Fakultät für Wirtschaftswissenschaften
Wismar Business School

Harald Mumm

Didaktischer Zugang zur Theorie und Praxis moderner Softwarebibliotheken (Frameworks) für die Unternehmensforschung (OR)

Heft 07/2018

W D P

Wismarer Diskussionspapiere / Wismar Discussion Papers

Die Fakultät für Wirtschaftswissenschaften der Hochschule Wismar, University of Applied Sciences – Technology, Business and Design bietet die Präsenzstudiengänge Betriebswirtschaft, Wirtschaftsinformatik und Wirtschaftsrecht sowie die Fernstudiengänge Betriebswirtschaft, Business Consulting, Business Systems, Facility Management, Quality Management, Sales and Marketing und Wirtschaftsinformatik an. Gegenstand der Ausbildung sind die verschiedenen Aspekte des Wirtschaftens in der Unternehmung, der modernen Verwaltungstätigkeit, der Verbindung von angewandter Informatik und Wirtschaftswissenschaften sowie des Rechts im Bereich der Wirtschaft.
Nähere Informationen zu Studienangebot, Forschung und Ansprechpartnern finden Sie auf unserer Homepage im World Wide Web (WWW): https://www.fww.hs-wismar.de/.

Herausgeber: Prof. Dr. Hans-Eggert Reimers
Fakultät für Wirtschaftswissenschaften
Hochschule Wismar
University of Applied Sciences – Technology, Business and Design
Philipp-Müller-Straße
Postfach 12 10
D – 23966 Wismar
Telefon: ++49/(0)3841/753 7601
Fax: ++49/(0)3841/753 7131
E-Mail: hans-eggert.reimers@hs-wismar.de

Vertrieb: Fakultät für Wirtschaftswissenschaften
Hochschule Wismar
Postfach 12 10
23952 Wismar
Telefon: ++49/(0)3841/753-7468
Fax: ++49/(0) 3841/753-7131
E-Mail: Silvia.Kaetelhoen@hs-wismar.de
Homepage: https://www.fww.hs-wismar.de/

ISSN 1612-0884
ISBN 978-3-942100-65-6
JEL- Klassifikation: C61

Printed in Germany

Inhaltsverzeichnis

1. Einleitung

Seit mehreren Jahren beschäftigt sich der Autor mit Tourenoptimierungsproblemen (siehe [MuRo2006], [Mu2011], [Mu2012], [Mu2017] und [Mu2018]). Dabei konnten Erfolge für bis zu zehn zu beliefernden Orten erzielt werden, aber nicht mehr, weil es sich um NP-schwierige Probleme handelt. (NP-Schwere bezeichnet eine Eigenschaft eines algorithmischen Problems. Dabei steht NP für nichtdeterministische Polynomialzeit.)
Für derartige NP-schwierige Optimierungsprobleme gibt es im Operations Research (OR) gewisse Frameworks, die wenigstens für praxisrelevante Eingabedaten (z. B. 100 Orte) und Parameter diese Probleme lösen. Diese Frameworks arbeiten mit math. Verfahren, z. B. dem Branch-And-Price-Verfahren (kurz BaP), die für sehr große Lineare Programme (100000 Variablen und mehr) ausgedacht wurden. Das Ziel der Arbeit bestand darin, eine Arbeitsumgebung für das Brand-and-Price-Verfahren der Optimierung bereitzustellen und die Eignung der Arbeitsumgebung anhand eines selbst programmierten Beispiels nachzuweisen. Gleichzeitig soll eine Empfehlung für eines der am Markt verfügbaren Frameworks gegeben werden. Das Thema ist sehr komplex, deshalb sind Kennnisse und Fähigkeiten in math. Modellierung und Programmierung erforderlich. Die Programmpakete sind sehr umfangreich und schlecht dokumentiert. Durch diese Arbeit soll Licht ins Dunkel gebracht werden.
Das Programmpaket 'SCIP' (siehe [SCIP]) besteht z. B. aus 270.000 Quelltextzeilen, woran zehn Mitarbeiter 12 Jahre programmiert haben. Im Rahmen dieser Arbeit sollte auch nach Alternativen gesucht werden, die für Master-Studenten besser handhabbar sind als SCIP. (Es wurde in dieser Arbeit die Version 3.0.2 verwendet.) Im Sinne eines Tutoriums sollten auch didaktische Beispiele zur Theorie angefertigt werden, denn ohne theoretisches Hintergrundwissen ist die Software nicht zu bedienen.
Ausserdem ist für die Software möglichst unter gängigen integrierten Entwicklungsumgebungen (engl. Integrated Development Environment (IDE)), wie z. B. Netbeans , die Anwendbarkeit zu erreichen. Damit wird allen Masterstudenten ein leicht bedienbares Werkzeug für die Optimierung komplexer Problemstellungen zur Verfügung gestellt, das sie im Rahmen ihrer Masterthesis für geeignete Themen nutzen können. Zur Illustration werden Beispiele aus folgenden Problemklassen verwendet: Tourenoptimierung, eindimensionales Zuschneiden (Cutting Stock) sowie das Behälterproblem (Bin-Packing).

2. Beschreibung des Problemtyps 'VRPTW'

Der Ausgangspunkt dieses Berichtes ist das VRPTW, weil der Autor dafür selber schon eigene Modelle und Algorithmen erstellt hat (siehe [Mu2012]).
Das sogenannte *Vehicle Routing Problem with Time Windows (VRPTW)* (ohne Mehrfachanfahrten auch splitting genannt) kann wie folgt umgangssprachlich und formal beschrieben werden:
Umgangssprachlich: Es sind diverse Kunden gegeben, die durch einen Standort und einen Bedarf an Gütern beschrieben werden. Der Bedarf der Kunden soll durch diverse Fahrzeuge mit gleicher Kapazität gedeckt werden, die von einem Depot die Güter abholen und am Ende, wenn sie leer sind, wieder ins Depot zurückkehren. Die Fahrzeuge sollen möglichst kurze Wege dabei fahren, d. h. die Summe aller zurückgelegten Strecken soll minimal sein. Gesucht ist ein Tourenplan mit diesen Eigenschaften, wobei noch einschränkend gilt, dass jeder Kunde nur einmal angefahren werden darf und zwar in dem von ihm vorgegebenen einen Zeitfenster. Es kann vorkommen, dass Fahrzeuge warten müssen, bis das Zeitfenster aufgeht.
Formal: Gegeben sei ein vollständig gerichteter Graph $G = (V, A)$, wobei $V = \{v_0, ..., v_n\}$ die Menge von Eckpunkten (Englisch: vertex) und A die Menge von Bögen (Englisch: arc) sei. Der Eckpunkt v_0 ist ein besonderer Eckpunkt und wird Depot genannt. Die Eckpunkte v_1 bis v_n repräsentieren die Kundenorte. Für jeden Bogen $(v_i, v_j) \in A$ gibt es einen Kostenwert c_{ij} und eine Fahrzeit t_{ij}. Allen Knoten $v_i \in V\backslash\{v_0\}$ sind ein positiver Bedarf d_i, ein Zeitfenster $[a_i, b_i]$ sowie eine positive Servicezeit (zum Abladen) $serv_i$ zugewiesen. Eine Flotte von U (beliebig aber fest) vielen Fahrzeugen mit der Kapazität Q steht für die Bedienung der Kunden zur Verfügung. Die Fahrzeuge müssen ihre Routen innerhalb des Zeitfensters $[a_0, b_0]$ des Depots beginnen und beenden. Die kumulative Summe der bedienten Bedarfe innerhalb einer Fahrzeugroute darf die Fahrzeugkapazität Q nicht überschreiten. Die Abladung der Waren für einen Kunden muss innerhalb seines Zeitfensters stattfinden, evt. muss ein Fahrzeug warten bis das Zeitfenster aufgeht, falls es zu früh ankommt. Es sind höchstens U Routen zu finden, auf denen alle Kunden unter den gesetzten Bedingungen einmalig beliefert werden und die minimale Kosten verursachen. Kosten und Fahrzeiten werden als identisch angesehen und müssen die Dreiecksungleichung erfüllen.

Die Zielfunktion für dieses Optimierungsproblem lautet demnach:

$$minimize \sum_{l \le u \le U} \sum_{(v_i, v_j) \in A} c_{ij} x_{ij}^u \tag{1}$$

Folgende Nebenbedingungen sind von zulässigen Lösungen einzuhalten:

$$\sum_{l \le u \le U} \sum_{\{v_j \in V | (v_i, v_j) \in A\}} x_{ij}^u \ge 1 \quad (v_i \in V \setminus \{v_0\}), \tag{2}$$

(Mit dieser Nebenbedingung (2) wird sichergestellt, dass jeder Kunde aufgesucht wird.)

$$\sum_{\{v_j \in V | (v_i, v_j) \in A\}} x_{ij}^u - \sum_{\{v_j \in V | (v_j, v_i) \in A\}} x_{ji}^u = 0 \quad (v_i \in V, 1 \leq u \leq U), \tag{3}$$

$$\sum_{\{v_i \in V | (v_0, v_i) \in A\}} x_{0i}^u \leq 1 \quad (1 \leq u \leq U), \tag{4}$$

(Mit den Nebenbedingungen (3) und (4) wird sichergestellt, dass jeder Kunde genauso oft aufgesucht wird, wie er verlassen wird und dass das Depot nur einmal verlassen wird.)

$$\sum_{(v_i, v_j) \in A} d_i x_{ij}^u \leq Q \quad (1 \leq u \leq U), \tag{5}$$

(Mit der Nebenbedingung (5) wird sichergestellt, dass jeder LKW auf einer Tour nicht mehr als seine Kapazität ausliefern kann.)
Die folgenden beiden Nebenbedingungen garantieren das Einhalten aller Zeitfenster bei den Kunden, wobei M eine sehr große ganze Zahl sei:

$$s_i^u + serv_i + c_{ij} - s_j^u + M x_{ij}^u <= M \quad ((v_i, v_j) \in A, v_j \neq v_0, 1 \leq u \leq U) \tag{6}$$

(Das Abladen beim Kunden j kann erst nach Ankunft dort beginnen, d. h. s_j^u muss grösser sein als die Summe $s_i^u + serv_i + c_{ij}$.)
und im Depot:

$$s_i^u + serv_i + c_{i0} - b_0 + M x_{i0}^u \leq M \quad ((v_i, v_0) \in A, 1 \leq u \leq U) \tag{7}$$

(Die Servicezeit beim letzten Kunden darf nicht dazu führen, dass der LKW erst nach Schliessung des Depots dort eintrifft. Das wäre der Fall, wenn die Summe der ersten 3 Summanden minus der Schliesszeit des Depots positiv wären.)

$$a_i \leq s_i^u \leq b_i (v_i \in V, 1 \leq u \leq U) \tag{8}$$

(Die Servicezeit muss innerhalb des Zeitfensters beginnen.)

$$x_{ij}^u \in \{0, 1\} ((v_i, v_j) \in A, 1 \leq u \leq U) \tag{9}$$

(Die Entscheidungsvariable x_{ij}^u ist binär.)
x_{ij}^u und s_i^u sind Entscheidungsvariablen, wobei die Erste gleich 1 ist, wenn der Bogen von i nach j durch den LKW u befahren wird und sonst gleich 0 ist und die Zweite die Startzeit des Service beim Kunden i darstellt. Die Abfahrtzeit im Depot wird durch s_0^u modelliert.

3. Die BaP-Methode für das VRPTW-Problem

3.1. Das Problem

Sei Ω die Menge aller gültigen Fahrzeugrouten, d. h. die Menge der Wege in G, ausgehend und zurückkehrend vom/zum Depot, den Kapazitäts und Zeitfensterbedingungen genügend und mindestens jeden Kunden einmal aufsuchend.
Seien c_k die Kosten der Route $r_k \in \Omega$. Sei weiterhin $a_{ik} = 1$, wenn die Route r_k den Kunden v_i aufsucht und sonst gleich 0. Sei weiterhin $b_{ijk} = 1$, wenn die Route r_k den Bogen (v_i, v_j) nutzt und sonst gleich 0.
Per Definitionem gilt $c_k = \sum_{(v_i,v_j)\in A} b_{ijk}c_{ij}$ und $a_{ik} = \sum_{\{v_j\in V|(v_i,v_j)\in A\}} b_{ijk}$

Das Ausgangsproblem kann dann wie folgt als **Mengenüberdeckungsproblem** beschrieben werden:

$$minimize \sum_{r_k\in\Omega} c_k\theta_k \tag{10}$$

Gültige Lösungen müssen folgenden Nebenbedingungen genügen:

$$\sum_{r_k\in\Omega} a_{ik}\theta_k \geq 1 \quad (v_i \in V \setminus \{v_0\}), \tag{11}$$

$$\sum_{r_k\in\Omega} \theta_k \leq U, \tag{12}$$

$$\theta_k \in \mathbb{N} \quad (r_k \in \Omega). \tag{13}$$

Die Variable θ_k zeigt an, ob die Route r_k in der Lösung verwendet wird ($\theta_k = 1$) oder nicht ($\theta_k = 0$).
Die Bedingung (11) ergibt sich aus Bedingung (2) und garantiert den Service bei jedem Kunden. Die Bedingung (12) begrenzt die Anzahl verwendeter LKWs. Es soll darauf hingewiesen werden, dass in der Nebenbedingung (13) die Variablen θ_k ganzzahlig und nicht binär definiert wurden. Der Grund dafür ist die Vermeidung der Nebenbedingung $\theta_k \leq 1$ in der linearen Relaxation des in den Formeln (10) bis (13) definierten Problems. Eine Lösung mit $\theta_k \geq 2$ kann nicht optimal sein wegen des Satzes von Dror und Trudeau.
Die neue Problemformulierung in (10-13) wurde gewählt, weil man damit eine bessere lineare Relaxation als im Ursprungsproblem (1-9) erwartet. Jede lineare Relaxation der zweiten Problemformulierung kann man in eine lineare Relaxation des Ursprungsproblems überführen aber nicht umgekehrt. Dieser Sachverhalt war auch zu erwarten, weil man ja schon Aufwand in die Bildung der Menge Ω gesteckt hat.
Trotzdem soll der Sachverhalt an einem Beispiel mit zwei Kunden verdeutlicht werden. Die lineare Relaxation des abgeleiteten Problems in (10-13) liefert hier sogar die optimale Lösung 40, wogegen die lineare Relaxation des Ausgangsproblems in (1-9) einen Zielfunktionswert von 2 liefert. Dies zeigt den großen Unterschied beider Lösungen.

