Belegarbeit

Präprozessortechniken für Pseudo-Boolean-Constraints

Timo Richter

22. März 2017

Technische Universität Dresden
Fakultät Informatik
Institut für Künstliche Intelligenz
Professur für Knowledge Representation and Reasoning

Betreut von:
Prof. Steffen Hölldobler
Dipl. Inf. Peter Steinke

Aufgabenstellung Belegarbeit

Name, Vorname:	Richter, Timo
Studiengang:	Informatik, Diplom
Matrikelnummer:	3643944
Thema:	***Präprozessortechniken für Pseudo-Boolean-Constraints***
Zielstellung:	Pseudo-Boolean (PB) Constraints bilden die Grundlage für viele Formale Problembeschreibungen. Dabei wird die gewichtete Summe von Literalen durch eine Konstante k eingeschränkt: $\sum_i^n w_i x_i \circ k$, mit $\circ \in \{=, >, <, \leq, \geq\}$ und w_i, k sind ganze Zahlen und x_i Literale. In dieser Belegarbeit soll die Möglichkeit untersucht werden, PB Constraints durch Präprozessortechniken zu vereinfachen, mit dem Ziel, dass die entstanden Constraints schneller von einem SAT-basierten Solver gelöst werden können als ohne Präprozessortechniken.

Schwerpunkte:

- Entwicklung und Implementierung verschiedener Präprozessortechniken.
- Empirische Untersuchung der implementierten Algorithmen.

Betreuer:	Dipl. Inf. Peter Steinke
Verantwortlicher Hochschullehrer:	Prof. Steffen Hölldobler

Institut:	Künstliche Intelligenz
Lehrstuhl:	Knowledge Representation and Reasoning
Beginn am:	07.11.2016
Einzureichen am:	27.03.2017

Inhaltsverzeichnis

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig, unter Angabe aller Zitate und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Dresden, den 22. März 2017

_____

Timo Richter

1 Einleitung

Ein Pseudo-Boolean-Constraint (PB) ist die Komparation einer konstanten Zahl mit der Summe von gewichteten negierten oder nicht negierten booleschen Variablen. Es wird in der Form $\sum_{j=1}^{n} w_j \cdot x_j \lhd k$ notiert mit w_j und k aus der Menge der ganzen Zahlen und x_j aus der Menge der booleschen Literale. PB-Constraints bilden die Grundlage für viele Formale Problembeschreibungen.

In dieser Belegarbeit wird die Möglichkeit untersucht, PB-Constraints durch Präprozessortechniken zu vereinfachen, mit dem Ziel, dass die entstanden Constraints schneller von einem SAT-basierten Solver gelöst werden können als ohne Präprozessortechniken. Schwerpunkte sind die Entwicklung und Implementierung verschiedener Präprozessortechniken sowie die Empirische Untersuchung der implementierten Algorithmen.

PB-Constraints enthalten ganze Zahlen und sind daher für den Menschen besser lesbar als die KNF. Zudem ist die Inferenz auf PB-Constraints meist einfacher möglich als in einer äquivalenten KNF. Einige sehr lange Konjunktionen von Klauseln lassen sich in wenigen PB-Constraints ausdrücken. Probleme aus vielen Bereichen können pseudoboolesch formuliert werden, darunter Probleme aus der Optimierungsrechnung, Graphentheorie, Kombinatorik, VLSI-Design, Industrie und Wirtschaft [RM09]. In der Ungleichung $x_1 + x_2 + x_3 \geq 1$ können beispielsweise die Literale gewichtet werden, sodass ein PB-Constraint entsteht: $2 \cdot x_1 + 1 \cdot x_2 + 1 \cdot x_3 \geq 2$. Dieses kann in die KNF umgeformt werden: $(x_1 \vee x_2) \wedge (x_1 \vee x_3)$. Wird bei einem Optimierungsproblem die Zielfunktion als PB-Constraint angegeben, dann lassen sich die Gewichte in der zu minimierenden bzw. maximierenden linken Seite leicht erkennen.

Es ist allerdings meist schneller ein PB-Constraint in KNF umzuwandeln und von einem SAT-Solver lösen zu lassen als ein nativer PB-SAT-Solver ein PB-Constraint löst. Es existieren bereits höchst effiziente SAT-Solver für KNF, gleichzeitig werden in regelmäßigen Wettbewerben stets neue entwickelt und verbessert [Sat].

Der Präprozessor kann zu geeigneten PB-Constraints sowohl äquivalente Klauseln finden als auch äquivalente kürzere PB-Constraints bilden. Bei komplexen PB-Constraints kann die äquivalente KNF sehr viele Klauseln enthalten, kurze PB-Constraints haben allerdings meist eine KNF mit wenig Klauseln und diese kann ein SAT-Solver meist schneller lösen [EB05]. Der Präprozessor bildet somit die Grundlage für eine hohe Performance des SAT-Solvers. Im folgenden Beispiel wird ein Constraint in die KNF umgeformt, indem alle Belegungen negiert werden, die das PB-Constraint nicht erfüllen.

Beispiel.

$$
\begin{aligned}
& 1 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 \geq 4 \\
\text{gdw. } & \overline{(x_1 \wedge x_2 \wedge \overline{x}_3)} \wedge \overline{(x_1 \wedge \overline{x}_2 \wedge \overline{x}_3)} \wedge \overline{(\overline{x}_1 \wedge x_2 \wedge \overline{x}_3)} \wedge \overline{(\overline{x}_1 \wedge \overline{x}_2 \wedge \overline{x}_3)} \wedge \\
& \overline{(\overline{x}_1 \wedge \overline{x}_2 \wedge x_3)} \\
\text{gdw. } & (\overline{x}_1 \vee \overline{x}_2 \vee x_3) \wedge (\overline{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3) \wedge \\
& (x_1 \vee x_2 \vee \overline{x}_3)
\end{aligned}
$$

Der Umformungsprozess lässt sich durch Präprozessortechniken so optimieren, dass die resultierenden Disjunktionen in der KNF oft kürzer ausfallen und triviale Unerfüllbarkeiten bereits früh erkannt werden.

Beispiel.

$$
\begin{aligned}
& 1 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 \geq 4 \\
\text{gdw. } & x_3 \wedge 1 \cdot x_1 + 2 \cdot x_2 \geq 1 \\
\text{gdw. } & x_3 \wedge (x_1 \vee x_2)
\end{aligned}
$$

2 Vorbemerkung und verwandte Arbeiten

Es werden Begriffe definiert, die in dieser Arbeit benutzt werden. Anschließend wird grundlegende Literatur zu PB-Constraints und Präprozessortechniken vorgestellt.

2.1 Begriffe

Variablen Es sei V eine feste unendliche Menge boolescher Variablen.

Literal Es sei L die Menge aller Literale. Ein Literal ist eine positive Variable v oder eine negative Variable $\neg v$.

Komplement Das Komplement eines Literals x wird notiert als $\overline{x}$, wobei

$$\overline{x} := \begin{cases} \neg v & \text{Falls x ein positives Literal v ist} \\ v & \text{Falls x ein negatives Literal } \neg \text{v ist} \end{cases}$$

PB-Constraint Ein PB-Constraint ist ein Ausdruck der Form

$\sum_{j=1}^{n} w_j \cdot x_j \lhd k.$

Dabei sind $x_j \in L$ Literale und $w_j \in \mathbb{Z}$ die Gewichte der Literale x_j mit $j \in [1..n]$. Auf der linken Seite des Ausdrucks erscheint die Summe $\sum_{j=1}^{n} w_j \cdot x_j$. Auf der rechten Seite wird die Konstante $k \in \mathbb{Z}$ notiert. Beide Seiten verbindet der Komparator $\lhd \in \{=, \leq, <, >, \geq\}$. Ein Summand eines PB-Constraints ist ein Teilausdruck der linken Seite von der Form $w \cdot x$ mit $w \in \mathbb{Z}$ und $x \in L$.

Klausel Eine Klausel ist ein PB-Constraint $1 \cdot x_1 + ... + 1 \cdot x_n \geq 1$ und wird notiert als Disjunktion von Literalen in der Form $(x_1 \vee ... \vee x_n)$. Die „Einerklausel“ (x_1) kann auch ohne Klammern notiert werden als x_1. Die leere Klausel wird als $\bot$ notiert.

Formel Eine Formel F ist eine Konjunktion von PB-Constraints $c_1 \wedge \ldots \wedge c_m$, wobei c_j PB-Constraints sind mit $j \in [1..m]$. Sie ist in der Konjunktiven Normalform, (KNF) wenn alle PB-Constraints in F Klauseln sind. Die leere Formel wird als $\top$ notiert. $k_1 \leq \sum_{i=j}^{n} w_j \cdot x_j \leq k_2$ ist eine Kurzschreibweise für $(\sum_{j=1}^{n} w_j \cdot x_j \geq k_1) \wedge (\sum_{j=1}^{n} w_j \cdot x_j \leq k_2)$.

Interpretation Eine Interpretation I ist eine Funktion $I : L \to \{0, 1\}$, die jedem Literal den Wert 0 oder 1 zuweist, sodass für jede Variable v genau dann $I(v) = 1$ gilt, wenn $I(\neg v) = 0$. Interpretationen stellen eine Menge von Literalen I dar, die für alle Variablen $v \in V$ genau ein Element enthält aus $\{v, \neg v\}$, mit $I(v) = 1$ gdw. $v \in I$ und $I(v) = 0$ gdw. $\neg v \in I$.

Die Erfüllungsrelation $\models$ ist wie folgt definiert:

$I \models \sum_{i=j}^{n} w_j \cdot x_j \lhd k$ gdw. $\sum_{i=j}^{n} w_j \cdot I(x_j) \lhd k$ erfüllt ist.

$I \models c_1 \wedge ... \wedge c_m$ gdw. $I \models c_j$ für $j \in [1..m]$ gilt. Wenn $I \models F$, dann ist I ein Modell für F. Wenn kein Modell für F existiert, ist F unerfüllbar, ansonsten ist F erfüllbar.

Äquivalenzrelation $c_1 \equiv c_2$ (c_1 ist äquivalent zu c_2) bedeutet, dass für jede Interpretation I gilt, dass $I \models c_1$ genau dann, wenn $I \models c_2$.

Sind F und G Formeln, dann bedeutet $F \equiv G$ (F ist äquivalent zu G), dass $I \models F$ gdw. $I \models G$ für alle Interpretationen I.

Literale eines PB-Constraints Die Funktion $Lits(c)$ bildet ein PB-Constraint c ab auf die Menge seiner Literale.

Variablen eines PB-Constraints Die Funktion $Vars(c)$ bildet ein PB-Constraint c ab auf die Menge seiner Variablen.

2.2 Verwandte Arbeiten

Die PBLib von Peter Steinke et al. [PS15] kodiert PB-Constraints in die konjunktive Normalform. Sie wandelt die Komparatoren aller Constraints in Kleinergleich oder Gleich um und optimiert die PB-Constraints nur teilweise. Ziel dieser Arbeit ist es diese Optimierungen zu erweitern. Sie werden in die konjunktive Normalform umgewandelt und können anschließend mittels SAT-Sovler gelöst werden.

In dem wissenschaftlichen Artikel von Eén et al. [ES06] werden Normalisierungstechniken für PB-Constraints aufgelistet. Normalisierte PB-Constraints

erfüllen eine Reihe von gemeinsamen Bedingungen und können daher mit einheitlichen Verfahren verarbeitet werden. Ein normalisiertes PB-Constraint trägt hier stets den Komparator Größer-gleich. All seine Gewichte sind positiv und jede Variable tritt nur einmal auf. Die Gewichte sind aufsteigend geordnet und es werden bereits Tautologien und Unerfüllbare Constraints erkannt. Gewichte, die größer als die rechte Seite sind, werden entsprechend reduziert. Die rechte Seite und die Gewichte werden weiterhin mit Hilfe des größten gemeinsamen Teilers verringert. Constraints mit genau einem Literal werden zum Propagieren genutzt. Beim Propagieren mit einer Einheitsklausel x und einem PB-Constraint c wird jedes Vorkommen von x in c durch 1 ersetzt und $\overline{x}$ durch 0. Danach wird entsprechend mathematischer Grundregeln c wieder in die Form eines PB constraints gebracht. Abschließend kann das PB-Constraint verkürzt werden, indem es in einen Klausel- und einen kürzeren PB-Teil gespaltet wird. Der Klauselteil wird ohne weitere Umformung direkt an den SAT-Solver geleitet.

