Dienstag, 21. Juni 2011: Dantzig-Wolfe Decomposition

1 Air Crew Rostering

In der letzten Stunde haben wir uns mit dem Problem beschäftigt, Flüge zu *Umläufen* zusammenzustellen. Ein Umlauf ist eine Abfolge von Flügen, Pausen und „Deadheads“, die in Bezug auf alle Arbeitszeitvorschriften und eine Reihe weiterer Nebenbedingungen zulässig ist und die von einer Crew bedient werden kann. Bis jetzt ist diese Zuordnung aber anonym: Wir wissen noch nicht, wie sich eine Crew konkret zusammensetzt, haben also noch keine Dienstpläne für die einzelnen Angestellten der Fluggesellschaft vorliegen. Das Problem, einzelne Personen auf die Umläufe zuzuweisen, bezeichnet man als *Rostering* oder *Dienstplanung* („roster“ = Dienstplan). In dieser Stunde lernen wir ein Netzwerkfluss-Modell kennen, mit dem sich die Dienstplanung abbilden lässt.

Wir gehen für die Dienstplanung davon aus, dass wir eine Reihe von Umläufen haben, für die Personalanforderungen bekannt sind. Im Folgenden betrachten wir nur die Piloten für unsere TUMair, für Copiloten und Kabinenpersonal funktioniert die Dienstplanung dann analog. Es kann natürlich Umläufe mit verschiedenen Anforderungen geben – beispielsweise schränkt die Art eines Fluges oder das Fluggerät die Auswahl an Piloten ein, manche Flüge kommen mit einem Piloten aus, andere benötigen vielleicht zwei. Wir gehen außerdem davon aus, dass unsere Piloten gewissen Vorlieben haben – einige fliegen vielleicht lieber Langstrecken, andere Kurzstrecken oder es gibt Vorlieben für gewisse Städte; für jeden Umlauf und jeden Piloten gibt es also eine Bewertung. Unsere Aufgabe ist es jetzt, jedem Umlauf zu allen möglichen Terminen (ein Umlauf könnte ja mehrmals stattfinden) genau die Anzahl an Piloten zuzuordnen, die benötigt werden. Wir gehen dabei von einer bestimmten Planungsperiode aus, typischerweise ein Monat oder eine Woche. Natürlich muss der Dienstplan neben den Dienstzeiten auch Freizeit und Pausen für unser Personal vorsehen. Für manche Umläufe wird es dabei nötig sein, anschließend eine gewissen Mindestpause einzuhalten, bevor ein neuer Umlauf beginnt.

Um das Problem darzustellen, übertragen wir es in einen Graphen. Dazu stellen wir jeden Umlauf, der zu planen ist, als *Umlaufkante* dar, Anfangs- und Endknoten repräsentieren dann jeweils einen Zeitpunkt. Wir verbinden den Endknoten eines Umlaufs mit dem Startknoten eines späteren Umlaufs, wenn die entsprechende Kombination mit der vorgesehenen Pause möglich ist. Zur Unterscheid sprechen wir bei solchen Verbindungen von einer *Pausenkante*. Daneben führen wir noch *Freizeitkanten* ein, die jeweils für einen komplett freien Tag stehen. Für jeden Piloten gibt es jeweils einen Anfangs- und einen Endkonten, gesucht ist dann für jeden Piloten ein Pfad vom Anfangs- zum Endknoten. Formal können wir das als ein Multicommodity Flow-Problem begreifen, bei der für jeden Piloten ein Gut steht und jeweils ein Fluss der Größe 1 gesucht ist. Diese Interpretation gestattet uns auch eine natürliche Einbeziehung der Anzahlbedingungen: Auf jeder Umlaufkante muss der Gesamtfluss exakt gleich der Anzahl benötigter Piloten sein (wir haben also formal eine Ober- und eine Untergrenze für den Gesamtfluss auf diesen Kanten). Wenn ein Pilot bestimmte Umläufe nicht bedienen kann (weil er e. g. den eingesetzten Flugzeugtyp nicht fliegen darf), so setzen wir zusätzlich die Kantenkapazität für diesen einzelnen Piloten auf 0 (der Graph kann also je nach Pilot etwas anders aussehen). Für die Optimierung können wir einfach die Vorlieben unserer Piloten auf die Umlaufkanten setzen und suchen dann nach einem möglichst gut bewerteten Umlauf. Hier kann man sich natürlich auch andere Kriterien überlegen, im Folgenden geht es uns hauptsächlich um die Gewinnung eines zulässigen Dienstplanes. Wir verwenden folgende Notation:

U	Menge der optimalen Umläufe des Pairing Problems
K	Menge der Flugkapitäne
$\mathbf{w}_{uk}$	Wert des Umlaufes $u \in U$ für Kapitän $k \in K$
$\mathbf{d}_u$	Anzahl der für Umlauf $u \in U$ benötigten Flugkapitäne
E	Menge aller Kanten (Umläufe, Pausen und Freizeit)
V	Menge aller Knoten (Start- und Endzeitpunkte der Kanten)

Den Fluss auf jeder Kante bezeichnen wir mit x_e, dabei ist e entweder eine Umlaufkante (also $e \in U$), eine Pausen- oder Freizeitkante, die gesamte Kantenmenge bezeichnen wir mit E. Die Bewertung für die Pausen- und Freizeitkanten setzen wir jeweils auf 0. Außerdem führen wir einen Startknoten s und einen Endknoten t zu Beginn bzw. zum Ende der Planungsperiode ein, die wir jeweils mit den ersten bzw. letzten Umläufen bzw. dem Start-/Endknoten der ersten bzw. letzten Freizeitkante verbinden. Damit erhalten wir eine klassische Multicommodity Flow-Formulierung:

$$
\begin{aligned}
\max \quad & \sum_{k \in K} \sum_{e \in E} w_{ke} x_{ke} \\
& \sum_{k \in K} x_{ku} = d_u \quad \text{für alle } u \in U \\
& \sum_{e \in \delta^+(v)} x_{ke} - \sum_{e \in \delta^-(v)} x_{ke} = 0 \quad \text{für alle } v \in V \backslash \{s,t\} \text{ und alle } k \in K \\
& \sum_{e \in \delta^-(s)} x_{ke} = 1 \quad \text{für alle } k \in K \\
& \sum_{e \in \delta^+(t)} x_{ke} = 1 \quad \text{für alle } k \in K \\
& x_{ke} \in \mathbb{N}_0 \quad \text{für alle } k \in K,\ e \in E
\end{aligned}
$$

Entscheidend ist nun die Struktur unseres Problems: Es gibt nur wenige Bedingungen, in denen tatsächlich alle (oder fast alle) Güter gemeinsam vorkommen, nämlich die Kapazitätsbeschränkungen der Umlaufkanten. Würde man diese Beschränkungen entfernen, so zerfiele das Problem in lauter kleine Teilprobleme, nämlich in ein einfaches Flussproblem für jeden unserer Piloten. Ziel der Dantzig-Wolfe-Decomposition ist es, solche Strukturen auszunutzen.

2 Dantzig-Wolfe-Decomposition

2.1 Grundidee

Wir stellen die Idee etwas allgemeiner dar: Unser Problem ist ein großes lineares Problem, bei dem die Nebenbedingungen überwiegend Blockdiagonalstruktur besitzen und darüber hinaus ein paar wenige „Kopplungsbedingungen“ existieren. (Anmerkung: Damit die Darstellung nicht unübersichtlich wird, verzichten wir im Folgenden manchmal auf das Transponieren – die w_k wie auch die x_k sind jeweils Vektoren, die passend

als Zeilen oder Spalten interpretiert werden sollen.)

$$\begin{aligned} &\max\,(w_1, w_2, \ldots, w_{|K|}) \cdot (x_1, x_2, \ldots, x_{|K|}) \\ &\begin{pmatrix} A_1 & A_2 & \cdots & A_{|K|} \\ B_1 & & & \\ & B_2 & & \\ & & \ddots & \\ & & & B_{|K|} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{|K|} \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{|K|} \end{pmatrix} \\ &x \geq 0 \end{aligned} \tag{1}$$

Das Problem beinhaltet also eine Reihe von Teilproblemen der Form $B_i x_i = b_i$, die alle über ein paar wenige Zeilen in der Form

$$A_1 x_1 + A_2 x_2 + \cdots + A_{|K|} x_{|K|} = b_0$$

miteinander verkoppelt sind. Solche Probleme entstehen häufig, wenn eine Reihe von Bedingungen aus realen Zusammenhängen entstammen, die einzelne Einheiten miteinander in Beziehung setzen. Das können einzelne Personen sein, Maschinen, die lokalen und globalen Betriebsbedingungen genügen müssen, geographische Zusammenhänge, zeitliche Blöcke und ähnliches mehr.