Die Bedarfe im Beispiel von Abbildung 1 seien gleich 1 und die Servicezeiten bei beiden Kunden gleich 0, die LKW-Kapazität sei gleich 10. Eine Lösung der linearen Relaxation des abgeleiteten Problems in (10-13) besteht aus zwei Pendeltouren zu den Kunden. Die Lösung der linearen Relaxation des Ursprungsproblems in (1-9) beinhaltet die Werte:
$s_1^1 = s_1^2 = s_2^1 = s_2^2 = 10$ sowie $x_{12}^1 = x_{12}^2 = x_{21}^1 = x_{21}^2 = 0.5$.

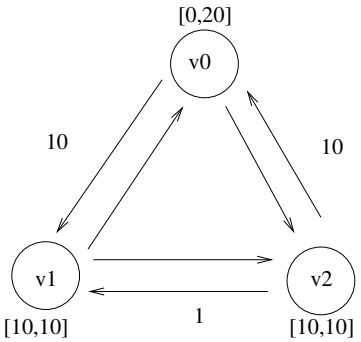

Abbildung 1: Beispiel für ein VRPTW mit zwei Kunden

3.2. Spaltengenerierung

Unglücklicherweise ist auch das Problem (10-13) mit dem Standardverfahren 'Branch-and-Bound' nicht gut lösbar, weil die Mächtigkeit der Menge Ω mit wachsender Anzahl von Kunden exponentiell schnell wächst. Bei 50 Kunden kann die Menge Ω bis zu 1 Billiarde, 125 Billionen, 899 Milliarden, 906 Millionen, 842 Tausend, 623 zulässige Touren aufweisen und bei 100 Kunden sogar 1 Quintillion, 267 Quadrilliarden , 650 Quadrillionen, 600 Trilliarden, 228 Trillionen, 229 Billiarden, 401 Billionen, 496 Milliarden, 703 Millionen, 205 Tausend, 375 Touren (1267650600228229401496703205376).
Dieses Problem versucht man für weniger als 100 Kunden mit einem Spalten-Generierungsverfahren zu bewältigen, indem anstelle von Branch-and-Bound das Verfahren 'Branch-and-Price' angewendet wird.
Die lineare Relaxation von (10-13) wird Master-Problem (kurz MP) genannt und die Beschränkung auf eine Teilmenge $\Omega_1 \subset \Omega$ heisst eingeschränktes Master-Problem (engl. restricted Master Problem oder kurz RMP oder auch $MP(\Omega_1)$).

$$minimize \sum_{r_k \in \Omega_1} c_k \theta_k \tag{14}$$

Gültige Lösungen müssen folgenden Nebenbedingungen genügen:

$$\sum_{r_k \in \Omega_1} a_{ik} \theta_k \geq 1 \quad (v_i \in V \setminus \{v_0\}), \tag{15}$$

$$\sum_{r_k \in \Omega_1} \theta_k \leq U, \tag{16}$$

$$\theta_k \geq 0 \quad (r_k \in \Omega_1). \tag{17}$$

Sei $D(\Omega_1)$ das duale Programm von $MP(\Omega_1)$:

$$maximize \sum_{v_i \in V \setminus \{v_0\}} \lambda_i + U\lambda_0 \tag{18}$$

Unter den Nebenbedingungen:

$$\sum_{v_i \in V \setminus \{v_0\}} a_{ik}\lambda_i + \lambda_0 \leq c_k \quad (r_k \in \Omega_1), \tag{19}$$

$$\lambda_i \geq 0 \quad (v_i \in V \setminus \{v_0\}), \tag{20}$$

$$\lambda_0 \leq 0. \tag{21}$$

In diesem Modell sind λ_i die nicht negativen Dualvariablen, die verbunden sind mit dem Besuch eines Kunden v_i (Bedingung (15)), und λ_0 ist die nicht positive Dualvariable, die verbunden ist mit der Flottengrößenbedingung (16).
Der Spaltengenerierungsalgorithmus basiert auf folgendem Satz:
Satz 1 Sei Ω_1 eine Teilmenge von Ω und λ^* die optimale Lösung von $D(\Omega_1)$. Wenn λ^* auch gültig ist für $D(\Omega)$, ist λ^* auch optimal für $D(\Omega)$.
Beweis Der Lösungsraum von $D(\Omega)$ ist eine Teilmenge des Lösungsraumes von $D(\Omega_1)$ und da die Zielfunktionen identisch sind, ist der Beweis erbracht.
Die optimale Lösung von $MP(\Omega_1)$ impliziert die optimale Lösung λ^* von $D(\Omega_1)$. Wenn diese Lösung gültig ist in $D(\Omega)$, dann sind $D(\Omega)$ und $MP(\Omega)$ nach Satz 1 gelöst. Im anderen Fall verletzen Routen von $\Omega \setminus \Omega_1$ eine oder mehrere Nebenbedingungen der Art (19). Diese müssen durch das Finden von optimalen Lösungen sogenannter Subprobleme identifiziert und anschliessend die dazugehörigen Variablen in der Menge $\Omega 1$ ergänzt werden.
Durch das abwechselnde Lösen von $MP(\Omega_1)$ und eines Subproblems nähert man sich der optimalen Lösung.
Wenn keine Nebenbedingungen mehr verletzt werden, ist der Algorithmus der Spaltengenerierung beendet.
Hier noch einmal der Algorithmus der Spaltengenerierung etwas formaler:

`Generiere eine Anfangsmenge von Spalten (Routen)` in $\Omega 1$.
`Wiederhole`
`Loese` $MP(\Omega 1)$
$\Gamma :=$ Spalte aus dem Subproblem
$\Omega 1 := \Omega 1 \cup \Gamma$
`solange` $\Gamma \neq \emptyset$

Abbildung 2: Algorithmus der Spaltengenerierung

Es muss bemerkt werden, dass das Hinzufügen von Variablen zu Ω_1 keine Garantie der Wertverbesserung der optimalen Lösung von $MP(\Omega_1)$ darstellt. Ausserdem muss nach Grünert und Irnich, Band 1, S. 157, die neue Tour keine optimale Lösung des Subproblems sein. Jede Lösung mit negativ reduzierten Kosten kann aufgenommen werden. Es können auch mehrere solcher Lösungen simultan in das RMP aufgenommen werden.

Der Name 'Spaltengenerierung' kommt daher, dass die dualen Nebenbedingungen in (19) des Problems $D(\Omega_1)$ zu Spalten (primäre Variablen) in $MP(\Omega_1)$ gehören, die die Kosten in $MP(\Omega_1)$ reduzieren.

3.3. Das Subproblem

Gesucht werden Routen $r_k = \Omega \setminus \Omega_1$, die die Bedingung (19) verletzen, dass also gilt:

$$c_k - \sum_{v_i \in V \setminus \{v_0\}} a_{ik}\lambda_i^* - \lambda_0^* \leq 0, \tag{22}$$

Routen mit der Eigenschaft (22) werden Routen mit reduzierten Kosten genannt. Die Bedingung (22) kann umgeformt werden zu:

$$\sum_{(v_i,v_j) \in A} b_{ijk}(c_{ij} - \lambda_i^*) \leq 0 \tag{23}$$

Es ist günstiger, die Suche auf die Menge Ω auszudehnen. Jede Route aus Ω_1 erfüllt diese Bedingung.
Routen, die die Eigenschaft (23) erfüllen, werden ab sofort Routen mit negativ reduzierten Kosten genannt.
Beim Subproblem handelt es sich um ein 'Kürzester Pfad'-Problem mit Ressouren-Bedingungen (ESPPRC). Dabei muss man elementare Wege vom Depot zum Depot finden, die die Zeitfenster- und Kapazitätsbedingungen einhalten und negativ reduzierte Kosten besitzen. Die Eigenschaft, dass der Pfad elementar sein soll, d. h. jeder Eckpunkt und jeder Bogen tritt nur einmal auf, ist nicht trivialerweise erfüllt! Leider ist das ESPPRC ebenfalls NP-schwierig.

3.4. Veranschaulichung des Algorithmus zur Spaltengenerierung am Beispiel

Das folgende Beispiel stammt aus [Feillet2010].
Es wird von folgenden Ortslagen von Depot und Kunden sowie Zeitfenstern ausgegangen. Die Serviczeiten seinen für alle Kunden gleich 0, ihre Bedarfe gleich 1. Die Anzahl der LKW sei 10, also irrelevant, ebenso sei die LKW-Kapazität gleich 10, also ebenfalls irrelevant.
Die Spaltengenerierung beginnt mit einer Anfangsmenge $\Omega 1$, die nur aus den drei Pendeltouren r_1, r_2 und r_3 zu den Kunden v_1, v_2 und v_3 besteht. Das eingeschränkte (engl. restricted) Masterprogramm und das dazugehörige Duale Programm ergeben sich dann wie folgt:

Die optimalen Lösungen mit einem Zielfunktionswert von **6,8** beider Linearen Programme erkennt man offensichtlich als:
$\theta = (1, 1, 1)$ sowie $\lambda = (2, 2.8, 2)$.
Das Subproblem wird die Route $r_4 = v_0 \to v_1 \to v_2 \to v_0$ als Route mit negativ reduzierten Kosten erkennen. Sie errechnen sich nach Formel (22) zu: $3,4 - 2 - 2,8 = -1,4$.

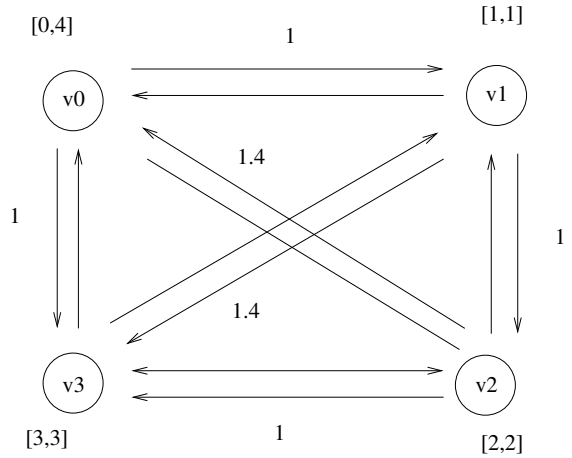

Abbildung 3: Beispiel für ein VRPTW mit drei Kunden

$$
\begin{array}{lll}
\textit{minimize} & 2\theta_1 + 2.8\theta_2 + 2\theta_3 & \\
\text{unter den Nebenbedingungen} & & \\
\theta_1 & \geq 1 & \\
\theta_2 & \geq 1 & \\
\theta_3 & \geq 1 & \\
\theta_1, \theta_2, \theta_3 & \geq 0 &
\end{array}
\qquad
\begin{array}{lll}
\textit{maximize} & \lambda_1 + \lambda_2 + \lambda_3 & \\
\lambda_1 & \leq 2 & \\
\lambda_2 & \leq 2.8 & \\
\lambda_3 & \leq 2 & \\
\lambda_1, \lambda_2, \lambda_3 & \geq 0 &
\end{array}
$$

Abbildung 4: Primales und Duales Linerares Programm für drei Spalten

Die Route wird r_4 genannt und zu Ω_1 hinzugefügt. Dadurch gibt es im $MP(\Omega_1)$ eine neue Spalte θ_4:

$$
\begin{array}{lll}
\textit{minimize} & 2\theta_1 + 2.8\theta_2 + 2\theta_3 + 3.4\theta_4 & \\
\text{unter den Nebenbedingungen} & & \\
\theta_1 + \theta_4 & \geq 1 & \\
\theta_2 + \theta_4 & \geq 1 & \\
\theta_3 & \geq 1 & \\
\theta_1, \theta_2, \theta_3, \theta_4 & \geq 0 &
\end{array}
\qquad
\begin{array}{lll}
\textit{maximize} & \lambda_1 + \lambda_2 + \lambda_3 & \\
\lambda_1 & \leq 2 & \\
\lambda_2 & \leq 2.8 & \\
\lambda_3 & \leq 2 & \\
\lambda_1 + \lambda_2 & \leq 3.4 & \\
\lambda_1, \lambda_2, \lambda_3 & \geq 0 &
\end{array}
$$

Abbildung 5: Primales und Duales Linerares Programm für vier Spalten

Die optimalen Lösungen mit einem Zielfunktionswert von **5,4** beider Linearen Programme erkennt man offensichtlich als:
$\theta = (0, 0, 1, 1)$ sowie $\lambda = (2, 1.4, 2)$.
Das Subproblem schlägt dann die Route $r_5 = v_0 \to v_1 \to v_2 \to v_3 \to v_0$ mit den reduzierten Kosten $4 - 2 - 1, 4 - 2 = -1, 4$ zur Aufnahme in die Menge Ω_1 vor.