Roussel et al. erläutern das Schnittebenen-Beweissystem in [RM09]. Mit Hilfe der Inferenz durch Addition, Subtraktion sowie Division wird eine Methode hergeleitet, um mit Größer-gleich-PB-Constraints Resolventen zu erschließen. Durch die erzeugten Constraints aus der Resolution kann der Lösungsraum weiter eingeschränkt werden, so kann ein SAT-Solver die Lösung in manchen Fällen schneller finden. Resolventen mit genau einem Literal können zum Propagieren verwendet werden, um andere Constraints der Formel direkt zu verkürzen.

In dieser Arbeit werden die bestehenden Normalisierungs- und Vereinfachungstechniken auf die Komparatoren $\leq$, $\geq$ und $=$ ausgeweitet. Es wurde außerdem eine ganze Reihe von zusätzlichen Bedingungen ausgearbeitet, um ein unerfüllbares oder allgemeingültiges PB-Constraint zu erkennen.

3 Präprozessortechniken

Durch die im Folgenden erläuterten Verfahren wird eine Formel so vorverarbeitet, dass ein SAT-Sovler auf höchst effiziente Weise ein Modell suchen kann.

3.1 Normalisierung

Bevor die Vereinfachungen auf ein Constraint angewendet werden können, muss es normalisiert werden. Die Normalisierungsschritte werden im folgenden erläutert.

3.1.1 Ändern des Komparators

In allen weiteren Kapiteln wird vorausgesetzt, dass der Komparator $\lhd$ Element von $\{\leq, \geq, =\}$ ist. Um ein Kleiner-als-Constraint in ein Kleiner-gleich-Constraint umzuwandeln, subtrahiere 1 von der rechten Seite und ändere den Komparator in $\leq$. Um ein Größer-als-Constraint in ein Größer-gleich-Constraint umzuwandeln, addiere 1 zu der rechten Seite und ändere den Komparator in $\geq$ [RM09].

Proposition 3.1. *Jedes Größer-als-Constraint kann in ein Größer-gleich-Constraint umgeformt werden, denn*
$w_1 \cdot x_1 + ... + w_n \cdot x_n > k$ *gdw.* $w_1 \cdot x_1 + ... + w_n \cdot x_n \geq k + 1$.

Beweis.

$$\begin{aligned} I \models & w_1 \cdot x_1 + ... + w_n \cdot x_n > k \text{ gdw.} \\ & w_1 \cdot I(x_1) + ... + w_n \cdot I(x_n) > k \text{ gdw.} \\ & w_1 \cdot I(x_1) + ... + w_n \cdot I(x_n) \geq k + 1 \text{ gdw.} \\ I \models & w_1 \cdot x_1 + ... + w_n \cdot x_n \geq k + 1 \end{aligned}$$

□

Proposition 3.2. *Jedes Kleiner-als-Constraint kann in ein Kleiner-gleich-Constraint umgeformt werden, denn*
$w_1 \cdot x_1 + ... + w_n \cdot x_n < k \equiv w_1 \cdot x_1 + ... + w_n \cdot x_n \leq k - 1$.

Beweis. analog zu Proposition 3.1. □

Proposition 3.3. *Ein Constraint in der Form* $\sum_{j=1}^{n} w_j \cdot x_j = k$ *kann ersetzt werden durch die Formel* $\sum w_j \cdot x_j \leq k \wedge \sum w_j \cdot x_j \geq k$.

Beweis. Folgt aus der Definition des Gleichheitszeichens. □

Wechsel des Komparators zwischen Größer-gleich und Kleiner-gleich
Sollte es gewünscht sein, den Komparator zu „invertieren", kann dies wie folgt geschehen. Laut Eén et al. [ES06] kann von $\leq$ auf $\geq$ und umgekehrt gewechselt werden, wenn gleichzeitig alle Gewichte und k jeweils mit (-1) multipliziert werden.

Proposition 3.4. *Durch die Äquivalenz*
$w_1 \cdot x_1 + ... + w_n \cdot x_n \leq k \equiv -w_1 \cdot x_1 - ... - w_n \cdot x_n \geq -k$
kann jedes Größer-gleich-Constraint in ein Kleiner-gleich-Constraint umgeformt werden und umgekehrt.

Beweis.

$$\begin{aligned} I \models & w_1 \cdot x_1 + ... + w_n \cdot x_n \leq k \text{ gdw.} \\ & w_1 \cdot I(x_1) + ... + w_n \cdot I(x_n) \leq k \text{ gdw.} \\ & w_1 \cdot I(x_1) + ... + w_n \cdot I(x_n) - k \leq 0 \text{ gdw.} \\ & -k \leq -w_1 \cdot I(x_1) - ... - w_n \cdot I(x_n) \text{ gdw.} \\ & -w_1 \cdot I(x_1) - ... - w_n \cdot I(x_n) \geq -k \text{ gdw.} \\ I \models & -w_1 \cdot x_1 - ... - w_n \cdot x_n \geq -k \end{aligned}$$

□

Siehe auch die effiziente Implementierungstechnik in 3.6.2.

Vorteile des Gleichheitskomparators In vorhergehender Literatur wird die Umwandelung aller Constraintkomparatoren in $\geq$ vorgenommen, um weitere Schritte auf einheitliche Vorgehensweisen zuschneiden zu können. So werden einige Schritte einfacher und es muss nicht zwischen drei Fällen für drei Komparatoren unterschieden werden.

In dieser Arbeit werden allerdings die drei Komparatoren beibehalten anstatt sie alle in $\geq$ umzuwandeln, denn dies bürgt einige Vorteile für den Präprozessor und dem Umwandeln in die KNF: Sollten keine Constraints mit Gleichheitskomparator zugelassen werden, würden alle Gleichheits-Constraints gemäß Proposition 3.3 durch zwei Constraints ersetzt werden. Bestimmte darauf folgende Verarbeitungsschritte müssten also doppelt berechnet werden, sodass dem Präprozessor redundante Arbeitsschritte entstehen. Schließlich würde an die PBLib eine größere Anzahl von Constraints weitergegeben werden, was bei der Umwandelung in eine KNF mehr Klauseln produzieren kann. Wird stattdessen der Gleichheitskomparator beibehalten, dann muss nur jeweils ein Constraint pro Gleichheitskomparator im folgenden Schritt verarbeitet werden. Das Gleichheits-Constraint kann zudem auf Erfüllbarkeit getestet werden (3.2.7). Durch den Gleichheitskomparator sind die Lösungen stark beschränkt und im Fall der Unerfüllbarkeit kann die Verarbeitung direkt abgebrochen werden. Ein Constraint in der Form $k_1 \leq \sum_{j=1}^{n} w_j \cdot x_j \leq k_2$ mit $k_1, k_2 \in \mathbb{Z}$ kann im Laufe der Verarbeitung in ein einfaches Größer-als- oder Kleiner-als-Constraint umgewandelt werden, sobald festgestellt wird, dass der andere Komparator zu einer Tautologie führt. So verbleibt in diesem Fall nur ein Größer-gleich- bzw. Kleiner-gleich Constraint, welches von der PBLib besser kodiert werden kann als ein Constraint mit zwei Komparatoren.

3.1.2 Wiederholt vorkommende Variablen zusammenfügen

Gegeben sei ein Constraint c, in dem mindestens eine Variable in mehr als einem Summanden auftritt. Nun wird das Constraint verkürzt, indem die Gewichte der identischen Variablen zusammengefasst werden, sodass jede Variable höchstens einmal c vorkommt.

Proposition 3.5. *Wiederholt vorkommende Variablen in einem PB-Constraint*

$$w_1 \cdot x_1 + ... + w_i \cdot x_i + w' \cdot x + w_{i+1} \cdot x_{i+1} + ... \\ + w_j \cdot x_j + w'' \cdot x + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k$$

können so zusammengefasst werden, dass folgendes äquivalentes PB-Constraint entsteht:

$$w_1 \cdot x_1 + ... + w_i \cdot x_i + (w' + w'') \cdot x + w_{i+1} \cdot x_{i+1} + ... \\ + w_j \cdot x_j + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k.$$

Beweis. Folgt aus dem Distributivgesetz. □

Sollte im Constraint c eine Variable sowohl positiv als auch negativ auftreten, können die Gewichte dennoch zusammengefasst werden; dazu müssen sich die Literale beider Gewichte jedoch gleichen. Zunächst wird das Constraint so umgeformt, dass alle Variablen positiv auftreten. Dies kann erreicht werden, indem das Gewicht der negierten Variable mit (-1) multipliziert und zudem von der rechten Seite subtrahiert wird.

Proposition 3.6. *Sei x ein Literal und I eine Interpretation, dann gilt $I(x) = 1 - I(\overline{x})$.*

Beweis. Folgt direkt aus der Definition einer Interpretation. □

Proposition 3.7. *Ein PB-Constraint*
$w_1 \cdot x_1 + ... + w_{i-1} \cdot x_{i-1} + w_i \cdot x_i + w_{i+1} \cdot x_{i+1} + ... + w_n \cdot x_n \lhd k$
ist äquivalent zu dem PB-Constraint
$w_1 \cdot x_1 + ... + w_{i-1} \cdot x_{i-1} - w_i \cdot \overline{x}_i + w_{i+1} \cdot x_{i+1} + ... + w_n \cdot x_n \lhd k - w_i.$

Beweis.

$$
\begin{aligned}
I \models & w_1 \cdot x_1 + ... + w_{i-1} \cdot x_{i-1} + w_i \cdot x_i + w_{i+1} \cdot x_{i+1} + ... + w_n \cdot x_n \lhd k \text{ gdw.} \\
& w_1 \cdot I(x_1) + ... + w_{i-1} \cdot I(x_{i-1}) + w_i \cdot I(x_i) + w_{i+1} \cdot I(x_{i+1}) \\
& \qquad + ... + w_n \cdot I(x_n) \lhd k \text{ gdw.} \\
& w_1 \cdot I(x_1) + ... + w_{i-1} \cdot I(x_{i-1}) + w_i \cdot (1 - I(\overline{x_i})) + w_{i+1} \cdot I(x_{i+1}) \\
& \qquad + ... + w_n \cdot I(x_n) \lhd k \text{ gdw.} \\
& w_1 \cdot I(x_1) + ... + w_{i-1} \cdot I(x_{i-1}) + w_i - w_i \cdot I(\overline{x_i}) + w_{i+1} \cdot I(x_{i+1}) \\
& \qquad + ... + w_n \cdot I(x_n) \lhd k \text{ gdw.} \\
& w_1 \cdot I(x_1) + ... + w_{i-1} \cdot I(x_{i-1}) - w_i \cdot I(\overline{x_i}) + w_{i+1} \cdot I(x_{i+1}) \\
& \qquad + ... + w_n \cdot I(x_n) \lhd k - w_i \text{ gdw.} \\
I \models & w_1 \cdot x_1 + ... + w_{i-1} \cdot x_{i-1} - w_i \cdot \overline{x}_i + w_{i+1} \cdot x_{i+1} + ... + w_n \cdot x_n \lhd k - w_i
\end{aligned}
$$

□

Für eine effiziente Implementierung siehe auch 3.6.1.

Durch diese Umformung können negative Gewichte entstanden sein. Um ein normalisiertes Constraint zu erhalten, müssen diese eliminiert werden.

3.1.3 Negative Gewichte eliminieren

Existiert ein w_j mit $j \in [1..n]$ und $w_j < 0$, dann muss das Constraint für die weitere Verarbeitung gemäß Proposition 3.7 umgeformt werden, sodass für alle w_i mit $i \in [1..n]$ gilt $w_i > 0$.

Beispiel.

$$\begin{aligned} & 1 \cdot x_1 - 2 \cdot x_2 \geq 1 \\ \text{gdw. } & 1 \cdot x_1 - 2 \cdot (1 - \overline{x}_2) \geq 1 \\ \text{gdw. } & 1 \cdot x_1 - 2 + 2 \cdot \overline{x}_2 \geq 1 \\ \text{gdw. } & 1 \cdot x_1 + 2 \cdot \overline{x}_2 \geq 1 + 2 \\ \text{gdw. } & 1 \cdot x_1 + 2 \cdot \overline{x}_2 \geq 3 \end{aligned}$$

Im gleichen Schritt sind Summanden zu entfernen, deren Gewicht exakt null ist, da sie irrelevant sind. Im Folgenden wird angenommen, dass für alle Gewichte w_i mit $i \in [1..n]$ gilt: $w_i > 0$.

