

SAT Solving und Anwendungen

MaxSAT und Pseudo Boolesche Optimierung

Prof. Dr. Wolfgang Küchlin
Rouven Walter, M.Sc. Informatik

Universität Tübingen

11. Januar 2018



Motivierendes Beispiel: Minimum Vertex Cover

Minimum Vertex Cover

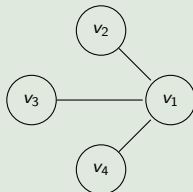
Gegeben:

- Graph $G = (V, E)$

Gesucht:

- Minimum Vertex Cover
 - Vertex Cover $U \subseteq V$: Für jede Kante $(v_i, v_j) \in E$ gilt $v_i \in U$ oder $v_j \in U$
 - Minimum Vertex Cover: Vertex Cover U mit minimaler Größe

Beispiel



Motivierendes Beispiel: Minimum Vertex Cover

Minimum Vertex Cover

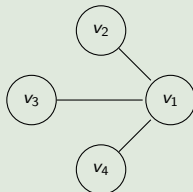
Gegeben:

- Graph $G = (V, E)$

Gesucht:

- Minimum Vertex Cover
 - Vertex Cover $U \subseteq V$: Für jede Kante $(v_i, v_j) \in E$ gilt $v_i \in U$ oder $v_j \in U$
 - Minimum Vertex Cover: Vertex Cover U mit minimaler Größe

Beispiel



- Mögliches Vertex Cover:
 $\{v_2, v_3, v_4\}$
- **Minimales Vertex Cover ???**

Motivierendes Beispiel: Minimum Vertex Cover

Minimum Vertex Cover

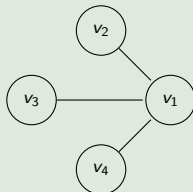
Gegeben:

- Graph $G = (V, E)$

Gesucht:

- Minimum Vertex Cover
 - Vertex Cover $U \subseteq V$: Für jede Kante $(v_i, v_j) \in E$ gilt $v_i \in U$ oder $v_j \in U$
 - Minimum Vertex Cover: Vertex Cover U mit minimaler Größe

Beispiel



- Mögliches Vertex Cover: $\{v_2, v_3, v_4\}$
- Minimales Vertex Cover: $\{v_1\}$

Fahrplan für heute

① Boolesche Optimierung

- MaxSAT
- Pseudo-Boolesche Optimierung (PBO)
- Konversion zwischen beiden Problemen

② Beispielanwendungen

- Software Package Konfiguration

③ Techniken

- Cardinality Constraints
- Pseudo-Boolesche Constraints

④ Praktische Algorithmen

- Branch & Bound
- Iteratives SAT Solving
- Core-Guided Algorithmen

Was ist MaxSAT?

Beispiel

$$x_6 \vee x_2$$

$$\neg x_6 \vee x_2$$

$$\neg x_2 \vee x_1$$

$$\neg x_1$$

$$\neg x_6 \vee x_8$$

$$x_6 \vee \neg x_8$$

$$x_2 \vee x_4$$

$$\neg x_4 \vee x_5$$

$$x_7 \vee x_5$$

$$\neg x_7 \vee x_5$$

$$\neg x_5 \vee x_3$$

$$\neg x_3$$

Ist diese Klauselmenge erfüllbar?

Was ist MaxSAT?

Nein!

$$x_6 \vee x_2$$

$$\neg x_6 \vee x_2$$

$$\neg x_2 \vee x_1$$

$$\neg x_1$$

$$\neg x_6 \vee x_8$$

$$x_6 \vee \neg x_8$$

$$x_2 \vee x_4$$

$$\neg x_4 \vee x_5$$

$$x_7 \vee x_5$$

$$\neg x_7 \vee x_5$$

$$\neg x_5 \vee x_3$$

$$\neg x_3$$

Formel ist unerfüllbar!

MaxSAT:

- Finde eine Belegung, welche die Anzahl erfüllter Klauseln maximiert
- In obiger Formel können maximal **10 Klauseln** gleichzeitig erfüllt sein
- Viele Varianten von MaxSAT

Varianten von MaxSAT — 1

MaxSAT

- Alle Klauseln sind weich (soft), d.h. müssen nicht unbedingt erfüllt werden.
- Maximiere Anzahl der erfüllten weichen Klauseln
- Minimiere Anzahl der unerfüllten weichen Klauseln

Partial MaxSAT

- Es gibt harte Klauseln, die erfüllt sein müssen
- Minimiere Anzahl der unerfüllten weichen Klauseln

Anwendungsbeispiel: Softwarekonfiguration

- Harte Klauseln: Bestimmte Pakete/Plugins müssen installiert werden/bleiben
- Weiche Klauseln: Zusätzliche Pakete/Plugins mit Abhängigkeiten
- Frage: Wie viele und welche Pakete können maximal zusätzlich installiert werden?

Varianten von MaxSAT — 2

Weighted MaxSAT

- Alle Klauseln sind weich
- Alle Klauseln werden mit Gewichten versehen
- Minimiere die Summe der Gewichte der **unerfüllten** Klauseln

Weighted Partial MaxSAT

- Es gibt harte Klauseln, die **erfüllt** sein müssen
- Weiche Klauseln werden mit Gewichten versehen
- Minimiere die Summe der Gewichte der **unerfüllten weichen** Klauseln

Anwendungsbeispiel: PC Konfiguration

- Harte Klauseln: Ein PC kann nur ein Motherboard haben, nur bestimmte Prozessoren funktionieren mit bestimmten Motherboards, ...
- Weiche Klauseln: Spezielle Kundenwünsche mit Priorisierung (Am liebsten 8GB Ram, wenn am Ende noch möglich ein Diskettenlaufwerk, ...)
- Frage: Optimal mögliche PC Konfiguration?

Notation

Notation: gewichtete Klausel

(c, w) : gewichtete Klausel

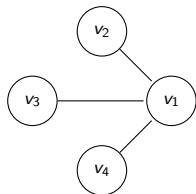
- c ist eine Menge an Literalen (Klausel)
- w ist ein nicht-negativer Integerwert oder ∞ (oder \top)
 - Kosten (Strafe), wenn man c nicht erfüllt

Notation: Klauselmenge

φ : Menge gewichteter Klauseln

- **Weiche Klauseln:** (c, w) mit $w < \infty$
 - Kosten, wenn man c nicht erfüllt
- **Harte Klauseln:** (c, ∞)
 - Klausel c muss erfüllt werden

Modellierungsbeispiel



Minimum Vertex Cover

- Vertex Cover $U \subseteq V$: Für jede Kante $(v_i, v_j) \in E$ gilt $v_i \in U$ oder $v_j \in U$
- Minimum Vertex Cover: Vertex Cover U mit minimaler Größe

Beispiel

Partial MaxSAT Codierung

- Variablen: x_i für jeden Knoten $v_i \in C$, mit $x_i = 1$, genau dann wenn $v_i \in U$
- **Harte Klauseln:** $(x_i \vee x_j)$ für jede Kante $(v_i, v_j) \in E$ (... ist Vertex Cover)
- **Weiche Klauseln:** $(\neg x_i)$ für jeden Knoten $v_i \in V$ (... ist minimal)
 - Je mehr Variablen auf 0 gesetzt werden, desto kleiner das Cover Set

Codierung:

- $\varphi_H = \{(x_1 \vee x_2), (x_1 \vee x_3), (x_1 \vee x_4)\}$
- $\varphi_S = \{(\neg x_1), (\neg x_2), (\neg x_3), (\neg x_4)\}$

MaxSAT vs MinUNSAT

MinUNSAT

- Finde Belegung, welche die Anzahl U der unerfüllten Klauseln minimiert
 - diese Belegung maximiert dann auch die Anzahl erfüllter Klauseln
- $\text{MinUNSAT}(\varphi) := U$.
- Varianten Partial, Weighted und Partial Weighted möglich (wie bei MaxSAT)
 - Weighted MinUNSAT: minimiere das Gewicht der unerfüllten Klauseln