Wir wollen die Blockstruktur der Matrix ausnutzen, um das Problem in eine Reihe kleinerer Teilprobleme aufzuspalten. Naheliegend wäre natürlich die Idee, die Zulässigkeitsprobleme

$$B_i x_i = b_i$$

zunächst getrennt zu lösen und die Lösung dann in die Kopplungsbedingungen einzusetzen. Das wird aber im Allgemeinen dazu führen, dass wir unzulässige Lösungen bekommen – schließlich kann man beim Lösen eines Teilproblems nicht absehen, wie die Teillösung mit anderen Teillösungen „zusammenspielen“ wird. Die entscheidende Idee ist nun, eben nicht nur jeweils eine Lösung der Teilprobleme zu generieren, sondern die Menge aller möglichen Lösungen für jedes Teilproblem zu betrachten und diese Teillösungen passend zu kombinieren. Wir formalisieren die Idee im Folgenden.

2.2 Minkowski-Formulierung

Das Dantzig-Wolfe-Verfahren beruht auf dem Satz von Minkowski. Wir formulieren das Verfahren und alle Aussagen im Folgenden immer nur für Polytope – für Polyeder gilt alles analog, wenn man noch Extremstrahlen und deren positive Hülle mit zulässt.

Theorem 1 *Sei $P = \{x \geq 0 : Bx = b\}$ ein Polytop und sei $\mathrm{Ext}(P)$ die Menge der Extrempunkte von P. Dann gilt: $P = \mathrm{conv}\,(\mathrm{Ext}(P))$.*

Wir können unser lineares Programm damit anders formulieren: Zu einem Vektor x bezeichnen wir mit x_i jeweils den „passenden“ Koordinatenanteil aus der Formulierung (1). Seien

$$P_i := \{x \geq 0 : B_i x_i = b_i\}$$
$$\text{und} \quad \mathrm{Ext}(P_i) = \left\{x_i^{(j)} : j \in J_i\right\}$$

mit einer passenden Indexmenge J_i. Dann können wir für x_i jeweils eine Konvexkombination der $x_i^{(j)}$ einsetzen, bei der die Parameter noch zu bestimmen sind, also

$$\begin{aligned} x_i &= \sum_{j \in J_i} \lambda_i^{(j)} x_i^{(j)}, \\ \sum_{j \in J_i} \lambda_i^{(j)} &= 1, \\ \lambda_i^{(j)} &\geq 0 \quad \text{für alle } j \in J_i. \end{aligned}$$

Setzt man das in unser Ausgangsproblem (1) ein, so erhält man das sogenannte *Master Problem*:

$$\begin{aligned} \max \ & \sum_{j \in J_1} \lambda_1^{(j)} \cdot w_1^T x_1^{(j)} + \sum_{j \in J_2} \lambda_2^{(j)} \cdot w_2^T x_2^{(j)} + \cdots + \sum_{j \in J_{|K|}} \lambda_{|K|}^{(j)} \cdot w_{|K|}^T x_{|K|}^{(j)} \\ & \sum_{j \in J_1} \lambda_1^{(j)} \cdot A_1 x_1^{(j)} + \sum_{j \in J_2} \lambda_2^{(j)} \cdot A_2 x_2^{(j)} + \cdots + \sum_{j \in J_{|K|}} \lambda_{|K|}^{(j)} \cdot A_{|K|} x_{|K|}^{(j)} = b_0 \\ & \sum_{j \in J_1} \lambda_1^{(j)} = 1 \\ & \sum_{j \in J_2} \lambda_2^{(j)} = 1 \\ & \quad \vdots \\ & \sum_{j \in J_{|K|}} \lambda_{|K|}^{(j)} = 1 \\ & \lambda_i^{(j)} \geq 0 \quad \forall i, j \end{aligned}$$