Die Lösungen $\theta = (0, 0, 0, 0, 1)$ und $\lambda = (2, 1.4, 0.6)$ sind mit einem Zielfunktionswert von **4** optimal.
Man kann leicht sehen, dass nur zwei Routen in Ω_1 fehlen und dass die reduzierten Kosten beider Routen positiv sind, der Algorithmus in Abbildung 2 also

$$
\begin{array}{ll}
\textit{minimize} & 2\theta_1 + 2.8\theta_2 + 2\theta_3 + 3.4\theta_4 + 4\theta_5 \\
\text{unter den Nebenbedingungen} & \\
\theta_1 + \theta_4 + \theta_5 \geq 1 & \\
\theta_2 + \theta_4 + \theta_5 \geq 1 & \\
\theta_3 + \theta_5 \geq 1 & \\
\theta_1, \theta_2, \theta_3, \theta_4, \theta_5 \geq 0 &
\end{array}
\qquad
\begin{array}{ll}
\textit{maximize} & \lambda_1 + \lambda_2 + \lambda_3 \\
\lambda_1 \leq 2 & \\
\lambda_2 \leq 2.8 & \\
\lambda_3 \leq 2 & \\
\lambda_1 + \lambda_2 \leq 3.4 & \\
\lambda_1 + \lambda_2 + \lambda_3 \leq 4 & \\
\lambda_1, \lambda_2, \lambda_3 \geq 0 &
\end{array}
$$

Abbildung 6: Primales und Duales Linerares Programm für fünf Spalten

stoppt. Die Lösung der linearen Relaxation stellt in diesem Fall direkt die optimale Lösung dar.
Leider wird in [Feillet2010] nicht das Sub-Programm angegeben. Der Autor hat deshalb folgendes Sub-Programm in GMPL formuliert.

```
# n+1 Orte in Distanzmatrix
# Kunden 1..n, Depot 0
param n, integer, >1;
param q,integer,>1; # Ladekapazitaet des LKWs
/* ein zeitfenster fuer  Depot und je Kunde */
param ab1{ 0..n}, integer, >=0;
param bis1{0..n}, integer;
/* Dualvariablenwerte*/
param lambda{0..n};
/* Reihe der Kunden-Bedarfe */
param b{1..n};
set   A := {(i,j) in 0..n cross 0..n: i!=j};
# HD-freier Matrixtyp
param d{A};
var x{(i,j) in A},integer, >= 0;
/* x[i,j] = 1 means that edge (i,j) belong to shortest path;
   x[i,j] = 0 means that edge (i,j) does not belong to shortest path;
   note that variables x[i,j] are binary, however, there is no need to
   declare them so due to the totally unimodular constraint matrix */
/* Ankunftszeit */
var ank{i in 0..n},>=0;
/* ins Depot darf nur einer raus und einer rein */
s.t.  r2:  sum {i in 1..n} x[0,i]=1;
s.t.  r3:  sum {i in 1..n} x[i,0]=1;
/* Flussbedingung  NB34*/
s.t. r{i in 1..n}: sum{(j,i) in A} x[j,i]  =  sum{(i,j) in A} x[i,j] ;
/* Zeitfensterbedingung NB35*/
s.t.ankunft{(i,j) in A:j!=0}: ank[i] +d[i,j] -BIG*(1 -x[i,j]) <= ank[j];
/* Zeitfensterbedingung NB36*/
s.t.ab{i in 0..n}: ab1[i]<= ank[i];
s.t.bis{i in 0..n}:bis1[i]>=ank[i];
/*Zielfunktion*/
```

```
minimize Z: sum{(i,j) in A} (d[i,j] -lambda[i]) * x[i,j];
/* objective function is the path length to be minimized */
data;
/* Anzahl Kunden */
param n := 3;
param q      := 40; #Ladekapazitaet
#Zeitfenster
param ab1 :=  0 0,1 1,2 2,3 3;
param bis1:=  0 4,1 1,2 2,3 3;
#Bedarfe
param b:= 1 1,2 1,3 1,4 1;
/* 1. Iterationsschritt  */
param lambda:= 0 0,1 2,2 2.8,3 2;
param d   # Distanzmatrix tutorial, 0=Depot
:  0   1   2   3   :=
0  .   1  1.4  1
1  1   .  1    1.4
2  1.4  1   .  1
3  1   1.4 1  .
;
end;
```

Dieses Sub-Programm schlägt dann sofort die Route:
$r_4 = v_0 \to v_1 \to v_2 \to v_3 \to v_0$ mit den reduzierten Kosten $4 - 2 - 2,8 - 2 = -2.8$ zur Aufnahme in die Menge Ω_1 vor. Das nächste RMP liefert dann den Zielfunktionswert von 4.
Das daraus abgeleitete Sub-Programm findet jetzt keine Routen mit negativ reduzierten Kosten mehr. Somit endet das Verfahren der Spaltengenerieurng.
Der Vollständigkeit halber sei noch das zu Abbildung Nr.4 analoge RMP im IBM-LP-Format notiert.

```
Minimize
AnzahlTouren:  2 y1 + 2.8 y2 + 2 y3 + 4 y4
Subject To
 B1: y1           + y4 >= 1
 B2:      y2      + y4 >= 1
 B3:           y3 + y4 >= 1
 Bounds
  y1 >= 0
  y2 >= 0
  y3 >= 0
  y4 >= 0
End
```

Die primale und duale Lösung lautet (0,0,0,1) sowie (2,2,0). Kurioserweise schlägt das Sub-Programm jetzt wieder die Rundtour:

$v_0 \to v_1 \to v_2 \to v_3 \to v_0$ vor, die allerdings keine negativ reduzierten Kosten $4 - 2 - 2 = 0$ mehr aufweist.
Es sei noch auf eine Besonderheit hingewiesen. Eigentlich wird bei dem Problem des kürzesten Weges von einer Adjazenzmatrix ausgegangen, die die Dreiecksungleichung verletzt. In dieser Arbeit wurde aber immer deren Gültigkeit eingefordert. Das ist kein Widerspruch, denn durch den Subtrahenten 'Lambda' in der Zielfunktion geht die Dreiecksungleichung verloren.

4. Beschreibung des Problems 'Eindimensionales Zuschneiden'

Zur Demonstration des Softwarepaketes 'JORLIB' (Java OR Library, siehe [JORLIB2015]) soll das eindimensionale Zuschnittproblem herangezogen werden. Dazu wird ein leicht abgewandeltes Zahlenbeispiel aus [LUTZ98] verwendet.
Zahlenbeispiel 20 m lange Holzleisten stehen zur Verfügung, woraus 60 Stück 12m-Leisten, 120 Stück 7m-Leisten sowie 150 Stück 3m-Leisten so zuzuschneiden sind, dass möglichst wenig Abfall auftritt.
Es gibt hier nur fünf Varianten (die keine wesentlichen Reste lassen) des Zuschnitts (Schnittmuster oder engl. pattern), was aber untypisch für diesen Problemtyp ist. Beim folgenden Beispiel gibt es für 13 Leistenarten bereits 308 verschiedene Schnittmuster.

Schnittmuster-Nr.	Anzahl 12m	Anzahl 7m	Anzahl 3m	Rest(m)
1	1	1	0	1
2	1	0	2	2
3	0	2	2	0
4	0	1	4	1
5	0	0	6	2

Tabelle 1: Schnittmuster für Zahlenbeispiel

Das ganzzahlig lineare Programm (IP) lautet dann schon in einer für die Spaltengenerierung geeigneten Form:

```
x1 + x2 +  x3 +     x4 +    x5  -> minimum

x1 + x2                              >= 60

x1       + 2 x3     + x4             >= 120

    2 x2 + 2 x3    + 4 x4 + 6 x5     >= 150

x1 bis x5  >=0 ganzzahlig
```

Abbildung 7: IP für alle sinnvollen Schnittmuster der Abb. 8

Hierbei ist x1 die Anzahl der Zuschnitte nach Schnittmuster Nr.1, x2 nach Schnittmuster Nr.2 usw.
Beim Verfahren der Spaltengenerierung verwendet man nicht sofort alle Zuschnittvarianten, sondern beginnt mit einer zulässigen aber einfachen Anfangslösung, die hier darin besteht, dass man aus jeder 20m-langen Holzleiste nur Teile einer Art herstellt und lässt die Forderung nach Ganzzahligkeit der Lösung fallen.
Die Lösung dieses LPs lautet Obj=145 und $x = (60, 60, 25)$.

```
Minimize
AnzahlLeisten:  x1 + x2 + x3
Subject To
 AnzahlArt1: x1                  >= 60
 AnzahlArt2:       2 x2          >= 120
 AnzahlArt3:               6 x3  >= 150
Bounds
  x1 >= 0
  x2 >= 0
  x3 >= 0
Generals
x1 x2 x3
End
```

Abbildung 8: Master-LP mit der Anfangslösung im IBM-Format

Das dazu duale LP lautet:

```
Maximize
DualObj:  60 pi1 + 120  pi2 + 150 pi3
Subject To
 B1:   pi1 <= 1
 B2: 2 pi2 <= 1
 B3: 6 pi3 <= 1
Bounds
  pi1 >= 0
  pi2 >= 0
  pi3 >= 0
Generals
pi1 pi2 pi3
End
```

Abbildung 9: Duales - Master-LP im IBM-Format

Die Lösung lautet Obj=145 und $pi = (1, 0.5, 0.166)$.
Es kann nun das Sub-IP (Rucksack-Problem) formuliert werden, um ein neues erfolgsversprechendes Zuschnittsmuster zu erhalten:
Der maximale Wert des 'Rucksacks' wird mit 1,5 erreicht bei $a = (1, 1, 0)$. Das neue Schnittmuster besteht also aus einem Teil der Art 1 (12 m) und einem Teil der Art 2 (7m). Die reduzierten Kosten betragen durch das neue Schnittmuster: 1-1-0.5-0.166= -0,666.

```
Maximize
SubObj:  a1 + 0.5 a2 + 0.166 a3
Subject To
 Laengenbegrenzung: 12 a1 + 7 a2 + 3 a3 <= 20
 Bounds
  a1 <= 1
  a2 <= 2
  a3 <= 6
Integer
a1 a2 a3
Generals
a1 a2 a3
End
```

Abbildung 10: Subproblem im IBM-Format

Die Erweiterung des Masterprogramms in Abb. 10 durch eine Variable für dieses neue Schnittmuster ergibt:

```
Minimize
AnzahlRollen:  x1 + x2      + x3    + x4
Subject To
 AnzahlArt1:   x1                  +  x4  >= 60
 AnzahlArt2:         2 x2          +  x4  >= 120
 AnzahlArt3:                6 x3          >= 150
Bounds
  x1 >= 0
  x2 >= 0
  x3 >= 0
  x4 >= 0
Generals
x1 x2 x3 x4
End
```

Abbildung 11: Master-LP mit der Anfangslösung plus eine neue Spalte(Variable) im IBM-Format

Die Lösung dieses LPs lautet Obj=115 und $x = (0, 30, 25, 60)$.

Das dazu duale LP lautet:

```
Maximize
DualObj:  60 pi1 + 120  pi2 + 150 pi3
Subject To
 B1: pi1            <= 1
 B2:     2 pi2      <= 1
 B3:          6 pi3 <= 1
 B4: pi1 + pi2      <= 1
Bounds
  pi1 >= 0
  pi2 >= 0
  pi3 >= 0
Generals
pi1 pi2 pi3 End
```

Abbildung 12: Duales - Master-LP im IBM-Format

Die Lösung hierfür lautet Obj=115 und $pi = (0.5, 0.5, 0.166)$.
Es kann nun das Sub-IP (Rucksack-Problem) formuliert werden, um evt. ein weiteres neues erfolgsversprechendes Zuschnittsmuster zu erhalten:

```
Maximize
Rucksackwert:  0.5 a1 + 0.5 a2 + 0.166 a3
Subject To
 Maximallaenge : 12 a1 + 7 a2 + 3 a3 <= 20
 Bounds
  a1 <= 1
  a2 <= 2
  a3 <= 6
Integer
a1 a2 a3
Generals
a1 a2 a3
End
```

Abbildung 13: Subproblem Nr.2 im IBM-Format

Der maximale Wert des 'Rucksacks' wird mit 1,332 erreicht bei $a = (0, 2, 2)$. Das neue Schnittmuster besteht also aus zwei Teilen der Art 2 (7 m) und zwei Teilen der Art 3 (3m). Die reduzierten Kosten betragen durch das neue Schnittmuster: 1-0.5-0.5-0.166= -0,166.
Das Masterprogramm in Abb. 13 wird durch eine Variable für dieses neue Schnittmuster erweitert und man erhält:

```
Minimize
AnzahlRollen:  x1 + x2 + x3 + x4 + x5
Subject To
 Anzahlart1: x1 + x4 >= 60
 Anzahlart2: 2 x2 + x4 + 2 x5 >= 120
Anzahlart 3: 6 x3 + 2 x5 >= 150
Bounds
  x1 >= 0
  x2 >= 0
  x3 >= 0
  x4 >= 0
  x5 >= 0
Generals
x1 x2 x3 x4 x5
End
```

Abbildung 14: Master-LP mit der Anfangslösung plus zwei neue Spalten (Variable) im IBM-Format

Die Lösung dieses LPs lautet Obj=105 und $x = (0, 0, 15, 60, 30)$. Die duale Lösung (auf das Vorzeigen des dualen Problems wird an dieser Stelle verzichtet) lautet $pi = (0.66, 0.33, 0.166)$.
Es kann nun das dritte Sub-IP (Rucksack-Problem) formuliert werden, um ein neues, erfolgsversprechendes Zuschnittsmuster zu erhalten:

```
Maximize
Rucksackwert:  0.66 a1 + 0.33 a2 + 0.166 a3
Subject To
 Maximallaenge: 12 a1 + 7 a2 + 3 a3 <= 20
 Bounds
  a1 <= 1
  a2 <= 2
  a3 <= 6
Integer
a1 a2 a3
Generals
a1 a2 a3
End
```

Abbildung 15: Subproblem Nr.3 im IBM-Format

Der maximale Wert des 'Rucksacks' wird mit 1 erreicht bei $a = (0, 0, 6)$. Die reduzierten Kosten betragen durch das neue Schnittmuster: 1–0.166*6=0, d. h. durch das neue Schnittmuster kann es keine Verbesserung der Lösung geben. Der Algorithmus ist beendet. Der optimale Wert lautet 105 Leisten. Sie müssen nach den Schnittmustern 3, 4 und 5 geschnitten werden.