Proposition 3.8. *Das Constraint*
$w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + 0 \cdot x_j + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k$
ist äquivalent zu
$w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k.$

Beweis. Es sei I eine Interpretation.

$$\begin{aligned} I \models & w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + 0 \cdot x_j + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k \text{ gdw.} \\ & w_1 \cdot I(x_1) + ... + w_{j-1} \cdot I(x_{j-1}) + 0 \cdot I(x_j) + w_{j+1} \cdot I(x_{j+1}) \\ & \qquad + ... + w_n \cdot I(x_n) \lhd k \text{ gdw.} \\ & w_1 \cdot I(x_1) + ... + w_{j-1} \cdot I(x_{j-1}) + w_{j+1} \cdot I(x_{j+1}) + ... + w_n \cdot I(x_n) \lhd k \text{ gdw.} \\ I \models & w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k \end{aligned}$$

□

3.2 Vereinfachungen

In einem normalisierten PB-Constraint $\sum_{j=1}^{n} w_j \cdot x_j \lhd k$ erscheint jede Variable $v \in V$ höchstens einmal. Sein Komparator $\lhd$ ist Kleiner-gleich, Größer-gleich oder gleich. Alle Gewichte $w_1, ..., w_n$ sind größer als null.

Für die Vereinfachungen wird angenommen, dass die Constraints zuvor normalisiert wurden.

3.2.1 Aufspüren trivialer Constraints und Einerklauseln

Unter bestimmten Bedingungen lässt sich ein PB-Constraint auf eine äquivalente simplere Aussage reduzieren (siehe Abbildung 1). Die Wahl der äquivalenten Aussage hängt von k und $\lhd$ ab. Der Wert sum ist die Summe der Gewichte: $sum = \sum_{j=1}^{n} w_j$. Das kleinste Gewicht w_{min} ist definiert als $w_{min} = min\{w_1, ..., w_n\}$.

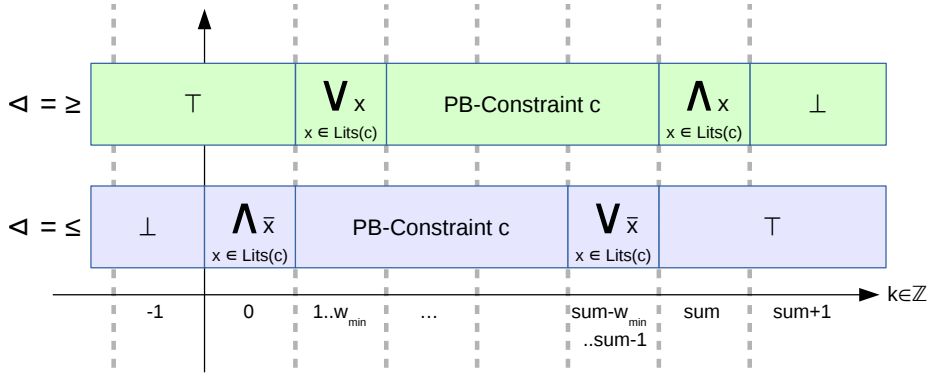


Abbildung 1: Äquivalente Aussagen zu einem normalisierten PB-Constraint c in Abhängigkeit von k und $\lhd$ für $n \geq 2$ und $w_{min} \geq 1$

Ein normalisiertes Constraint in der Form $\sum_{j=1}^{n} w_j \cdot x_j \geq -5$ entspricht beispielsweise einer Tautologie, da $w_j > 0$ für alle $j \in [1..n]$. Ein normalisiertes Constraint in der Form $\sum w_j \cdot x_j \leq 0$ entspricht der Aussage $\bigwedge_{x \in Lits(c)} \overline{x}$. PB-Constraints, die in keine äquivalente, nicht pseudo-boolesche Aussage umgewandelt werden können, heißen „echte“ PB-Constraints. Lediglich diese müssen als PB-Constraint weiterverarbeitet werden.

Proposition 3.9. *Die linke Seite eines normalisierten Constraints ist unter allen Interpretationen stets größer oder gleich null.*

Beweis. Laut Defintion gilt $\forall x \in L : I(x) \geq 0$.

Für ein normalisiertes PB-Constraint gilt $\forall j : w_j > 0$ laut 3.1.3.

Für die Multiplikation gilt $\forall a, b \in \mathbb{N} : a \cdot b \geq 0$ und für die Addition $\forall a, b \in \mathbb{N} : a + b \geq 0$.

Also ist $\sum w_j \cdot I(x_j)$ stets größer oder gleich null. □

Die folgenden Propositionen bilden die Grundlage für die referenzierte Abbildung.

Proposition 3.10. *Das Constraint $\sum w_j \cdot x_j \geq k$ entspricht einer Tautologie für $k \leq 0$.*

Beweis. Folgt direkt aus Proposition 3.9. □

Proposition 3.11. *Das Constraint $\sum w_j \cdot x_j \leq k$ ist unerfüllbar für $k \leq -1$.*

Beweis. Folgt direkt aus Proposition 3.9. □

Proposition 3.12. *Das Constraint*
$w_1 \cdot x_1 + ... + w_n \cdot x_n \leq 0$
ist äquivalent zu
$\overline{x}_1 \wedge ... \wedge \overline{x}_n$ *mit* $w_i \geq 1$ *für alle* $i \in [1..n]$.

Beweis. Es sei I eine Interpretation.

$$
\begin{aligned}
I \models & w_1 \cdot x_1 + ... + w_n \cdot x_n \leq 0 \text{ gdw.} \\
& w_1 \cdot I(x_1) + ... + w_n \cdot I(x_n) \leq 0 \text{ gdw.} \\
& I(x_1) = 0 \wedge ... \wedge I(x_n) = 0 \text{ gdw.} \\
I \models & \overline{x}_1 \wedge ... \wedge \overline{x}_n \text{ per Definition (2.1)}
\end{aligned}
$$

□

Proposition 3.13. *Das Constraint* $w_1 \cdot x_1 + ... + w_n \cdot x_n \geq k$ *ist äquivalent zu* $x_1 \vee ... \vee x_n$ *für alle* $k \in [1..w_{min}]$ *und* $w_{min} \geq 1$.

Beweis.

$$
\begin{aligned}
I \models & w_1 \cdot x_1 + ... + w_n \cdot x_n \geq k \text{ gdw.} \\
I \not\models & w_1 \cdot x_1 + ... + w_n \cdot x_n < k \text{ gdw.} \\
I \not\models & \overline{x}_1 \wedge ... \wedge \overline{x}_n \text{ gdw.} && \text{da } k \in [1..w_{min}] \\
I \models & \overline{\overline{x}_1 \wedge ... \wedge \overline{x}_n} \text{ gdw.} \\
I \models & x_1 \vee ... \vee x_n
\end{aligned}
$$

□

Proposition 3.14. *Das Constraint*
$w_1 \cdot x_1 + ... + w_n \cdot x_n \geq sum$
ist äquivalent zu
$x_1 \wedge ... \wedge x_n$ *mit* $w_i \geq 1$ *für alle* $i \in [1..n]$.

Beweis. Wenn der Komparator umgewandelt wird gemäß Proposition 3.4 und dann das Constraint wieder in die normalisierte Form gebracht wird gemäß Proposition 3.7, dann geht der Beweis hervor aus Proposition 3.12.

$$
\begin{aligned}
& w_1 \cdot x_1 + ... + w_n \cdot x_n \geq sum \equiv \\
& w_1 \cdot \overline{x}_1 + ... + w_n \cdot \overline{x}_n \leq sum - sum \text{ gdw.} \\
& x_1 \wedge ... \wedge x_n \text{ (gemäß Proposition 3.12)}
\end{aligned}
$$

□

Proposition 3.15. *Das Constraint*
$w_1 \cdot x_1 + ... + w_n \cdot x_n \leq k$
ist äquivalent zu
$\overline{x}_1 \vee ... \vee \overline{x}_n$ *für alle* $k \in [sum - w_{min}..sum - 1]$ *mit* $w_i \geq 1$ *für* $i \in [1..n]$.

Beweis. Wenn der Komparator umgewandelt wird gemäß Proposition 3.4 und dann das Constraint wieder in die normalisierte Form gebracht wird gemäß Proposition 3.7, dann geht der Beweis hervor aus Proposition 3.13. □

Proposition 3.16. *Das Constraint $w_1 \cdot x_1 + ... + w_n \cdot x_n \geq k$ ist unerfüllbar für alle $k > sum$.*

Beweis. Folgt trivialerweise aus der Bedingung, dass sum die Summe aller Gewichte ist. □

Proposition 3.17. *Das Constraint* $w_1 \cdot x_1 + ... + w_n \cdot x_n \leq k$ entspricht einer Tautologie für alle $k \geq sum$.

Beweis. Folgt trivialerweise aus der Bedingung, dass sum die Summe aller Gewichte ist. □

Einerconstraints von der Form $w \cdot x \lhd k$ können stets in Klauseln umgewandelt werden, siehe dazu 3.2.6.

Gleichheitsconstraints werden sowohl in die Kleiner-gleich-Zeile als auch in die Größer-gleich-Zeile eingeordnet. Das Constraint ist äquivalent zu der Konjunktion der beiden Aussagen.

3.2.2 Gewichte sättigen

Zu einem Constraint $\sum_{j=1}^{n} w_j \cdot x_j \geq k$ existiert ein äquivalentes Constraint $\sum_{j=1}^{n} w'_j \cdot x_j \geq k$, sodass für alle w'_j gilt: $w'_j \leq k$. Constraints von der Form $\sum_{j=1}^{n} w_j \cdot x_j \geq k$ werden „gesättigt“ genannt, wenn für alle w_j gilt: $w_j \leq k$.

Größer-gleich-Constraints Angenommen das Constraint $\sum_{j=1}^{n} w_j \cdot x_j \geq k$ enthält den Summanden $w_i \cdot x_i$ mit $i \in [1..n]$ und $w_i > k$. Da im Fall $I(x_i) = 1$ das Constraint mit dem Summand $w_i \cdot x_i$ unter I als erfüllt gilt, gilt es ebenso als erfüllt, wenn der Summand $k \cdot x_i$ beträgt. Laut der „Sättigungsregel“ auf Seite 706 in [**?**] kann deshalb in diesem Fall w_i stets durch k ersetzt werden.

Beispiel.

$$\begin{aligned} & 1 \cdot x_1 + 5 \cdot x_2 \geq 3 \\ \equiv & 1 \cdot x_1 + 3 \cdot x_2 \geq 3 \end{aligned}$$

Kleiner-gleich-Constraints können in Größer-gleich-Constraints umgeformt werden, damit sich die Sättigungsregel anwenden lässt. Wie die Komparatorumformung, die Sättigung und die Rückumformung in einem Schritt ausgeführt werden können wird in 3.6.4 erläutert.

3.2.3 Gewichte verringern mit Hilfe des ggT

Die Gewichte in einem PB-Constraint lassen sich verringern, indem sie durch ihren größten gemeinsamen Teiler (ggT) geteilt werden. Die rechte Seite k wird durch die selbe Zahl geteilt. Dieser Schritt ist insbesondere wichtig, um die Äquivalenz zwischen einem PB-Constraint und einer Klausel zu finden, zum Beispiel $2 \cdot x_1 + 2 \cdot x_2 \geq 2 \equiv 1 \cdot x_1 + 1 \cdot x_2 \geq 1 \equiv x_1 \vee x_2$.

Proposition 3.18. *Das Constraint* $\sum_{j=1}^{n} w_j \cdot x_j \lhd k$ *ist äquivalent zu*

$\sum_{j=1}^{n} w_j / ggT(\{w_1, ..., w_n\}) \cdot x_j \leq \lfloor k / ggT(\{w_1, ..., w_n\}) \rfloor$ *für* $\lhd = \leq$ *und zu*
$\sum_{j=1}^{n} w_j / ggT(\{w_1, ..., w_n\}) \cdot x_j \geq \lceil k / ggT(\{w_1, ..., w_n\}) \rceil$ *für* $\lhd = \geq$.

Beweis. Folgt aus grundlegenden mathematischen Erkenntnissen. □

Beispiel.

$$\begin{aligned} & 12 \cdot x_1 + 18 \cdot x_2 + 24 \cdot x_3 \leq 43 \\ \text{gdw. } & 12/6 \cdot x_1 + 18/6 \cdot x_2 + 24/6 \cdot x_3 \leq \lfloor 43/6 \rfloor \\ \text{gdw. } & 2 \cdot x_1 + 3 \cdot x_2 + 4 \cdot x_3 \leq 7 \end{aligned}$$

3.2.4 Triviale Belegung von Variablen

Es können in PB-Constraints mit der Probing-Methode Literale belegt werden.

