Logikbasierte Optimierungsverfahren für die Bedarfsprognose

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Dipl.-Math. Thore Kübart
aus Berlin

Tübingen
2015

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	16. Dezember 2015
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Wolfgang Küchlin
2. Berichterstatter:	Prof. Dr. Peter Hauck

Inhaltsverzeichnis

1 Einleitung

Die kundenindividuelle Massenproduktion („Mass Customization“) ist ein Trend, dem immer mehr produzierende Unternehmen folgen. So besteht – je nach Branche – ein entscheidender Wettbewerbsfaktor in der Befriedigung der individuellen Kundenbedürfnisse, ohne gleichzeitig die Kosten gegenüber einer Standardvariante stark erhöhen zu müssen. Die durch die kundenindividuelle Produktkonfiguration gewonnenen Informationen dienen außerdem der Marktforschung und werden bei der Entwicklung neuer Produkte eingesetzt.
Im Fall von technisch komplexen Produkten, wie Automobilen der Premiumklasse, führt die Variantenvielfalt jedoch zu einer Vielzahl von Herausforderungen. Bereits die korrekte Dokumentation der technisch produzierbaren Fahrzeugvarianten ist nicht ohne moderne Validierungsverfahren der Informatik möglich. So kann der Kunde seine individuelle Konfiguration auf Basis mehrerer hundert Ausstattungsoptionen zusammenstellen. Die dabei einzuhaltenden Restriktionen werden mittels Aussagenlogik in der Produktdokumentation beschrieben, deren Komplexität nur noch vom Computer beherrschbar ist.
Die exorbitant hohe Anzahl möglicher Konfigurationsvarianten stellt auch die Bedarfsplanung vor Herausforderungen, in der z. B. die Verbauraten zukünftiger Sonderausstattungen prognostiziert werden. In dieser Arbeit werden mathematisch exakte Methoden entwickelt, mit denen ein Planer von Verbauraten den durch die Produktdokumentation beschriebenen Raum der möglichen Konfigurationsvarianten vollständig berücksichtigen kann. Die Methoden ermöglichen es, einen zur Produktdokumentation konsistenten Bedarfsplan zu entwickeln. Die Teilebedarfsprognose unterstützend können außerdem die durch einen Bedarfsplan induzierten Verbauraten aller Bauteile evaluiert werden, die mit der Produktdokumentation logisch verknüpft sind.

1.1 Zweistufige Produktdokumentation

Premiumhersteller der Automobilindustrie dokumentieren ihre Palette variantenreicher Serienprodukte auf zwei Ebenen (siehe Abbildung 1).
Die erste Ebene ist die *Merkmalsebene*. Typische Merkmale sind durch die Baureihe, die Karosserieform, den verbauten Motor, das Verbraucherland, die Lenkung („Linkslenker“ oder „Rechtslenker“) und die Ausstattungslinie gegeben. Neben den möglichen Ausprägungen dieser Merkmale können bei der Produktkonfiguration zusätzliche Optionen, wie z. B. „Schiebedach“ oder „Spurhalte-Assistent“, ausgewählt werden. Sowohl die Merkmalsausprägungen als auch die Sonderausstattungen werden mittels sogenannter Option-Codes (kurz: Codes) dargestellt, so dass ein Kundenauftrag letztendlich einer Codekombination bzw. einer Codeliste entspricht. Im Falle der Mercedes-Benz S-Klasse existieren z. B. etwa 200 kundenwählbare Sonderausstattungscodes in Deutsch-

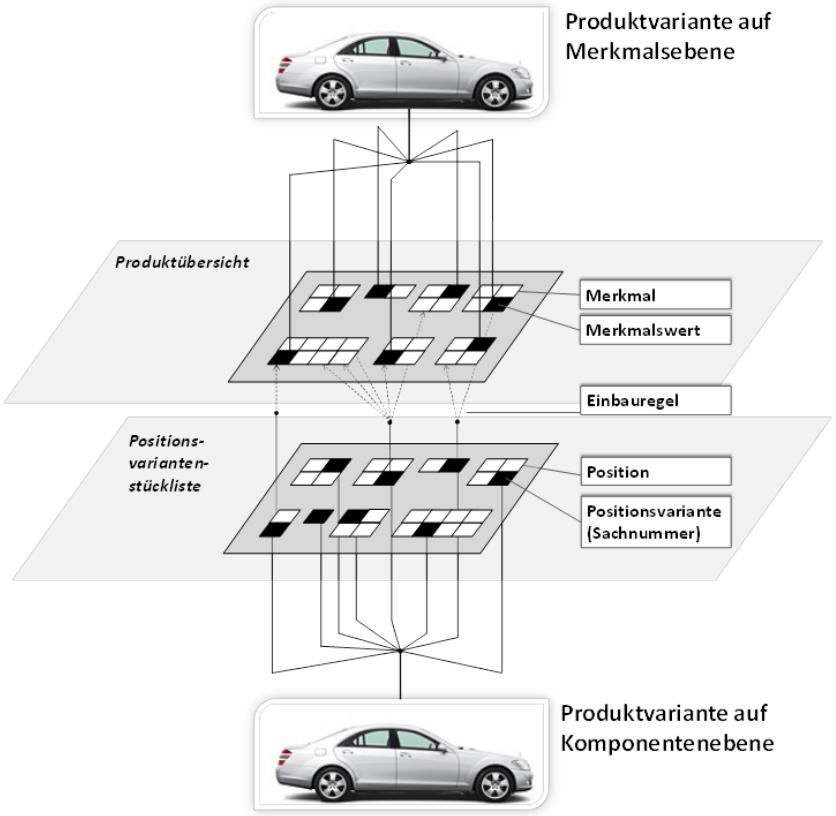

Abbildung 1: Zweistufige Produktdokumentation, entnommen aus [Stä07], S. 62

land, wobei für eine typische Kundenvariante die Codeliste 30 bis 40 der wählbaren Codes enthält.
Die *Produktübersicht* definiert auf der Merkmalsebene, welche Codekombinationen in welchem Markt über welchen Zeitraum angeboten werden. Sie ist im Wesentlichen durch technische, vertriebliche, juristische und marktspezifische Restriktionen bestimmt. Dabei werden die Restriktionen in Form sogenannter *Coderegeln* dokumentiert, d. h. als aussagenlogische Regeln mit den Codes als Variablen.
In Abbildung 2 sind beispielhafte Sonderausstattungen zusammen mit ihren Codes aufgeführt. Die Darstellung ist aus der aktuellen Preisliste der E-Klasse entnommen und zeigt einige Restriktionen, die der Kunde bei der Auftragskonfiguration zu beachten hat.

Die zweite Ebene der Produktdokumentation ist die *Teilebene*. Dort werden alle zum Fahrzeugaufbau benötigten Teile in Form von *Sachnummern* dokumentiert. Die korrekte Zuordnung der Sachnummern erfolgt bei Daimler durch die sogenannte *Positionsvariantenstückliste*. An einer *Stücklistenposition* werden alle die Sachnummern als *Positionsvarianten* zusammengefasst, deren Teile im Fahrzeug an einem Ort verbaut werden können. Jeder möglichen Positionsvariante an einer Stücklistenposition ist eine Einbauregel (Coderegel) zugeordnet. Eine Einbauregel evaluiert für eine Fahrzeugvari-

Sonderausstattungen. Limousine und T-Modell.

L = nur als Limousine erhältlich

● lieferbar gegen Mehrpreis
○ als Sonderausstattung ohne Mehrpreis lieferbar
□ Serienausstattung
– nicht lieferbar

	Code	E 200 BlueTEC	E 220 BlueTEC	E 220 BlueTEC[M] Edition	E 250 BlueTEC	E 250 BlueTEC 4MATIC	E 300 BlueTEC	E 300 BlueTEC HYBRID	E 350 BlueTEC	E 350 BlueTEC 4MATIC	E 200
ANBAUTEILE											
Anhängevorrichtung mit ESP® Anhängerstabilisierung [1) 2)] mechanisch abklappbar	550	●	●	● [10]	●	●	●	● [3]	●	●	●
ANTRIEB UND FAHRWERK											
7G-TRONIC PLUS inklusive DIRECT SELECT Wählhebel und Lenkradschaltpaddles	427	●	–	–	–	□	–	□	–	□	●
9G-TRONIC inklusive DIRECT SELECT Wählhebel und Lenkradschaltpaddles	421	–	●	●	●	–	□	–	□	–	–
AIRMATIC mit stufenloser Dämpfungsregelung [4)]	489	● [11]	● [11]	● [10] –	● [11]	● [11]	● [11]	– [11]	● [11]	● [11]	● [11]
DIRECT CONTROL Fahrwerk tiefergelegt und sportlich abgestimmt	677	⑤ [7]	⑤ [7]	□ –	⑤ [7]	6 [7]	⑤ [7]	⑤ [7]	⑤ [7]	6 [7]	⑤ [7]
DIRECT CONTROL Fahrwerk komfortabel abgestimmt [8) 9)]	485	○	○	○	○	○	○	○	○	○	○
Zweiflutige Abgasanlage mit eckigen, im Stoßfänger integrierten Endrohrblenden in Edelstahl [12)]	B16	●	●	●	●	●	●	●	●	●	●

1) Informationen über die zulässige Anhängelast entnehmen Sie den Technischen Daten
2) nicht in Verbindung mit Leichtmetallräder, Code R17, 644, oder 770
3) max. 300 kg Anhängelast (z.B. Fahrradträger)
4) nur in Verbindung mit AVANTGARDE, Code 954, oder ELEGANCE, Code 955 und 7G-TRONIC PLUS, Code 427, oder 9G-TRONIC, Code 421
5) im Lieferumfang Sport-Paket Exterieur, Code P96, oder Sport-Paket AMG, Code 950, ohne Mehrpreis lieferbar
6) im Lieferumfang Sport-Paket Exterieur, Code P96, oder Sport-Paket AMG, Code 950, bereits enthalten
7) in Verbindung mit Serienausstattung, Code 953, oder ELEGANCE, Code 955; im Lieferumfang AVANTGARDE, Code 954, bereits enthalten
8) im Lieferumfang Serienausstattung, Code 953, oder ELEGANCE, Code 955, bereits enthalten
9) nur in Verbindung mit AVANTGARDE, Code 954; nicht in Verbindung mit Sport-Paket Exterieur, Code P96, oder Sport-Paket AMG, Code 950
10) nicht in Verbindung mit Leichtmetallräder, Code R12, 34R (Limousine), oder 08R, 44R (T-Modell)
11) nur für T-Modell
12) nur in Verbindung mit Serienausstattung, Code 953, oder ELEGANCE, Code 955;
im Lieferumfang AVANTGARDE, Code 954, Sport-Paket Exterieur, Code P96, bereits enthalten; nicht in Verbindung mit Sport-Paket AMG, Code 950

Abbildung 2: Sonderausstattungen in der Preisliste

ante genau dann zum Wahrheitswert *true*, wenn die zugehörige Positionsvariante für den Fahrzeugaufbau an dieser Position zu verbauen ist. Über eine *Stücklistenauflösung* werden so insgesamt ca. 10.000 Bauteile für eine typische Konfiguration der E-Klasse ermittelt. In Tabelle 1 sind verschiedene Bauteile und deren Einbauregeln aufgelistet. Die Stoßfänger A und B sind in der originalen Stückliste einer Position zugeordnet. Wie in den entsprechenden Einbauregeln zu erkennen ist („+“= $\wedge$, „/“= $\vee$, „-“= $\neg$), schließen sich die beiden Positionsvarianten gegenseitig aus.

Eine detaillierte Darstellung der zweistufigen Produktdokumentation ist zum Beispiel in den Arbeiten von Sinz [KS00, Sin03, SKK03] zu finden. Dort werden SAT-basierte[1] Validierungsverfahren zur Konsistenzprüfung der Produktdokumentation beschrieben.

[1] SAT ist das Erfüllbarkeitsproblem der Aussagenlogik, d. h. das Problem zu entscheiden, ob eine aussagenlogische Formel erfüllbar ist.

Sachnummer	**Bezeichnung**	**Einbauregel**
A 219 540 57 45	STEUERGERÄT	((470/475)+(494/762/763))
A 219 735 08 10	GLASSCHEIBE	((494+(691+-596))/(691+-596))
A 219 820 05 15	LEITUNGSSATZ	((359+(389+(807+(-386+-388))))/(359+(-386+(-388+-807))))
A 219 820 07 61	LEUCHTEINHEIT	(494+(615+(-614+(-613+-616))))
A 219 880 06 71	STOSSFAENGER A	(220+(889+-**772**))
A 219 880 11 71	STOSSFAENGER B	(((M001+(M113+(M55+(220+(**772**+889)))))/(M156+(M63+(220+(**772**+889)))))+-((M001+(M113+(M55+030)))))

Tabelle 1: Einbauregeln für Positionsvarianten

Der Test auf Doppeltreffer überprüft beispielsweise, ob eine Fahrzeugkonfiguration existiert, die einerseits die Produktübersicht erfüllt, andererseits jedoch an einer Position in der Stückliste mehrere Positionsvarianten zieht.

1.2 Materialbedarfsplanung

Eine weitere Herausforderung für Anbieter komplexer, kundenindividueller Serienprodukte ist die Materialbedarfsplanung. Schon weit vor Eingang der vollständig spezifizierten Kundenaufträge müssen Premiumautomobilhersteller planerische Vorbereitungen treffen, um die internen und externen Produktionskapazitäten einzurichten. Das Einhalten kurzer Lieferzeiten bei einer gleichzeitig hohen Liefertreue ist entscheidend für die Kundenzufriedenheit. Durch eine frühzeitige Prognose des Teilebedarfs (Sekundärbedarfs) werden die Kosten reduziert, die aufgrund zu hoher bzw. zu niedriger Produktionskapazitäten entstehen. Die Komplexität einer solchen Sekundärbedarfsprognose leitet sich aus den beiden folgenden Tatsachen ab:

1. Es gibt Märkte, wie beispielsweise Deutschland, in denen nahezu keine zwei Kundenaufträge identisch sind. Die tatsächlich bestellten Fahrzeugkonfigurationen stellen dennoch nur einen sehr geringen Bruchteil der tatsächlich möglichen Codekombinationen dar. So existieren für die Mercedes-Benz E-Klasse mehr als 10^{50} unterschiedliche Bestellmöglichkeiten.

2. Die Produktdokumentation ist durch neue Baureihen, vertriebliche Entscheidungen usw. einem ständigen Wandel unterzogen. Diesem Wandel unterliegen auch die für den Zusammenbau eingesetzten Bauteile.

Eine einfache Zeitreihenanalyse für die aus planerischer Sicht kritischen Bauteile ist demnach nicht sinnvoll (vgl. [Stä07], S.83-85). Die zukünftigen Bedarfe werden viel-

mehr auf der Merkmalsebene abgeschätzt. Auf dieser können vergleichbare Sonderausstattungen der Vergangenheit verhältnismäßig einfach identifiziert und in die Zukunft projiziert werden.
In Tabelle 2 ist ein beispielhafter Ausschnitt einer *Sonderausstattungsprognose* des Automobilherstellers Daimler dargestellt. Derartige Sonderausstattungsprognosen beinhalten etwa 150 bis 200 *Bedarfsintervalle* und werden für einen Zielzeitraum erstellt, der 2 Jahre in der Zukunft liegt und 12 Monate lang ist. Die Bedarfsintervalle geben an, in welchem Bereich die jeweilige Verbauquote eines Codes erwartet wird. Sonderausstattungsprognosen beziehen sich bei Daimler üblicherweise auf die Ebene, in der die Merkmale „7-stelliges Baumuster“ und Verbrauchermarkt fixiert sind. Ein 7-stelliges Baumuster (BM7) bezeichnet eine Kombination bestehend aus Typklasse (Baureihe plus Karosserieform), Motor und Lenkung. Bei mehr als 1000 verschiedenen 7-stelligen Baumustern und etwa 100 verschiedenen Verbrauchermärkten werden Sonderausstattungsprognosen daher nur für die umsatzstärksten BM7/Markt-Kombinationen angefertigt. Bei geringeren Absatzzahlen werden Märkte in Marktgruppen zusammengefasst oder die Prognose wird unter Umständen auf der höheren Ebene der Typklasse erstellt. Da die Angaben einer Sonderausstattungsprognose relativ sind, werden außerdem die Anteile der einzelnen BM7/Markt-Kombinationen an der Typklasse prognostiziert. Die tatsächlichen Stückzahlen ergeben sich aus dem jeweiligen, gegebenenfalls noch angepassten BM7-Anteil und der absoluten Stückzahl der Typklasse. Aus den prognostizierten Stückzahlen leiten sich zudem auch die Bedarfe der Serienausstattungen ab.

Dieses auf der Merkmalsebene vorhergesagte Kundenverhalten muss in die Teileebene transformiert werden. In der Literatur sind unterschiedliche Ansätze zu finden, die sich dieser Herausforderung widmen:

1. Prognose von Auftragsprogrammen[2] mit anschließender Auflösung der Stückliste, [SR13, SR14]
2. Modellierung von Abhängigkeitsstrukturen und Auswertungen mit Methoden des Data-Mining, [Hol00]
3. Punktprognosen auf Basis der Einbauregeln, [Ohl00]
4. Intervallprognosen auf Basis der Einbauregeln, [Stä07, KB11].

Diesen Verfahren dienen sogenannte *Plansachverhalte* als Eingangsdaten, bestehend aus:

- der Produktdokumentation (Produktübersicht, Stückliste)
- statistischen Vorgaben (Code-Prognosen, Stückzahlen, Bauteilerestriktionen usw.)

[2] Auftragsprogramme werden durch eine Multimenge von Fahrzeugkonfigurationen beschrieben, die die Produktübersicht des Zielzeitraumes erfüllen.

Sonderausstattung	**Code**	**Bedarfsintervall**
Reifendruckkontrolle low-line	470	$20\% - 25\%$
Reifendruckkontrolle high-line	475	$2\% - 10\%$
USA-Ausführung	494	$18\% - 23\%$
Funkfernbedienung mit Panic Schalter	762	$3\% - 10\%$
Funkfernbedienung ohne Panic Schalter	763	$20\% - 25\%$

Tabelle 2: Auszug aus einer Sonderausstattungsprognose

Die ersten 3 Ansätze evaluieren Häufigkeitswerte für Bauteile. Der 4. Ansatz wurde erstmals von Thomas Stäblein formuliert. In seiner Dissertation [Stä07] kritisierte er an den ersten 3 Ansätzen, dass Plansachverhalte üblicherweise keine festen Verbauhäufigkeiten auf der Teileebene induzieren. In der Regel sind mehrere Auftragsprogramme möglich, die die Plansachverhalte erfüllen. Unterschiedliche Auftragsprogramme führen wiederum zu unterschiedlichen Teilebedarfen.
Ein Beispiel hierzu liefert die Einbauregel des Steuergerätes aus Tabelle 1 in Kombination mit der Sonderausstattungsprognose aus Tabelle 2. Für die in der Einbauregel enthaltenen Codes gibt die Sonderausstattungsprognose Bedarfsintervalle vor. Angenommen, die Produktübersicht erlaubt höchstens eine Reifendruckkontrolle und höchstens eine Funkfernbedienung. Ansonsten seien die 5 Codes frei kombinierbar. Dann können die insgesamt 18 baubaren Code-Kombinationen der 5 Codes derart unterschiedlich verteilt vorliegen, dass die induzierten Code-Häufigkeiten zwar jeweils in den Bedarfsintervallen der Sonderausstattungsprognose liegen, der Bedarf des Steuergerätes, d. h. die induzierte Häufigkeit, mit der die Einbauregel zu *true* evaluiert, jedoch um den Faktor 3 variiert.
Stäblein schlägt ein Verfahren vor, das auf Basis von

- Einbauregeln (Coderegeln) und
- Bedarfsintervallen für Codes (Code-Intervalle)

induzierte Bedarfsintervalle für Teile (Teile-Intevalle) mit folgender *Eigenschaft (I)* berechnet: Unter der Voraussetzung, dass die Code-Intervalle eintreten, liegt die tatsächliche Verbauhäufigkeit des Bauteiles im berechneten Intervall.
Dazu stellt Stäblein die Einbauregeln in binären Formelbäumen dar, um Unter- und Obergrenzen, beginnend bei den Blättern – den Codes – entlang den Baumstrukturen, bis hin zu den Wurzelknoten, zu propagieren [Stä07].
Mit der Berechnung entsprechender Intervalle gelingt es Stäblein den Bauteilebedarf einzugrenzen. Da in seinem Verfahren jedoch keine Informationen aus der Produktübersicht genutzt werden und die von ihm berechneten Intervalle von der Darstellung der Einbauregeln abhängen, ist es ihm nicht möglich zu beantworten, inwieweit die Code-Intervalle aufgrund einer vorliegenden Produktdokumentation noch engere Teile-

Intervalle induzieren. Für eine verlustfreie Übersetzung der Plansachverhalte in Sekundärbedarfe sind die Intervalle zu bestimmen, die die Eigenschaft (I) erfüllen und außerdem so klein wie möglich sind. Intervalle mit dieser Eigenschaft werden in dieser Arbeit auch als *Minimalintervalle* bezeichnet. Sie sind genau dann eindeutig, wenn die vorgegebenen Plansachverhalte von einem Auftragsprogramm erfüllt werden können.

Die Methode von Stäblein wurde durch Jochen Kappler in seiner Dissertation weiterentwickelt, indem er die Auswertung der Einbauregeln verfeinert hat [KB11]. Mit der Unterstützung eines SAT-Solvers berechnet er für einen inneren Knoten im binären Formelbaum, zu welchen Wahrheitswerten die beiden Teilformeln unter der Voraussetzung evaluieren können, dass die Produktübersicht simultan erfüllt wird. Je nach Konstellation ist es ihm so möglich, schärfere Intervallschranken zu berechnen. Allerdings sind auch die von Kappler berechneten Teile-Intervalle nicht minimal. So besteht weiterhin das Problem, dass die berechneten Intervalle von der Darstellung der Einbauregeln abhängen. Auch werden bei Kapplers Vorgehen nicht alle impliziten Restriktionen der Produktübersicht berücksichtigt.

Die korrekte und verlustfreie Übersetzung der Plansachverhalte in die für die Logistik relevanten Sekundärbedarfe – bei vollständiger Berücksichtigung der Produktdokumentation – ist ein bisher ungelöstes Problem. Hier setzt die vorliegende Arbeit an.

1.3 Ziele der Arbeit

Wie im vorangegangenen Abschnitt erläutert wurde, existieren zwei unterschiedliche Klassen von Verfahren, die Plansachverhalte in Sekundärbedarfe transformieren:

- Scheingenaue Verfahren, die auf fixe Verbauquoten der Bauteile schließen.
- Verlustbehaftete Verfahren, die zu große Verbauquotenintervalle berechnen.

Das primäre Ziel dieser Arbeit ist die Entwicklung einer Methodik, die in möglichst kurzer Zeit unter anderem Folgendes leistet:

(Z1) Die korrekte und verlustfreie Übersetzung von vorgegebenen Plansachverhalten in Sekundärbedarfe, d. h. die Berechnung der in Unterabschnitt 1.2 eingeführten Minimalintervalle.

Der Begriff „Minimalintervalle“ ist jedoch nur dann sinnvoll, wenn die statistischen Vorgaben konsistent zur Produktdokumentation sind, d. h. wenn mindestens ein Auftragsprogramm existiert, das die Produktdokumentation und die statistischen Vorgaben erfüllt. Stäblein und Kappler war kein vollständiges Entscheidungsverfahren bekannt, das diese Art der Konsistenz nachweist [Stä07], S.2:

„Eine durchgängige Möglichkeit zur Verifikation von Prognosen im Blicke der Konsistenz zu den Planungsvorgaben wie beispielsweise der Produktdokumentation oder dem Produktionsprogramm besteht heute nicht."
Daraus leitet sich als zweites Ziel dieser Arbeit ab:

(Z2) Die zu entwickelnde Methodik muss entscheiden können, ob eine Statistik, d. h. eine Menge statistischer Aussagen (Code-Prognosen, Bauteilerestriktionen, ...), konsistent zur Produktdokumentation ist.

Für einen Planer von Sonderausstattungen ist die Konsistenz ein nur schwer einzuhaltendes Kriterium. Insbesondere wegen der sich stets ändernden Produktdokumentation ist die Wahrscheinlichkeit hoch, dass der Planer nicht alle nötigen Zusammenhänge überblickt. Um nicht nur die Inkonsistenz der Plansachverhalte nachzuweisen und eine Übersetzung in die Sekundärbedarfe frühzeitig abbrechen zu müssen, wird als drittes Ziel dieser Arbeit formuliert:

(Z3) Es ist ein Verfahren zu entwickeln, das es dem Planer erlaubt, eine zur Produktdokumentation inkonsistente Menge von Bedarfsintervallen (z. B. inkonsistente Code-Prognosen), gemäß individuell definierter Prioritäten, in eine konsistente Menge von Bedarfsintervallen zu überführen.

Die Formulierung der individuell definierten Prioritäten soll beispielsweise durch eine quadratische Zielfunktion Q erfolgen können, die, als Metrik aufgefasst, den Abstand zwischen zwei Mengen von Bedarfsintervallen misst:
Löse

$$\underset{I \text{ ist konsistent zu PD}}{\arg\min} \; Q(\tilde{I}, I)$$

für die zur Produktdokumentation PD inkonsistente Menge von Bedarfsintervallen $\tilde{I}$.

1.4 Aufbau der Arbeit

In Kapitel 2 dieser Arbeit geht es zunächst darum, mit der Produktdokumentation zu rechnen. Dazu werden die Grundlagen von SAT-basierter Produktkonfiguration bereitgestellt. Anschließend wird am Beispiel des Produktdokumentationssystems PPM (Product-, Price- and Master-Data) aufgezeigt, wie eine Produktübersicht in eine aussagenlogische Formel übersetzbar ist. Die im System PPM enthaltene Produktübersicht entspricht einem aussagenlogischen Regelwerk, dessen iterative Anwendung innerhalb einer Validierung nichtbaubare Codekombinationen zu baubaren Codekombinationen verändert und somit implizit die Abhängigkeiten zwischen den verschiedenen Codes beschreibt. PPM umfasst das komplette Konfigurationswissen der Marken Mercedes-Benz und Smart. Neben den technischen Restriktionen werden in PPM insbesondere auch spezifische Verkaufsrestriktionen der einzelnen Märkte dokumentiert. Die von

Sinz [Sin97] beschriebene Formalisierung der Produktübersicht DIALOG ist im Rahmen einer Materialbedarfsplanung nicht ausreichend, da DIALOG nicht die marktspezifischen Restriktionen beinhaltet.
Auf Basis der in diesem Kapitel entwickelten Produktübersichtsformel von PPM werden abschließend numerische Ergebnisse zur SAT-basierten Konfiguration betrachtet.

Das Kapitel 3 dient der Vorbereitung für Kapitel 4. Es baut auf der in Kapitel 2 generierten Produktübersichtsformel auf und behandelt die optimale Produktkonfiguration unter einer linearen Zielfunktion. Das zugrunde liegende pseudo-boolesche Optimierungsproblem (PBO) wird mit einem SAT-basierten PBO-Solver und mit einem Standardsolver der ganzzahligen Programmierung gelöst. Dabei wird insbesondere auch auf die Theorie SAT-basierter PBO-Solver eingegangen. In Kombination mit den numerischen Ergebnissen wird eine optimale Solverkonfiguration für Kapitel 4 entwickelt.

In Kapitel 4 werden Verfahren konstruiert, mit denen die Ziele (Z1) bis (Z3) aus Abschnitt 1.3 erreichbar sind. Zunächst werden die zugehörigen Problemstellungen mathematisch definiert. Dies erlaubt einen Zusammenhang zum probabilistischen Erfüllbarkeitsproblem (PSAT) herzustellen. PSAT ist eine Verallgemeinerung von SAT und somit NP-vollständig. Analog zum modernsten Lösungsansatz für PSAT wird ein auf Column Generation basierendes Verfahren entwickelt. Motiviert durch die numerischen Ergebnisse aus Kapitel 2 wird das Verfahren in einem neuartigen Ansatz so erweitert, dass durch Primimplikanten gewährte Freiheitsgrade der Produktübersicht optimal ausgenutzt werden können.
Abschließend werden die numerischen Ergebnisse präsentiert. Dabei wird unter anderem der Informationsgewinn analysiert, der durch die Berechnung von Minimalintervallen entsteht. Für einen Vergleich wird außerdem der Informationsgewinn eines unvollständigen Verfahrens herangezogen, das Intervallschranken im Mikrosekundenbereich berechnet, simultan jedoch nur eine Auswahl von Plansachverhalten berücksichtigt.

Alle numerischen Ergebnisse dieser Arbeit wurden auf einem handelsüblichen Laptop erzielt, der die folgenden Daten aufweist:

- Windows 7 Professional 64 Bit,
- Intel(R) Core(TM) i7-4800MQ CPU mit 2.70 GHz,
- 16 GB Arbeitsspeicher.

2 Produktkonfiguration mit Aussagenlogik

2.1 Grundlagen

2.1.1 Aussagenlogik

Die *Syntax* der Aussagenlogik definiert den Aufbau aussagenlogischer Formeln. Eine aussagenlogische Formel setzt sich zusammen aus Aussagenvariablen ($x, y, z, \dots$), Konstanten ($false$ und $true$), Junktoren ($\neg$, $\vee$ und $\wedge$) und/oder Klammersymbolen. Die Menge aller aussagenlogischen Formeln $\mathcal{F}$ wird induktiv definiert:

- Aussagenvariablen sind Formeln,
- die beiden Konstanten sind Formeln,
- sind F und G Formeln, dann auch $(\neg F)$, $(F \wedge G)$ und $(F \vee G)$,
- nichts sonst ist eine Formel.

Neben den oben definierten Junktoren werden zusätzlich die Junktoren $\rightarrow, \leftrightarrow$ durch

$$(F \rightarrow G) := ((\neg F) \vee G) \text{ und } (F \leftrightarrow G) := ((F \rightarrow G) \wedge (G \rightarrow F))$$

eingeführt. Die Folge $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ listet die Junktoren mit abnehmender Bindungsstärke auf, sodass beispielsweise die Klammerung der Formel $(x \vee (\neg(y \wedge z)))$ auf $x \vee \neg(y \wedge z)$ reduziert werden kann.
Bezeichnet $\mathcal{V}$ die Menge aller Aussagenvariablen, dann definiert $\mathcal{L} := \mathcal{V} \cup \{\neg x | x \in \mathcal{V}\}$ die Menge aller *Literale*. Zu einem Literal l wird das Gegenstück $\bar{l}$ durch

$$\bar{l} := \begin{cases} x, & \text{falls } l = \neg x \\ \neg x, & \text{falls } l = x \end{cases}$$

definiert.
Für eine aussagenlogische Formel F ist $Var(F)$ die Menge der enthaltenen Aussagenvariablen.

Aussagenvariablen können mit den Wahrheitswerten 0 oder 1 belegt werden. Die *Semantik* der Aussagenlogik definiert, zu welchem Wahrheitswert $\nu(F)$ eine Formel F unter einer Variablenbelegung $\beta : Var(F) \rightarrow \{0, 1\}$ evaluiert. Die Definition erfolgt induktiv über den Formelaufbau:

- $\nu(x) := \beta(x)$, $x \in \mathcal{V}$,
- $\nu(true) := 1$, $\nu(false) := 0$,
- $\nu(\neg F) := 1 - \nu(F)$, $F \in \mathcal{F}$,

- $\nu(F \wedge G) := \min(\nu(F), \nu(G)),\ F, G \in \mathcal{F}$,

- $\nu(F \vee G) := \max(\nu(F), \nu(G)),\ F, G \in \mathcal{F}$.

Diese Definition setzt ein feste Belegung β voraus. Um den Wahrheitswert in Abhängigkeit der Belegung β formulieren zu können, wird für eine Formel F die Auswertung

$$F(\beta) := \nu(F)$$

unter β definiert, wobei ν passend zu β definiert ist.
Evaluiert eine aussagenlogische Formel F unter einer Variablenbelegung β zum Wahrheitswert 1 (d. h. $F(\beta) = 1$), so ist β ein *Modell* von F (Notation: $\beta \models F$), d. h. „die Belegung β erfüllt die Formel F“.

Definition 1 (Erfüllbarkeit)
Eine Formel F heißt *erfüllbar*, falls eine Variablenbelegung existiert, die F erfüllt.

Existiert hingegen kein Modell von F, so ist F eine *Kontradiktion* und heißt *unerfüllbar*. Eine Formel, die von jeder Variablenbelegung erfüllt wird, heißt *Tautologie*. Eine Formel G *folgt semantisch* aus einer Formel F (Notation: $F \models G$), falls für jede Variablenbelegung

$$\beta : M \to \{0, 1\},\ Var(F) \cup Var(G) \subseteq M$$

aus $\beta \models F$ die Beziehung $\beta \models G$ folgt. Dies ist äquivalent dazu, dass die Formel $F \to G$ eine Tautologie ist. Zwei Formeln F und G heißen *semantisch äquivalent*, falls $F \models G$ und $G \models F$ gilt (Notation: $F \equiv G$).

Eine zentrale Fragestellung der Aussagenlogik ist durch das Erfüllbarkeitsproblem (SAT) gegeben.

Definition 2 (SAT)
SAT ist das Problem, zu entscheiden, ob eine aussagenlogische Formel erfüllbar ist.

Im nachfolgenden Unterabschnitt werden Algorithmen behandelt, die das Erfüllbarkeitsproblem entscheiden. In diesem Zusammenhang wird an dieser Stelle die *konjunktive Normalform* (KNF) definiert. Sie ist eine Konjunktion von *Klauseln* (Literal-Disjunktionen) und genügt somit der Form

$$K_1 \wedge \cdots \wedge K_m, \quad K_i = l_{i_1} \vee \cdots \vee l_{i_k}, \quad l_{i_j} \in \mathcal{L}. \tag{1}$$

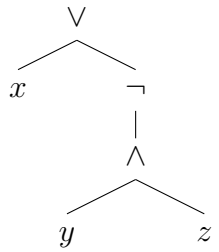

Abbildung 3: Formelbaum für $x \vee \neg(y \wedge z)$

Jede aussagenlogische Formel kann mit Äquivalenzumformungen in eine KNF überführt werden. Die dafür nötige Anwendung des Distributivgesetzes, d. h.

$$(F \wedge G) \vee H \equiv (F \vee H) \wedge (G \vee H),$$

kann jedoch zu exponentiell vielen Klauseln führen. Durch das Einführen von Hilfsvariablen kann mit der Tseitin-Transformation jede aussagenlogische Formel ohne exponentielles Wachstum in eine erfüllbarkeitsäquivalente KNF überführt werden [Tse83]. Genauer gilt:

- Zu jedem Modell der originalen Formel existiert genau eine Erweiterung, die ein Modell der Tseitin-KNF darstellt.
- Werden die Hilfsvariablen in einem Modell der Tseitin-KNF ignoriert, so ergibt sich ein Modell der originalen Formel.

Die eingeführten Hilfsvariablen h substituieren Teilformeln ϕ_h der originalen Formel, d. h. durch Anwenden der Tseitin-Transformation werden zusätzliche Constraints der Form $h \leftrightarrow \phi_h$ formuliert. Dieser Ansatz wurde durch Plaisted und Greenbaum verfeinert [PG86]. Abhängig von der Polarität der Teilformel ϕ_h führen sie nur einen der beiden Constraints $h \rightarrow \phi_h$ und $h \leftarrow \phi_h$ ein. Die *Polarität* einer Teilformel ist über die Anzahl der Negationen ($\neg$) definiert, die im Formelbaum zwischen Wurzel und Teilformel auftreten. Bei gerader Anzahl liegt eine positive Polarität vor, bei ungerader Anzahl eine negative. Ein Beispiel für einen Formelbaum ist in Abbildung 3 zu sehen. Eine Einschränkung der Transformation nach Plaisted und Greenbaum ist dadurch gegeben, dass die Erweiterung eines Modells der originalen Formel zu einem Modell der berechneten KNF nicht eindeutig ist. So kann beispielsweise eine mit Plaisted und Greenbaum erzeugte KNF nicht zum Zählen der Modelle eingesetzt werden.

2.1.2 Automatisches Beweisen

SAT ist NP-vollständig [Coo71] und gehört somit zu den Problemen der Komplexitätsklasse NP: Ein existierender Beweis für die Erfüllbarkeit einer Formel ist mit polynomiellem Aufwand zu verifizieren, das Finden eines Beweises kann jedoch exponentiellen Aufwand benötigen (P$\neq$NP vorausgesetzt).

In der Industrie gibt es zahlreiche Problemstellungen, die sich als Instanzen von SAT formulieren lassen, teilweise Millionen von Aussagenvariablen benötigen und dennoch von modernen SAT-Solvern in nützlicher Zeit gelöst werden können. Prominente Beispiele sind durch die Hardware- und Software-Verifikation [CK96, KCY03] und durch die KFZ-Konfiguration gegeben [KS00, SKK03].

Beispiel 1
Für die Produktkonfiguration wird eine Fahrzeugvariante durch eine Codebelegung $\beta : C \to \{0,1\}$, d. h. durch eine Auswahl möglicher Option-Codes (kurz: Codes), beschrieben. Die baubaren Codekombinationen sind in einem logischen Regelwerk codiert, das als aussagenlogische Formel POF formuliert werden kann - die Modelle von POF sind genau die baubaren Varianten. Stellt sich beispielsweise die Frage, ob die Codes c_1 und c_2 in einer Variante zusammen verbaut werden können, so ist die Formel $POF \wedge c_1 \wedge c_2$ auf Erfüllbarkeit zu überprüfen.
Im Rahmen dieser Arbeit wurde die Formel der Daimler-Produktübersicht PPM aufgestellt (siehe Abschnitt 2.2).

Eine Möglichkeit, eine als KNF gegebene aussagenlogische Formel auf Erfüllbarkeit zu überprüfen, ist durch die aussagenlogische Resolution gegeben. Resolution beruht auf der Anwendung der Resolventenregel. Die Resolventenregel erzeugt aus zwei Klauseln eine weitere Klausel, die logisch folgt. Sie wird für Klauseln

$$K_1 = \neg z \vee l_1 \vee \cdots \vee l_m,\ K_2 = z \vee L_1 \vee \cdots \vee L_n \text{ durch}$$
$$Res(K_1, K_2) := l_1 \vee \cdots \vee l_m \vee L_1 \vee \cdots \vee L_n$$

definiert. Ist es möglich durch wiederholte Anwendung der Resolventenregel die leere Klausel $Res(\neg z, z)$ abzuleiten, so ist die vorliegende KNF unerfüllbar. Für einen Nachweis der Erfüllbarkeit müssen alle mögliche Resolventen gebildet werden, um ausschließen zu können, dass die leere Klausel ableitbar ist.

Die aktuell schnellsten Verfahren zum Lösen von SAT-Problemen basieren auf dem DPLL-Algorithmus (Davis-Putnam-Logemann-Loveland) von 1962 [DLL62], der eine Verbesserung des Davis-Putnam-Algorithmus von 1960 [DP60] darstellt. DPLL operiert auf einem binären Entscheidungsbaum - ein innerer Knoten repräsentiert eine Variable x, der linke Teilbaum von x die Entscheidung $\beta(x) = 0$, der rechte Teilbaum die Entscheidung $\beta(x) = 1$. Ziel ist es eine widerspruchsfreie Verzweigung zu einem Blatt, d. h. zu einem vollständigen Modell β, zu berechnen. In diesem Zusammenhang wird die Menge der Wahrheitswerte zu $\{0, 1, *\}$ erweitert, wobei $\beta(x) = *$ dahingehend zu interpretieren ist, dass x im Entscheidungsbaum noch nicht belegt wurde.
Eine Voraussetzung von DPLL ist, dass die Formel in konjunktiver Normalform (KNF) vorliegt. Die KNF-Darstellung ermöglicht es nach einer getroffenen Entscheidung mittels sogenannter *Unit-Propagation* weitere Variablenbelegungen abzuleiten: Existiert in

einer Klausel nur noch ein einziges Literal l mit $\beta(l) \neq 0$, so ist die Klausel *unit* und $\beta(l) = 1$ wird propagiert.
Führen die getroffenen Entscheidungen dazu, dass für alle Literale einer Klausel $\beta(l) = 0$ gilt, so liegt ein Konflikt vor. Mittels sogenanntem Backtracking wird die letzte Entscheidung zurückgenommen, deren benachbarter Teilbaum noch nicht untersucht wurde (chronological Backtracking). Wird eine Belegung β gefunden, unter der für jede Klausel mindestens ein Literal mit $\beta(l) = 1$ existiert, so ist die Erfüllbarkeit der Formel durch das Modell β nachgewiesen.
DPLL-basierte SAT-Solver haben sich durchgesetzt, da sie in SAT-Problemen aus der Praxis üblicherweise häufiger propagieren als entscheiden. Eine Instanz mit 100 Variablen, in der durchschnittlich 90% der Belegungen propagiert werden, hat höchstens die Komplexität einer Worst-Case-Instanz mit 10 Variablen - der tatsächliche Suchraum reduziert sich von $\leq 2^{100}$ auf $\leq 2^{10}$ Belegungen.

Der klassische DPLL-Algorithmus wurde in den Jahren um die Jahrtausendwende entscheidend verfeinert. Mit der Implementierung des *2-Watched-Literal-Konzeptes* konnte die Unit-Propagation wesentlich beschleunigt werden [MMZ+01]: Im Anschluss an eine Variablenbelegung ist es nicht nötig, jede Klausel auf eine mögliche Propagation zu überprüfen. Eine Klausel, die vor der Variablenbelegung noch 2 Literale mit $\beta(l) \neq 0$ beinhaltete, kann frühstens jetzt eine Variablenbelegung propagieren. Daher ist es ausreichend für jede Klausel zwei Literale zu beobachten, die $\beta(l) \neq 0$ erfüllen. Wird im Suchprozess eine neue Variable belegt, so müssen nur die Klauseln analysiert werden, für die eine der beiden beobachteten Literale in den Status falsified (d. h. $\beta(l) = 0$) übergeht. Unter den Literalen einer entsprechenden Klausel wird nach einem unbelegten Literal gesucht, welches ersatzweise beobachtet wird. Wird dabei ein Literal entdeckt, das schon mit dem Wahrheitswert 1 belegt ist, so ist die Klausel bereits erfüllt und muss bis zu einem entsprechenden Backtrack nicht weiter berücksichtigt werden. Existiert kein weiteres Literal mit $\beta(l) = 0$, so wird das zur Beobachtung verbliebende Literal propagiert.

Mit der Einführung einer Konfliktanalyse [SS96], bestehend aus den Techniken *Clause-Learning* und *non-chronological Backtracking*, konnten irrelevante Bereiche bei der Suche auf dem Entscheidungsbaum übersprungen werden. Moderne SAT-Solver lernen in der Konfliktanalyse redundante Klauseln, um unerfüllbare Teilbelegungen im weiteren Suchprozess direkt durch Unit-Propagation zu verhindern. Dabei ist die Auflösung eines Konfliktes nicht eindeutig. Die zu lernende Klausel wird so gewählt, dass genau ein Literal aus der Konfliktebene enthalten ist. Das Backtracking erfolgt in die nächst höhere Entscheidungsebene, in der die gelernte Klausel unit ist. Dort wird die Variablenbelegung aus der Konfliktebene per Unit-Propagation umgedreht. Die so gelernten Klauseln werden daher auch *Unique Implication Point* (UIP) genannt.

Die Berechnung der UIPs wird mit aussagenlogischer Resolution umgesetzt: Ausgehend von der Konfliktklausel K_1, deren Literale alle mit 0 belegt sind, und der Klausel R_1, die das letzte Literal l_1 aus K_1 propagiert hat, wird zunächst die Resolvente $K_2 = Res(K_1, R_1)$ berechnet. Auch K_2 ist eine Konfliktklausel – jedoch ohne l_1. Sollte K_2 noch mehrere Literale aus dem Konfliktlevel enthalten, so kann mit dem Resolventen $K_3 = Res(K_2, R_2)$ ein weiteres Literal l_2 entfernt werden. Dabei ist l_2 das zuletzt propagierte Literal aus K_2 und R_2 ist der Grund für l_2. Nach und nach können so Konfliktklauseln berechnet werden, in denen ein weiteres Literal aus der Konfliktebene eliminiert wird. Mindestens eine Resolvente der entstehenden Folge von Konfliktklauseln (K_i) ist daher ein UIP.

Die dargestellten Techniken der Literalbeobachtung und der Konfliktanalyse werden in Abschnitt 3.4 im Zusammenhang mit pseudo-booleschen Constraints erneut aufgegriffen. Weitere Fortschritte in der Implementierung von SAT-Solvern konnten durch den Einsatz optimierter Entscheidungsheuristiken [MMZ+01] und durch Restart-Strategien [GSK98] erzielt werden.

SAT-Solver, die das oben beschriebenen Klauselllernen umsetzen, werden auch CDCL-Solver (Conflict-Driven Clause Learning) genannt. Um eine Überlastung des Arbeitsspeichers zu verhindern, wird der DPLL-Algorithmus von modernen Solvern iterativ und nicht rekursiv implementiert. Ein entsprechender Pseudocode ist in Algorithmus 1 dargestellt. Solange der Algorithmus keine erfüllende Belegung gefunden hat und nicht

Algorithmus 1 : Iterative Variante von DPLL

```
   Input  : SAT-Instanz P
   Output : Erfüllbarkeitsentscheidung für P
 1 while true do
 2 |  if SolutionFound (P) then
 3 |  |  return true;
 4 |  end
 5 |  Decide (P);
 6 |  while Deduce (P)=CONFLICT do
 7 |  |  if Diagnose (P)=CONFLICT then
 8 |  |  |  return false;
 9 |  |  end
10 |  end
11 end
```

die Unerfüllbarkeit zeigen konnte, wird in Zeile 5 eine weitere Entscheidung getroffen. In Zeile 6 werden mit Unit-Propagation induzierte Belegungen berechnet. Kommt es dabei zu einem Konflikt, so erfolgt in Zeile 7 die Konfliktanalyse. Sollte die gelernte Konfliktklausel – der UIP – auch noch im 0. Entscheidungslevel (keine Entscheidungen) im Konflikt stehen (da entsprechende Belegungen im 0. Level propagiert wurden), so

ist die vorliegende Instanz unerfüllbar (Zeile 8). Im Anschluss an die Konfliktanalyse werden mögliche, induzierte Variablenbelegungen mittels Unit-Propagation berechnet.