Zusammenhang

Sei φ Klauselmenge, dann gilt:

$$|\varphi| = \text{MaxSAT}(\varphi) + \text{MinUNSAT}(\varphi)$$

Hinweis: Zusammenhang gilt auch für alle Varianten

- Oft werden Algorithmen für MinUNSAT angegeben, welche aufgrund des Zusammenhangs auch für MaxSAT benutzt werden können

MaxSAT vs MinUNSAT — Beispiel

Beispiel

- Sei $\varphi = \text{Hard} \dot{\cup} \text{Soft}$ mit

$$\text{Hard} = \{\{x\}\}$$

$$\text{Soft} = \{(\{\neg x\}, 2), (\{y\}, 6), (\{\neg x, \neg y\}, 5)\}$$

- $\text{PartialWeightedMaxSAT}(\varphi) = 6$
Belegung: $x \mapsto 1, y \mapsto 1$
- $\text{PartialWeightedMinUNSAT}(\varphi) = 7$
Belegung: $x \mapsto 1, y \mapsto 1$
- Es gilt:

$$13 = \text{PartialWeightedMaxSAT}(\varphi) + \text{PartialWeightedMinUNSAT}(\varphi)$$

Pseudo-Boolesche Constraints & Optimierung

Pseudo-Boolesche Constraints

- Boolesche Variablen: x_1, \dots, x_n
- Lineare Ungleichungen

$$\sum_{i \in N} a_i \ell_i \geq b, \quad \ell_i \in \{x_i, \bar{x}_i\}, x_i \in \{0, 1\}, a_i, b \in \mathbb{N}_0$$

Pseudo-Boolesche Optimierung

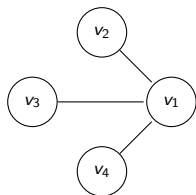
Minimiere

$$\sum_{h \in N} w_h \cdot x_h$$

bezüglich n PB Constraints

$$\sum_{i \in N} a_{ij} \ell_i \geq b_j, \quad j \in \{1 \dots n\}, \ell_i \in \{x_i, \bar{x}_i\}, x_i \in \{0, 1\}, a_{ij}, b_j \in \mathbb{N}_0$$

Modellierungsbeispiel



Minimum Vertex Cover

- Vertex Cover $U \subseteq V$: Für jede Kante $(v_i, v_j) \in E$ gilt $v_i \in U$ oder $v_j \in U$
- Minimum Vertex Cover: Vertex Cover U mit minimaler Größe

Codierung als PBO

- Variablen: x_i für jeden Knoten $v_i \in V$, mit $x_i = 1$ wenn $v_i \in U$
- PB Constraints: $x_i + x_j \geq 1$ für jede Kante $(v_i, v_j) \in E$
- Zielfunktion: Minimiere die Anzahl an Variablen, die auf 1 gesetzt sind
 \Rightarrow D.h. minimiere Anzahl an Knoten in der Vertex Cover

Probleminstanz: $x_1 + x_2 \geq 1, x_1 + x_3 \geq 1, x_1 + x_4 \geq 1$

Zielfunktion: Minimiere $x_1 + x_2 + x_3 + x_4$

Blockieren und Selektieren von Klauseln

Manchmal möchte man in einer CNF Formel φ einige Klauseln selektiv „ausschalten“ oder „einschalten“. In einem unerfüllbaren φ möchte man maximal viele Klauseln einschalten und minimal viele Klauseln ausschalten.

Ausschalten: Blockiervariablen

- Ersetze jede zu blockierende Klausel c_i durch $c'_i = c_i \cup \{b_i\}$
- b_i ist eine neue Variable (Blockiervariable, *blocking variable*)
- Ist b_i mit *true* belegt, dann ist c_i in der Wirkung blockiert (muss nicht mehr erfüllt werden)

Einschalten: Selektorvariablen

- Ersetze jede Klausel c_i durch $c'_i = c_i \cup \{\neg s_i\}$
- s_i ist eine neue Variable (Selektionsvariable, *selector variable*)
- Ist s_i mit *true* belegt, dann ist c_i in der Wirkung aktiviert (muss erfüllt werden); andernfalls ist c_i in der Wirkung blockiert.

Offensichtlich braucht man nur entweder Selektor- oder Blockiervariablen. Man verwendet jeweils den natürlichen Zugang zum Problem.

Von (reinem) MaxSAT zu PBO

Ausgehend von einer CNF Formel φ :

① Generiere φ' aus φ :

- Füge zu jeder Klausel c_i eine Blockiervariable b_i hinzu: $c'_i = c_i \cup \{b_i\}$
- Problem ist jetzt lösbar, indem alle b_i auf 1 gesetzt werden
- Ziel ist es, möglichst wenige der b_i zu „verwenden“ (auf 1 zu setzen).

② Minimiere nun Zielfunktion $\sum b_i$

Beispiel

- CNF Formel φ :

$$\varphi = \{\{x_1, \neg x_2\}, \{x_1, x_2\}, \{\neg x_1\}\}$$

- Modifizierte Formel φ' :

$$\varphi = \{\{x_1, \neg x_2, b_1\}, \{x_1, x_2, b_2\}, \{\neg x_1, b_3\}\}$$

- PB Constraints: $x_1 + \bar{x}_2 + b_1 \geq 1$, $x_1 + x_2 + b_2 \geq 1$, $\bar{x}_1 + b_3 \geq 1$
- Zielfunktion zu minimieren: $b_1 + b_2 + b_3$

Von Partial (Weighted) MaxSAT zu PBO

Ausgehend von einer Partial (Weighted) MaxSAT Instanz mit φ_H und φ_S

Generiere PBO Instanz:

minimiere $\sum w_i b_i$, so dass φ_T gilt, mit

- $\varphi_T = \varphi'_H \cup \varphi'_S$
- Jede harte Klausel (c, ∞) wird auf eine Klausel c in φ'_H abgebildet
- Jede weiche Klausel (c, w) wird auf eine Klausel $(c_i \vee b_i)$ abgebildet und der Term $w_i b_i$ wird zur Zielfunktion addiert

Beispiel

- Originalproblem: $(\{x, y, \neg z\}, \infty), (\{x, \neg y\}, 4), (\{\neg x\}, 8), (\{x, z\}, 2)$
- $\varphi_T = \underbrace{(x, y, \neg z)}_{\varphi'_H}, \underbrace{(x, \neg y, b_1), (\neg x, b_2), (x, z, b_3)}_{\varphi'_S}$
- PB Constraints: $x + y + \bar{z} \geq 1, x + \bar{y} + b_1 \geq 1, \bar{x} + b_2 \geq 1, x + z + b_3 \geq 1$
- Zielfunktion zu minimieren: $4b_1 + 8b_2 + 2b_3$

Anwendung: Software Package Upgrades

- Mögliche Software Pakete: $P = \{p_1, \dots, p_n\}$
- Variable x_i für jedes Paket $p_i \in P$. $x_i = 1$, gdw. p_i installiert ist
- Constraints für jedes Paket p_i : (p_i, D_i, C_i)
 - D_i : Abhängigkeiten (**required packages**) beim Installieren von p_i
 - C_i : Konflikte (**disallowed packages**) beim Installieren von p_i
- Beispielproblem: **Maximum Installability**
 - Maximale Anzahl an Paketen, die installiert werden können
 - Paket Constraints sind **harte** Klauseln
 - Jedes einzelne Paket ist eine **weiche** Klausel

Beispiel

Paket Constraints

$(p_1, \{p_2 \vee p_3\}, \{p_4\})$
 $(p_2, \{p_3\}, \{p_4\})$
 $(p_3, \{p_2\}, \emptyset)$
 $(p_4, \{p_2 \wedge p_3\}, \emptyset)$

MaxSAT Codierung

$\varphi_H = \{(\neg x_1 \vee x_2 \vee x_3), (\neg x_1 \vee \neg x_4),$
 $(\neg x_2 \vee x_3), (\neg x_2 \vee \neg x_4), (\neg x_3 \vee x_2),$
 $(\neg x_4 \vee x_2), (\neg x_4 \vee x_3)$
 $\varphi_S = \{(x_1), (x_2), (x_3), (x_4)\}$

Cardinality Constraints

Fragestellung: Wie behandelt man Cardinality Constraints

- Allgemeine Form: $\sum_{j=1}^n x_j \bowtie k$ mit $\bowtie \in \{<, \leq, =, \geq, >\}$
- Im Speziellen AtMost1 Constraints: $\sum_{j=1}^n x_j \leq 1$

Lösung 1

Benutze speziellen PB Solver

- Schwer, mit Fortschritten im SAT Bereich mitzuhalten
- Für SAT/UNSAT codieren die besten Solver bereits in CNF
 - Z.B. Minisat+, QMaxSat, MSUnCore, (W)PM2

Lösung 2

- Codiere Cardinality Constraints in CNF
- Benutze SAT solver

Equals, AtLeast1 & AtMost1 Constraints

Spezielle Behandlung von Constraints mit 1 auf der rechten Seite — kommen sehr häufig vor:

- Fahrzeug muss genau einen Motor haben ...
- Genau ein Grafikkartentreiber muss ausgewählt sein ...
- Höchstens ein SAT Solver kann in Eclipse installiert sein ...