4.1. Allgemeine Mathematische Formulierung für eindimensionales Zuschneiden mit Mustern

Mit dem vorherigen Beispiel fällt die allgemeine Formulierung als IP (Integer-Programm) nicht mehr schwer:

$$IP : minimize \sum_{i=1}^{n} x_i \tag{24}$$

Folgende Nebenbedingung ist von zulässigen Lösungen einzuhalten:

$$\sum_{i=1}^{n} a_{ij} x_i \geq b_j, j = 1...m \tag{25}$$

$$x \geq 0, ganzzahlig \tag{26}$$

Hierbei steht $b = (b_1, ..., b_m)$ für den Bedarfsvektor (m-Anzahl der Teilearten) und $x = (x_1, ..., x_n)$ für den Vektor der Anzahlen Zuschnitte nach allen möglichen Mustern $1..n$ und a_{ij} stellt die Zahl der möglichen Teile der Größe s_j im Muster $i \in 1...n$ dar.
Für alle $j \in 1..m$ gilt $\sum_{i=1}^{n} a_{ij} * s_j \leq W$, wenn W die Länge des Grundmaterials darstellt.
Das Masterproblem MP zu (25-27) ergibt sich, wenn man im IP die Ganzzahligkeitsbedingung (27) weglässt.
Das duale Masterproblem (DMP) lautet dann:

$$DMP : maximize \sum_{j=1}^{m} b_i pi_i \tag{27}$$

und die Nebenbedingungen:

$$\sum_{j=1}^{m} a_{ij} pi_j \leq 1, i = 1...n \tag{28}$$

sowie

$$p_j \geq 0, j = 1...m \tag{29}$$

Nach dem Grundprinzip des Spaltengenerierungsverfahrens müssen also Schnittmuster gefunden werden, die die Bedingung (29) verletzen. Um diese zu finden, wird folgendes Subproblem gebildet:

$$SUB : maximize \sum_{j=1}^{m} pi_j a_j \tag{30}$$

unter den Nebenbedingungen:

$$\sum_{j=1}^{m} w_j a_j \leq L \tag{31}$$

sowie

$$a_j, j = 1..m, ganzzahlig \tag{32}$$

Beim Subproblem ändert sich lediglich die Zielfunktion in der nächsten Iteration. Da das MP eine lineare Relaxation des IPs ist, kann man nicht davon ausgehen, dass die optimale Lösung, die durch die Spaltengenerierung ermittelt wurde, auch die optimale Lösung des IPs ist, also eine ganzzahlige Lösung darstellt. In diesem Fall muss man ein anderes Verfahren anwenden: Branch-and-Price. Beim IP der Abbildung 9 war es zufällig so, dass die Spaltengenerierung schon die optimale Lösung des IPs ermittelte.

4.2. Allgemeine mathematische Formulierung für eindimensionales Zuschneiden ohne Muster

Es wird wieder von N (beliebig kardinal aber fest) Teilearten (Iterator n) ausgegangen, die durch N Größen g_n und N Bedarfe b_n beschrieben werden. Die Bedarfe sollen durch Zuschnitt von M (Iterator m) Grundmaterialien der einheitlichen Länge 'L' gedeckt werden. Die Zahl M wurde durch grobe Abschätzung nach oben ermittelt, indem z. B. nur immer eine Teileart aus dem Grundmaterial herausgeschnitten wird.

$$minimize \sum_{m=1}^{M} y_m, y_m \in \{0, 1\} \tag{33}$$

Zuerst werden die Nebenbedingung zur Deckung der Bedarfe wie folgt formuliert:

$$Bedarfe(n) : \sum_{m=1}^{M} x_{mn} = b_n, n = 1...N \tag{34}$$

$$x_{mn} \geq 0, ganzzahlig \tag{35}$$

x_{mn} stellt die ganzzahlige Anzahl der möglichen Teile der Größe g_n im Grundmaterial $m \in 1...M$ dar.
Es folgt jetzt die Nebenbedingungsformulieurng zur Einhaltung der Länge des Grundmaterials.

$$Längengrenze(m) : \sum_{n=1}^{N} g_n x_{mn} = Ly_m, m = 1...M \tag{36}$$

4.3. Berechnungsergebnisse mit Jorlib für Cutting-Stock

Die folgenden Beispiele zeigen Rechnerausdrucke zuerst eines Beispiels mit ganzzahliger Lösung nach der Spaltengenerierung und danach ein Beispiel mit nicht ganzzahliger Lösung.

```
public final int nrFinals=3; //Number of different finals
public final int rollWidth=20; //Width of the raws
```

```
public final int[] finals={12, 7, 3}; //Size of the finals
public final int[] demandForFinals={60, 120, 150};
//Requested quantity of each final

run:
......

20:38:56.733 [main] DEBUG o.j.f.c.io.SimpleDebugger -
 Finished Column Generation for instance  CuttingStockExample
================ Solution ================
CG terminated with objective: 105.0
Number of iterations: 4
Time spent on master: 15 time spent on pricing: 156
Columns (only non-zero columns are returned):
Value: 15.0 Cutting pattern: [0, 0, 6] creator: exactPricing
Value: 30.0 Cutting pattern: [0, 2, 2] creator: exactPricing
Value: 60.0 Cutting pattern: [1, 1, 0] creator: exactPricing
BUILD SUCCESSFUL (total time: 0 seconds)
```

Die Spaltengenerierung stösst an ihre Grenzen, wie dieses Beispiel zeigt:

```
Eingabedaten:
************
Anzahl Teilearten=6
Laenge Basismaterial =100
Bedarfe:
,1,1,1,1,1,1
Groessen:
,50,60,30,70,50,40

Value: 0.5 Cutting pattern: [2, 0, 0, 0, 0, 0] creator: exactPricing
Value: 0.5 Cutting pattern: [0, 0, 0, 0, 2, 0] creator: exactPricing
Value: 1.0 Cutting pattern: [0, 1, 0, 0, 0, 1] creator: exactPricing
Value: 1.0 Cutting pattern: [0, 0, 1, 1, 0, 0] creator: exactPricing
```

Diese Lösung ist nicht mehr ganzzahlig und offensichtlich würden hier drei statt vier Schnittmuster reichen.

5. Bin-Packing als spezielles Cutting-Stock-Problem

Weil es für das Cutting-Stock-Problem schwierig ist, geeignete Verzweigungsregeln zu finden, wird in dieser Arbeit auf das Bin-Packing-Problem ausgewichen. Es ist ein Spezialfall des Cutting-Stock-Problems, wenn alle Teile nur einmal benötigt werden. Im Subproblem (Rucksack) ist darauf zu achten, dass die Variablen binär sind, jedes Teil also nur einmal im Rucksack erscheinen darf. Das kann man leicht in der JORLIB-Demo 'CuttingStockCG' durch folgenden Quelltext erreichen (Klasse 'ExactPricingProblemSolver', Methode 'build').

```
//Create the variables
//1 fuer Bin-packing
vars=cplex.intVarArray(dataModel.nrFinals,0,/*Integer.MAX_VALUE*/1);
```

Die Rechnerausgaben für ein Bin-Packing-Beispiel sahen wie folgt aus:

```
09:21:34.018 [main] - CG solving CuttingStockExample -
 Initial upper bound: 20
09:21:34.033 [main] - ============== MASTER 1 ==============
09:21:34.033 [main] - Finished master -> CG objective: 5.0,
 CG bound: 2.0, CG cutoff: 20
09:21:34.033 [main] - ============== PRICING 1 ==============
09:21:34.049 [main] - Finished pricing (1 columns generated) ->
 CG objective: 5.0, CG bound: 2.0,
09:21:34.049 [main] - Value: 0.0 Cutting pattern: [1, 1, 1, 0, 0]
 creator: exactPricing
09:21:34.049 [main] - ============== MASTER 2 ==============
09:21:34.049 [main] - Finished master -> CG objective: 3.0,
 CG bound: 2.0, CG cutoff: 20
09:21:34.049 [main] - ============== PRICING 2 ==============
09:21:34.049 [main] - Finished pricing (1 columns generated) ->
 CG objective: 3.0, CG bound: 2.0,
09:21:34.049 [main] - Value: 0.0 Cutting pattern: [0, 0, 1, 1, 1]
 creator: exactPricing
09:21:34.049 [main] - ============== MASTER 3 ==============
09:21:34.049 [main] - Finished master -> CG objective: 3.0,
 CG bound: 2.0, CG cutoff: 20
09:21:34.049 [main] - ============== PRICING 3 ==============
09:21:34.064 [main] - Finished pricing (1 columns generated) ->
 CG objective: 3.0, CG bound: 2.0,
09:21:34.064 [main] - Value: 0.0 Cutting pattern: [0, 1, 0, 1, 1]
 creator: exactPricing
09:21:34.064 [main] - ============== MASTER 4 ==============
09:21:34.064 [main] - Finished master -> CG objective: 2.0,
CG bound: 2.0,
09:21:34.064 [main] - Finished Column Generation for instance
 CuttingStockExample
================ Solution ================
```

```
CG terminated with objective: 2.0
Number of iterations: 4
Time spent on master: 0 time spent on pricing: 31
Columns (only non-zero columns are returned):
Muster=Value: 0.5 Cutting pattern: [1, 0, 0, 0, 0] creator: initSolution
Muster=Value: 0.5 Cutting pattern: [1, 1, 1, 0, 0] creator: exactPricing
Muster=Value: 0.5 Cutting pattern: [0, 0, 1, 1, 1] creator: exactPricing
Muster=Value: 0.5 Cutting pattern: [0, 1, 0, 1, 1] creator: exactPricing
BUILD SUCCESSFUL (total time: 0 seconds)
```

Wie man sieht, ist die Lösung nicht ganzzahlig und bedarf eines Branch-and-Price-Verfahrens. Dazu werden die Verzweigungsregeln aus der Arbeit von [VANCE93] verwendet.
Ebenda ist beschrieben, dass man den rechten Zweig im Verzweigungsbaum durch das Hinzufügen einer Bedingung im Subproblem betritt, die gebrochene Lösungen verhindern soll, hier ein Beispiel dafür in B2:

```
Maximize
DualObj:  a1 + a4 + a5
Subject To
 B1: 138 a1 + 152 a2 + 156 a3 + 171 a4 + 182 a5 <= 560
 B2: a1 + a2 <= 1
 Bounds
  a1 <= 1
  a2 <= 1
  a3 <= 1
  a4 <= 1
  a5 <= 1
Integer
a1 a2 a3 a4 a5
Generals
a1 a2 a3 a4 a5
End
```

Die beiden Indizes in B2 ’1’ und ’2’ ergeben sich aus der Arbeit von [VANCE93].

6. Erfahrungen mit dem Framework 'Jorlib' und der Implementierung von Beispielen

6.1. Installation

Die Installation des Demobeispiels 'CuttingStock' aus dem Jorlib-Projekt unter der Java-IDE 'Netbeans8.2' auf einem Windows-Rechner dauerte alleine zwei Tage. Grundsätzlich liegt es daran, dass Jorlib mit dem Build-Tool 'Maven' erstellt wurde und Netbeans mit dem Build-Tool 'Ant' arbeitet.
Im Detail war die Installation so schwierig, weil die Klasse mit den Testdaten dreimal in den Jorlib-Bibliotheken vorkommt, einmal als .java in der Datei 'jorlib-Demo-111-Source-jar' und zweimal als .class in den Hilfslibraries 'jorlib-demo-111.jar' sowie 'jorlib-demo-111-uber.jar'. Das Demo-CuttingStock-Paket muss in beiden Hilfslibraries gelöscht werden und das neue Netbeansprojekt mit der Option 'Java-Project-with existing-sources' angelegt werden. Zusätzlich zu den beiden Hilfslibraries muss noch eine cplex.jar als Library bekannt gemacht werden und der Parameter '-Djava.library.path' bei der run-property auf den Ordner mit der cplex1261.dll gesetzt werden. Liegt eine andere Version von CPLEX vor, reicht es, die dll-Datei entsprechend umzubennen. Ansonsten verlief die Installation problemlos.