2.2 Konfiguration am Beispiel der Produktübersicht PPM

An das Beispiel 1 anschließend wird in diesem Abschnitt aufgezeigt, wie die Daimler-Produktübersicht PPM (Product-, Price- and Master-Data) durch eine aussagenlogische Formel mathematisch beschrieben werden kann.
Eine die KFZ-Baubarkeit beschreibende Formel wurde bereits 1997 von Carsten Sinz aufgestellt [Sin97, KS00]. Als Datengrundlage diente Sinz das Dokumentationssystem DIALOG, welches auf Ebene der Typklassen die technisch möglichen Konfigurationsvarianten beschreibt.
Die Informationen zur technischen Baubarkeit aus DIALOG werden nach PPM transferiert. Darüber hinaus werden in PPM marktspezifische Anforderungen definiert, die die Menge der tatsächlich produzierbaren Varianten weiter einschränken und im Rahmen dieser Arbeit unverzichtbares Wissen darstellen.
Da in PPM zahlreiche Zusammenhänge baureihenübergreifend dokumentiert werden, wird die in diesem Abschnitt definierte Produktübersichtsformel baureihenunabhängig aufgestellt. Sie enthält das komplette Konfigurationswissen von Mercedes-Benz und Smart. Eine Möglichkeit die Formel effektiv auf den gewünschten Umfang zu reduzieren wird zum Ende des Abschnittes vorgestellt.

2.2.1 PPM-Auftrag

Ohne Anspruch auf Vollständigkeit definiert sich ein *PPM-Auftrag* im Wesentlichen durch die folgenden Attribute:

- 7-stelliges Baumuster, d. h. Produktgruppe (Mercedes-Benz, Smart), Baureihe, Typklasse (Aufbau-Art), Motor und Lenkung
- Besteller- und Verbrauchermarkt
- Produktionsdatum
- Aufbauwerk (Produktionsstätte)
- Lackierung
- Ausstattungslinie
- Codeleiste (Liste von Optionen inklusive Sonderausstattungen)

Für „Auftrag“ werden synonym auch die Begriffe „Variante“ und „Konfiguration“ verwendet.

2.2.2 PPM-Validierung und PPM-Baubarkeit

Die *PPM-Validierung* verändert eingehende Aufträge, um deren technische Baubarkeit und um vordefinierte Voraussetzungen des Besteller- und Verbrauchermarktes abzusichern. In sich teilweise wiederholenden Iterationen können Codes aufgrund der aktuellen Konfiguration gelöscht bzw. hinzugesteuert werden. In anderen Fällen kann eine verletzte aussagenlogische Bedingung sofort zum Abbruch der Validierung mit dem Ergebnis „nicht baubar“ führen. Detaillierter wird auf die Validierung in Unterabschnitt 2.2.5 eingegangen. Dort werden die Constraints aufgestellt, die sich aus der Validierung an einen baubaren PPM-Auftrag ableiten.

Definition 3 (PPM-Baubarkeit)
Ein Auftrag wird als *PPM-baubar* bezeichnet, wenn er die PPM-Validierung ohne Änderungen durchläuft. Die Menge aller baubaren PPM-Aufträge wird mit B_{PPM} bezeichnet.

2.2.3 Der grundlegende Ansatz

Das Ziel ist die Beschreibung aller PPM-baubaren Aufträge B_{PPM} durch eine aussagenlogische Formel POF_{PPM} (PPM-Produktübersichtsformel). Dazu sind die PPM-Aufträge als Belegungen aussagenlogischer Variablen zu interpretieren. Die Formel POF_{PPM} soll unter einem Auftrag genau dann zum Wahrheitswert 1 evaluieren, wenn der Auftrag PPM-baubar ist, d. h. $B_{\mathrm{PPM}} = \{\beta \mid POF_{\mathrm{PPM}}(\beta) = 1\}$.

In Unterabschnitt 2.2.4 wird die Menge $V_{\mathrm{PPM}} = \{v_1, \ldots, v_n\}$ der Variablen definiert, die Bestandteil der Produktübersichtsformel POF_{PPM} sind. Für eine vereinfachte Darstellung werden die Variablenbelegungen

$$\beta : V_{\mathrm{PPM}} \to \{0, 1\}$$

in Kapitel 4 auch als Vektoren $x^\beta \in \{0,1\}^n$ interpretiert:

$$\forall i, \quad x_i^\beta = 0 \iff \beta(v_i) = 0.$$

Dementsprechend kann die Formel POF_{PPM} als boolesche Funktion

$$POF_{\mathrm{PPM}} : \{0,1\}^n \longrightarrow \{0,1\}$$

betrachtet werden.

POF_{PPM} ist als boolesche Funktion eindeutig, nicht aber als aussagenlogische Formel. Die konjunktive Normalform (KNF) stellt ein geeignetes Format für die Weiterverar-

beitung in Solvern dar, deren Aufgabe die Berechnung eines Modells aus B_{PPM} ist (vgl. Abschnitt 2.1).

Für eine erste Darstellung von POF_{PPM} wird die Konjunktion aller Bedingungen (Constraints) aufgestellt, die sich aus PPM an einen baubaren Auftrag ableiten (siehe Unterabschnitt 2.2.5). Die Herausforderung liegt in dem Erfassen aller Constraints und ihrem Übersetzen in die Aussagenlogik. Ist dies gelungen, so ist es ausreichend, jeden einzelnen Constraint separat in eine KNF zu transformieren, um eine gewünschte KNF-Darstellung von POF_{PPM} zu erhalten.

2.2.4 PPM-Variablen

In diesem Unterabschnitt werden die aussagenlogischen Variablen V_{PPM} der Produktübersichtsformel POF_{PPM} definiert. Um die Variablen bezüglich ihrer Bedeutung zu differenzieren, werden sie zunächst in verschiedene Kategorien A1, A2, H1 und H2 eingeteilt. Die Variablen der Kategorien A1 und A2 sind einem PPM-Attribut zugeordnet und werden auch Attributsvariablen genannt. Variablen der Kategorie H1 und H2 stellen Hilfsvariablen dar.

(A1) Alle Variablen dieser Kategorie beschreiben das Auftreten von Attributen aus PPM. Das durch die Variable beschriebene Attribut muss zusätzlich Bestandteil der Sprache sein, welche zur Beschreibung der PPM-Aufträge verwendet wird. Beispiele für derartige Attribute bilden die Baumuster oder auch die möglichen Codes der Codeleiste.
Eine Variablenbelegung β mit $\beta(2051421@BM) = 1$ entspricht einer Variante des 7-stelligen Baumusters 205.142.1.

(A2) Die Variablen dieser Kategorie repräsentieren das Auftreten von Attributen aus PPM, welche nicht der Beschreibung von PPM-Aufträgen dienen. Beispielsweise liegt bei $\beta(\text{EUR@LD}) = 1,\ \beta(537@\text{LD}) = 1$ ein Auftrag mit dem Bestellermarkt Großbritannien in Europa vor. Wird die Information Bestellermarkt Europa im Gegensatz zur Information Bestellermarkt Großbritannien nicht verwendet, um den PPM-Auftrag in der Anwendung zu beschreiben, so ist EUR@LD eine Variable der Kategorie (A2) und 537@LD eine Variable der Kategorie (A1).

(H1) Hier sind den Variablen keine Attribute aus PPM zugeordnet. Theoretisch sind diese Variablen nicht notwendig, um eine gewünschte Formel POF_{PPM} aufzustellen. Sie sind dennoch von großer Relevanz, da sie häufig auftretende Teilformeln substituieren und dadurch die Länge der Formel beschränken. Darüber hinaus werden die substituierten Teilformeln bei Evaluierung der Formel POF_{PPM} nur einmalig ausgewertet.

(H2) Die Variablen dieser Kategorie stellen ebenfalls Hilfsvariablen dar, die Teilformeln substituieren. Im Gegensatz zur Kategorie (H1) werden Sie jedoch nicht vom Designer der Formel POF_{PPM} angelegt. Die sogenannten Tseitin-Variablen entstehen erst bei der Transformation von POF_{PPM} in eine KNF und sorgen dafür, dass die Anzahl der Klauseln nicht exponentiell wächst.

Eine Aufteilung der Attributsvariablen zwischen Kategorie (A1) und Kategorie (A2) wird implizit durch die Auswahl der Attribute festgelegt, die in der Anwendung die PPM-Aufträge beschreiben.
Im Falle einer erfüllbaren Variablenbelegung ist der Anwender nicht an den Belegungen der Variablen aus den Kategorien (A2), (H1) und (H2) (bezeichnet mit $h_1, \ldots, h_p$) interessiert. Für den Anwender ist die aussagenlogische Formel

$$\exists h_1 \cdots \exists h_p \; POF_{\text{PPM}} \text{ mit } \exists x F := F_{|x \equiv true} \vee F_{|x \equiv false}$$

ausreichend, die ausschließlich die für ihn interessanten Variablen enthält.
Soll die Formel POF_{PPM} zur Konfiguration PPM-baubarer Aufträge verwendet werden, so wird eine Methode benötigt, welche die Variablen der Kategorie (A1) in die entsprechenden Attribute aus PPM übersetzt. Für eine Belegung β, die POF_{PPM} erfüllt, ist ihre Interpretation als PPM-Auftrag durch die Übersetzung der positiv belegten Variablen aus Kategorie (A1) gegeben.

Es folgt ein kurzer Überblick zur Benennung der verschiedenen PPM-Variablen. Die Namensgebung der Attributs-Variablen orientiert sich stark an der Benennung der Attribute in PPM. Die Belegung einer Attributs-Variable v gibt stets an, ob das entsprechende Attribut aus PPM in einem Auftrag β vorhanden ist ($\beta(v) = 1$) oder nicht vorhanden ist ($\beta(v) = 0$).

Attribut Baumuster
Baumuster-Attribute stehen in PPM unter anderem für die Produktgruppe, die Baureihe, die Typklasse, das 6-stellige Baumuster[3] und das 7-stellige Baumuster. Die Namen der zugehörigen Variablen enden alle mit der Endung „@BM“. Beispiele sind in der Tabelle 3 dargestellt.

Attribut Baubarkeitsgebiet
Die Baubarkeitsgebiete (BGBTs) dienen der Beschreibung des Bestellermarktes. Die Variablennamen der Baubarkeitsgebiete zeichnen sich durch die Endung „@LD“ aus. Beispiele sind die bereits zuvor erwähnten Variablen EUR@LD und 537@LD. Die Variable $ALL@LD$ steht für das Baubarkeitsgebiet Welt und fasst alle Baubarkeitsgebiete zusammen.

[3]Das 6-stellige Baumuster enthält bis auf die Lenkung alle Informationen des 7-stelligen Baumusters

Variable	Attribut	Attributsbeschreibung
SMART@BM	PG SMART	Produktgruppe
MBPKW@BM	PG MB-PKW	Produktgruppe
205@BM	BR 205	Baureihe
W205@BM	TKL W205	Aufbau-Art
205004@BM	BM6 205.004	6-stelliges Baumuster
2050041@BM	BM7 205.004.1	7-stelliges Baumuster

Tabelle 3: Beispielhafte Variablennamen für Baumuster-Attribute

Attribut Produktionsdatum

Das Produktionsdatum beschreibt den Tag der Produktion und hat wesentlichen Einfluss auf fast alle Constraints, die sich aus PPM ableiten. Für jedes in PPM vorkommende Produktionsdatum <TT.MM.JJJJ> wird eine aussagenlogische Variable der Form <JJJJMMTT>@DT angelegt.

Sonstige Attribute

Die Endungen „LD“, „BM“ und „DT“ entsprechen in PPM sogenannten Prüfelementen. Beispielsweise evaluiert in PPM die Prüfung „BM = 205 212“ genau dann zum Wahrheitswert 1, falls der Auftrag entweder der Baureihe 205 oder der Baureihe 212 angehört. Jedes Attribut aus PPM ist einem Prüfelement zugeordnet.

Um die Variablen eindeutig interpretieren zu können, enthalten die Variablennamen der restlichen Attribute aus PPM neben dem Prüfelement auch die Produktgruppe und die Marktbeschränkung (VBET-Nummer). Die Variablennamen haben den Aufbau

<Name>@<Produktgruppe><Markt>@<Prüfelement>.

Beispiele sind in der Tabelle 4 dargestellt.

Variable	PG	Markt	Attributsbeschreibung
805@P1@CO	MB-PKW	Zentrale	technische Änderungen
U26@P1@CO	MB-PKW	Zentrale	AMG Fußmatten
NL8N@P81@CO	MB-PKW	Niederlande	NST-Code
532L@F1@CO	SMART	Zentrale	Land Frankreich
005N@F88@CO	SMART	Tschechien	Brabus

Tabelle 4: Beispielhafte Variablennamen für sonstige PPM-Attribute

Nicht zu jedem PPM-Attribut sind die Informationen Produktgruppe und Markt für eine eindeutige Zuordnung notwendig. Der hier gewählte Aufbau der Variablennamen

setzt jedoch weniger Expertenwissen und weniger Fallunterscheidungen in der Implementierung voraus.

Hilfsvariablen
Die Variablen der Kategorie (H1) werden durchnummeriert und erhalten die Endung „AUX“. Sie haben die Form <NUMMER>AUX.
Die durch eine Tseitin-Transformation erzeugten Variablen der Kategorie (H2) werden ebenso durchnummeriert. Ihnen ist ein Präfix vorangestellt: TS___<NUMMER>.

2.2.5 PPM-Constraints

Der Ansatz, die Produktübersichtsformel POF_{PPM} darzustellen, wurde bereits beschrieben. Aussagenlogische Constraints C_i werden so zusammengetragen, dass

$$POF_{\text{PPM}} = \bigwedge_{i=1}^{l} C_i \tag{2}$$

gilt. Diese Gleichung kann in zwei Forderungen übersetzt werden:

1. Jeder Constraint C_i ist von jeder PPM-baubaren Variante zu erfüllen.

2. Jede nicht PPM-baubare Belegung β erfüllt mindestens einen Constraint C_i nicht.

Falls alle aus PPM abgeleiteten Constraints korrekt erfasst und richtig in die Aussagenlogik übersetzt werden, ist die erste Forderung erfüllt. Die zweite Forderung wird erfüllt, wenn alle nötigen Constraints erfasst werden.
Im Folgenden werden die verschiedenen PPM-Constraints vorgestellt.

Regelcluster
Regelcluster werden in PPM eingesetzt, um an „Auftragsrümpfen“ mittels sogenannter Aktionen nötige Änderungen vorzunehmen. Die Auftragsrümpfe werden durch das Löschen bzw. Hinzusteuern von PPM-Attributen zu baubaren Aufträgen verändert.
Ein Regelcluster wird nur dann ausgeführt, wenn der Produktionstag DT, die Produktgruppe PG, der Markt VN und eine dem Regelcluster zugeordnete aussagenlogische Regel ARG – das Argument – zum Wahrheitswert *true* evaluieren. Eine vorliegende Konfiguration muss die Formel

$$\phi = d_1 \leq \text{DT} \leq d_2 \wedge \text{PG} \wedge \text{VN} \wedge \text{ARG}$$

erfüllen, damit die Aktionen des Regelclusters zum Tragen kommen. Dabei definieren die Daten d_1 und d_2 ein Intervall, in dem der Produktionstag DT liegen muss. Sind $\text{DT}_1, \ldots, \text{DT}_n$ die möglichen Produktionstage zwischen d_1 und d_2, so kann der Teilconstraint $d_1 \leq \text{DT} \leq d_2$ als Klausel $\text{DT}_1 \vee \cdots \vee \text{DT}_n$ formuliert werden.

Jeder Aktion i eines Regelcluster ist zusätzlich eine aussagenlogische Bedingung α_i zugeordnet. Nur wenn diese unter dem aktuellen Auftragsrumpf zum Wahrheitswert 1 evaluiert, wird die i-te Aktion ausgeführt.
Ein Regelcluster besitzt nur löschende oder nur einsteuernde Aktionen.

Einsteuernde Regelcluster
Es sei die i-te Aktion des Regelclusters ein Einsteuerer für das Attribut v_i. Die aktuelle Konfiguration eines Auftrages wird durch die i-te Aktion genau dann verändert, falls sie $(\phi \wedge \alpha_i) \wedge \neg v_i$ erfüllt. Als notwendige Voraussetzung an einen PPM-baubaren Auftrag ergibt sich somit der Constraint

$$\neg\Big[(\phi \wedge \alpha_i) \wedge \neg v_i\Big] \equiv \neg(\phi \wedge \alpha_i) \vee v_i \equiv (\phi \wedge \alpha_i) \rightarrow v_i. \tag{3}$$

Je nachdem wie viele Aktionen ein Regelcluster besitzt, kann es lohnenswert sein die sich wiederholende Teilformel ϕ mit einer Hilfsvariablen der Kategorie (H1) zu substituieren (siehe Unterabschnitt 2.2.4).

Löschende Regelcluster
Unter identischen Bezeichnungen sei die i-te Aktion nun löschend. Die Aktion verändert genau die Konfigurationen, die $(\phi \wedge \alpha_i) \wedge v_i$ erfüllen. Der resultierende Constraint ist somit durch

$$\neg\Big[(\phi \wedge \alpha_i) \wedge v_i\Big] \equiv \neg(\phi \wedge \alpha_i) \vee \neg v_i \equiv (\phi \wedge \alpha_i) \rightarrow \neg v_i \tag{4}$$

gegeben.

Nicht wählbare Codes
Bestimmte Codes werden vor der eigentlichen Validierung immer aus den Aufträgen gelöscht und können nachfolgend nur über einsteuernde Regelcluster in die Aufträge gelangen. Ist ein solcher Code v gegeben und existiert keine einsteuernde Aktion für v, so wird der Constraint $\neg v$ angelegt. Andernfalls iteriere σ_j über alle Zusteuerbedingungen für v, welche der Form $(\phi \wedge \alpha_i)$ aus (3) für einsteuernde Regelcluster genügen. Tritt der Code v in einem Auftrag auf, so muss der Auftrag mindestens eine Regel σ_j erfüllen. Das ist äquivalent dazu, dass jeder PPM-baubare Auftrag den Constraint

$$v \rightarrow \bigvee_i \sigma_i \equiv \neg v \vee \bigvee_i \sigma_i \tag{5}$$

erfüllt.

Hierarchien

Baumuster, Baubarkeitsgebiete und Märkte sind in hierarchischen Baumstrukturen angeordnet. Zum Beispiel sind der Produktgruppe MBPKW@BM unter anderem die Baureihen 212@BM und 205@BM als direkte Kindknoten zugeordnet. Direkte Kindknoten von 212@BM sind wiederum die Typklassen W212@BM und S212@BM. Für jede Beziehung zwischen einem Vaterknoten v und seinen direkten Kindknoten $\mathrm{k}_1, \ldots, \mathrm{k}_n$ werden die Constraints

$$\mathrm{k}_i \rightarrow v \quad \equiv \quad \neg \mathrm{k}_i \vee v, \qquad \forall i \tag{6}$$

$$v \rightarrow \Big(\bigvee_i \mathrm{k}_i \Big) \quad \equiv \quad \neg v \vee \Big(\bigvee_i \mathrm{k}_i \Big) \tag{7}$$

angelegt, die die gesamten Baumstrukturen abbilden.

Gültigkeiten

Für die Baubarkeit bestimmter PPM-Attribute müssen entsprechend der vorliegenden Konfiguration passende zeitliche Gültigkeiten hinterlegt sein. Zunächst wird mit den Code-Gültigkeiten der komplexere Fall besprochen. Im Anschluss wird auf die Baumuster-Gültigkeiten eingegangen, deren Constraints sich leicht aus denen der Code-Gültigkeiten ableiten.

Code-Gültigkeiten

Bestimmte Codes sind nur mit einem *passenden Gültigkeitseintrag* baubar. Ein *Code-Gültigkeitseintrag* setzt sich zusammen aus einem 6-stelligen bzw. 7-stelligen Baumuster (abhängig vom Code), einem Baubarkeitsgebiet und einem Zeitintervall $[d_1, d_2]$. Vom Baumuster und vom Baubarkeitsgebiet hängt unter anderem ab, ob der Gültigkeitseintrag unter der vorliegenden Konfiguration der *passende* Eintrag ist, d. h. ob das Zeitintervall des Eintrags die Gültigkeit des Codes bestimmt. Die Gültigkeit eines Codes c wird wie folgt überprüft (vgl. parallel Abbildung 4):

1. Jeder Gültigkeitseintrag ist einem *Gültigkeitskontext* zugeordnet. Ein Gültigkeitskontext definiert sich durch einen Prioritätswert und eine aussagenlogische Regel. Es wird die Reihenfolge r_k, r_{k-1},...,r_0 der Kontextregeln entsprechend ihrer absteigenden Priorität betrachtet. In einem ersten Schritt wird in der Validierung der zutreffende Kontext ermittelt. Dazu werden die Kontextregeln entsprechend ihrer Priorität unter der aktuellen Konfiguration ausgewertet. Der erste Kontext, bei dem die Regel zum Wahrheitswert 1 evaluiert, ist der zutreffende Kontext. Der i-te Kontext ist also genau dann zutreffend, wenn die aktuelle Konfiguration die Formel

$$\gamma_i = r_i \wedge \bigwedge_{j=i+1}^{k} \neg r_j \tag{8}$$

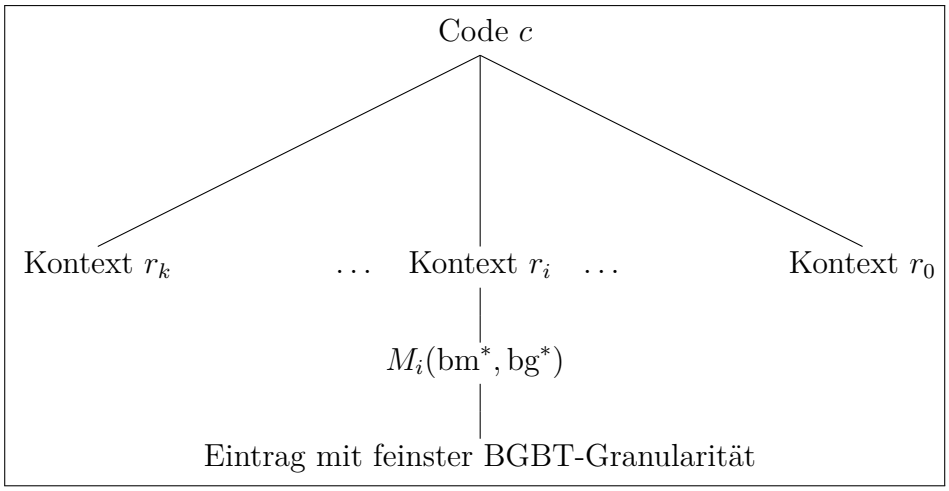

Abbildung 4: Auswahl des passenden Code-Gültigkeitseintrages in PPM

erfüllt. Trifft kein Kontext zu, so ist der Code c in der vorliegenden Konfiguration nicht gültig und somit nicht baubar.

2. Der passende Kontext sei der i-te Kontext. In der Validierung wird nun unter den Gültigkeitseinträgen, die dem i-ten Kontext zugeordnet sind, der passende gesucht:

   - Für ein Baumuster bm sei $M_i(\mathrm{bm})$ die Menge der Gültigkeitseinträge des i-ten Kontextes, die dem Baumuster bm zugeordnet sind. Ist $M_i(\mathrm{bm}^*)$ für das vorliegende Baumuster bm^* leer, so ist der Code c in der vorliegenden Konfiguration nicht gültig. Andernfalls wird in $M_i(\mathrm{bm}^*)$ der passende Gültigkeitseintrag gesucht.

   - Neben dem Baumuster muss auch das Baubarkeitsgebiet zum Gültigkeitseintrag passen. Für ein Baubarkeitsgebiet feinster Granularität bg seien mit $M_i(\mathrm{bg})$ die Gültigkeitseinträge des i-ten Kontextes gegeben, denen ein Baubarkeitsgebiet zugeordnet ist, das auf dem Hierarchie-Pfad zwischen bg und ALL@LD liegt. Ist bg^* das Baubarkeitsgebiet geringster Granularität der vorliegenden Konfiguration, so wird in der Validierung nach dem passenden Gültigkeitseintrag in

     $$M_i(\mathrm{bm}^*, \mathrm{bg}^*) := M_i(\mathrm{bm}^*) \cap M_i(\mathrm{bg}^*)$$

     gesucht. Ist $M_i(\mathrm{bm}^*, \mathrm{bg}^*)$ leer, so ist der Code c unter der vorliegenden Konfiguration nicht gültig. Ansonsten ist der passende Kontext durch den Gültigkeitseintrag in $M_i(\mathrm{bm}^*, \mathrm{bg}^*)$ gegeben, dessen Baubarkeitsgebiet auf dem Hierarchie-Pfad zwischen bg^* und ALL@LD die geringste Granularität besitzt.

Mit dem Wissen darüber, wie in der Validierung die Code-Gültigkeiten ausgewertet werden, ist es möglich die für den Code c resultierenden Constraints aufzustellen.

Existiert kein Kontext zum Code c, so muss jeder PPM-baubare Auftrag dem Constraint $\neg c$ genügen.
Für den Fall, dass Kontexte existieren, wird zunächst der Constraint

$$\Big(\bigwedge_{j=1}^{k} \neg r_j\Big) \rightarrow \neg c \equiv \Big(\bigvee_{j=1}^{k} r_j\Big) \vee \neg c \tag{9}$$

aufgestellt, der den Code c nur dann erlaubt, wenn mindestens ein Kontext zutrifft.

Im Folgenden werden die Constraints formuliert, die sich aus dem i-ten Kontext ergeben. Eine Voraussetzung an jeden dieser Constraints ist, dass der i-te Kontext der zutreffende Kontext ist, dass also γ_i aus (8) gilt.

1. Zunächst wird der Constraint beschrieben, der den Code c verbietet falls die Menge $M_i(\mathrm{bm}^*, \mathrm{bg}^*)$ leer ist. Einem Gültigkeitseintrag ist ein Paar $(\mathrm{bm}, \mathrm{bg})$, bestehend aus Baumuster und Baubarkeitsgebiet, zugeordnet. Iteriert $(\mathrm{bm}, \mathrm{bg})$ über alle Paare des i-ten Kontextes, so ergibt sich für den Constraint folgende Darstellung

$$\gamma_i \rightarrow \Big[\neg\Big(\bigvee_{(\mathrm{bm},\mathrm{bg})} (\mathrm{bm} \wedge \mathrm{bg})\Big) \rightarrow \neg c\Big] \equiv \neg\gamma_i \vee \bigvee_{(\mathrm{bm},\mathrm{bg})} (\mathrm{bm} \wedge \mathrm{bg}) \vee \neg c. \tag{10}$$

2. Der verbleibende Fall für die Ungültigkeit des Codes c ist darin begründet, dass das Intervall $[d_1, d_2]$ des passenden Gültigkeitseintrages nicht den Produktionstag DT der vorliegenden Konfiguration enthält. Hierzu wird ein Gültigkeitseintrag mit Baumuster bm und Baubarkeitsgebiet bg betrachtet. Mit $C_i(\mathrm{bm}, \mathrm{bg})$ sei die Menge aller Baubarkeitsgebiete $\mathrm{bg}_{\mathrm{child}}$ beschrieben, für die zum i-ten Kontext ein Gültigkeitseintrag $(\mathrm{bm}, \mathrm{bg}_{\mathrm{child}})$ existiert, so dass bg auf dem Hierarchie-Pfad zwischen $\mathrm{bg}_{\mathrm{child}}$ und ALL@LD liegt. Es ergibt sich der Constraint

$$\gamma_i \rightarrow \Big[bm \rightarrow \Big(\Big(bg \wedge \bigwedge_{\mathrm{bg}_{\mathrm{child}} \in C_i(\mathrm{bm},\mathrm{bg})} \neg bg_{\mathrm{child}}\Big) \rightarrow \big(\neg(d_1 \leq \mathrm{DT} \leq d_2) \rightarrow \neg c\big)\Big)\Big],$$

welcher äquivalent zu

$$\bigvee_{\mathrm{bg}_{\mathrm{child}} \in C_i(\mathrm{bm},\mathrm{bg})} bg_{\mathrm{child}} \vee \neg\gamma_i \vee \neg bm \vee \neg bg \vee \neg c \vee d_1 \leq \mathrm{DT} \leq d_2 \tag{11}$$

ist. Der Teilconstraint $d_1 \leq \mathrm{DT} \leq d_2$ kann genau wie im Fall der Regelcluster behandelt werden.

Mit den Constraints der Form (9) bis (11) wurden all jene erfasst, die sich aus den Code-Gültigkeiten ergeben. Es bietet sich an, die sich gegebenenfalls wiederholenden Teilformeln r_j und γ_i in Abhängigkeit von ihrer Größe und ihrer Häufigkeit in der Gesamtformel POF_{PPM} mit Hilfsvariablen der Kategorie (H1) zu substituieren.

Baumuster-Gültigkeiten

Ein 7-stelliges Baumuster bm^* ist nur dann gültig, wenn ein entsprechender Gültigkeitseintrag existiert. Baumuster-Gültigkeiten unterscheiden sich von Code-Gültigkeiten in nur einem wesentlichen Punkt: Ihren Gültigkeitseinträgen sind keine 6-stelligen bzw. 7-stelligen Baumuster zugeordnet. Dies berücksichtigend leiten sich ihre Constraints direkt aus den Constraints der Code-Gültigkeiten ab.

Existieren zu dem Baumuster bm^* keine Gültigkeitskontexte, so muss jeder PPM-Auftrag den Constraint $\neg\text{bm}^*$ erfüllen. Existieren Kontexte, so muss jeder PPM-Auftrag die folgenden Bedingungen erfüllen.

Ein Kontext triff zu oder das Baumuster bm^* ist nicht gültig (vgl. (9)):

$$\Big(\bigvee_{j=1}^{k} r_j\Big) \vee \neg\text{bm}^*. \tag{12}$$

Ein Gültigkeitseintrag trifft zu oder das Baumuster ist nicht gültig (vgl. (10)):

$$\neg\gamma_i \vee \bigvee_{\text{bg}} \text{bg} \vee \neg\text{bm}^*. \tag{13}$$

Ist ein Gültigkeitseintrag passend, so muss der Produktionstag DT im zugehörigen Zeitintervall $[d_1, d_2]$ liegen oder das Baumuster bm^* ist nicht gültig (vgl. (11)):

$$\bigvee_{\text{bg}_{\text{child}} \in C_i(\text{bg})} bg_{\text{child}} \vee \neg\gamma_i \vee \neg bg \vee \neg\text{bm}^* \vee d_1 \leq \text{DT} \leq d_2. \tag{14}$$

In Constraint (13) iteriert bg über alle Baubarkeitsgebiete, die in einem Gültigkeitseintrag des i-ten Kontextes auftreten. Die Menge $C_i(\text{bg})$ aus Constraint (14) beschreibt die Menge aller Baubarkeitsgebiete bg_{child}, für die zum i-ten Kontext ein Gültigkeitseintrag für bg_{child} existiert, so dass bg auf dem Hierarchie-Pfad zwischen bg_{child} und ALL@LD liegt.

Primärgruppen

Eine *Primärgruppe* stellt eine Gruppe von Attributen dar, unter denen genau eines in einem baubaren PPM-Auftrag vorkommt. Beispiele hierfür bilden die verschiedenen Ausstattungspakete, die 7-stelligen Baumuster oder auch die Produktionstage. Pri-

märgruppen $\{v_1, \ldots, v_n\}$ werden durch sogenannte *exactly-1-Constraints* repräsentiert (vgl. Unterabschnitt 3.1.2). Deren naive Übersetzung in eine KNF benötigt eine in n quadratische Anzahl von Klauseln:

$$v_1 \vee \cdots \vee v_n \tag{15}$$

$$\neg v_i \vee \neg v_j \qquad \forall i \neq j \tag{16}$$

Die in (16) dargestellten Klauseln repräsentieren den anteiligen *atmost-1-Constraint*, welcher die quadratische Ordnung begründet. Insbesondere im Fall der 7-stelligen Baumuster oder der Produktionstage stellen *atmost-1-Constraints* bei naiver Codierung einen großen Teil der Produktübersichtsformel dar. Die zugehörigen Primärgruppen besitzen ca. 3.000 Elemente.
Eine Möglichkeit *exactly-1-Constraints* mit $\mathcal{O}(n)$ Klauseln und $\mathcal{O}(n)$ Hilfsvariablen zu codieren, wird von Kieber et al. in [KK07] vorgestellt. Die Autoren gruppieren die Variablen der *exactly-1-Constraints* und weisen jeder Gruppe eine Vatervariable (Hilfsvariable) zu. Die Vatervariablen selbst werden ebenso in Gruppen eingeteilt, welche in einem nächst höheren Level eigene Vatervariablen erhalten. Auf diese Weise wird fortgefahren bis in einem höchsten Level nur noch eine Vatervariable definiert ist. Eine Vatervariable evaluiert genau dann zum Wahrheitswert 1, falls genau eine Kindvariable zu 1 evaluiert. Ein *exactly-1-Constraint* wird somit genau dann erfüllt, wenn die oberste Vatervariable zu 1 evaluiert. Die *exactly-1-Constraints* der einzelnen Gruppen werden naiv codiert. Die Anzahl der entstehenden Klauseln ist dann besonders niedrig, wenn Gruppen von jeweils 3 Variablen gebildet werden.
Dieser Ansatz wurde für die Codierung der Primärgruppen aus PPM umgesetzt. Weitere alternative Codierungen linearer Ordnung werden in [Sin05] verglichen.

Wird kein Wert auf eine Produktübersichtsformel gelegt, welche über alle Produktionstage oder über alle Baureihen gültig ist, so können die problematischen Primärgruppen auch erst bei Spezialisierung der Produktübersichtsformel (siehe Abschnitt 2.2.6) behandelt werden. Soll zum Beispiel eine Produktübersichtsformel für zwei Monate aufgestellt werden, so wäre bei der Spezialisierung ein *exactly-1-Constraint* für alle Produktionstage innerhalb der zwei Monate anzulegen; alle restlichen Produktionstage wären zu negieren.

2.2.6 Numerische Ergebnisse zur Komplexität der Formel POF_{PPM}

Im Rahmen dieser Arbeit wurde eine KNF einer „maximal allgemeingültigen“ Produktübersichtsformel POF_{PPM} generiert, die zum vorliegenden Datenstand die baubaren Varianten aller 59 Baureihen (Historie, Gegenwart und Zukunft) für alle Märkte und alle Produktionstage codiert.
Die Codierung der Primärgruppen erfolgte mit dem in [KK07] vorgestellten Ansatz,

der linear viele Klauseln benötigt. So konnten gegenüber dem naiven Ansatz in etwa 9.000.000 Klauseln eingespart werden. Die sonstigen in Abschnitt 2.2.5 dargestellten Constraints wurden durch einfaches Ausmultiplizieren mit dem Distributivgesetz in eine KNF umgewandelt. Entstehende Tautologien, d. h. Klauseln, in denen eine Variable positiv wie negativ vorkommt, wurden dabei gestrichen. In ca. 300 Fällen wurden beim Ausmultiplizieren mehr als 10.000 Klauseln generiert, weshalb unterstützend die Tseitin-Transformation eingesetzt wurde (vgl. Unterabschnitt 2.1.1). Insgesamt wurden mit diesem Vorgehen etwa 17 Millionen Klauseln generiert. Gut 98% der insgesamt $1,8$ Millionen Variablen sind selbst definierte Hilfsvariablen der in Abschnitt 2.2.4 beschriebenen Kategorie (H1). Unter Verwendung der Variablennamen aus Abschnitt 2.2.4 benötigt die Formel als Textdatei circa 550 MB Festplattenspeicher.
Den meisten Anwendungen bedarf es nicht einer derart allgemeingültigen Produktübersichtsformel. So bezieht sich beispielsweise eine auf Konsistenz zu überprüfende Sonderausstattungs-Prognose auf ein 7-stelliges Baumuster und ein Land als Markt. Zu dem ist es üblich einen Produktionstag festzulegen – zu mindestens solange für den Zielplanungshorizont eine entsprechende Differenzierung der tagesabhängigen Konfigurationsmöglichkeiten noch nicht vorliegt.

Zur Spezialisierung der berechneten Formel POF_{PPM} wurde ein 64-Bit-Kompilat der **C++**-Implementierung des SAT-Solvers MiniSAT verwendet [SE03]. Die einmalige Ladezeit der allumfassenden Ausgangsformel betrug weniger als zwei Minuten und benötigte knapp 3 GB Arbeitsspeicher. Daraufhin konnten für eine Reihe von Settings die spezialisierten Formeln in jeweils weniger als einer Sekunde berechnet werden. Hierzu wurden dem SAT-Solver die Variablenbelegungen eines Settings vorgegeben, um anschließend weitere, induzierte Variablenbelegungen mit Unit-Propagation (UP) zu berechnen. Durch Löschen der Literale $\beta(l) = 0$ und durch Löschen der Klauseln, die ein Literal $\beta(l) = 1$ enthalten, wurde die KNF spezialisiert. In Tabelle 5 sind die berechneten Settings zu finden und die jeweilige Anzahl noch nicht erfüllter Klauseln und noch nicht belegter Variablen. Alle dargestellten Settings wurden zusätzlich unter einem festen Produktionsdatum spezialisiert - jedoch so, dass zu jedem Setting mindestens eine baubare Variante existiert.
Hervorzuheben ist, dass sich die Anzahl der nicht belegten Variablen für die in der Code-Planung relevanten Granularität „7-stelliges Baumuster/Markt“ (BM7/Markt) auf etwa 500 bis 1.500 reduziert.

Für das NP-vollständige SAT-Problem existieren erfüllbare Instanzen mit nur 1.000 Variablen, für die moderne SAT-Solver wie MiniSAT weit mehr als eine Stunde zum Auffinden eines Modelles benötigen [SC15]. Um die Komplexität der vorliegenden Produktübersichtsformel einschätzen zu können, wurden randomisierte Testrechnungen durchgeführt. Die Produktübersichtsformel wurde zunächst auf ein aktuelles Produk-

Granularität	**Spezialisierung**	**#Variablen**	**#Klauseln**
PG / Welt	MBPKW@BM	1178674	10881217
PG / Welt	SMART@BM	41585	420217
BR / Welt	205@BM	140142	786685
BR / Welt	212@BM	131152	1222193
BR / Welt	222@BM	61920	520952
BR / Welt	246@BM	29156	413523
TKL / Welt	V205@BM	15621	301239
TKL / Welt	S205@BM	40309	409463
TKL / Welt	W205@BM	59500	482783
TKL / Welt	V212@BM	8027	124701
TKL / Welt	W212@BM	76132	976009
TKL / Welt	S212@BM	59659	521142
TKL / Welt	W222@BM	20245	339623
TKL / Welt	S176@BM	30770	494271
BM7 / Welt	2050042@BM	4197	126654
BM7 / Welt	2050041@BM	4400	132595
BM7 / Welt	2050401@BM	4267	148265
BM7 / Welt	2050402@BM	3199	131567
BM7 / Welt	2120021@BM	5349	160678
BM7 / Welt	2122051@BM	4129	129658
BM7 / Welt	2120362@BM	5085	159130
BM7 / Welt	2221621@BM	1459	12825
BM7 / Welt	2221821@BM	5340	149383
BM7 / Welt	1760422@BM	4240	142235
BM7 / Welt	1760522@BM	4186	142277
BM7 / Welt	2462121@BM	4279	131304
BM7 / Welt	2462471@BM	4220	127894
BM7 / Markt	2050042@BM+537@LD	677	3052
BM7 / Markt	2050041@BM+543@LD	753	3174
BM7 / Markt	2050041@BM+374@LD	746	3196
BM7 / Markt	2050402@BM+839@LD	603	2419
BM7 / Markt	2050401@BM+581@LD	710	3315
BM7 / Markt	2120021@BM+374@LD	1394	5262
BM7 / Markt	2122051@BM+374@LD	812	2971
BM7 / Markt	2120362@BM+537@LD	1308	4847
BM7 / Markt	2221821@BM+705@LD	574	1664
BM7 / Markt	1760422@BM+839@LD	521	1214
BM7 / Markt	1760522@BM+839@LD	494	850
BM7 / Markt	2462121@BM+531@LD	652	1713
BM7 / Markt	2462471@BM+571@LD	673	1595

Tabelle 5: Charakteristika der mit UP spezialisierten Produktübersichten

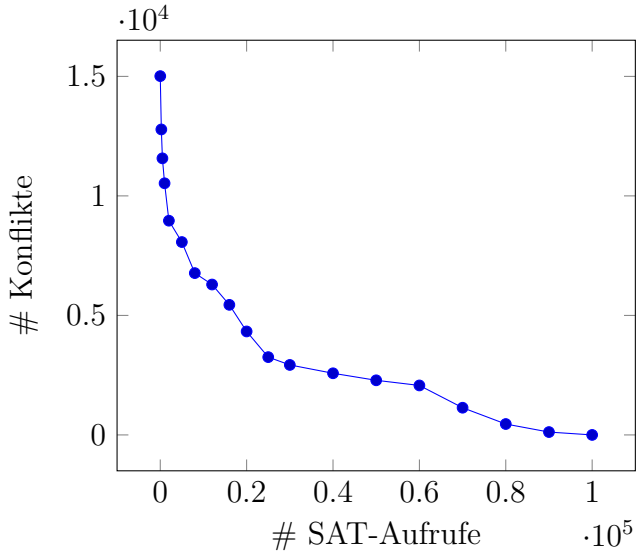

Abbildung 5: Abnahme der Konfliktdichte bei permanentem Klausel-Lernen

tionsdatum spezialisiert, für das schon alle Informationen in PPM eingepflegt wurden. Die resultierende KNF umfasste noch ca. $1,2$ Millionen Variablen und ca. $11,6$ Millionen Klauseln. Diese Formel wurde in MiniSAT einmalig eingeladen, um in der Folge 100.000 Modelle zu berechnen. MiniSAT wurde dazu wie folgt konfiguriert:

- Die Auswahl der nächsten Entscheidungsvariable erfolgte immer zufällig und mit Gleichverteilung.
- Die zufällig ausgewählte Variable wurde immer auf den Wahrheitswert 0 gesetzt.
- Gelernte Klauseln wurden nicht gelöscht und für die Berechnungen der nachfolgenden Modelle beibehalten.

Insgesamt ist der so eingestellte SAT-Solver bei der Berechnung der 100.000 Modelle auf nur ca. 15.000 Konflikte gestoßen. Die nichtproportionale Abnahme der verbleibenden Konflikte, d. h.

$$f(i) := \sum_{j=i+1}^{100.000} s_j, \quad s_j := \#\{\text{Konflikte bei Berechnung des } j\text{-ten Modells}\},$$

ist in Abbildung 5 dargestellt. So traten die ersten 5.000 Konflikte während der ersten 1.000 Modelle auf. Innerhalb der letzten 50.000 Modelle traten nur noch knapp 2.300 Konflikte auf. Nach dem randomisierten Anlernen der ersten knapp 13.000 Klauseln kamen auf einen Konflikt etwa 20 konfliktfreie Modellberechnungen, wobei nur für 525 Modelle der letzten 50.000 Modelle überhaupt Konflikte auftraten. Die maximale Anzahl von Konflikten, die dabei während einer Modellberechnung auftrat lag bei 620 und ist als Peak zu werten. Im Falle einer konfliktfreien Belegung der knapp $11,6$ Millionen Variablen benötigte der Rechner ca. eine halbe Sekunde.

Segment	Anz. Konfl. in %	Ant. prop. VarB. in %
2050041@BM+374@LD	1.83	46.99
2120021@BM+374@LD	1.68	68.16
2122051@BM+374@LD	1.02	46.14
2462121@BM+531@LD	0.70	39.99
2050042@BM+537@LD	1.84	46.71
2120362@BM+537@LD	0.40	72.39
2050041@BM+543@LD	1.10	50.16
2462471@BM+571@LD	0.45	43.47
2050401@BM+581@LD	1.41	46.63
2221821@BM+705@LD	0.32	36.10
1760422@BM+839@LD	0.26	31.08
1760522@BM+839@LD	0.09	27.70
2050402@BM+839@LD	1.05	43.49

Tabelle 6: Statistiken zur Modellberechnung (Mittelw. zu $N = 100.000$)

Des Weiteren wurde das Verhältnis zwischen entschiedenen Variablenbelegungen und propagierten Variablenbelegungen untersucht. Bereits die Ergebnisse zur Spezialisierung aus Tabelle 5 lassen einen hohen Anteil der propagierten Variablenbelegungen erkennen. Tatsächlich lag der Anteil bei durchschnittlich $99,94\%$ mit einer mittleren Abweichung von $0,02\%$ für die 100.000 berechneten Modelle.

In analogen Testrechnungen wurden jeweils 100.000 Modelle zu den spezialisierten KNFs der Granularität BM7/Markt aus Tabelle 5 berechnet. Dabei wurden die oben beschriebenen Solver-Einstellungen beibehalten. In Tabelle 6 ist für jedes Segment die durchschnittliche Anzahl von Konflikten pro Aufruf angegeben. Aus den Daten ergibt sich, dass der Solver für jedes Segment weniger als 2.000 Klauseln angelernt hat. Aufgrund der wenigen Aussagenvariablen und der geringen Klauselanzahl auf Segmentebene konnten – nach dem randomisierten Anlernen der ersten Klauseln – nahezu alle Modelle in weniger als einer Millisekunde berechnet werden.
Ebenso wurde der durchschnittliche Anteil der propagierten Variablenbelegungen auf der Granularität BM7/Markt untersucht (siehe Tabelle 6). Dabei wurden ausschließlich solche Variablen berücksichtigt, die nach der Spezialisierung noch unbelegt waren. Die Auswertungen zeigen, dass der durchschnittliche Anteil nach der Festlegung auf ein 7-stelliges Baumuster und auf einen Markt signifikant niedriger ist als im Fall der allumfassenden Produktübersichtsformel.
Der Anteil der entschiedenen Variablenbelegungen ist demnach relativ hoch und unter Berücksichtigung der sehr geringen Konfliktdichte lässt dies den Schluss zu, dass die Entscheidung $x = 0$ nur sehr selten zu Konflikten führt und im Sinne einer effizienten Modellberechnung „richtig“ ist. Es ist jedoch zu klären, ob die Belegung $x = 1$ für

die nicht propagierten Variablen genauso gut möglich wäre. In diesem Zusammenhang wird im nächsten Abschnitt auf Primimplikanten eingegangen.