Kodierungen

- $\sum_{j=1}^n x_j = 1$: Kodiere mit $(\sum_{j=1}^n x_j \leq 1) \wedge (\sum_{j=1}^n x_j \geq 1)$
- $\sum_{j=1}^n x_j \geq 1$: Kodiere mit $(x_1 \vee x_2 \vee \dots \vee x_n)$
- $\sum_{j=1}^n x_j \leq 1$: Kodiere mit:
 - Paarweise Kodierung:
 - $(\neg x_1 \vee \neg x_2), (\neg x_1 \vee \neg x_3), (\neg x_1 \vee \neg x_4), \dots, (\neg x_2 \vee \neg x_3), (\neg x_2 \vee \neg x_4), \dots, (\neg x_3 \vee \neg x_4), \dots$
 - Anzahl Klauseln in $\mathcal{O}(n^2)$; keine Hilfsvariablen
 - Sequentieller Zähler
 - Anzahl Klauseln in $\mathcal{O}(n)$; Hilfsvariablen in $\mathcal{O}(n)$
 - Bitweise Kodierung:
 - Anzahl Klauseln in $\mathcal{O}(n \log n)$; Hilfsvariablen in $\mathcal{O}(\log n)$

Allgemeine Cardinality Constraints

Allgemeine Form: $\sum_{j=1}^n x_j \leq k$ (oder $\sum_{j=1}^n x_j \geq k$)

Kodierungen

- Sequentielle Zähler
 - Klauseln/Variablen: $\mathcal{O}(nk)$
- BDDs
 - Klauseln/Variablen: $\mathcal{O}(nk)$
- Sortiernetzwerke
 - Klauseln/Variablen: $\mathcal{O}(n \log^2 n)$
- Cardinality Netzwerke
 - Klauseln/Variablen: $\mathcal{O}(n \log^2 k)$

Pseudo-Boolesche Constraints

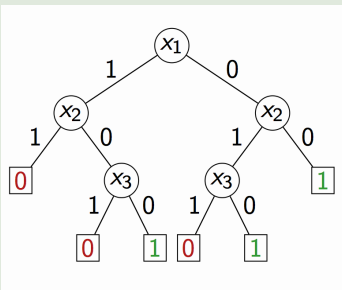
Allgemeine Form: $\sum_{j=1}^n a_j x_j \leq b$

- Kodierung z.B. mit BDD (worst case: Exponentielle Anzahl an Klauseln!)

Beispiel

$$3x_1 + 3x_2 + x_3 \leq 3$$

- Kodiere BDD, d.h. analysiere Variablen durch Abziehen der Koeffizienten
- Konvertiere BDD in CNF
- Simplifiziere



Pseudo-Boolesche Constraints

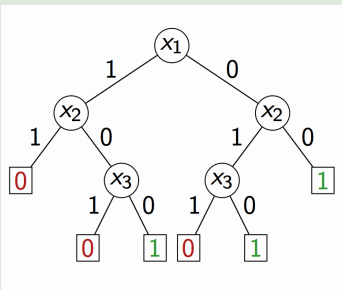
Allgemeine Form: $\sum_{j=1}^n a_j x_j \leq b$

- Kodierung z.B. mit BDD (worst case: Exponentielle Anzahl an Klauseln!)

Beispiel

$$3x_1 + 3x_2 + x_3 \leq 3$$

- Kodiere BDD, d.h. analysiere Variablen durch Abziehen der Koeffizienten
- Konvertiere BDD in CNF
- Simplifiziere



- Konversionsformel am Knoten x :

$$F \equiv (\neg x \vee F|_{x=1}) \wedge (x \vee F|_{x=0})$$
- ergibt: $(\neg x_1 \vee \neg x_2) \wedge$

$$(\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$$

Branch & Bound Algorithmen — 1

- Betrachte MinUNSAT (die Minimierungs-Variante von MaxSAT), die die Anzahl unerfüllter Klauseln uc minimiert.
- Exploriere den Suchraum aller Belegungen und betrachte zu jedem Zeitpunkt die Obergrenze $UB \geq uc$ und die Untergrenze $LB \leq uc$ für die Anzahl unerfüllter Klauseln uc , also insgesamt $UB \geq uc \geq LB$.
- UB ist der beste (globale) Wert von uc , der für eine der bisher betrachteten vollständigen Belegungen erzielt wurde – schlechter wird das Ergebnis nicht.
- Wir senken durch weitere Suche die Obergrenze UB bis zu einem Minimum.
- Die Untergrenze LB ist eine (konservative) Abschätzung der momentanen (lokalen) Situation im Suchraum: besser kann das Ergebnis ausgehend von der momentanen partiellen Belegung π nicht mehr werden.
- LB ist die Summe aus den unter π schon leeren Klauseln plus einer minimalen Anzahl von Klauseln, die zusätzlich leer werden, egal wie π weiter vervollständigt wird.
- Vereinfache das Problem durch Inferenzregeln, um Suchraum einzuschränken. Die Vereinfachungen müssen für jede Belegung die Anzahl unerfüllter Klauseln unverändert lassen.

Branch & Bound Algorithmen — 2

Grundlegender B&B Algorithmus

Gegeben MinUNSAT Instanz φ

- B&B exploriert den Suchbaum, der den Suchraum aller möglichen Belegungen von φ repräsentiert, in Tiefensuche
- An jedem Knoten:
 - **UB** Upper Bound/Obergrenze: die beste Lösung für die Anzahl unerfüllter Klauseln, die bisher für eine komplette Belegung gefunden wurde, also $UB \geq uc$
 - **LB** Lower Bound/Untergrenze: Summe der Anzahl der Klauseln, die mit der aktuellen partiellen Belegung unerfüllt sind + Minimum *min* der Anzahl an Klauseln, die noch unerfüllt werden, wenn die aktuelle Belegung komplettiert wird. *LB* ist also ein Schätzwert (underestimation) mit $uc \geq LB$.
- Vergleiche an jedem Knoten *UB* mit *LB*
 - $LB \geq UB$: Es muss nicht mehr weitergesucht werden; Backtracking zu einem höheren Level; Untersuchung eines anderen Teilbaumes
 - $LB < UB$: Versuche bessere Lösung zu finden; Belege eine weitere Variable
- **Finale Lösung:** Wert von *UB* wenn der ganze Suchbaum exploriert wurde

Der Branch & Bound Algorithmus

Der Algorithmus

Algorithm 1: $BnB(\varphi, UB)$

Input: MinUNSAT Instanz φ , Obergrenze UB

$\varphi = \text{simplifyFormula}(\varphi)$

if $\varphi = \emptyset$ *oder* φ *enthält nur leere Klauseln* **then**

└ **return** $\#emptyClauses(\varphi)$

$LB = \#emptyClauses(\varphi) + \text{underestimation}(\varphi)$

if $LB \geq UB$ **then**

└ **return** UB

$x = \text{selectVariable}(\varphi)$

$UB = \min(UB, BnB(\varphi_x, UB))$

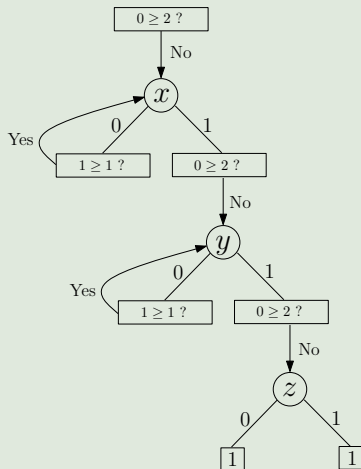
return $\min(UB, BnB(\varphi_{\bar{x}}, UB))$

- UB ist initial die Gesamtzahl an Klauseln oder die Anzahl derjenigen Klauseln, die durch eine beliebige Belegung unerfüllt sind
- **Noch offen (s.u.):** Welche Methoden zur Formel-Vereinfachung gibt es? Wie wird $\text{underestimation}(\varphi)$ implementiert?

