Freitag, 10. Juni 2011: Column Generation

1 Column Generation Modellierung

Anstelle von Variablen, welche die Crews zeitlich und örtlich verfolgen, betrachten wir Variablen, welche über die Umläufe indiziert sind. Sei U die Menge der zulässigen Umläufe. Das heißt, jedes Element aus U erfüllt bereits alle Bedingungen:

- Flugzeiten wurden berücksichtigt
- Start/Ende an Heimatbasis der Crew
- maximale Dauer eines Umlaufs wird eingehalten
- maximale Flugdauer während eines Umlaufs wird eingehalten
- minimale Anschlusszeit zwischen Flügen wird eingehalten
- vorgeschriebene Pausen zwischen Flügen wird eingehalten

Identifiziert man die Variablen

$$x_u = 1 \qquad \text{genau dann, wenn Umlauf } u \text{ von einer Crew gemacht wird}$$

so ergibt sich folgende Modellierung des Airline Crew Scheduling Problems:

$$\begin{aligned} \min \sum_{u \in U} & c_u x_u \\ \sum_{u \in U : f \in u} & x_u = 1 \qquad \forall\, f \in F \\ & x_u \in \{0,1\} \qquad \forall\, u \in U \end{aligned} \tag{SP}$$

wobei c_u die Kosten des Umlaufes u und F die Menge aller von der Fluggesellschaft angebotenen Flüge. (Es ergibt sich also ein Set Partitioning Problem.)

Die Menge U der zulässigen Umläufe ist sehr groß und die Modellierung hat exponentiell viele Variablen. Daher betrachtet man zunächst ein Problem mit einer kleineren Variablenmenge mit der Indexteilmenge $U' \subset U$:

$$\begin{aligned} \min \sum_{u \in U'} & c_u x_u \\ \sum_{u \in U' : f \in u} & x_u = 1 \qquad \forall\, f \in F \\ & x_u \in \{0,1\} \qquad \forall\, u \in U' \end{aligned} \tag{SP'}$$

(Eine andere Interpretation: Die Variablenmenge wird nicht reduziert, sondern die Variablen teilweise auf Null gesetzt.)

2 Reduzierte Kosten

Wir betrachten nun die Relaxation unseres Problems:

$$\begin{aligned} \min \sum_{u \in U} c_u x_u & \\ \sum_{u \in U : f \in u} x_u = 1 & \qquad \forall\, f \in F \\ x_u \geq 0 & \qquad \forall\, u \in U \end{aligned} \tag{P}$$

So ein lineares Problem würden wir mit dem Simplex-Algorithmus lösen. Erinnern wir uns kurz daran, wie der Simplex-Algorithmus vorgeht:

1. Wir starten in einer Ecke des zulässigen Bereichs
2. Wir bestimmen in dieser Ecke eine Kante, entlang derer der Zielfunktionswert verbessert werden kann (wenn es dafür mehrere Möglichkeiten gibt, müssen wir eine auswählen – die Auswahlregel dafür bezeichnet man als *Pivotregel*).
3. Wir laufen entlang der gewählten Kante zur nächsten Ecke.
4. Das ganze wird wiederholt, bis wir uns nicht mehr verbessern können. Der Algorithmus endet also an einer optimalen Ecke.

Ein paar Details schauen wir uns noch etwas genauer an: Erstens arbeitet der Simplex-Algorithmus eigentlich nicht mit Ecken, sondern mit *Basislösungen*. Das heißt, wir betrachten jeweils nur so viele Nebenbedingungen (die *Basis*), wie benötigt werden, um eine Ecke eindeutig festzulegen (bei n Variablen brauchen wir also n Gleicheitsbedingungen). Das führt dazu, dass zu einer Ecke mehrere Basislösungen gehören können, ein Simplex-Schritt kann dann auch darin bestehen, die Basis auszutauschen, aber trotzdem an derselben Ecke hängenzubleiben. Es gibt Pivotregeln, die sicherstellen, dass der Algorithmus dabei nicht ins Zykeln kommt.