Eine mögliche Erklärung für die geringe Konfliktdichte bei der Standardentscheidung $x = 0$ ist durch den hohen Anteil von Hornklauseln in der Produktübersichtsformel POF_{PPM} gegeben. Dieser wurde mit $96,1\%$ ausgezählt. *Hornklauseln* sind Klauseln in denen höchstens eine Variable positiv, d. h. nicht negiert vorkommt. *Hornformeln* sind konjunktive Normalformen, die ausschließlich aus Hornklauseln bestehen. Wird im Entscheidungsbaum immer mit $x = 0$ verzweigt, so berechnet der DPLL-Algorithmus ein Modell einer erfüllbaren Hornformel immer konfliktfrei. Die Strategie stets $x = 0$ zu entscheiden simuliert den sogenannten Markierungsalgorithmus für Hornformeln [DG84], wobei das Propagieren dem Markieren entspricht.

Als Ergebnis der numerischen Untersuchungen wird zusammenfassend festgehalten, dass das Anlernen von verhältnismäßig wenigen Klauseln ausreicht, um die Konfliktdichte bei der Modellberechnung auf ein derart geringes Niveau zu reduzieren, dass bei zufälliger Auswahl der Entscheidungsvariablen nur etwa jede hundertste Modellberechnung konfliktbehaftet ist. Auf eine Konfliktanalyse, wie sie in modernen SAT-Solvern implementiert ist, kann dennoch nicht verzichten werden, da vereinzelte Bereiche des Suchraumes zu einer Vielzahl von Konflikten führen.

2.3 Implikanten

2.3.1 Definitionen

Ein Modell einer aussagenlogischen Formel ϕ ist eine Belegung der Variablen $Var(\phi)$, so dass ϕ zum Wahrheitswert 1 evaluiert. Modelle weisen somit die Erfüllbarkeit von Formeln nach. Bezüglich dieser Eigenschaft verallgemeinert der Begriff der Implikante den Begriff des Modells.

Definition 4 (Implikante)
Für eine Menge M von Variablen ist die Variablenbelegung $\alpha : M \to \{0,1\}$ eine Implikante der Formel ϕ, falls jede Variablenbelegung β mit

$$\beta : N \to \{0,1\},\ M, Var(\phi) \subseteq N,\ \beta_{|M} = \alpha$$

ein Modell von ϕ ist. Die Menge M wird als Basis der Implikante α bezeichnet.

Definition 5 (Primimplikante)
Eine Implikante einer Formel ϕ mit Basis M ist eine Primimplikante, falls keine Implikante von ϕ mit Basis $M' \subset M$ existiert.

Eine Implikante mit einer Basis M und $k = \left|Var(\phi) \setminus M\right|$ repräsentiert 2^k Modelle der Formel ϕ. Im Gegensatz zu Modellen enthalten Implikanten somit Informationen über Freiheitsgrade der Formel ϕ. Derartige Freiheitsgrade der Produktübersichtsformel werden im 4. Kapitel ausgenutzt.

2.3.2 Ein Modell-basierter Algorithmus

Déharbe et al. zeigen, wie Primimplikanten auf Basis eines Modells berechnet werden können [DFLBM13]. Des Weiteren erläutern sie, wie Informationen, die bei der Modellberechnung mit DPLL abfallen, für eine effizientere Berechnung der Primimplikanten genutzt werden können. Die grundlegende Idee besteht darin, Variablenbelegungen aus dem berechneten Modell sukzessive so zu streichen, dass stets eine Implikante vorliegt. Kann keine Variablenbelegung mehr gestrichen werden, so dass die Eigenschaft der Implikante erhalten bleibt, liegt eine Primimplikante vor.
In ihrem Paper geben sie eine in $\mathcal{O}(1)$ zu überprüfende, hinreichende Bedingung dafür an, dass eine Variablenbelegung nicht gestrichen werden darf.

Lemma 1
Es sei ϕ eine Formel und α eine Variablenbelegung. Weiter sei die Formel ω eine Konsequenz aus ϕ, d. h. $\phi \models \omega$. Ist α keine Implikante von ω, so ist α auch keine Implikante von ϕ.

Beweis. Aus $\phi \models \omega$ folgt die Kontraposition: Ist α eine Implikante von ϕ, so ist α auch eine Implikante von ω. □

Satz 1
Es sei β ein Modell der Formel ϕ, das mit dem DPLL-Algorithmus berechnet wurde. Des Weiteren sei $P \subseteq Var(\phi)$ die Menge der Variablen, deren Belegungen in β propagiert wurden. Ist α eine Implikante von ϕ mit Basis M, die eine Einschränkung von β darstellt (d. h. $\beta_{|M} = \alpha$), so gilt $P \subseteq M$.

Beweis nach [DFLBM13]. Es sei $x \in P$. Die Annahme $x \notin M$ wird zum Widerspruch geführt. Mit der Formel ω sei die Klausel gegeben, aufgrund derer die Belegung $x \mapsto \beta(x)$ propagiert wurde. Es gilt $\phi \models \omega$. Wegen Lemma 1 reicht es zu begründen, dass α keine Implikante von ω ist. Nach Voraussetzung ist das Modell β eine Erweiterung von α. Wegen $x \notin M$ ist ebenso die Abbildung

$$\tilde{\beta}(y) = \begin{cases} \beta(y), & y \neq x \\ 1 - \beta(y), & y = x \end{cases}$$

eine Erweiterung von α. Für die Erweiterung $\tilde{\beta}$ von α gilt jedoch $\tilde{\beta} \not\models \omega$, da ω unter β nur aufgrund der Belegung von x zu 1 evaluiert – ansonsten hätte ω nicht die Belegung $x \mapsto \beta(x)$ propagiert. Somit kann α keine Implikante von ω sein. □

Variablen propagierter Belegungen dürfen somit nicht aus der Basis gelöscht werden. Die Frage, ob eine Variable, deren Belegung entschieden wurde, gelöscht werden darf, ist algorithmisch aufwändiger zu beantworten.
Für eine einzelne Klausel ω kann eine Variable x genau dann aus der Basis M einer Implikante α von ω gelöscht werden, wenn einer der beiden folgenden Fälle gilt. Dabei sei l das Literal der Variablen x, das unter β zu 1 evaluiert.

(F1) ω enthält nicht das Literal l.

(F2) ω enthält ein Literal $\tilde{l} \neq l$, das unter α zu 1 evaluiert.

Eine Variablenbelegung α ist genau dann eine Implikante einer KNF ϕ, falls α Implikante aller Klauseln ω aus ϕ ist. Eine Methode, die für eine KNF entscheidet, ob eine Variable x aus der Basis M einer Implikante gelöscht werden darf, muss gegebenenfalls für jede Klausel nachweisen, dass einer der beiden Fälle (F1) oder (F2) gilt.
Algorithmus 2 stellt in knapper Form die Berechnung von Primimplikanten dar. Der Ausdruck $Req(\phi, \alpha, x)$ in Zeile 5 beschreibt den Fall, dass eine Klausel in ϕ existiert, für die x nicht aus der Basis von α gelöscht werden darf. Der Algorithmus geht nach

Algorithmus 2 : Implikantenberechnung aus einem DPLL-Modell [DFLBM13]

```
   Input  : KNF φ, DPLL-Modell β,
            Variablenmenge P der propagierten Belegungen
   Output : Primimplikante α mit Basis M und der Eigenschaft α = β_|M
 1 M ← Var(β);
 2 while M \ P ≠ ∅ do
 3 |   x ←Select(M \ P);
 4 |   α ← β_|M;
 5 |   if Req(φ, α, x) then
 6 |   |   P ← P ∪ {x};
 7 |   end
 8 |   else
 9 |   |   M ← M \ {x};
10 |   |   α ← α_|M;
11 |   end
12 end
13 return α;
```

und nach die Basisvariablen $M \setminus P$ durch, deren Belegung noch nicht propagiert wurde. Wird eine Variable x in der Basis benötigt, so darf sich die Belegung von x nicht ändern und x wird in die Menge P der Variablen mit propagierter Belegung aufgenommen (Zeile 6). Wird eine Variable hingegen nicht in der Basis benötigt, so wird sie

direkt gestrichen (Zeile 9 und 10). Insgesamt wird bei diesem Vorgehen für die nicht vom DPLL-Algorithmus propagierten Variablen x genau einmal $Req(\phi, \alpha, x)$ entschieden. Die hierfür gewählte Reihenfolge der Variablen wird mittels der Methode `Select` aus Zeile 3 definiert und ist entscheidend dafür, welche Primimplikante berechnet wird. Zuerst gewählte Variablen werden bevorzugt aus der Basis gelöscht.

Für eine effiziente Implementierung von Algorithmus 2 schlagen Déharbe et. al Verweislisten $W(l)$ vor, die für ein Literal l die Klauseln der KNF ϕ auflisten, in denen das Literal l vorkommt. Ist nun $Req(\phi, \alpha, x)$ für eine Variable x zu entscheiden, so reicht es, die Klauseln aus $W(x)$ im Fall $\beta(x) = 1$ bzw. die Klauseln aus $W(\neg x)$ im Fall $\beta(x) = 0$ zu betrachten. Nur für diese Klauseln ist es möglich, dass der oben beschriebene Fall (F1) nicht eintritt. Um darüber hinaus effizient zu erkennen, in welchen Fällen es möglich ist, dass Fall (F2) für eine Klausel aus $W(l)$ nicht eintritt, schlagen sie die Verwendung von Countern vor. Ein Counter zählt dabei für eine Klausel ω aus ϕ die Anzahl der enthaltenen Literale, die unter α zu 1 evaluieren. Mit dem Einsatz von Countern ist es möglich, den Algorithmus 2 so zu implementieren, dass er eine lineare Laufzeit besitzt, d. h. eine Laufzeit in $\mathcal{O}(\sum_{\omega \in \phi} |\omega|)$, wobei $|\omega|$ die Anzahl der Literale einer Klausel ω bezeichnet.

2.3.3 Numerische Ergebnisse für POF_{PPM}

In den Statistiken dieses Abschnittes wurden ausschließlich Variablen erfasst, deren Belegung noch nicht durch das Spezialisieren der Produktübersichtsformel fixiert wurde.

Mit den Einstellungen des SAT-Solvers aus Unterabschnitt 2.2.6 wurden zu den bereits dort untersuchten Settings der BM7/Markt-Granularität und BR-Granularität jeweils 100.000 Modelle berechnet. Auf der Basis dieser Modelle wurde jeweils eine Primimplikante mit dem Algorithmus 2 bestimmt. Die Variablen wurden dabei zufällig und unter Gleichverteilung mit der Methode `Select` ausgewählt.
In der Tabelle 7 ist der prozentuale Anteil der Variablen dargestellt, die im Modell eine propagierte Belegung aufweisen. Dem wird der prozentuale Anteil der Variablen gegenübergestellt, die der Basis der Primimplikante angehören. Für die BM7/Markt-Granularität ist zu erkennen, dass unter den Variablen mit nicht propagierten Belegungen der Anteil der Nicht-Basisvariablen überwiegt.
Dieses Ergebnis lässt vermuten, dass mit Primimplikanten eine verhältnismäßig große Anzahl von Konfigurationsvarianten repräsentiert werden können. Um dies zu bestätigen, wurde der Anteil der Nicht-Basisvariablen unter den kundenwählbaren Variablen (d. h. Variablen der Kategorie (A1), vgl. Unterabschnitt 2.2.4) untersucht. Die Nicht-Basisvariablen, die auch als „Dont't Cares“ (DC) bezeichnet werden, wurden in zwei Klassen eingeteilt: In solche, die in keinem der 100.000 berechneten Modelle einen Teil

Segment	Ant. prop. Modellvar. in %	Ant. Basisvar. in %
205@BM	99.53	99.82
212@BM	99.06	99.77
222@BM	97.99	99.59
246@BM	97.68	99.06
2050041@BM+374@LD	46.98	64.82
2120021@BM+374@LD	68.17	77.16
2122051@BM+374@LD	46.14	60.91
2462121@BM+531@LD	39.99	54.91
2050042@BM+537@LD	46.71	62.80
2120362@BM+537@LD	72.39	78.20
2050041@BM+543@LD	50.16	66.01
2462471@BM+571@LD	43.47	57.16
2050401@BM+581@LD	46.63	64.99
2221821@BM+705@LD	36.10	55.45
1760422@BM+839@LD	31.08	45.20
1760522@BM+839@LD	27.69	41.58
2050402@BM+839@LD	43.49	59.22

Tabelle 7: Statistiken zur Implikantenberechnung (Mittelw. zu $N = 100.000$)

Segment	modellunabh. DC in %	modellabh. DC in %
205@BM	26.01	16.71
212@BM	9.57	20.61
222@BM	14.13	8.85
246@BM	21.90	9.51
2050041@BM+374@LD	40.80	13.27
2120021@BM+374@LD	42.23	6.36
2122051@BM+374@LD	48.48	6.42
2462121@BM+531@LD	54.95	0.17
2050042@BM+537@LD	44.63	6.38
2120362@BM+537@LD	44.18	3.93
2050041@BM+543@LD	41.81	1.71
2462471@BM+571@LD	56.36	0.18
2050401@BM+581@LD	41.77	0.10
2221821@BM+705@LD	52.61	0.10
1760422@BM+839@LD	63.72	4.77
1760522@BM+839@LD	68.34	1.26
2050402@BM+839@LD	47.63	1.03

Tabelle 8: Anteil der „Don't Cares“ unter den kundenwählbaren Variablen

der Basis darstellten und in solche, die abhängig vom berechneten Modell in der Basis auftraten – d. h. mindestens einmal Teil der Basis und mindestens einmal nicht Teil der Basis waren. Die Ergebnisse sind in Tabelle 8 zu sehen. Sie zeigen, dass für die BM7/Markt-Granularität etwa $40\% - 70\%$ der kundenwählbaren Variablen stets beliebig belegt werden können. Des Weiteren kann aus den Daten gefolgert werden, dass eine feste Auswahl von mindestens 30% der kundenwählbaren Variablen in jeder Basis auftritt.
Auf der gröberen BR-Granularität ist im Vergleich zur BM7/Markt-Granularität der Anteil der modellabhängigen Nicht-Basisvariablen höher und der Anteil der modellunabhängigen Nicht-Basisvariablen niedriger. Daraus ergibt sich als Konsequenz, dass sich die modellunabhängigen Basisvariablen in den unterschiedlichen Segmenten der Granularität BM7/Markt nur teilweise gleichen. Diese Beobachtung ist für die Interpretation nachfolgender Ergebnisse nützlich.

Die beobachteten Freiheitsgrade der Produktübersicht werden in Kapitel 4 bei der Entwicklung eines effizienten Verfahrens für die Konsistenz- und Intervallrechnung berücksichtigt.

3 Produktkonfiguration mit pseudo-boolescher Optimierung

In diesem Kapitel wird beschrieben, wie Konfigurationen berechnet werden können, die einerseits eine lineare Zielfunktion optimieren und andererseits die Produktübersicht erfüllen. Für Testinstanzen, die auf der in Abschnitt 2.2 entwickelten Produktübersichtsformel POF_{PPM} basieren, werden numerische Ergebnisse präsentiert. Die effiziente Berechnung optimaler Konfigurationen wird anschließend in Kapitel 4 benötigt.

3.1 Pseudo-boolesche Constraints

3.1.1 Pseudo-boolesche Funktionen

Für n aussagenlogische Variablen $x_1, \ldots, x_n$ existieren 2^n verschiedene Variablenbelegungen. Diese können durch eine *boolesche Funktion*

$$f_B : \{0,1\}^n \to \{0,1\},$$

differenziert werden, z. B. bezüglich einer Gültigkeit wie Baubarkeit (vgl. Unterabschnitt 2.2.3). Mit funktional vollständigen Junktorenmengen, wie $\{\neg, \wedge, \vee\}$, $\{\neg, \wedge\}$ oder $\{\neg, \vee\}$, kann jede boolesche Funktion durch eine aussagenlogische Formel dargestellt werden.

Eine feinere Differenzierung der Variablenbelegungen ist mit den *pseudo-booleschen Funktionen*

$$f_{PB} : \{0,1\}^n \to \mathbb{Q}$$

möglich. Sie ordnen jeder Variablenbelegung eine rationale Zahl zu, mit welcher beispielsweise Kosten modelliert werden können. Eine pseudo-boolesche Funktion wird üblicherweise als gewichtete Summe von Konjunktionen

$$f_{PB} = \sum_{i=1}^{m} a_i \bigwedge_{j=1}^{k_i} l_{ij}, \qquad a_i \in \mathbb{Q} \tag{17}$$

dargestellt. Um in dieser Darstellung den Funktionswert für eine Variablenbelegung zu ermitteln, werden die Koeffizienten a_i aller zum Wahrheitswert 1 evaluierenden Konjunktionen aufsummiert.
Iteriert β über alle 2^n verschiedenen Variablenbelegungen, so ist die kanonische Darstellung einer pseudo-booleschen Funktion f_{PB} durch $\sum_\beta f_{PB}(\beta) \bigwedge_{l \in \beta} l$ gegeben, wobei $l \in \beta$ genau dann gilt, wenn das Literal l unter der Variablenbelegung β zum Wahrheitswert 1 evaluiert.

Definition 6 (lineare pseudo-boolesche Funktion)
Eine pseudo-boolesche Funktion f_{LPB} ist *linear*, falls sie sich durch

$$f_{LPB} = \sum_{i=1}^{m} a_i l_i, \qquad a_i \in \mathbb{Q} \tag{18}$$

darstellen lässt.

In dieser Darstellung hängen die zur Summe beitragenden Koeffizienten a_i direkt von einzelnen Literalen l_i ab.
Für eine Charakterisierung der Linearität seien zwei beliebige Belegungen α und β gegeben, unter denen das Literal l zum Wahrheitswert 1 evaluiert. Zusätzlich seien mit α' und β' Variablenbelegungen gegeben, die

1. $\alpha'(l) = \beta'(l) = 0$
2. $\alpha_{|M} \equiv \alpha'_{|M},\ \beta_{|M} \equiv \beta'_{|M},\ M := Var(\alpha) \setminus Var(l)$

erfüllen. Die Linearität von f_{LPB} drückt sich dann in der Gleichung

$$f_{LPB}(\alpha) - f_{LPB}(\alpha') = f_{LPB}(\beta) - f_{LPB}(\beta') \tag{19}$$

aus. Andererseits ist jede pseudo-boolesche Funktion linear, die die Gleichung (19) für alle Literale l und für alle entsprechenden Variablenbelegungen α, β, α' und β' erfüllt.

3.1.2 Pseudo-boolesche Constraints

Mit Constraints werden im Folgenden immer Bedingungen an Variablenbelegungen bezeichnet. Beispielsweise ist ein *boolescher Constraint* durch eine boolesche Funktion f_B (meist in Form einer booleschen Formel) gegeben, der von einer Variablenbelegung β genau dann erfüllt wird, falls f_B unter β zum Wahrheitswert 1 evaluiert.

Definition 7 (PB-Constraint)
Ein *pseudo-boolescher Constraint* (PB-Constraint) ist durch eine pseudo-boolesche Funktion f_{PB}, einen Vergleichsoperator $\triangleright \in \{<, \leq, >, \geq, =\}$ und eine rechte Seite $b \in \mathbb{Q}$ gegeben. Er wird genau dann von einer Variablenbelegung β erfüllt, falls $f_{PB}(\beta) \triangleright b$ gilt.

Üblicherweise werden PB-Constraints in der Form

$$\sum_{i=1}^{m} a_i \bigwedge_{j=1}^{k_i} l_{ij} \triangleright b \tag{20}$$

dargestellt.

Definition 8 (LPB-Constraint)
Ein *linearer pseudo-boolescher Constraint* (LPB-Constraint) ist ein PB-Constraint mit linearer pseudo-boolescher Funktion:

$$\sum_{i=1}^{m} a_i l_i \triangleright b. \tag{21}$$

Satz 2
Jeder PB-Constraint kann durch eine Menge von LPB-Constraints erfüllbarkeitsäquivalent repräsentiert werden.

Beweis. Zunächst wird in (20) jede Konjunktion aus Literalen $\bigwedge_{j=1}^{k_i} l_{ij}$ durch eine Hilfsvariable v_i ersetzt. Der resultierende Constraint ist linear. Mit je 2 weiteren LPB-Constraints wird für jede der m Konjunktionen die Äquivalenz zur zugehörigen Hilfsvariable erzwungen:

$$\begin{aligned} &\text{„}\rightarrow\text{“}: && \textstyle\sum_{j=1}^{k_i} \overline{l_{ij}} + v_i \geq 1 && i = 1, \ldots, m \\ &\text{„}\leftarrow\text{“}: && \textstyle\sum_{j=1}^{k_i} l_{ij} - m v_i \geq 0 && i = 1, \ldots, m. \end{aligned}$$

□

3.1.3 Cardinality Constraints

Eine wichtige Unterklasse der LPB-Constraints bilden die *Cardinality Constraints* (vgl. Primärgruppen aus Unterabschnitt 2.2.5). Sie sind dadurch charakterisiert, dass alle in dem Constraint auftretenden Literale einen gleichwertigen Koeffizienten besitzen. Standardmäßig wird zur Beschreibung eines Cardinality Constraints der Koeffizient $1 \in \mathbb{N}$ und somit eine rechte Seite $k \in \mathbb{N}$ gewählt. Demnach reicht es aus, zwischen drei verschiedenen Typen von Cardinality Constraints zu unterscheiden.
Der Cardinality Constraint $atleast(k, \{l_1, \ldots, l_n\})$ entspricht dem LPB-Constraint

$$l_1 + \cdots + l_n \geq k \tag{22}$$

und fordert, dass mindestens k der n enthaltenen Literale zum Wahrheitswert 1 evaluieren. Die beiden anderen Cardinality Constraints lassen sich mit dem atleast-Constraint darstellen:

$$\begin{aligned} atmost(k, \{l_1, \ldots, l_n\}) &\equiv \text{atleast}(n-k, \{\overline{l_1}, \ldots, \overline{l_n}\}), \\ exactly(k, \{l_1, \ldots, l_n\}) &\equiv \text{atleast}(k, \{l_1, \ldots, l_n\}) \\ &\quad \wedge \text{atmost}(k, \{l_1, \ldots, l_n\}). \end{aligned}$$

Wird der Vergleichsoperator „≥“ in (22) durch einen strikten Vergleichsoperator „>“ oder „<“ ersetzt, so ergibt sich der Constraint atleast$(k+1, \{l_1, \ldots, l_n\})$ bzw. der Constraint atmost$(k-1, \{l_1, \ldots, l_n\})$.

Klauseln können als spezielle LPB-Constraints verstanden werden. So ist eine Klausel $(l_1 \vee \cdots \vee l_n)$ genau dann erfüllt, wenn mindestens eines seiner Literale l_i zum Wahrheitswert 1 evaluiert. Eine Klausel entspricht daher dem Cardinality Constraint atleast$(1, \{l_1, \ldots, l_n\})$ und hat als LPB-Constraint die Darstellung

$$l_1 + \cdots + l_n \geq 1. \tag{23}$$

3.1.4 Ausdrucksstärke von pseudo-booleschen Constraints

Wird ein boolescher Constraint durch eine KNF mit n Klauseln repräsentiert, so kann dieser mit n LPB-Constraints der Form (23) ausgedrückt werden. Andererseits kann ein LPB-Constraint exponentiell viele Klauseln benötigen, um als KNF dargestellt zu werden. Dazu wird der Constraint atleast$(k, \{l_1, \ldots, l_n\})$ betrachtet. Er ist genau dann erfüllt, wenn nicht mehr als $n-k$ seiner Literale falsified sind. Daher impliziert der atleast-Constraint jede Klausel, die sich aus $n-k+1$ seiner Literale zusammensetzt. Die Menge dieser $\binom{n-k+1}{n}$ Klauseln ist redundanzfrei und hat für $k = \lfloor \frac{n}{2} \rfloor$ eine exponentielle Größe in n.
Mit der Einführung von Hilfsvariablen ist es möglich, jeden LPB-Constraint mit nur einer linearen Anzahl von Klauseln zu übersetzen [War98]. Die Linearität bezieht sich dabei auf die Größe des Constraints, welche linear von der Anzahl der enthaltenden Literale und der Stellenanzahl der binären Koeffizientendarstellungen abhängt.

3.1.5 Normalisierung von pseudo-booleschen Constraints

Moderne SAT-Solver wie MiniSAT operieren auf einer erfüllbarkeitsäquivalenten KNF. Die Klauseln der KNF können als normalisierte Constraints verstanden werden. Sie erlauben ein besonders effizientes Propagieren von Literalen und eine, im Falle eines Konfliktes, einfache Analyse mittels gewöhnlicher Resolution (vgl. Unterabschnitt 2.1.2).
Für LPB-Constraints ist ebenso eine Normalform definiert [CK03], deren Motivation die Verwendung in Solvern mit CDCL-ähnlicher Architektur ist.

Definition 9 (Normalform von LPB-Constraints)
Ein LPB-Constraint ist in *Normalform*, falls er die Form

$$\sum_i a_i l_i \geq b, \quad a_i, b \in \mathbb{Z}^+, \tag{24}$$

besitzt.

Satz 3
Jeder PB-Constraint kann äquivalent als Menge von normalisierten LPB-Constraints formuliert werden.

Beweis. Wegen Satz 2 reicht es zu begründen, dass jeder LPB-Constraint der Form (21) in die Normalform transformiert werden kann. Eine Möglichkeit ist die Abarbeitung der folgenden Äquivalenzumformungen:

(T1) Multipliziere beide Seiten mit dem Hauptnenner von a_i und b.

(T2) Ersetze „=“ gegebenenfalls durch zwei LPB-Constraints mit „≤“ und „≥“.

(T3) Im Fall von „<“ oder „≤“ multipliziere beide Seiten mit -1.

(T4) Ersetze „>“ gegebenenfalls durch „≥“ und setze $b := b + 1$.

(T5) Falls a_i negativ ist, so ersetze l_i durch $1 - \overline{l_i}$.

(T6) Bringe alle freien Koeffizienten auf die rechte Seite.

□

Beispiel 2
Der LPB-Constraint

$$2x_1 - 3x_2 + \overline{x_3} < 0$$

ist äquivalent zu $x_2 \wedge (\overline{x_1} \vee x_3)$.

Transformation in Normalform:

$$\begin{aligned}
2x_1 - 3x_2 + \overline{x_3} &< 0 \\
-2x_1 + 3x_2 - \overline{x_3} &> 0 \quad (T3) \\
-2x_1 + 3x_2 - \overline{x_3} &\geq 1 \quad (T4) \\
-2(1 - \overline{x_1}) + 3x_2 - (1 - x_3) &\geq 1 \quad (T5) \\
2\overline{x_1} + 3x_2 + x_3 &\geq 4 \quad (T6)
\end{aligned}$$

Der LPB-Constraint nach dem Transformationsschritt (T6) ist in Normalform und ebenso äquivalent zu $x_2 \wedge (\overline{x_1} \vee x_3)$.

3.2 Erfüllbarkeit und Optimierung

Definition 10 (PBS)
Das *pseudo-boolesche Erfüllbarkeitsproblem* (PBS) ist das Problem, zu entscheiden, ob eine Variablenbelegung existiert, die eine gegebene Menge von LPB-Constraints erfüllt.

Satz 3 erlaubt es, eine Menge von PB-Constraints stets durch eine erfüllbarkeitsäquivalente Menge von normalisierten LPB-Constraints

$$\sum_i a_{ij} l_i \geq b_j \quad \forall j \tag{25}$$

zu ersetzen. Die *Normalform* einer PBS-Instanz wird dadurch definiert, dass alle ihre LPB-Constraints in Normalform sind. Eine normalisierte PBS-Instanz mit nur einem LPB-Constraint ist bei n Variablen immer in $\mathcal{O}(n)$ lösbar. Es reicht in diesem Fall zu überprüfen, ob die Summe der Koeffizienten der linken Seite größer-gleich der rechten Seite ist. Schon mit zwei normalisierten LPB-Constraints ist es möglich, eine NP-vollständige Instanz von PBS zu formulieren. So codiert zum Beispiel $\sum_{i=1}^{n} a_i x_i = k$ das NP-vollständige „subset sum problem“. Das Problem besteht darin, aus einer Menge von natürlichen Zahlen a_i eine Teilmenge so zu bestimmen, dass sich die Zahlen der Teilmenge zur Zahl k aufsummieren.

Um zwischen den erfüllenden Variablenbelegungen einer erfüllbaren PBS-Instanz zu differenzieren, kann zusätzlich eine lineare pseudo-boolesche Funktion definiert werden.

Definition 11 (PBO)
Das pseudo-boolesche Optimierungsproblem (PBO) beschreibt die Aufgabe, eine lineare pseudo-boolesche Funktion unter einer Menge von LPB-Constraints zu minimieren bzw. zu maximieren.

PBO ist ein Spezialfall der ganzzahligen linearen Programmierung (ILP) und wird dort üblicherweise in der Form

$$\begin{aligned} & \min \quad z(x) = \textstyle\sum_i c_i x_i \\ \text{s.th.} & \\ & \textstyle\sum_i a_{ij} x_i \geq b_j \quad \forall j \\ & x_i \in \{0,1\} \quad \forall i \end{aligned} \tag{26}$$

dargestellt, wobei $a_{ij}, c_i \in \mathbb{Q}$ gilt. Diese Darstellung fordert explizit, dass die Entscheidungsvariablen x_i aussagenlogischen Variablen entsprechen. Somit könnte diese Darstellung ohne Weiteres einem ILP-Standardsolver übergeben werden.
Eine normalisierte PBS-Instanz (25) wäre für einen ILP-Standardsolver in die Form

$$\begin{aligned} & \min \quad z(x) = 0 \\ \text{s.th.} & \\ & \textstyle\sum_i a_{ij} l_i \geq b_j \quad \forall j \\ & x_i + \overline{x_i} = 1 \quad \forall i \\ & x_i, \overline{x_i} \in \{0,1\} \quad \forall i \end{aligned} \tag{27}$$

zu überführen. Da in normalisierten LPB-Constraints auch negative Literale zugelassen sind, muss die Abhängigkeit zu den jeweiligen positiven Literalen codiert werden. Alternativ kann auch auf die Constraints $x_i + \overline{x_i} = 1$ verzichtet werden, indem negative Literale $\overline{x_i}$ mit $1 - x_i$ substituiert werden. Dann sind die LPB-Constraints nicht mehr zwingend normalisiert.
In Abschnitt 3.4 wird erläutert, wie PBS bzw. PBO mit CDCL-ähnlichen Techniken eines auf DPLL basierenden Solvers berechnet werden können. In diesem Zusammenhang wird von nun an die implizite Logik zwischen positiven und negativen Literalen stets vorausgesetzt.
Analog zu PBS wird für PBO eine vereinfachte, normalisierte Darstellung mit normalisierten LPB-Constraints als Nebenbedingungen eingeführt:

$$\begin{aligned} & \min \quad z(x) = \textstyle\sum_i c_i l_i \\ \text{s.th.} \quad & \\ & \textstyle\sum_i a_{ij} l_i \geq b_j \quad \forall j. \end{aligned} \tag{28}$$

Beispiel 3
Eine Fahrzeugvariante ist baubar, wenn sie jeden Constraint $C \in POF$ der Produktübersicht POF erfüllt (vgl. Abschnitt 2.2). Die Constraints $C \in POF$ sind im Wesentlichen durch Klauseln und Cardinality Constraints gegeben. So wird der Sachverhalt, dass die Ausstattung x in Land l für das Baumuster bm nicht verfügbar ist, durch die Klausel $\overline{x} + \overline{l} + \overline{bm} \geq 1$ codiert. Mit dem Cardinality Constraint exactly$(1, \{l_1, \ldots, l_n\})$ wird ausgedrückt, dass jede Fahrzeugvariante genau einem Land l_i zuzuordnen ist.
Den Ausstattungen $x_1, \ldots, x_n$ seien im Land l für das Baumuster bm die Kosten $c_1, \ldots, c_n \in \mathbb{Z}$ zugeordnet. Eine beispielhafte PBS-Instanz ist die Frage danach, ob eine baubare Variante von Baumuster bm in Land l existiert, die über die Ausstattungen x_{i_1} und x_{i_2} verfügt und höchstens $K \in \mathbb{Z}$ kostet. Anders gefragt: Ist die normalisierte PBS-Instanz

$$\begin{aligned} C & \qquad \forall C \in POF \\ l + bm + x_{i_1} + x_{i_2} & \geq 4 \\ \textstyle\sum_i c_i \overline{x_i} & \geq -K + \textstyle\sum_i c_i \end{aligned}$$

erfüllbar?

Beispiel 4
Sollen in Beispiel 3 die Kosten minimiert werden, so ergibt sich die PBO-Instanz

$$\begin{aligned} & \min \ \textstyle\sum_i c_i x_i \\ \text{s.th.} \quad & \\ & C \qquad \forall C \in POF \\ & l + bm + x_{i_1} + x_{i_2} \geq 4. \end{aligned}$$

3.3 Branch and Cut

Bevor in Abschnitt 3.4 erläutert wird, wie Instanzen von PBS und PBO mit einer Variante des DPLL-Algorithmus berechnet werden können, wird vorbereitend zunächst auf Branch and Cut eingegangen – den klassischen Ansatz der ganzzahligen Programmierung. Mit Branch and Cut wird die Verwendung von Cutting Planes in einem auf Branch and Bound basierenden Verfahren bezeichnet.

3.3.1 Branch and Bound

Branch and Bound ist eine Lösungsstrategie für kombinatorische Optimierungsprobleme, welche 1960 von A. H. Land und A. G. Doig vorgeschlagen wurde [LD60]. Branch and Bound operiert auf einem Entscheidungsbaum, dessen Blätter die Grundmenge des Optimierungsproblems darstellen. Eine Implementierung von Branch and Bound zur Lösung gemischter ganzzahliger Probleme erfolgte erstmals 1965 durch R. J. Dakin [Dak65].
Im Falle einer durch (26) gegebenen 0/1-Optimierungsaufgabe ist der Entscheidungsbaum binär (siehe Abbildung 6) und hält in seinen Blättern die 2^n möglichen Belegungen der n Entscheidungsvariablen x_i. Das Ziel besteht darin, eine Verzweigung (engl.: Branch) zu einem Blatt einer zulässigen, optimalen Lösung zu bestimmen. Um nicht alle 2^n möglichen Pfade überprüfen zu müssen, werden während der Suche Teilbäume immer dann mit einer Schranke (engl.: Bound) vom Lösungsraum abgegrenzt, wenn für sie bekannt ist, dass sie keine bessere Lösung als die beste bereits gefundene beinhalten. Eine bessere Lösung kann in einem Teilbaum ausgeschlossen werden, falls einer der beiden folgenden Fälle sicher gilt:

1. Der Teilbaum enthält keine zulässige Lösung.

2. Jede zulässige Lösung im Teilbaum hat einen Zielfunktionswert höchstens so gut, wie die bisher beste bekannte Lösung x_{opt}.

Ist in den Nebenbedingungen beispielsweise die Ungleichung

$$2x_1 - 3x_2 + \overline{x_3} < 0$$

aus Beispiel 2 vertreten, so ist eine zulässige Lösung in Teilbäumen unterhalb der Verzweigung $x_2 = 0$ nicht möglich.
Bezogen auf den 2. Fall wird eine untere Schranke lb für den besten Zielfunktionswert ermittelt, der im Teilbaum auftritt. Gilt $lb \geq z(x_{\text{opt}})$, so darf der Teilbaum ignoriert werden.
Die Effizienz von Branch and Bound hängt insbesondere von der Güte der berechneten unteren Schranke lb ab. Zur Berechnung von lb wird entsprechend der Verzweigung des inneren Knotens das lokale ganzzahlige Programm von (26) aufgestellt. Mit lb ist

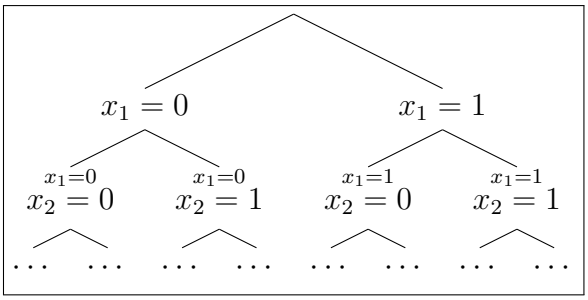

Abbildung 6: Binärer Entscheidungsbaum

dann eine untere Schranke für das lokale Programm zu bestimmen. In der ganzzahligen Programmierung werden hierfür die Constraints $x_i \in \{0, 1\}$ zu $x_i \in [0, 1]$ abgeschwächt – es wird das sogenannte relaxierte lineare Programm betrachtet. Die optimale Lösung des relaxierten Programmes stellt eine untere Schranke lb dar und kann beispielsweise mit dem Simplex-Algorithmus berechnet werden (vgl. Unterabschnitt 4.4.3).
Mit dem Einsatz von Cutting Planes als zusätzliche Constraints im relaxierten Teilproblem können stärkere untere Schranken lb berechnet werden.

3.3.2 Cutting Planes

Die Grundmenge des relaxierten Teilproblems aus Branch and Bound ist ein Polyeder $P = \{x \mid Ax \geq b\}$. Angenommen $x_E \in P$ ist jener Punkt, welcher bei Branch and Bound zur Schranke $lb = z(x_E)$ führt. Ist x_E ganzzahlig, so wird die bisher beste bekannte Lösung x_{opt} gegebenenfalls durch x_E ersetzt. In jedem Fall wird der nachfolgende Teilbaum nicht weiter verzweigt, da er keine bessere Lösung beinhalten kann. Ist x_E nicht ganzzahlig und gilt $lb = z(x_E) < z(x_{\text{opt}})$, so kann im nächsten Schritt der Teilbaum verzweigt werden oder in die Berechnung einer besseren unteren Schranke $lb' \geq lb$ investiert werden. Letzteres kann mit Cutting Planes realisiert werden. Ein *Cutting Plane* ist eine Ungleichung $a^t x \geq \alpha$, welche P ausschließlich um nicht ganzzahlige Punkte beschneidet. Er ist demnach durch

$$P \cap \mathbb{Z}^n = P' \cap \mathbb{Z}^n, \; P' := P \cap \{a^t x \geq \alpha\} \tag{29}$$

charakterisiert. Diese Eigenschaft garantiert, dass auch

$$lb' := \min\{z(x) | x \in P'\} \geq lb \tag{30}$$

eine gültige untere Schranke für das ganzzahlige Programm darstellt. Der gewünschte Fall $lb' > lb$ kann nur eintreten, falls der Cutting Plane den Punkt x_E von P separiert, wenn also $a^t x_E < \alpha$, d. h. $x_E \notin P'$ gilt.

Grundsätzlich ist auch der alleinige Einsatz von Cutting Planes zum Lösen ganzzahliger

Optimierungsprobleme möglich. Die sogenannten Schnittebenenverfahren beschneiden das relaxierte Programm des globalen ganzzahligen Programmes solange mit Cutting Planes, bis die Lösung ganzzahlig ist. Ein derartiges Schnittebenenverfahren wurde erstmals Mitte der 1950er Jahre von G. Dantzig, R. Fulkerson und S. Johnson zum Lösen des Problems vom Handlungsreisenden eingesetzt [DFJ54]. Die dafür verwendeten Cutting Planes nutzen die dem Problem zugrunde liegende Struktur aus. Problemunabhängige Cutting Planes wurden 1958 von R. Gomory veröffentlicht [Gom58]. Die Gomory-Schnitte waren zunächst nicht praxisrelevant, da sie sich für eine schnelle Konvergenz im Schnittebenenverfahren als zu schwach herausstellten. Im Zusammenwirken mit Branch and Bound wurden sie jedoch Anfang der 1990er Jahre wiederentdeckt [Cor07] und sind nun Bestandteil generischer „state of the art“ Solver wie CPLEX.

Als ein repräsentatives Beispiel werden im Folgenden Hooker's Cutting Planes definiert. Sie stehen in direktem Zusammenhang zur Resolution der Aussagenlogik [Hoo03] und verallgemeinern diese für LPB-Constraints. Einerseits sind sie daher besonders geeignet, um die Beispiele 3 und 4 mit Branch and Cut zu lösen. Andererseits können sie eingesetzt werden, um PBS-Solver, die auf der CDCL-Architektur basieren, so zu erweitern, dass beim Suchprozess ausdrucksstärkere LPB-Constraints – und nicht ausschließlich Klauseln – gelernt werden (vgl. Unterabschnitt 3.4.2).

Definition 12 (Hooker's Cutting Planes)
Die Cutting Planes von Hooker sind durch die Ableitungsregeln *verallgemeinerte Resolution* und *Sättigung* definiert.

$$\frac{\begin{array}{c}\sum_i a_i l_i \geq b \\ \sum_i c_i l_i \geq d \\ \alpha, \beta \in \mathbb{N} \\ \alpha, \beta > 0\end{array}}{\sum_i (\alpha a_i + \beta c_i) l_i \geq \alpha b + \beta d} \qquad \text{(verallg. Resol.)}$$

$$\frac{\begin{array}{c}\sum_i a_i l_i \geq b \\ a_k > b\end{array}}{b l_k + \sum_{i \neq k} a_i l_i \geq b} \qquad \text{(Sättigung)}$$

In Kombination bilden beide Ableitungsregeln ein vollständiges Beweissystem: Ist ein System von LPB-Constraints unerfüllbar, so kann mit verallgemeinerter Resolution und Sättigung der Widerspruch $0 \geq 1$ abgeleitet werden [RM09]. Dazu wird das folgende Beispiel betrachtet.

Beispiel 5
Die LPB-Constraints (1) - (4) entsprechen Klauseln und bilden eine unerfüllbare KNF.

$$x_1 + x_2 \geq 1 \qquad (1)$$
$$x_1 + \overline{x_2} \geq 1 \qquad (2)$$
$$\overline{x_1} + x_2 \geq 1 \qquad (3)$$
$$\overline{x_1} + \overline{x_2} \geq 1 \qquad (4)$$
$$2x_1 \geq 1 \qquad (1)+(2)=(5)$$
$$2\overline{x_1} \geq 1 \qquad (3)+(4)=(6)$$
$$x_1 \geq 1 \qquad \text{Sättigung (5)} \rightarrow (7)$$
$$\overline{x_1} \geq 1 \qquad \text{Sättigung (6)} \rightarrow (8)$$
$$0 \geq 1 \qquad (7)+(8)=(9)$$

Mit aussagenlogischer Resolution ist aus den Klauseln (1) und (2) die Unit-Klausel x_1 und aus den Klauseln (3) und (4) die Unit-Klausel $\neg x_1$ ableitbar, weshalb die leere Klausel folgt (vgl. Unterabschnitt 2.1.2). Wird die verallgemeinerte Resolution angewandt, so ergeben sich zunächst die LPB-Constraints (5) und (6). Eine Addition dieser beiden Constraints würde mit $0 \geq 0$ nicht den gewünschten Widerspruch liefern. Die Begründung liegt darin, dass die nicht ganzzahlige Lösung $x_1 = \frac{1}{2}$ beide LPB-Constraints (5) und (6) erfüllt. Daher ist die Lösung $x_1 = \frac{1}{2}$ in den LPB-Constraints (5) und (6) zunächst mit jeweiliger Sättigung „abzuschneiden“.

3.4 DPLL-Algorithmus und LPB-Constraints

In Unterabschnitt 2.1.2 wurde der DPLL-Algortihmus vorgestellt. Er bildet die Basis moderner CDCL-Solver, wie z. B. MiniSAT, die sich im Wesentlichen durch effiziente *Unit-Propagation* und eine Konfliktanalyse mit *Clause-Learning* und *non-chronological Backtracking* definieren. Die in diesem Zusammenhang eingesetzten Techniken können derart verallgemeinert werden, dass es CDCL-Solvern ermöglicht wird auf LPB-Constraints und nicht auf Klauseln zu operieren [CK03, SS05]. Entsprechende Ansätze werden in diesem Abschnitt erläutert. Dabei wird den Darstellungen des Übersichtsartikels [RM09] gefolgt. Als Konsequenz ist der DPLL-Algorithmus in der Form von Algorithmus 1 auch zum Lösen von PBS einsetzbar.
Unter dem Einsatz von LPB-Constraints ist es gegebenenfalls möglich, Modelle zu berechnen, die eine Schranke bzgl. einer linearen Zielfunktion z unterschreiten. Folglich kann der DPLL-Algorithmus für PBS ohne großen Aufwand zu einem Algorithmus für PBO erweitert werden. Eine Möglichkeit ist die sogenannte *lineare Suche*, die in Algorithmus 3 dargestellt ist: Wurde ein Modell bestimmt, das die Nebenbedingungen erfüllt, so wird im nächsten Schritt ein Modell mit besserem Zielfunktionswert

bestimmt. Existiert kein Modell mit besserem Zielfunktionswert, so war das letzte Modell schon optimal.
Eine alternativer Ansatz ist durch die *binäre Suche* gegeben: Liegt der minimale Ziel-

Algorithmus 3 : PBO - lineare Suche mit DPLL

Input : PBS-Instanz P, Lineare Zielfunktion z auf Var(P)
Output : Optimaler Zielfunktionswert eines Modells von P

```
1  ub ← ∞
2  while true do
3      if SolutionFound (P) then
4          x ←LastModel(P);
5          ub ← z(x);
6          AddConstraint (P, z(x) ≤ ub − 1);
7      end
8      Decide (P);
9      while Deduce (P)=CONFLICT do
10         if Diagnose (P)=CONFLICT then
11             return ub;
12         end
13     end
14 end
```

funktionswert im Intervall $[L, U]$, so wird überprüft, ob ein Modell mit Zielfunktionswert $\leq M := (L + U)/2$ existiert. Ist dies der Fall, so liegt der minimale Zielfunktionswert im Intervall $[L, M]$, ansonsten im Intervall $[M, U]$. Der Suchraum wird mit jeder einzelnen PBS-Anfrage halbiert.
Ein wesentlicher Vorteil der linearen Suche gegenüber der binären Suche ist die Möglichkeit des Warmstarts. Bisher gesetzte Oberschranken sind nach einer weiteren Verschärfung der Oberschranke redundant, so dass keine Neuinitialisierung des PBS-Solvers nötig ist – bereits gelernte Constraints bleiben gültig.