Der Branch & Bound Algorithmus — Beispiel

Beispiel

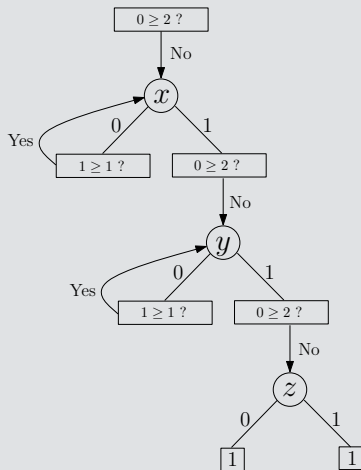
- $\varphi = \{\{x\}, \{\neg x \vee y\}, \{z \vee \neg y\}, \{x \vee z\}, \{\neg z\}\}$
- Initiale obere Grenze: $UB = 2$
(alle Variablen auf false gesetzt)
- Keine Formelvereinfachung
- Berechnung LB nur als $\#emptyClauses(\varphi)$
d.h. hier immer $underestimation(\varphi) = 0$
- Variablenordnung: x, y, z
- Ergebnis: 1



Der Branch & Bound Algorithmus — Beispiel

Trace

- $\varphi = \{\{x\}, \{\neg x \vee y\}, \{z \vee \neg y\}, \{x \vee z\}, \{\neg z\}\}$
- $BnB(\varphi, 2) : LB = 0$; call $BnB(\varphi_x, 2)$ with
- $\varphi_x = \{\{y\}, \{z \vee \neg y\}, \{\neg z\}\}$:
 $LB = 0$; call $BnB(\varphi_{xy}, 2)$ with
- $\varphi_{xy} = \{\{z\}, \{\neg z\}\}$:
 $LB = 0$; call $BnB(\varphi_{xyz}, 2)$ with:
- $\varphi_{xyz} = \{\{\}\}$: *return*(1) to $BnB(\varphi_{xy}, 2)$:
- $UB = \min(2, 1)$; call $BnB(\varphi_{xy\bar{z}}, 1)$ with:
- $\varphi_{xy\bar{z}} = \{\{\}\}$: *return*(1) to $BnB(\varphi_{xy}, 2)$:
- $UB = \min(1, 1)$; call $BnB(\varphi_{x\bar{y}}, 1)$ with:
- $\varphi_{x\bar{y}} = \{\{\}, \{\neg z\}\}$: $LB = 1 \geq UB$; *return*(1)
to $BnB(\varphi, 2)$; call $BnB(\varphi_{\bar{x}}, 1)$ with:
- $\varphi_{\bar{x}} = \{\{\}, \{z \vee \neg y\}, \{z\}, \{\neg z\}\}$:
 $LB = 1 \geq UB$; *return*(1) to $BnB(\varphi, 2)$;
return(1)



Wie findet man eine gute Unterbewertung?

Effiziente Berechnung der Lower Bound ausgehend von partieller Belegung π :

- Einfach: Zähle alle Klauseln, die durch π falsifiziert sind
- Besser: Schätze durch untere Schranke ab, wie viele Klauseln in jedem Fall noch falsifiziert werden, egal wie π komplettiert wird.

Basisvariante nach Wallace und Freuder

$$LB(\varphi) = \# \text{emptyClauses}(\varphi) + \sum_{x \text{ occurs in } \varphi} \min(ic(x), ic(\bar{x}))$$

- φ : CNF mit aktueller partiellen Belegung
- $ic(x)$ bzw. $ic(\bar{x})$: Inconsistency Count. Anzahl der Unit Klauseln von φ , die x bzw. \bar{x} enthalten.

Star Rule

Suche Teilmenge der Art $\{\{\ell_1\}, \{\ell_2\}, \dots, \{\ell_k\}, \{\bar{\ell}_1 \vee \bar{\ell}_2 \vee \dots \vee \bar{\ell}_k\}\}$. Jede solche Klauselmengende erzwingt für jede Belegung mindestens *eine* unerfüllte Klausel.

Weitere Regeln existieren.

Formel-Vereinfachung durch Inferenzregeln — 1

- Transformiere ein MaxSAT Problem φ in ein äquivalentes Problem φ' , das dieselbe Anzahl unerfüllter Klauseln für jede Belegung hat. Eine solche Transformation ist *sound*.
- Boolesche Äquivalenz oder Erfüllbarkeitsäquivalenz genügen nicht.

Beispiel

- $\text{MaxSAT}(\{\{x\}\}) \neq \text{MaxSAT}(\{\{x\}, \{x\}\})$.
 - $\text{MinUNSAT}(\{\{x\}, \{\neg x\}\}) \neq \text{MinUNSAT}(\{\{x\}, \{x\}, \{\neg x\}, \{\neg x\}\})$.
 - $\text{MinUNSAT}(\{\{\}\}) \neq \text{MinUNSAT}(\{\{\}, \{\}\})$
-
- Man arbeitet mit Transformationsregeln, die eine Teilmenge der Klauseln durch eine andere Teilmenge ersetzen. Das Hinzufügen von Klauseln verändert i.A. das Ergebnis: $\text{MaxSAT}(\{\{x\}\}) \neq \text{MaxSAT}(\{\{x\}, \{x\}\})$.
 - Es sind nur weniger und schwächere Inferenzen möglich als bei DPLL.

Formel-Vereinfachung durch Inferenzregeln — 2

Ungültige Inferenzen

- **Unit Propagation:** $\{\{x_1\}, \{\neg x_1\}, \{\neg x_1\}\}$. Je nach UP-Sequenz erhalten wir $\{\{x_1\}, \{\}, \{\}\}$ (uc=2) oder $\{\{\}, \{\neg x_1\}, \{\neg x_1\}\}$ (uc=1).
- **Resolution:** Nicht erlaubt, denn durch Resolution kann sich die Anzahl erfüllter Klauseln erhöhen.
- **Lernen:** Nicht erlaubt, denn Resolution ist nicht sound.

Gültige Transformationen

- **One-Literal Rule:** nach der Entscheidung $\ell = \top$, lösche alle Klauseln, die ℓ enthalten und lösche $\neg \ell$ aus allen anderen Klauseln.
- **Pure-Literal Rule:** Falls ℓ nur in einer Polarität auftritt, dann lösche alle Klauseln, die ℓ enthalten.
- **Complementary Unit Clause Rule:** Gibt es genau zwei Unit Klauseln $\{\ell\}$ und $\{\neg \ell\}$, so ersetze diese durch eine leere Klausel.

Zähle für die obigen Transformationen, wieviele Klauseln erfüllt wurden und addiere den Wert zum Ergebnis der transformierten MaxSAT Instanz hinzu.

MaxSAT durch iteratives SAT Solving

MaxSAT lösen mit Hilfe von SAT Solving

- Idee:
 - Rufe iterativ SAT Solver auf, bis Optimum gefunden ist
 - Ermögliche dabei Ausschaltung von Klauseln durch Blocking-Variablen
- Vorteil:
 - Verwendung von effizienten SAT-Techniken (Unit Propagation mit Watched Literals, Clause Learning mit Non-Chronological Backtracking, etc.), welche nicht oder nur teilweise auf MaxSAT übertragen werden können
- Zwei Ansätze:
 - Reduktion auf (reines) SAT Solving
 - Reduktion auf SAT Solving mit *unsatisfiable core* Erweiterung

Grundidee: Schrittweise Annäherung von unten

- Führe zu jedem Constraint C_i neue Blocking-Variable b_i ein: $C'_i = C_i \vee b_i$.
- Führe zusätzlichen Cardinality Constraint ein: $\sum_{i=1}^m b_i \leq k$
- Löse iterativ $SAT((\bigcup C'_i) \cup CNF(\sum_{i=1}^m b_i \leq k))$ für $k = 0, 1, \dots$ bis genügend (MinUNSAT) viele Klauseln ausgeschaltet sind.