Wir behalten also nur die Kopplungsbedingungen bei, die Teilprobleme ersetzen wir durch die Menge der Ecken der zugehörigen Polyeder. Das führt zu einem LP in den (sehr vielen) Variablen $\lambda_i^{(j)}$. Das sind natürlich viel zu viele Variablen, dafür haben wir aber alle B_i-Bedingungen aus der Formulierung entfernt (besser gesagt, wir haben sie in die Definition der $x_i^{(j)}$ „ausgelagert“). Natürlich müssten wir zunächst einmal alle Extrempunkte der Teilprobleme berechnen, um das Problem aufzustellen – das klingt zunächst nicht nach einer besonders intelligenten Idee, schließlich müsste man da sehr viel Rechenarbeit investieren und dabei das Problem um ein Vielfaches vergrößern, das wollen wir tunlichst vermeiden. Was hilft uns die Idee also? Wer sich an die letzte Stunde erinnert, wird jetzt sicher an Column Generation denken!

2.3 Column Generation

Die entscheidende Idee ist es, eben nicht alle Ecken der Teilpolyeder auf einmal zu generieren, sondern diese erst nach und nach auf Anforderung zu berechnen. Dabei sollen natürlich Ecken berechnet werden, die auch zu einer Verbesserung der Zielfunktion beitragen können. Wir versuchen also, das Problem mit einem Column Generation-Ansatz zu verbinden.

Um die reduzierten Kosten zu bestimmen, dualisieren wir zunächst das Master Problem:

$$\begin{aligned}
\min\ & \sigma^T b_0 + \pi_1 + \cdots + \pi_{|K|} \\
& \sigma^T A_1 x_1^{(j)} + \pi_1 \geq w_1^T x_1^{(j)} \quad \text{für alle } j \in J_1 \\
& \sigma^T A_2 x_2^{(j)} + \pi_2 \geq w_2^T x_2^{(j)} \quad \text{für alle } j \in J_2 \\
& \quad \vdots \\
& \sigma^T A_{|K|} x_{|K|}^{(j)} + \pi_{|K|} \geq w_{|K|}^T x_{|K|}^{(j)} \quad \text{für alle } j \in J_{|K|} \\
& \sigma \in \mathbb{R}^m \\
& \pi_i \in \mathbb{R} \quad \text{für } i = 1, \ldots, |K|
\end{aligned}$$

(dabei ist m die Anzahl von Kopplungsbedingungen, also die Zeilenzahl der Matrizen $A_1, \ldots, A_{|K|}$). Die reduzierten Kosten berechnen sich dann zu

$$\begin{pmatrix}
w_1^T x_1^{(j)} - \pi_1 - \sigma^T A_1 x_1^{(j)} \\
w_2^T x_2^{(j)} - \pi_2 - \sigma^T A_2 x_2^{(j)} \\
\vdots \\
w_{|K|}^T x_{|K|}^{(j)} - \pi_{|K|} - \sigma^T A_{|K|} x_{|K|}^{(j)}
\end{pmatrix}.$$

Jetzt führen wir eine Column Generation durch: Sei $J = \bigcup_{i=1}^{|K|} J_i$, $J' \subset J$ und $J_i' = J' \cap J_i$. Dann bezeichnen wir das Problem

$$\begin{aligned}
\max\ & \sum_{j \in J_1'} \lambda_1^{(j)} \cdot w_1^T x_1^{(j)} + \sum_{j \in J_2'} \lambda_2^{(j)} \cdot w_2^T x_2^{(j)} + \cdots + \sum_{j \in J_{|K|}'} \lambda_{|K|}^{(j)} \cdot w_{|K|}^T x_{|K|}^{(j)} \\
& \sum_{j \in J_1'} \lambda_1^{(j)} \cdot A_1 x_1^{(j)} + \sum_{j \in J_2'} \lambda_2^{(j)} \cdot A_2 x_2^{(j)} + \cdots + \sum_{j \in J_{|K|}'} \lambda_{|K|}^{(j)} \cdot A_{|K|} x_{|K|}^{(j)} = b_0 \\
& \sum_{j \in J_1'} \lambda_1^{(j)} = 1 \\
& \sum_{j \in J_2'} \lambda_2^{(j)} = 1 \\
& \quad \vdots \\
& \sum_{j \in J_{|K|}'} \lambda_{|K|}^{(j)} = 1 \\
& \lambda_i^{(j)} \geq 0 \quad \forall i, j
\end{aligned}$$