3.4.1 Propagation

Nach einer Entscheidung bzw. nach einem Backtrack kommt es zu einer neuen Variablenbelegung. Eine in diesem Zusammenhang unverzichtbare Komponente von CDCL-Solvern ist Unit-Propagation. Mit Unit-Propagation werden induzierte Variablenbelegungen bzw. Literale berechnet: Ein bisher nicht belegtes Literal einer Klausel K wird genau dann aufgrund von K propagiert, wenn alle anderen Literale der Klausel K negiert sind. Ein propagiertes Literal kann zur Propagation weiterer Literale führen.
Im Falle von Cardinality Constraints ist das im Unterabschnitt 2.1.2 beschriebene 2-Watched-Literal-Konzept leicht zu verallgemeinern. Liegt ein Cardinality Constraint des Typs „$\geq k$“ vor, so sind $k + 1$ Literale zu beobachten. Existieren keine $k + 1$ Lite-

rale, die nicht negiert sind, so werden bis zu k Literale simultan propagiert.
Literal-Watching wird in [CK03] für LPB-Constraints

$$a_1 l_1 + \cdots + a_n l_n \geq b,\ a_i \geq 0,\ A := \sum_i a_i. \tag{31}$$

in Normalform konzipiert. Dies stellt nach Satz 3 keine Einschränkung dar.
Unter einer Teilbelegung ist der Constraint (31) genau dann nicht erfüllbar, wenn der *Slack*

$$s := \Big(A - \sum_{l_i \text{ negiert}} a_i\Big) - b$$

negativ ist. Der Term $A - \sum_{l_i \text{ negiert}} a_i$ beschreibt in diesem Zusammenhang das unter der Teilbelegung verbleibende Potential.
Ein unbelegtes Literal l_i mit

$$s - a_i < 0$$

ist zu propagieren, da der Slack negativ werden würde, wäre l_i negiert. Die Anzahl der zu beobachtenden Literale ist abhängig von der aktuellen Teilbelegung. Mit

$$M := \max_{l_i \text{ unbelegt}} (a_i)$$

ist eine Teilmenge W von nicht negierten Literalen zu beobachten, die

$$\sum_{l_i \in W} a_i \geq b + M$$

erfüllt. Erst nach Belegung von einer weiteren Variablen kann der Fall $s - a_i < 0$ eintreten. Um möglichst wenig Literale beobachten zu müssen, werden die Literale entsprechend der Größe ihrer Koeffizienten priorisiert.

Beispiel 6
Betrachtet wird der LPB-Constraint

$$8l_1 + 7l_2 + l_3 + 2l_4 + 3l_5 \geq 10.$$

Ist keines der enthaltenen Literale belegt, so ist $s = 11$ und kein Literal wird propagiert. Wegen $M = 8$ reicht es, die Literale l_1, l_2 und l_5 zu beobachten. Angenommen es wird $l_1 = 0$ gesetzt. Dann wird wegen $s = 3$ das Literal l_2 propagiert. Folglich ist $M = 3$ und es werden die Literale l_2, l_3, l_4 und l_5 beobachtet. Die Entscheidung $l_3 = 0$ würde nun das Literal l_5 propagieren und der Constraint wäre erfüllt.

3.4.2 Konfliktanalyse

In Unterabschnitt 2.1.2 wurde die Konfliktanalyse moderner CDCL-Solver erläutert. Auch im Falle eines auf DPLL basierenden PBS-Solvers können mittels aussagenlogischer Resolution UIPs (Unique Implication Points) gelernt werden. Dazu wird der LPB-Constraint aus Beispiel 6 betrachtet. Für die Belegung $l_1 = 0$ propagiert der Constraint das Literal l_2. Der Constraint impliziert somit die Klausel $l_1 \vee l_2$, welche innerhalb einer Konfliktanalyse unter der Belegung $l_1 = 0$ einen Grund für l_2 darstellt. Dies lässt sich verallgemeinern: Ist ω ein LPB-Constraint, der unter einer Teilbelegung β das Literal $\tilde{l}$ propagiert, so impliziert ω die Klausel

$$R(\tilde{l}) := \bigvee_{l \in \omega,\ \beta(l)=0} l \vee \tilde{l},\ \omega \models R(\tilde{l}), \tag{32}$$

welche einen Grund für eine positive Belegung von $\tilde{l}$ darstellt.
Für eine Konfliktanalyse mittels Resolution bedarf es noch einer Konfliktklausel K, die aus dem im Konflikt stehenden LPB-Constraint ω_K abgeleitet wird und den Startpunkt für die Berechnung des UIP bildet (vgl. Unterabschnitt 2.1.2). Es sei β die Teilbelegung, die in ω_K zum Konflikt führt:

$$K := \bigvee_{l \in \omega_K,\ \beta(l)=0} l. \tag{33}$$

Eventuell können die Klauseln $R(\tilde{l})$ und K noch verschärft werden – nicht immer sind alle Literale mit $\beta(l) = 0$ für die Literalpropagation bzw. für das Auftreten des Konfliktes nötig.

Wie in Unterabschnitt 3.1.4 erläutert wurde, sind LPB-Constraints ausdrucksstärker als Klauseln. In Bezug auf den Nachweis der Unerfüllbarkeit einer aussagenlogischen Formel mittels Resolution zeigt sich dies auch am Schubfachprinzip (engl.: pigeonhole principle). Das Schubfachprinzip wird beispielsweise angewendet, um zu begründen, dass p Tauben nur dann jeweils ein eigenes Fach von h Fächern zusteht, wenn $p \leq h$ gilt. Wird das Problem, die Tauben auf jeweils eigene Fächer aufzuteilen, als KNF codiert, so benötigt ein aussagenlogischer Resolutionsbeweis exponentiell viele Klauseln um zu zeigen, dass für $p > h$ keine Lösung existiert [Hak85]. Mit den gelernten Resolventen ist nur eine „langsame“ Annäherung an die leere Klausel möglich.
Wird das Problem hingegen mit Cardinality Constraints codiert, so ist mit den Cutting Planes von Hooker aus Unterabschnitt 3.3.2 ein Beweis der Unerfüllbarkeit für $p > h$ in quadratischer Länge möglich [RM09]. Dies motiviert das Lernen von LPB-Constraints in auf DPLL basierenden PBS- bzw. PBO-Solvern.

Analog dem Lernen von Klauseln muss der gelernte Constraint Folgendes erfüllen:

1. Der Constraint muss im Konfliktlevel unerfüllbar sein.
2. Der Constraint ist in einem höheren Entscheidungslevel unit, d. h. er propagiert mindesten ein Literal.

Das nächste Beispiel zeigt, dass die 1. Forderung nicht selbstverständlich ist.

Beispiel 7
Für die LPB-Constraints

$$\begin{aligned} \omega_1: &\quad \overline{x_2} + \overline{x_3} + x_6 \geq 2, &\quad s = 0 \\ \omega_2: &\quad 3\overline{x_1} + x_3 + x_5 + x_9 \geq 3, &\quad s = 2 \\ \omega_3: &\quad 3x_1 + 2x_2 + x_7 + 2\overline{x_8} \geq 3, &\quad s = -2 \end{aligned}$$

seien die folgenden Belegungen gegeben

$$x_8 = 1@1,\ x_6 = 0@2,\ x_2 = 0@2[\omega_1],\ x_3 = 0@2[\omega_1],\ x_1 = 0@2[\omega_2].$$

Mit „$@L$“ sind die Entscheidungslevel L der Belegungen angegeben – für die propagierten Belegungen sind in eckigen Klammern die Reason-Constraints zu finden. Die Entscheidungen $x_8 = 1$ und $x_6 = 0$ führen zu einem Widerspruch in dem Constraint ω_3, welcher einen negativen Slack aufweist. Der Grund für die letzte Belegung einer Variablen aus ω_3 ($x_1 = 0$) ist durch den Constraint ω_2 gegeben. Analog zur Konfliktanalyse mit Resolution gilt es, aus ω_3 und ω_2 einen Constraint abzuleiten, der im Konfliktlevel unerfüllbar ist und nicht das Literal x_1 enthält. Der direkte Versuch der einfachen Addition

$$(\omega_2 + \omega_3): \quad 2x_2 + x_7 + 2\overline{x_8} + x_3 + x_5 + x_9 \geq 3, \quad s = 0$$

scheitert, da die Addition der Slacks zu $s = 2 + (-2) = 0$ führt, d. h. $(\omega_2 + \omega_3)$ ist im Konfliktlevel erfüllbar. Der Constraint ω_2 ist mit dem Slack $s = 2$ „over-satisfied“. Dieses Problem kann umgangen werden, indem ω_2 zunächst abgeschwächt wird:

$$\text{weakening}(\omega_2) = \omega_2': \quad 3\overline{x_1} + x_3 + x_5 \geq 2, \quad s = 2.$$

Die abgeschwächte Aussage ω_2' ist unabhängig vom Literal x_9, da dieses eliminiert wurde. Dadurch konnte im Gegenzug die rechte Seite reduziert werden. Der Slack hat sich mit $s = 2$ nicht verändert, kann nun jedoch mit Saturation verringert werden:

$$\mathit{saturation}(\omega_2') = \omega_2'': \quad 2\overline{x_1} + x_3 + x_5 \geq 2, \quad s = 1.$$

Der Constraint ω_2'' erlaubt nun die gewünschte Kombination:

$$(3\omega_2'' + 2\omega_3) = \omega_4 : \quad 4x_2 + 2x_7 + 4\overline{x_8} + 3x_3 + 3x_5 \geq 6, \quad s = -1.$$

Das Literal x_1 wurde eliminiert und der abgeleitete Constraint ω_4 ist im Konfliktlevel unerfüllbar. Bei einem Backtrack würde ω_4 jedoch keine Literale propagieren. Der LPB-Constraint ω_4 ist somit kein Kandidat, um gelernt zu werden. Es müssen weitere Literale eliminiert werden, die im Konfliktlevel gesetzt wurden:

$$\begin{aligned}(3\omega_1 + \omega_4) = \omega_5 : &\quad x_2 + 2x_7 + 4\overline{x_8} + 3x_5 + 3x_6 \geq 6, \quad s = -1 \\ (\omega_1 + \omega_5) = \omega_6 : &\quad \overline{x_3} + 2x_7 + 4\overline{x_8} + 3x_5 + 4x_6 \geq 7, \quad s = -1\end{aligned}$$

Mit ω_6 wurde ein Constraint abgeleitet, der im Konfliktlevel unerfüllbar ist und beim Backtracking in das erste Entscheidungslevel das Literal x_6 propagiert.

Um das beispielhaft beschriebene Lernen von LPB-Constraints allgemeingültig zu formulieren wird die bereits im obigen Beispiel verwendete Abschwächung von LPB-Constraints definiert.

Definition 13 (Abschwächung)
Mit *Abschwächung* wird die Ableitungsregel

$$\frac{\begin{array}{c}\sum_i a_i l_i \geq b \\ a_k > 0\end{array}}{\sum_{i \neq k} a_i l_i \geq b - a_k} \qquad (\textit{Abschwächung})$$

bezeichnet.

Abschwächung ist eine Anwendung von verallgemeinerter Resolution: Der Constraint $\sum_i a_i l_i \geq b$ wird mit der Tautologie $-a_k l_k \geq -a_k$ kombiniert. In der Konfliktanalyse ist Abschwächung jedoch nur für Literale $l_k \neq 0$ anzuwenden. Wäre bei der Abschwächung von ω_2 im obigen Beispiel das Literal x_3 entfernt worden, so hätten ω_2' mit $s = 3$ und ω_2'' mit $s = 2$ zu große Slacks.

Zusammenfassend ist eine verallgemeinerte Konfliktanalyse für LPB-Constraints wie folgt durchführbar:

1. C sei der originale Konflikt-Constraint.

2. Nimm den Constraint R, der den Grund für die letzte Zuweisung eines Literals $l \in C$ darstellt.

3. Schwäche R so ab, dass beim Eliminieren von l die Kombination $\alpha R + \beta C$ nicht erfüllt ist (Abschwächung und Sättigung).

4. Ersetze C durch $\alpha R + \beta C$

5. Wiederhole 2. bis 4. solange, bis C in einem höheren Level ein Literal propagiert.

Im nächsten Abschnitt werden numerische Ergebnisse der Produktkonfiguration mit pseudo-boolescher Optimierung zusammengestellt. Im Zusammenhang mit der DPLL-basierten linearen Suche wird dabei unter anderem das Lernen von Klauseln mit dem Lernen von Cutting Planes verglichen.

3.5 Numerische Ergebnisse

Die Produktkonfiguration mittels pseudo-boolescher Optimierung wurde im Rahmen dieser Arbeit anhand von randomisierten Tests durchgeführt. Für eine Testreihe wurden zu einer festen, spezialisierten Produktübersicht zufällige Zielfunktionen generiert. Um eine Zielfunktion zu bestimmen, wurden n kundenwählbare Variablen (Variablen der Kategorie (A1), vgl. Unterabschnitt 2.2.4) zufällig ausgewählt, die noch nicht durch die Spezialisierung der Produktübersichtsformel belegt sind. Den ausgewählten Variablen v_i wurde ein zufälliger, ganzzahliger Koeffizient a_i aus dem Zahlenbereich $[-10.000.000, \ldots, 10.000.000]$ zugeordnet. Für die Variablenanzahlen n mit $n = 10, 20, \ldots, 200$ wurden auf diese Weise jeweils 1000 Zielfunktionen $\sum a_i v_i$ generiert.
Entsprechend wurden Testreihen für die 13 verschiedenen BM7/Markt-Segmente aus Tabelle 5 erstellt. Zusätzlich wurden 9 Testreihen auf der Ebene Typklasse/Markt generiert. In diesen Testreihen waren die Typklassen S205@BM, V205@BM, W205@BM, S212@BM, V212@BM und W212@BM für das Land 374@LD (Deutschland) und die Typklassen S205@BM, V205@BM und V212@BM für das Land 839@BM (China) vertreten.

Die Instanzen der erstellten Testreihen wurden mit dem ILP-Standardsolver CPLEX [CPL15] und mit der auf DPLL basierenden linearen Suche berechnet.
Als Implementierung von DPLL wurde Sat4j verwendet – eine JAVA-Version von MiniSAT, die den Umgang mit LPB-Constraints gemäß der Unterabschnitte 3.4.1 und 3.4.2 beherrscht [LBP10].
Die Auswahl der nächsten Belegung im DPLL-Entscheidungsprozess erfolgte mittels der VSIDS-Heuristik (Variable State Independent, Decaying Sum) aus MiniSAT [SE03]. Für die Konfliktanalyse wurde sowohl das Lernen von Klauseln als auch das Lernen von Cutting Planes angewandt (vgl. Unterabschnitt 3.4.2). Ein Vergleich beider Lernmethoden ist in Abbildung 7 zu sehen. In der linken Abbildung ist die durchschnittliche Anzahl von Konflikten und in der rechten Abbildung die durchschnittlich benötigte

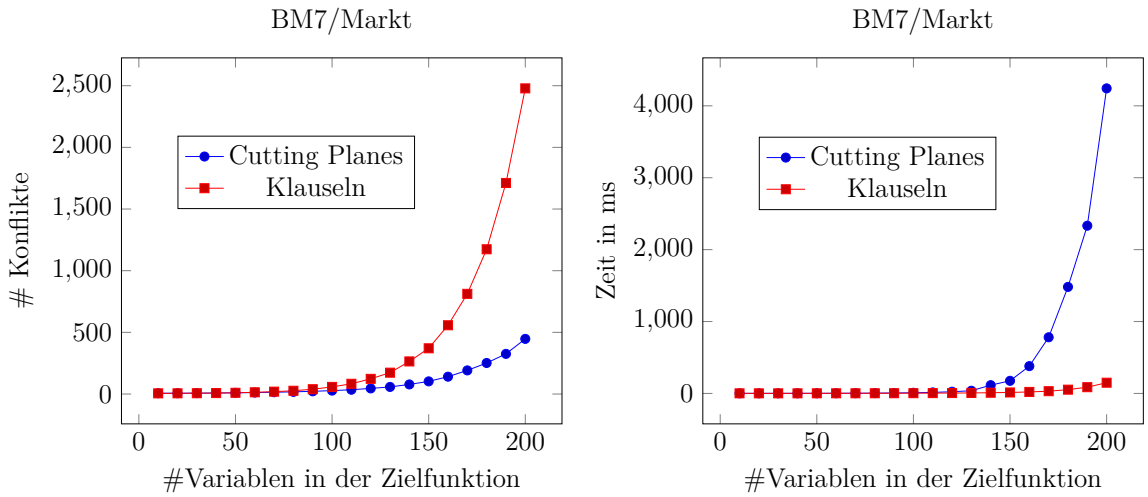

Abbildung 7: Vergleich der DPLL-Lernmethoden bzgl. Konfliktdichte und Laufzeit

Zeit zum Lösen einer Instanz dargestellt – jeweils in Abhängigkeit von der Anzahl der Variablen in der Zielfunktion. Die Berechnung der Durchschnitte erfolgte hierbei über alle Instanzen aller Testreihen der Ebene BM7/Markt.

Es ist zu erkennen, dass beim Lernen von Cutting Planes insgesamt weniger Konflikte auftreten. Dies ist insbesondere mit der stärkeren Aussagekraft der Cutting Planes zu erklären, die im DPLL-Suchprozess eine größere Anzahl von Variablen propagieren.

In der rechten Abbildung zeigt sich jedoch, dass der Einsatz von Cutting Planes zum Lösen der vorliegenden PBO-Instanzen im Vergleich zum Lernen von Klauseln zu keinen kürzeren Laufzeiten führt. Im Falle des Klausel-Lernens können die Instanzen mit längeren Zielfunktionen sogar deutlich schneller gelöst werden.

Damit ist beim Einsatz von Cutting Planes der Vorteil der geringeren Konfliktdichte nicht ausreichend, um die sonstigen Nachteile auszugleichen. Nachteilig wirkt sich insbesondere die aufwändigere Beobachtung von Literalen und die damit verbundene ineffizientere Propagation der Variablenbelegungen aus.

Die Ergebnissen aus Abbildung 8 zeigen, dass ein wesentlicher Einfluss der unterschiedlichen Lernmethoden auf die Anzahl der Iterationen bei der linearen Suche ausgeschlossen ist. Die gemittelte Anzahl der Iterationen fällt bei beiden Lernmethoden unerwartet gering aus und belegt, dass die vorliegenden PBO-Instanzen besonders geeignet sind, um mit linearer Suche berechnet zu werden.

In der weiteren Anwendung der DPLL-basierten, linearen Suche wurde aufgrund der deutlich kürzeren Laufzeiten ausschließlich auf das Lernen von Klauseln gesetzt. Eine detaillierte Darstellung der ermittelten Laufzeiten für das Lernen von Klauseln ist in Abbildung 9 zu sehen. Hier sind die Ergebnisse der unterschiedlichen BM7/Markt-Segmente nicht akkumuliert; die y-Achse ist logarithmisch. Die Legende ist entsprechend der bei $n = 200$ ermittelten Laufzeiten sortiert. Eine große Varianz ist insbesondere zwischen den Laufzeitergebnissen der unterschiedlichen BM7/Markt-Segmente

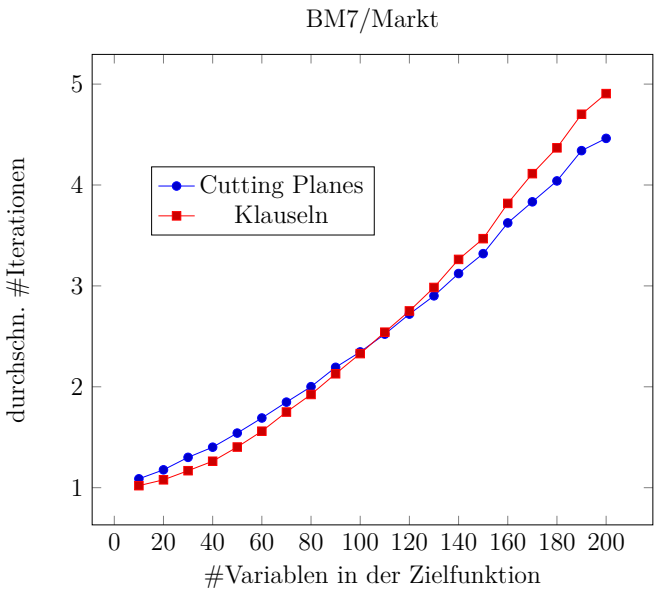

Abbildung 8: Iterationen beim Lösen von PBO mit DPLL-basierter, linearer Suche

zu sehen. Ein nichtlineares Wachstum der Laufzeit bei steigender Anzahl der Variablen in der Zielfunktion liegt für alle BM7/Markt-Segmente vor.

Des Weiteren wurde die DPLL-basierte lineare Suche (Sat4j) mit der Branch-and-Cut-Implementierung von CPLEX verglichen. In der Abbildung 10 sind links die Laufzeitergebnisse bezogen auf die Granularität BM7/Markt und rechts die Laufzeitergebnisse bezogen auf die Granularität Typklasse/Markt dargestellt. Nach Unterabschnitt 2.2.6 umfassen Produktübersichtsformeln, die den Testreihen der linken Abbildung zugrunde liegen, zwischen 2.000 und 20.000 Klauseln. Dem stehen in der rechten Abbildung Klauselanzahlen zwischen 20.000 und 150.000 gegenüber.
In der linken Abbildung ist zu erkennen, dass bezüglich der BM7/Markt-Granularität die lineare Suche mit DPLL zu wesentlich kürzeren Laufzeiten führt, sofern die Zielfunktionen weniger als 170 Variablen umfassen. Für längere Zielfunktionen kommt es bei Anwendung der linearen Suche zu einem steilen Anstieg der Laufzeit. Die Laufzeiten für CPLEX nehmen mit steigender Variablenanzahl in den Zielfunktionen hingegen nur schwach zu, so dass ab einer gewissen Länge der Zielfunktion die Anwendung moderner Branch-and-Cut-Implementierungen vorteilhaft ist.
In der rechten Abbildung ist wiederum zu erkennen, dass bei steigender Anzahl von Klauseln in der Produktübersichtsformel die Anwendung von CPLEX nicht zu besseren Laufzeiten bei Zielfunktionen mit bis zu 200 Variablen führt.
Der Vorteil von DPLL-basierten Solvern wie Sat4j ist es, mit einer schlanken Implementierung sehr effizient Modelle einer Produktübersichtsformel berechnen zu können. Dieser Vorteil gegenüber CPLEX geht verloren, sobald die 0/1-Optimierungsaufgaben durch länger werdende Zielfunktionen komplexer werden.

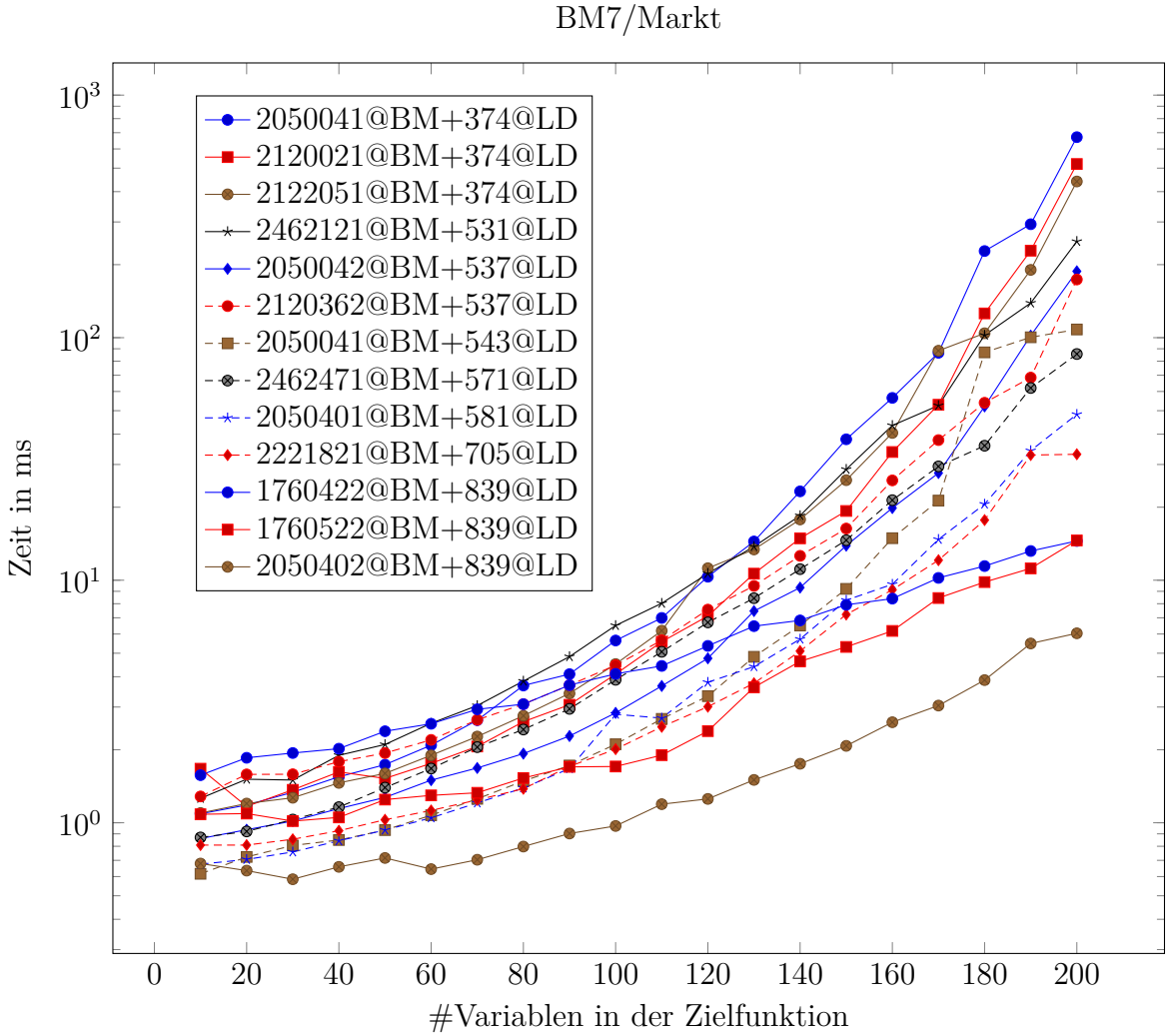

Abbildung 9: Laufzeiten der linearen Suche für verschiedene BM7/Markt-Segmente

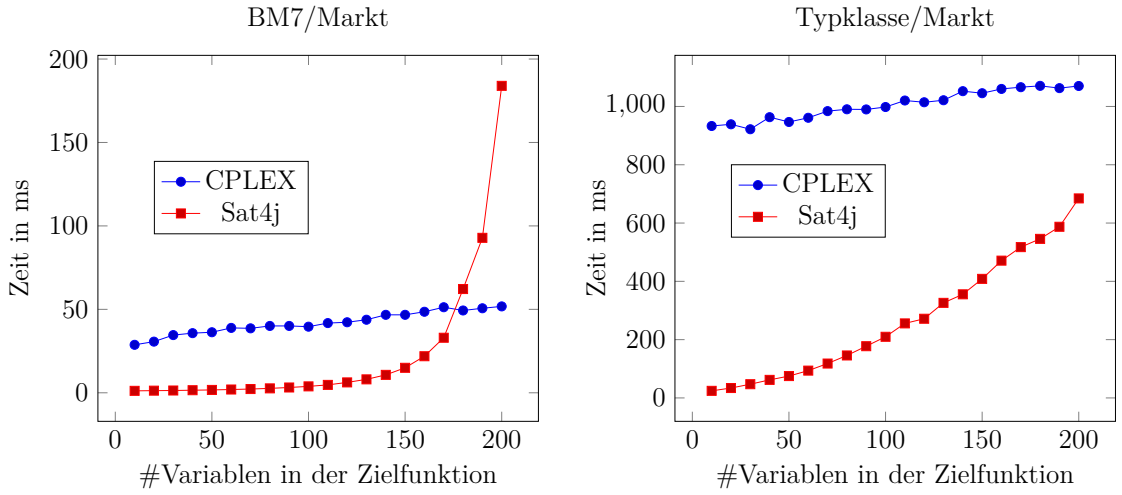

Abbildung 10: Vergleich zwischen Branch and Cut und DPLL-basierter, linearen Suche

4 Konsistente Prognose und induzierte Verbauratenintervalle

In Kapitel 2 wurde beispielhaft dargestellt, wie eine Produktübersicht in eine aussagenlogische Formel übersetzt werden kann. Anschließend wurde in Kapitel 3 erläutert, wie auf Basis einer Produktübersichtsformel optimale Varianten berechnet werden können. Das Berechnen optimaler Varianten ist wiederum ein unverzichtbarer Baustein für die Konsistenz- und Intervallrechnung, die Gegenstand dieses Kapitels ist.
Mit der Konsistenzrechnung wird die Frage beantwortet, ob ein Programm (eine Menge) von Fahrzeugvarianten zusammengestellt werden kann, das bestimmte Verbauquoten erfüllt. Die Generierung von synthetisierten Auftragsprogrammen nach vorgegebenen statistischen Features findet beispielsweise in der Bedarfsprognose Anwendung [SR13]. Dort werden die Teilequoten synthetisierter Auftragsprogramme ausgezählt und als Schätzer für die Bedarfsprognose verwendet. Die auf diese Weise ermittelten Schätzer sind im Allgemeinen jedoch nicht eindeutig, da das Auftragsprogramm, das die vorgegeben Verbauquoten erfüllt, nicht eindeutig ist. Die Aufgabe der Intervallrechnung ist es zu evaluieren, in welchem Bereich die ausgezählten Teilebedarfe liegen können.

4.1 Mathematische Problembeschreibung

Zunächst folgen die notwendigen Begriffsdefinitionen der Konsistenz- und Intervallrechnung. Ihnen sei jeweils eine Produktübersichtsformel POF zugrunde gelegt.

Definition 14 (zulässige Variante)
Eine Variante (Variablenbelegung) $v : C \to \{0,1\}$, $C \subseteq Var(POF)$ wird als zulässig bezeichnet, falls eine Erweiterung von v auf $Var(POF)$ existiert, unter der die Produktübersichtsformel POF zu 1 evaluiert. Die endliche Menge aller *zulässigen Variablenbelegungen* auf C wird mit $\mathbb{V}_C(POF)$ bezeichnet. Außerdem wird $\mathbb{V}(POF) := \mathbb{V}_{Var(POF)}(POF)$ definiert.

Für den Begriff der „Variante“ werden auch die Begriffe „Konfiguration“ und „Auftrag“ verwendet. Der Begriff „baubar“ wird synonym zu „zulässig“ verwendet. Wie bereits im Kapitel 1 angemerkt wurde, existieren für die aktuelle Mercedes-Benz E-Klasse mehr als 10^{50} zulässige Varianten.

Definition 15 (Variantenverteilung)
Eine *Variantenverteilung* x entspricht einer Abbildung $x : \mathbb{V}_C(POF) \longrightarrow [0,1]$, die jeder zulässigen Variante $v \in \mathbb{V}_C(POF)$ eine Häufigkeit $x(v) \geq 0$ zuordnet und $\sum x(v) = 1$ erfüllt.

Alle Variablen $Var(POF)$ der Produktübersichtsformel POF werden im Weiteren *Codes* genannt. Somit ist es möglich, nicht nur den Verbau von kundenwählbaren Optionen, sondern auch den Verbau von Positionsvarianten durch Codes auszudrücken. Die Einbauregel ϕ_t von einer Positionsvariante t kann durch einen Hilfscode c_t repräsentiert werden. Dazu ist die Produktübersichtsformel um den entsprechenden Constraint $c_t \leftrightarrow \phi_t$ zu erweitern (vgl. Abschnitt 1.1). Anzumerken ist, das aus planerischer Sicht die Verbauraten der tatsächlichen Bauteile und nicht die der Positionsvarianten entscheidend sind. Das Problem, dass ein Bauteil möglicherweise an mehreren Positionsvarianten verbaut werden kann, wird jedoch von dem in diesem Kapitel vorgestellten Verfahren der Konsistenz- und Intervallrechnung beherrscht.
Diese Erweiterung der Semantik des Begriffs „Code“ ermöglicht eine vereinfachte Darstellung der Konsistenz- und Intervallrechnung, ohne dabei die mathematische Modellierung einzuschränken.

Definition 16 (Code-Statistik)
Es sei $C \subseteq Var(POF)$. Eine *Code-Statistik* auf C ist eine Abbildung $s : C \longrightarrow [0, 1]$, die jedem Code $c \in C$ eine Häufigkeit $s(c)$ zuordnet.
Zu einer Variantenverteilung x wird die Code-Statistik

$$s_x(c) := \sum_{v \in \mathbb{V}_C(POF), v(c)=1} x(v), \quad \forall c \in C$$

definiert. Sie beschreibt die Häufigkeit mit der ein Code $c \in C$ in der Verteilung x vorkommt und wird als die durch x *induzierte Code-Statistik* auf C bezeichnet.

Definition 17 (Code-Intervalle)
Es sei $C \subseteq Var(POF)$. Mit l und u seien zwei Code-Statistiken auf C gegeben, die $l \leq u$ erfüllen, d. h. für alle $c \in C$ gilt $l(c) \leq u(c)$. Dann sind durch das Paar (l, u) *Code-Intervalle* auf C definiert.

Definition 18 (Konsistenz)
Eine Code-Statistik s auf $C \subseteq Var(POF)$ ist *konsistent*, falls eine Variantenverteilung x existiert, sodass s identisch mit der induzierten Statistik s_x auf C ist. Die Menge der konsistenten Code-Statistiken auf C unter einer Produktübersichtsformel POF sei durch $\mathcal{S}_C(POF)$ gegeben.
Code-Intervalle (l, u) auf $C \subseteq Var(POF)$ sind *konsistent*, falls eine konsistente Code-Statistik s auf C existiert, sodass $l \leq s \leq u$ gilt. Die Menge der konsistenten Code-Intervalle auf C unter einer Produktübersichtsformel POF sei durch $\mathcal{I}_C(POF)$ gegeben.

Eine Code-Statistik s ist genau dann konsistent, wenn die Code-Intervalle (s, s) konsistent sind, d. h. $s \in \mathcal{S}_C(POF) \Leftrightarrow (s, s) \in \mathcal{I}_C(POF)$.

Definition 19 (Konsistenzrechnung)
Das *Konsistenzproblem* besteht darin, zu entscheiden, ob gegebene Code-Intervalle unter einer Produktübersichtsformel konsistent sind.
Mit dem Begriff *Konsistenzrechnung* wird das algorithmische Lösen des Konsistenzproblems bezeichnet.

Beobachtung 1
Es sei eine Produktübersichtsformel POF und eine Codemenge $C \subseteq Var(POF)$ gegeben. Darüber hinaus wird eine beliebige Produktübersichtsformel $POF' \models POF$ und eine beliebige Codeteilmenge $C' \subseteq C$ betrachtet. Aus der Definition von Konsistenz ergeben sich die folgenden Zusammenhänge:

1. $S_C(POF') \subseteq S_C(POF)$ und $\mathcal{I}_C(POF') \subseteq \mathcal{I}_C(POF)$, d. h. bei zusätzlichen Constraints in der Produktübersichtformel nimmt die Menge konsistenter Statistiken bzw. konsistenter Intervalle ab.

2. $s \in S_C(POF) \Rightarrow s_{|C'} \in S_{C'}(POF)$ und
   $(l, u) \in \mathcal{I}_C(POF) \Rightarrow (l_{|C'}, u_{|C'}) \in \mathcal{I}_{C'}(POF)$, d. h. Konsistenz bleibt bei Reduzierung der Vorgaben erhalten.

3. $s' \in S_{C'}(POF) \Rightarrow \exists s \in S_C(POF) : \; s_{|C'} = s'$ und
   $(l', u') \in \mathcal{I}_{C'}(POF) \Rightarrow \exists (l, u) \in \mathcal{I}_C(POF) : \; l_{|C'} = l', \; u_{|C'} = u'$, d. h. Konsistenz auf einer Teilmenge von Codes lässt sich für eine Obermenge von Codes immer ergänzen.

4. $(l, u) \in \mathcal{I}_C(POF), \; l' \leq l, \; u' \geq u \Rightarrow (l', u') \in \mathcal{I}_C(POF)$, d. h. konsistente Code-Intervalle können beliebig aufgeweitet werden, ohne dabei inkonsistente Code-Intervalle zu erzeugen

Im vierten Zusammenhang aus Beobachtung 1 kann die Menge der konsistenten Code-Statistiken, die innerhalb der aufgeweiteten Code-Intervalle (l', u') liegen, eine echte Obermenge gegenüber der Menge der konsistenten Code-Statistiken darstellen, die innerhalb der ursprünglichen Code-Intervalle (l, u) liegen.
In der Praxis stellt sich jedoch die Frage, ob kleinere Code-Intervalle dieselbe Menge konsistenter Code-Statistiken repräsentieren können: Sind konsistente Code-Intervalle (l, u) gegeben, so ist zu klären, inwieweit diese zu $(l', u'), \; l' \geq l, \; u' \leq u$ eingeschränkt werden können, ohne dabei konsistente Code-Statistiken „wegzuschneiden“. Das verlustfreie Einschränken von Code-Intervallen führt zu Informationsgewinn.

Beispiel 8
Es wird die Produktübersicht betrachtet, die durch die Formel

$$POF = (c_1 \vee \neg c_2) \wedge (\neg c_1 \vee c_2)$$

gegeben ist. Sie definiert mit

$$\begin{aligned} \beta_1(c_1) = 0, \quad \beta_1(c_2) = 0 \\ \beta_2(c_1) = 1, \quad \beta_2(c_2) = 1 \end{aligned}$$

genau zwei zulässige Varianten. Ein Code-Verbauquotenplaner, welcher die Zusammenhänge der Produktübersicht nicht vollständig überblickt, plant mit

$$\begin{aligned} l(c_1) := 30\%, \quad l(c_2) := 40\% \\ u(c_1) := 50\%, \quad u(c_2) := 60\% \end{aligned}$$

die Code-Verbauquotenintervalle (l, u). Mit Intervallgrößen von jeweils 20% scheint der Planer den Codes c_1 und c_2 relativ große Schwankungsbreiten zuzugestehen. Tatsächlich jedoch enthalten die Code-Intervalle (l, u) nur konsistente Code-Statistiken s mit

$$40\% \leq s(c_1), s(c_2) \leq 50\%.$$

Dies ist dadurch begründet, dass für die zwei zulässigen Varianten β_1 und β_2 nur Code-Statistiken s mit $s(c_1) = s(c_2)$ konsistent sind. Die Code-Intervalle (l^*, u^*) mit

$$\begin{aligned} l^*(c_1) := 40\%, \quad l^*(c_2) := 40\%, \\ u^*(c_1) := 50\%, \quad u^*(c_2) := 50\%, \end{aligned}$$

sind bezüglich ihrer restriktiven Aussagekraft – unter der Produktübersichtsformel POF – äquivalent zu (l, u). Sie stellen jedoch die tatsächlich möglichen Schwankungsbreiten von jeweils 10% dar.

Definition 20 (induzierte Code-Intervalle)
Mit $I = (l, u)$ seien konsistente Code-Intervalle auf $C \subseteq Var(POF)$ gegeben. Das von den Code-Intervallen I unter der Produktübersichtsformel POF *induzierte Code-Intervall* $[L^*(c), U^*(c)]$ für einen Code $c \in Var(POF)$ wird durch

$$\begin{aligned} L^*(c) &:= \min \left\{ \tilde{s}(c) \;\middle|\; \tilde{s} \in \mathcal{S}_{C \cup \{c\}}(POF),\ l \leq \tilde{s}_{|C} \leq u \right\} \\ U^*(c) &:= \max \left\{ \tilde{s}(c) \;\middle|\; \tilde{s} \in \mathcal{S}_{C \cup \{c\}}(POF),\ l \leq \tilde{s}_{|C} \leq u \right\} \end{aligned}$$

definiert. Die Gesamtheit aller induzierten Code-Intervalle auf $Var(POF)$ wird mit $I^* := (L^*, U^*)$ bezeichnet. Für eine Teilmenge $C' \subseteq Var(POF)$ seien zudem die Code-Intervalle (l^*, u^*) auf C' durch $l^* := L^*_{|C'}$ und $u^* := U^*_{|C'}$ definiert. Mit der Notation

$$(l, u) \xrightarrow[C']{POF} (l^*, u^*)$$

wird ausgedrückt, dass (l^*, u^*) die Code-Intervalle auf C' darstellt, die unter POF von (l, u) induziert werden.

Induzierte Intervalle werden wie in Kapitel 1 auch als Minimalintervalle bezeichnet.

Beobachtung 2
Es seien Code-Intervalle (l, u), (l^*, u^*) und $(\tilde{l}, \tilde{u})$ auf $C \subseteq Var(POF)$ mit

$$(l, u) \xrightarrow[C]{POF} (l^*, u^*)$$

gegeben. Dann gelten die folgenden Zusammenhänge:

1. $l \leq l^*,\ u \geq u^*$, d. h. die induzierten Intervalle sind in den Ausgangsintervallen enthalten.

2. $\left\{s \;\middle|\; s \in \mathcal{S}_C(POF),\ l \leq s \leq u\right\} = \left\{s \;\middle|\; s \in \mathcal{S}_C(POF),\ l^* \leq s \leq u^*\right\}$, d. h. die induzierten Intervalle verändern nicht die Menge der enthaltenen konsistenten Code-Statistiken.

3. $l \leq \tilde{l},\ u \geq \tilde{u},\ \exists c \in C:\ l^*(c) < \tilde{l}(c)$ oder $u^*(c) > \tilde{u}(c)$
   $\Rightarrow \left\{s \;\middle|\; s \in \mathcal{S}_C(POF),\ \tilde{l} \leq s \leq \tilde{u}\right\} \subsetneq \left\{s \;\middle|\; s \in \mathcal{S}_C(POF),\ l \leq s \leq u\right\}$, d. h. durch das weitere Einschränken von induzierten Code-Intervallen werden konsistente Code-Statistiken „weggeschnitten“.

Definition 21 (Intervallrechnung)
Die Berechnung von induzierten Intervallen gemäß Definition 20 wird als *Intervallrechnung* bezeichnet.

Beispiel 9
Es wird die Produktübersichtsformel

$$POF = (c_1 \vee \neg c_2) \wedge (\neg c_1 \vee c_2) \wedge (\neg c_2 \vee \neg c_3)$$

betrachtet, die insgesamt 3 zulässige Varianten definiert. Für die Codes in $C = \{c_2, c_3\}$ seien mit den auf C definierten Code-Intervallen (l, u),

$$l(c_2) := 20\%, \quad l(c_3) := 50\%,$$
$$u(c_2) := 30\%, \quad u(c_3) := 60\%,$$

intervallbasierte Prognosen gegeben. Eine wichtige Anwendung der Intervallrechnung besteht darin, die Auswirkungen einer derartigen Code-Prognose auf den Teilebedarf zu bestimmen. Beispielsweise sei das Bauteil T mit der Teileauswahlregel $\phi_T = (c_1 \vee c_3)$ gegeben, d. h. das Teil T wird genau dann in einer Variante verbaut, wenn diese ϕ_T erfüllt. Um in der Intervallrechnung das Teil T mit einzubeziehen wird die Produktübersichtformel um einen Code c_T erweitert, der genau in den Varianten mit dem Wahrheitswert 1 belegt wird, in denen das Teil T verbaut wird:

$$POF' = POF \wedge (c_T \leftrightarrow \phi_T).$$

Ist (l^*, u^*) durch

$$(l, u) \xrightarrow[C']{POF'} (l^*, u^*), \; C' = \{c_t\}$$

gegeben, so liegt die Verbauquote von Bauteil T im Intervall

$$I_T^* = [l^*(c_T), u^*(c_T)] = [70\%, 90\%],$$

sofern die Prognosen (l, u) eintreten. Darüber hinaus existiert zu jeder Intervallgrenze von I_T^* eine konsistente Code-Statistik, die die Intervallgrenze trifft und innerhalb der Prognose liegt.

Um mit Hilfe der Konsistenz- bzw. Intervallrechnung Code-Intervalle unter einer Produktübersicht evaluieren zu können, werden innerhalb der weiteren Abschnitte dieses Kapitels entsprechende Berechnungsverfahren vorgestellt.
Mit dem folgenden Satz wird begründet, dass es vom Standpunkt der Algorithmik ausreichend ist, Verfahren zu entwickeln, die anstelle der gegebenen Code-Intervalle generierte, abgeleitete Code-Statistiken evaluieren.

Satz 4
Das Problem der Konsistenz von Code-Intervallen kann auf das Problem der Konsistenz einer Code-Statistik reduziert werden.

Beweis. Für ein beliebiges Tupel $\big(POF, (l, u)\big)$, bestehend aus einer Produktübersichtsformel POF und Code-Intervallen (l, u) auf einer Menge $C \subseteq Var(POF)$, ist eine Übersetzung $\big(POF', s'\big)$ so anzugeben, dass

1. s' eine Code-Statistik auf einer Menge $C' \subseteq Var(POF')$ darstellt,
2. (l, u) unter POF genau dann konsistent ist, wenn s' unter POF' konsistent ist.

Im ersten Schritt wird die Übersetzung angegeben, im zweiten Schritt wird die Äquivalenz in der Konsistenz gezeigt.