Iteratives MaxSAT: Sprungweise Annäherung von oben

Der Algorithmus von Daniel LeBerre

Algorithm 2: LeBerre(φ)

Input: MaxSAT Instanz $\varphi = \{C_1, \dots, C_m\}$

$BV = \{b_1, \dots, b_m\}$

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$ // Add blocking variables

$ub \leftarrow m$

while SAT($\varphi \cup \text{CNF}(\sum_{i=1}^m b_i < ub)$) **do**

 // Are blocking variables used at all ?

if #satisfiedBlockingVariables > 0 **then**

$ub \leftarrow \text{\#satisfiedBlockingVariables}$

else

return 0

return ub

Literatur: Biere et al. Handbook of Satisfiability, 19.6, S.625. IOS Press, 2009.

Binäre Suche für MaxSAT

Der Algorithmus

Algorithm 3: BinarySearch(φ)

Input: MaxSAT Instanz $\varphi = \{C_1, \dots, C_m\}$

$\varphi \leftarrow \{C_1 \vee b_1, \dots, C_m \vee b_m\}$

// Add blocking variables

$lb \leftarrow 0, ub \leftarrow m$

$mid \leftarrow \lfloor \frac{ub+lb}{2} \rfloor$

while $lb < ub$ **do**

if SAT($\varphi \cup \text{CNF}(\sum_{i=1}^m b_i \leq mid)$) **then**

$ub \leftarrow mid$

else

$lb \leftarrow mid + 1$

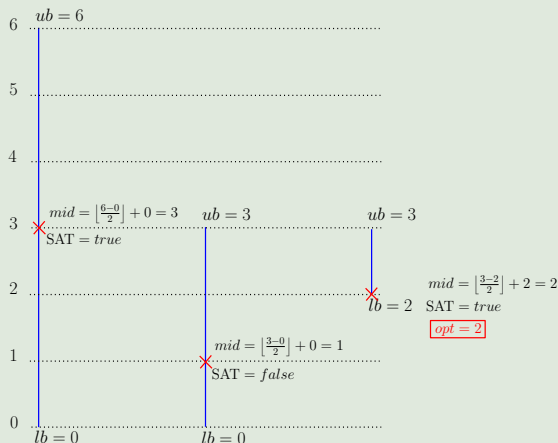
$mid \leftarrow \lfloor \frac{ub+lb}{2} \rfloor$

return ub

Binäre Suche für MaxSAT — Beispiel

Beispiel

$$\varphi = \{\{x\}, \{y\}, \{\neg x\}, \{\neg y\}, \{\neg x, \neg y\}, \{x, \neg y\}\}$$



Unsatisfiable Cores

Unsatisfiable Core (UC)

Sei φ eine Menge von Klauseln.

- Eine unerfüllbare Teilmenge von φ heißt *unsatisfiable core* (unsat core, UC)
- Ein unsatisfiable core φ_c heißt *minimal unsatisfiable core* (MUC), falls jede echte Teilmenge von φ_c erfüllbar ist. In einem MUC m ist $\text{MinUNSAT}(m)=1$.

Beispiel

$\varphi = \{\{y\}, \{x\}, \{y, \neg z\}, \{\neg y\}, \{\neg x, z\}\}$

- φ ist unsat core, aber kein minimal unsat core (z.B. $\{\{y\}, \{\neg y\}\}$ unerfüllbar)
- $\{\{y\}, \{\neg y\}\}$ ist ein minimal unsat core (global)
- $\{\{x\}, \{y, \neg z\}, \{\neg y\}, \{\neg x, z\}\}$ ist ein minimal unsat core (lokal)
- Ein unsat core u kann mehrere MUC enthalten. Diese können disjunkt sein oder gemeinsame Klauseln enthalten. Daher ist $\text{MinUNSAT}(u) \geq 1$.
- Liefert ein SAT Solver einen unsat core, so kann diese Information genutzt werden für MaxSAT (Core Guided Algorithms)

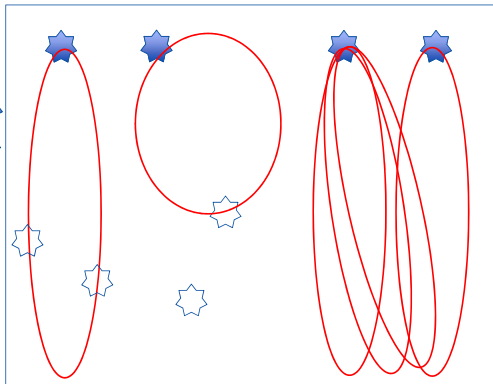
Visualizing Unsatisfiable Cores

A Landscape of Minimal Unsat Cores

SAT state: all MUCs resolved

Set of Constraints

MUC ○
blocked constraint ★
unblocked constr. ☆



Fu & Malik Algorithmus für Partial MaxSAT — Idee

Idee des Fu & Malik Algorithmus

- Prüfe iterativ mit Hilfe eines SAT Solvers (mit unsat core Unterstützung), ob Instanz erfüllbar ist. Falls ja, ist Optimum gefunden (alle Klauseln erfüllbar).
- Falls nein, so enthält der gelieferte unsat core (UC) *mindestens einen* MUC, sodass **mindestens eine Klausel unerfüllt** bleiben muss! Welche genau, ist noch nicht bekannt.
 - Erzeuge eine neue Menge B^c von blocking Variablen und füge jeder soft Klausel im UC eine davon hinzu
 - Füge cardinality constraint ($\sum_i b_i^c = 1$) über die Variablen in B^c hinzu, so dass genau eine Variable erfüllt sein muss. D.h. genau eine soft Klausel im UC wird bei der nächsten Iteration ausgeschlossen.
 - Erhöhe Kosten (unerfüllte Klauseln) um 1 und iteriere.
 - Die Instanz wird erfüllt, sobald genügend blocking Variablen vorhanden sind, dass in jedem enthaltenen MUC genau eine Klausel blockiert werden konnte.
- Spezialfall: Befindet sich im unsat core keine soft Klausel, so sind die hard Klauseln bereits unerfüllbar, d.h. es gibt kein Optimum
- Literatur: Fu, Malik. On Solving the Partial MAX-SAT Problem. SAT2006. Biere et al. Handbook of Satisfiability, 19.6, S.625–626. IOS Press, 2009.

Fu & Malik Algorithmus für Partial MaxSAT

Der Algorithmus

Algorithm 4: Fu&Malik(φ)

Input: Partial MaxSAT Instanz φ

$cost \leftarrow 0$

while *true* **do**

$(st, \varphi_c) \leftarrow \text{SAT}(\varphi)$ // st = SAT solver state; φ_c = unsat core

if $st = \text{SAT}$ **then return** $cost$

$BV \leftarrow \emptyset$ // Set of blocking variables

foreach $C \in \varphi_c$ **do**

if C is soft **then**

$b \leftarrow$ new blocking variable

$\varphi \leftarrow \varphi \setminus \{C\} \cup \{C \vee b\}$ // Add blocking variable

$BV \leftarrow BV \cup \{b\}$

if $BV = \emptyset$ **then return** None

$\varphi \leftarrow \varphi \cup \text{CNF}(\sum_{b \in BV} b = 1)$ // Cardinality constraint is hard

$cost \leftarrow cost + 1$

Fu & Malik Algorithmus für Partial MaxSAT — Beispiel 1

Beispiel (Fu & Malik 1)