Proposition 3.19. Aus $c = w_1 \cdot x_1 + ... + w_j \cdot x_j + ... + w_n \cdot x_n \leq k$ mit $w_j > k$ folgt $c \equiv c \wedge \overline{x}_j$.

Beweis. Folgt aus der Definition des Vergleichsoperators. □

Beispiel.

$$\begin{aligned} & 1 \cdot x_1 + 9 \cdot x_2 + 2 \cdot x_3 \leq 5 \\ \equiv & 1 \cdot x_1 + 9 \cdot x_2 + 2 \cdot x_3 \leq 5 \wedge \overline{x}_2 \end{aligned}$$

Proposition 3.20. Aus $c = w_1 \cdot x_1 + ... + w_j \cdot x_j + ... + w_n \cdot x_n \geq k$ mit $(\sum_{i=1}^{n} w_i) - w_j < k$ folgt $c \equiv c \wedge x_j$.

Beweis. Es sei I eine Interpretation.
Mit $(\sum_{i=1}^{n} w_i) - w_j < k$ gilt: Aus $I \models c$ folgt $I(x_j) = 1$.
Daraus folgt: $I \models c$ gdw. $I \models c \wedge I \models x_j$ □

Beispiel.

$$\begin{aligned} & I \models 1 \cdot x_1 + 1 \cdot x_2 + 5 \cdot x_3 \geq 6 \\ \text{gdw. } & I \models 1 \cdot x_1 + 1 \cdot x_2 + 5 \cdot x_3 \geq 6 \wedge x_3 \qquad \text{da } w_1 + w_2 < k \end{aligned}$$

3.2.5 Constraint in PB- und Klauselpart unterteilen

Laut Eén et. al [ES06] lässt sich die Anzahl der Summanden eines echten PB-Constraints verkürzen, falls ein Klauselteil extrahiert werden kann. Ein Größer-gleich-Constraint enthält einen Klauselteil, wenn es ein Gewicht w_j gibt mit $j \in [1..n]$, sodass $w_j = k$. Das Aufteilen eines PB-Constraints c in einen reinen Klauselteil c_K und einen verbleibenden echten PB-Teil c_{PB} hat den Vorteil, dass das verbleibende PB-Constraint c_{PB} kürzer ausfällt als das ursprüngliche Constraint c. Dies ist allerdings nur der Fall, sofern mindestens zwei Literale in den Klauselpart verschoben werden konnten. Denn sowohl c_K als auch c_{PB} wird ein neues Literal hinzugefügt. Die Aufteilung ist vorteilhaft, denn einerseits wird c_{PB} meist in eine kürzere KNF transformiert als c. Andererseits liegt der Klauselteil c_K bereits in der konjunktiven Normalform vor. Zunächst wird eine frische Variable $z \in V$ benötigt mit $z \notin Vars(c)$. Weiterhin sind c_{PB} und c_K definiert wie folgt:

$$c_K \coloneqq \bigvee x_j \vee \overline{z} \qquad \text{mit } j \in [1..n] \text{ und } w_j = k$$

$$c_{PB} \coloneqq k \cdot z + \sum w_j \cdot x_j \ \geq \ k \qquad \text{mit } j \in [1..n] \text{ und } w_j \neq k$$

Es gilt $\sum w_j \cdot x_j \geq k$ ist erfüllbar gdw. $c_{PB} \wedge c_K$ erfüllbar ist. Sowohl alle Literale von c mit dem Gewicht k als auch $\overline{z}$ bilden die Summanden in c_K mit jeweils dem Gewicht 1. c_{PB} enthält sowohl die Summanden aller anderen Literale von c als auch $k \cdot z$.

3.2.6 Constraints mit einem oder zwei Literalen

Normalisierte PB-Constraints mit weniger als 3 Variablen können direkt im Präprozessor in eine KNF transformiert werden. Ein PB-Constraint c mit $n = 2$ ist in der Form $w_1 \cdot x_1 + w_2 \cdot x_2 \lhd k$ und wird nun erläutert. Es wird $w_1 \leq w_2$ angenommen ohne Beschränkung der Allgemeinheit.

Größer-gleich-Constraints

$$c \equiv \begin{cases} \top & \text{für } k \leq 0 \\ x_1 \vee x_2 & \text{für } k \in [1..w_1] \\ x_2 & \text{für } k \in (w_1..w_2] \\ x_1 \wedge x_2 & \text{für } k \in (w_2..w_1 + w_2] \\ \bot & \text{für } k > w_1 + w_2 \end{cases}$$

Falls $k \leq 0$, dann ist c eine Tautologie gemäß Proposition 3.10.

Falls $k \in [1..w_1]$, dann ist $c \equiv x_1 \vee x_2$ gemäß der Sättigungsregel (3.2.2) und Proposition 3.18.

Falls $k \in (w_1..w_2]$, dann gilt gemäß Proposition 3.20:
$c \equiv x_2 \wedge w_1 \cdot x_1 \geq k - w_2 \equiv x_2$.

Falls $k \in (w_2..w_1 + w_2]$, dann ist $c \equiv x_1 \wedge x_2$ gemäß Proposition 3.20.

Falls $k > w_1 + w_2$, dann ist c unerfüllbar gemäß Proposition 3.16.

Kleiner-gleich-Constraints $c \equiv \begin{cases} \bot & \text{für } k \leq -1 \\ \overline{x}_1 \wedge \overline{x}_2 & \text{für } k \in [0..w_1) \\ \overline{x}_2 & \text{für } k \in [w_1..w_2) \\ \overline{x}_1 \vee \overline{x}_2 & \text{für } k \in [w_2..w_1 + w_2) \\ \top & \text{für } k \geq w_1 + w_2 \end{cases}$

Falls $k \leq -1$, dann ist c unerfüllbar gemäß Proposition 3.11.

Falls $k \in [0..w_1)$, dann ist $c \equiv \overline{x}_1 \wedge \overline{x}_2$ gemäß Proposition 3.19.

Falls $k \in [w_1..w_2)$, dann gilt gemäß Proposition 3.19:
$c \equiv \overline{x}_2 \wedge w_1 \cdot x_1 \leq k \equiv \overline{x}_2$.

Falls $k \in [w_2..w_1 + w_2)$, dann ist $c \equiv \overline{x}_1 \vee \overline{x}_2$. Gemäß Proposition 3.26 und Proposition 3.18 kann das Constraint in die Form $1 \cdot x_1 + 1 \cdot x_2 \leq 1$ gebracht werden. Der Komparator wird gemäß Proposition 3.4 in ein Größer-gleich umgeformt und es verbleibt eine Klausel.

Falls $k \geq w_1 + w_2$, dann ist c eine Tautologie gemäß Proposition 3.17.

Einerconstraints Gegeben sei das Constraint $c = w \cdot x \lhd k$. Für Größer-gleich-Constraints gilt:

$$c \equiv \begin{cases} \top & \text{für } k \leq 0 \\ x & \text{für } k \in [1..w] \\ \bot & \text{für } k > w \end{cases}$$

Und für Kleiner-gleich-Constraints:

$$c \equiv \begin{cases} \bot & \text{für } k \leq -1 \\ \overline{x} & \text{für } k \in [0..w-1] \\ \top & \text{für } k \geq w \end{cases}$$

Einerklauseln werden an dieser Stelle für das Propagieren vorgemerkt.

Gleichheitsconstraints Das Constraint $\sum w_j \cdot x_j = k$ ist äquivalent zu $\sum w_j \cdot x_j \leq k \wedge \sum w_j \cdot x_j \geq k$. Also können hier sowohl die Vereinfachungen für Größer-gleich-Constraints als auch die Vereinfachungen für Kleiner-gleich-Constraints berücksichtigt werden. Beide Ausdrücke sind in einer Konjunktion zu vereinen.

Beispiel.

$$\begin{aligned} & 1 \cdot x_1 + 2 \cdot x_2 = 2 \\ \text{gdw. } & 1 \cdot x_1 + 2 \cdot x_2 \geq 2 \ \wedge \ 1 \cdot x_1 + 2 \cdot x_2 \leq 2 \\ \text{gdw. } & x_2 \ \wedge \ (\overline{x}_1 \vee \overline{x}_2) \\ \text{gdw. } & x_2 \wedge \overline{x}_1 \end{aligned}$$

3.2.7 Erfüllbarkeit von Gleichheits-Constraints prüfen

Unerfüllbare Gleichheitsconstraints können direkt durch $\bot$ ersetzt werden, ohne sie in weiteren Schritten umformen zu müssen. Es kommt vor, dass Gleichungen

nicht erfüllbar sind: nämlich wenn bei einem Größer-gleich-Constraint die rechte Seite k einen unerreichbar hohen Wert repräsentiert — oder wenn ein normalisiertes Kleiner-gleich-Constraint ein $k < 0$ aufweist. Bei Größer-gleich- und bei Kleiner-gleich-Constraints wird bereits die Erfüllbarkeit im Abschnitt 3.2.1 überprüft.

Etwas komplexer ist das Testen der Erfüllbarkeit einer Gleichung c mit Gleichheitskomparator. Hier kann mit Hilfe der Brute-Force-Methode versucht werden eine Interpretation zu finden, die c erfüllt. Wie die Lösungsmenge effizient zu berechnen ist wird in 3.6.5 erläutert.

3.3 Propagieren

Eine Klausel x kann benutzt werden, um das Literal x bzw. $\overline{x}$ in jedem anderen Constraint der selben Formel zu eliminieren. Eine „Einerklausel“ kann entweder in der Formel vorkommen oder aber in anderen Schritten erzeugt worden sein (3.2.1, 3.2.4, 3.2.5, 3.4).

Das Propagieren mit Klauseln wurde bereits in [E$^+$01] von Ehrig et al. erläutert. Es sei k_1 eine Einerklausel mit dem positiven Literal $x \in L$ und k_2 eine beliebige Klausel aus einer Formel F. Enthält k_2 das Literal x positiv, dann kann k_2 aus F entfernt werden. Enthält k_2 das negative Literal $\overline{x}$, dann wird jedes Vorkommen von $\overline{x}$ in k_2 aus den Summanden entfernt. Dieses Verfahren kann auf PB-Constraints ausgeweitet werden. Beim Propagieren mit k_1 und einem PB-Constraint c wird jedes Vorkommen von $w \cdot x$ mit $w \in \mathbb{Z}$ entfernt und w von der rechten Seite subtrahiert.

Proposition 3.21. *Die Formel*

$x_j \wedge w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w_j \cdot x_j + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k$

ist äquivalent zu

$x_j \wedge w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k - w_j$.

Beweis.

$$
\begin{aligned}
I \models\ & x_j \wedge w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w_j \cdot x_j + w_{j+1} \cdot x_{j+1} + ... \\
& \qquad + w_n \cdot x_n \lhd k \text{ gdw.} \\
I(x_j) = 1 \wedge\ & w_1 \cdot I(x_1) + ... + w_{j-1} \cdot I(x_{j-1}) + w_j \cdot I(x_j) + w_{j+1} \cdot I(x_{j+1}) \\
& \qquad + ... + w_n \cdot I(x_n) \lhd k \text{ gdw.} \\
I(x_j) = 1 \wedge\ & w_1 \cdot I(x_1) + ... + w_{j-1} \cdot I(x_{j-1}) + w_j \cdot 1 + w_{j+1} \cdot I(x_{j+1}) \\
& \qquad + ... + w_n \cdot I(x_n) \lhd k \text{ gdw.} \\
I(x_j) = 1 \wedge\ & w_1 \cdot I(x_1) + ... + w_{j-1} \cdot I(x_{j-1}) + w_{j+1} \cdot I(x_{j+1}) \\
& \qquad + ... + w_n \cdot I(x_n) \lhd k - w_j \text{ gdw.} \\
I \models\ & w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k - w_j
\end{aligned}
$$

□

Sofern vorhanden, wird $w \cdot \overline{x}$ mit $w \in \mathbb{Z}$ aus den Summanden von c entfernt.