Wenn wir Problem (P) genauer betrachten, stellen wir fest, dass zu den $|U|$ Variablen bereits $|F|$ Gleichheitsbedingungen vorhanden sind. Um eine Ecke des zulässigen Bereichs eindeutig festzulegen, benötigen wir $|U|$ Gleichheitsbedingungen, die fehlenden Bedingungen müssen daher aus den „$x_u \geq 0$“-Bedingungen „aufgefüllt“ werden. Es gilt also für *jede Ecke* des zulässigen Bereichs: Mindestens $|U| - |F|$ der x_u-Variablen sind gleich 0. Weil U sehr (sehr, sehr!) groß ist und F relativ klein, sind das „fast alle“ Variablen. Das bedeutet, an jeder Ecke (und damit auch im Optimum) von (P) haben fast alle x_u-Variablen den Wert 0, wir könnten diese Variablen also einfach weglassen (und wären damit bei einem Problem vom Typ (SP') bzw. dessen Relaxation). Damit wäre das Problem klein genug, um es aufzuschreiben und an einen Solver zu „verfüttern“. Allerdings: Wir wissen ja nicht vorher, welche der Variablen in einer optimalen Lösung nicht 0 sind!

Betrachten wir nochmals den Basiswechsel beim Simplex-Schritt: Eine neue Basis zu finden bedeutet in unserem Problem (P), dass wir eine Variable x_u finden müssen, die aktuell den Wert 0 hat (die wir bisher also ignoriert haben), und für die wir jetzt positive Werte zulassen wollen (das entspricht dem Entlanglaufen auf einer Kante des zulässigen Bereichs). Wie entscheiden wir, welche Variable wir da nehmen sollen?

Weil es ein bisschen übersichtlicher wird, stellen wir das Verfahren zunächst in allgemeiner Matrix-Vektor-Formulierung vor, unser Problem und sein Duales lauten dann:

$$\begin{aligned} \min c^T x & \qquad\qquad \max \mathbb{1}^T y \\ Ax = \mathbb{1} & \qquad\qquad A^T y \leq c \\ x \geq 0 & \end{aligned}$$

Die Variablen, die man für die Definition der aktuellen Ecke benötigt, die also den Wert 0 haben, sind zunächst einmal uninteressant, sie werden als *Nichtbasisvariablen* bezeichnet. Wir schreiben x_N für den entsprechenden Teil des x-Vektors und A_N bzw. c_N für die zugehörigen Teile der Matrix A bzw. des Zielfunktionsvektors c. Alle anderen Variablen dürfen Werte ungleich 0 annehmen, man nennt sie *Basisvariablen* x_B (entsprechend A_B, c_B). Wegen der Komplementaritätsbedingungen

$$x^T(c - A^T y) = 0$$

muss dann auch $c_B - A_B^T y = 0$ gelten, also $c_B = A_B^T y$.

Hier kommt das zweite Detail des Simplex-Algorithmus: Wir wollen herausfinden, welche der x_N-Variablen wir erhöhen sollten, um die Zielfunktion zu verbessern. Das kann man natürlich nicht ganz isoliert betrachten – es soll ja weiter $Ax = \mathbb{1}$ gelten, also müssen wir auch x_B-Werte verändern, wenn eine Nichtbasisvariable geändert werden soll, um weiter zulässig zu bleiben. Wie genau wirkt sich so eine Änderung also im Detail aus? Wir versuchen dazu, den Zielfunktionswert nur in Abhängigkeit von x_N auszudrücken. Dazu stellen wir zunächst fest:

$$\begin{aligned} & Ax = \mathbb{1} \\ \Leftrightarrow & A_B x_B + A_N x_N = \mathbb{1} \\ \Leftrightarrow & x_B = A_B^{-1} \cdot (\mathbb{1} - A_N x_N) \end{aligned}$$

Das können wir nun in die Zielfunktion einsetzen:

$$\begin{aligned} c^T x &= c_B^T x_B + c_N^T x_N \\ &= c_B^T A_B^{-1} \mathbb{1} - c_B^T A_B^{-1} A_N x_N + c_N^T x_N \\ &= c_B^T A_B^{-1} \mathbb{1} + \left(c_N^T - c_B^T A_B^{-1} A_N \right) \cdot x_N \end{aligned}$$