6.2. UML-Klassendiagramm CuttingStock-Demo

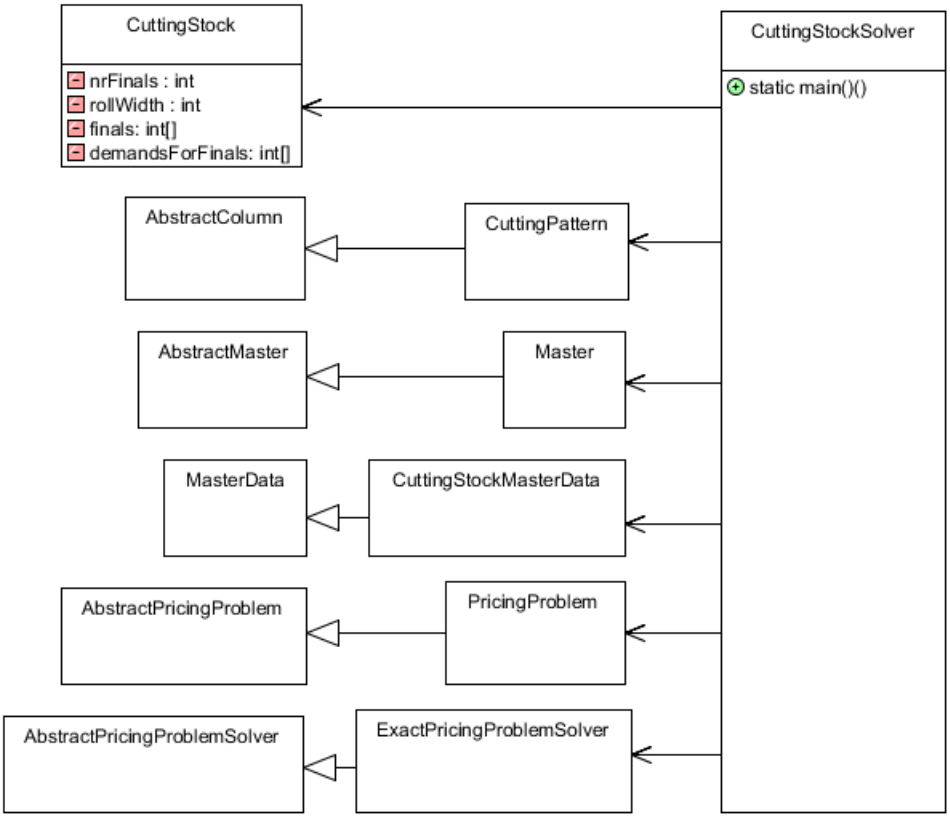

Abbildung 16: UML-Klassendiagramm für ein Cutting-Stock-Programm mit Spaltengenerierung ohne Branch-Verfahren

Dieses Klassendiagramm modelliert eine Lösung mit einfacher Spaltengenerierung aus dem JORLIB-Demo-Paket.

6.3. UML-Klassendiagramm Colorierung eines Graphen mit Branch-and-Price

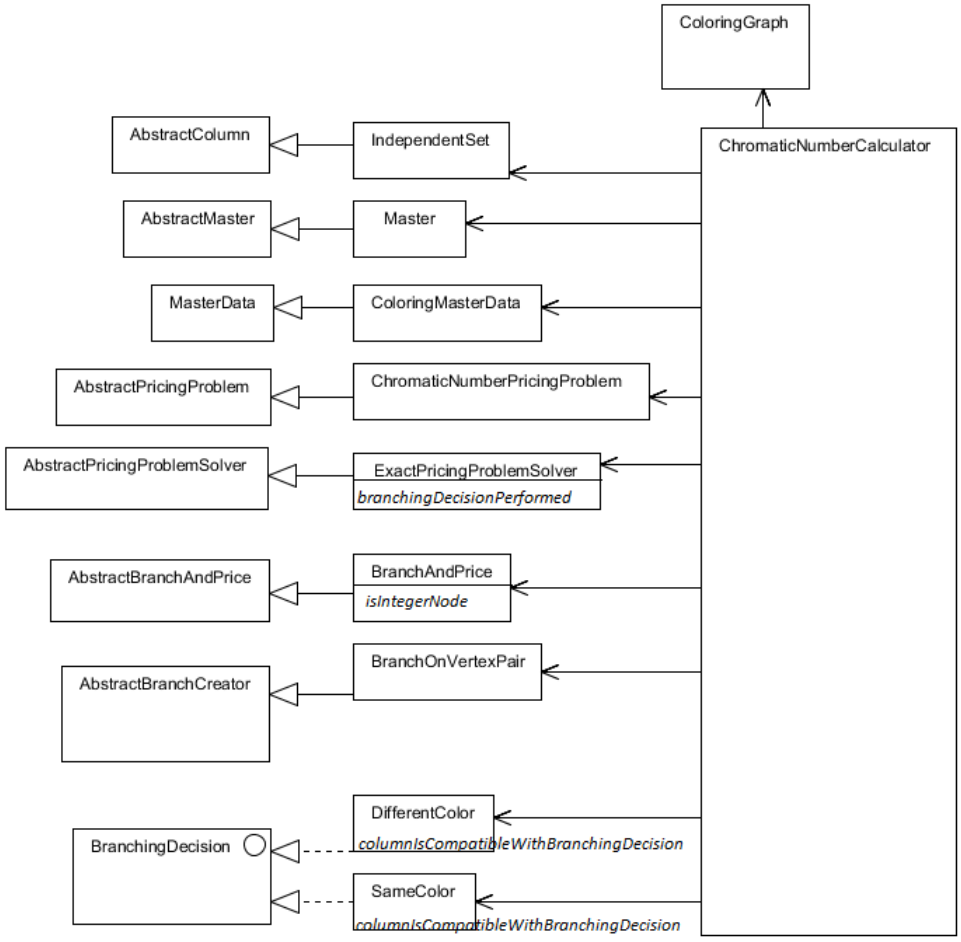

Abbildung 17: UML-Klassendiagramm für ein Graph-Colorierungsprogramm mit Spaltengenerierung und Branch-Verfahren aus dem JORLIB-Demo-Paket

Dieses und das vorherige Klassendiagramm in Abbildung 16 weisen Gemeinsamkeiten und Unterschiede auf. Das liegt daran, dass beim BaP Sohnprobleme erzeugt (branch) werden, die ebenfalls mit der Spaltengenerierung (price) gelöst werden. Das vorherige Beispiel implementiert nur eine Spaltengenerierung ohne Verzweigung. Es versagt, wenn die Lösung nicht ganzzahlig ist, obwohl sie es aufgrund der Anforderungen der realen Welt sein müsste.

6.4. UML-Klassendiagramm Traveling Salesman Problem mit Branch-and-Price

Das folgende Klassendiagramm lässt Gemeinsamkeiten aller BaP-Programme erkennen.

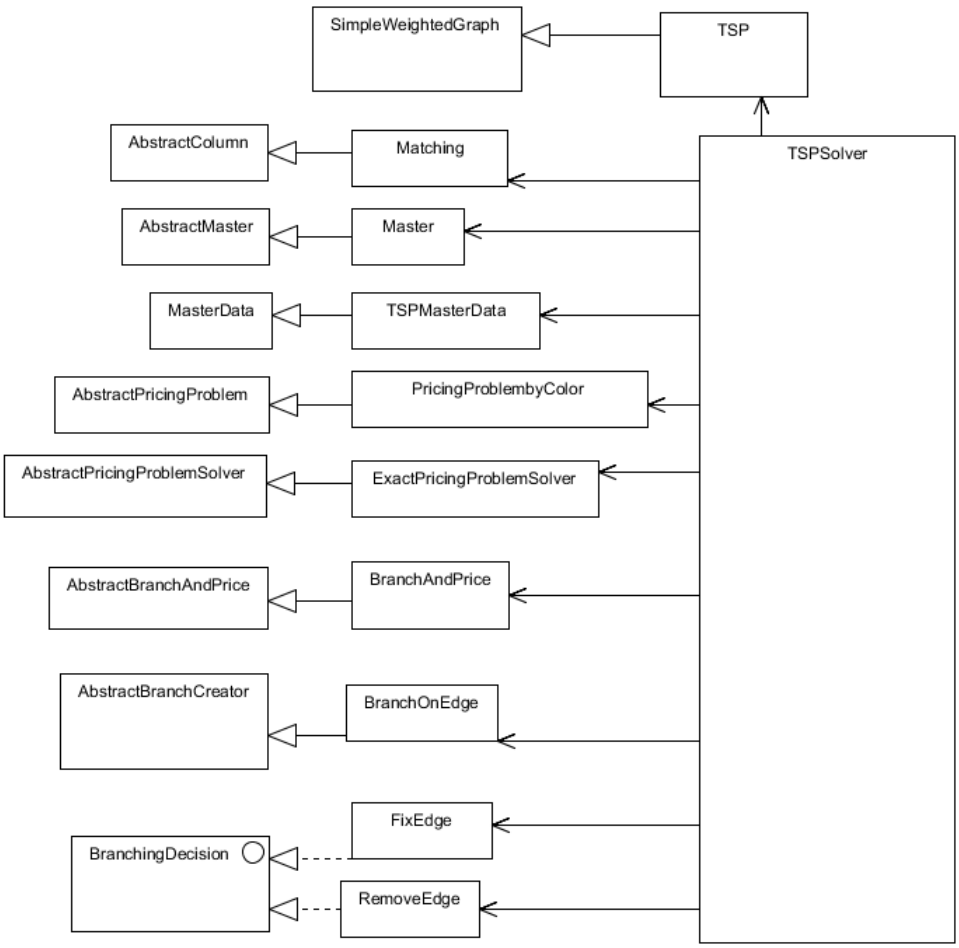

Abbildung 18: UML-Klassendiagramm für ein Traveling-Salesman-Programm mit Spaltengenerierung und Branch-Verfahren aus dem JORLIB-Demo-Paket

6.5. Allgemeine Ablaufsteuerung beim Branch-and-Price-Prozedere

Es muss eine Klasse implementiert werden, die die Basisklasse `AbstractBranchCreator<....>` erweitert. Diese heisst im Graph-Coloring-Beispiel der JORLIB-Demo `BranchOnVertexPair`.
In dieser Klasse gibt es die Methode `canPerformBranching`, in der die Verzweigungsbedingung ausprogrammiert ist und wichtige Parameter für die Verzweigung ermittelt werden. Diese Parameterwerte landen in Instanzvariablen dieser Klasse.

Desweiteren gibt es in dieser Klasse die Methode `getBranches`, die dann in Auswertung der Verzweigungsparameter die beiden Sohnknoten vom Typ `BAPNode` bereitstellt. Sie wird aufgerufen, wenn `canPerformBranching` den Wert 'true' zurückgeliefert hat.
In jedem Sohn-Knoten ist ein Objekt hinterlegt, dessen Klasse das Interface `BranchingDecision` implementiert. Es gibt zwei Söhne also auch zwei Klassen, die dieses Interface implementieren. Eine wichtige Methodenschnittstelle in diesem Interface heisst `columnIsCompatibleWithBranchingDecision`, deren Implementierung prüft, ob die bisherigen Lösungselemente, wie z. B. Schnittmustern beim Cutting-Stock-Problem, den Verzweigungsregeln des linken bzw. rechten Sohnes genügen oder nicht. Bei Nichterfüllung werden Sie durch das JORLIB-Framework automatisch gelöscht.

Eine weitere Klasse, die die Basisklasse `AbstractPricingProblemSolver` erweitert, implementiert u.a.m. die Methode `branchingDecisionPerformed`, die in Auswertung des jeweiligen Falles (linker oder rechter Sohn) die Veränderungen im Subproblem vornimmt, also z. B. zusätzliche Nebenbedingungen einfügt. Da in dieser Klasse sowieso das Subproblem gebildet wurde, ist es kein Problem, das Subproblem mit zusätzlichen Nebenbedingungen auszustatten. Die gleiche Methode kann auch in der Klasse `Master` für Änderungen im Master-Problem (RMP) implementiert werden. Die Reihenfolge ist: Zuerst Änderungen im Master-Problem, danach im Sub-Problem. Als dritten Schritt wird in der Methode `BranchandPrice.generateInitialFeasableSolution` eine Anfangslösung automatisch von JORLIB ermittelt.

6.6. Das eigene Bin-Packing-Demonstrationsbeispiel

Das Framework JORLIB soll mit einem eigenen Bin-Packing-Problem getestet und erschlossen werden. Es wurden folgende Eingabedaten gewählt:

```
public final int nrFinals=5; //Number of different finals
public final int rollWidth=5600; //Width of the raws
public final int[] finals= {1380,1520,1560,1710,1820};
public final int[] demandForFinals={1,1,1,1,1};
```

Die Ausgabedaten nach der Spaltengenerierung sehen wie folgt aus:

```
Muster=Value: 0.5 Cutting pattern: [1, 0, 0, 0, 0] creator:
initSolution
Muster=Value: 0.5 Cutting pattern: [1, 1, 1, 0, 0] creator:
 exactPricing
Muster=Value: 0.5 Cutting pattern: [0, 0, 1, 1, 1] creator:
 exactPricing
Muster=Value: 0.5 Cutting pattern: [0, 1, 0, 1, 1] creator:
exactPricing
```

Auf Grund der gebrochenen Ergebniswerte von 0,5 muss ein Branch-And-Bound-Verfahren angeschoben werden.

Dieses liefert dann das optimale Ergebnis:

```
Value: 1.0 Cutting pattern: [0, 1, 0, 1, 1] creator: exactPricing
Value: 1.0 Cutting pattern: [1, 0, 1, 0, 0] creator: exactPricing
```

Für das Branch-And-Bound-Verfahren bedarf es Verzweigungsregeln.
Es folgt ein Entwurf eines BaP-Programmes für das Bin-Packing-Problem mittels UML-Klassendiagrammen.