Proposition 3.22. *Die Formel*
$\overline{x}_j \wedge w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w_j \cdot x_j + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k$
ist äquivalent zu
$\overline{x}_j \wedge w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k.$

Beweis.

$$\begin{aligned}
&\overline{x}_j \wedge w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w_j \cdot x_j + w_{j+1} \cdot x_{j+1} \\
&\qquad\qquad + ... + w_n \cdot x_n \lhd k \text{ gdw.} \\
&I(x_j) = 0 \wedge w_1 \cdot I(x_1) + ... + w_{j-1} \cdot I(x_{j-1}) + w_j \cdot I(x_j) + w_{j+1} \cdot I(x_{j+1}) \\
&\qquad\qquad + ... + w_n \cdot I(x_n) \lhd k \text{ gdw.} \\
&I(x_j) = 0 \wedge w_1 \cdot I(x_1) + ... + w_{j-1} \cdot I(x_{j-1}) + w_{j+1} \cdot I(x_{j+1}) \\
&\qquad\qquad + ... + w_n \cdot I(x_n) \lhd k \text{ gdw.} \\
&\overline{x}_j \wedge w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k
\end{aligned}$$

Das Propagieren muss stets wiederholt werden, nachdem eine „Einerklausel“ entsteht. Dieser Fall kann auch direkt nach dem Propagieren eintreten. □

Beispiel.

$$\begin{aligned}
&x_2 \wedge 1 \cdot x_1 + 2 \cdot x_2 \geq 3 \wedge 2 \cdot x_1 + 2 \cdot x_3 = 2 \equiv \\
&x_2 \wedge 1 \cdot x_1 \geq 1 \qquad \wedge 2 \cdot x_1 + 2 \cdot x_3 = 2 \equiv \\
&x_2 \wedge x_1 \qquad \wedge 2 \cdot x_1 + 2 \cdot x_3 = 2 \equiv \\
&x_2 \wedge x_1 \qquad \wedge 2 \cdot x_3 = 0 \equiv \\
&x_2 \wedge x_1 \qquad \wedge \overline{x}_3
\end{aligned}$$

3.4 Resolution

Mithilfe des Resolutionsverfahrens kann aus zwei PB-Constraints c_1 und c_2 der selben Formel ein weiteres Constraint, die Resolvente c_3, gebildet werden. Die Resolvente enthält alle Variablen aus c_1 und c_2, wobei mindestens eine Variable eliminiert wird. So kann ein besonders kurzes Constraint c_3 entstehen, das die Lösungsmöglichkeiten der Formel stark einschränkt oder ein Modell explizit sichtbar werden lässt.

Nun wird erläutert, wie das Resolutionsverfahren angewendet wird. Mit den Regeln des Schnittebenenverfahrens können PB-Constraints addiert, multipliziert und dividiert werden. Zur Resolution wird eine Kombination aus der Multiplikations- und Additionsregel verwendet [RM09]. Es erlaubt das Schließen von zwei Constraints c_1 und c_2 auf ein neues Constraint c_3, sodass mindestens ein Literal eliminiert wird. Aus der Resolvente c_3 und einem anderen geeigneten

Constraint kann ggf. wiederum eine Resolvente gebildet werden. Im Optimalfall wird eine Resolvente gefunden, die die Unerfüllbarkeit der Formel oder als Einerklausel die Belegung einer Variable zeigen kann.

Es seien $w_p, w_q \in \mathbb{N}$, $k_1, k_2 \in \mathbb{Z}$ und $x \in L$. Weiterhin sei $w_p \cdot x$ Summand von c_1 und $w_q \cdot \overline{x}$ Summand von c_2. Es werden alle Gewichte und die rechte Seite der Gleichung c_1 multipliziert mit $\alpha = w_q / ggT(w_p, w_q)$. Ebenso werden alle Gewichte und die rechte Seite der Gleichung c_2 multipliziert mit $\beta = w_p / ggT(w_p, w_q)$. Kommt ein Literal nur in einem der beiden Constraints vor, wird hier angenommen, dass es mit dem Gewicht null auch im anderen vorkommt. c_3 wird wie folgt erschlossen:

$$\frac{\begin{array}{c} c_1 = \sum a_j \cdot x_j \lhd k_1 \\ c_2 = \sum b_j \cdot x_j \lhd k_2 \end{array}}{c_3 = \sum(\alpha \cdot a_j + \beta \cdot b_j) \cdot x_j \lhd \alpha \cdot k_1 + \beta \cdot k_2}$$

Nun enthält c_3 den Summanden $w_p \cdot w_q / ggT(w_p, w_q) \cdot (x + \overline{x})$. Somit wird x in c_3 eliminiert:

Proposition 3.23. *Wenn $x \in Lits(c_1)$ und $\overline{x} \in Lits(c_2)$ gilt, dann lässt sich c_3 so umformen, dass $x \notin Lits(c_3)$ und $\overline{x} \notin Lits(c_3)$.*

Beweis. Es sei I eine Interpretation. Wenn $x \in Lits(c_1)$ und $\overline{x} \in Lits(c_2)$ gilt, dann ist c_3 in der Form $w_1 \cdot x_1 + ... + w_p \cdot w_q / ggT(w_p, w_q) \cdot (x + \overline{x}) + ... + w_n \cdot x_n \lhd k$.

$I \models c_3$ gdw.
$I \models w_1 \cdot x_1 + ... + w_p \cdot w_q / ggT(w_p, w_q) \cdot (x + \overline{x}) + ... + w_n \cdot x_n \lhd k$ gdw.
$w_1 \cdot I(x_1) + ... + w_p \cdot w_q / ggT(w_p, w_q) \cdot (I(x) + I(\overline{x})) + ... + w_n \cdot x_n \lhd k$ gdw.
$I \models w_1 \cdot x_1 + ... + w_p \cdot w_q / ggT(w_p, w_q) \cdot 1 + ... + w_n \cdot x_n \lhd k$ gdw.
$I \models w_1 \cdot x_1 + ... + w_n \cdot x_n \lhd k - w_p \cdot w_q / ggT(w_p, w_q) \implies$
$Lits(w_1 \cdot x_1 + ... + w_n \cdot x_n \lhd k - w_p \cdot w_q / ggT(w_p, w_q)) \cap \{x, \overline{x}\} = \emptyset$

□

Aus der Unerfüllbarkeit der Resolvente c_3 folgt zur Unerfüllbarkeit von c_1 und c_2:

Proposition 3.24. *Ist I ein Modell für $c_1 \wedge c_2$, dann ist I ein Modell für c_3.*

Beweis. Es sei $F = c_1 \wedge c_2 \wedge ...$ eine Formel. Dann folgt aus den Reglen der Resolution, dass $F \equiv F \wedge c_3$. Daraus folgt, dass $I \models c_1 \wedge c_2 \implies I \models c_3$. □

Beispiel.

$$\frac{\begin{array}{c} c_1 = 1 \cdot x_1 + 1 \cdot \overline{x}_2 + 3 \cdot x_3 \geq 5 \\ c_2 = 1 \cdot x_4 + 2 \cdot x_2 + 4 \cdot x_5 \geq 6 \end{array}}{c_3 = 2 \cdot x_1 + 2 \cdot \overline{x}_2 + 6 \cdot x_3 + 1 \cdot x_4 + 2 \cdot x_2 + 4 \cdot x_5 \geq 10 + 6}$$

c_3 gdw.
$2 \cdot x_1 + 2 \cdot (x_2 + \overline{x}_2) + 6 \cdot x_3 + 1 \cdot x_4 + 4 \cdot x_5 \geq 16$ gdw.
$2 \cdot x_1 + 2 \cdot 1 \quad + 6 \cdot x_3 + 1 \cdot x_4 + 4 \cdot x_5 \geq 16$ gdw.
$2 \cdot x_1 \quad + 6 \cdot x_3 + 1 \cdot x_4 + 4 \cdot x_5 \geq 14.$
$2 \quad + 6 \quad + 1 \quad + 4 \quad \not\geq 14 \implies$
$c_3 \equiv \bot \implies$
$c_1 \wedge c_2 \equiv \bot$

Es sei $F = c_1 \wedge c_2$ eine Formel. Die normalisierten Constraints c_1 und c_2 eignen sich zur Resolution, wenn ihre Resolvente c_3 nicht in F ist, ein Litreal in c_1 positiv und in c_2 negativ vorkommt und sich die Komparatoren von c_1 und c_2 gleichen. Constraints c_1 und c_2 mit unterschiedlichen Komparatoren können für dieses Verfahren umgeformt werden (3.1.1), sodass $\lhd_1$ und $\lhd_2$ identisch sind. Die Resolvente wird gemäß Abbildung 2 (3.5) so weiterverarbeitet, dass sie normalisiert, ggf. in eine Klausel umgewandelt, zum Propagieren genutzt und für weitere Resolutionsschritte berücksichtigt wird. Sie wird abschließend der Formel F hinzugefügt, sodass $F' = c_1 \wedge c_2 \wedge c_3$.

Es könnte einen Grenznutzen der Resolution geben: Die Resolvente c_3 kann bis zu $|Lits(c_1)| + |Lits(c_2)| - 2$ Literale enthalten und deshalb sehr lang werden. Es empfiehlt sich aus praktischen Gründen eine Maximallänge zu definieren. Beim Erschließen der Resolvente werden außerdem Gewichte multipliziert, sodass sehr große Zahlen entstehen können, die das Preprocessing wiederum verlangsamen. Bei Verarbeitung mittels Computer müssen Zahlenprodukte vermieden werden, die den Arbeitsspeicher übersteigen.

3.5 Abhängigkeiten der Verarbeitungsschritte

Die Reihenfolge der Verarbeitungsschritte ist so zu wählen, dass jeder Schritt mindestens einmal, aber möglichst selten durchgeführt wird. Einige Verarbeitungsschritte können jedoch das Ziel eines anderen Schritts rückgängig machen oder sogar verbessern. Nachdem in den PB-Constraints Literalen konkrete Werte zugewiesen wurden – wie in 3.3 oder 3.2.1 – wird die rechte Seite k geändert und die Anzahl der Literale im Constraint verändert sich. Somit lassen sich möglicherweise durch den Schritt „Aufspüren trivialer Constraints und Einerklauseln“ weitere Literale festlegen. Sollte sich der größte gemeinsame Teiler (ggT) aller Literale nach dem Entfernen eines Literals vergrößern, dann sollte der Schritt „Gewichte verringern mit Hilfe des ggT“ wiederholt werden. Ebenfalls ist es möglich, dass nach dem Entfernen bzw. Propagieren von Literalen ein Teil im PB-Constraint entsteht, der sich als Klausel formulieren lässt. Der Schritt „Constraint in PB- und Klauselpart unterteilen“ sollte nun durchgeführt werden, um einen Klausel- vom PB-Part zu trennen.

Die gewählte Reihenfolge der Schritte im Text von Eén ([ES06], Seite 3f) ist folgende:

1. Alle Komparatoren in Größer-gleich umwandeln
2. Negative Gewichte eliminieren
3. Wiederholt vorkommende Variablen zusammenfügen
4. Gewichte aufsteigend sortieren
5. Aufspüren trivialer Constraints
6. Gewichte sättigen
7. Gewichte verringern mit Hilfe des ggT
8. Propagieren
9. Constraint in PB- und Klauselpart unterteilen

Allerdings ist es möglich, dass nach Schritt 9 die Ausführung von Schritt 5 und anschließend Schritt 8 sinnvoll ist. Ebenso könnte nach Schritt 8 wiederum Schritt 7 erforderlich sein. Sollte sich nämlich nach dem Propagieren der ggT einer Gleichung ändern, ist das Ziel von Schritt 7 nicht mehr erfüllt. Sollte der PB-Teil nach Schritt 9 nur ein Literal enthalten, könnte dies in Schritt 5 in eine Einerklausel umgewandelt werden. Somit ist Propagieren sinnvoll (Schritt 8). Abschließend wird eine neu entwickelte Reihenfolge vorgestellt. Zusätzlich wurden in dieser Arbeit folgende Schritte hinzugefügt:

- Constraints mit einem oder zwei Literalen
- Triviale Belegung von Variablen
- Erfüllbarkeit von Gleichheits-Constraints prüfen

Die Abhängigkeiten der Verarbeitungsschritte werden nun aufgezählt.