Der erste Term ist konstant, für den zweiten Term gilt $x_N \geq 0$. Die Zielfunktion ändert sich also genau dann in der gewünschten Richtung (hin zu kleineren Werten), wenn die Klammer negativ ist:

$$(c')^T := c_N^T - c_B^T A_B^{-1} A_N < 0$$

Den Wert c' bezeichnet man auch als *reduzierte Kosten*. Wir können ihn noch ein bisschen einfacher ausdrücken, wenn wir auf das duale Problem zurückgreifen und $A_B^T y = c_B$ verwenden:

$$(c')^T = c_N^T - y^T A_B A_B^{-1} A_N = c_N^T - y^T A_N$$

Außerdem gilt (wegen der Komplementaritätsbedingungen) ja auch $c_B - y^T A_B = 0$ (die x_B dürfen ja positiv sein!), damit können wir die Unterscheidung in Basis- und Nichtbasisteil hier wieder aufgeben und einfach schreiben:

$$c' = c - A^T y$$

Diese reduzierten Kosten geben also die potentielle Verbesserung der Zielfunktion an, wenn man eine Variable von ihrem „Null-Joch“ erlöst.

Der entscheidende Trick ist nun: Wir müssen es schaffen, eine Variable mit negativen reduzierten Kosten zu finden (also eine, deren Hinzunahme zu einer besseren Zielfunktion führen könnte), ohne die reduzierten Kosten für alle Möglichkeiten auszurechnen. Wir versuchen also, unter den potentiellen Kanten eine Verbesserungsrichtung zu finden, ohne alle Kanten ausrechnen zu müssen. Um das hinzubekommen, schreiben wir uns zunächst einmal das Duale für unser konkretes Problem auf und leiten daraus einen Term für die reduzierten Kosten ab.

3 Dualisierung

Wir dualisieren (P) mit dem spieltheoretischen Ansatz. Das primale Problem entspricht dem Folgenden:

$$\min_{x \geq 0} \max_{\pi} \sum_{u \in U} c_u x_u + \sum_{f \in F} \pi_f \left(1 - \sum_{u \in U : f \in u} x_u\right)$$

Dualisieren durch Vertauschen von Minimum und Maximum:

$$\max_{\pi} \min_{x \geq 0} \sum_{u \in U} c_u x_u + \sum_{f \in F} \pi_f \left(1 - \sum_{u \in U : f \in u} x_u\right)$$

Nach dem Umstellen ergibt sich:

$$\max_{\pi} \min_{x \geq 0} \sum_{f \in F} \pi_f + \sum_{u \in U} x_u \left(c_u - \sum_{f \in u} \pi_f\right)$$

Das duale Problem lautet daher:

$$\begin{aligned} \max \sum_{f \in F} \pi_f & \\ c_u - \sum_{f \in u} \pi_f \geq 0 & \qquad \forall\ u \in U \\ \pi_f \geq 0 & \qquad \forall\ f \in F \end{aligned} \tag{D}$$

Der Wert

$$c_u - \sum_{f \in u} \pi_f$$

heißt *reduzierte Kosten* von u. Umläufe mit negativen reduzierten Kosten verbessern den Zielfunktionswert.

4 Column Generation

Die Grundstruktur der Column Generation ist in Algorithmus 1 dargestellt.

Input: Primales LP (P) mit Variablenmenge $\{x_i : i \in I\}$, reduziertes LP (P') mit Variablenmenge $\{x_i : i \in I'\}$, wobei $I' \subset I$, duale LPs (D) und (D')
Output: Optimallösung von (P)
repeat
  $x^* :=$ Optimallösung von (P')
  $J := \{i \in I - I' :$ reduzierte Kosten von x_i negativ$\}$
  Wähle $J' \subset J$
  $I' := I' \cup J'$
until $J \neq \emptyset$;
Rückgabe der Lösung

$$x_i = \begin{cases} x_i^* & i \in I' \\ 0 & sonst \end{cases}$$

Algorithm 1: Column Generation

Das Problem eine Variablenmenge mit negativen reduzierten Kosten zu finden wird als *Pricing Problem* bezeichnet.