- $\varphi = \text{Hard} \dot{\cup} \text{Soft}$ mit

$$\text{Hard} = \{\{x, y\}\}$$

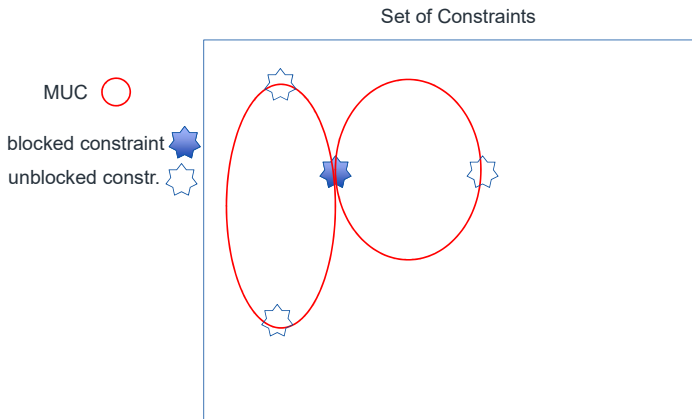
$$\text{Soft} = \{\{\neg x\}, \{x\}, \{\neg y\}\}$$

Hinweis: φ enthält unter anderem die unsat cores:

- $\{\{\neg x\}, \{x\}\}$
- $\{\{\neg x\}, \{x, y\}, \{\neg y\}\}$
- 1. Iteration: $\text{SAT}(\varphi) = \text{false}$
 $\varphi_c \leftarrow \{\{\neg x\}, \{x\}\}$
 $\varphi' \leftarrow \{\{x, y\}\} \cup \{\{\neg x, b_1^1\}, \{x, b_2^1\}, \{\neg y\}\} \cup \text{CNF} \left(\sum_{i=1}^2 b_i^1 = 1 \right)$
 $\text{cost} \leftarrow 1$
- 2. Iteration: $\text{SAT}(\varphi') = \text{true}$
 Belegung: $x \mapsto 1, y \mapsto 0, b_1^1 \mapsto 1, b_2^1 \mapsto 0$
 Ergebnis: $\text{cost} = 1$.
- Zweiter unsat core von φ wird nicht betrachtet. In φ' gibt es keinen UC mehr, da durch „blockierte“ Klausel $\{\neg x, b_1^1\}$ alle constraints erfüllbar sind.

Visualizing Unsatisfiable Cores

MUCs in Beispiel Fu & Malik 1



Fu & Malik Algorithmus für Partial MaxSAT — Beispiel 2

Beispiel (Fu & Malik 2)

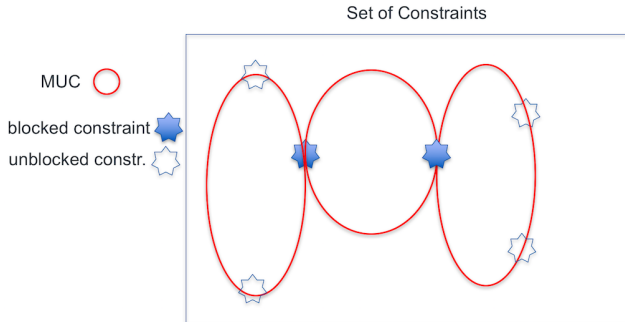
- $\varphi = \text{Hard} \dot{\cup} \text{Soft}$, mit $\text{Hard} = \{\{\neg x, y\}, \{x, z\}\}$
 $\text{Soft} = \{\{\neg x\}, \{x\}, \{\neg y\}, \{\neg z\}\}$

Hinweis: φ enthält die minimal unsat cores (MUC):

- $\{\{\neg x\}, \{x\}\}$
- $\{\{x\}, \{\neg x, y\}, \{\neg y\}\}$
- $\{\{\neg x\}, \{x, z\}, \{\neg z\}\}$
- 1. Iteration: $\text{SAT}(\varphi) = \text{false}$, $\text{cost} \leftarrow 1$
 $\varphi_c \leftarrow \{\{\neg x\}, \{x\}\}$ (ein möglicher MUC)
 $\varphi' \leftarrow \{\{b_1^1, \neg x\}, \{b_2^1, x\}, \{\neg x, y\}, \{x, z\}, \{\neg y\}, \{\neg z\}\} \cup \text{CNF} \left(\sum_{i=1}^2 b_i^1 = 1 \right)$
- 2. Iteration: $\text{SAT}(\varphi) = \text{false}$, $\text{cost} \leftarrow \text{cost} + 1$
 $\varphi'_c \leftarrow \{\{b_1^1, \neg x\}, \{b_2^1, x\}, \{\neg x, y\}, \{x, z\}, \{\neg y\}, \{\neg z\}\} \cup \text{CNF} \left(\sum_{i=1}^2 b_i^1 = 1 \right)$
 Frische blocking Variablen für alle Soft Klauseln aus diesem MUC:
 $\varphi'' \leftarrow \{\{b_1^2, b_1^1, \neg x\}, \{b_2^2, b_2^1, x\}, \{\neg x, y\}, \{x, z\}, \{b_3^2, \neg y\}, \{b_4^2, \neg z\}\} \cup$
 $\text{CNF} \left(\sum_{i=1}^2 b_i^1 = 1 \right) \cup \text{CNF} \left(\sum_{i=1}^4 b_i^2 = 1 \right). \{\bar{x}, b_2^1, z, \bar{y}, b_4^2\} \models \varphi''. \text{cost} = 2 \square$

Unsatisfiable Cores in Beispiel Fu & Malik 2

Das Beispiel Fu & Malik 2 enthält 3 MUCs. Nach der ersten Iteration wird ein UC zurückgegeben, der *mindestens* einen der MUCs enthält. Im Bsp. ist $UC = MUC$, der nun blocking Variablen erhält, von denen genau eine verwendet werden darf (1. cardinality constraint). In der zweiten Iteration gibt es nur noch *einen* MUC, gebildet aus der Vereinigung der ursprünglichen MUCs und dem ersten cardinality constraint. Alle soft Klauseln dieses MUC erhalten neue blocking Variablen, von denen genau eine verwendet werden darf (2. cardinality constraint). Danach ist diese angereicherte Formel erfüllbar.



PM2 für Partial MaxSAT — Idee

Nachteil des Algorithmus von Fu & Malik

- Jede soft Klausel innerhalb eines unsat cores erhält neue blocking Variable.
- Sind max. 2 MUC miteinander über eine gemeinsame Klausel c verbunden, so bekommt c nur *eine* blocking Variable (Beisp. 1). Wenn aber die selbe „überflüssige“ Klausel in mehreren (nicht minimalen) unsat cores vorkommt, erhält sie auch mehrere blocking Variablen. Aber auch wenn eine Klausel in mehreren MUC vorkommt, erhält sie u.U. mehrere blocking Var. (Bsp. 2).
- Suchraum für SAT Solver wird vergrößert!

Idee des PM2 Algorithmus

- Idee: Füge jeder soft Klausel nur eine einzige blocking Variable hinzu und limitiere die Erfüllbarkeit der blocking Variablen durch cardinality constraints
- Zwar hat man nur eine einzige blocking Variable pro Klausel (Vorteil), aber die Anzahl der cardinality constraints wird (je nach Codierung) sehr groß, d.h. sehr viele neue Klauseln (Nachteil).
- Literatur: Ansótegui, Bonet, Levy. On Solving MaxSAT Through SAT.
POS-10 (Pragmatics of SAT).

PM2 für Partial MaxSAT (abstrakt)

Grundzüge des PM2 Algorithmus

- Jede der n soft Klauseln erhält genau eine blocking Variable.
- Erlaube iterativ die Verwendung von genau $c = 0, 1, 2, \dots, n - 1$ blocking Variablen, bis die Instanz zum ersten Mal SAT ist mit Kosten c .
- Nachteil: Der Solver hat in jeder Iteration n blocking Variablen für die Suche zur Verfügung (großer Suchraum), um daraus c Stück mit *true* zu belegen.

Grundzüge des PM2 Algorithmus — Verbesserungen

- Bemerkung 1: $\text{constraint} = c$ für blocking Variablen ist unnötig, $\text{constraint} \leq c$ genügt (kleinere Codierung).
- Bemerkung 2: In jeder Iteration müssen weitere zu erfüllende blocking Variablen nur innerhalb des gelieferten unsat cores gefunden werden. Das schränkt den Suchraum für Belegungen ein.
- Bemerkung 3: Falls bereits früher $k - 1$ weitere UC als Teilmengen des jetzt gelieferten UC gefunden wurden, so weiß man, dass man nun mindestens k blocking Variablen im UC verwenden muss (kleinerer Suchraum).