Nach dem Ändern des Komparators:

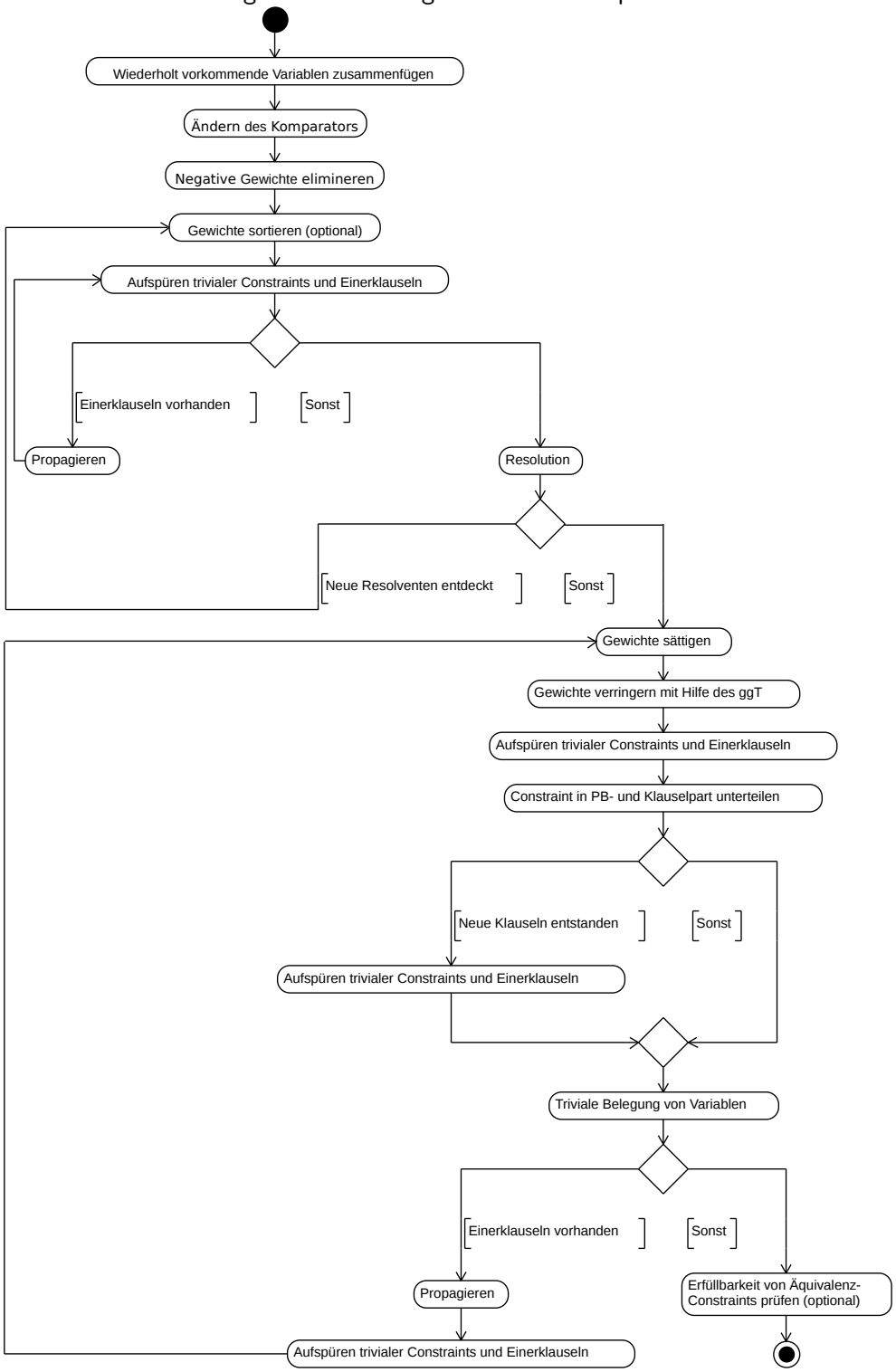
- Negative Gewichte eliminieren

Nach dem Zusammenfügen wiederholt vorkommender Variablen:

- Negative Gewichte eliminieren
- Gewichte sättigen
- Aufspüren trivialer Constraints und Einerklauseln

Nach dem Eliminieren negativer Gewichte:

Abbildung 2: Verarbeitungsschritte des Präprozessors



- Gewichte aufsteigend sortieren
- Aufspüren trivialer Constraints und Einerklauseln
- Triviale Belegung von Variablen

Nach dem Sättigen der Gewichte:

- Aufspüren trivialer Constraints und Einerklauseln
- Constraint in PB- und Klauselpart unterteilen

Nach dem Gewichte verringern mit Hilfe des ggT:

- Aufspüren trivialer Constraints und Einerklauseln

Nach der trivialen Belegung von Variablen:

- Propagieren

Nach dem Propagieren, falls Constraints verändert wurden:

- Aufspüren trivialer Constraints und Einerklauseln
- Gewichte sättigen
- Gewichte verringern mit Hilfe des ggT
- Constraint in PB- und Klauselpart unterteilen

Nach dem Klauselpart Extrahieren, falls Constraints verändert wurden:

- Aufspüren trivialer Constraints und Einerklauseln
- Triviale Belegung von Variablen

Nach dem Auffinden oder Erzeugen von Einerklauseln:

- Propagieren

Für den Schritt „Erfüllbarkeit von Gleichheits-Constraints prüfen“ sind kurze Constraints günstig. Sie sollten deshalb am Schluss ausgeführt werden. Die Resolution sollte aus Performancegründen vor dem „Constraint in PB- und Klauselpart unterteilen“ durchgeführt werden. Mittels Resolution würde andernfalls aus dem aufgespalteten Klausel- und PB-Teil eines Constraints c eine Resolvente erstellt werden, die dem ursprünglichen Constraint c gleicht. Um die Resolution mit allen Constraints durchzuführen, würden mehr Schritte benötigt, denn es würde eine neue Resolvente erstellt werden aus dem PB-Teil von c mit jeder Resolventen, die aus dem Klauselteil und einem weiteren Constraint generiert wurde.

Vor dem Sättigen der Gewichte (3.2.2) ist das „Aufspüren trivialer Constraints und Einerklauseln“ nötig. Bei einem Größer-gleich-Constraint mit negativem k würden alle Gewichte durch das negative k ersetzt werden. Dieses Constraint wäre im Schritt „Aufspüren trivialer Constraints und Einerklauseln“ zuvor allerdings als Tautologie erkannt worden.

Die demzufolge sinnvolle Reihenfolge ist in Abbildung 2 veranschaulicht. Alle Schritte beziehen sich auf PB-Constraints. Klauseln müssen nur in den Schritten „Propagieren“ und „Resolution“ berücksichtigt werden. Zudem wird die Verarbeitung für ein PB-Constraint abgebrochen, sobald es vollständig in eine Klausel umgewandelt werden konnte. Dies tritt u.a. ein, wenn nach dem Schritt „Triviale Belegung von Variablen“ alle Variablen belegt werden konnten oder bei der Unterteilung in Klausel- und PB-Teil der verbleibende PB-Teil leer ist.

Für jeden Schritt gilt: Wird die Unerfüllbarkeit festgestellt, dann endet der Präprozessor. Dies kann geschehen, wenn „triviale Constraints“ aufgespürt werden. Im Schritt „Ändern des Komparators“ wird entschieden, welchen Komparator die Ausgabeconstraints aufweisen sollen. Der Schritt „Gewichte aufsteigend sortieren“ (3.6.3) ist für diese Arbeit nicht notwendig, führt aber in der Implementierung dazu, dass das größte bzw. kleinste Gewicht schneller gefunden werden kann. Ist die Verarbeitung am Endzustand angekommen, werden die Constraints der PBLib übergeben.

3.6 Implementierungstechniken

Es wurde bereits gezeigt wie die Vereinfachungen auszuführen sind. Hier wird nun erläutert, wie sie ressourcensparend angewendet werden können mit Hilfe zusätzlicher Propositionen.

3.6.1 Wiederholt vorkommende Variablen zusammenfügen

Eine wiederholt vorkommende Variable v kann in einem PB-Constraint ebenso zusammengefügt werden, falls v sowohl positiv als auch negativ im Constraint vorkommt. Dabei kann gemäß Proposition 3.7 das Literal mit Hilfe des Komplements so umgeformt werden, dass alle Variablen positiv auftreten. Anschließend können gemäß Proposition 3.5 die Variablen zusammengefasst werden. Im Folgenden werden die genannten Propositionen miteinander vereint, sodass lediglich ein Schritt ausgeführt werden muss.

Proposition 3.25. *Das Constraint*

$$w_1 \cdot x_1 + ... + w_{i-1} \cdot x_{i-1} + w_i \cdot x + w_{i+1} \cdot x_{i+1} + ... \\ + w_{j-1} \cdot x_{j-1} + w_j \cdot \overline{x} + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k$$

ist äquivalent zu

$$w_1 \cdot x_1 + ... + w_{i-1} \cdot x_{i-1} + (w_i - w_j) \cdot x + w_{i+1} \cdot x_{i+1} + ... \\ + w_{j-1} \cdot x_{j-1} + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \lhd k - w_j.$$

Beweis. Geht hervor aus Proposition 3.7 und Proposition 3.5. □

Beispiel.

$$9 \cdot x_1 + 3 \cdot x_2 + 5 \cdot \overline{x}_2 + 6 \cdot \overline{x}_2 \leq 16 \text{ gdw.}$$
$$9 \cdot x_1 + (+3 - 5 - 6) \cdot x_2 \leq 16 - 5 - 6 \text{ gdw.}$$
$$9 \cdot x_1 - 8 \cdot x_2 \leq 5$$

3.6.2 Wechsel des Komparators zwischen Kleiner-gleich und Größergleich

Der Komparator kann gewechselt werden, sodass das umgeformte Constraint in normalisierter Form vorliegt und für die Umformung nur ein einziges Mal über alle Summanden iteriert werden muss. Wird der Komparator eines bereits normalisierten Constraints geändert, dann würde hierbei ein nichtnormalisiertes entstehen, denn durch diese Umformung entstehen negative Gewichte. Deshalb ist es vorzuziehen den Komparator so zu invertieren, dass negative Gewichte ebenfalls eliminiert werden. Dazu wird das Verfahren aus 3.1.1 kombiniert mit dem Verfahren „Negative Gewichte eliminieren“ (3.1.3). Es werden dann nicht alle Gewichte mit (-1) multipliziert, sondern nur jene, welche negativ vorkommen. Alle Literale mit positivem Gewicht müssen negiert werden. Aufgrund von Proposition 3.4 muss das k negiert werden. Wegen der Umformung aus Proposition 3.7 müssen zu der rechten Seite alle positiv vorkommenden Gewichte addiert werden. Der Vorteil dieser Variante gegenüber 3.1.1 und anschließendem 3.1.3 liegt darin, dass hier nur einmal über die Summanden iteriert werden muss.

Algorithmus:

1. Negiere alle Literale mit positivem Gewicht
2. Setze $k' \coloneqq (\sum_{j \in [1..n] \wedge w_j > 0} w_j) - k$
3. $\lhd' = \begin{cases} \leq & \text{falls } \lhd = \geq \\ \geq & \text{falls } \lhd = \leq \end{cases}$
4. Multipliziere alle negativen Gewichte mit (-1)

Das umgeformte Constraint ist in der Form $c' = \sum_{j=1}^{n} w_j \cdot x_j \lhd' k'$. Das Symbol $\Longrightarrow_s$ mit $s \in \mathbb{N}$ bedeutet keine Folgerung, sondern lediglich die Ausführung des Schrittes s.

Beispiel.

$$\begin{aligned} c_1 = &-1 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 \leq 1 \ \leadsto_1 \\ &-1 \cdot x_1 + 2 \cdot \overline{x}_2 + 3 \cdot \overline{x}_3 \leq 1 \ \leadsto_2 \\ &-1 \cdot x_1 + 2 \cdot \overline{x}_2 + 3 \cdot \overline{x}_3 \leq (2+3) - 1 \ \leadsto_3 \\ &-1 \cdot x_1 + 2 \cdot \overline{x}_2 + 3 \cdot \overline{x}_3 \geq 4 \ \leadsto_4 \\ &+1 \cdot x_1 + 2 \cdot \overline{x}_2 + 3 \cdot \overline{x}_3 \geq 4 \ \equiv \ c_1 \end{aligned}$$

3.6.3 Gewichte aufsteigend sortieren

Das Sortieren der Summanden nach Gewicht ermöglicht in den weiteren Schritten eine schnellere Verarbeitung. Ein sortiertes Constraint ist in der Form $\sum_{j=1}^{n} w_j \cdot x_j \lhd k$, wobei $w_i \leq w_{i+1}$ mit $i \in [1..n-1]$. Bei allen Vereinfachungsschritten bleibt die Sortierung erhalten. Sollen nur Gewichte verarbeitet werden, die größer sind als ein bestimmter Wert (z. B. $w_i \geq k$ mit $i \in [1..n]$), müssten bei unsortierten Constraints alle Summanden überprüft werden. Bei einem sortierten Constraint kann die Iteration direkt abgebrochen werden, sobald der Index auf ein Gewicht trifft, das den Grenzwert unterschreitet ($w_j < k$ mit $j \in [1..n]$). Die Verarbeitung von sortierten Constraints ermöglicht daher u. a. bei folgenden Aufgaben eine effizientere Verarbeitung:

- Gewichte sättigen
- Triviale Belegung von Variablen
- Constraint in PB- und Klauselpart unterteilen

Es lässt sich zudem besonders effizient das minimale bzw. das maximale Gewicht eines Constraints bestimmen, da dies das erste bzw. letzte Gewicht des Constraint darstellt. Das minimale Gewicht wird z. B. bei den Vereinfachungen aus Abschnitt 3.2.1 benötigt. Zudem hilft die Sortierung beim Implementieren der Probing-Methode. Sei $c = 1 \cdot x_1 + 2 \cdot x_2 + 5 \cdot x_3 \geq 6$ ein Größer-gleich-PB-Constraint und I eine Interpretation mit $I \models c$, dann ist zum Beispiel $I(x_3) = 1$ festzulegen aufgrund des hohen Gewichts w_3. Hier sollte mit der Untersuchung der hoch gewichteten Literale begonnen werden, also bei aufsteigend sortierten Gewichten mit dem letzten Element. Wird bei diesem Vorgehen auf ein Summand mit einem geringen Gewicht gestoßen, dessen Literal nicht von allen Modellen gleich interpretiert wird, dann werden auch „leichtere“ Literale nicht von allen Modellen gleich interpretiert. Um die in I zu belegenden Literale effizient finden zu können, muss durch eine sortierte Summandenreihenfolge iteriert werden. Selbst beim Propagieren bleibt die Sortierung erhalten. Wenn $5 \cdot x_3$ aus c entfernt wird, ist aufgrund der Sortierung das letzte Gewicht (hier: $w_2 = 2$) immer noch das größte.