als *reduziertes Master-Problem*. Wir starten also wie üblich mit einer Teilmenge der zulässigen Spalten (= Variablen) J' des gesamten Problems. Haben wir ein Optimum über dieser reduzierten Variablenmenge gefunden und die zugehörigen Werte der dualen Variablen bestimmt, so können wir nach neuen Spalten mit negativen reduzierten Kosten suchen, die noch zu einer Verbesserung des aktuellen Zielfunktionswertes führen könnten. Dazu müssen wir das Problem

$$\max_{j \in J_i} \left\{ w_i^T x_i^{(j)} - \pi_i - \sigma^T A_i x_i^{(j)} \right\}$$

lösen. Das bedeutet aber nichts anderes, als eine optimale Ecke für das LP

$$\begin{aligned}
\max\ & w_i^T x_i - \pi_i - \sigma^T A_i x_i \\
& B_i x_i = b_i \\
& x_i \geq 0
\end{aligned}$$

zu bestimmen. Ein relativ kleines LP, das wir gut im Griff haben sollten – für unser konkretes Beispiel war das ja sogar ein Flussproblem. Zudem lassen sich alle LPs dieser Form unabhängig voneinander lösen, hier kann man also sogar parallelisieren (was in Zeiten von Multicore-Prozessoren durchaus interessant ist). Jedes LP liefert dann einen Kandidaten für eine neue Spalte (oder die Auskunft, dass ein Teilproblem keinen solchen Kandidaten mehr ermöglicht). Diese Kandidaten nehmen wir alle in unser reduziertes Master-Problem auf und lösen wieder von vorne. Der Algorithmus endet, wenn kein Teilproblem mehr eine neue Spalte liefert – in diesem Fall haben wir das Optimum erreicht.

2.4 Zusammenfassung

Zusammenfassen kann man den Dantzig-Wolfe-Algorithmus also wie folgt:

1. Bringe das Problem in Blockdiagonalform mit möglichst wenigen Kopplungsbedingungen (gegebenenfalls kann man mit Zeilen- und Spaltentausch versuchen, so eine Struktur herzustellen) und stelle mit Hilfe des Satzes von Minkowski das Master-Problem auf. Bestimme für jedes Teilproblem mindestens eine Ecke, das ergibt die Indexmenge J', das reduzierte Master-Problem und die zugehörigen Punkte $x_i^{(j)}$.
2. Löse das reduzierte Master-Problem und bestimme die zugehörige duale Lösung.
3. Übergib die Werte der dualen Variablen an jedes Teilproblem und löse das Teilproblem. Jedes Teilproblem liefert also eine neue Ecke $x_i^{(j)}$ und einen neuen Index für J' zurück – oder aber die Information, dass sich hier keine profitablen Ecken mehr generieren lassen.
4. Wenn kein Teilproblem neue Ecken generiert hat, ist die aktuelle Lösung optimal. Ansonsten gehe zu Punkt 2.
5. Natürlich kann man den Algorithmus auch nach einer bestimmten Höchstzahl von Iterationen abbrechen oder dann, wenn man mit der Güte der Lösung zufrieden ist.
6. Statt Ecken zu generieren kann man genauso die Menge aller zulässigen (ggf. ganzzahligen) Punkte der Teilprobleme betrachten, die Argumentation funktioniert im Wesentlichen genauso (allerdings gibt es evtl. deutlich mehr Kombinationsmöglichkeiten, was zu Symmetrieproblemen führen könnte). Häufig ist das Generieren zulässiger ganzzahliger Punkte einfacher und schneller als bei der Einschränkung auf Ecken. Die Einschränkung auf ganzzahlige Punkte (bei ganzzahligen Problemen) kann sogar dazu führen, dass man bessere Relaxationen im Master-Problem erhält.
7. Integriert man Dantzig-Wolfe-Decomposition in ein Branch & Bound-Verfahren, so brancht man meistens auf den ursprünglichen Variablen (was dann einem Constraint Branching in der Minkowski-Formulierung entspricht), weil das besser zur eigentlichen Problemstruktur passt (und typische Probleme wie Symmetrie und unbalancierte Branch & Bound-Bäume vermeiden hilft).