PM2 für Partial MaxSAT

Der Algorithmus

Algorithm 5: PM2(φ)

Input: Partial MaxSAT Instanz $\varphi = \text{Hard} \dot{\cup} \text{Soft}$ mit $\text{Soft} = \{C_1, \dots, C_m\}$

$BV \leftarrow \{b_1, \dots, b_m\}$ // Blocking variables

$\varphi_w \leftarrow \text{Hard} \cup \{C_1 \vee b_1, \dots, C_m \vee b_m\}$ // Allow blocking of soft clauses

$\text{cost} \leftarrow 0; L \leftarrow \emptyset$ // L = set of unsat cores

while true do

$(st, \varphi_c) \leftarrow \text{SAT}(\varphi_w \cup \text{CNF}(\sum_{b \in BV} b \leq \text{cost}))$

if $st = \text{SAT}$ **then return** cost

$\text{cost} \leftarrow \text{cost} + 1$

PM2 für Partial MaxSAT

Der Algorithmus

Algorithm 6: PM2(φ)

Input: Partial MaxSAT Instanz $\varphi = \text{Hard} \dot{\cup} \text{Soft}$ mit $\text{Soft} = \{C_1, \dots, C_m\}$

$BV \leftarrow \{b_1, \dots, b_m\}$ // Blocking variables

$\varphi_w \leftarrow \text{Hard} \cup \{C_1 \vee b_1, \dots, C_m \vee b_m\}$ // Allow blocking of soft clauses

$\text{cost} \leftarrow 0; L \leftarrow \emptyset$ // L = set of unsat cores

while true do

$(st, \varphi_c) \leftarrow \text{SAT}(\varphi_w \cup \text{CNF}(\sum_{b \in BV} b \leq \text{cost}))$

if $st = \text{SAT}$ **then return** cost

 Remove hard clauses from φ_c

if $\varphi_c = \emptyset$ **then return** None

$B \leftarrow \emptyset$ // Blocking variables of the unsat core

foreach $C = C_i \vee b_i \in \varphi_c$ **do**

$B \leftarrow B \cup \{b_i\}$

$\varphi_w \leftarrow \varphi_w \cup \text{CNF}(\sum_{b \in B} b \geq 1)$

$\text{cost} \leftarrow \text{cost} + 1$

PM2 für Partial MaxSAT

Der Algorithmus

Algorithm 7: PM2(φ)

Input: Partial MaxSAT Instanz $\varphi = \text{Hard} \dot{\cup} \text{Soft}$ mit $\text{Soft} = \{C_1, \dots, C_m\}$

```

BV  $\leftarrow \{b_1, \dots, b_m\}$  // Blocking variables
 $\varphi_w \leftarrow \text{Hard} \cup \{C_1 \vee b_1, \dots, C_m \vee b_m\}$  // Allow blocking of soft clauses
cost  $\leftarrow 0$ ; L  $\leftarrow \emptyset$  // L = set of unsat cores
while true do
     $(st, \varphi_c) \leftarrow \text{SAT}(\varphi_w \cup \text{CNF}(\sum_{b \in BV} b \leq \text{cost}))$ 
    if st = SAT then return cost
    Remove hard clauses from  $\varphi_c$ 
    if  $\varphi_c = \emptyset$  then return None
    B  $\leftarrow \emptyset$  // Blocking variables of the unsat core
    foreach  $C = C_i \vee b_i \in \varphi_c$  do
        B  $\leftarrow B \cup \{b_i\}$ 
    L  $\leftarrow L \cup \{\varphi_c\}$ 
    k  $\leftarrow |\{\psi \in L \mid \psi \subseteq \varphi_c\}|$  // Number of unsat cores contained in  $\varphi_c$ 
     $\varphi_w \leftarrow \varphi_w \cup \text{CNF}(\sum_{b \in B} b \geq k)$ 
    cost  $\leftarrow \text{cost} + 1$ 

```

PM2 für Partial MaxSAT — Beispiel 1

Beispiel

- $\varphi = \varphi_{\text{Soft}} = \{\{\neg x\}, \{x\}, \{\neg y\}, \{y\}, \dots\}$ mit nur diesen beiden MUC.
Füge blocking Variablen $b_1, b_2, b_3, b_4, \dots, b_n$ zu φ hinzu.
 $\varphi = \{\{\neg x, b_1\}, \{x, b_2\}, \{\neg y, b_3\}, \{y, b_4\}, \dots\}$
- 1. Iteration: $L \leftarrow \emptyset$, $\text{SAT}(\varphi \cup \text{CNF}(\sum_{i=1}^n b_i \leq 0)) = \text{false}$
 $\varphi_c \leftarrow \{\{\neg x, b_1\}, \{x, b_2\}\} \cup \text{CNF}(\sum_{i=1}^n b_i \leq 0)$
 Ohne harte Klauseln: $\varphi_c \leftarrow \{\{\neg x, b_1\}, \{x, b_2\}\}$
 $L \leftarrow L \cup \varphi_c$, $k \leftarrow 1$ (denn $\varphi_c \subseteq \varphi$)
 $\varphi' \leftarrow \varphi \cup \text{CNF}(\sum_{i=1}^2 b_i \geq 1)$, $\text{cost} \leftarrow 1$
- 2. Iteration: $\text{SAT}(\varphi' \cup \text{CNF}(\sum_{i=1}^n b_i \leq 1)) = \text{false}$
 $\varphi'_c \leftarrow \{\{\neg y, b_3\}, \{y, b_4\}\} \cup \text{CNF}(\sum_{i=1}^2 b_i \geq 1) \cup \text{CNF}(\sum_{i=1}^n b_i \leq 1)$
 Ohne harte Klauseln: $\varphi'_c \leftarrow \{\{\neg y, b_3\}, \{y, b_4\}\}$
 $L \leftarrow L \cup \varphi'_c$, $k \leftarrow 1$ (denn $\varphi'_c \subseteq \varphi'$ aber $\varphi_c \not\subseteq \varphi'_c$)
 $\varphi'' \leftarrow \varphi' \cup \text{CNF}(\sum_{i=3}^4 b_i \geq 1)$, $\text{cost} \leftarrow \text{cost} + 1$.
- 3. Iteration: $\text{SAT}(\varphi'' \cup \text{CNF}(\sum_{i=1}^n b_i \leq 2)) = \text{true}$
 mit Modell $\{\bar{x}, b_2, \bar{y}, b_4, \dots\}$. Resultat: $\text{cost} = 2 \square$

PM2 für Partial MaxSAT — Beispiel 2

Beispiel 2 zeigt den Sinn der Berechnung von L und k am Vergleich zu Beispiel 1. Gleiche Problemstellung, aber ein anderer möglicher Verlauf des Algorithmus.

Beispiel

- 1. Iteration: $cost = 0$, $L \leftarrow \emptyset$, $SAT(\varphi \cup CNF(\sum_{i=1}^n b_i \leq 0)) = false$
 $\varphi_c \leftarrow \{\{\neg x, b_1\}, \{x, b_2\}\} \cup CNF(\sum_{i=1}^n b_i \leq 0)$
 Ohne harte Klauseln: $\varphi_c \leftarrow \{\{\neg x, b_1\}, \{x, b_2\}\}$
 $L \leftarrow L \cup \varphi_c$, $k \leftarrow 1$ (denn $\varphi_c \subseteq \varphi$)
 $\varphi' \leftarrow \varphi \cup CNF(\sum_{i=1}^2 b_i \geq 1)$, $cost \leftarrow 1$
- 2. Iteration: $SAT(\varphi' \cup CNF(\sum_{i=1}^n b_i \leq 1)) = false$
 $\varphi'_c \leftarrow \{\{\neg x, b_1\}, \{x, b_2\}, \{\neg y, b_3\}, \{y, b_4\}\} \cup CNF(\sum_{i=1}^2 b_i \geq 1) \cup$
 $CNF(\sum_{i=1}^n b_i \leq 1)$
 Ohne harte Klauseln: $\varphi'_c \leftarrow \{\{\neg x, b_1\}, \{x, b_2\}, \{\neg y, b_3\}, \{y, b_4\}\}$
 $L \leftarrow L \cup \varphi'_c$, $k \leftarrow 2$ (denn $\varphi'_c \subseteq \varphi'$ und $\varphi_c \subseteq \varphi'_c$)
 $\varphi'' \leftarrow \varphi' \cup CNF(\sum_{i=1}^4 b_i \geq 2)$, $cost \leftarrow cost + 1$.
- 3. Iteration: $SAT(\varphi'' \cup CNF(\sum_{i=1}^n b_i \leq 1)) = true$
 mit Modell $\{\bar{x}, b_2, \bar{y}, b_4, \dots\}$. Resultat: $cost = 2 \square$