1. Es werden aussagenlogische Variablen c_l, c_u für alle $c \in C$ eingeführt und es wird

$$POF' := POF \wedge \Big(\bigwedge_{c \in C} (c_l \to c) \Big) \wedge \Big(\bigwedge_{c \in C} (c \to c_u) \Big)$$

definiert. Für $L := \{c_l \mid c \in C\},\ U := \{c_u \mid c \in C\}$ und $C' := L \cup U$ sei

$$s'(c) := \begin{cases} l(c) & c \in L \\ u(c) & c \in U \end{cases}, \qquad \forall c \in C'.$$

2. Zunächst folgen einige Beobachtungen. Jede zulässige Variante $\beta \in \mathbb{V}(POF)$ stellt bezüglich POF' eine erfüllbare Teilbelegung dar. Ein $\beta \in \mathbb{V}(POF)$ kann zu einem $\beta' \in \mathbb{V}(POF')$ erweitert werden. Für alle $c \in C$ gilt:

(a.1) $\beta(c) = \beta'(c) = 0$, so ist $\beta'(c_l) = 0$ erzwungen

(b.1) $\beta(c) = \beta'(c) = 1$, so ist $\beta'(c_l)$ beliebig

(a.2) $\beta(c) = \beta'(c) = 1$, so ist $\beta'(c_u) = 1$ erzwungen

(b.2) $\beta(c) = \beta'(c) = 0$, so ist $\beta'(c_u)$ beliebig

Es sei nun (l, u) auf C konsistent. Es ist zu zeigen, dass s' auf C' konsistent ist. Dazu sei mit x eine Verteilung auf $\mathbb{V}_C(POF)$ gegeben, so dass die auf C induzierte Code-Statistik s_x die Ungleichungen $l \leq s_x \leq u$ erfüllt. Es werden die von s_x induzierten Intervalle (l^*, u^*) auf C' betrachtet, d. h.

$$(s_x, s_x) \xrightarrow[C']{POF'} (l^*, u^*).$$

Aus (a.1) und (b.1) folgt

$$l^*(c_l) = 0 \text{ und } u^*(c_l) = s_x(c) \geq l(c)$$

für $c \in C$ und zugehöriges $c_l \in L \subseteq C'$.
Analog folgt aus (a.2) und (b.2)

$$l^*(c_u) = s_x(c) \leq u(c) \text{ und } u^*(c_u) = 1$$

für $c \in C$ und zugehöriges $c_u \in U \subseteq C'$.
Aufgrund der durch (b.1) und (b.2) gegebenen Freiheiten ist sogar jede Code-Statistik s mit $l^* \leq s \leq u^*$ konsistent, also auch $s := s'$.

Für die Gegenrichtung sei s' auf C' konsistent. In diesem Fall werden die von s' auf C induzierten Intervalle (l^*, u^*) betrachtet, d. h.

$$(s', s') \xrightarrow[C]{POF'} (l^*, u^*).$$

Wegen (a.1) gilt

$$l(c) = s'(c_l) \leq l^*(c)$$

für $c \in C$ und zugehöriges $c_l \in L \subseteq C'$ und wegen (a.2) gilt

$$u(c) = s'(c_u) \geq u^*(c)$$

für $c \in C$ und zugehöriges $c_u \in L \subseteq C'$.
Da (l^*, u^*) konsistent auf C ist, ist auch (l, u) wegen $l \leq l^*$ und $u \geq u^*$ auf C konsistent. Ist (l, u) unter POF' auf C konsistent, so ist (l, u) auch unter POF auf C konsistent (vgl. Beobachtung 1).

□

4.2 PSAT – Probabilistische Erfüllbarkeit

Mit dem probabilistischen Erfüllbarkeitsproblem PSAT und den diesbezüglich eingesetzten Lösungstechniken stellt die Literatur nahezu maßgeschneiderte Werkzeuge bereit, um Verfahren für die im vorangegangenen Abschnitt definierte Konsistenz- und Intervallrechnung zu entwickeln. Die Grundlagen der linearen Algebra, die in diesem Zusammenhang unverzichtbares Basiswissen darstellen, können im Anhang nachgelesen werden.

4.2.1 Problemdefinitionen aus der Literatur

Das *probabilistische Erfüllbarkeitsproblem* (PSAT) wird in der Literatur wie folgt formuliert [Nil86, GKP88, HJA+00]:

Es seien aussagenlogische Formeln

$$F_1, \ldots, F_m \text{ mit } Var(F_1), \ldots, Var(F_m) \subseteq \{c_1, \ldots, c_n\}$$

gegeben, denen Wahrscheinlichkeiten $p_1, \ldots, p_m \in [0, 1]$ dafür zugeordnet sind, dass sie zum Wahrheitswert 1 evaluieren. Frage: Sind diese Wahrscheinlichkeiten konsistent?

Die Wahrscheinlichkeit $x(F)$, dass eine Formel F zu 1 evaluiert, wird über eine Wahrscheinlichkeitsverteilung x auf den 2^n verschiedenen Variablenbelegungen β induziert:

$$x(F) := \sum_{\beta : F(\beta) = 1} x(\beta).$$

Gegebene Wahrscheinlichkeiten $p_1, \ldots, p_m$ sind demnach *konsistent*, falls eine Wahrscheinlichkeitsverteilung x auf den Belegungen existiert, die

$$x(F_i) = p_i,\ i = 1, \ldots, m$$

erfüllt.

Beispiel 10
Die Wahrscheinlichkeiten $p_1 = 0,9,\ p_2 = 0,7,\ p_3 = 0,7$ zu

$$F_1 = \neg c_1 \vee \neg c_2,\ F_2 = c_1,\ F_3 = c_2$$

sind inkonsistent: Die 4 Belegungen von c_1 und c_2 sind durch

$$\begin{aligned}
\beta_0(c_1) &:= 0,\ \beta_0(c_2) := 0 \\
\beta_1(c_1) &:= 0,\ \beta_1(c_2) := 1 \\
\beta_2(c_1) &:= 1,\ \beta_2(c_2) := 0 \\
\beta_3(c_1) &:= 1,\ \beta_3(c_2) := 1
\end{aligned}$$

gegeben. Für eine Wahrscheinlichkeitsverteilung x der 4 Belegungen, die $x(F_i) = p_i$ für $i = 1, 2, 3$ induziert, muss

$$\begin{array}{ccccccccc}
x(\beta_0) & + & x(\beta_1) & + & x(\beta_2) & + & x(\beta_3) & = & 1 \\
x(\beta_0) & + & x(\beta_1) & + & x(\beta_2) & & & = & 0,9 \\
 & & x(\beta_1) & & & + & x(\beta_3) & = & 0,7 \\
 & & & & x(\beta_2) & + & x(\beta_3) & = & 0,7
\end{array}$$

gelten, wobei die erste Gleichung eine Grundvoraussetzung an eine Wahrscheinlichkeitsverteilung darstellt: Die Vereinigung aller Elementarereignisse besitzt die Wahrscheinlichkeit 1. Inkonsistenz liegt vor, da das Gleichungssystem die unzulässige Elementarwahrscheinlichkeit $x(\beta_0) = -0,3 < 0$ induziert.

Für die in der Literatur [HJA+00] übliche Darstellung von PSAT wird die Matrix $(a_{ij}) = A \in \{0,1\}^{m \times 2^n}$ mit

$$a_{ij} := F_i(\beta_j)$$

definiert. PSAT ist somit das Problem, zu entscheiden, ob ein Vektor $x \in \mathbb{R}^{2^n}$ mit

$$\begin{aligned}
\mathbf{1}^t x &= 1 \\
Ax &= p \\
x &\geq \mathbf{0}
\end{aligned} \tag{34}$$

existiert, wobei p dem Vektor $(p_1, \ldots, p_m)^t$ entspricht.

Eine natürliche Verallgemeinerung von PSAT ist die Frage nach der Konsistenz von Wahrscheinlichkeitsintervallen und damit die Frage nach der Existenz eines Vektors $x \in \mathbb{R}^{2^n}$, der

$$\begin{array}{rcl} \mathbf{1}^t x & = & 1 \\ l \leq \; Ax & & \leq u \\ x & \geq & \mathbf{0} \end{array} \tag{35}$$

erfüllt. Die Vektoren $l, u \in \mathbb{R}^m$ repräsentieren in diesem Zusammenhang die Intervallgrenzen l_i, u_i der Wahrscheinlichkeiten $x(F_i)$.

Bei vorliegenden konsistenten Wahrscheinlichkeitsintervallen (l, u) für aussagenlogische Formeln $F_1, \ldots, F_m$ ergibt sich als anschließende Problemstellung die Berechnung des induzierten Intervalls für eine weitere aussagenlogische Formel F_{m+1}, $Var(F_{m+1}) \subset \{x_1, \ldots, x_n\}$. Dieses Problem wird in der Literatur üblicherweise als *probabilistic entailment* bezeichnet [HJA+00]. Für die Formel F_{m+1} sei der zugehörige Zeilenvektor $a^t_{m+1}\mathbb{R}^{2^n}$ gemäß der Definition von A gegeben, d. h. $a_{m+1,j} = F_{m+1}(\beta_j)$. Die linearen Programme

$$\begin{array}{lrcl} & \min / \max & a^t_{m+1} x & \\ \text{s.th.} & & & \\ & \mathbf{1}^t x & = & 1, \\ & l \leq \; Ax & & \leq u, \\ & x & \geq & \mathbf{0} \end{array} \tag{36}$$

formulieren die induzierte minimale Wahrscheinlichkeit und die induzierte maximale Wahrscheinlichkeit dafür, dass F_{m+1} zum Wahrheitswert 1 evaluiert.

4.2.2 Konsistenz- und Intervallrechnung als Spezialfall von PSAT

In diesem Unterabschnitt wird zunächst der Zusammenhang zwischen der Konsistenz- und der Intervallrechnung und dem probabilistischen Erfüllbarkeitsproblem erläutert.

Satz 5
Es sei eine Produktübersichtsformel POF und eine Code-Statistik $s : C \to [0, 1]$ für eine Codemenge $\{c_1, \ldots, c_m\} = C \subseteq Var(POF)$ gegeben. Die Code-Statistik s ist unter der Produktübersichtsformel POF genau dann konsistent, wenn die PSAT-Instanz

$$\begin{array}{cccc} F_0 := POF, & F_1 := c_1, & \ldots & , F_m := c_m \\ p_0 := 1, & p_1 := s(c_1), & \ldots & , p_m := s(c_m) \end{array} \tag{37}$$

erfüllbar ist.

Beweis. Ist die Code-Statistik s konsistent, dann existiert eine Variantenverteilung x, die s induziert. Da die Variantenverteilung x ausschließlich solchen Codebelegungen β eine Wahrscheinlichkeit größer 0 zuordnet, die die Produktübersichtsformel POF erfüllen, liefert x als Wahrscheinlichkeitsverteilung der Codebelegungen einen Nachweis für die Erfüllbarkeit der PSAT-Instanz (37).
Andererseits sei x eine Wahrscheinlichkeitsverteilung, die die PSAT-Instanz (37) erfüllt. Ihre Einschränkung auf die Codebelegungen, die die Produktübersichtsformel POF erfüllen, induziert die Code-Statistik s. □

Das Konsistenzproblem für Intervalle aus Definition 19 kann analog als probabilistisches Erfüllbarkeitsproblem formuliert werden. Ebenso ist die der Intervallrechnung zugrunde liegende Problemstellung mit dem Problem der induzierten Wahrscheinlichkeiten (probabilistic entailment) formulierbar. In allen Fällen besteht die entscheidende Idee darin, zu fordern, dass die Produktübersichtsformel POF mit der Wahrscheinlichkeit $p = 1$ zum Wahrheitswert 1 evaluiert.
Die beobachtete Verbindung zwischen dem probabilistischen Erfüllbarkeitsproblem und der Konsistenz- und Intervallrechnung wird in den beiden folgenden Abschnitten genutzt, um effektive Verfahren für die Konsistenz und Intervallrechnung zu entwickeln. Dabei wird problemspezifisch, d. h. unter der Voraussetzung $x(POF) = 1$, auf die beiden prominentesten Lösungsansätze für PSAT eingegangen.

Der *lokale Lösungsansatz* beruht auf wiederholt angewendeten Ableitungsregeln, deren Aufgabe das Propagieren verschärfter Wahrscheinlichkeitsintervalle ist [FH94, JP09]. Verfahren, die diesen Ansatz umsetzen, sind unvollständig [Han97], d. h. mit ihnen kann PSAT im Allgemeinen nicht entschieden werden und die abgeleiteten Wahrscheinlichkeitsintervalle sind nicht minimal.
Andererseits besitzen Verfahren, die dem lokalen Ansatz folgen, vernachlässigbar kurze Laufzeiten und gefundene, nachgewiesene Inkonsistenzen – d. h. im konkreten Fall „Unterschranke größer Oberschranke“ – können mittels eines „Proof Trace“ lokalisiert und erklärt werden.

Im *globalen Lösungsansatz* von PSAT werden die linearen Ungleichungssysteme bzw. die linearen Programme aus Unterabschnitt 4.2.1 gelöst. Somit ist der globale Ansatz vollständig, d. h. er kann PSAT entscheiden und berechnet die exakten Wahrscheinlichkeitsintervalle. Hingegen kann Inkonsistenz mit dem globalen Ansatz nur nachgewiesen, nicht aber lokalisiert werden.
Da die in Unterabschnitt 4.2.1 definierte Matrix A eine gegenüber ihrer Zeilenanzahl exponentiell große Spaltenanzahl aufweist, bezeichnete Nilsson den globalen Ansatz im Jahr 1986 noch als „impractical“ [Nil86]. Zu Beginn der 1990er Jahre zeigte sich jedoch, das Nilssons Einschätzung zu pessimistisch war. Mit dem aus der linearen Optimierung

bekannten Ansatz *Column Generation* gelang es erstmals PSAT-Instanzen mit mehreren hundert Formeln F_i zu berechnen [KP90, JHA91]. Column Generation beruht auf dem Simplex-Algortihmus, wobei im Unterschied zu ihm eine neue Basisspalte stets durch das exakte bzw. approximative Lösen einer PBO-Instanz berechnet wird (vgl. Kapitel 3 und Unterabschnitte 4.4.2, 4.4.3 und 4.4.4).

4.3 Lokaler Lösungsansatz

Im lokalen Lösungsansatz für PSAT (siehe Unterabschnitt 4.2.2) werden induzierte Wahrscheinlichkeitsintervalle lokaler, abgeleiteter PSAT-Instanzen berechnet. Eine lokale PSAT-Instanz ist durch eine Teilmenge der Formeln F_i und die zugehörigen Intervalle (l_i, u_i) gegeben. Induzierte Wahrscheinlichkeitsintervalle von Formeln F_{m+1} werden mit Ableitungsregeln bzw. gemäß Unterabschnitt 4.2.1 mit einem linearen Programm bestimmt. Lokal induzierte Intervalle beinhalten die induzierten Intervalle der globalen PSAT-Instanz und werden innerhalb des lokalen Ansatzes in weiteren lokalen Rechnungen verwendet, um sukzessive engere Intervallschranken zu propagieren.

4.3.1 Ein Beispiel

In [JP09] wird mit AD-PSAT ein Algorithmus vorgestellt, der den lokalen Ansatz umsetzt. Er zeichnet sich dadurch aus, dass er auf Klauseln Wahrscheinlichkeitsintervalle für Literale propagiert. Die lokalen PSAT-Instanzen setzen sich somit aus einzelnen Klauseln und Wahrscheinlichkeitsintervallen für die enthaltenen Literale zusammen. Im Zusammenhang mit der Konsistenz- und Intervallrechnung wird folgendes Beispiel betrachtet, das den Nachteil derartiger lokaler Instanzen aufzeigt. Im nachfolgenden Unterabschnitt wird anschließend ein Verfahren vorgeschlagen, das die Strukturen einer typischen Produktübersichtsformel besser ausnutzt.

Beispiel 11
Es sei die PSAT-Instanz

$$\begin{aligned} F_1 &= c_1 \vee c_2 \vee c_3, & I_1 &= [1.0, 1.0] \\ F_2 &= \neg c_1 \vee \neg c_2, & I_2 &= [1.0, 1.0] \\ F_3 &= \neg c_1 \vee \neg c_3, & I_3 &= [1.0, 1.0] \\ F_4 &= \neg c_2 \vee \neg c_3, & I_4 &= [1.0, 1.0] \\ F_5 &= c_1, & I_5 &= [0.2, 0.3] \\ F_6 &= c_2, & I_6 &= [0.3, 0.4] \end{aligned}$$

gegeben. Die aussagenlogischen Formeln F_1 bis F_4 entsprechen dem LPB-Constraint exactly$(1, \{c_1, c_2, c_3\})$. Ist eine Wahrscheinlichkeitsverteilung x der Codebelegung gegeben, die die PSAT-Instanz erfüllt, so genügt jede Code-Belegung β mit $x(\beta) > 0$ dem

exactly-1-Constraint. Für den Code c_3 leitet sich demnach das induzierte Wahrscheinlichkeitsintervall $I_{\text{global}}(c_3) = [0.3, 0.5]$ ab.
Nun soll das Intervall $I_{\text{lokal}}(c_3) = [l_{\text{lokal}}(c_3), u_{\text{lokal}}(c_3)]$ für c_3 mit dem lokalen Ansatz berechnet werden, wobei die lokalen Rechnungen derart limitiert seien, dass nur eine der Formeln F_1 bis F_4 in einer lokalen PSAT-Instanz enthalten sein darf. Zunächst können aus (F_5, I_5) und (F_6, I_6) die Intervalle

$$F_7 = \neg c_1, \quad I_5 = [0.7, 0.8]$$
$$F_8 = \neg c_2, \quad I_6 = [0.6, 0.7]$$

abgeleitet werden. Aus (F_1, I_1), (F_5, I_5) und (F_6, I_6) ergibt sich $l_{\text{lokal}}(c_3) = 0.3$. Aus (F_3, I_3) und (F_7, I_7) folgt $l_{\text{lokal}}(\neg c_3) = 0.2$. Mit (F_4, I_4) und (F_7, I_7) kommt es zum Update auf $l_{\text{lokal}}(\neg c_3) = 0.3$. Diese Schranke induziert $u_{\text{lokal}}(c_3) = 0.7$. Das berechnete Intervall $[0.3, 0.7]$ für c_3 kann mit dem gewählten lokalen Ansatz nicht weiter verkleinert werden.

4.3.2 Ableitungsregeln einer lokalen Intervallrechnung

In diesem Unterabschnitt wird ein unvollständiges Verfahren für die Konsistenz- und Intervallrechnung beschrieben, das auf dem lokalen Ansatz von PSAT beruht.
Für die Produktübersichtsformel POF ist das Wahrscheinlichkeitsintervall $[1, 1]$ zu fordern. Dies ist gleichbedeutend damit, dass alle in der Produktübersichtsformel POF enthaltenen Constraints C_i (d. h. $POF = \bigwedge_i C_i$) das Wahrscheinlichkeitsintervall $[1, 1]$ erfüllen. Neben der Produktübersichtsformel POF seien Code-Intervalle (l, u) auf der Code-Menge $Var(POF)$ vorgegeben.
In Abschnitt 2.2 wurde beispielhaft eine Produktübersichtsformel POF_{PPM} aufgestellt. Ein großer Teil der darin enthaltenen Constraints ist durch Primärgruppen gegeben. Wie das Beispiel 11 verdeutlicht, ist es nicht sinnvoll, die Primärgruppen der Produktübersichtformel – die exactly-1-Constraints – in eine KNF zu überführen. Im Folgenden werden daher Ableitungsregeln definiert, welche auf atleast-1-Constraints und auf atmost-1-Constraints Wahrscheinlichkeitsintervalle für Literale propagieren.
Alle Constraints der Produktübersichtformel POF, die nicht exactly-1-Constraints entsprechen, seien als KNF gegeben, so dass POF der Darstellung

$$POF = \bigwedge_k \text{atleast}(1, M_k) \wedge \bigwedge_k \text{atmost}(1, N_k), \qquad M_k,\ N_k \text{ Literalmenge}, \tag{38}$$

genüge.

Definition 22 (lokale Ableitungsregeln der Konsistenz- und Intervallrechnung)
Für alle Literale L sei ein Wahrscheinlichkeitsintervall durch die Untergrenze $l(L)$

und die Obergrenze $u(L)$ definiert. Für die Intervallgrenzen werden die folgenden Ableitungsregeln definiert.

$$\frac{l_i = l(L_i)}{u(\overline{L}_i) = \min\left(u(\overline{L}_i), 1 - l_i\right)} \qquad (l\text{-Propagation})$$

$$\frac{u_i = u(L_i)}{l(\overline{L}_i) = \max\left(l(\overline{L}_i), 1 - u_i\right)} \qquad (u\text{-Propagation})$$

$$\frac{\begin{array}{c} L_1 + L_2 + \cdots + L_n \geq 1 \\ u_i = u(L_i) \;\; \forall i \\ j \in \{1, \ldots, n\} \end{array}}{l(L_j) = \max\left(l(L_j), 1 - \sum_{i \neq j} u_i\right)} \qquad (\text{atleast-1-Propagation})$$

$$\frac{\begin{array}{c} L_1 + L_2 + \cdots + L_n \leq 1 \\ l_i = l(L_i) \;\; \forall i \\ j \in \{1, \ldots, n\} \end{array}}{u(L_j) = \min\left(u(L_j), 1 - \sum_{i \neq j} l_i\right)} \qquad (\text{atmost-1-Propagation})$$

Ein im Rahmen dieser Arbeit implementierter Algorithmus der Konsistenz- und Intervallrechnung ist durch die iterative Anwendung der vier oben definierten Ableitungsregeln gegeben. Der Algorithmus stoppt, falls keine engeren Intervallgrenzen propagiert werden können bzw. falls ein Code c mit $l(c) > u(c)$ eine Inkonsistenz nachweist.

Zur effizienten Implementierung der beiden Ableitungsregeln atleast-1-Propagation und atmost-1-Propagation werden die Zahlen

$$U_k := \sum_{l \in M_k} u(l), \;\; L_k := \sum_{l \in N_k} l(l) \tag{39}$$

initialisiert, wobei mit M_k bzw. N_k die Literalmengen der atleast-1-Constraints bzw. die Literalmengen der atmost-1-Constraints bezeichnet werden (vgl. Gleichung (38)). Bei einem Update einer Intervallschranke werden die betroffenen Zahlen U_k und L_k reduziert bzw. erhöht. Für eine aktualisierte Zahl U_K bzw. L_K wird die Ableitungsregel atleast-1-Propagation bzw. atmost-1-Propagation auf die Literale $L_j \in M_K$ bzw. $L_j \in N_K$ angewandt. Dabei wird die Summe der Oberschranken $\sum_{i \neq j} u_i$ bzw. die Summe der Unterschranken $\sum_{i \neq j} l_i$ in $\mathcal{O}(1)$ aus U_k bzw. L_k berechnet. In Abschnitt 4.6 werden die numerischen Ergebnisse des so implementierten Algorithmus vorgestellt.

Wird für jede Propagation einer Intervallschranke der Grund gespeichert, so wird für jede mit dem Algorithmus berechnete Intervallschranke ein sogenannter Proof-Trace aufgezeichnet. Ein solcher Proof-Trace, wie er auch von CDCL-Solvern gespeichert werden kann, dient beispielsweise zur Erklärung einer gefundenen Inkonsistenz.
Im globalen Lösungsansatz von PSAT ist kein erklärender Proof-Trace ableitbar. Hingegen werden vorliegende Inkonsistenzen garantiert erkannt und die berechneten induzierten Wahrscheinlichkeitsintervalle sind garantiert minimal.

4.4 Globaler Lösungsansatz

Motiviert durch die Überlegungen aus Unterabschnitt 4.2.2 und den in der Literatur beschriebenen Einsatz von Column Generation zur vollständigen Lösung von PSAT [KP90, JHA91] werden zunächst die linearen Programme der Konsistenz- und Intervallrechnung aufgestellt. Anschließend werden die für Column Generation nötigen Grundlagen der linearen Optimierung erläutert. Eine wichtige Voraussetzung bilden die im Anhang aufgeführten Grundlagen der linearen Algebra. Die Theorie wird auf einen ausreichenden, angepassten Umfang reduziert. In begleitenden Beobachtungen wird auf die Zusammenhänge zum Anwendungsfall der Konsistenz- und Intervallrechnung hingewiesen. Begriffe wie *Konsistenz* und *induzierte Intervalle* beziehen sich wie in den vorangegangenen Abschnitten stets auf eine feste Produktübersichtsformel POF und auf statistische Vorgaben für eine feste Menge $C \subseteq Var(POF)$ von $m = |C|$ Codes.
Im abschließenden Unterabschnitt 4.4.5 wird ein globales Verfahren für die Konsistenz- und Intervallrechnung vorgestellt, das Column Generation – unter vorteilhafter Berücksichtigung der numerischen Ergebnisse aus den Kapiteln 2 und 3 – umsetzt.
Die in diesem Abschnitt behandelten Grundlagen der linearen Optimierung sind außerdem für Abschnitt 4.5 erforderlich, in dem eine Methode zur automatischen Reparatur von zur Produktübersicht inkonsistenten Code-Intervallen vorgestellt wird.

4.4.1 Lineare Programme der Konsistenz und Intervallrechnung

Lineare Optimierung bezeichnet das Berechnen linearer Programme. Ein lineares Programm beschreibt die Aufgabe, eine *lineare Zielfunktion* $c : \mathbb{R}^n \longrightarrow \mathbb{R}$ unter linearen Nebenbedingungen zu minimieren bzw. zu maximieren. *Lineare Nebenbedingungen* sind Gleichungen bzw. Ungleichungen der Form $a_i^t x \triangleright b_i,\ \triangleright \in \{\leq, \geq, =\},\ a_i \in \mathbb{R}^n, b_i \in \mathbb{R}$. Ein Vektor $x \in \mathbb{R}^n$ stellt für ein lineares Programm eine *zulässige Lösung* dar, falls er alle Nebenbedingungen erfüllt. Eine *optimale Lösung* ist eine zulässige Lösung, sodass der Zielfunktionswert für keine andere zulässige Lösung kleiner bzw. größer ist.
Als eine Standardmethode zum Berechnen linearer Programme hat sich der *Simplex-Algorithmus* von G.Dantzig, vgl. [Dan90], durchgesetzt, der die Grundlage von Column Generation darstellt. Mit dem Dualitätstheorem der linearen Optimierung wird im folgenden Unterabschnitt zunächst die zugrunde liegende Theorie erläutert, bevor der

Simplex-Algorithmus nachfolgend in Unterabschnitt 4.4.3 skizziert wird. Die Grundversion des Simplex-Algorithmus berechnet ein lineares Programm der Form

$$\begin{aligned} & \min && c^t x \\ \text{s.th.} & && \\ & && Ax = b, \\ & && x \geq \mathbf{0}. \end{aligned} \tag{P}$$

Lineare Programme mit derartigen Nebenbedingungen werden auch als *kanonische Programme* bezeichnet. Ein Vektor x erfüllt die Nebenbedingungen eines kanonischen Programms, falls die Linearkombination Ax der Spalten aus A eine konische Darstellung von b ist.

Die nachfolgende Beobachtung formuliert die Nebenbedingungen an einen Vektor x, der eine Variantenverteilung repräsentiert, welche eine Code-Statistik bzw. Code-Intervalle erfüllt. Dabei ist auf den Unterschied zu den PSAT-Formulierungen (34) und (35) aus Unterabschnitt 4.2.1 zu achten.

Beobachtung 3
Ein Auftrag entspricht einer Belegung aussagenlogischer Variablen (Codes). Dementsprechend kann ein Auftrag v_j als ein Vektor $a_j \in \{0,1\}^m$ dargestellt werden. Gilt $a_{ij} = 1$ für die i-te Komponente a_{ij} des Vektors a_j, so ist der i-te Code c_i im Auftrag vertreten. Gilt $a_{ij} = 0$, so kommt der i-te Code c_i nicht im Auftrag a_j vor.
Es sei $A \in \mathbb{R}^{m \times n}$ die Matrix, die spaltenweise die n zulässigen Varianten $\mathbb{V}_C(POF)$ der Produktübersicht POF enthält. Für eine durch einen Vektor gegebene Variantenverteilung x mit $x_j = x(a_j),\ x \in \mathbb{R}^n$, ist die induzierte Code-Statistik durch

$$s_x = Ax \tag{40}$$

gegeben. Eine Verteilung x der Varianten

$$a_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix},\ a_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix},\ a_3 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ mit } x = \begin{pmatrix} 0,2 \\ 0,3 \\ 0,5 \end{pmatrix}$$

induziert beispielsweise die Code-Statistik

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0,2 \\ 0,3 \\ 0,5 \end{pmatrix} = 0,2 \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 0,3 \begin{pmatrix} 0 \\ 1 \end{pmatrix} + 0,5 \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0,7 \\ 0,8 \end{pmatrix}.$$

Das Problem der Konsistenz für eine Code-Statistik $b \in \mathbb{R}^m$ ist daher äquivalent mit

der Existenzfrage nach einem Vektor $x \in \mathbb{R}^n$, der $Ax = b$ erfüllt und die Eigenschaften einer Verteilung besitzt, also $x \geq \mathbf{0}$ und $\sum x_i = 1$. In einer kompakteren Darstellung muss der Vektor x somit

$$\begin{pmatrix} \mathbf{1}^t \\ A \end{pmatrix} x = \begin{pmatrix} 1 \\ b \end{pmatrix}, \; x \geq \mathbf{0} \tag{41}$$

erfüllen.
Analog kann die Konsistenz von Code-Intervallen $(l, u) \in \mathbb{R}^m \times \mathbb{R}^m$ beschrieben werden. Konsistenz liegt genau dann vor, falls ein Vektor $x \in \mathbb{R}^n$ existiert, so dass

$$\begin{pmatrix} 1 \\ l \end{pmatrix} \leq \begin{pmatrix} \mathbf{1}^t \\ A \end{pmatrix} x \leq \begin{pmatrix} 1 \\ u \end{pmatrix}, \; x \geq \mathbf{0} \tag{42}$$

gilt.

In den weiteren Ausführungen werden teilweise nur Nebenbedingungen des Typs (41) betrachtet. Dies stellt mindestens wegen Satz 4 keine Einschränkung dar. Die nächste Beobachtung formuliert die Berechnung einer induzierten Intervallschranke als ein kanonisches Programm.

Beobachtung 4
Es sei ein Code $d \in Var(POF)$ vorgegeben. Die Matrix $A_{\neg d}$ enthalte spaltenweise alle baubaren Varianten $C \longrightarrow \{0, 1\}$, in denen der Code d nicht enthalten sein muss. Mit A_d seien im Gegensatz alle baubaren Varianten gegeben, die den Code d enthalten können. Gilt $d \notin C$, so sind $Sp(A_{\neg d})$ und $Sp(A_d)$ nicht zwingend disjunkt. Die von einer Code-Statistik $b \in \mathbb{R}^m$ induzierte Unterschranke/Oberschranke für c ist durch das Optimum des folgenden kanonischen Programms gegeben:

$$\begin{aligned} \min / \max \quad & \begin{pmatrix} \mathbf{0^t} & \mathbf{1}^t \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \\ \text{s.th.} \quad & \begin{pmatrix} \mathbf{1}^t & \mathbf{1}^t \\ A_{\neg d} & A_d \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 1 \\ b \end{pmatrix}, \qquad (L/U) \\ & x_0, \; x_1 \geq \mathbf{0}. \end{aligned}$$

Die Formulierungen aus Beobachtung 3 zur Konsistenzrechnung werden ebenso mit einem linearen Programm berechnet. Auf das Bestimmen einer beliebigen zulässigen Lösung, der sogenannten Phase 1 des Simplex-Algorithmus, wird zum Ende des Unterabschnittes 4.4.3 eingegangen.

4.4.2 Dualitätstheorem der linearen Optimierung

In der Regel ist es nicht möglich, die linearen Programme (L/U) einem Standardsolver wie CPLEX [CPL15] zu übergeben und lösen zu lassen. Schon eine explizite Aufzählung aller baubaren Varianten ist üblicherweise nicht zu realisieren, da die Anzahl der baubaren Varianten exponentiell in $m = |C|$ wächst. Die Standardtechnik zum Lösen linearer Programme mit sehr großem n ist *Column Generation*. Column Generation basiert auf dem Dualitätstheorem und wendet den Simplex-Algorithmus zum Lösen linearer Programme wiederholt an.
Der nachfolgende Satz ist als ein Ergebnis von Arbeiten der Mathematiker Farkas, Minkowsky, Caratheodory und Weyl fundamental für das Dualitätstheorem der linearen Programmierung. Darüber hinaus liefert er wichtige Erkenntnisse zu konsistenten Code-Statistiken. Eine etwas allgemeiner formulierte Version ist in *Theory of Linear and Integer Programming* von A. Schrijver zu finden (S. 85, [Sch86]). Der Beweis des Satzes ist im Wesentlichen durch eine versteckte Variante des Simplex-Algorithmus gegeben und kann im Anhang nachgelesen werden.

Satz 6
Es seien Vektoren $a_1, \dots a_n \in \mathbb{R}^m$ mit $\text{lin}\{a_1, \dots a_n\} = \mathbb{R}^m$ gegeben. Für beliebige $b \in \mathbb{R}^m$ gilt genau eine der beiden folgenden Aussagen:

1. Es existieren m linear unabhängige Vektoren $a_{i_1}, \dots, a_{i_m}$ aus $\{a_1, \dots a_n\}$ mit denen b konisch darstellbar ist, d. h. $b \in \text{cone}\{a_{i_1}, \dots, a_{i_m}\}$.

2. Es existieren $m-1$ Vektoren $a_{i_1}, \dots, a_{i_{m-1}}$ aus $\{a_1, \dots a_n\}$, die eine Hyperebene $\{x | c^t x = 0\}$ aufspannen, so dass $c^t b < 0$ und $c^t a_1, \dots, c^t a_n \geq 0$ gilt.

Die Hauptaussage von Satz 6 ist aus geometrischer Sicht leicht zu verifizieren: Entweder liegt ein Ortsvektor b im von $a_1, \dots, a_n$ aufgespannten Kegel C oder es existiert eine Hyperebene, die b und C räumlich trennt. Diese Aussage ist Grundlage für die folgenden zwei Lemmata und somit für das anschließende Dualitätstheorem der linearen Programmierung. Des Weiteren folgt aus Satz 6 eine wichtige Beobachtung zu konsistenten Code-Statistiken.

Beobachtung 5
Repräsentiert $b \in \mathbb{R}^m$ eine konsistente Code-Statistik, so existiert eine Variantenverteilung x mit induzierter Code-Statistik $s_x = b$, die höchstens $m+1$ Varianten eine echt positive Häufigkeit zuordnet, d. h. $|\{i | x_i > 0\}| \leq m+1$.

Beweis. Das lineare System aus (41) sei mit $\tilde{A} \in \mathbb{R}^{\tilde{m} \times n}$, $\tilde{b} \in \mathbb{R}^{\tilde{m}}$, $\tilde{m} \leq m$ so um redundante Zeilen reduziert, dass

$$\text{rang} \begin{pmatrix} \mathbf{1}^t & 1 \\ \tilde{A} & \tilde{b} \end{pmatrix} = \tilde{m} + 1$$

gilt. Da b konsistent ist, ist auch $\tilde{b}$ konsistent und es gilt sogar

$$\text{rang} \begin{pmatrix} \mathbf{1}^t \\ \tilde{A} \end{pmatrix} = \tilde{m} + 1, \text{ d. h. lin } Sp \begin{pmatrix} \mathbf{1}^t \\ \tilde{A} \end{pmatrix} = \mathbb{R}^{\tilde{m}+1}$$

Da nur redundante Zeilen entfernt wurden, gilt außerdem: Ein Vektor x repräsentiert eine Verteilung mit $s_x = b$, falls er

$$\begin{pmatrix} \mathbf{1}^t \\ \tilde{A} \end{pmatrix} x = \begin{pmatrix} 1 \\ \tilde{b} \end{pmatrix}, \ x \geq \mathbf{0} \tag{43}$$

erfüllt.
Da $\tilde{b}$ konsistent ist, gibt es ein x, das (43) erfüllt. Somit gilt die erste Aussage von Satz 6 für die Spalten der Matrix $\begin{pmatrix} \mathbf{1}^t \\ \tilde{A} \end{pmatrix}$ und es existiert eine Verteilung x mit

$$|\{i|x_i > 0\}| \leq \tilde{m} + 1 \leq m + 1.$$

□

Die Beweise der beiden folgenden Lemmata können im Anhang oder in [Sch86] nachgelesen werden. Der Beweis von Farkas II benötigt das Lemma Farkas I.

Lemma 2 (Farkas I)
Sei $A \in \mathbb{R}^{m \times n}$ eine Matrix mit $\text{rang}(A) = m$ und $b \in \mathbb{R}^m$ ein Vektor. Dann ist genau eines der beiden Systeme lösbar:

1. $Ax = b, \ x \geq \mathbf{0}$,
2. $y^t A \geq \mathbf{0}, \ b^t y < 0$.

Lemma 3 (Farkas II)
Sei $A \in \mathbb{R}^{m \times n}$ eine Matrix und $b \in \mathbb{R}^m$ ein Vektor. Dann ist genau eines der beiden Systeme lösbar:

1. $Ax \leq b$,
2. $y^t A = \mathbf{0}, \ y \geq \mathbf{0}, \ y^t b < 0$.

Ausgehend von einem linearen Programm (primales Programm) existiert immer auch ein zugehöriges duales lineares Programm. Das Dualitätstheorem liefert in jeder seiner Ausprägungen den folgenden Zusammenhang: Besitzen primales und duales Programm mindestens eine zulässige Lösung, so besitzen beide Programme ein identisches Optimum. Der folgende Satz gibt das Dualitätstheorem für ein kanonisches Programm mit zu minimierender Zielfunktion an und formuliert das zugehörige duale Programm.

Satz 7 (Dualitätstheorem der linearen Programmierung)
Sei $A \in \mathbb{R}^{m\times n}$ eine Matrix und seien $c \in \mathbb{R}^n,\ b \in \mathbb{R}^m$ Vektoren. Besitzen die linearen Programme

$$\begin{aligned} &\min && c^t x \\ &\text{s.th.} && \\ & && Ax = b, \\ & && x \geq \mathbf{0} \end{aligned} \tag{P}$$

und

$$\begin{aligned} &\max && b^t y \\ &\text{s.th.} && \\ & && A^t y \leq c, \end{aligned} \tag{D}$$

zulässige Lösungen, dann besitzen sie auch optimale Lösungen $x^*,\ y^*$ mit $c^t x^* = b^t y^*$.

Beweis nach [Sch86]. Seien ein $x \geq \mathbf{0}$ mit $Ax = b$ und ein y mit $A^t y \leq c$ gegeben. Dann gilt

$$c^t x = x^t c \geq x^t A^t y = b^t y.$$

Damit folgt sofort die Aussage „min $\geq$ max“.
Für die Umkehrung reicht es, die Existenz von x, y mit

$$x \geq \mathbf{0},\ Ax = b,\ A^t y \leq c,\ c^t x \leq b^t y \tag{$*$}$$

nachzuweisen. Zwei Vektoren x, y erfüllen genau dann $(*)$, wenn der Vektor $\begin{pmatrix} x \\ y \end{pmatrix}$ das System

$$\begin{pmatrix} -I_n & \mathbf{0} \\ A & \mathbf{0} \\ -A & \mathbf{0} \\ \mathbf{0} & A^t \\ c^t & -b^t \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \leq \begin{pmatrix} \mathbf{0} \\ b \\ -b \\ c \\ 0 \end{pmatrix} \tag{$**$}$$

erfüllt. Die Existenz von einem Vektor, der $(**)$ erfüllt, wird mit Hilfe von Lemma 3 nachgewiesen. Sei ein Vektor

$$\begin{pmatrix} s \\ u \\ v \\ w \\ \lambda \end{pmatrix}$$

mit $u, v \in \mathbb{R}^m$, $s, w \in \mathbb{R}^n,\ \lambda \in \mathbb{R}$ und

$$s, u, v, w, \lambda \geq \mathbf{0} \tag{44}$$

$$-s^t + u^t A - v^t A + \lambda c^t = \mathbf{0} \tag{45}$$

$$w^t A^t - \lambda b^t = \mathbf{0} \tag{46}$$

vorgegeben. Es ist die Gültigkeit der Ungleichung

$$u^t b - v^t b + w^t c \geq 0$$

zu begründen. Sei zunächst $\lambda > 0$. Dann gilt

$$\begin{aligned} w^t c &= \lambda^{-1}\lambda c^t w \\ &\overset{(45)}{=} \lambda^{-1}(v^t - u^t)Aw + s^t w \\ &\overset{(44)}{\geq} \lambda^{-1}(v^t - u^t)Aw \\ &\overset{(46)}{=} \lambda^{-1}(v^t - u^t)b\lambda \\ &= v^t b - u^t b \end{aligned}$$

Für den Fall $\lambda = 0$ seien Vektoren x_0, y_0 mit $Ax_0 = b,\ x_0 \geq \mathbf{0}$, und $A^t y_0 \leq c$ gegeben. Es gilt

$$\begin{aligned} w^t c &\geq w^t A^t y_0 \\ &\overset{(46)}{=} 0 \\ &\overset{(45)}{=} s^t x_0 + (v^t - u^t)Ax_0 \\ &\overset{(44)}{\geq} (v^t - u^t)Ax_0 \\ &= v^t b - u^t b. \end{aligned}$$

□

Eine für Unterabschnitt 4.4.5 wichtige Variante des Dualitätstheorems liefert das folgende Korollar.

Korollar 1
Sei $A \in \mathbb{R}^{m\times n}$ eine Matrix und seien $c \in \mathbb{R}^n,\ b \in \mathbb{R}^m$ Vektoren. Besitzen die linearen Programme

$$\begin{aligned} &\min && c^t x \\ &\text{s.th.} && \\ & && Ax \leq b, \\ & && x \geq \mathbf{0} \end{aligned} \tag{$P_{\leq}$}$$

und

$$\begin{aligned} &\max && b^t y \\ \text{s.th.} & \\ & && A^t y \leq c, \\ & && y \leq \mathbf{0} \end{aligned} \tag{$D_{\leq}$}$$

zulässige Lösungen, dann besitzen sie auch optimale Lösungen $x^*,\ y^*$ mit $c^t x^* = b^t y^*$.

Beweis. Das kanonische Programm

$$\begin{aligned} &\min && \begin{pmatrix} c^t & \mathbf{0}^t \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} \\ \text{s.th.} & \\ & && \begin{pmatrix} A & I_m \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} = b, \\ & && x, z \geq \mathbf{0} \end{aligned}$$

ist äquivalent zu $(P_{\leq})$ (d. h. x^*, z^* ist genau dann eine optimale Lösung, wenn x^* eine optimale Lösung von $(P_{\leq})$ ist) und sein duales Programm

$$\begin{aligned} &\max && b^t y \\ \text{s.th.} & \\ & && \begin{pmatrix} A^t \\ I_m \end{pmatrix} y \leq \begin{pmatrix} c \\ \mathbf{0} \end{pmatrix} \end{aligned}$$

entspricht dem Programm $(D_{\leq})$. □

Jedes lineare Programm kann in ein kanonisches Programm überführt werden. Der Beweis zu Korollar 1 illustriert, dass Ungleichungen durch die Einführung von Hilfsvariablen zu Gleichungen transformiert werden können. Fehlt in einem linearen Programm hingegen die Bedingung, dass die Entscheidungsvariablen positiv sind, so kann diese mit einer Umformung erzwungen werden, wie sie im Folgenden exemplarisch umgesetzt ist:

$$\begin{aligned} &\min && c^t x \\ \text{s.th.} & \\ & && Ax \geq b \end{aligned}$$

$$
\begin{aligned}
= \quad & \\
& \min \qquad \begin{pmatrix} c^t & -c^t \end{pmatrix} \begin{pmatrix} x^+ \\ x^- \end{pmatrix} \\
& \text{s.th.} \\
& \qquad \begin{pmatrix} A & -A \end{pmatrix} \begin{pmatrix} x^+ \\ x^- \end{pmatrix} \geq b, \\
& \qquad x^+, x^- \geq \mathbf{0}.
\end{aligned}
$$

Tritt der Constraint $x \leq \mathbf{0}$ auf, so können die Entscheidungsvariablen mit $-\tilde{x} := x$ substituiert werden.
Als eine Konsequenz daraus, dass jedes lineare Programm in ein kanonisches Programm transformierbar ist, wird der Simplex-Algorithmus im folgenden Unterabschnitt ausschließlich für kanonische Programme dargestellt.

4.4.3 Simplex-Algorithmus

In diesem Unterabschnitt wird mit dem Simplex-Algorithmus ein Standardalgorithmus zum Lösen linearer Programme betrachtet. Das Verfahren wurde 1947 von G. Dantzig vorgestellt, vgl. [Dan90]. In der ursprünglichen Version wird ein kanonisches Programm (P) vorausgesetzt, dessen Zielfunktion zu minimieren ist. Dies stellt keine Einschränkung dar, da jedes lineare Programm in ein kanonisches Programm überführt werden kann und da das Maximieren einer Zielfunktion c^t gleichbedeutend mit dem Minimieren der Zielfunktion $-c^t$ ist.
Bei der Berechnung einer optimalen Lösung durch den Simplex-Algorithmus wird immer auch eine optimale Lösung des dualen Programms berechnet. Diese wird im nachfolgenden Unterabschnitt für Column Generation benötigt.

Der Simplex-Algorithmus iteriert über zulässige Lösungen entlang der Zielfunktion hin zu einer optimalen Lösung. Der nachfolgende Satz 8 formuliert die wesentlichen Eigenschaften, die die Menge der zulässigen Lösungen eines kanonischen Programms (P) erfüllt. Der Satz folgt den Darstellungen aus [Aig07], S.307-310. Sein Beweis kann im Anhang nachgelesen werden.

Mit

$$M := \{x \mid Ax = b, x \geq \mathbf{0}\} \subseteq \mathbb{R}^n \tag{47}$$

ist die Menge der zulässigen Lösungen von (P) gegeben. Als Schnitt von Halbräumen stellt M einen *Polyeder* dar und ist insbesondere *konvex*, d. h. für $x', x'' \in M$ und jeden Parameter $\lambda \in [0, 1]$ gilt $\lambda x' + (1 - \lambda)x'' \in M$:

$$A(\lambda x' + (1 - \lambda)x'') = \lambda A x' + (1 - \lambda)Ax'' = \lambda b + (1 - \lambda)b = b.$$

Geometrisch bedeutet dies, dass jeder Punkt auf der Verbindungsstrecke $\overline{x'x''}$ in M liegt. Bei gegebener Konvexität einer Menge M ist der Begriff der *Ecke* $p \in M$ wie folgt definiert :

$$\text{Aus } p = \lambda x' + (1-\lambda)x'',\ x', x'' \in M,\ \lambda \in [0,1] \text{ folgt } p = x' \text{ oder } p = x''.$$

Eine Ecke zeichnet sich somit dadurch aus, dass sie nicht im Inneren einer Verbindungsstrecke $\overline{x'x''}$ liegt.

