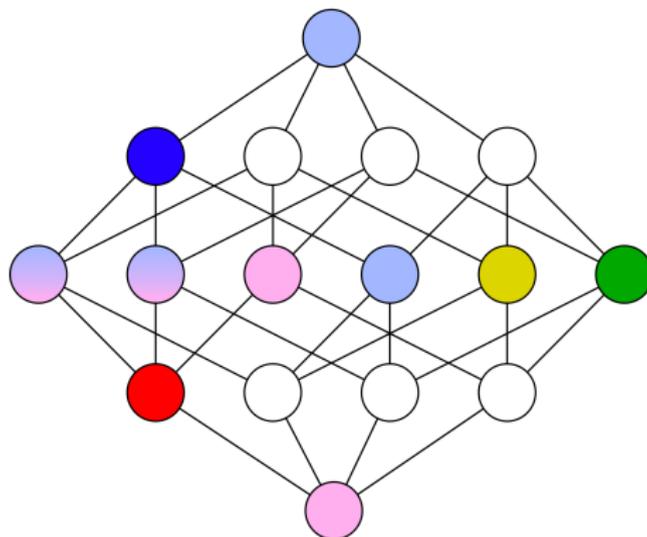
Überdeckungscode - Beispiel 1

Überdeckungscode mit Radius 1:



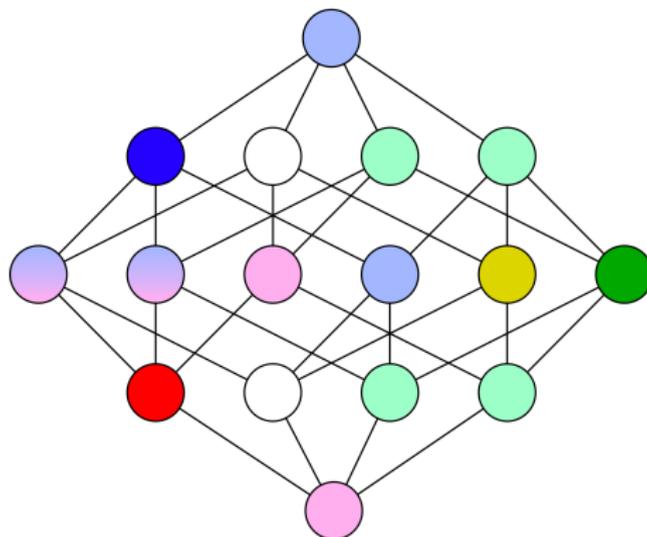
Überdeckungscode - Beispiel 1

Überdeckungscode mit Radius 1:



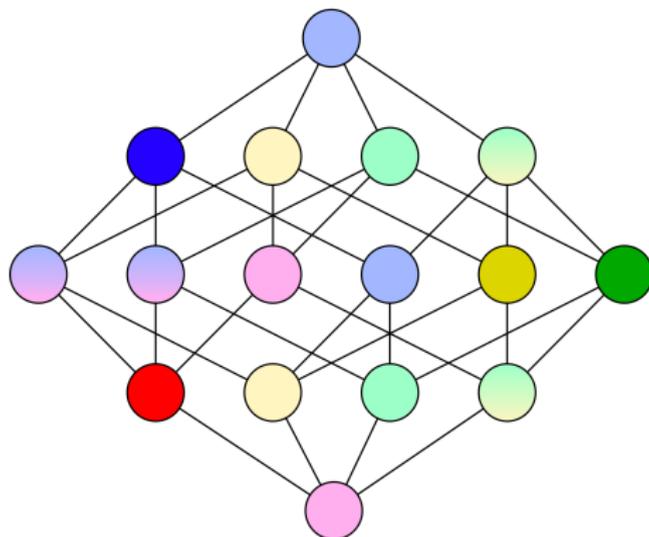
Überdeckungscode - Beispiel 1

Überdeckungscode mit Radius 1:



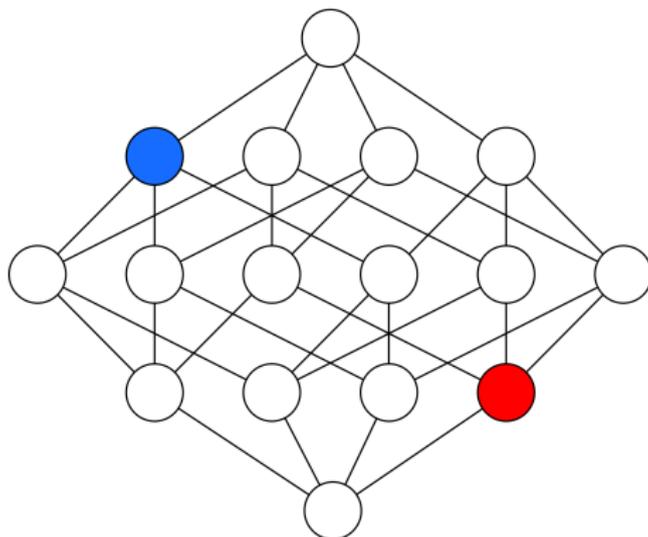
Überdeckungscode - Beispiel 1

Überdeckungscode mit Radius 1:



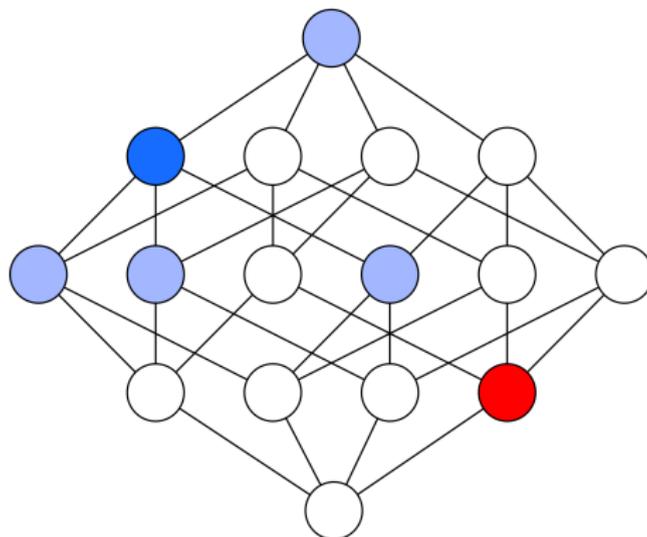
Überdeckungscode - Beispiel 2

Überdeckungscode mit Radius 2:



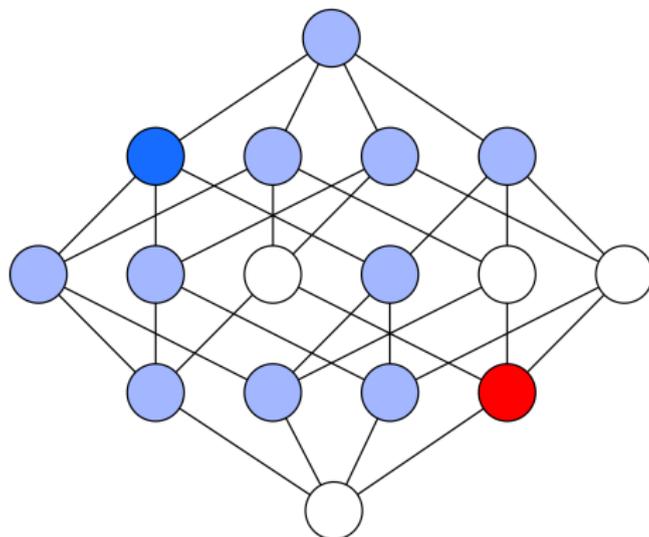
Überdeckungscode - Beispiel 2

Überdeckungscode mit Radius 2:



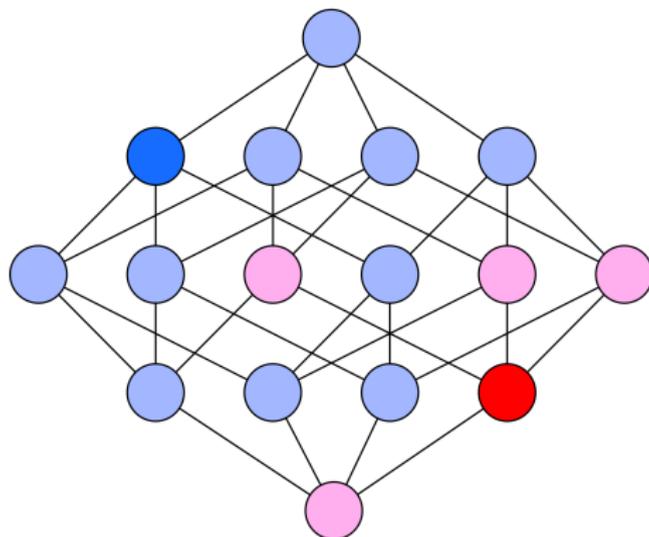
Überdeckungscode - Beispiel 2

Überdeckungscode mit Radius 2:



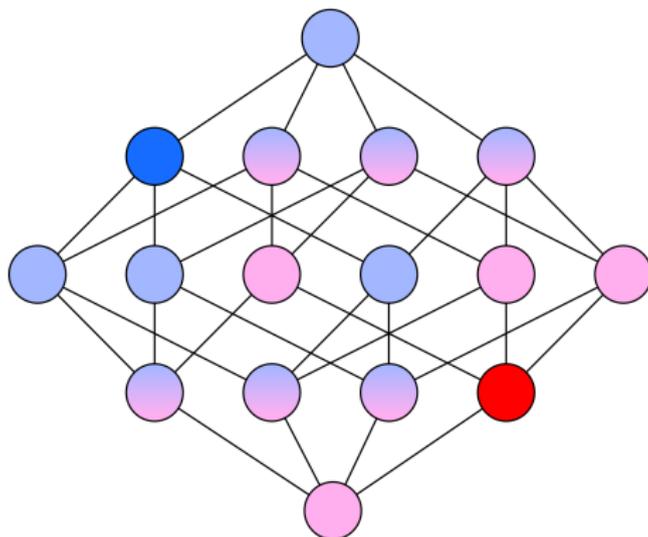
Überdeckungscode - Beispiel 2

Überdeckungscode mit Radius 2:



Überdeckungscode - Beispiel 2

Überdeckungscode mit Radius 2:



Berechnung von minimalen Überdeckungs_codes

Kann zurückgeführt werden auf ein klassisches NP-vollständiges Problem (Karp, 1972): SET-COVER (Mengenüberdeckung)

SET-COVER als Entscheidungsproblem

Gibt es zu einer Menge U und n Teilmengen $S_j \subset U$ und einer natürlichen Zahl $k \leq n$ eine Vereinigung von k oder weniger Teilmengen S_j , so dass $\bigcup_{j=1}^k S_j = U$.

SET-COVER als Optimierungsproblem

Suche ein minimales k und zugehörige Teilmengen $S_j \subset U$, so dass $\bigcup_{j=1}^k S_j = U$.

Bemerkung: SET-COVER wird bei der Minimierung boolescher Ausdrücke (z.B. nach Quine-McCluskey) zur Auswahl einer minimalen Teilmenge von Prim-Implikanten verwendet, die äquivalent zum Ausdruck ist.

SET-COVER

Beispiel (SET-COVER)

- $U = \{1, 2, 3, 4, 5, 6, 7, 8\}$
- 6 Teilmengen: $S_1 = \{1, 3, 5\}$, $S_2 = \{2, 4, 8\}$, $S_3 = \{1, 3, 4, 6\}$, $S_4 = \{7, 8\}$, $S_5 = \{3, 4, 6\}$, $S_6 = \{2, 6, 7\}$

Verschiedene Überdeckungen:

- $S_1 \cup S_2 \cup S_6$
- $S_1 \cup S_3 \cup S_4 \cup S_6$
- $S_1 \cup S_2 \cup S_4 \cup S_5$

Analogie zu Überdeckungs-codes:

- Menge U = Menge aller möglichen Belegungen α
- Teilmengen S_j = Alle Hammingkugeln vom Radius r
- Gesuchte minimale Teilmenge = Anfangsbelegungen des Algorithmus

Greedy Algorithmus zur Berechnung von SET-COVER

Algorithmus

- 1 Setze $B = \emptyset$
- 2 Solange es noch unüberdeckte Elemente in U gibt:
 - 1 Wähle ein S_j , das möglichst viele noch unüberdeckte Elemente aus U enthält
 - 2 $B = B \cup \{S_j\}$
- 3 Falls $\bigcup B = U$, so ist B die Lösung, ansonsten gibt es keine

Beispiel (Greedy Algorithmus)

- $U = \{1, 2, 3, 4, 5, 6, 7, 8\}$
- 6 Teilmengen:
 $S_1 = \{1, 3, 5\}, S_2 = \{2, 4, 8\}, S_3 = \{1, 3, 4, 6\}, S_4 = \{7, 8\}, S_5 = \{3, 4, 6\}, S_6 = \{2, 6, 7\}$
- 1 Wähle S_3 , dann bleibt noch: $\{2, 5, 7, 8\}$
- 2 Wähle S_4 , dann bleibt noch: $\{2, 5\}$
- 3 Wähle S_1 , dann bleibt noch: $\{2\}$
- 4 Wähle S_2 , dann bleibt noch: \emptyset

$B = \{S_3, S_4, S_1, S_2\}$ (aber nicht optimal)

Zusammenfassung

GSAT Algorithmus

Algorithm 3: GSAT

```
for  $i = 0$  to  $MAX\_TRIES$  do
   $\alpha =$  random assignment to all variables;
  for  $j = 0$  to  $MAX\_FLIPS$  do
    if  $\alpha$  satisfies all clauses then
      return true
     $x =$  variable that produces the highest diffscore;
    flip  $x$ ;
  return Nothing
```

- Ersetze innere Schleife durch Hammingkugel-Suche (deterministisch)
 - Ersetze äußere Schleife durch Finden eines minimalen Überdeckungs_codes mit Greedy Algorithmus (deterministisch)
- Derandomisierter Algorithmus (nach Dantsin et al.), der beweisbare obere Schranke für 3-SAT von 1.481^n hat.