5 Das Pricing Problem

Erinnerung: Die reduzierten Kosten eines Umlaufes $\widetilde{u}$ sind

$$c_{\widetilde{u}} - \sum_{f \in \widetilde{u}} \pi_f$$

Dabei bezeichnet $c_{\widetilde{u}}$ die Kosten des Umlaufes.

Zur Vereinfachung nehmen wir an, dass wir einen Umlauf als Pfad auf einem Graphen darstellen können und die Kosten des Umlaufes sich als Summe der Gewichte der Pfadkanten ergeben: Sei

$$G = (\{\text{Basis}_{\text{start}}, \text{Basis}_{\text{end}}\} \cup F, A, g)$$

ein gerichteter gewichteter Graph, bei dem zwei Knoten durch eine gerichtete Kante verbunden sind, wenn eine Crew die zugehörigen Flüge hintereinander ausführen kann. Das Gewicht der Kante repräsentiert die Kosten für eventuelle Deadheads.

Betrachten wir die Gewichtsfunktion $g' : E \to \mathbb{R}$ mit

$$g'_{ij} = \begin{cases} g_{ij} - \pi_j & \text{falls } \{i,j\} \in E \text{und } j \in F \\ g_{ij} & \text{falls } \{i,j\} \in E \text{und } j = \{\text{Heimatbasis}\} \end{cases}$$

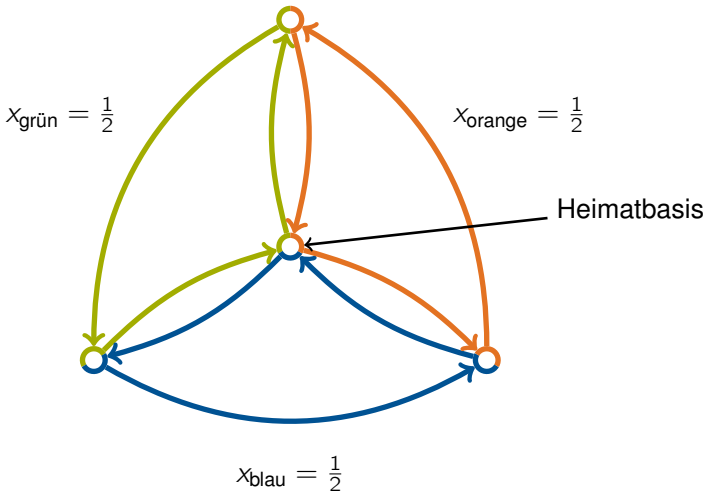
Dann entspricht das Pricing Problem dem Finden eines kürzesten Pfades von $\text{Basis}_{\text{start}}$ nach $\text{Basis}_{\text{end}}$. Ist die Länge dieses Pfades negativ, hat der zugehörige Umlauf negative reduzierte Kosten. Ist sie aber positiv, gibt es keinen solchen Umlauf.

Da der Graph kreisfrei ist, gibt es insbesondere keine Kreise negativer Länge und das Problem kann mit dem Algorithmus von Bellman und Ford oder dem Algorithmus von Floyd und Warshall gelöst werden. (Der Dijkstra-Algorithmus kommt mit den negativen Kantengewichten nicht zurecht.)

Möchte man auch Personalkosten und Übernachtungskosten berücksichtigen, muss man das entsprechende Problem mit Zeitfenstern betrachten.

6 Branch & Price

Leider ist die Lösung der Relaxation der Column Generation Modellierung nicht notwendig ganzzahlig. So könnte beispielsweise folgender Fall auftreten:



Sind die dargestellten Umläufe die einzigen mit der Column Generation generierten, gibt es sogar gar keine zulässige ganzzahlige Lösung. Dann müssen wir Branch&Bound Verfahren und Column Generation kombinieren: Um in einem Baumknoten die Relaxation des zugehörigen Subproblems zu lösen, verwenden wir die Column Generation.

Literatur

[BSSW05] Ralf Borndörfer, Uwe Schelten, Thomas Schlechte und Steffen Weider. *A column generation approach to airline crew scheduling*. 2005.