3.6.4 Kleiner-gleich-Constraints sättigen

Es sei $\sum_{j=1}^{n} w_j \cdot x_j \geq k$ ein PB-Constraint. Falls ein w_j existiert mit $j \in [1..n]$, $w_j > w'_j$ und $w'_j := -k + \sum_{i=1}^{n} w_i$, dann lässt sich das Gewicht w_j reduzieren auf den Wert w'_j.

Proposition 3.26. *Das Constraint*

$w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w_j \cdot x_j + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \leq k$ *mit* $w_j > w'_j$ *ist äquivalent zu*

$w_1 \cdot x_1 + ... + w_{j-1} \cdot x_{j-1} + w'_j \cdot x_j + w_{j+1} \cdot x_{j+1} + ... + w_n \cdot x_n \leq k + w'_j - w_j$.

Beweis. Gemäß Proposition 3.4 kann der Komparator zu Größer-gleich umgewandelt werden. Nun kann w_j gemäß der „Sättigungsregel“ (3.2.2) durch w'_j ersetzt werden. Beim anschließenden Umwandeln zurück in ein Kleiner-gleich-Constraint ist k' gemäß Proposition 3.7 so festzulegen, indem alle Gewichte von k subtrahiert werden. Der Unterschied zwischen k vom Ausgangsconstraint und k' ist exakt die Differenz zwischen w'_j und w_j. Diese muss nun zu k addiert werden. □

3.6.5 Lösungsmenge eines PB-Constraints berechnen

Für den Schritt 3.2.7 muss die Lösungsmenge eines PB-Constraints berechnet werden. Bei der Brute-Force-Methode werden alle möglichen Interpretationen getestet. Dieses Vorgehen kostet jedoch viel Rechenleistung. Mit der folgenden Methode allerdings kann eine Lösung schnell gefunden werden.

Proposition 3.27. *Für jedes Größer-gleich-Constraint c gilt: Es sei $w_j \cdot x_j$ ein Summand des Constraints c mit $w_j \geq k$. Alle Interpretationen I mit $I(x_j) = 1$ sind dann ein Modell für c.*

Beweis.

$$
\begin{aligned}
& w_1 \cdot I(x_1) + ... + w_j \cdot I(x_j) + ... + w_n \cdot I(x_n) \geq k \\
\text{gdw. } & w_1 \cdot I(x_1) + ... + w_j \cdot 1 + ... + w_n \cdot I(x_n) \geq k \\
\text{gdw. } & \top \qquad\qquad \text{da } w_j \geq k
\end{aligned}
$$

□

Bei dem Kleiner-gleich- oder Gleichheits-Constraint c kann die Backtracking-Methode angewendet werden. Hierbei sollen Interpretationen a priori ausgeschlossen werden, die dazu führen, dass die linke Seite echt größer als k wird. Von der folgenden Proposition kann beim Backtracking Gebrauch gemacht werden.

Proposition 3.28. *Ist I ein Modell für c und $I(x_j) = 1$ mit $j \in [1..n]$, dann ist $I(x_i) = 0$ für alle $i \in [1..n] \backslash \{j\}$ mit $w_i > k - w_j$.*

Beweis. Folgt aus der Definition des Vergleichsoperators. □

Beispiel. Angenommen c lautet $1 \cdot x_1 + 3 \cdot x_2 + 5 \cdot x_3 = 6$. Wird dann ein Modell I gesucht und für den ersten Lösungsversuch $I(x_3) = 1$ angenommen, so müssen die Summanden $1 \cdot I(x_1) + 3 \cdot I(x_2)$ genau 1 ergeben, denn

$$
\begin{aligned}
&1 \cdot I(x_1) + 3 \cdot I(x_2) + 5 \cdot I(x_3) = 6 \text{ gdw.} \\
&1 \cdot I(x_1) + 3 \cdot I(x_2) + 5 \cdot 1 = 6 \text{ gdw.} \\
&1 \cdot I(x_1) + 3 \cdot I(x_2) = 1.
\end{aligned}
$$

Die hier entwickelte Methode überspringt nun das Prüfen der Interpretationen I' mit $(I'(x_1); I'(x_2); I'(x_3)) = (0; 1; 1)$ sowie I'' mit $(I''(x_1); I''(x_2); I''(x_3)) = (1; 1; 1)$, da sowohl I' als auch I'' zu einer linken Seite größer als k führen würden. Stattdessen wird $I(x_2) = 0$ angenommen. Die Interpretation I mit

$I(x_1) = 1$, $I(x_2) = 0$, $I(x_3) = 1$ führt zur Lösung. Somit unterscheidet sich die vorgestellte Methode von der Brute-Force-Methode. Sie kann mit gleich vielen oder weniger Vergleichen Lösungsmengen eines Constraints finden.

Sind w_1 und w_2 die niedrigsten Gewichte eines Gleichheitsconstraints c mit $k \in [1..w_1 + w_2)$, dann lässt sich c direkt ohne Backtracking auf Erfüllbarkeit untersuchen, da jedes Modell für c genau ein Literal aus $Lits(c)$ auf eins abbildet.

Proposition 3.29. *Es sei c ein PB-Constraint mit Gleichheitskomparator und $k \in [1..w_1 + w_2)$. Dann gilt $\sum_{x \in Lits(c)} I(x) = 1$ für alle Interpretationen I mit $I \models c$.*

Beweis.

Für $\sum_{x \in Lits(c)} I(x) > 1 \rightarrow I \not\models c$:

$\sum_{x \in Lits(c)} I(x) > 1 \Longrightarrow I(x_p) = 1 \wedge I(x_q) = 1$ mit $p, q \in [1..n]$, $p \neq q$

$k \in [1..w_1 + w_2) \implies$

$w_p \cdot 1 + w_q \cdot 1 > k \implies$

$\sum_{j=1}^{n} w_j \cdot I(x_j) > k \implies$

$\sum_{j=1}^{n} w_j \cdot I(x_j) \neq k \implies$

$I \not\models c$

Für $\sum_{x \in Lits(c)} I(x) = 0 \rightarrow I \not\models c$:

$k \geq 1 \implies$

$w_1 \cdot I(x_1) + ... + w_n \cdot I(x_n) \geq 1 \implies$

$w_1 \cdot 0 + ... + w_n \cdot 0 \geq 1 \implies$

$I \not\models c$

□

Proposition 3.30. *Es sei c ein PB-Constraint mit Gleichheitskomparator und $k \in [1..w_1 + w_2)$, dann gilt $c \equiv \sum_{j \in [1..n] \wedge w_j = k} 1 \cdot x_j = 1 \wedge \bigwedge_{j \in [1..n] \wedge w_j \neq k} \overline{x}_j$ bzw. $c \equiv \bot$, falls kein w_j existiert mit $w_j = k$.*

Beweis. Geht hervor aus Proposition 3.29. □

Beispiel.

$1 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 = 2$ gdw.

$1 \cdot x_2 = 1 \wedge \overline{x}_1 \wedge \overline{x}_3$ gdw.

$x_2 \wedge \overline{x}_1 \wedge \overline{x}_3$

Ist $k \in [1..w_1 + w_2)$ und gilt $w_j \neq k$ für alle $j \in [1..n]$, dann ist $c \equiv \bot$ gemäß Proposition 3.30.

4 Empirische Untersuchung

Die genannten Präprozesstechniken wurden untersucht anhand der 2290 Instanzen der PB-Competition 2016 mit einem Zeitlimit von einer Stunde und einem Speicherlimit von 7 GB. Die Kaktusplots zeigen zu jedem Zeitlimit t die Anzahl der Instanzen, die in maximal t Sekunden gelöst werden konnten.

Abbildung 3: Propagieren

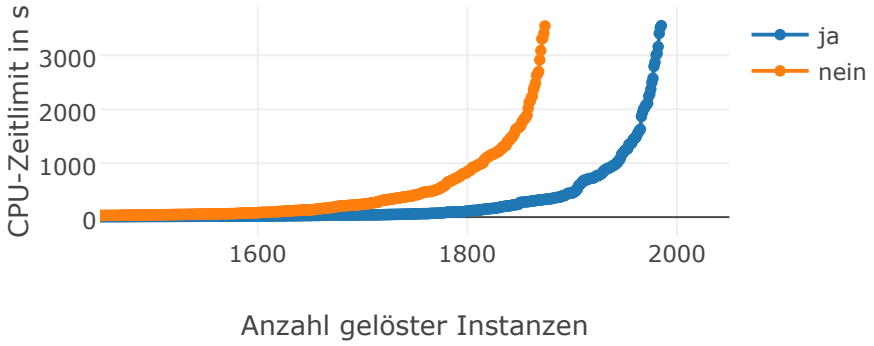


Abbildung 4: Aufteilen von PB-Constraints $\sum_{j=1}^{n} w_j \cdot x_j \leq k$ in PB- und Klauselteil

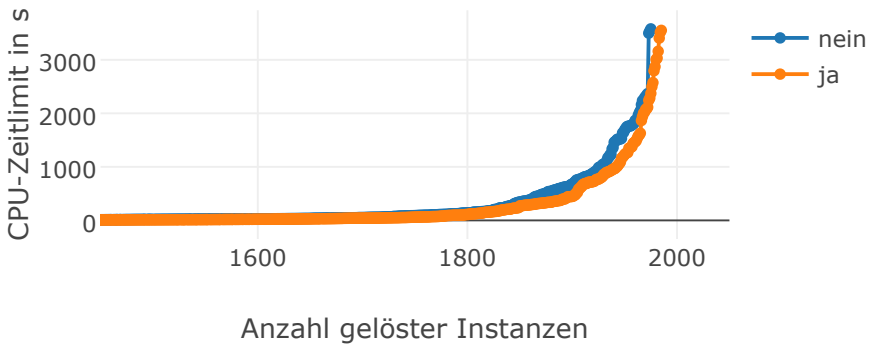
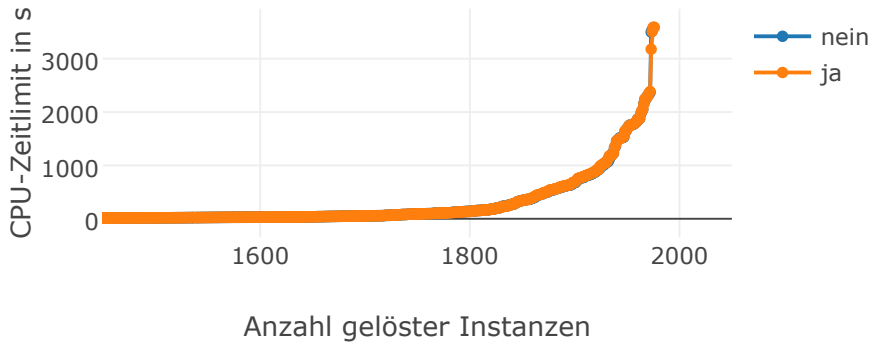


Abbildung 5: Belegen von Literalen durch Probing



Das Propagieren allgemein beschleunigt den Lösungsprozess stark (Abb. 3). Das Aufteilen von Constraints in PB- und Klauselteil (siehe Abschnitt 3.2.5) führt nur zu einem geringen Vorteile (Abb. 4). Diese Technik sollte auch auf lange Constraints ($n \geq 5$) angewendet werden. Die Probingmethode (Abschnitt 3.2.4) führt zu keinen deutlichen Verbesserungen (Abb. 5).