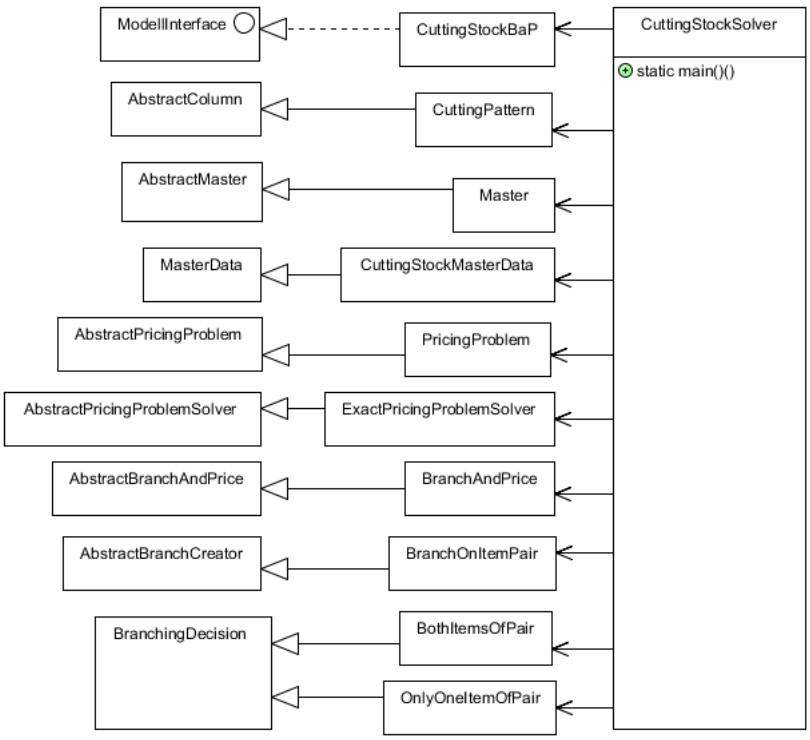

Abbildung 19: UML-Klassendiagramm für ein eigenes Bin-Packing-Programm mit BaP

Die Klasse 'CuttingStockSolver' stellt das Hauptprogramm dar. Die Klasse 'CuttingPattern' nimmt das Schnittnmuster auf. Die Klasse 'CuttingStockBaP' nimmt alle Eingabedaten und Parameter auf. Die Klassen 'BothItemsOfPair' sowie 'OnlyOneItemOfPair' implementieren die Veränderungen im Master und/oder Sub aufgrund der Verzweigungsregeln. Die Klasse 'BranchOnItemPair' kreiert als Hauptaufgabe die beiden Sohnobjekte. Die Klasse 'BranchAndPrice' prüft die Ganzzahligkeit einer Lösung und generiert für die Söhne eine Anfangslösung. In der Klasse 'ExactPricingProblemSolver' werden Spalten für den Master generiert, indem die Subprobleme (Rucksack-Probleme) gelöst werden. In der Klasse 'Master' wird der neue Master gelöst, nachdem gew. Spalten hinzugefügt wurden.

6.7. Verzweigungsregel für das Bin-Packing-Problem

Die Grundidee dafür lautet: Man wähle zwei Teile, von denen man entweder fordert, a) dass sie zusammen gepackt werden oder b) dass sie nicht zusammen gepackt werden dürfen.
Wenn die Lösung nicht ganzzahlig ist, gibt es in der Koeffizientenmatrix des Masterproblems mindestens eine dieser Teilmatrizen:

Teil	Muster1	Muster2
l	1	1
m	0	1

Tabelle 2: Indikator für gebrochene Lösung im Masterproblem

Die Teile l und m treten in Abbildung 2 sowohl gemeinsam (Muster 2) als auch einzeln (Muster 1) auf.

Die Idee der Verzweigungregel für das Bin-Packing-Problem geht auf Ryan und Foster zurück (siehe [RYAN81]). In [VANCE93] wird die Regel ausführlich hergeleitet. Sie besagt, dass im rechten Sohn nur Muster aufreten dürfen, wo die Teile l und m nicht paarweise vorkommen, sondern nur einzeln oder gar nicht und im linken Sohn nur Muster auftreten dürfen, wo beide Teile paarweise vorkommen.
Drei Aufgaben sind zu implementieren:

1. Bestimmung des Teilepaares,
2. Umsetzung der Verzweigungsregeln und
3. Erstellung einer 'artifical solution', damit der Master immer lösbar wird.

Der rechte Zweig wird dadurch implementiert, dass das Sub-IP je Paar (l,m) eine zusätzliche Nebenbedingung erhält, und zwar: $x_l + x_m \leq 1$. Der linke Zweig wird dadurch implementiert, dass die Teile l und m zu einem neuen Teil zusammengefasst werden. Dadurch ändert sich nicht nur das Sub- sondern auch das Master-Problem.
Nach [GRUIRN05] S.171 kann diese Branching-Regel auch für andere Partitionierungs- und Packungsprobleme verwendet werden.

6.8. Quelltext zur Bestimmung des Teilepaares nach Ryan/-Foster in der Klasse 'BranchOnItemPair'

Der folgende Java-Quelltext zeigt, wo und wie man das Teilepaar nach der Verzweigungsregel von Ryan und Foster implementiert. Die folgende Methode stammt aus der Klasse 'BranchOnItemPair'.

Listing 1: Bestimmung des Teilepaares

```
protected boolean canPerformBranching(List<CuttingPattern>
solution) { CuttingPattern p=null;
 //Bestimmung der Masterkoeffizienten
 int master[][]=
 new int[p.yieldVector.length][solution.size()];
 for(int i=0; i<solution.size(); i++)
             for (int z=0;z<master.length;z++)
             {      p=solution.get(i);
                  master[z][i]=p.yieldVector[z];
                              };
//pruefen, ob Quadrat nach Vanze/Ryan/Foster
// existiert
  boolean erfolg=false;
 int l=-1; int m=-1; int ks=-1; int k2s=-1;
 for (int z1=0;z1<master.length && !erfolg;z1++)
 //traversiere alle Zeilen
 for (int s1=0;s1<master[0].length && !erfolg;s1++)
 //traversiere alle Spalten
 if (master[z1][s1]==1) //suche 1. Eins
    for (int s2=s1+1;s2<master[0].length&& !erfolg;s2++)
//suche 2. Eins
   if (master[z1][s2]==1) {l=z1;ks=s1;k2s=s2;

for (int z2=z1+1;z2<master.length && !erfolg;z2++)
//traversiere Zeilen  nach z1
 if (master[z2][ks]==0 && master[z2][k2s]==1)
   {l=z1;m=z2; erfolg=true;}
// in zeile m : 1 und 0 abfragen fehlt, zur zeit wird nur
//  0 und 1 abgefragt
//traversieren Zeilen vor z1 fehlt auch noch
 };

 k1st=ks;k2st=k2s;
 el=l;em=m;
 if (l>=0 && m>=0) {
candidateItemPair=new ItemPair<>(l,m);
  return true;}
 else return false;
 }
```

6.9. Veränderung des Sub-LP im Fall 'Single' nach Ryan/-Foster

Vor dem ersten Branch sieht das Sub-LP beispielsweise wie folgt aus:

```
//Sub vor dem Verzweigen
Maximize
 obj: x1 + x2 + x3 + x4 + x5
Subject To
 c1: 1380 x1 + 1520 x2 + 1560 x3 + 1710 x4 + 1820 x5 <= 5600
Bounds
 0 <= x1 <= 1
 0 <= x2 <= 1
 0 <= x3 <= 1
 0 <= x4 <= 1
 0 <= x5 <= 1
Generals
 x1  x2  x3  x4  x5
End
```

Liefert die Spaltengenerierung eine nicht ganzzahlige Lösung, muss verzweigt werden. Der folgende Quelltext erzeugt das neue Sub-LP für den rechten Sohn, der erzwingt, dass z. B. die Teile 1 und 2 nur einzeln in einer Spalte (Muster) auftreten.

Listing 2: Gestaltung des neuen Sub-Programms für den Zweig 'single'

```
1  //Klasse ExactPricingProblemSolver
2  public void branchingDecisionPerformed(BranchingDecision bd
3  ){ if(bd instanceof BothItemsOfPair){
4  .....          }
5  else if(bd instanceof OnlyOneItemOfPair){
6  //krit. Teile duerfen n. beide im Schnittmuster auftauchen
7
8  OnlyOneItemOfPair differentItemDecision=
9   (OnlyOneItemOfPair) bd;
10  try{
11  IloConstraint branchingConstraint=cplex.addLe(
12        cplex.sum(vars[differentItemDecision.itemPair.
13  getFirst()],
14  vars[differentColorDecision.vertexPair.getSecond()]), 1);
15   branchingConstraints.put(differentItemDecision.itemPair,
16                                          branchingConstraint);
17                    }
18  catch (IloException e) {
19  e.printStackTrace();
20  }}}
```

Das Sub-LP wurde durch diesen Quelltext um die Nebenbedingung c2 ergänzt und sieht dann wie folgt aus.

```
//Sub nach dem Verzweigen im Fall 'Single'
Maximize
 obj: x1 + x2 + x3 + x4 + x5
Subject To
 c1: 1380 x1 + 1520 x2 + 1560 x3 + 1710 x4 + 1820 x5 <= 5600
 c2: x1 + x2 <= 1
Bounds
 0 <= x1 <= 1
 0 <= x2 <= 1
 0 <= x3 <= 1
 0 <= x4 <= 1
 0 <= x5 <= 1
Generals
 x1  x2  x3  x4  x5
End
```

6.10. Veränderung des Master- und Sub-LPs im Fall 'Pair' nach Ryan/Foster

Vor dem ersten Branch sieht das Sub-LP beispielsweise wie folgt aus:

```
//Sub-LP vor dem Zusammenlegen der ersten beiden Teile
Maximize
 obj: x1 + x2 - x3 + x4 + x5
Subject To
 c1: 1380 x1 + 1520 x2 + 1560 x3 + 1710 x4 + 1820 x5 <= 5600
Bounds
 0 <= x1 <= 1
 0 <= x2 <= 1
 0 <= x3 <= 1
 0 <= x4 <= 1
 0 <= x5 <= 1
Generals
 x1  x2  x3  x4  x5
End
```

Liefert die Spaltengenerierung eine nicht ganzzahlige Lösung, muss verzweigt werden. Der folgende Quelltext erzeugt das neue Master-LP für den linken Sohn, der erzwingt, dass z. B. die Teile 1 und 2 nur paarweise in einer Spalte (Muster) auftreten.

Listing 3: Gestaltung des neuen Master-Programms für den Zweig 'pair'

```
1 //Klasse Master
2 public void branchingDecisionPerformed(BranchingDecision bd
3 ) {
4   ....
```

```
5  if(bd instanceof BothItemsOfPair){
6
7  //Aenderungen nach Ryan Foster
8  int nrfin=this.dataModel.nrFinals;
9  int finals[]=this.dataModel.finals;
10 int demandFin[]= new int[nrfin-1];  //ein teil weniger
11 int bingroesse=this.dataModel.rollWidth;
12
13 this.dataModel.demandForFinals= null;
14 CuttingStockBaP  dm = new CuttingStockBaP();
15 dm.nrFinals=nrfin-1;
16 dm.finals= new int[dm.nrFinals];
17  //2  Teile werden zusammengefasst
18 dm.demandForFinals= new int[nrfin];
19
20  //beide teile eines Paares zusammenfassen
21 BothItemsOfPair sameColorDecision = (BothItemsOfPair) bd;
22 int t1=sameColorDecision.vertexPair.getFirst();
23 int t2=sameColorDecision.vertexPair.getSecond();
24 int index=0;
25 for (int lauf=0;lauf<nrfin;lauf++)
26 {if (lauf!=t1 && lauf !=t2)
27 { dm.finals[index]=finals[lauf];
28 index++;
29 }  }
30 dm.finals[index]=finals[t1]+finals[t2];
31 //neues Teil  mit Groesse von Summe zweier Teile t1 und t2
32 //setze neuen Demandvektor
33 for (int lauf=0;lauf<demandFin.length;lauf++)
34 dm.demandForFinals[lauf]=1;
35  //Setzen der neuen  dataModel
36 this.dataModel.nrFinals=dm.nrFinals;
37 this.dataModel.finals=dm.finals;
38 this.dataModel.demandForFinals=dm.demandForFinals;
39  //erzeugen des neuen Masters
40 buildModel();
```

Die Veränderung im Master-LP hat auch Auswirkungen auf das Sub-LP, welches dann wie folgt aussieht.

```
 //Sub-LP nach dem Zusammenlegen der ersten beiden Teile
Maximize
 obj: x1 + x2 + x3 + x4
Subject To
 c1: 1560 x1 + 1710 x2 + 1820 x3 + 2900 x4 <= 5600
Bounds
```

```
 0 <= x1 <= 1
 0 <= x2 <= 1
 0 <= x3 <= 1
 0 <= x4 <= 1
Generals
 x1  x2  x3  x4
End
```

Man beachte das neue Teil 4, welches durch Zusammenlegen der ehemaligen Teile 1 und 2 entstanden ist.