WPM1 für Partial Weighted MaxSAT — Idee

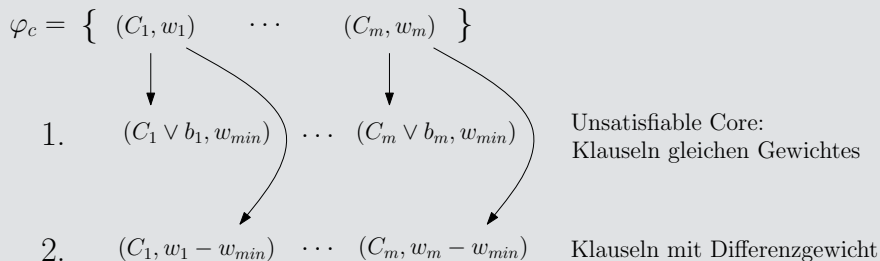
Idee des WPM1 Algorithmus

- Erweiterung des Fu & Malik Algorithmus von Partial MaxSAT zu Partial **Weighted** MaxSAT
- Prüfe iterativ mit Hilfe eines SAT Solvers (mit unsat core Unterstützung), ob Instanz erfüllbar ist
- Falls ja, so ist Optimum gefunden (alle Klauseln erfüllbar)
- Falls nein, so kann **mindestens** das Minimum w_{\min} der Gewichte der soft Klauseln aus dem gelieferten unsat core **nicht** erfüllt werden!
 - Spalte jede soft Klausel (C, w) wie folgt in 2 Klauseln auf:
 - ① Klausel $(C \vee b, w_{\min})$ mit blocking Variable b
 - ② Klausel $(C, w - w_{\min})$
 - Füge cardinality constraint über die eingeführten blocking Variablen hinzu, so dass genau eine Variable erfüllt sein muss. D.h. eine soft Klausel mit Gewicht w_{\min} im unsat core wird bei der nächsten Iteration ausgeschlossen.
 - Erhöhe Kosten um w_{\min}

WPM1 für Partial Weighted MaxSAT — Idee

Idee des WPM1 Algorithmus (continued)

- Gefundener unsat core wird aufgeteilt:
 - Klauseln mit blocking Variable und Gewicht w_{\min} (Unsat core mit Klauseln gleichen Gewichtes)
 - Klauseln ohne blocking Variable und Restgewicht (Rest des unsat cores)



- Literatur: Ansótegui, Bonet, Levy. On Solving MaxSAT Through SAT. POS-10. Pragmatics of SAT.

WPM1 für Partial Weighted MaxSAT

Der Algorithmus

Algorithm 8: WPM1(φ)

Input: P. W. MaxSAT Instanz $\varphi = \text{Hard} \dot{\cup} \text{Soft}$ mit $\text{Soft} = \{(C_1, w_1), \dots, (C_m, w_m)\}$
 $\text{cost} \leftarrow 0$

while *true* **do**

- $(st, \varphi_c) \leftarrow \text{SAT}(\varphi)$ // SAT solver call
- if** $st = \text{SAT}$ **then return** cost
- $BV \leftarrow \emptyset$
- $w_{\min} \leftarrow \min\{w_i \mid C_i \in \varphi_c \setminus \text{Hard}\}$
- foreach** $C_i \in \varphi_c$ **do**
 - if** C_i *is soft* **then**
 - $b \leftarrow$ new blocking variable
 - $\varphi \leftarrow \varphi \setminus \{(C_i, w_i)\} \cup \{(C_i \vee b, w_{\min})\}$
 - if** $w_i - w_{\min} > 0$ **then** $\varphi \leftarrow \varphi \cup \{(C_i, w_i - w_{\min})\}$
 - $BV \leftarrow BV \cup \{b\}$
- if** $BV = \emptyset$ **then return** None
- $\varphi \leftarrow \varphi \cup \text{CNF}(\sum_{b \in BV} b = 1)$ // Cardinality Constraint is hard
- $\text{cost} \leftarrow \text{cost} + w_{\min}$

WPM1 für Partial Weighted MaxSAT — Beispiel

Beispiel

- $\varphi = \text{Hard} \dot{\cup} \text{Soft}$ mit

$$\text{Hard} = \{\{x, y\}\}$$

$$\text{Soft} = \{(\{\neg x\}, 3), (\{x\}, 2), (\{\neg y\}, 5)\}$$

- 1. Iteration: $\text{SAT}(\varphi) = \text{false}$

$$\varphi_c \leftarrow \{(\neg x, 3), (x, 2)\}$$

$$w_{\min} \leftarrow 2$$

$$\varphi \leftarrow$$

$$\{\{x, y\}\} \cup \{(\{\neg x, b_1^1\}, 2), (\{x, b_2^1\}, 2), (\{\neg x\}, 1), (\{\neg y\}, 5)\} \cup \text{CNF} (\sum_{i=1}^2 b_i^1 = 1)$$

$$\text{cost} \leftarrow 0 + w_{\min} = 2$$

- Veranschaulichung der Aufteilung:

$$\varphi_c = \{ (\neg x, 3), (x, 2) \}$$

$$1. (\neg x \vee b_1^1, 2)$$

$$(x \vee b_2^1, 2)$$

Unsatisfiable Core:
Klauseln gleichen Gewichtes

$$2. (\neg x, 1)$$

$$(x, 0)$$

Klauseln mit Differenzgewicht

WPM1 für Partial Weighted MaxSAT — Beispiel

Beispiel (continued)

- Aus der vorherigen Iteration:

$$\varphi \leftarrow$$

$$\{\{x, y\}\} \cup \{(\{\neg x, b_1^1\}, 2), (\{x, b_2^1\}, 2), (\{\neg x\}, 1), (\{\neg y\}, 5)\} \cup \text{CNF} \left(\sum_{i=1}^2 b_i^1 = 1 \right)$$

- 2. Iteration: $\text{SAT}(\varphi) = \text{false}$

$$\varphi_c \leftarrow \{\{x, y\}, (\{\neg x\}, 1), (\{\neg y\}, 5)\}$$

$$w_{\min} \leftarrow 1$$

$$\varphi \leftarrow \{\{x, y\}\} \cup \{(\{\neg x, b_1^1\}, 2), (\{x, b_2^1\}, 2), (\{\neg x, b_1^2\}, 1), (\{\neg y, b_2^2\}, 1), (\{\neg y\}, 4)\}$$

$$\cup \text{CNF} \left(\sum_{i=1}^2 b_i^1 = 1 \right) \cup \text{CNF} \left(\sum_{i=1}^2 b_i^2 = 1 \right)$$

$$\text{cost} \leftarrow 2 + w_{\min} = 3$$

- 3. Iteration: $\text{SAT}(\varphi) = \text{true}$

Belegung: $x \mapsto 1, y \mapsto 0, b_1^1 \mapsto 1, b_2^1 \mapsto 1$, restliche b_i^j auf 0 belegt

Ergebnis: $\text{cost} = 3$

Literatur

- Chu Min Li, Felip Manyà. MaxSAT. In: Biere et al. *Handbook of Satisfiability*, Kap. 19.6, S.625–626. IOS Press, 2009.
- Fu, Malik. On Solving the Partial MAX-SAT Problem. In: *Proc. SAT2006*.
- Ansótegui, Bonet, Levy. On Solving MaxSAT Through SAT. In: *Proc. POS-10. Pragmatics of SAT*.