4.1 PB-Constraints als Disjunktionen

Ein PB-Constraint kann gemäß den Präpositionen 3.13 und 3.15 in eine Disjunktion umgewandelt werden. Diese Umwandelung ist nicht in jedem Fall hilfreich (Abb. 6). Es ist vorteilhaft, eventuelle Klauseln als PB-Constraint an den PB-Solver zu übergeben (Abb. 7). Die Umwandelung von $\sum_{j=1}^{n} w_j \cdot x_j \leq k$ in eine Disjunktion kann an weitere Bedingungen geknüpft werden, nämlich falls $\top$, falls $n \leq 5$ oder falls $\bot$. PB-Constraints in Disjunktionen umzuwandeln verlangsamt den Lösungsprozess bei Erfüllbarkeitsproblemen der Familien Elffers, Nossum und Quimper (Abb. 8). Bei gleichem Zeitlimit konnten mehr Bigint-Optimierungsprobleme der Familie Lion9-Single-Obj gelöst werden, wenn PB-Constraints bis zur Länge 5 in Klauseln umgewandelt wurden (Abb. 9). Die Optimierungsprobleme mit kleinen Integers konnten mit und ohne Umwandelung von PB-Constraints in Klauseln innerhalb von einer Sekunde gelöst werden.

4.2 Resolution

Das Erstellen von Resolventen bringt einen geringen Vorteil. Es sollte maximal eine Resolvente $\sum_{j=1}^{n} w_j \cdot x_j \leq k$ pro Formel erstellt werden (Abb. 10). Sie sollte nur hinzugefügt werden, falls $n \leq 2$ (Abb. 11). Ein derartiges Constraint kann anschließend in eine Klausel umgewandelt werden (siehe Kapitel 3.2.6).

Abbildung 6: Umwandelung von PB-Constraints in Klauseln, wenn möglich

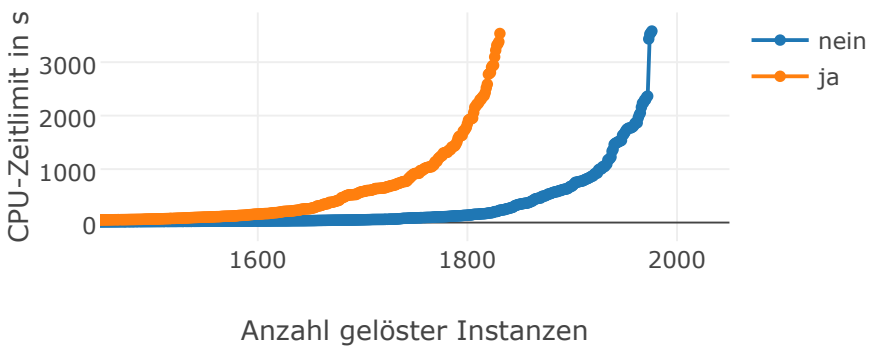


Abbildung 7: Weitergabe der Klauseln als ...

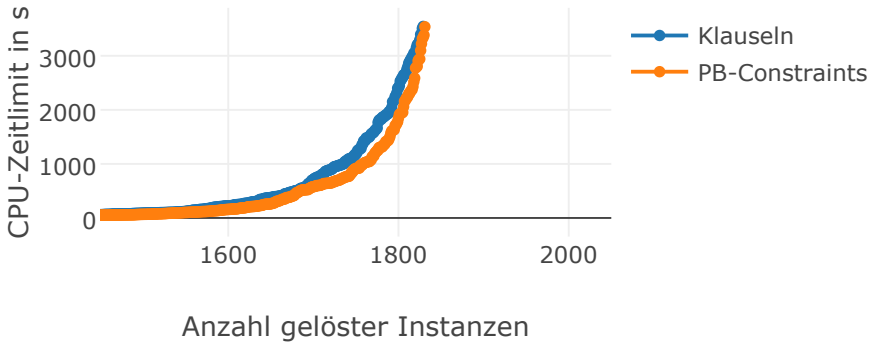


Abbildung 8: Umwandelung von PB-Constraints in Klauseln, wenn möglich, für Elffers, Nossum und Quimper, wenn

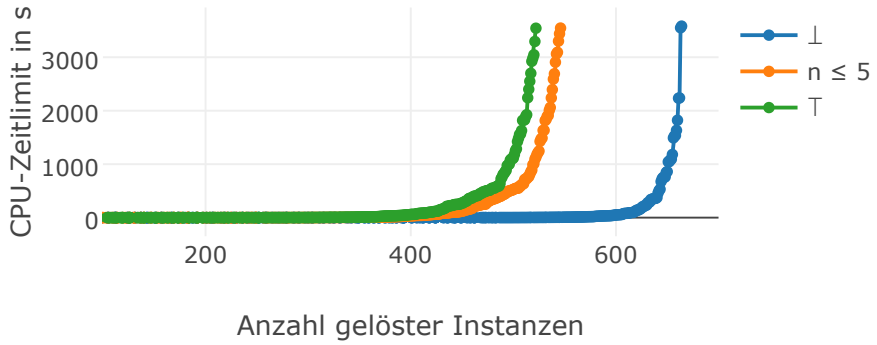


Abbildung 9: Umwandelung von PB-Constraints in Klauseln, wenn möglich, für Lion9-Single-Obj, OPT-BIGINT-LIN, wenn

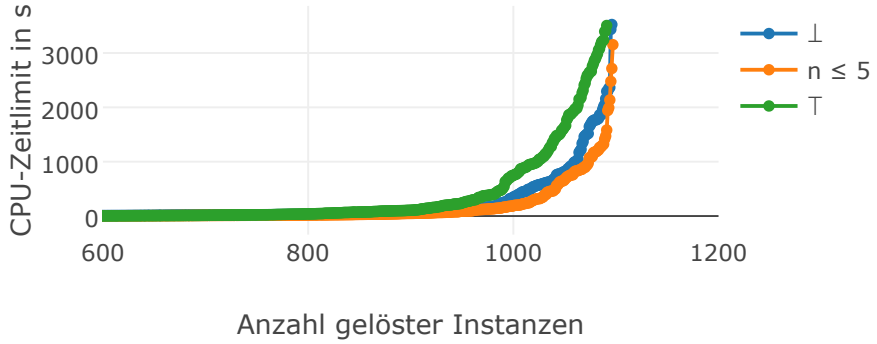


Abbildung 10: Maximale Anzahl der erstellten Resolventen

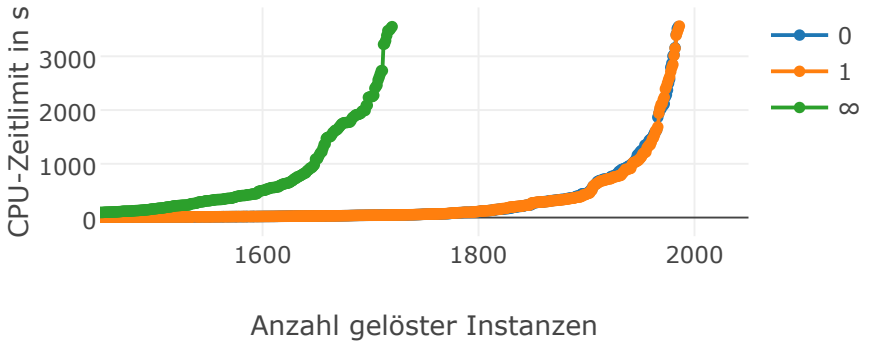


Abbildung 11: Maximale Länge der erstellten Resolventen

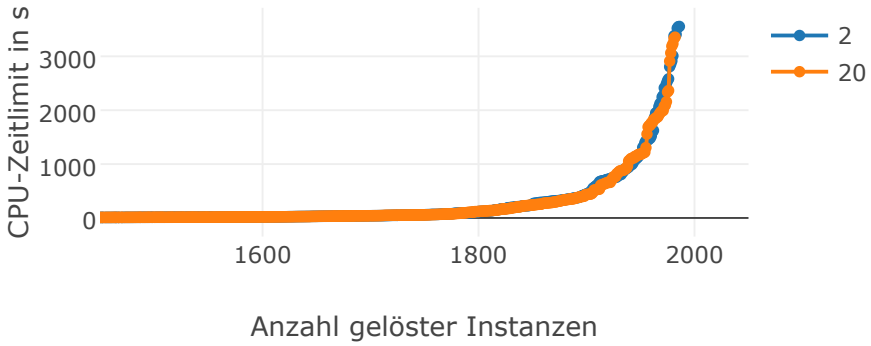
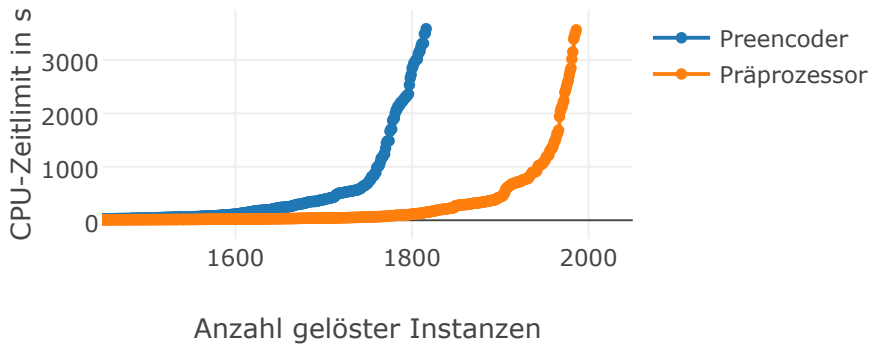


Abbildung 12: Performancegewinn durch den Präprozessor gegenüber Preencoder



5 Fazit und weiterführende Arbeiten

Durch Präprozessortechniken können in der selben Zeit mehr Formeln gelöst werden (Abb. 12). Sehr vorteilhaft sind Propagieren, Aufteilen von PB-Constraints in PB- und Klauselteil, das Hinzufügen maximal einer Resolvente mit der maximalen Länge 2, dabei keine Umformung von PB-Constraints in Klauseln.

5.1 Ausblick

Lange Disjunktionen als PB-Constraint zu belassen statt sie in Klauseln umzuwandeln führt bei 4 von 5 getesteten Instanzenfamilien zu einem Performancegewinn (Abschnitt 4.1). Es erweist sich sogar als Vorteil, dem PB-Solver Klauseln als PB-Constraint zu übergeben. Es stellt sich die Frage, ob sich die Performance eines SAT-Solvers ebenfalls steigert, wenn eine Klausel in ein PB-Constraint umgewandelt und anschließend per PB-Encoding in mehrere Klauseln und Hilfsvariablen kodiert wird.

Literaturverzeichnis

[E+01] **Ehrig**, Hartmut u. a.: *Mathematisch-strukturelle Grundlagen der Informatik*. Springer, 2001. – 321 S.

[EB05] **Een**, Niklas ; **Biere**, Armin: Effective preprocessing in SAT through variable and clause elimination. In: *Lecture notes in computer science* 3569 (2005), S. 61–75

[ES06] **Eén**, Niklas ; **Sörensson**, Niklas: Translating Pseudo-Boolean Constraints into SAT. In: *JSAT* 2 (2006), Nr. 1-4, 1–26. http://dblp.uni-trier.de/db/journals/jsat/jsat2.html#EenS06

[PS15] **Philipp**, Tobias ; **Steinke**, Peter: PBLib – A Library for Encoding Pseudo-Boolean Constraints into CNF. In: **Heule**, Marijn (Hrsg.) ; **Weaver**, Sean (Hrsg.): *Theory and Applications of Satisfiability Testing – SAT 2015* Bd. 9340. Springer International Publishing, 2015. – ISBN 978–3–319–24317–7, S. 9–16

[RM09] **Roussel**, Olivier ; **Manquinho**, Vasco M.: Pseudo-Boolean and Cardinality Constraints. Version: 2009. http://dblp.uni-trier.de/db/series/faia/faia185.html#RousselM09;http://dx.doi.org/10.3233/978-1-58603-929-5-695. In: **Biere**, Armin (Hrsg.) ; **Heule**, Marijn (Hrsg.) ; **Maaren**, Hans van (Hrsg.) ; **Walsh**, Toby (Hrsg.): *Handbook of Satisfiability* Bd. 185. IOS Press, 2009. – ISBN 978–1–58603–929–5, 695–733

[Sat] *The international SAT Competitions web page*. www.satcompetition.org, . – 14.03.2017