6.11. Änderung der Bin-Größe

Gegenüber dem obigen Demonstrationsbeispiel wird der Wert 'rollwidth' geringfügig auf 5040 verringert. Die Teile 2, 4 und 5 passen dann nicht mehr in ein Bin, d. h.f die obige Lösung gilt dann nicht mehr.
Die Spaltengenerierung liefert wieder kein ganzzahliges Ergebnis sondern:

```
//width=5040
Muster=Value: 0.5 Cutting pattern: [0, 0, 0, 0, 1] creator:
 initSolution
Muster=Value: 0.5 Cutting pattern: [1, 0, 1, 1, 0] creator:
 exactPricing
Muster=Value: 0.5 Cutting pattern: [1, 1, 0, 1, 0] creator:
 exactPricing
Muster=Value: 0.5 Cutting pattern: [0, 1, 1, 0, 1] creator:
exactPricing
```

Aber das Branch-and-Price-Verfahren ergibt eine ganzzahlige Lösung:

```
================ Loesung ================
CS-BAP beendet with objective number of raws): 2
Total Number of iterations: 10
Total Number of processed nodes: 3
Total Time spent on master problems:
0 Total time spent on pricing problems: 204
Solution is optimal: true
Columns (only non-zero columns are returned):
Value: 1.0 Cutting pattern: [0, 0, 1, 1] creator: exactPricing
Value: 1.0 Cutting pattern: [1, 1, 0, 0] creator: exactPricing
BUILD SUCCESSFUL (total time: 0 seconds)
```

Es verwundert, dass die Muster jetzt nur 4 Elemente haben. Das hat folgenden Grund.
Durch die Verzweigungsregel für Paare wurden die Teile 1 und 2 zusammengefasst und stellen nun den 4. Wert dar. Das Muster [1,1,0,0] bedeutet dann die Teile mit den Längen 1560 und 1710 (ehemals Teile Nr. 3 und 4) und das Muster [0,

0, 1, 1] die Teile mit den Längen 1820 (ehemals Teil Nr.5) sowie 1380 und 1520 (ehemals Teile Nr. 1 und 2). Die 4. Zahl steht also hier immer für zwei Teilearten. Als allgemeine Initiallösung wurde eine Next-fit-Strategie gewählt, bei der das Bin solange mit Teilen gefüllt wird, bis es voll ist. Als allgemeine künstliche Lösung (engl. artifical solution) wurden n Einheitsvektoren gewählt. Mittels der künstlichen Lösung wird garantiert, dass die Master-Probleme der Söhne stets lösbar sind.

7. Verbale Beschreibung der erstellten Software für das Bin-Packing-Problem

Klasse BranchAndPrice:

In der Methode 'generateInitialFeasibleSolution' wird eine künstliche Lösung erzeugt, mit der jeder Sohn-Master lösbar ist. Ihr werden hohe Kosten zugewiesen, damit sie nicht im Endergebnis auftaucht. Der Autor wählte die Einheitsvektoren hierfür.

In der Methode 'isIntegerNode' muss programmiert werden, wann eine Lösung nicht ganzzahlig ist. Zu beachten ist, dass es nicht reicht, den Zielfunktionswert zu überprüfen, weil der ganzzahlig sein kann, obwohl einzelne Schnittmuster gebrochen bewertet wurden.

Klasse BranchOnItemPair:

In der Methode 'canPerformBranching' wird die Verzweigungsbedingung, bei Bin-Packing nach Vance (siehe [VANCE93]), geprüft und Parameter ermittelt, hier z. B. die Indizes l und m der Teile, die entweder alleine (rechter Zweig) oder paarweise (linker Zweig) auftreten müssen. Die ermittelten Parameter werden in Klassenvariablen abgelegt, damit Methoden anderer Klassen darauf zugreifen können.

In der Methode 'getBranches' werden die beiden Verzweigungsobjekte vom Typ 'BothItemsOfPair' sowie 'OnlyOneItemOfPair' angelegt und zurückgeliefert.

Klasse BothItemsOfPair:

In der Methode 'columnIsCompatibleWithBranchingDecision' wird entschieden, ob die Lösungsmuster des Vater-Masters für den Sohn-Master kompatibel sind. Zusätzliche Instanzvariablen nehmen die Parameter der Verzweigung-Regel auf.

Klasse OnlyOneItemOfPair:

In der Methode 'columnIsCompatibleWithBranchingDecision' wird entschieden, ob ein Vater-Muster für den rechten Zweig (Single) kompatibel ist für den Sohn. Zusätzliche Instanzvariablen nehmen die Parameter der Verzweigungsregel auf.

Klasse CuttingPattern:

Instanzvariablen dieser Klasse nehmen das Schnittmuster und deren Kosten auf.

Klasse CuttingStockMasterData:

Eine Instanzvariable nimmt eine CPLEX-Instanz auf.

Klasse ExactPricingProblemSolver:

In der Methode 'buildModel' wird das Sub-LP für den Solver (CPLEX) gebildet.

In der Methode 'generateNewColumns' wird durch das Lösen eines Subproblems evt. eine neue Spalte für das Master-LP generiert, wenn es negative Kosten erzeugt.

In der Methode 'branchingDecisionReversed' werden Änderungen im Subproblem zurückgenommen, falls Backtracking erforderlich ist.

In der Methode 'branchingDecisionPerformed' werden die beiden Sub-Lps für den linken und rechten Zweig erzeugt. Im Falle 'Single' muss eine neue Nebenbedingung eingefügt werden, die garantiert, dass nur ein Teil des Paares l und m vorhanden ist.

Klasse Master:

In der Methode 'buildModel' wird das Master-LP gebildet.

Die Methode 'solveMasterProblem' löst das Master-Problem.

Die Methode 'addColumn' fügt dem Master-LP eine neue Spalte hinzu, entweder eine vom Vater oder eine, die vom Sub-LP erzeugt wurde.

Die Methode 'getSolution' liefert die Lösung des Master-LPs.

Die Methode 'branchingDecisionPerformed' staucht das Problem im linken Zweig (Pair) und bildet den Master für beide Zweige neu, wenn Spalten hinzugekommen sind.

Die Klasse CuttingStockBaP:

Diese Klasse beinhaltet die Eingabedaten.

Die Klasse CSBaPSolver:

Diese Klasse beinhaltet das Hauptprogramm, welches den BaP-Prozess anstösst sowie die Anfangslösung für das Master-LP sowie den unteren und oberen Grenzwert ermittelt. Die Anfangslösung wird nach der Strategie, 'Füllen des Bins bis es keinen Platz mehr hat für das nächste Teil' erstellt.

8. Rechenzeiten von SCIP/CPLEX für das Bin-Packing-Problem

Im SCIP-Paket wird eine lauffähige Lösung für das Bin-Packing-Problem auf Basis des Ryan-Foster-Branch mitgeliefert. Die folgende Tabelle veranschaulicht die Rechenzeiten ausgewählter Beispiele aus dem SCIP-Paket.

Anzahl Teile	Anzahl Bin	Rechenzeiten(Millisek.)
40	17	1090
80	33	15600
250	99	93650
500	200	1813000 (GAP=1,22%)
1000	404	1807000 (GAP=1,4 %)

Abbildung 20: SCIP-Rechenzeiten für das BaP-BinPacking-Programm für verschiedene Anzahlen von Teilen und einer Bin-Größe von 150 auf Lenovo G580

Die genaue Spezifikation des Rechners Lenovo G580 lautet: Intel(R) Core(TM) i5-3320M CPU, 2.6 GHZ, 2 Kerne, 4 logische Prozessoren, Windows 7 Enterprise, Systemmodell 2429AT6, x64-basierter PC.

Dass sich der Aufwand für BaP lohnt, zeigt die Rechenzeit von 8.000.000 Millisekunden bis zu einem GAP von 1,45% im Fall der 250 Teile, wenn man die klassische Variante der Problemstellung, also ohne Spaltengenerierung, wählt bei gleichem Solver.

9. Rechenzeiten(RZ) der eigenen JORLIB-basierten Implementierung für das Bin-Packing-Problem

Wenn man ohne einen schnellen Sub-Solver für das erweiterte Rucksack-Problem, also auch mit einem LP, im JORLIB-Paket arbeitet, sind die Rechenzeiten mit JORLIB relativ zu SCIP sehr hoch.

Anzahl Teile	Bin-Größe	RZ- Master	RZ- Sub	Anzahl Bins
20	12000	221	72224	8
	9000	295	78500	6
	7000	140	23754	4
	5000	266	31058	3
30	5000	720	201437	12

Tabelle 3: JORLIB-Rechenzeiten für das BaP-BinPacking-Programm für verschiedene Anzahlen von Teilen auf Lenovo G580, Prozessor i5-3210M, 2,5 GHZ Taktfrequenz, 2 Kerne, 4 logische Prozessoren

Der Vollständigkeit halber folgen die Größen der Einzelteile:

```
n=20
{1380,1520,1560,1710,1820,1880,1930,2000,2050
,2100,2140,2150,2200,1630,1680,1450,1470,
1510,1320,1380};
*********************************************
n=30
{1380,1520,1560,1710,1820,1880,1930,2000,2050,
2100,2140,2150,2200,1630,1680,1450,1470,
1510,1320,1380,500,520,600,630,700,740,800,
850,900,960};
```

Beim Vergleich der SCIP- und JORLIB-Rechenzeiten fällt auf, dass die JORLIB-Rechenzeiten viel höher sind als die SCIP-Rechenzeiten bei vergleichbar vielen Einzelteilen. Auch die CPLEX-Zeiten ohne BaP sind viel besser als mit BaP. Bei einem Beispiel mit 32 Teilen benötigte CPLEX 0,19 Sek. und JORLIB 90 Sekunden. Die Ursache liegt im langsamen Sub-Solver(CPLEX). Folgende Tabelle zeigt, dass zumindest an zwei Beispielen die Geschwindigkeit der SCIP-Lösung auch mit JORLIB erreicht werden kann, wenn man den JORLIB-Knapsack-Solver für die Sub-Programme verwendet.

Anzahl Teile	RZ(Millisek.)	RZ(Millisek.) Master	RZ(Millisek.) Sub
40	2868	640	2228
80	8177	2928	5249

Tabelle 4: JORLIB-Rechenzeiten für das BaP-BinPacking-Programm bei Verwendung eines schnellen Knapsack-Solvers auf einem Lenovo T530 Notebook, i5-3210M CPU, 2,5 GHZ Taktfrequenz, 2 Kerne, 4 logische Prozessoren

10. Ein- und Ausgabedaten für ein Bin-Packing Beispiel mit großer Einsparung vs. der Next-Fit-Strategie

Bei diesem Beispiel spart die Optimierung 37,5 % gegenüber der Next-Fit-Strategie ein. Next-Fit benötigt 8 Bins und im Optimum werden nur 5 Bins benötigt. Bei der Next-Fit-Strategie werden die Teile in der Reihenfolge ihres Eintreffens in das Bin gelegt, solange es noch hinein passt. Also bei den Testdaten unten zuerst ein Teil der Länge 4, dann ein Teil der Länge 1. Nun passt das nächste Teil der Größe 4 nicht mehr in das erste Bin der Größe 8, und man muss ein zweites Bin verwenden usw. Das Beispiel ist so konstruiert, dass man leicht sieht, dass 5 Bins ausreichen.

```
Bin-Packing mit BaP

Eingabedaten:
************
Anzahl Teilearten=16
Laenge Basismaterial =8
Bedarfe:
,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
Groessen:
,4,1,4,1,4,1,4,1,4,1,4,1,4,1,4,1

Ausgabedaten:
************
Pricing Problem instantiiert
Master instantiiert
Solver instantiiert
Kontrolldruck initial Solution,size=8
Musterkontrolle vor Abspeicherung
,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0**
Musterkontrolle vor Abspeicherung
,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0**
Musterkontrolle vor Abspeicherung
,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0,0**
Musterkontrolle vor Abspeicherung
,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0**
Musterkontrolle vor Abspeicherung
,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0**
Musterkontrolle vor Abspeicherung
,0,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0**
Musterkontrolle vor Abspeicherung
,0,0,0,0,0,0,0,0,0,0,0,0,1,1,0,0**
Musterkontrolle vor Abspeicherung
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1**
```

Ergebnis der Spaltengenerierung ohne BaP:

```
Trace: BranchAndPrice.isIntegerNode=false
16:07:15.296 [main] DEBUG o.j.f.c.io.SimpleDebugger - Node 0 is fractional.
Objective: 4.999999999999999, bound: 5.0
Trace: BranchOnItemPair.canPerformingBranching
Kontrolldruck aller Muster
Value: 0.2857142857142857 Cutting pattern:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
creator: exactPricing
Value: 0.5714285714285714 Cutting pattern:
 [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1]
 creator: exactPricing
Value: 0.05714285714285718 Cutting pattern:
 [0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1]
 creator: exactPricing
Value: 0.11428571428571438 Cutting pattern:
 [0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1]
 creator: exactPricing
Value: 0.8285714285714286 Cutting pattern:
 [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
 creator: exactPricing
Value: 0.17142857142857137 Cutting pattern:
 [0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]
 creator: exactPricing
Value: 0.8857142857142857 Cutting pattern:
 [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
 creator: exactPricing
Value: 0.3714285714285715 Cutting pattern:
 [0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0]
 creator: exactPricing
Value: 0.19999999999999993 Cutting pattern:
 [0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0]
creator: exactPricing
Value: 0.7142857142857143 Cutting pattern:
 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
creator: exactPricing
Value: 0.22857142857142868 Cutting pattern:
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0]
 creator: exactPricing
Value: 0.17142857142857137 Cutting pattern:
 [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1]
creator: exactPricing
Value: 0.05714285714285718 Cutting pattern:
 [0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1]
creator: exactPricing
```

```
Value: 0.2857142857142857 Cutting pattern:
 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
creator: exactPricing
Value: 0.028571428571428557 Cutting pattern:
 [0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1]
 creator: exactPricing
Value: 0.028571428571428636 Cutting pattern:
 [0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0]
 creator: exactPricing

Ergebnis von Branch-and-Price:

================ Loesung ================
Bin-Packing-BAP beendet with objective number of raws): 5
Total Number of iterations: 45
Total Number of processed nodes: 9
Total Time spent on master problems:
46 Total time spent on pricing problems: 2920
Solution is optimal: true
Columns (only non-zero columns are returned):
Value: 1.0 Cutting pattern:
 [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0] creator: exactPricing
Value: 1.0 Cutting pattern:
 [0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0] creator: exactPricing
Value: 1.0 Cutting pattern:
 [0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0] creator: exactPricing
Value: 1.0 Cutting pattern:
 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0] creator: exactPricing
Value: 1.0 Cutting pattern:
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1] creator: exactPricing
BUILD SUCCESSFUL (total time: 3 seconds)
```

11. FAQs beim Bin-Packing Beispiel

1. Wo wurde das Stauchen eines veträglichen Musters (des Vaters) für den Paar-Zweig programmiert?
Antwort In der Klasse 'Master', und zwar in der Methode 'branchingDecisionPerformed'.