Satz 8
Es sei M die Menge der zulässigen Lösungen eines kanonischen Programms (P).

1. Ein Punkt $x \in M$ ist genau dann eine Ecke, falls die Spaltenmenge $\{a_i \mid x_i > 0\}$ linear unabhängig ist.

2. Ist $M \neq \emptyset$, so besitzt M eine Ecke.

3. Existieren in M optimale Lösungen, so existiert unter den optimalen Lösungen eine Ecke.

Für eine Teilmenge von Spaltenindizes $B \subseteq \{1, \ldots, n\}$, die mit m linear unabhängigen Spalten von A korrespondiert, sei $\tilde{x}$ durch $A_B\tilde{x} = b$, also $\tilde{x} = A_B^{-1}b$ gegeben. Des Weiteren sei $N := \{1, \ldots, n\} \setminus B$ die Menge der restlichen Spaltenindizes. Nach Satz 8 ist der Punkt $x \in \mathbb{R}^n$ mit $x_B := \tilde{x}$ und $x_N := \mathbf{0}$ eine Ecke von M, falls $x \in M$, d. h. $x_B \geq \mathbf{0}$ gilt. In diesem Fall wird B als *Basislösung* bezeichnet. Somit entspricht jede Basislösung einer Ecke in M. Andererseits wird eine Ecke durch mindestens eine Basislösung repräsentiert. Mehrere Basislösungen zu einer Ecke existieren genau dann, falls $x_i = 0$ für ein $i \in B$ und eine Basislösung B gilt. Dann ist der Index i aus B austauschbar und die Ecke wird als *entartet* bezeichnet (sonst als *nichtentartet*).

Ist bekannt, dass ein kanonisches Programm eine optimale Lösung besitzt, so ist theoretisch folgendes Vorgehen möglich: Betrachte alle potentiellen Basislösungen und berechne alle Ecken von M. Wähle unter allen Ecken eine Ecke x^* mit $c^t x^*$ minimal.
Diese Idee wird durch den Simplex-Algorithmus verfeinert. Anstatt alle Ecken zu berücksichtigen, werden nur solche Ecken betrachtet, die möglichst zu einer Verbesserung und mindestens zu keiner Verschlechterung in der Zielfunktion führen. Dazu „läuft“ der Simplex-Algorithmus von Ecke zu Ecke in die Richtung der Zielfunktion. Er stoppt, falls er die Optimalität einer Ecke feststellt bzw. erkennt, dass das lineare Programm unbeschränkt ist.

Angenommen mit B ist eine Basislösung definiert. Die zugehörige Ecke $x \in \mathbb{R}^n$ ist durch $x_B = A_B^{-1}b$ und $x_N = \mathbf{0}$ gegeben. Die nachfolgende Ecke im Simplex-Algorithmus

sei durch $x + p,\ p \in \mathbb{R}^n$ gegeben. Um $x + p \in M$ zu garantieren, ist $A(x + p) = b$ und somit $Ap = \mathbf{0}$ zwingend erforderlich. Es ergibt sich $Ap = A_B p_B + A_N p_N = \mathbf{0}$, d. h. $p_B = -A_B^{-1} A_N p_N$. Die durch p_N gegebenen Änderungen in x_N definieren somit die durch p_B gegebenen Änderungen in x_B.
Der klassische Simplex-Algorithmus schränkt die möglichen Änderungen in x_N auf Vektoren der Form $p_N = \alpha \mathbf{e}_s \in \mathbb{R}^{n-m},\ \alpha \in \mathbb{R}_{\geq 0}$ ein. Es gilt somit $p_B = -\alpha u$ für $u := A_B^{-1} a_{i_s}$. Des Weiteren existiert eine Basislösung $\tilde{B}$ der Ecke $x+p$ mit $\tilde{B} \subset B \cup \{i_s\}$ für $i_s \in N = \{i_1, \ldots, i_{n-m}\}$. Um also aus einer Basislösung von x eine Basislösung von $x + p$ zu erhalten, reicht es zwischen B und N einen Index zu tauschen.
Damit $x_B + p_B \geq \mathbf{0}$ und somit $x + p \geq \mathbf{0}$ gewährleistet ist, muss $x_{Bi} \geq \alpha u_i$ für alle $i \in \{1, \ldots, m\}$ gelten. Die maximale *Schrittweite* α_{max} in die Richtung von p ist daher durch

$$\alpha_{max} = \min_{\{i|u_i>0\}} \frac{x_{Bi}}{u_i} \text{ bzw. } \alpha_{max} = \infty \text{ für } u \leq 0 \tag{48}$$

gegeben. Gilt $\alpha_{max} = 0$, so gilt $p = \mathbf{0}$ und die nachfolgende Basislösung würde ebenfalls zur Ecke x korrespondieren, welche damit degeneriert wäre.

Es wird nun erläutert, wie der Simplex-Algorithmus vorgeht, um einen Index i_s zu finden, der die Basislösung möglichst verbessert. Dazu werden die Auswirkungen auf die Zielfunktion c^t betrachtet. Es gilt:

$$\begin{aligned} c^t(x + p) &= c^t x + c^t p \\ &= c^t x + c_B^t p_B + c_N^t p_N \\ &= c^t x + \alpha\Big(c_{i_s} - c_B^t A_B^{-1} a_{i_s}\Big). \end{aligned}$$

Die Zahl $\tilde{c}_{i_s} := (c_{i_s} - c_B^t A_B^{-1} a_{i_s})$ ist der i_s-te Eintrag des Vektors

$$\tilde{c}^t := (c^t - c_B^t A_B^{-1} A) \in \mathbb{R}^n. \tag{49}$$

Der Vektor $\tilde{c}^t$ wird auch als Vektor der *reduzierten Kosten* bezeichnet. Er gibt für jede potentielle Basisspalte i_s – in Relation zur Schrittweite – die Veränderung in der Zielfunktion an. Es gilt $\tilde{c}_B^t = \mathbf{0}$. Für $\tilde{c}_N^t$ werden zwei Fälle unterschieden.

1. Gilt $\tilde{c}_N^t \geq \mathbf{0}$, so ist $\tilde{c}^t \geq \mathbf{0}$ und $y^t := c_B^t A_B^{-1}$ ist wegen $y^t A \leq c^t$ eine zulässige Lösung des dualen Programms (D). Wegen der Gleichung

   $$c^t x = c_B^t x_B = c_B^t A_B^{-1} b = y^t b = b^t y$$

   sind x und y nach dem Dualitätstheorem (Satz 7) optimale Lösungen des primalen bzw. dualen Programms. Die Basislösung B beschreibt demnach eine schon optimale Ecke in M und der Algorithmus bricht ab.

2. Ein Index $i_s \in N$ mit $\tilde{c}^t_{i_s} < 0$ stellt einen Kandidaten für eine neue Basislösung dar. Eine garantierte Verbesserung ist mit dem Index i_s möglich, falls für die zugehörige maximale Schrittweise $\alpha_{max} > 0$ gilt. Gilt sogar $\alpha_{max} = \infty$, so ist das kanonische Programm unbeschränkt und der Simplex-Algorithmus bricht ab.
Ist $j_t \in B = \{j_1, \dots j_m\}$ ein Index mit $x_{j_t} + \alpha_{max} u_t = 0$, so ist $B' := B \setminus \{j_t\} \cup \{i_s\}$ eine neue Basislösung mit einem besseren Zielfunktionswert gdw. $\alpha_{max} > 0$. Ein Beweis, dass die Spalten von B' linear unabhängig sind, ist zum Beispiel in *Linear optimization and extensions* von M. W. Padberg zu finden [Pad95], S. 45-47.

Der Simplex Algorithmus setzt zu Beginn eine Basislösung voraus. Diese wird mittels einfachem Indexwechsel zwischen B und N solange iterativ verändert, bis eine Basislösung mit reduzierten Kosten $\tilde{c} \geq \mathbf{0}$ ermittelt wurde. Wie oben erläutert, repräsentiert eine derartige Basislösung eine optimale primale Lösung $x \in \mathbb{R}^n$ mit $x_B = A_B^{-1} b,\ x_N = \mathbf{0}$, und eine optimale duale Lösung $y^t := c_B^t A_B^{-1} \in \mathbb{R}^m$.
Es ist jedoch nicht selbstverständlich, dass der Simplex-Algorithmus terminiert. Werden die Indexe $i_s \in N$ und $j_t \in B$ ungünstig gewählt, so können bereits berechnete Basislösungen erneut auftreten. Ein Vorgehen unter dem der Algorithmus garantiert terminiert, ist beispielsweise durch die Regel von Bland gegeben:

- Wähle den kleinsten Index $i_s \in N$ mit $\tilde{c}_{i_s} < 0$.
- Für die damit festgelegte Schrittweite α_{max} wähle den kleinsten Index $j_t \in B$ mit $x_{j_t} + \alpha_{max} u_t = 0$.

Ein Beweis, dass sich bei Anwendung dieser Regel keine Basislösungen wiederholen, ist ebenso in [Pad95] zu finden (S. 69-70).

Es wird zusammenfassend festgehalten: Besitzt ein kanonisches Programm eine zulässige Lösung, so kann mit dem Simplex-Algorithmus eine optimale Ecke bestimmt werden bzw. nachgewiesen werden, dass das Programm unbeschränkt ist. Im Falle einer optimalen Lösung liefert der Simplex-Algorithmus auch eine optimale duale Lösung.
Das Problem, zu entscheiden, ob eine zulässige Lösung existiert, wird in der Regel mit einem zusätzlichen linearen Programm gelöst. Die Idee der sogenannten Phase I des Simplex-Algorithmus besteht darin, Hilfsvariablen z einzuführen, die es erlauben, eine Basislösung direkt anzugeben. Die Zielfunktion wird in der Phase I so gewählt, dass genau die Hilfsvariablen bestraft werden. Gegebenenfalls liefert dann eine Basislösung mit $z = \mathbf{0}$ eine Basislösung im originalen Programm.
Ein Beispiel ist durch die folgende Beobachtung gegeben, die das Problem der Konsistenz einer Code-Statistik als ein lineares Programm formuliert.

Beobachtung 6
Wie in Beobachtung 3 sei $A \in \mathbb{R}^{m \times n}$ die Matrix aller n Varianten $a \in \mathbb{V}_C(POF)$ für

eine Produktübersicht POF und eine Codemenge $C \subseteq Var(POF)$, $|C| = m$. Eine Code-Statistik $b \in \mathbb{R}^m$ ist genau dann konsistent, wenn das kanonische Programm

$$\begin{aligned} \min \quad & \begin{pmatrix} 0 & \mathbf{1}^t & \mathbf{1}^t \end{pmatrix} \begin{pmatrix} x \\ z^+ \\ z^- \end{pmatrix} \\ \text{s.th.} \quad & \begin{pmatrix} \mathbf{1}^t & 0 & 0 \\ A & I_m & -I_m \end{pmatrix} \begin{pmatrix} x \\ z^+ \\ z^- \end{pmatrix} = \begin{pmatrix} 1 \\ b \end{pmatrix}, \\ & x,\ z^+,\ z^- \geq \mathbf{0} \end{aligned} \tag{K}$$

eine zulässige Lösung mit $z^+ = z^- = \mathbf{0}$ besitzt. Dies ist genau dann der Fall, wenn das Minimum des Programms 0 ist. Allerdings kann auch dieses lineare Programm in der Regel nicht direkt mit dem Simplex-Algorithmus berechnet werden, da die Anzahl der baubaren Varianten zu groß ist.

4.4.4 Column Generation

In diesem Unterabschnitt wird eine kurze Erläuterung der Grundidee von Column Generation gegeben. Eine ausführliche Darstellung mit weiteren Anwendungen ist im Übersichtsartikel [DL05] von Desrosiers et. al zu finden.

Vorausgesetzt sei ein kanonisches Programm (P) mit Existenz einer optimalen Lösung bei zu minimierender Zielfunktion. Nach Satz 8 existiert eine optimale Ecke im Polyeder der zulässigen Lösungen von (P). Column Generation basiert genau wie der Simplex-Algorithmus darauf, dass in einer Ecke höchstens m Entscheidungsvariablen x_i echt positiv sind. Die grundlegende Idee drückt das folgende Lemma aus.

Lemma 4
Es sei $B \subseteq \{1, \ldots n\}$ eine Basislösung einer optimalen Ecke von (P). Ist I eine Spaltenindexmenge mit $B \subseteq I \subseteq \{1, \ldots, n\}$, so ist jede optimale Basislösung B' von

$$\begin{aligned} \min \quad & c_I^t x \\ \text{s.th.} \quad & A_I x = b, \\ & x \geq \mathbf{0} \end{aligned} \tag{P_I}$$

auch eine optimale Basislösung im kanonischen Programm (P).

Beweis. Jede Basislösung von (P_I) ist eine Basislösung von (P). Somit ist das Optimum von (P) mindestens so klein wie das von (P_I). Andererseits ist B auch eine Basislösung von (P_I) und die Optima müssen identisch sein. □

Es ist also nicht nötig alle Spalten von A explizit zu kennen. Es ist ausreichend eine Teilmenge I von Spaltenindizes zu generieren, so dass eine optimale Basislösung B von I überdeckt wird. Ein hinreichendes Kriterium dafür liefert das Dualitätstheorem der linearen Optimierung (Satz 7) und wird im Folgenden erläutert.
Angenommen I repräsentiert eine generierte Menge von Spalten, sodass (P_I) eine zulässige Lösung enthält. Mit x sei eine optimale primale Lösung und mit y eine optimale duale Lösung von (P_I) gegeben. Diese können mit dem Simplex-Algorithmus bestimmt werden. Es stellen sich folgende Fragen:

- Ist eine zur Lösung x korrespondierende Basislösung schon eine optimale Basislösung in (P)?
- Falls nein, wie ist I zu erweitern?

Ist y nicht nur eine optimale Lösung von

$$\begin{aligned} &\min && b^t y \\ \text{s.th.} & && \\ & && A_I^t y \leq c, \end{aligned} \tag{D_I}$$

sondern auch eine zulässige Lösung von (D), d. h. gilt $y^t A \leq c^t$, so kann die erste Frage aufgrund des Dualitätstheorems bejaht werden: Wegen $max(D) \leq max(D_I)$ gilt sogar $max(D) = max(D_I)$ und somit

$$min(P_I) = max(D_I) = max(D) = min(\text{P}). \tag{50}$$

Ein hinreichendes Kriterium für $min(\text{P}) = min(P_I)$ ist somit die Nichtexistenz einer Spalte a_i in A mit

$$c_i - y^t a_i < 0. \tag{51}$$

In Antwort auf die zweite Frage sei a_i eine Spalte, die die Ungleichung (51) erfüllt. Bezogen auf (P) entspricht dies im Simplex-Algorithmus folgender Situation:
Für eine Basislösung $\tilde{B}$ von (P_I) (und somit von (P)), die y durch $y^t = c_{\tilde{B}}^t A_{\tilde{B}}^{-1}$ repräsentiert, ist $c_i - y^t a_i$ ein negativer Eintrag im Vektor der reduzierten Kosten

$$\tilde{c}^t := (c^t - c_{\tilde{B}}^t A_{\tilde{B}}^{-1} A) \in \mathbb{R}^n.$$

Der Index der Spalte a_i ist somit ein Kandidat für eine neue Basislösung mit besserem Zielfunktionswert bzw. ein Kandidat für eine Erweiterung von I.

Zusammengefasst setzt sich eine Iteration von Column Generation aus den folgenden Schritten zusammen:

1. Bestimme eine optimale primale Lösung x und eine optimale duale Lösung y von (P_I) mit dem Simplex-Algorithmus.

2. Existiert keine Spalte a_i in A mit $c_i - y^t a_i < 0$, so stoppe mit der Spaltengenerierung, da x zu einer optimalen Basislösung von (P) korrespondiert.

3. Erweitere I um den Index einer Spalte a_i mit $c_i - y^t a_i < 0$.

Offen bleibt zunächst, wie entschieden wird, ob eine Spalte a_i mit $c_i - y^t a_i < 0$ existiert und wie gegebenenfalls eine derartige Spalte generiert wird. Dies ist in der Regel problemspezifisch und wird im nachfolgenden Unterabschnitt für die Konsistenz- und Intervallrechnung erläutert. Zunächst noch einige Anmerkungen:

- Standardsolver wie CPLEX unterstützen einen Warmstart. Wird also (P_I) durch Erweiterung von I verändert, so wird der Simplex-Algorithmus bei der optimalen Basislösung der vorangegangenen Iteration „fortgesetzt“.

- Die Problematik zyklischer Basislösungen stellt sich Column Generation nicht, sofern sie vom verwendeten Simplex-Algorithmus beherrscht wird.

- Das Problem von degenerierten Basislösungen, die zusammen eine einzige Ecke repräsentieren, tritt auch in Column Generation auf. Es äußert sich dadurch, dass die generierte Spalte im nächsten Schritt von Column Generation mit 0 gewichtet wird. Führen degenerierte Basislösungen dazu, dass Column Generation auf einer Stelle verharrt, so können Techniken angewandt werden, wie sie im Artikel „Stabilized Column Generation“ von du Merle et. al beschrieben werden [MVDH99].

- Die Auswahl einer Spalte a_i, um deren Index die Indexmenge I erweitert wird, ist nur selten eindeutig. Es ist üblich a_i so zu wählen, dass $c_i - y^t a_i < 0$ möglichst klein ist. Bei gleicher Schrittweite ist folglich die Verbesserung im Zielfunktionswert größer.

4.4.5 Auf Implikanten basierende Konsistenz- und Intervallrechnung

Mit Column Generation können die linearen Programme (K) und (L/U) der Konsistenz- und Intervallrechnung berechnet werden. Im Falle der Konsistenzrechnung wird dafür in jeder Iteration von Column Generation ein lineares Programm der Form

$$\begin{aligned} &\min && \begin{pmatrix} 0 & \mathbf{1}^t & \mathbf{1}^t \end{pmatrix} \begin{pmatrix} x \\ z^+ \\ z^- \end{pmatrix} \\ &\text{s.th.} \\ & && \begin{pmatrix} \mathbf{1}^t & 0 & 0 \\ A_I & I_m & -I_m \end{pmatrix} \begin{pmatrix} x \\ z^+ \\ z^- \end{pmatrix} = \begin{pmatrix} 1 \\ b \end{pmatrix}, \qquad (K_I) \\ & && x,\ z^+,\ z^- \geq \mathbf{0}, \end{aligned}$$

berechnet. Die Teilmatrix A_I ist mit einer beliebigen Variante a der Produktübersicht POF initialisierbar, d. h.

$$A_{I_0} := \begin{pmatrix} | \\ a \\ | \end{pmatrix}.$$

Durch

$$x = x_1 := 1,\ z_i^+ := \max(b_i - a, 0) \text{ und } z_i^- := \max(a - b_i, 0)$$

ist folglich die einzige zulässige Lösung für das initiale lineare Programm (K_{I_0}) gegeben. In jeder Iteration von Column Generation wird A_I um eine Spalte (eine zulässige Variante) erweitert, sofern das Minimum von (K_I) noch nicht 0 ist bzw. nachgewiesen wurde, dass die Minima von (K_I) und (K) identisch sind.
Angenommen das Minimum von (K_I) ist größer als 0. Es wird die optimale duale Lösung $y \in \mathrm{R}^{m+1}$ von (K_I) betrachtet. Gemäß der Ungleichung (51) ist ein Modell a der Produktübersichtsformel POF gesucht, das

$$0 - \begin{pmatrix} y_0 & \tilde{y}^t \end{pmatrix} \begin{pmatrix} 1 \\ a \end{pmatrix} < 0, \quad y = \begin{pmatrix} y_0 \\ \tilde{y} \end{pmatrix},\ \tilde{y} \in \mathrm{R}^m,\ y_0 \in \mathrm{R} \qquad (52)$$

erfüllt. Bei Existenz einer entsprechenden Variante a wird A_I erweitert,

$$A_I = \begin{pmatrix} & | \\ A_I & a \\ & | \end{pmatrix},$$

und in der nächsten Iteration von Column Generation wird das aktualisierte lineare Programm (K_I) berechnet. Existiert hingegen kein Modell a der Produktübersichtsformel POF mit

$$-\tilde{y}^t a < y_0, \tag{53}$$

so sind die Minima von (K_I) und (K) identisch und die Code-Statistik b ist inkonsistent zur Produktübersicht POF.

Die Berechnung einer Variante a, die (53) erfüllt, bzw. der Nachweis ihrer Nichtexistenz, entspricht einer typischen PBS-Instanz, wie sie in Kapitel 3 effizient gelöst wird. Um darüber hinaus die Anzahl der Iterationen von Column Generation möglichst gering zu halten, ist es nach Unterabschnitt 4.4.3 erstrebenswert, eine Variante a zu berechnen, die die PBO-Instanz

$$\begin{aligned} &\min && -y^t a \\ &\text{s.th.} && \\ & && POF(a) = 1 \end{aligned} \tag{54}$$

löst. Mit einer solchen Variante wird sichergestellt, dass bei gleicher Schrittweite im Polyeder von (K) die Annäherung an das Optimum größtmöglich ist.

Eine weitere Möglichkeit, die Anzahl der Iterationen von Column Generation zu reduzieren, besteht darin, die Matrix A_I in jeder Iteration um mehrere Varianten zu erweitern. Dies wird nun mit der Grundidee umgesetzt, die Matrix A_I mit Primimplikanten der Produktübersichtsformel POF und nicht mit vollständig auf $C \subseteq Var(POF)$ spezifizierten Varianten zu befüllen. Dazu wird mit dem Algorithmus aus Unterabschnitt 2.3.2 auf Basis einer berechneten Variante a, die die Ungleichung (53) erfüllt, eine Primimplikante ermittelt. Enthält diese Primimplikante beispielsweise 20 *Don't Cares*, so repräsentiert sie 2^{20} Varianten der Produktübersicht, unter denen mindestens die Ausgangsvariante a die Ungleichung (53) erfüllt. Die folgende Beschreibung der technischen Umsetzung geht vom allgemeinen Fall aus, in dem eine Variantenverteilung zu bestimmen ist, die einerseits ein System von Code-Intervallen erfüllt und andererseits eine beliebige lineare Zielfunktion optimiert.

Es sei eine Produktübersichtsformel POF und eine Teilmenge $C = \{c_1, \ldots, c_m\} \subseteq Var(POF)$ der Codes gegeben. Darüber hinaus seien Code-Intervalle $(l, u) \in [0, 1]^{m \times m}$ auf C und eine lineare Zielfunktion $c \in \mathbb{R}^n$ gegeben, wobei c jeder Variante der Produktübersichtsformel POF Kosten zuordnet. Das Ziel ist es, eine Verteilung x der baubaren Varianten zu bestimmen, die die Zielfunktion c^t unter der Voraussetzung optimiert, dass die induzierte Code-Statistik s_x in den Code-Intervallen (l, u) liegt.
Wiederum sei A die Matrix, die spaltenweise die charakteristischen Vektoren aller zulässigen Varianten $\beta : C \to \{0, 1\}$ der Produktübersichtsformel POF enthält. Nach

Beobachtung 3 ist die Konsistenz von Code-Intervallen (l, u) gleichbedeutend mit der Existenz eines Vektors $x \geq 0$, welcher $l \leq Ax \leq b,\ \sum x_i = 1$ erfüllt. Dieses Kriterium für Konsistenz wird nun mit der Motivation verallgemeinert, die durch die Primimplikanten gegebenen Freiheitsgrade in der linearen Optimierung auszunutzen.
Angenommen mit $\alpha_1, \ldots, \alpha_p$ sind alle Primimplikanten der Produktübersichtsformel POF gegeben. Mit $M_1, \ldots, M_p$ seien die zugehörigen Basen gegeben. Aus den Primimplikanten werden zwei Matrizen $A^0, A^1 \in \mathbb{R}^{m \times p}$ abgeleitet. Die Matrix A^0 enthält spaltenweise die aus den α_i resultierenden *Minimalvarianten*

$$\alpha_i^{min} : C \to \{0,1\},\ {\alpha_i^{min}}_{|(C \setminus M_i)} \equiv 0,\ {\alpha_i^{min}}_{|M_i} \equiv \alpha_i. \tag{55}$$

Analog bezeichnet A^1 die Matrix, die spaltenweise die *Maximalvarianten*

$$\alpha_i^{max} : C \to \{0,1\},\ {\alpha_i^{max}}_{|(C \setminus M_i)} \equiv 1,\ {\alpha_i^{max}}_{|M_i} \equiv \alpha_i \tag{56}$$

enthält.

Satz 9
Die Code-Intervalle (l, u) sind genau dann konsistent, wenn

$$\begin{pmatrix} \mathbf{1}^t \\ -\mathbf{1}^t \\ A^0 \\ -A^1 \end{pmatrix} x \leq \begin{pmatrix} 1 \\ -1 \\ u \\ -l \end{pmatrix},\ x \geq \mathbf{0} \tag{57}$$

eine Lösung besitzt.
Ist x eine Lösung, so kann für jede Code-Statistik $b \in \mathbb{R}^m$ mit

$$A^0 x \leq b \leq A^1 x \tag{58}$$

eine Variantenverteilung $\tilde{x}$ mit $A\tilde{x} = b$ in $\mathcal{O}(m + k)$ bestimmt werden. Dabei ist k die Anzahl der echt positiven x_i, deren zugehörige Implikanten als bekannt vorausgesetzt werden.

Beweis. Zunächst sei (l, u) konsistent. D. h. es existiere ein

$$\tilde{x} \in \mathbb{R}^n \text{ mit } l \leq A\tilde{x} \leq u,\ \sum \tilde{x}_i = 1,\ \tilde{x} \geq \mathbf{0}.$$

Im Folgenden wird ein $x \in \mathbb{R}^p$ generiert, welches (57) erfüllt. Zunächst wird x mit $x = \mathbf{0}$ initialisiert. Anschließend wird für jedes $\tilde{x}_i > 0$ mit zugehöriger Spalte a_i eine Spalte

$$\tilde{a}_j := \begin{pmatrix} 1 \\ -1 \\ a_0 \\ -a_1 \end{pmatrix} \text{ der Matrix } \begin{pmatrix} \mathbf{1}^t \\ -\mathbf{1}^t \\ \tilde{A}^0 \\ -\tilde{A}^1 \end{pmatrix}$$

mit $a_0 \leq a_i \leq a_1$ gewählt. Dies ist möglich, da jede Variante eine Erweiterung von mindestens einer Primimplikante ist. Der Vektor x wird entsprechend in der j-ten Koordinate erhöht, d. h. $x_j = x_j + \tilde{x_i}$. Das resultierende x erfüllt (57).

Umgekehrt sei x eine Lösung von (57). Es wird im Weiteren ein Algorithmus angegeben, der aus x und jedem $b \in \mathbb{R}^m$, das (58) erfüllt, eine Variantenverteilung $\tilde{x}$ mit $A\tilde{x} = b$ in $O(m+k)$ generiert. Um die Konsistenz von (l, u) nachzuweisen, kann abschließend argumentiert werden, dass wegen $A^0x \leq A^1x$ und $A^0x \leq u,\ l \leq A^1x$ tatsächlich ein $b \in \mathbb{R}^m$ mit $l \leq b \leq u$ existiert, das (58) erfüllt.
Sei also ein beliebiges b gegeben, welches (58) erfüllt. Zu den Einträgen $x_i > 0$ werden die zugehörigen Implikanten α_i mit den Basen M_i betrachtet. Aus diesen Implikanten werden die Varianten generiert, die in der zu definierenden Variantenverteilung $\tilde{x}$ eine echt positive Häufigkeit besitzen. Die Varianten, die auf Basis einer Implikante α_i generiert werden, seien durch $\beta^i_1, \ldots, \beta^i_{m_i}$ gegeben. Die Häufigkeiten $\tilde{x}(\beta^i_j) > 0$ werden so definiert, dass

$$\sum_{j=1}^{m_i} \tilde{x}(\beta^i_j) = x_i$$

gilt. Dadurch ist $\sum \tilde{x}_i = 1$ sichergestellt. Die Generierung der Varianten β^i_j und die Wahl der passenden Häufigkeiten $\tilde{x}(\beta^i_j) > 0$ beschreibt Algorithmus 4:
In Zeile 2 wird mit $d \geq 0$ der Vektor berechnet, der die Differenz zu b angibt, wenn die Häufigkeiten von x übernommen und aus den zugehörigen Implikanten die Minimalvarianten generiert würden. Die Aufgabe besteht nun darin, die „Don't Cares“ der Implikanten so zu belegen, dass für die Differenz $d = \mathbf{0}$ gilt. Gegebenenfalls besteht dabei die Notwendigkeit aus einer Implikante mehrere Varianten zu generieren.
Die Implikanten α_i werden nacheinander durchlaufen. Gilt schon $d = \mathbf{0}$ so wird in Zeile 4 die Minimalvariante der Implikante generiert und entsprechend der Implikante gewichtet.
Nachfolgend wird eine Iteration für den Fall betrachtet, dass $d \neq \mathbf{0}$ gilt. In Zeile 6 wird mit δ die Zahl initialisiert, die angibt, welche Häufigkeit noch auf die Varianten der Implikante α_i zu verteilen ist. In Zeile 7 wird eine Permutation ϕ der Indizes $1, \ldots, m$ berechnet, um dann über die Vektoreinträge aus d von Klein nach Groß zu iterieren. Wird bei dieser Iteration ein Eintrag $d_{\phi(j)} > 0$ gefunden, so wird eine Variante β erzeugt. Dabei gilt für Codes $c_{\phi(j)}, c_{\phi(h)} \in C \setminus M_i$ stets

$$d_{\phi(h)} = 0 \text{ für } h < j \text{ bzw. } d_{\phi(j)} \leq d_{\phi(h)} \text{ für } j \leq h. \qquad (*)$$

Algorithmus 4 : Variantengenerierung aus gewichteten Implikanten

Input : alle Tripel (x_i, α_i, M_i) für $x_i > 0$, x erfüllt das System (57), α_i ist zugehörige Implikante mit Basis M_i ; Code-Statistik $b \in \mathbb{R}^m$, die (58) erfüllt

Output : alle Tupel $(\tilde{x}_i, \beta_i)$ für $\tilde{x}_i > 0$, $\tilde{x}$ ist Variantenverteilung mit $A\tilde{x} = b$, β_i ist zugehörige Variante

```
1  returnList← ∅;
2  d ← (b − A^0 x);
3  foreach (x_i, α_i, M_i) do
4      if d = 0 then returnList.Add(x_i, α_i^min) ;
5      else
6          δ ← x_i;
7          Berechne Permutation φ, sodass d_φ(j) ≤ d_φ(j') für j < j';
8          for j=1,...,m do
9              if d_φ(j) > 0 then
10                 w ← min(d_φ(j), δ);
11                 for h=1,...,m do
12                     if c_φ(h) ∈ M_i then β(c_φ(h)) := α_i(c_φ(h)) ;
13                     else
14                         if h < j then β(c_φ(h)) := 0;
15                         else
16                             β(c_φ(h)) := 1;
17                             d_φ(h) ← (d_φ(h) − w);
18                         end
19                     end
20                 end
21                 returnList.Add(w, β);
22                 δ ← (δ − w);
23                 if (δ = 0) then break;
24             end
25         end
26     end
27 end
28 return returnList;
```

Diese zentrale Eigenschaft des Algorithmus ist durch die weiteren Schritte begründet. In Zeile 10 wird mit $w := \min(d_{\phi(j)}, \delta)$ die Häufigkeit der Variante β festgelegt, wobei β in der darauf folgenden for-Schleife generiert wird. Die Variante β stellt eine Erweiterung von α_i auf C dar. Für Codes $c_{\phi(h)} \in C \setminus M_i$ wird $\beta(c_{\phi(h)}) := 1$ genau im Fall $j \leq h$ definiert. Durch die Variante β werden also alle $d_{\phi(h)}$ mit $c_{\phi(h)} \in C \setminus M_i$ und $j \leq h$ um w reduziert. Somit bleibt die Eigenschaft $(*)$ bestehen. Wegen $w = \min(d_{\phi(j)}, \delta)$, also $w \leq d_{\phi(j)}$, bleibt $d \geq \mathbf{0}$ erhalten. Im Falle von $w = \delta$ wird eine letzte Variante β zur Implikante α_i generiert (Zeile 22 und 23).
Die Generierung der Varianten folgt somit dem Grundsatz, frei wählbare Codes $c_{\phi(h)} \in C \setminus M_i$ im Fall $d_{\phi(h)} > 0$ stets zu nutzen, um $d_{\phi(h)}$ zu reduzieren. Wegen $b \leq A^1 x$ ist $d \leq \mathbf{0}$ möglich und $d = \mathbf{0}$ wird vom Algorithmus erreicht. □

Beobachtung 7 (Auf Primimplikanten basierende Konsistenzrechnung)
Die Code-Intervalle (l, u) sind genau dann konsistent, wenn das Optimum des linearen Programms

$$\begin{aligned} \min \quad & \begin{pmatrix} \mathbf{0}^t & \mathbf{1}^t & \mathbf{1}^t \end{pmatrix} \begin{pmatrix} x \\ z^- \\ z^+ \end{pmatrix} \\ \text{s.th.} \quad & \begin{pmatrix} \mathbf{1}^t & \mathbf{0}^t & \mathbf{0}^t \\ -\mathbf{1}^t & \mathbf{0}^t & \mathbf{0}^t \\ A^0 & -I_m & \mathbf{0} \\ -A^1 & \mathbf{0} & -I_m \end{pmatrix} \begin{pmatrix} x \\ z^- \\ z^+ \end{pmatrix} \leq \begin{pmatrix} 1 \\ -1 \\ u \\ -l \end{pmatrix}, \qquad (K^{\text{Impl}}) \\ & x, z^-, z^+ \geq \mathbf{0}, \end{aligned}$$

durch 0 gegeben ist.

Das Lineare Programm (K^{Impl}) ist der Ausgangspunkt für eine *auf Primimplikanten basierende Konsistenzrechnung*. Der nachfolgende Satz liefert die Grundlage, um mit dem Einsatz von Implikanten optimale Variantenverteilungen für beliebige lineare Zielfunktionen $c \in \mathbb{R}^n$ zu berechnen. In (K^{Impl}) hat eine Spalte

$$\begin{pmatrix} 1 \\ -1 \\ a_0 \\ -a_1 \end{pmatrix} \text{ der Matrix } \begin{pmatrix} \mathbf{1}^t \\ -\mathbf{1}^t \\ A^0 \\ -A^1 \end{pmatrix}$$

die Eigenschaft, dass alle Belegungen a mit $a_0 \leq a \leq a_1$ die Produktübersichtsformel POF erfüllen. Des Weiteren existiert zu jeder Variante a der Produktübersichtsformel POF eine Spalte mit $a_0 \leq a \leq a_1$. Diese beiden Voraussetzungen sind für beliebige Zielfunktionen $c \in \mathbb{R}^n$ nicht ausreichend.

Satz 10

Die Matrix A enthalte genau die Varianten der Produktübersichtsformel POF, wobei Wiederholungen erlaubt sind. Aus einer optimalen Lösung des linearen Programms

$$\begin{aligned} &\min && \tilde{c}^t x \\ &\text{s.th.} && \\ & && \begin{pmatrix} \mathbf{1}^t \\ -\mathbf{1}^t \\ \tilde{A}^0 \\ -\tilde{A}^1 \end{pmatrix} x \leq \begin{pmatrix} 1 \\ -1 \\ u \\ -l \end{pmatrix}, \qquad (P^{\text{Impl}}) \\ & && x \geq \mathbf{0}, \end{aligned}$$

lässt sich mit Algorithmus 4 eine für das lineare Programm

$$\begin{aligned} &\min && c^t x \\ &\text{s.th.} && \\ & && \begin{pmatrix} \mathbf{1}^t \\ -\mathbf{1}^t \\ A \\ -A \end{pmatrix} x \leq \begin{pmatrix} 1 \\ -1 \\ u \\ -l \end{pmatrix}, \qquad (P) \\ & && x \geq \mathbf{0}, \end{aligned}$$

optimale Variantenverteilung x mit selbem Zielfunktionswert ableiten, falls Folgendes gilt:

(I) Ist

$$\tilde{a}_i := \begin{pmatrix} 1 \\ -1 \\ a_0 \\ -a_1 \end{pmatrix} \text{ eine Spalte der Matrix } \tilde{\mathbf{A}} := \begin{pmatrix} \mathbf{1}^t \\ -\mathbf{1}^t \\ \tilde{A}^0 \\ -\tilde{A}^1 \end{pmatrix}$$

mit Zielfunktionskoeffizient $\tilde{c}_i$, so ist jede Belegung a_j mit $a_0 \leq a_j \leq a_1$ eine Variante der Produktübersichtsformel POF und es existiert eine zugehörige Spalte in (P), die den selben Zielfunktionskoeffizienten $c_j = \tilde{c}_i$ hat.

(II) Ist a_j eine zulässige Variante der Produktübersichtsformel POF, für die eine zugehörige Spalte in (P) den Zielfunktionskoeffizienten c_j hat, so existiert eine Spalte $\tilde{a}_i$ in (P^{Impl}), die $a_0 \leq a_j \leq a_1$ erfüllt und die den selben Zielfunktionskoeffizienten $\tilde{c}_i = c_j$ hat.

Beweis. Den Beweis von Satz 9 berücksichtigend leiten sich die Aussagen ab:

- Die Eigenschaft (I) garantiert, dass mit Algorithmus 4 eine zulässige Lösung von (P^{Impl}) in eine zulässige von (P) mit identischem Zielfunktionswert transformierbar ist.

- Die Eigenschaft (II) garantiert, dass jeder Zielfunktionswert, der in (P) auftritt, auch in (P^{Impl}) auftritt.

□

Beobachtung 8 (Auf Primimplikanten basierende Intervallrechnung)
Es wird gezeigt, wie die Matrix $\tilde{\mathbf{A}}$ aus (P^{Impl}) und die Zielfunktion $\tilde{c}^t$ aus (P^{Impl}) für eine *auf Primimplikanten basierende Intervallrechnung* definiert werden können. Für den Code d sei das induzierte Intervall zu bestimmen. Mit $C = \{c_1, \ldots, c_m\}$ sei für jede Variante $\beta : C \cup \{d\} \to \{0, 1\}$ der Produktübersichtsformel POF eine Primimplikante α von POF mit Basis M und $\alpha = \beta_{|M}$ gegeben. Derartige Primimplikanten können mit dem Algorithmus 2 bestimmt werden. Die Matrix $\tilde{\mathbf{A}}$ entstehe dadurch, dass für die Primimplikanten α die Minimal- und Maximalvarianten $\alpha^{min}, \alpha^{max} : C \to \{0, 1\}$ abgeleitet und als jeweils eine Spalte in $\tilde{\mathbf{A}}$ eingetragen werden. Der zu definierende Zielfunktionskoeffizient für eine eingetragene Spalte ist von $\beta(d)$ und von der Intervallschranke abhängig, die berechnet werden soll. Bei der Berechnung der Oberschranke von d wird einer Spalte der Zielfunktionskoeffizient 0 im Fall $\beta(d) = 1$ und der Zielfunktionskoeffizient 1 im Fall $\beta(d) = 0$ zugewiesen.
Das so definierte lineare Programm (P^{Impl}) erfüllt die Voraussetzungen, um nach Satz 10 aus einer optimalen Lösung eine optimale Lösung für das lineare Programm

$$\begin{aligned} \min \quad & \begin{pmatrix} \mathbf{0^t} & \mathbf{1}^t \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \\ \text{s.th.} \quad & \begin{pmatrix} \mathbf{1}^t & \mathbf{1}^t \\ -\mathbf{1}^t & -\mathbf{1}^t \\ A_d & A_{\neg d} \\ -A_d & -A_{\neg d} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} \leq \begin{pmatrix} 1 \\ -1 \\ u \\ -l \end{pmatrix}, \qquad (P) \\ & x_0,\ x_1 \geq \mathbf{0}. \end{aligned}$$

abzuleiten. Dabei sind die Matrizen A_d und $A_{\neg d}$ wie in Beobachtung 4 definiert; das lineare Programm (P) hat die von den Code-Intervallen (l, u) induzierte Oberschranke für d als Optimum.

Weder in der Konsistenzrechnung noch in der Intervallrechnung kann die Matrix $\tilde{\mathbf{A}}$ für nützliche Probleminstanzen mit all ihren Spalten dargestellt werden. Es ist jedoch

ausreichend, mittels Column Generation eine Teilmenge der Spalten so zu bestimmen, dass die echt positiv gewichteten Spalten einer optimalen Ecke enthalten sind. Für die auf Primimplikanten basierende Optimierung wird nun das zur Ungleichung (51) äquivalente Kriterium aufgestellt, welches die zu generierenden Spalten erfüllen müssen. Das zum linearen Programm (P^{Impl}) zugehörige Dualitätstheorem ist in der Form von Korollar 1 gegeben. In einer Iteration von Column Generation wird somit zu einer Indexmenge $I \subseteq \{1, \ldots, p\}$ die optimale Lösung x von

$$\begin{aligned} &\min && \tilde{c_I}^t x \\ &\text{s.th.} \\ &&& \begin{pmatrix} \mathbf{1}^t \\ -\mathbf{1}^t \\ \tilde{A}_I^0 \\ -\tilde{A}_I^1 \end{pmatrix} x \leq \begin{pmatrix} 1 \\ -1 \\ u \\ -l \end{pmatrix}, \qquad (P_I^{\text{Impl}}) \\ &&& x \geq \mathbf{0}, \end{aligned}$$

und die optimale Lösung $y^t = \begin{pmatrix} \omega_0 & \omega_1 & y_0^t & y_1^t \end{pmatrix} \in \mathbb{R}^{2m+2}$ von

$$\begin{aligned} &\max && \begin{pmatrix} 1 & -1 & u^t & -l^t \end{pmatrix} \begin{pmatrix} \omega_0 \\ \omega_1 \\ y_0 \\ y_1 \end{pmatrix} \\ &\text{s.th.} \\ &&& \begin{pmatrix} \mathbf{1} & -\mathbf{1} & \tilde{A}_I^{0\,t} & -\tilde{A}_I^{1\,t} \end{pmatrix} \begin{pmatrix} \omega_0 \\ \omega_1 \\ y_0 \\ y_1 \end{pmatrix} \leq \tilde{c}_I \qquad (D_I^{\text{Impl}}) \\ &&& \omega_0,\ \omega_1 \leq 0,\ y_0, y_1 \leq \mathbf{0}, \end{aligned}$$

berechnet. Analog zur Gleichungskette (50) ist ein Kriterium für

$$min(P_I^{\text{Impl}}) = min(P^{\text{Impl}}) \tag{59}$$

aus dem Dualitätstheorem ableitbar: Existiert keine Spalte

$$\tilde{a_i} := \begin{pmatrix} 1 \\ -1 \\ a_0 \\ -a_1 \end{pmatrix} \text{ der Matrix } \tilde{\mathbf{A}} = \begin{pmatrix} \mathbf{1}^t \\ -\mathbf{1}^t \\ \tilde{A}^0 \\ -\tilde{A}^1 \end{pmatrix},$$

die

$$\omega_0 - \omega_1 + a_0^t y_0 - a_1^t y_1 > \tilde{c}_i \tag{60}$$

erfüllt, so gilt (59). Da eine Spalte $\tilde{a}_i$ der Matrix $\tilde{\mathbf{A}}$ die Eigenschaft (I) aus Satz 10 erfüllen muss, stellt (60) jedoch ein nicht zu überprüfendes Kriterium dar.

Satz 11
Es seien lineare Programme (P^{Impl}) und (P) gemäß Satz 10 gegeben, die die Eigenschaften (I) und (II) erfüllen. Des Weiteren sei $y^t = \begin{pmatrix}\omega_0 & \omega_1 & y_0^t & y_1^t\end{pmatrix} \in \mathbb{R}^{2m+2}$ die optimale duale Lösung von (P_I^{Impl}) für eine Indexmenge $I \subseteq \{1, \ldots, p\}$. Existiert keine Variante a der Produktübersichtsformel POF mit

$$\omega_0 - \omega_1 + a^t(y_0 - y_1) > \tilde{c}_i, \tag{61}$$

so gilt $min(P_I^{\text{Impl}}) = min(P^{\text{Impl}})$. Dabei ist $\tilde{c}_i$ der minimale Zielfunktionskoeffizient, so dass eine Spalte $\tilde{a}_i$ aus $\tilde{\mathbf{A}}$ mit $a_0 \leq a \leq a_1$ existiert.

Beweis. Gibt es keine Variante a der Produktübersichtsformel POF, die (61) erfüllt, so wäre mit keiner Spalte aus (P) eine bessere Basislösung möglich. Es gilt somit

$$min(P_I^{\text{Impl}}) \leq min(P) = min(P^{\text{Impl}}).$$

□

Eine Möglichkeit, das Maximum der linken Seite von Ungleichung (60) zu bestimmen, ist nicht bekannt. Für die Ungleichung (61) ergibt sich hingegen eine PBO-Instanz. So ist gemäß der Beobachtung 8, im Falle der Berechnung der Oberschranke eines Codes d, eine Variante $\beta : \{c_1, \ldots, c_m, d\} \rightarrow \{0, 1\}$ der Produktübersichtsformel POF zu berechnen, die für $\tilde{y} = y_0 - y_1$ die Zielfunktion

$$z(\beta) = \tilde{c}_i \beta(d) + \sum_j \tilde{y}_j \beta(c_j) \tag{62}$$

maximiert. Ist der Zielfunktionswert größer als $1 - \omega_0 + \omega_1$, so ist auf Basis von β eine neue Spalte für $\tilde{\mathbf{A}}_I$ zu generieren. Die zur Schrittweite relative Annäherung an das Optimum von (P^{Impl}), die durch eine solche Spalte ermöglicht wird, ist mindestens so groß, wie die maximale relative Annäherung, die durch eine Spalte in (P) möglich wäre.