2. Wo wird die künstliche Lösung ('artifical solution') programmiert und wozu dient sie?
Antwort In der Klasse 'BranchAndPrice', die die Klasse 'AbstractBranchAndPrice' erweitert. Sie soll garantieren, dass alle Sohn-Master lösbar sind.

3. Wo werden die Parameter für die Verzweigungsregel ermittelt?
Antwort In der Klasse 'BranchOnItemPair', die die Klasse 'AbstractBranchCreator' erweitert, und zwar in der Methode 'canPerformBranching'.

4. Wie werden die in 3. ermittelten Parameter weitergereicht?
Antwort In der Klasse 'BranchOnItemPair' werden auch die beiden Söhne bereitgestellt. Beide erhalten die Paramter.

5. Was passiert, wenn die künstliche Lösung zufällig schon optimal ist?
Antwort Dann funktioniert JORLIB nicht richtig.

6. Wo wird die Regel für den Single-Fall umgesetzt?
Antwort In der Klasse 'ExactPricingProblemSolver', und zwar in der Methode 'branchingDecisionPerformed'.

7. Müssen die Muster der künstlichen Lösung verträglich mit der Verzweigungsregel 'Single' sein?
Antwort Ja, weshalb Einheitsvektoren gewählt wurden.

8. Müssen die Muster der artifical solution verträglich mit der Verzweigungsregel 'Pair' sein?
Antwort Sind sie automatisch, wegen der Stauchung.

9. Wo wird die Stauchung des verträglichen Exact-Pricing-Musters des Vaters für den Sohn programmiert?
Antwort In der Methode 'addColumn' der Klasse 'Master'.

12. Resultatsermittlung und -bewertung

Ausgangspunkt dieser Arbeit waren Laufzeittests mit der Software 'SCIP' für das VRP.
Es konnte die alternative Software 'JORLIB' zu SCIP erschlossen und unter der IDE 'Netbeans 8.2' die Einsatzfähigkeit erreicht werden.
Die Software 'JORLIB' besteht aus ca. 500 Klassen und die Software 'SCIP' aus 270.000 Quelltextzeilen, was die Komplexität des Themas unter Beweis stellt.
Die Software 'JORLIB' ist wesentlich besser dokumentiert als die Software 'SCIP'.
Es konnten didaktische Beispiele für das Spaltengenerierungsverfahren sowohl zum VRP als auch zum Cutting-Stock-Problem geschaffen werden (ohne Implementierung).
Im Vergleich der Softwaren 'SCIP' und 'JORLIB' empfiehlt der Autor die kostenlose Software 'JORLIB', und zwar in der durch den Autor modifizierten Struktur für die IDE Netbeans 8.2 . Für deren Nutzung sind exzellente Java-Kenntnisse notwendig.
Kritisch anzumerken ist bei der Software 'JORLIB', dass dessen Autor ein programmierender Mathematiker zu sein scheint, der die Regeln der Softwaretechnik nicht immer korrekt anwendet. Zum Beispiel ist es nicht üblich, dass die gesamte Funktionalität einer Klasse im Konstruktor aufgerufen wird, wie in der Demo-Klasse 'CuttingStockSolver' im mitgelieferten Demo-Paket geschehen.
Für die weitere Nutzung der JORLIB-Software ist eine Auseinandersetzung mit Verzweigungsregeln für unterschiedliche Problemtypen unumgänglich, da die Software dafür keine Unterstützung gibt.
Interessant ist auch, wie sich die Rechenzeit beim Branch-and-Price-Verfahren auf Master- und Pricing-Problem aufteilt. Beim mitgelieferten BaP-Beispiel der Colorierung eines Graphen betrug das Verhältnis 1:50.
JORLIB hält nicht, was es im Manual verspricht. Es besitzt nämlich keinen eigenen allgemeinen Rucksack-Solver. Rucksack-Subprobleme werden im Demo-Projekt auch mit dem CPLEX-Programm gelöst.
Es ist allerdings schon gelungen, in einem Spezialfall (32 Teile) die Rechenzeit eines Bin-Packing-Beispiels von 1 Minute, 29 Sekunden auf 6 Sekunden zu reduzieren, indem für den Zweig 'Pair', das Sub-Problem ist ein einfaches Knapsack-Problem ohne zusätzliche Nebenbedingungen wie beim Zweig 'Single', der CPLEX-Solver durch einen JORLIB-Binary-Knapsack-Solver ersetzt wurde.
Weitere Beispiele mit bis zu 80 Teilen ergaben eine um den Divisor 32 kleinere Rechenzeit bei der Verwendung des Knapsack-Solvers gegenüber dem CPLEX-Solver (statt 13 Minuten nur noch 25 Sekunden).
Die Fehlersuche in dem sehr grossen JORLIB-Framework (ca. 500 Klassen) erwiess sich als recht schwierig. Dieses Framework wird lediglich anhand von drei kommentierten Demonstrationsbeispielen (Cutting Stock, Graph Coloring und Traveling Salesman) erläutert. Weitere Beschreibungen und Dokumentationen gibt es nicht. Das macht die Fehlersuche schwierig, wenn insbesondere bei eigenen Beispielen Methoden vom Framework aufgerufen werden, für die es keine vergleichbaren Methodenaufrufe in den Demonstrationsbeispielen gibt.
Ursprünglich stand das VRP als eigenes Demo-Projekt im Mittelpunkt der Arbeit, aber wegen der Komplexität der Software wurde zuerst das Cutting-Stock-

Problem und als Spezialfall das Bin-Packing-Problem als eigenes Problem ausgewählt, um es mit JORLIB zu lösen, was auch gelang.
Ein Ziel der Arbeit, eine Arbeitsumgebung für das Brand-and-Price-Verfahren der Optimierung bereitzustellen und die Eignung der Arbeitsumgebung anhand eines eigenen Beispiels ('Bin-Packing) nachzuweisen, wurde erreicht. Zur Auswahl standen die Frameworks 'SCIP' und 'JORLIB'. Die Empfehlung des Autors lautet: **JORLIB**, weil es besser dokumentiert ist und in einem Tutorium gut erklärt wird. Es bleiben allerdings auch bei JORLIB eine Reihe von Fragen offen, z.B. welche Backtracking-Strategie implementiert wurde? Aufgrund der Komplexität von JORLIB eignet sich auch dieses Paket nur für das Master-Studium, z. B. der Wirtschaftsinformatik oder Digitalen Logistik.
Das zweite Ziel, eine BaP-Lösung für das VRPTW, konnte leider nicht erreicht werden, weil die Komplexität der Materie schon beim Einführungsbeispiel 'Bin-Packing' viel höher war als erwartet.

Literatur

[Clark1964] Clarke, G., Wright, J.W., Scheduling vehicles from a central delivery depot to a number of delivery points, ORQ 12 (1964), S. 568-581.

[Des2010] Guy Desaulniers, Branch-and-price-and-cut for the split delivery vehicle routing problem with time windows, Operations Research 58(1), 179-192 (2010).

[Epel2004] C. Lee, M. Epelman, C. White, Y. Bozer, A shortest path approach to the multiple-vehicle routing problem with split pick-ups, Transportation Research Part B: Methodological 40(4): 265-284, 2006.

[Feillet2010] Dominique Feillet, A tutorial on column generation and branch-and-price for vehicle routing problems, Operations Research 2010, S.407 bis 424.

[GRUIRN05] Grünert T., Irnich S.: Optimierung im Transport, Band 1, Shaker 2005.

[JORLIB2015] Java Operations Research Library, `https://github.com/coin-or/jorlib/releases`.

[LUTZ98] Lutz, Michael: Operations Research Verfahren, Fortis Verlag FH, 1994.

[MuRo2006] Harald Mumm, Hans Röck, Developing operation and dicision support tools for a split-delivery vehicle routing application domain, Business Informatics Research BIR 2006, Kaunas (Lithuania).

[Mu2011] Harald Mumm, Benchmark zur Tourenoptimierung, Wismarer Diskussionspapiere, Heft 7/ 2011 .

[Mu2012] Harald Mumm, Optimale Lösungen von Tourenoptimierungsproblemen mit geteilter Belieferung, Zeitfenstern, Servicezeiten und vier LKW-Typen, Wismarer Diskussionspapiere, Heft 8/ 2012 .

[Mu2017] Harald Mumm, Atlas optimaler Touren, Wismarer Diskussionspapiere, Heft 3/ 2017 .

[Mu2018] Harald Mumm, Logistika 1.0, Android App, Tourenplanung als Strategiespiel, Google Play Store 2018 .

[RYAN81] D. M. Ryan and B. A. Foster: An Integer Programming Approach to Scheduling, In Computer scheduling of public transport: Urban passenger vehicle and crew scheduling, A. Wren editor, North-Holland 1981, 269-280.

[SCIP] Solving Constraint Integer Programs , `http://scip.zib.de`, Konrad-Zuse-Institut Berlin.

[VANCE93] Vance, Pamela H. u.a.m.: Solving Binary Cutting Stock Problems by Column Generation and Branch-and-Bound, Computational Optimization and Applications,3, 111-130 (1994).

Autorenangaben
Prof. Dr. rer. nat. Harald Mumm
Fachbereich Wirtschaft
Hochschule Wismar
Philipp-Müller-Straße 14
Postfach 12 10
D-23966 Wismar
Telefon: ++49 / (0)3841 / 753 450
Fax: ++49 / (0)3841 / 753 131
E-mail: harald.mumm@wi.hs-wismar.de

WDP - Wismarer Diskussionspapiere / Wismar Discussion Papers

Heft 04/2013: Frederik Schirdewahn: Analyse der Effizienz einzelner Maßnahmen zur Reduzierung des CO2 -Ausstoßes in der Transportlogistik

Heft 05/2013: Hans-Eggert Reimers: Remarks on the euro crisis

Heft 01/2014: Antje Bernier (Hrsg.): Na, altes Haus? – Stadt und Umland im Wandel. Planungs- und Entwicklungsinstrumente mit demografischer Chance, Konferenz der Hochschule Wismar am 14. Okt. 2013 in Schwerin

Heft 02/2014: Stefan Voll/Daniel Alt: „Das große Ziel immer im Auge behalten" Sportimmanente Indikatoren des Trainerstils von Jürgen Klopp – Transfermöglichkeiten für Führungskräfte in Genossenschaftsbanken

Heft 03/2014: Günther Ringle: Genossenschaftliche Solidarität auf dem Prüfstand

Heft 04/2014: Barbara Bojack: Alkoholmissbrauch, Alkoholabhängigkeit

Heft 01/2015: Dieter Gerdesmeier/ Hans-Eggert Reimers/ Barbara Roffia: Consumer and asset prices: some recent evidence

Heft 02/2015: Katrin Schmallowsky: Unternehmensbewertung mit Monte-Carlo-Simulationen

Heft 03/2015: Jan Bublitz/ Uwe Lämmel: Semantische Wiki und TopicMap-Visualisierung

Heft 04/2015: Herbert Müller: Der II. Hauptsatz der Thermodynamik, die Philosophie und die gesellschaftliche Praxis – eine Neubetrachtung

Heft 05/2015: Friederike Diaby-Pentzlin: Auslandsinvestitions-

recht und Entwicklungspolitik: Derzeitiges bloßes internationales Investitionsschutzrecht vertieft Armut

Heft 02/2016: Günther Ringle: Die soziale Funktion von Genossenschaften im Wandel

Heft 01/2017: Benjamin Reimers: Momentumeffekt: Eine empirische Analyse der DAXsector Indizes des deutschen Prime Standards

Heft 02/2017: Florian Knebel, Uwe Lämmel: Einsatz von Wiki-Systemen im Wissensmanagement

Heft 03/2017: Harald Mumm: Atlas optimaler Touren

Heft 01/2018: Günther Ringle: Verfremdung der Genossenschaften im Nationalsozialismus – Gemeinnutzvorrang und Führerprinzip

Heft 02/2018: Sonderheft: Jürgen Cleve, Erhard Alde, Matthias Wißotzki (Hrsg.) WIWITA 2018. 11. Wismarer Wirtschaftsinformatiktage 7. Juni 2018. Proceedings

Heft 03/2018: Andreas Kneule: Betriebswirtschaftliche Einsatzmöglichkeiten von Cognitive Computing

Heft 04/2018: Claudia Walden-Bergmann: Nutzen und Nutzung von E-Learning-Angeboten im Präsenzstudium Analyse von Daten des Moduls Investition

Heft 05/2018: Sonderheft: Katrin Schmallowsky, Christian Feuerhake, Empirische Studie zum Messeverhalten von kleinen und mittleren Unternehmen in Mecklenburg-Vorpommern

Heft 06/2018: Unravelling the secrets of euro area inflation – a frequency decomposition approach: Dieter Gerdesmeier, Barbara Roffia, Hans-Eggert Reimers

ISBN 978-3-942100-65-6