4.5 Automatisierte Reparatur von Inkonsistenzen

Das Erstellen einer konsistenten Prognose von Code-Intervallen setzt genaue Kenntnisse über die Zusammenhänge in der Produktübersicht voraus. Dieses auf Erfahrung basierende Wissen ist insbesondere bei der Prognose von Sonderausstattungen für neue

Baureihen noch nicht vorhanden. Folglich sind die erstellten Prognosen mit hoher Wahrscheinlichkeit inkonsistent zur Produktübersicht.
Um die globale Intervallrechnung im Falle einer inkonsistenten Prognose anwenden zu können, muss gegebenenfalls zunächst eine vorliegende Inkonsistenz repariert werden. Dazu sind die prognostizierten Code-Intervalle zu verschieben bzw. aufzuweiten. Wie dies unter einer Zielfunktion höherer Ordnung möglich ist, wird in diesem Abschnitt vorgestellt.

Angenommen mit dem globalen Verfahren der Konsistenzrechnung wird durch das Berechnen einer optimalen Lösung $\begin{pmatrix} x^t & z^{-t} & z^{+t} \end{pmatrix}$ von (K^{Impl}) die Inkonsistenz einer intervallbasierten Prognose

$$(l^p, u^p) \in [0,1]^m \times [0,1]^m$$

nachgewiesen, d. h. $z^- \neq \mathbf{0}$ oder $z^+ \neq \mathbf{0}$. Es werden die konsistenten Code-Intervalle $(l^{\mathrm{alt}}, u^{\mathrm{alt}})$ betrachtet, die sich aus x ableiten, also

$$l_i^{\mathrm{alt}} := \min(l_i^p, t_i),\ u_i^{\mathrm{alt}} := \max(u_i^p, s_i), \quad s := A^0 x,\ t := A^1 x.$$

Als eine Folge von Satz 9 minimieren sie unter allen konsistenten Code-Intervallen (l, u) den linearen Abstand d_1 zu (l^p, u^p),

$$d_1\Big((l^p, u^p), (l, u)\Big) := \sum_i \Big(|l_i^p - l_i| + |u_i^p - u_i|\Big). \tag{63}$$

Beliebige Code-Intervalle, die den linearen Abstand minimieren, stellen jedoch noch keine geeignete Alternative zu der inkonsistenten Prognose dar. So ist davon auszugehen, dass die Alternative $(l^{\mathrm{alt}}, u^{\mathrm{alt}})$ die Inkonsistenz der Prognose (l^p, u^p) ungleichmäßig auf die verschiedenen Codes verteilt und somit die Varianz im Ausdruck $|l_i^p - l_i^{\mathrm{alt}}| + |u_i^p - u_i^{\mathrm{alt}}|$ größer als nötig ist. Ist die Last der Inkonsistenz ungleichmäßig auf die Codes verteilt, so werden einige Codes der alternativen Prognose stark unter- bzw. stark überrepräsentiert.

Es wird nun ein Verfahren vorgeschlagen, mit dem (beispielsweise) konsistente Code-Intervalle $(l^{\mathrm{alt}}, u^{\mathrm{alt}})$ berechnet werden können, die, unter allen zur Produktübersichtsformel POF konsistenten Code-Intervallen, den quadratischen Abstand

$$d_2\Big((l^p, u^p), (l, u)\Big) := \sum_i \Big((l_i^p - l_i)^2 + (u_i^p - u_i)^2\Big) \tag{64}$$

zur inkonsistenten Prognose (l^p, u^p) minimieren. Der dazu beschriebene Algorithmus ist eine Erweiterung der auf Column Generation basierenden Konsistenzrechnung.

Zunächst folgt eine zentrale Beobachtung. Dazu sei mit $(K^{\text{Impl}}_{(l,u)})$ das primale Programm gegeben, welches im Falle eines Optimums bei 0 die Konsistenz von (l,u) nachweist (vgl. Unterabschnitt 4.4.5):

$$\begin{aligned} \min \quad & \begin{pmatrix} \mathbf{0}^t & \mathbf{1}^t & \mathbf{1}^t \end{pmatrix} \begin{pmatrix} x \\ z^- \\ z^+ \end{pmatrix} \\ \text{s.th.} \quad & \begin{pmatrix} \mathbf{1}^t & \mathbf{0}^t & \mathbf{0}^t \\ -\mathbf{1}^t & \mathbf{0}^t & \mathbf{0}^t \\ A^0 & -I_m & \mathbf{0} \\ -A^1 & \mathbf{0} & -I_m \end{pmatrix} \begin{pmatrix} x \\ z^- \\ z^+ \end{pmatrix} \leq \begin{pmatrix} 1 \\ -1 \\ u \\ -l \end{pmatrix}, \qquad (K^{\text{Impl}}_{(l,u)}) \\ & x, z^-, z^+ \geq \mathbf{0}, \end{aligned}$$

Das zugehörige duale Programm $(D^{\text{Impl}}_{(l,u)})$ ist durch

$$\begin{aligned} \max \quad & \begin{pmatrix} 1 & -1 & u^t & -l^t \end{pmatrix} \begin{pmatrix} \omega_0 \\ \omega_1 \\ y_0 \\ y_1 \end{pmatrix} \\ \text{s.th.} \quad & \begin{pmatrix} \mathbf{1} & -\mathbf{1} & A^{0t} & -A^{1t} \\ \mathbf{0} & \mathbf{0} & -I_m & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & -I_m \end{pmatrix} \begin{pmatrix} \omega_0 \\ \omega_1 \\ y_0 \\ y_1 \end{pmatrix} \leq \begin{pmatrix} \mathbf{0} \\ \mathbf{1} \\ \mathbf{1} \end{pmatrix} \qquad (D^{\text{Impl}}_{(l,u)}) \\ & \omega_0, \omega_1 \leq 0, \ y_0, y_1 \leq \mathbf{0}, \end{aligned}$$

gegeben.

Beobachtung 9
Für jedes duale Programm $(D^{\text{Impl}}_{(l,u)})$ ist die Menge der zulässigen Lösungen

$$Y := \left\{ y^t = \begin{pmatrix} \omega_0 & \omega_1 & y_0^t & y_1^t \end{pmatrix} \leq \mathbf{0} \;\middle|\; (\omega_0 - \omega_1)\mathbf{1} + y_0^t A^0 - y_1^t A^1 \leq \mathbf{0}, \ -y_0, -y_1 \leq \mathbf{1} \right\}$$

identisch, also unabhängig von (l,u). Aufgrund des Dualitätstheorems in der Form von Korollar 1 ist die Menge der konsistenten Code-Intervalle (l,u) durch

$$S \ := \ \left\{ (l,u) \mid (K^{\text{Impl}}_{(l,u)}) \text{ hat das Minimum } 0 \right\}$$

$$
\begin{aligned}
&= \Big\{(l,u) \mid (D^{\text{Impl}}_{(l,u)}) \text{ hat das Maximum } 0\Big\} \\
&= \Big\{(l,u) \mid b^t := \begin{pmatrix} 1 & -1 & u^t & -l^t \end{pmatrix},\ \forall y \in Y:\ b^t y \leq 0,\ \exists \tilde{y} \in Y:\ b^t \tilde{y} = 0\Big\}
\end{aligned}
$$

gegeben.
Angenommen mit Column Generation wird für $(l^*, u^*) := (l^p, u^p)$ eine optimale Lösung $y^* \in Y$ von $(D^{\text{Impl}}_{(l^*,u^*)})$ berechnet, welche durch

$$b^{*t} y^* > 0,\ b^{*t} := \begin{pmatrix} 1 & -1 & u^{*t} & -l^{*t} \end{pmatrix}$$

die Inkonsistenz von (l^*, u^*) bestätigt. Dann stellt die duale Lösung $y^* \in Y$ nicht nur eine Begründung für $(l^*, u^*) \notin S$ dar, sie liefert mit

$$b^t y^* \leq 0,\ b^t := \begin{pmatrix} 1 & -1 & u^t & -l^t \end{pmatrix} \tag{65}$$

auch einen Constraint, den jedes $(l,u) \in S$ erfüllen muss.

Aus dieser Beobachtung leitet sich Algorithmus 5 ab, der konsistente Code-Intervalle berechnet, die einen minimalen quadratischen Abstand zu der inkonsistenten Prognose (l^p, u^p) haben. Als Input dient neben der inkonsistenten Prognose (l^p, u^p) die Klasse der

Algorithmus 5 : Konsistenz mit minimalem quadratischen Abstand

Input : Prognose (l^p, u^p), $\{(K^{\text{Impl}}_{(l,u)}) \mid (l,u)$ Code-Intervalle$\}$, $\epsilon \geq 0$
Output : Code-Intervalle (l^*, u^*), die (66) und (67) erfüllen

$Z \leftarrow \emptyset$;
$(l^*, u^*) \leftarrow (l^p, u^p)$;
while *true* **do**
  $y^* \leftarrow$`Dual(ColumnGen(`$(K^{Impl}_{(l^*,u^*)})$`))`;
  if $\Big(\begin{pmatrix} 1 & -1 & u^{*t} & -l^{*t} \end{pmatrix} y^* \leq \epsilon\Big)$ **then return** (l^*, u^*);
  $Z \leftarrow Z \cup \{y^*\}$;
  $(l^*, u^*) \leftarrow \arg\min \Big\{d_2\Big((l^p, u^p), (l,u)\Big) \mid \forall y \in Z : \begin{pmatrix} 1 & -1 & u^t & -l^t \end{pmatrix} y \leq 0\Big\}$;
end

primalen Programme $(K^{\text{Impl}}_{(l,u)})$, mit denen Code-Intervalle (l,u) auf Konsistenz überprüft werden sowie ein Abbruchkriterium $\epsilon \geq 0$, welches die Genauigkeit der Rechnung festlegt. Der Output ist durch Code-Intervalle (l^*, u^*) gegeben, die die folgenden zwei Eigenschaften aufweisen:

$$d_2\Big((l^p, u^p), (l^*, u^*)\Big) \leq \min_{(l,u) \in S} d_2\Big((l^p, u^p), (l,u)\Big) \tag{66}$$

$$\exists (l,u) \in S: \quad d_1\Big((l,u), (l^*, u^*)\Big) \leq \epsilon. \tag{67}$$

Um diese Eigenschaften einzusehen, werden die einzelnen Schritte im Algorithmus betrachtet:
Zunächst wird in der Zeile 1 die Menge Z mit der leeren Menge initialisiert. Zu Z werden iterativ duale Lösungen $y \in Y$ hinzugefügt. In Zeile 2 werden die Code-Intervalle (l^*, u^*) mit den originalen Code-Intervallen (l^p, u^p) initialisiert. Innerhalb der while-Schleife wird in Zeile 4 die Konsistenzrechnung für die aktuellen Code-Intervalle (l^*, u^*) aufgerufen. Die zugehörige duale Lösung wird in y^* abgelegt. Ist das Optimum kleiner als das Abbruchkriterium ϵ, so wird (l^*, u^*) als Lösung ausgegeben. Dies ist nach Definition der linearen Programme $(K^{\text{Impl}}_{(l^*,u^*)})$ genau dann der Fall, wenn die Eigenschaft (67) von (l^*, u^*) erfüllt wird. Wird die Eigenschaft (67) nicht von (l^*, u^*) erfüllt, so wird ein neues (l^*, u^*) berechnet. Die duale Lösung y^* zu den aktuellen Code-Intervallen (l^*, u^*) liefert mit (65) ein Kriterium für alle konsistenten $(l, u) \in S$ und wird in Zeile 6 der Menge Z hinzugefügt. In jeder Iteration gilt $Z \subseteq Y$, so dass für

$$T_Z := \Big\{(l,u) \mid \forall y \in Z: \ \begin{pmatrix} 1 & -1 & u^t & -l^t \end{pmatrix} y \leq 0\Big\}$$

in jeder Iteration $S \subseteq T_Z$ gilt. Mit der Zuweisung aus Zeile 7 gilt für die neu erzeugten Code-Intervalle (l^*, u^*) die Ungleichung (66):

$$d_2\Big((l^p,u^p),(l^*,u^*)\Big) = \min_{(l,u)\in T_Z} d_2\Big((l^p,u^p),(l,u)\Big) \leq \min_{(l,u)\in S} d_2\Big((l^p,u^p),(l,u)\Big).$$

Satz 12
Algorithmus 5 terminiert nach endlich vielen Iterationen.

Beweis. Zunächst wird begründet, dass die Menge Z in ihrem Wachstum beschränkt ist. Es sei $\mathbf{A} \in \mathbb{R}^{(2m+2)\times(n+2m)}$ die Gesamtmatrix aus dem primalen Programm $(K^{\text{Impl}}_{(l^*,u^*)})$ und $c \in \mathbb{R}^{(n+2m)}$ die zugehörige Zielfunktion. Zusätzlich seien mit $x^* \in \mathbb{R}^{(n+2m)}$ die primale und mit $y^* \in \mathbb{R}^{(2m+2)}$ die duale Lösung von $(K^{\text{Impl}}_{(l^*,u^*)})$ gegeben, die mit Column Generation in Zeile 4 berechnet werden. Der optimalen Lösung x^* ist eine optimale Basis

$$B \subseteq \{1, \ldots, n+2m\}, \ |B| = 2m+2$$

von $\mathbf{A}$ zugeordnet. Nach Unterabschnitt 4.4.4 gilt

$$y^* = c_B^t {\mathbf{A}_B}^{-1}$$

für die duale Lösung. Die duale Lösung y^* definiert sich also ausschließlich über die optimale Basis B und die Konstanten $\mathbf{A}$ und c^t. Da höchstens $\binom{n+2m}{2m+2}$ verschiedene Basen B von $\mathbf{A}$ existieren, ist die Menge Z in ihrem Wachstum beschränkt.
Es reicht nun zu zeigen, dass der Algorithmus sich in seiner letzten Iteration befindet,

falls ein y^* berechnet wird, welches zuvor bereits in Z aufgenommen wurde. Sei das berechnete y^* schon in Z enthalten. Dann war zuvor die Ungleichung

$$\begin{pmatrix} 1 & -1 & u^{*t} & -l^{*t} \end{pmatrix} y^* \leq 0$$

eine Nebenbedingung bei der Berechnung von (l^*, u^*). Neben $c^t x^* \geq 0$ gilt somit, als Folge des Dualitätstheorems, $c^t x^* = \begin{pmatrix} 1 & -1 & u^{*t} & -l^{*t} \end{pmatrix} y^* \leq 0$, d. h. $c^t x^* = 0$. □

Satz 13

Mit Algorithmus 5 werden implizit konsistente Code-Intervalle $(l^{*\text{alt}}, u^{*\text{alt}}) \in S$ berechnet, die

$$d_2\Big((l^p, u^p), (l^{*\text{alt}}, u^{*\text{alt}})\Big) \leq \min_{(l,u)\in S} d_2\Big((l^p, u^p), (l, u)\Big) + \epsilon$$

erfüllen.

Beweis. Es sei $\begin{pmatrix} x^t & z^{-t} & z^{+t} \end{pmatrix}$ die primale Lösung zu den Output-Intervallen (l^*, u^*). Es werden die konsistenten Code-Intervalle $(l^{*\text{alt}}, u^{*\text{alt}})$ definiert:

$$l_i^{*\text{alt}} := \min(l_i^*, t_i),\ u_i^{*\text{alt}} := \max(u_i^*, s_i), \quad s := A^0 x,\ t := A^1 x.$$

Sie erfüllen mit

$$d_1\Big((l^{*\text{alt}}, u^{*\text{alt}}), (l^*, u^*)\Big) \leq \epsilon.$$

die Ungleichung der Eigenschaft (67). Es folgt unter Anwendung der Dreiecksungleichung Δ für Metriken:

$$\begin{aligned} d_2\Big((l^p, u^p), (l^{*\text{alt}}, u^{*\text{alt}})\Big) \quad - \quad & \min_{(l,b)\in S} d_2\Big((l^p, u^p), (l, u)\Big) \\ \overset{(66)}{\leq} \quad & d_2\Big((l^p, u^p), (l^{*\text{alt}}, u^{*\text{alt}})\Big) - d_2\Big((l^p, u^p), (l^*, u^*)\Big) \\ \overset{\Delta}{\leq} \quad & d_2\Big((l^{*\text{alt}}, u^{*\text{alt}}), (l^*, u^*)\Big) \ \leq\ d_1\Big((l^{*\text{alt}}, u^{*\text{alt}}), (l^*, u^*)\Big) \ \leq\ \epsilon. \end{aligned}$$

□

Die Idee von Algorithmus 5 besteht darin, die inkonsistenten Code-Intervalle Stück für Stück anzupassen. Die hierfür in Zeile 7 aufgestellten Optimierungsproblemen mit quadratischer Zielfunktion und linearen Nebenbedingungen sind verhältnismäßig klein und können effizient mit CPLEX berechnet werden.

Grundsätzlich kann der Algorithmus 5 dazu verwendet werden, beliebige, durch den Anwender definierte Metriken, zu minimieren. Entscheidend ist, dass ein Verfahren existiert, welches die Optimierungsprobleme aus Zeile 7 löst. Mit welcher Geschwindigkeit der Algorithmus 5 ein nützliches $\epsilon \geq 0$ erreicht, wird im folgenden Abschnitt geklärt.

4.6 Numerische Ergebnisse

Um die Konsistenz von Code-Intervallen (l, u) zu einer Produktübersichtformel POF nachzuweisen, wurde im Rahmen dieser Arbeit ein auf Column Generation basierendes Verfahren implementiert, das das lineare Programm (K^{Impl}) aus Unterabschnitt 4.4.5 berechnet. Darüber hinaus ist das Verfahren angepasst wurden, um induzierte, minimale Code-Intervalle mit den linearen Programmen der Form (P^{Impl}) zu berechnen. Zusätzlich wurde die globale Konsistenzrechnung um die automatische Reparatur von zur Produktübersicht inkonsistenten Code-Intervallen nach Abschnitt 4.5 ergänzt.
Im Folgenden werden die numerischen Ergebnisse der globalen Konsistenzrechnung, der globalen Intervallrechnung und der automatischen Reparatur vorgestellt. Die Ergebnisse der globalen Intervallrechnung werden außerdem mit den Ergebnissen einer Implementierung des lokalen Verfahrens aus Abschnitt 4.3 verglichen.

4.6.1 Konsistenzrechnung

Die auf Primimplikanten basierende Konsistenzrechnung wurde an eigens generierten Testinstanzen untersucht. Alle Testinstanzen sind durch eine spezialisierte Produktübersichtsformel von POF_{PPM} und durch zur ihr konsistente Code-Intervalle gegeben. Somit ist für alle Instanzen bekannt, dass 0 das Optimum von (K^{Impl}) ist. Demnach stoppt die Konsistenzrechnung in jedem der generierten Fälle, sobald der Zielfunktionswert 0 erreicht ist. Das Terminieren des implementierten Verfahrens aufgrund der Ungleichung (61) wird in den nächsten beiden Unterabschnitten nachgewiesen, in denen die numerischen Ergebnisse zur Intervallrechnung und zur automatischen Reparatur von Inkonsistenzen beschrieben werden.
Das Generieren der konsistenten Code-Intervalle (l, u) für eine spezialisierte Produktübersichtsformel POF wurde wie folgt umgesetzt: Zunächst wurden 100.000 zufällige Modelle von POF mit dem SAT-Solver MiniSAT und den Solvereinstellungen aus Unterabschnitt 2.2.6 erzeugt. Anschließend wurde die Code-Statistik $b \in \mathbb{R}^m$ der generierten Varianten ausgezählt. Dabei wurden nur die m Codes berücksichtigt, die kundenwählbar sind und deren Belegung noch nicht durch die Spezialisierung fixiert ist. Aus der gewonnenen konsistenten Code-Statistik b wurden nun Code-Intervalle (l, u) für drei Instanzen unterschiedlicher Komplexität abgeleitet:

$$
\begin{aligned}
&\text{(ABS1-20)} \quad u_i := b_i - s_i t_i d, \quad l_i := b_i + (1 - s_i) t_i d, \quad s_i \in [0,1],\ t_i \in [0.05, 1],\ d = 20\%,\\
&\text{(ABS1)} \quad\ \ u_i := b_i - s_i d, \quad\ \ l_i := b_i + (1 - s_i) d, \quad\ \ s_i \in [0,1],\ d = 1\%,\\
&\text{(REL1)} \quad\ \ u_i := b_i - s d b_i, \quad l_i := b_i + (1 - s) d b_i, \quad s_i \in [0,1],\ d = 1\%.
\end{aligned}
$$

Die Code-Intervalle der ersten beiden Instanzen (ABS1-20) und (ABS1) besitzen eine absolute Größe von 1% bis 20% bzw. 1%. In der dritten Instanz (REL1) ist die Größe der Code-Intervalle von der jeweiligen Code-Häufigkeit in b abhängig und beträgt re-

Probleminstanz		**mit Primimpl.**		**ohne Primimpl.**	
Spezialisierung	**#Intervalle**	**#Iter**	**Zeit(s)**	**#Iter**	**Zeit(s)**
S205@BM+374@LD	650	128	1.530	142	1.562
S212@BM+374@LD	692	109	3.592	120	3.904
V205@BM+374@LD	456	58	0.313	52	0.234
V212@BM+374@LD	353	61	1.031	61	0.562
W205@BM+374@LD	677	124	2.093	128	2.233
W212@BM+374@LD	698	110	4.419	117	4.326
2050041@BM+374@LD	361	85	0.578	83	0.593
2120021@BM+374@LD	374	108	0.219	106	0.203
2122051@BM+374@LD	324	89	0.125	92	0.126
2462121@BM+531@LD	239	73	0.078	80	0.079
2050042@BM+537@LD	316	75	0.125	85	0.125
2120362@BM+537@LD	210	73	0.093	77	0.093
2050041@BM+543@LD	340	94	0.156	103	0.173
2462471@BM+571@LD	220	79	0.078	85	0.078
2050401@BM+581@LD	264	86	0.140	110	0.156
2221821@BM+705@LD	210	70	0.062	74	0.063
1760422@BM+839@LD	144	54	0.048	60	0.047
1760522@BM+839@LD	120	54	0.032	54	0.032
2050402@BM+839@LD	242	67	0.078	66	0.078
2221621@BM+CHINA@LD	129	48	0.110	52	0.078

Tabelle 9: Laufzeiten der Konsistenzrechnung für Instanzen der Klasse (ABS1-20)

lative 1%. Für alle Code-Intervalle wurde ein zufälliger Wert $s_i \in [0,1]$ bestimmt, der den Punkt b_i im Intervall $[l_i, u_i]$ fixiert. Für die Instanz (ABS1-20) wurde außerdem zu jedem Code ein zufälliger Parameter t_i gewählt, der die Größe des Intervalls auf einen Wert zwischen 1% und 20% festlegt.
Auf diese Weise wurden insgesamt 60 Instanzen zu 20 verschiedenen Produktübersichten generiert. Unter den Produktübersichten sind auch die Spezialisierungen der Granularität BM7/Markt aus den Kapiteln 2 und 3 vertreten. Auf ihr werden üblicherweise die Sonderausstattungsprognosen formuliert. Außerdem wurden Instanzen zur gröberen Granularität Typklasse/Markt generiert.

Alle Instanzen wurden einmal mit dem Einsatz von Primimplikanten und einmal ohne den Einsatz von Primimplikanten gelöst. Für die Konsistenzrechnungen ohne Primimplikanten wurden die Matrizen A^0 und A^1 in (K^{Impl}) mit vollständig spezifizierten Varianten statt mit Primimplikanten aufgefüllt. Die notwendigen Eigenschaften (I) und (II) der Matrizen A^0 und A^1 (vgl. Satz 10) bleiben bei diesem Vorgehen erhalten. Die Tabellen 9, 10 und 11 zeigen den Vergleich beider Ansätze für jeweils eine Klasse generierter Code-Intervalle. Neben der Laufzeit ist jeweils auch die Anzahl der benötigten Iterationen angegeben.

Probleminstanz		mit Primimpl.		ohne Primimpl.	
Spezialisierung	**#Intervalle**	**#Iter**	**Zeit(s)**	**#Iter**	**Zeit(s)**
S205@BM+374@LD	650	206	2.967	191	2.498
S212@BM+374@LD	692	190	6.168	188	5.839
V205@BM+374@LD	456	84	0.515	92	0.483
V212@BM+374@LD	353	108	1.796	106	0.953
W205@BM+374@LD	677	199	1.040	204	3.499
W212@BM+374@LD	698	170	6.964	188	7.713
2050041@BM+374@LD	361	119	0.562	138	0.624
2120021@BM+374@LD	374	139	0.406	158	0.453
2122051@BM+374@LD	324	119	0.250	134	0.281
2462121@BM+531@LD	239	87	0.124	96	0.140
2050042@BM+537@LD	316	123	0.296	185	0.484
2120362@BM+537@LD	210	101	0.157	96	0.156
2050041@BM+543@LD	340	155	0.421	153	0.406
2462471@BM+571@LD	220	113	0.141	114	0.156
2050401@BM+581@LD	264	124	0.234	170	0.344
2221821@BM+705@LD	210	100	0.125	106	0.125
1760422@BM+839@LD	144	79	0.078	80	0.078
1760522@BM+839@LD	120	66	0.046	59	0.046
2050402@BM+839@LD	242	92	0.156	114	0.203
2221621@BM+CHINA@LD	129	73	0.156	71	0.094

Tabelle 10: Laufzeiten der Konsistenzrechnung für Instanzen der Klasse (ABS1)

Für die Instanzen der Klassen (ABS1-20) und (ABS1) führt der Einsatz von Primimplikanten nur in wenigen Fällen zu einer schnelleren Konsistenzrechnung. Die im Durchschnitt leicht geringere Anzahl an Iterationen rechtfertigt hier nicht den Mehraufwand, der durch die Berechnung der Primimplikanten entsteht.
Erst im Falle der Instanzen (REL1) ist ein Grad an Komplexität erreicht, der den Nutzen einer auf Primimplikanten basierenden Optimierung signifikant nachweist. So führt hier die Verwendung von Primimplikanten zu teilweise erheblichen Verbesserungen in der Laufzeit. Beispielsweise konnte die Laufzeit für die Instanz der Produktübersichtsformel POF_{PPM} + S205@BM+374@LD von 142 Sekunden auf 8 Sekunden reduziert werden. Für die Instanz der Formel POF_{PPM} + 2050041@BM+374@LD verringerte sich die Laufzeit von 8,6 Sekunden auf 1,4 Sekunden.
Insgesamt fällt auf, dass die Instanzen der Granularität Typklasse/Markt stärker von den Primimplikanten profitieren. Eine Erklärung liefert die in Unterabschnitt 2.3.3 beschriebene Beobachtung, dass die Don't Cares in den unterschiedlichen BM7/Markt-Segmenten durch verschiedene Codes vertreten sind. Dadurch ist der relative Anteil von Codes, welche durch den Einsatz von Primimplikanten „entlastet“ werden, auf der Ebene Typklasse/Markt erhöht.

Probleminstanz		mit Primimpl.		ohne Primimpl.	
Spezialisierung	**#Intervalle**	**#Iter**	**Zeit(s)**	**#Iter**	**Zeit(s)**
S205@BM+374@LD	650	411	7.995	1484	142.759
S212@BM+374@LD	692	356	14.240	820	49.097
V205@BM+374@LD	456	174	1.186	508	3.780
V212@BM+374@LD	353	146	2.452	300	2.747
W205@BM+374@LD	677	497	16.051	1284	164.025
W212@BM+374@LD	698	380	19.036	946	91.970
2050041@BM+374@LD	361	251	1.405	522	8.866
2120021@BM+374@LD	374	248	1.124	331	1.546
2122051@BM+374@LD	324	182	0.547	302	1.202
2462121@BM+531@LD	239	129	0.219	365	1.046
2050042@BM+537@LD	316	448	2.905	471	3.654
2120362@BM+537@LD	210	149	0.297	174	0.343
2050041@BM+543@LD	340	275	1.156	466	4.138
2462471@BM+571@LD	220	169	0.312	347	0.827
2050401@BM+581@LD	264	202	0.485	294	0.749
2221821@BM+705@LD	210	136	0.312	175	0.297
1760422@BM+839@LD	144	106	0.124	143	0.171
1760522@BM+839@LD	120	75	0.078	94	0.094
2050402@BM+839@LD	242	147	0.344	274	0.843
2221621@BM+CHINA@LD	129	107	0.234	125	0.187

Tabelle 11: Laufzeiten der Konsistenzrechnung für Instanzen der Klasse (REL1)

Alle 60 Instanzen konnten innerhalb sehr nützlicher Laufzeiten gelöst werden. Es zeigt sich, dass die Konsistenzprüfung einer Sonderausstattungsprognose, welche üblicherweise auf der Ebene BM7/Markt erstellt wird und sich aus ca. 200 Code-Intervallen zusammensetzt, in unter einer Sekunde gerechnet werden kann. Die Ergebnisse zu den Testinstanzen der Granularität Typklasse/Markt belegen außerdem, dass eine schnelle Konsistenzrechnung mit weiteren statistischen Vorgaben möglich ist.

In Abbildung 11 ist die durch Column Generation typische Annäherung an das Optimum dargestellt. Im linken Graphen ist zu erkennen, dass ein Großteil der Iterationen dazu verwendet wird, den letzten kleinen Schritt hin zum Optimum zu bewältigen. Der rechte Graph hingegen verdeutlicht, dass mit der Annäherung an das Optimum die PBO-Instanzen der einzelnen Iterationen zusätzlich komplexer werden, da die Anzahl der Variablen in den Zielfunktionen zunimmt.

4.6.2 Intervallrechnung

Um die Implementierung der globalen, auf Primimplikanten basierenden Intervallrechnung zu testen, wurden die für die Konsistenzrechnung generierten Testinstanzen der Granularität BM7/Markt herangezogen. Für jeden Code bzw. für jede Variable in der

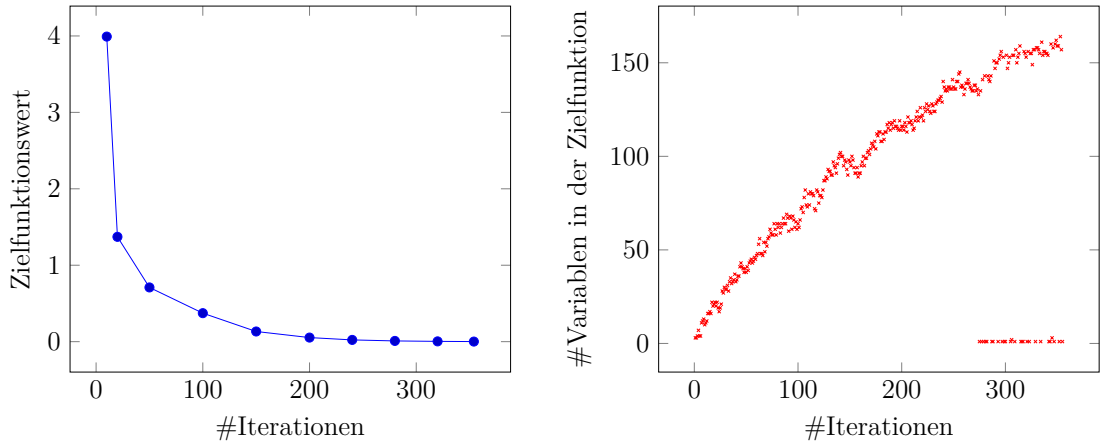

Abbildung 11: Konvergenzverhalten für die (REL1)-Instanz S212@BM+374@LD

Produktübersichtsformel wurde das durch die vorgegebenen Code-Intervalle induzierte Häufigkeitsintervall mit zwei linearen Programmen des Typs (P^{Impl}) ermittelt.
Dazu wurde für jede Testinstanz zunächst eine Konsistenzrechnung durchgeführt. Die dabei ermittelten Primimplikanten dienten als zulässige Startlösung für alle weiteren Optimierungsprobleme der Intervallrechnung. Zusätzlich wurden die im Zuge der Intervallrechnung ermittelten Primimplikanten für die Berechnungen der verbleibenden Intervallschranken beibehalten.
Tabelle 12 listet zeilenweise eine Instanz der Konsistenzrechnung, die Anzahl der vorgegebenen Code-Intervalle, die Anzahl N der Variablen der Produktübersichtsformel und die beobachtete Laufzeit zum Berechnen aller N induzierten Code-Intervalle auf. Für die Instanzen mit den praxisrelevanten Intervallgrößen, d. h. für (ABS1-20) und (ABS1), konnte die Gesamtheit aller induzierten Intervalle in jeweils unter 20 Sekunden berechnet werden. Im Durchschnitt lassen sich somit die Optimierungsprobleme der Intervallrechnung schneller lösen als die Optimierungsprobleme der Konsistenzrechnung. Dies ist damit zu begründen, dass bei der Intervallrechnung bereits eine zum Teil große Auswahl geeigneter Primimplikanten zur Verfügung steht. So war für eine Vielzahl induzierter Intervallschranken lediglich eine Umgewichtung der bereits bekannten Primimplikanten nötig, mit dem anschließenden Nachweis, dass keine „verbessernde“ Primimplikante existiert.

Im Folgenden wird der Informationsgewinn betrachtet, der mit der Berechnung aller Minimalintervalle generiert wurde. Für jede Testinstanz wird das Intervallsystem bezüglich der durchschnittlichen Intervallgröße vor und nach der Intervallrechnung bewertet. Dazu seien (l, u) die auf $C \subseteq Var(POF)$ vorgegebenen Intervalle und (l^*, u^*) die berechneten induzierten Intervalle auf $Var(POF)$, d. h.

$$(l, u) \xrightarrow[Var(POF)]{POF} (l^*, u^*).$$

Probleminstanz	**#Intervalle**	**#Variablen**	**Zeit(s)**
2050041@BM+374@LD_ABS1-20	361	498	6.84
2050041@BM+374@LD_ABS1	361	498	8.17
2050041@BM+374@LD_REL1	361	498	247.27
2120021@BM+374@LD_ABS1-20	374	1096	17.74
2120021@BM+374@LD_ABS1	374	1096	17.54
2120021@BM+374@LD_REL1	374	1096	56.26
2122051@BM+374@LD_ABS1-20	324	514	5.29
2122051@BM+374@LD_ABS1	324	514	5.67
2122051@BM+374@LD_REL1	324	514	34.82
2462121@BM+531@LD_ABS1-20	239	366	1.95
2462121@BM+531@LD_ABS1	239	366	2.22
2462121@BM+531@LD_REL1	239	366	4.57
2120362@BM+537@LD_ABS1-20	210	1023	12.26
2120362@BM+537@LD_ABS1	210	1023	12.44
2120362@BM+537@LD_REL1	210	1023	14.79
2050041@BM+543@LD_ABS1-20	340	510	7.20
2050041@BM+543@LD_ABS1	340	510	7.57
2050041@BM+543@LD_REL1	340	510	119.64
2462471@BM+571@LD_ABS1-20	220	389	2.12
2462471@BM+571@LD_ABS1	220	389	2.25
2462471@BM+571@LD_REL1	220	389	4.03
2050401@BM+581@LD_ABS1-20	264	542	10.51
2050401@BM+581@LD_ABS1	264	542	14.63
2050401@BM+581@LD_REL1	264	542	80.60
2221821@BM+705@LD_ABS1-20	210	320	1.78
2221821@BM+705@LD_ABS1	210	320	1.69
2221821@BM+705@LD_REL1	210	320	2.78
1760422@BM+839@LD_ABS1-20	144	231	0.86
1760422@BM+839@LD_ABS1	144	231	0.94
1760422@BM+839@LD_REL1	144	231	1.11
1760522@BM+839@LD_ABS1-20	120	204	0.62
1760522@BM+839@LD_ABS1	120	204	0.62
1760522@BM+839@LD_REL1	120	204	0.83
2050402@BM+839@LD_ABS1-20	242	358	2.75
2050402@BM+839@LD_ABS1	242	358	2.75
2050402@BM+839@LD_REL1	242	358	3.70
2221621@BM+CHINA@LD_ABS1-20	129	1223	27.89
2221621@BM+CHINA@LD_ABS1	129	1223	27.59
2221621@BM+CHINA@LD_REL1	129	1223	27.61

Tabelle 12: Laufzeiten zum Berechnen aller induzierten Intervalle

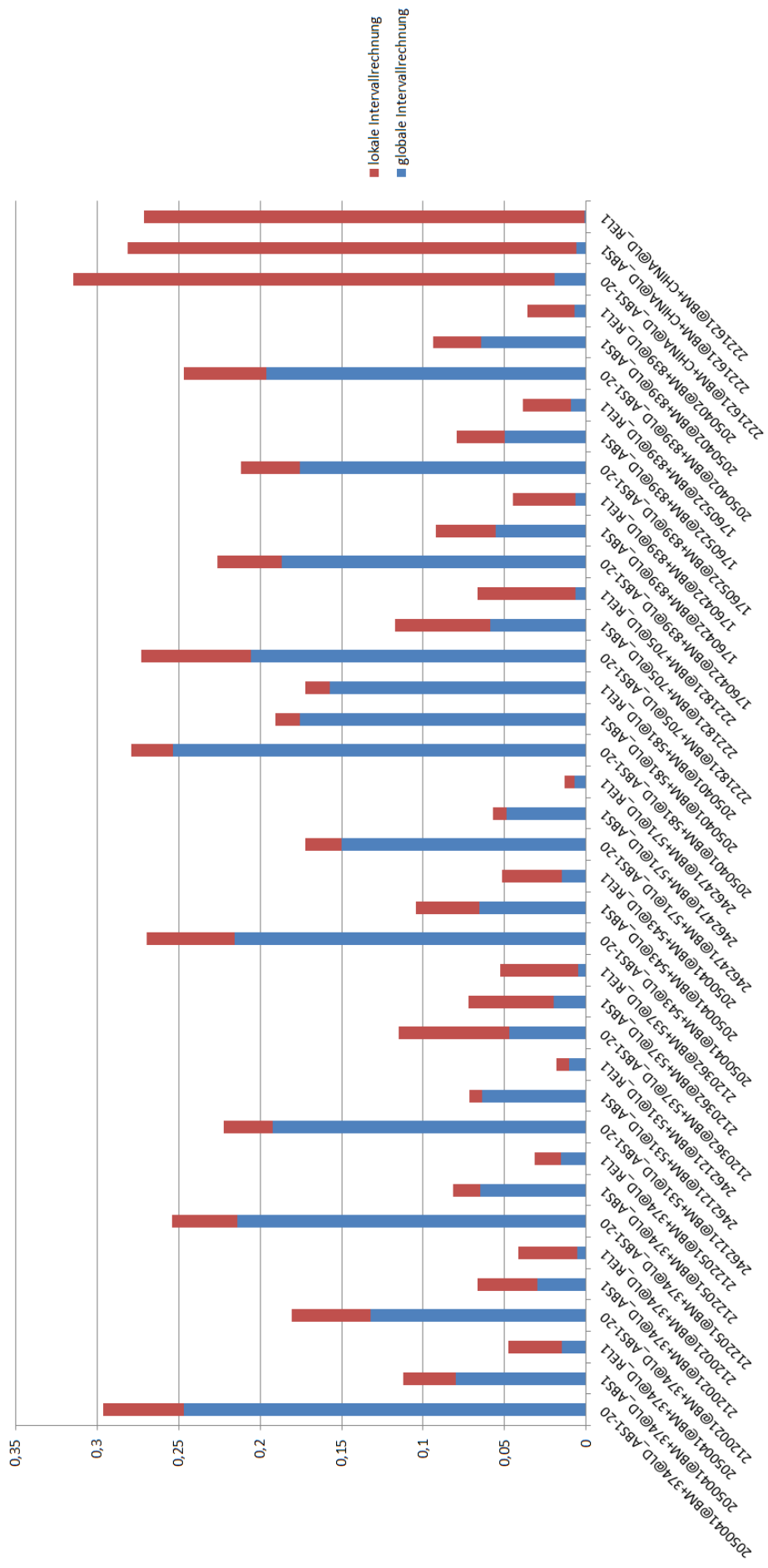

Abbildung 12: Relative Intervallgrößen nach Anwendung der Intervallrechnung

Im Balkendiagramm der Abbildung 12 ist mit

$$\begin{aligned} S &:= |Var(POF) \setminus C| + \sum_{c \in C} \Big(u(c) - l(c) \Big), \\ S^* &:= \sum_{c \in Var(POF)} \Big(u^*(c) - l^*(c) \Big), \end{aligned}$$

der Quotient

$$\sigma_{\text{global}} = \frac{S^*}{S} \tag{68}$$

dargestellt. Die Zahl $1 - \sigma_{\text{global}}$ beschreibt die durchschnittliche, relative Reduzierung in der Intervallgröße. Zum Vergleich des generierten Informationsgewinns zwischen globaler und lokaler Intervallrechnung sind außerdem die analog definierten Quotienten σ_{lokal} dargestellt, die auf Basis der lokalen Intervallrechnung nach Abschnitt 4.3 bestimmt wurden. In der Differenz $\sigma_{\text{lokal}} - \sigma_{\text{global}}$ zeigt sich, wie viel Prozent der induzierten Intervallgrößenreduzierung nicht mit dem lokalen Verfahren erklärbar ist – d. h. auf der Ebene der atmost-1-Constraints und der atleast-1-Constraints.
Im Fall der Produktübersichtsformel 2050401@BM+581@LD generiert das lokale Verfahren einen fast so großen Informationsgewinn wie das globale Verfahren. Für die Produktübersichtsformel 2221621@BM+CHINA@LD suggeriert das lokale Verfahren hingegen viel zu große Code-Intervalle. Somit ist keine zuverlässige Approximation der Minimalintervalle mit dem lokalen Verfahren aus Unterabschnitt 4.3 möglich.

Die Ergebnisse belegen, dass die vollständigen Auswirkungen einer Sonderausstattungsprognose (vgl. Beispiele 8 und 9) in nahezu Echtzeit berechenbar sind. Der erfolgte Machbarkeitsnachweis der globalen Intervallrechnung und die numerischen Ergebnisse der Konsistenzrechnung lassen weiter erwarten, dass die Einbindung einer Vielzahl zusätzlicher statistischer Vorgaben möglich ist. Neben der Anzahl der vorgegebenen Intervalle ist insbesondere auch die Größe der vorgegebenen Intervalle ein entscheidender Faktor, der die Konvergenzgeschwindigkeit beeinflusst.

4.6.3 Automatisierte Reparatur

In Abschnitt 4.5 wurde ein Verfahren zur Reparatur von inkonsistenten Prognosen vorgestellt. Im Weiteren wird das numerische Verhalten des dort beschriebenen Algorithmus 5 anhand einer mittelschweren Instanz aus der Konsistenzrechnung (vgl. Unterabschnitt 4.6.1) dargestellt. Auf Basis der konsistenten Instanz 2050041@BM+374@LD der Klasse (ABS1) wurden mehrere Testinstanzen mit inkonsistenten Code-Intervallen unterschiedlich starker Ausprägung erzeugt: Dazu wurden $x\%$ der Code-Intervalle zufällig ausgewählt und verschoben – die Mittelpunkte wurden unter Gleichverteilung im Intervall $[0, 1]$ neu platziert, ohne dabei die Intervallgrößen zu ändern.
Für die so erzeugten inkonsistenten Code-Intervalle wurden die konsistenten Code-

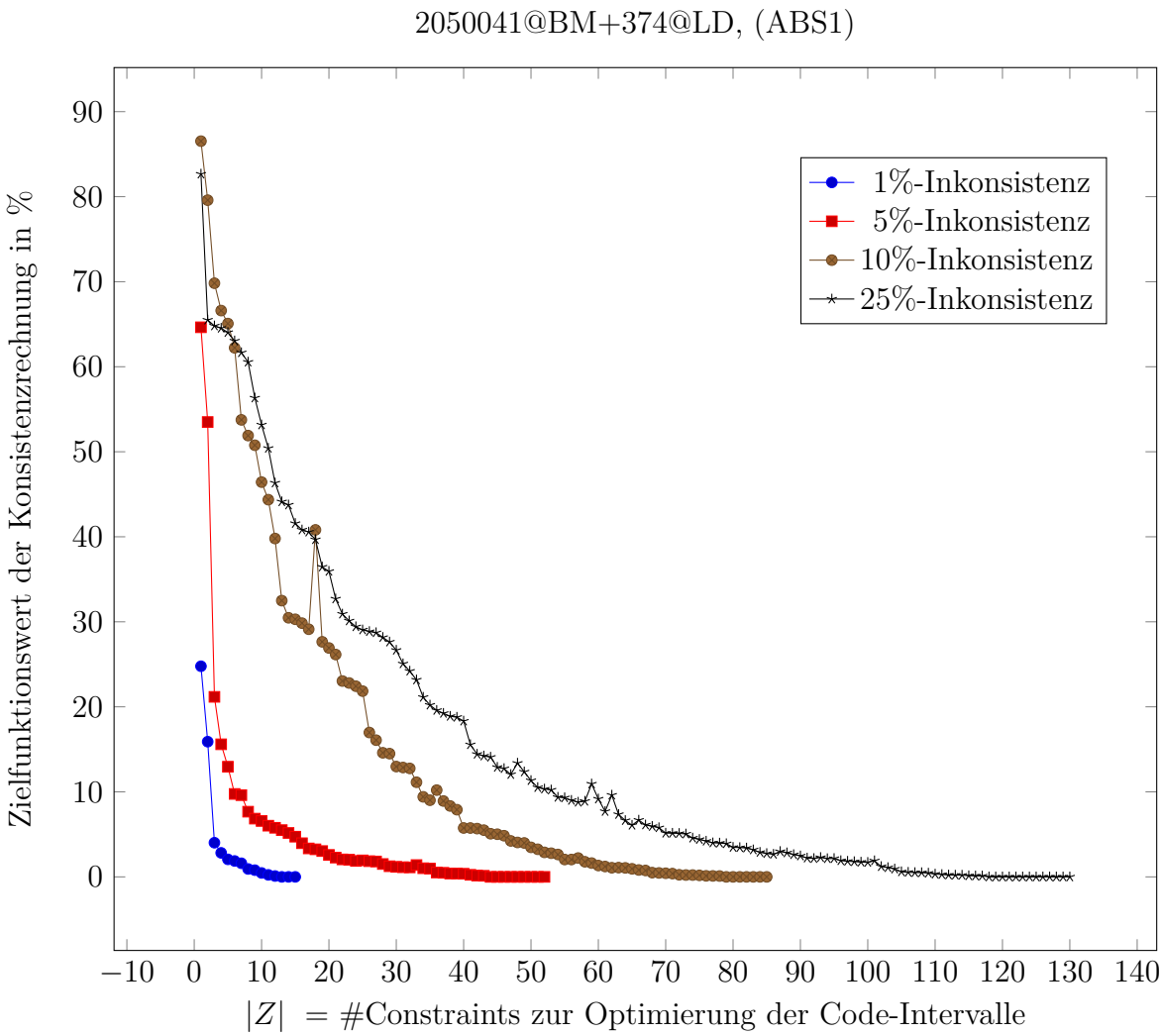

Abbildung 13: Automatische Reparatur von unterschiedlich starken Inkonsistenzen

Intervalle mit minimalem quadratischen Abstand bestimmt. Als Abbruchkriterium diente $\epsilon = 0$.

Das typische Konvergenzverhalten von Algorithmus 5 ist in Abbildung 13 für Instanzen mit Inkonsistenzen der Stärke 1%, 5%, 10% bzw. 25% zu sehen. Die x-Achse stellt die Anzahl $|Z|$ der Constraints aus der wachsenden Menge Z dar, d. h. die Anzahl an Constraints, unter denen in einer Iteration die nachfolgenden Code-Intervalle berechnet wurden. Die y-Achse repräsentiert den optimalen Zielfunktionswert in der Konsistenzrechnung, der mit den entsprechenden Code-Intervallen erreichbar ist.

Die zugehörigen Laufzeiten wurden mit 7, 61, 269 bzw. 366 Sekunden gemessen und spiegeln die Ausprägungen der Inkonsistenzen wieder. So waren für die 1%-Inkonsistenz nur knapp 20 Konsistenzrechnungen nötig, um die konsistenten Code-Intervalle mit minimalem quadratischen Abstand zu bestimmen. Im Fall der 25%-Inkonsistenz mussten hingegen circa 130 Konsistenzrechnungen durchgeführt werden.

Die Laufzeiten der automatischen Reparatur sind demnach davon abhängig, inwieweit es dem Planer gelingt, die Produktdokumentation bei der Erstellung der Plansachver-

halte zu berücksichtigen. Je kleiner die „Verspannung“ zwischen der Produktdokumentation und den statistischen Vorgaben ist, desto schneller ist auch die automatische Reparatur. Beispielsweise sind für den Fall, dass ein Planer bereits eine konsistente Prognose erstellt hat und nun die begrenzte Verfügbarkeit eines Bauteiles einzuarbeiten hat, sehr zügige Laufzeiten zu erwarten.

5 Zusammenfassung

In der automobilen Materialbedarfsplanung werden Plansachverhalte eingesetzt, um Zukunftsszenarien zu beschreiben. Plansachverhalte umfassen einerseits die Produktdokumentation und andererseits eine Menge von statistischen Vorgaben, wie sie beispielsweise durch Sonderausstattungsprognosen gegeben sind (siehe Kapitel 1).
Die Hauptmotivation für diese Arbeit bestand in der Entwicklung eines mathematisch exakten Verfahrens, das Plansachverhalte verlustfrei in Sekundärbedarfe übersetzt. Im Folgenden werden die in diesem Zusammenhang gewonnenen Erkenntnisse und entwickelten Verfahren zusammenfassend dargestellt:

- **Berechnung optimaler Variantenverteilungen.** Es wurde ein auf Column Generation basierendes Verfahren entwickelt und implementiert, das für eine beliebige lineare Zielfunktion eine optimale Verteilung auf dem Raum der möglichen Konfigurationsvarianten bestimmt.[4] Den unterschiedlichen Konfigurationsvarianten werden Prioritäten durch die Zielfunktion zugeordnet. Als Nebenbedingungen können statistische Aussagen formuliert werden, wie beispielsweise intervallbasierte Prognosen von Sonderausstattungen oder begrenzte Verfügbarkeiten von Bauteilen.

- **Beschreibung des Variantenraumes durch Aussagenlogik.**[5] Aus Sicht der Materialbedarfsplanung wird der Raum der möglichen Konfigurationsvarianten durch eine Validierung definiert, wie sie im für Daimler eingesetzten System PPM enthalten ist. Neben den technischen Restriktionen sind in PPM insbesondere auch marktspezifische Restriktionen abgelegt. Nicht baubare Codekombinationen werden durch die Validierung in einem iterativen Prozess zu baubaren Codekombinationen abgeändert.
  Mittels Aussagenlogik wurden genau die Codekombinationen beschrieben, die nicht mehr von der PPM-Validierung verändert werden. Der so formalisierte Variantenraum wurde unter anderem auf Primimplikanten untersucht. Dies motivierte die auf Primimplikanten basierende Berechnung optimaler Variantenverteilungen, in der die Freiheitsgerade der Primimplikanten ausgenutzt werden.

- **Optimale Produktkonfiguration.** Um nützliche Laufzeiten für das auf Basis von Column Generation entwickelte Verfahren zu erzielen, ist die schnelle Berechnung optimaler Konfigurationsvarianten eine wesentliche Voraussetzung.

[4] Ebenfalls auf Column Generation basierend berechnen Singh et. al optimale Multimengen von Konfigurationen [SR13].
[5] Die Idee zur Übersetzung der Produktdokumentation in eine aussagenlogische Formel stammt von Carsten Sinz [Sin97, KS00].

In [SR13] werden optimale Konfigurationsvarianten mit dem ILP-Solver CPLEX berechnet. In dieser Arbeit wurde beobachtet, dass die Spaltengenerierung mit einer CDCL-basierenden, linearen Suche unter bestimmten Bedingungen wesentlich effizienter ist. Dieses Ergebnis ist außerdem für die in [WZK13] beschriebenen Anwendungen relevant, wie z. B. für die Rekonfiguration nicht baubarer Kundenaufträge.

- **Übersetzung von Plansachverhalten in Sekundärbedarfe** – als direkte Anwendung der Optimierung von Variantenverteilungen. Dazu werden die statistischen Vorgaben der Plansachverhalte als Nebenbedingungen im Optimierungsmodell aufgefasst. Über die Zielfunktion werden die Konfigurationsvarianten so verteilt, dass die Verbauquote eines gegebenen Bauteils maximiert bzw. minimiert wird. Die auf diese Weise berechneten Minimalintervalle repräsentieren genau die Verbauquoten, die unter den vorgegebenen Plansachverhalten möglich sind. Die erzielten numerischen Ergebnisse weisen nicht nur die Realisierbarkeit der sogenannten *Intervallrechnung* nach – auf der Granularität, auf der die Sonderausstattungsprognosen formuliert werden, ist die Berechnung von Minimalintervallen nahezu in Echtzeit möglich.

- **Nachweis der Konsistenz von Plansachverhalten.** Die zugrunde liegende Problemstellung wurde als Instanz von PSAT formuliert und motivierte damit die in dieser Arbeit erfolgte Entwicklung eines auf Column Generation basierenden Verfahrens. Die Optimierung von Variantenverteilungen erlaubt es – gemäß der Phase I des Simplex-Algorithmus – statistische Vorgaben auf Konsistenz zur Produktübersicht zu überprüfen. Die sogenannte *Konsistenzrechnung* ist zugleich ein zentraler Bestandteil der Intervallrechnung, da die Konsistenz der Nebenbedingungen eine grundlegende Voraussetzung in der Intervallrechnung darstellt.

- **Automatische Reparatur von inkonsistenten Plansachverhalten.** Auf Basis des Dualitätstheorems der linearen Optimierung wurde ein Verfahren entwickelt, das das Optimierungsproblem

$$\underset{I \text{ ist konsistent zu PD}}{\arg\min} \quad Q(\tilde{I}, I)$$

löst. Dabei ist $\tilde{I}$ ein zur Produktübersicht PD inkonsistentes System von Verbauquotenintervallen. Mit der Zielfunktion Q kann der Planer priorisieren, welche Intervalle bevorzugt abgeändert werden, um ein konsistentes System von Intervallen I zu generieren. Im Sinne der Gleichbehandlung aller statistischen Vorgaben wurde die Machbarkeit anhand der quadratischen Abstandsfunktion nachgewiesen. Die sehr nützlichen Laufzeiten sind von der „Verspannung“ abhängig, die zwischen der Produktübersicht PD und den Verbauquotenintervallen $\tilde{I}$ vorliegt.

In dieser Arbeit wurden Methoden des automatischen Beweisens mit Methoden der linearen Optimierung kombiniert, um planerische Verbauraten von kundenwählbaren Optionen unter der Produktdokumentation eines Premiumautomobilherstellers zu validieren. Der entwickelte Ansatz ermöglicht die Erstellung konsistenter Plansachverhalte und erlaubt das Ableiten induzierter Verbauraten, wie zum Beispiel für eine Teilebedarfsprognose. Ein Schwerpunkt der Arbeit lag in der möglichst effizienten Berechnung der zugrundeliegenden Optimierungsprobleme. So wurde aufgezeigt, wie Freiheitsgrade der Produktdokumentation ausgenutzt werden können und wie optimale Konfigurationsvarianten mit möglichst kurzen Laufzeiten für den auf Column Generation basierenden Ansatz generierbar sind.

Literaturverzeichnis

[Aig07] Aigner, Martin: *Diskrete Mathematik:*. Vieweg+Teubner Verlag, 2007 (Aufbaukurs Mathematik)

[CK96] Clarke, Edmund M. ; Kurshan, Robert P.: Computer-aided Verification. In: *IEEE Spectr.* 33 (1996), Juni, Nr. 6, S. 61–67

[CK03] Chai, Donald ; Kuehlmann, Andreas: A Fast Pseudo-boolean Constraint Solver. In: *Proceedings of the 40th Annual Design Automation Conference*. New York, NY, USA : ACM, 2003 (DAC '03), S. 830–835

[Coo71] Cook, Stephen A.: The Complexity of Theorem-proving Procedures. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. New York, NY, USA : ACM, 1971 (STOC '71), S. 151–158

[Cor07] Cornuéjols, Gérard: Revival of the Gomory cuts in the 1990's. In: *Annals of Operations Research* 149 (2007), Nr. 1, S. 63–66

[CPL15] *IBM ILOG CPLEX Optimizer*. `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`. Version: April 2015

[Dak65] Dakin, R. J.: A tree-search algorithm for mixed integer programming problems. In: *The Computer Journal* 8 (1965), Nr. 3, S. 250–255

[Dan90] Dantzig, George B.: Origins of the Simplex Method. In: Nash, Stephen G. (Hrsg.): *A History of Scientific Computing*. ACM, 1990, S. 141–151

[DFJ54] Dantzig, G ; Fulkerson, R ; Johnson, S: Solution of a large-scale traveling-salesman problem. In: *Operations Research* 2 (1954), S. 393–410

[DFLBM13] Deharbe, David ; Fontaine, Pascal ; Le Berre, Daniel ; Mazure, Bertrand: Computing prime implicants. In: *Formal Methods in Computer-Aided Design (FMCAD), 2013* IEEE, 2013, S. 46–52

[DG84] Dowling, William F. ; Gallier, Jean H.: Linear-time algorithms for testing the satisfiability of propositional Horn formulae. In: *The Journal of Logic Programming* 1 (1984), Nr. 3, S. 267 – 284

[DL05] Desrosiers, Jacques ; Lubbecke, Marco E.: A Primer in Column Generation. In: Desaulniers, Guy (Hrsg.) ; Desrosiers, Jacques (Hrsg.) ; Solomon, MariusM. (Hrsg.): *Column Generation*. Springer US, 2005, S. 1–32

[DLL62] Davis, Martin ; Logemann, George ; Loveland, Donald: A Machine Program for Theorem-proving. In: *Commun. ACM* 5 (1962), Juli, Nr. 7, S. 394–397

[DP60] Davis, Martin ; Putnam, Hilary: A Computing Procedure for Quantification Theory. In: *J. ACM* 7 (1960), Juli, Nr. 3, S. 201–215

[FH94] Frisch, Alan ; Haddawy, Peter: Anytime Deduction for Probabilistic Logic. In: *Artif. Intell* 69 (1994), S. 93–122

[GKP88] Georgakopoulos, George ; Kavvadias, Dimitris ; Papadimitriou, Christos H.: Probabilistic satisfiability. In: *Journal of Complexity* 4 (1988), Nr. 1, S. 1 – 11

[Gom58] Gomory, Ralph E.: Outline of an algorithm for integer solutions to linear programs. In: *Bulletin of the American Mathematical Society* 64 (1958), 09, Nr. 5, S. 275–278

[GSK98] Gomes, Carla P. ; Selman, Bart ; Kautz, Henry: Boosting Combinatorial Search Through Randomization. In: *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*. Menlo Park, CA, USA : American Association for Artificial Intelligence, 1998 (AAAI '98/IAAI '98), S. 431–437

[Hak85] Haken, Armin: The intractability of resolution. In: *Theoretical Computer Science* 39 (1985), Nr. 0, S. 297 – 308. – Third Conference on Foundations of Software Technology and Theoretical Computer Science

[Han97] Hansen, Pierre: *Correctness of Anytime Deduction for Probabilistic Logic*. GERAD, École des hautes études commerciales, 1997 (Cahiers du GÉRAD)

[HJA+00] Hansen, Pierre ; Jaumard, Brigitte ; Aragão, Marcus P. ; Chauny, Fabien ; Perron, Sylvain: Probabilistic satisfiability with imprecise probabilities. In: *International Journal of Approximate Reasoning* 24 (2000), Nr. 2–3, S. 171 – 189

[Hol00] Holtze, Peter: *Data mining in der Bedarfsprognose von Komponenten und Teilen: neue Ansätze zur Planung einer variantenreichen Serienfertigung*. Shaker, 2000

[Hoo03] Hooker, J. N.: *Optimization Methods in Logic*. 2003

[JHA91] Jaumard, Brigitte ; Hansen, Pierre ; Aragao, Marcus P.: Column Generation Methods for Probabilistic Logic. In: *ORSA Journal on Computing* 3 (1991), Nr. 2, S. 135–148

[JP09] Jaumard, B. ; Parreira, A.D.: An anytime deduction algorithm for the probabilistic logic and entailment problems. In: *International Journal of Approximate Reasoning* 50 (2009), Nr. 1, S. 92 – 103. – Special section on recent advances in soft computing in image processing and Special Section on Selected papers from {NAFIPS} 2006

[KB11] Kappler, Jochen ; Bracht, Uwe: *Robuste intervallbasierte Primär- und Sekundärbedarfsplanung in automobilen Neuproduktprojekten*. Shaker, 2011

[KCY03] Kroening, Daniel ; Clarke, Edmund ; Yorav, Karen: Behavioral Consistency of C and Verilog Programs Using Bounded Model Checking. In: *Proceedings of DAC 2003*, ACM Press, 2003, S. 368–371

[KK07] Klieber, Will ; Kwon, Gihwon: Efficient CNF encoding for selecting 1 from N objects. In: *International Workshop on Constraints in Formal Verification* (2007)

[KP90] Kavvadias, Dimitris ; Papadimitriou, Christos H.: A linear programming approach to reasoning about probabilities. In: *Annals of Mathematics and Artificial Intelligence* 1 (1990), Nr. 1-4, S. 189–205

[KS00] Küchlin, Wolfgang ; Sinz, Carsten: Proving Consistency Assertions for Automotive Product Data Management. In: *J. Automated Reasoning* 24 (2000), Februar, Nr. 1–2, S. 145–163

[LBP10] Le Berre, Daniel ; Parrain, Anne: The Sat4j library, release 2.2. In: *Journal on Satisfiability, Boolean Modeling and Computation* 7 (2010), S. 59–64

[LD60] Land, A. H. ; Doig, A. G.: An Automatic Method of Solving Discrete Programming Problems. In: *Econometrica* 28 (1960), Nr. 3, S. 497–520

[MMZ+01] Moskewicz, Matthew W. ; Madigan, Conor F. ; Zhao, Ying ; Zhang, Lintao ; Malik, Sharad: Chaff: Engineering an Efficient SAT Solver. In: *Proceedings of the 38th Annual Design Automation Conference*. New York, NY, USA : ACM, 2001 (DAC '01), S. 530–535

[MVDH99] Merle, Olivier du ; Villeneuve, Daniel ; Desrosiers, Jacques ; Hansen, Pierre: Stabilized column generation. In: *Discrete Mathematics* 194 (1999), Nr. 1–3, S. 229 – 237

[Nil86] Nilsson, Nils J.: Probabilistic logic. In: *Artificial Intelligence* 28 (1986), Nr. 1, S. 71 – 87

[Ohl00] Ohl, Stefan: *Prognose und Planung variantenreicher Produkte am Beispiel der Automobilindustrie*. VDI-Verlag, 2000

[Pad95] Padberg, Manfred W.: *Linear optimization and extensions*. Berlin, New York : Springer, 1995 (Algorithms and combinatorics)

[PG86] Plaisted, David A. ; Greenbaum, Steven: A Structure-preserving Clause Form Translation. In: *J. Symb. Comput.* 2 (1986), September, Nr. 3, S. 293–304

[RM09] Roussel, Olivier ; Manquinho, Vasco M.: Pseudo-Boolean and Cardinality Constraints. In: Biere, Armin (Hrsg.) ; Heule, Marijn (Hrsg.) ; Maaren, Hans van (Hrsg.) ; Walsh, Toby (Hrsg.): *Handbook of Satisfiability* Bd. 185. IOS Press, 2009, S. 695–733

[SC15] Sat-Competition: *The international SAT Competitions web page*. 2015. – `www.satcompetition.org`

[Sch86] Schrijver, Alexander: *Theory of Linear and Integer Programming*. New York, NY, USA : John Wiley & Sons, Inc., 1986

[SE03] Sörensson, Niklas ; Een, Niklas: An Extensible SAT-solver. In: *Lecture Notes in Computer Science. 6th International Conference on Theory and Applications of Satisfiability Testing, Santa Margherita Ligure, Italy, 5-8 May 2003* Bd. 2919, 2003, S. 502–518

[Sin97] Sinz, Carsten: *Baubarkeitsprüfung von Kraftfahrzeugen durch automatisches Beweisen*, Universität Tübingen, Diplomarbeit, Dezember 1997

[Sin03] Sinz, Carsten: *Verifikation regelbasierter Konfigurationssysteme*. Tübingen, Germany, Fakultät für Informations- und Kognitionswissenschaften, Universität Tübingen, Dissertation, Dezember 2003

[Sin05] Sinz, Carsten: Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In: Beek, Peter van (Hrsg.): *Principles and Practice of Constraint Programming - CP 2005* Bd. 3709. Springer Berlin Heidelberg, 2005, S. 827–831

[SKK03] Sinz, Carsten ; Kaiser, Andreas ; Küchlin, Wolfgang: Formal Methods for the Validation of Automotive Product Configuration Data. In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 17 (2003), Januar, Nr. 1, S. 75–97. – Special issue on configuration.

[SR13] Singh, Tilak R. ; Rangaraj, Narayan: Generation of predictive configurations for production planning. In: *15 th International Configuration Workshop*, 2013, S. 79

[SR14] Singh, Tilak R. ; Rangaraj, Narayan: Optimization based framework for transforming automotive configurations for production planning. In: *16th International Configuration Workshop*, 2014, S. 31

[SS96] Silva, Joo P. M. ; Sakallah, Karem A.: GRASP – A New Search Algorithm for Satisfiability. In: *Proceedings of the 1996 IEEE/ACM International Conference on Computer-aided Design*. Washington, DC, USA : IEEE Computer Society, 1996 (ICCAD '96), S. 220–227

[SS05] Sheini, Hossein M. ; Sakallah, Karem A.: Pueblo: A Modern Pseudo-Boolean SAT Solver. In: *Design, Automation; Test in Europe Conference; Exhibition* 2 (2005), S. 684–685

[Stä07] Stäblein, Thomas: *Integrierte Planung des Materialbedarfs bei kundenauftragsorientierter Fertigung von komplexen und variantenreichen Serienprodukten*. Shaker, 2007

[Tse83] Tseitin, G. S.: On the Complexity of Derivation in Propositional Calculus. In: Siekmann, J. (Hrsg.) ; Wrightson, G. (Hrsg.): *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*. Berlin, Heidelberg : Springer, 1983, S. 466–483

[War98] Warners, Joost P.: A linear-time transformation of linear inequalities into conjunctive normal form. In: *Information Processing Letters* 68 (1998), Nr. 2, S. 63–69

[WZK13] Walter, Rouven ; Zengler, Christoph ; Küchlin, Wolfgang: Applications of MaxSAT in automotive configuration. In: *15 th International Configuration Workshop* Bd. 1, 2013, S. 21

Tabellenverzeichnis

Abbildungsverzeichnis

Liste der Algorithmen

Anhang

Grundlagen der linearen Algebra

Matrizen $A \in \mathbb{R}^{m\times n}$ besitzen m Zeilen und n Spalten. Sie werden in der Regel mit Großbuchstaben bezeichnet. Der Eintrag der i-ten Zeile und j-ten Spalte einer Matrix A wird mit a_{ij} angegeben. Das Produkt einer *Matrixmultiplikation* $AB = C \in \mathbb{R}^{m\times n}$ ist für $A \in \mathbb{R}^{m\times l}$ und $B \in \mathbb{R}^{l\times n}$ durch $c_{ij} := \sum_k a_{ik}b_{kj}$ definiert. Für ein Skalar $\lambda \in \mathbb{R}$ ist die *Skalarmultiplikation* $C = \lambda A$ durch $c_{ij} := \lambda a_{ij}$ definiert. Die *Summe* $C = A + B$ der Matrizen $A,\ B \in \mathbb{R}^{m\times n}$ ist durch paarweise Addition der Einträge $c_{ij} := a_{ij} + b_{ij}$ gegeben.

Vektoren $x \in \mathbb{R}^n = \mathbb{R}^{n\times 1}$ sind stets als Spaltenvektoren aufzufassen. Sie sind eine Matrix mit einer Spalte und werden durch einen Kleinbuchstaben dargestellt. Der *Nullvektor* bzw. die *Nullmatrix* besteht ausschließlich aus Nullen und wird mit $\mathbf{0}$ bezeichnet. Der *Einsvektor* $\mathbf{1}$ enthält analog nur Einsen. Ein Einheitsvektor $\mathbf{e}_i$ besitzt bis auf eine Eins in der i-ten Komponente nur Nulleinträge. Einträge a_i können je nach Kontext den i-ten Eintrag eines Vektors a oder die i-te Spalte einer Matrix A repräsentieren.

Die *Transposition* wird durch den t-Operator dargestellt und erwirkt das Vertauschen von Zeilen und Spalten, d. h. $B = A^t :\Leftrightarrow \forall i, j :\ b_{ij} = a_{ji}$. Für eine Matrix $A \in \mathbb{R}^{m\times n}$ ist A^t eine Matrix mit n Zeilen und m Spalten. *Zeilenvektoren* x^t entsprechen demnach transponierten Spaltenvektoren - also Matrizen mit genau einer Zeile. Kontextabhängig werden die Zeilen einer Matrix A auch mit a_i^t bezeichnet. Des Weiteren gilt $(A^t)^t = A$ und $(AB)^t = B^tA^t$, woraus sich zum Beispiel die Rechenregel $(Ax)^t = x^tA^t$ ableitet.

Matrizen können durch *Untermatrizen* dargestellt werden. Beispielsweise ist für $A \in \mathbb{R}^{m\times n},\ B \in \mathbb{R}^{m\times l}, A' \in \mathbb{R}^{k\times n},\ B' \in \mathbb{R}^{k\times l}$ die Matrix

$$C := \begin{pmatrix} A & B \\ A' & B' \end{pmatrix} \in \mathbb{R}^{(m+k)\times(n+l)}$$

definiert, wobei die Zeilen und die Spalten auf natürliche Art und Weise paarweise miteinander konkateniert werden.
Für eine Matrix $A \in \mathbb{R}^{m\times n}$ und eine Indexteilmenge $I := \{i_1, \ldots, i_k\} \subseteq \{1, \ldots n\}$ wird die Untermatrix

$$A_I := \begin{pmatrix} | & & | \\ a_{i_1} & \cdots & a_{i_k} \\ | & & | \end{pmatrix} \in A \in \mathbb{R}^{m\times k}$$

definiert, die sich aus den Spalten $a_{i_1}, \ldots, a_{i_k}$ von A zusammensetzt. Für einen Spaltenvektor b und eine Indexteilmenge I wird $b_I := (b_I^t)^t$ definiert.

Eine *Linearkombination* von Vektoren $a_1, \ldots, a_n \in \mathbb{R}^m$ ist eine Summe der Form $\sum_i \lambda_i a_i$. Dabei stellen die λ_i reelle Koeffizienten dar, mit denen die Vektoren a_i skalar multipliziert werden. Gilt zusätzlich $\lambda_i \geq 0$, so liegt eine *konische Kombination* der Vektoren a_i vor.

Die Vektoren, die als Linearkombination bzw. konische Kombination von $a_1, \ldots, a_n$ dargestellt werden können, werden mit $\text{lin}\{a_1, \ldots, a_n\}$ bzw. $\text{cone}\{a_1, \ldots, a_n\}$ zu einer Menge zusammen gefasst. Die Menge $\text{lin}\{a_1, \ldots, a_n\}$ bzw. $\text{cone}\{a_1, \ldots, a_n\}$ wird auch als der durch $a_1, \ldots, a_n$ *aufgespannte Unterraum* bzw. *aufgespannte Kegel* bezeichnet.

Eine *Matrix-Vektor-Multiplikation* Ax entspricht einer Linearkombination der Spalten von A, wobei die Koeffizienten in x stehen. Andersherum entspricht $x^t A$ einer Linearkombination der Zeilen von A.

Bei fester Matrix $A \in \mathbb{R}^{m \times n}$ kann die Matrix-Vektor-Multiplikation auch als *lineare Abbildung* $f_A : \mathbb{R}^n \longrightarrow \mathbb{R}^m,\ f_A(x) := Ax$ verstanden werden – es gilt $f_A(\lambda_1 x_1 + \lambda_2 x_2) = \lambda_1 f_A(x_1) + \lambda_2 f_A(x_2)$. Insbesondere stellt ein Vektor $c^t,\ c \in \mathbb{R}^n$ eine lineare Funktion $\mathbb{R}^n \longrightarrow \mathbb{R}$ dar.

Vektoren $a_1, \ldots, a_k$ heißen *linear unabhängig*, falls der Nullvektor aus $a_1, \ldots, a_k$ nur über die triviale Linearkombination (d. h. $\lambda_i = 0\ \forall i$) dargestellt werden kann. Anderenfalls heißen die Vektoren *linear abhängig*. Eine Menge von Vektoren ist somit genau dann linear abhängig, wenn ein Vektor existiert, der als Linearkombination der restlichen Vektoren dargestellt werden kann. Ist $a \in \text{lin}\{a_1, \ldots, a_n\}$, so ist a von $a_1, \ldots, a_n$ linear abhängig und es gilt $\text{lin}\{a_1, \ldots, a_n\} = \text{lin}\{a, a_1, \ldots, a_n\}$.

Im $\mathbb{R}^m$ können höchstens m Vektoren zueinander linear unabhängig sein. Ist dies der Fall, so bilden sie eine *Basis* des $\mathbb{R}^m$, d. h. jeder Vektor des $\mathbb{R}^m$ kann über eine eindeutige Linearkombination aus den m Basisvektoren dargestellt werden. Allgemein bilden $k \leq m$ linear unabhängige $a_1, \ldots, a_k$ Vektoren des $\mathbb{R}^m$ eine Basis für den k-dimensionalen Unterraum $\text{lin}\{a_1, \ldots, a_k\}$. Andererseits besitzt jeder Unterraum U eine Basis B und die *Dimension* von U ist durch $dim(U) := |B|$ wohldefiniert.

Unterräume der Dimension $m-1$ heißen *Hyperebenen* des $\mathbb{R}^m$. Eine Hyperebene H kann durch $m-1$ linear unabhängige Vektoren $a_1, \ldots, a_{m-1} \in \mathbb{R}^m$ oder alternativ durch einen *Normalenvektor* $c \in \mathbb{R}^m$ dargestellt werden, d. h. $H = \text{lin}\{a_1, \ldots, a_{m-1}\} = \{x | c^t x = 0\}$. Der Unterraum, der durch die Zeilen (bzw. Spalten) einer Matrix A aufgespannt wird,

ist als Zeilenraum (bzw. Spaltenraum) definiert. Die Dimension des Zeilenraumes ist identisch mit der Dimension des Spaltenraumes und wird mit rang(A) bezeichnet. Die Zeilenmenge bzw. Spaltenmenge einer Matrix A wird mit $Z(A)$ bzw. $Sp(A)$ abgekürzt.

Eine lineare Abbildung einer quadratischen Matrix $A \in \mathbb{R}^{m\times m}$ ist genau dann bijektiv, falls rang$(A) = m$ gilt. In diesem Fall existiert eine *Inverse* $A^{-1} \in \mathbb{R}^{m\times m}$, die die Umkehrabbildung darstellt. Es gilt $AA^{-1} = I_m$, wobei I_m die *Identitätsmatrix* bezeichnet, welche auf der Diagonalen Einsen und sonst Nullen enthält.

Für eine Matrix $A \in \mathbb{R}^{m\times n}$ mit Zeilen a_i^t und für einen Vektor $b \in \mathbb{R}^m$ ist durch $Ax = b$ ein lineares Gleichungssystem (LGS) gegeben, bestehend aus m Gleichungen $a_i^t x = b_i$ und n Unbekannten x_i.
Abschließend wird ein Lemma zur Lösungsmenge linearer Gleichungssysteme bereitgestellt.

Lemma 5
Es seien zwei Gleichungssysteme $Ax = b$ und $\tilde{A}x = \tilde{b}$ gegeben. Sind die Zeilenräume der Matrizen

$$\begin{pmatrix} A & b \end{pmatrix} \text{ und } \begin{pmatrix} \tilde{A} & \tilde{b} \end{pmatrix}$$

identisch, so sind die Lösungsmengen der beiden Gleichungssysteme identisch.

Beweis. Es sei x' eine Lösung von $Ax = b$. Mit $\tilde{a}_i^t x = \tilde{b}_i$ sei eine beliebige Gleichung von $\tilde{A}x = \tilde{b}$ gegeben. Es ist $\tilde{a}_i^t x' = \tilde{b}_i$ zu zeigen. Nach Voraussetzung existiert eine Linearkombination

$$\begin{pmatrix} \tilde{a}_i^t & \tilde{b}_i \end{pmatrix} = \sum_j \lambda_j \begin{pmatrix} a_j^t & b_j \end{pmatrix}.$$

Es gilt

$$\begin{aligned} Ax' = b \;&\Rightarrow\; \forall j : \begin{pmatrix} a_j^t & b_j \end{pmatrix} \begin{pmatrix} x' \\ -1 \end{pmatrix} = 0 \\ &\Rightarrow\; 0 = \sum_j \lambda_j \begin{pmatrix} a_j^t & b_j \end{pmatrix} \begin{pmatrix} x' \\ -1 \end{pmatrix} = \begin{pmatrix} \tilde{a}_i^t & \tilde{b}_i \end{pmatrix} \begin{pmatrix} x' \\ -1 \end{pmatrix} \\ &\Rightarrow\; \tilde{a}_i^t x' = \tilde{b}_i \end{aligned}$$

□

Als direkte Konsequenz ergibt sich, dass Gleichungen $a_i^t x = b_i$ in einem Gleichungssystem $Ax = b$ *redundant* sind, falls sie sich als Linearkombination anderer Gleichungen darstellen lassen. Das Entfernen entsprechender Zeilen ändert nicht den Zeilenraum.

Beweise zu Sätzen der linearen Optimierung

Satz 6

Es seien Vektoren $a_1, \ldots, a_n \in \mathbb{R}^m$ mit $\text{lin}\{a_1, \ldots, a_n\} = \mathbb{R}^m$ gegeben. Für beliebige $b \in \mathbb{R}^m$ gilt genau eine der beiden folgenden Aussagen:

1. Es existieren m linear unabhängige Vektoren $a_{i_1}, \ldots, a_{i_m}$ aus $\{a_1, \ldots, a_n\}$ mit denen b konisch darstellbar ist, d. h. $b \in \text{cone}\{a_{i_1}, \ldots, a_{i_m}\}$.

2. Es existieren $m-1$ Vektoren $a_{i_1}, \ldots, a_{i_{m-1}}$ aus $\{a_1, \ldots, a_n\}$, die eine Hyperebene $\{x | c^t x = 0\}$ aufspannen, so dass $c^t b < 0$ und $c^t a_1, \ldots, c^t a_n \geq 0$ gilt.

Beweis nach [Sch86]. Angenommen beide Aussagen treffen zu. Dann existieren einerseits $\lambda_i \geq 0$ mit $b = \sum_{i=1}^{n} \lambda_i a_i$, wobei höchstens m der λ_i echt positiv sind. Andererseits existiert ein c mit $c^t b < 0$ und $c^t a_1, \ldots, c^t a_n \geq 0$. Somit folgt der Widerspruch

$$0 > c^t b = \sum_{i=1}^{n} \lambda_i c^t a_i \geq 0.$$

Es bleibt zu zeigen, dass eine der beiden Aussagen wahr ist. Sei dazu mit $D = \{a_{i_1}, \ldots, a_{i_m}\}$ eine Auswahl von m linear unabhängigen Vektoren aus $\{a_1, \ldots a_n\}$ gegeben. Die Menge D wird nun iterativ solange abgeändert, bis sie Vektoren enthält, die es erlauben, eine der beiden Aussagen nachzuweisen. Eine Iteration umfasst dabei drei Schritte:

1. Betrachte die Koeffizienten λ_{i_j} der eindeutigen Linearkombination $b = \sum_{j=1}^{m} \lambda_{i_j} a_{i_j}$. Gilt $\lambda_{i_j} \geq 0$ für alle j, so stoppe, da die erste Aussage wahr ist.

2. Wähle den kleinsten Index $h \in \{i_1, \ldots i_m\}$ mit $\lambda_h < 0$ und betrachte die Hyperebene $\{x | c^t x = 0\} = \text{lin}(D \setminus \{a_h\})$, wobei c so normalisiert ist, dass $c^t a_h = 1$ gilt. Mit $c^t a = 0$ für $a \in D \setminus \{a_h\}$ folgt die Ungleichung $c^t b = \lambda_h < 0$. Stoppe nun, falls zusätzlich $c^t a_1, \ldots, c^t a_n \geq 0$ und somit die zweite Aussage gilt.

3. Wähle den kleinsten Index $s \in \{1, \ldots n\}$ mit $c^t a_s < 0$ ($a_s \overset{!}{\notin} D$) und definiere $D := (D \setminus \{a_h\}) \cup \{a_s\}$.

Es muss noch gezeigt werden, dass die beschriebene Iteration nach wiederholter Anwendung irgendwann terminiert. Mit D_k sei die Menge D vor der k-ten Iteration beschrieben. Die Menge D tritt nur in endlich vielen verschiedenen Varianten auf, da nur endlich viele m-elementige Teilmengen einer n-elementigen Menge existieren. Es ist also ausreichend, die Annahme der erneuten Generierung eines zuvor schon generierten D's zum Widerspruch zu führen. Dafür seien Iterationsschritte $k < l$ mit $D_k = D_l$ gegeben. Der Index $r \in \{1, \ldots, n\}$ sei der größte, für den ein a_r in einer Iteration $p \in \{k, \ldots, l-1\}$ aus D entfernt wurde. Wegen $D_k = D_l$ muss es auch eine Iteration

$q \in \{k, \ldots, l-1\}$ geben, in der a_r zu D hinzugefügt wurde. Die Wahl von r impliziert insbesondere

$$D_p \cap \{a_{r+1}, \ldots, a_n\} = D_q \cap \{a_{r+1}, \ldots, a_n\}. \qquad (*)$$

Mit $D_p = \{a_{i_1}, \ldots, a_{i_m}\}$ sei $b = \sum_{i=1}^{m} \lambda_{i_j} a_{i_j}$ die eindeutige Linearkombination von b. Des Weiteren sei c der normalisierte Normalenvektor der Hyperebene aus Iteration q. Es werden die folgenden Beobachtungen bzgl. der Indexe $i_1, \ldots, i_m$ benötigt. Für $i_j < r$ gilt $\lambda_{i_j} \geq 0$ und $c^t a_{i_j} \geq 0$, da in der p-ten Iteration r der kleinste Index mit $\lambda_r < 0$ ist bzw. r der kleinste Index der q-ten Iteration ist, für den $c^t a_r < 0$ gilt. Für $i_j = r$ gilt also $\lambda_{i_j} < 0$ und $c^t a_{i_j} < 0$. Für $i_j > r$ folgt $c^t a_{i_j} = 0$ aus $(*)$ und dem zweiten Schritt der q-ten Iteration. Es ergibt sich nun der Widerspruch

$$0 > c^t b = c^t (\sum_{i=1}^{n} \lambda_{i_j} a_{i_j}) = \lambda_{i_j} c^t a_{i_j} + \cdots + \lambda_{i_j} c^t a_{i_j} > 0,$$

wobei die erste Ungleichung eine Beobachtung zum zweiten Iterationsschritt war. □

Lemma 2 (Farkas I)
Sei $A \in \mathbb{R}^{m \times n}$ eine Matrix mit $\operatorname{rang}(A) = m$ und $b \in \mathbb{R}^m$ ein Vektor. Dann ist genau eines der beiden Systeme lösbar:

1. $Ax = b,\ x \geq \mathbf{0}$,

2. $y^t A \geq \mathbf{0},\ b^t y < 0$.

Beweis nach [Sch86]. Für ein beliebiges $x \geq \mathbf{0}$ mit $Ax = b$ und ein beliebiges y mit $y^t A \geq \mathbf{0}$ gilt offensichtlich $y^t b = y^t A x \geq 0$. Andersherum sei $a_1, \ldots, a_n$ die Auflistung der Spalten von A. Existiert kein $x \geq \mathbf{0}$ mit $Ax = b$, d. h. $b \notin \operatorname{cone}(a_1, \ldots, a_n)$, so existiert nach Satz 6 ein y mit $y^t b < 0$ und $y^t a_1, \ldots, y^t a_n \geq 0$, d. h. $y^t A \geq \mathbf{0}$. □

Lemma 3 (Farkas II)
Sei $A \in \mathbb{R}^{m \times n}$ eine Matrix und $b \in \mathbb{R}^m$ ein Vektor. Dann ist genau eines der beiden Systeme lösbar:

1. $Ax \leq b$,

2. $y^t A = \mathbf{0},\ y \geq \mathbf{0},\ y^t b < 0$.

Beweis nach [Sch86]. Sei

$$A' := \begin{pmatrix} I_m & A & -A \end{pmatrix} \in \mathbb{R}^{m \times (m+2n)}.$$

Zunächst wird gezeigt, dass die Systeme

$$Ax \leq b \text{ (I) und } A'x' = b,\ x' \geq \mathbf{0} \text{ (II)}$$

in x und x' erfüllbarkeitsäquivalent sind. Ist ein x mit $Ax \leq b$ gegeben, so genügt

$$x' := \begin{pmatrix} b - Ax \\ x^+ \\ x^- \end{pmatrix},\ x_i^+ := \max\{0, x_i\},\ x_i^- := \max\{0, -x_i\},\ i \in \{1, \ldots, n\}$$

dem System (II):

$$\begin{aligned} A'x' &= \begin{pmatrix} I_m & A & -A \end{pmatrix} \begin{pmatrix} b - Ax \\ x^+ \\ x^- \end{pmatrix} \\ &= b - Ax + Ax^+ - Ax^- \\ &= b - Ax + Ax \\ &= b. \end{aligned}$$

Ist andererseits ein $x' \geq \mathbf{0}$ mit $A'x' = b$ gegeben, so efüllt

$$x := x'_{I^+} - x'_{I^-},\ I^+ := \{m+1, \ldots, m+n\},\ I^- := \{m+n+1, \ldots, m+2n\}$$

das System (I):

$$\begin{aligned} Ax &= (Ax'_{I^+} - Ax'_{I^-}) + (x'_{\{1,\ldots,m\}} - x'_{\{1,\ldots,m\}}) \\ &= A'x' - x'_{\{1,\ldots,m\}} \\ &= b - x'_{\{1,\ldots,m\}} \\ &\leq b. \end{aligned}$$

Wegen $\operatorname{rang}(A') = m$ kann Lemma 2 auf das System (II) angewandt werden. Daher hat $Ax \leq b$ genau dann eine Lösung, falls für alle $y \in \mathbb{R}^m$ mit $y^t A' \geq \mathbf{0}$ die Ungleichung $y^t b \geq 0$ gilt. Abschließend reicht es zu beobachten, dass die beiden Systeme $y^t A' \geq \mathbf{0}$ und $y \geq \mathbf{0},\ y^t A = \mathbf{0}$ die selbe Lösungsmenge besitzen. □

Satz 8
Es sei M die Menge der zulässigen Lösungen eines kanonischen Programmes.

1. Ein Punkt $x \in M$ ist genau dann eine Ecke, falls die Spaltenmenge $\{a_i \mid x_i > 0\}$ linear unabhängig ist.

2. Ist $M \neq \emptyset$, so besitzt M eine Ecke.

3. Existieren in M optimale Lösungen, so existiert unter den optimalen Lösungen eine Ecke.

Beweis nach [Aig07], S.308-310. Es wird $M \neq \emptyset$ angenommen. Des Weiteren kann vorausgesetzt werden, dass $M = \{x \mid Ax = b,\ x \geq \mathbf{0}\}$ für eine Matrix $A \in \mathbb{R}^{m \times n}$ mit $\text{rang}(A) = m$ definiert ist und damit das System $Ax = b$ keine redundanten Gleichungen enthält.

1. Sei $x \in M$. Es wird die Indexmenge $Z := \{i \mid x_i > 0\}$ definiert.
   Ist x keine Ecke, dann existieren $x' \neq x'' \in M,\ \lambda \in [0,1]$ mit $x = \lambda x' + (1-\lambda)x''$.
   Für $j \neq Z$ ist $x'_j = x''_j = 0$ wegen $x_j = 0$ und $x',\ x'' \geq \mathbf{0}$ zwingend. Es gilt somit

   $$\sum_{i \in Z} x'_i a_i = b = \sum_{i \in Z} x''_i a_i.$$

   Daraus folgt $\sum_{i \in Z}(x'_i - x''_i)a_i = \mathbf{0}$, womit aufgrund von $x' \neq x''$ eine nichttriviale Linearkombination der $\mathbf{0}$ gegeben ist, d. h. $\{a_i \mid i \in Z\} = \{a_i \mid x_i > 0\}$ ist linear abhängig.
   Andererseits sei $\{a_i \mid i \in Z\}$ linear abhängig. Es existiert eine nichttriviale Linearkombination $\sum_{i \in Z} v_i a_i = \mathbf{0}$ mit $v_{i^*} \neq 0$ für mindestens ein $i^* \in Z$. Wegen $x_i > 0$ für alle $i \in Z$, existiert ein $\delta > 0$, sodass $x_i - |\delta v_i| > 0$ für alle $i \in Z$. Es werden $x', x'' \in M$ durch $x'_i := x_i + \delta v_i,\ x''_i := x_i + \delta v_i$ für $i \in Z$ und $x'_j := x''_j := 0$ für $j \in Z$ definiert:

   $$Ax' = Ax + \delta \sum_{i \in Z} v_i a_i = Ax = b = Ax - \delta \sum_{i \in Z} v_i a_i = Ax''.$$

   Wegen $x = \frac{1}{2}x' + \frac{1}{2}x''$ und $x \neq x',\ x \neq x''$ $(v_{i^*} \neq 0)$ ist x keine Ecke.

2. Da M nicht leer ist, existiert nach Satz 6 eine Teilmenge Z der Spaltenindexe von A, sodass $\{a_i \mid i \in Z\}$ linear unabhängig ist und b als konische Kombination $b = \sum_{i \in Z} \lambda_i a_i,\ \lambda_i \geq 0$ dargestellt werden kann. Somit ist x mit $x_i := \lambda_i,\ i \in Z$ und $x_j := 0,\ j \notin Z$ eine Ecke von M.

3. Es sei $c^t \in \mathbb{R}^n$ eine zu minimierende lineare Zielfunktion und $\gamma \in \mathbb{R}$ der optimale Wert des kanonisches Programmes. Die 2. Aussage des Satzes auf $M' = M \cap \{x \mid c^t x = \gamma\}$ angewandt liefert eine Ecke x^* von M', die zugleich eine optimale Lösung des kanonisches Programmes darstellt. Es muss gezeigt werden, dass x^* auch eine Ecke von M ist. Angenommen x^* ist keine Ecke von M. Dann existieren $x', x'' \in M, \lambda \in]0,1[$ mit $x^* = \lambda x' + (1-\lambda)x''$. Es folgt $\gamma = \lambda c^t x' + (1-\lambda)c^t x''$ und wegen $c^t x',\ c^t x'' \geq \gamma$ gilt $\gamma = c^t x' = c^t x''$, d. h. $x', x'' \in M'$. Dies ist ein Widerspruch dazu, dass x^* eine Ecke von M' ist.

□

Danksagung

Ich möchte mich bei all jenen bedanken, die mich beim Anfertigen dieser Dissertation begleitet und unterstützt haben.

Meinem Erstgutachter Herrn Prof. Dr. Wolfgang Küchlin danke ich für die wissenschaftliche Betreuung und für die zahlreichen, konstruktiven und lehrreichen Einzelgespräche. Herrn Prof. Dr. Peter Hauck danke ich für die Erstellung des Zweitgutachtens.

Der Daimler AG verdanke ich die Finanzierung und die fachliche Betreuung dieser Arbeit. Mein Dank gilt insbesondere meinem ehemaligen Teamchef Herrn Dr. Andreas Schütte, der mir dieses interessante und komplexe Thema anvertraute, mir in vielen Diskussionen die praxisrelevanten Aspekte verdeutlichte und zum schnellen Gelingen dieser Arbeit beigetragen hat.
Für die zusätzliche Unterstützung und den fachlichen Austausch danke ich weiter meinen ehemaligen Teamkollegen Herrn Dr. Christian Schlange, Frau Birgit Burmeister, Herrn Mathias Porten, Herrn Dr. Marcus Ziegler, Frau Dr. Lena Michailidis, Herrn Dr. Christian Franz, Herrn Merten Frenzel und Herrn Maximilian Hess, sowie Herrn Tilak Singh von Mercedes-Benz Research and Development India und Herrn Dr. Jens Averdunk von der eXXcellent solutions GmbH.

Nicht zuletzt wäre diese Arbeit nicht ohne meine große Familie möglich gewesen, die mich mit viel liebevoller Geduld und mit großem Verständnis begleitet hat.

Tübingen, Dezember 2015 Thore Kübart