Technische Universität München
Zentrum Mathematik

Diskrete Optimierung: Fallstudien aus der Praxis

Barbara Wilhelm | Michael Ritter

Das Cutting Stock-Problem

1 Problem

Eine Papierfabrik produziert Papierrollen der Breite B. Für die Auslieferung an die Kunden müssen diese Rollen auf kleinere Breiten (z. B. A4, A3, Zeitschriftenbögen etc.) geschnitten werden (sie werden aber weiter als Rollen ausgeliefert). Die Papierfabrik bietet die Breiten $b_1, \ldots, b_k$ an, die Anzahl der vorliegenden Bestellungen für die Breite b_i sei jeweils d_i (in Stück Rollen). Natürlich ist die Papierfabrik daran interessiert, möglichst wenig Abfall zu produzieren. Wie sollen die Papierrollen geschnitten werden, um alle Bestellungen ausliefern zu können, wenn möglichst wenige Papierrollen benutzt werden sollen? Wieviele Rollen benötigt die Fabrik? Formulieren Sie das Problem als ILP!

- Rollenbreite B
- bestellbare Breiten $b_1, \ldots, b_k$
- bestellte Anzahl $d_1, \ldots, d_k$

2 Einfache Formulierung

2.1 Entscheidungsvariablen

$$y_i = \begin{cases} 0, & \text{Rolle } i \text{ wird nicht verwendet} \\ 1, & \text{Rolle } i \text{ wird verwendet} \end{cases}$$

$x_i^{(j)} =$ Anzahl der Stücke der Breite b_j, die aus Rolle i geschnitten werden

2.2 Modell

Sei M eine Obergrenze für die Anzahl benötigter Papierrollen, z. B. $M := \sum_{i=1}^{k} d_i$.

$$\begin{aligned}
\min \quad & \sum_{i=1}^{M} y_i \\
& \sum_{j=1}^{k} x_i^{(j)} b_j \leq y_i B \quad \text{für alle } i = 1, \ldots, M \\
& \sum_{i=1}^{M} x_i^{(j)} = d_j \quad \text{für alle } j = 1, \ldots, k \\
& y \in \{0,1\}^M \\
& x \in \mathbb{N}_0^{kM}
\end{aligned}$$

2.3 Diskussion

Das Modell hat mehrere gravierende Nachteile, am problematischsten dürfte die Symmetrie sein: Jede Lösung entspricht einer Anzahl von Schnittmustern, die sich natürlich beliebig auf die Rollen verteilen lassen (bei gleichem Zielfunktionswert), es gibt also im schlimmsten Fall $M!$ optimale Lösungen. Selbst für eine gut gewählte Obergrenze M ist das eine enorm große Zahl, die beim Branch & Bound für erhebliche Probleme sorgen wird. Außerdem enthält das ILP eine große Zahl von Bedingungen der Form $\sum_{j=1}^{k} x_i^{(j)} b_j \leq y_i B$, die so etwas wie „big M"-Bedingungen sind. Ist B im Vergleich zu den b_i groß, so muss man damit rechnen, dass diese Bedingungen für die LP-Relaxation nur sehr schwache Ungleichungen liefern, was die Lösung zusätzlich erschwert.

3 Schnittmuster-Formulierung

Die Schnittmuster-Formulierung verwendet eine Liste aller möglichen Schnittmuster. Unter einem *Schnittmuster* verstehen wir eine „Zerlegung" einer kompletten Rolle in Teilstücke. Dafür definieren wir eine Matrix $A = \left(a_i^{(j)}\right)$. Die Spalte $a^{(j)} \in \mathbb{N}_0^k$ der Matrix steht für das j-te Schnittmuster, der Eintrag $a_i^{(j)}$ gibt an, wie viele Stücke der Breite b_i im j-ten Schnittmuster enthalten sind (die Reihenfolge der Stücke spielt ja keine Rolle, ein Schnittmuster ist für uns also durch diese Anzahlen eindeutig charakterisiert). Natürlich gilt für jedes Schnittmuster j die Ungleichung

$$\sum_{i=1}^{k} a_i^{(j)} \leq B.$$

Die Matrix A hat also k Zeilen und eine sehr, sehr, sehr, sehr, sehr große Anzahl an Spalten.

3.1 Entscheidungsvariablen

$y_j =$ Anzahl der Rollen, für die Schnittmuster j benutzt wird

3.2 Modell

Sei m die Anzahl möglicher Schnittmuster (also die Spaltenzahl der Matrix A).

$$\begin{aligned} \min \sum_{j=1}^{m} y_j & \\ \sum_{j=1}^{m} a_i^{(j)} y_j = d_i & \quad \text{für alle } i = 1, \ldots, k \\ y \in \mathbb{N}_0^m & \end{aligned}$$

3.3 Diskussion

Das Modell sieht auf den ersten Blick deutlich einfacher aus als die zuvor betrachtete Formulierung, die dort angesprochenen Symmetrieprobleme werden vermieden. Es gibt aber einen gravierenden Nachteil: Die Anzahl der Variablen entspricht der Anzahl möglicher Schnittmuster, und die ist enorm groß. Außerdem müsste man (um die Matrix A zu kennen) alle möglichen Schnittmuster bestimmen – wie das gehen soll, ist a priori auch nicht klar und sicher mit sehr großem Aufwand verbunden. Allein das Abspeichern der Matrix A dürfte exorbitant viel Speicherplatz benötigen! All diese Probleme haben noch nicht einmal etwas mit der Ganzzahligkeit zu tun, schon die LP-Relaxation des Modells ist riesign. Damit scheidet das Schnittmuster-Modell eigentlich für die praktische Anwendung aus. Es gibt aber eine Lösung für das angesprochene Problem – die sogenannte „Column Generation“!

4 Column Generation

Im Schnittmuster-Modell haben wir eine LP-Relaxation in dualer Form vorliegen:

$$\begin{aligned} \min \mathbb{1}^T y & \\ Ay = d & \\ y \geq 0, y \in \mathbb{R}^m & \end{aligned}$$

Da wir direkt in der dualen Form arbeiten, schreiben wir gleich $A \in \mathbb{R}^{k \times m}$ statt A^T. Die Matrix A hat also k Zeilen, $m \gg k$ Spalten und vollen Rang k (das unterstellen wir einfach, sonst führt man vorher geeignete Reduktionen durch). Problematisch:

- sehr hohe Spaltenzahl von A (und damit sehr hohe Anzahl dualer Variablen)
- nicht alle Spalten von A können gespeichert werden (bzw. es sollen gar nicht alle Spalten explizit ausgerechnet werden, es gibt nur eine implizite Beschreibung der Spalten)

Der Simplex-Algorithmus benutzt im Dualen aber gar nicht alle Spalten der Matrix (so wie er im Primalen nicht alle Zeilen benutzt). Es wird eine Basis $B \subseteq \{1, \ldots, m\}$ gewählt, so dass $A_B y_B = d$ (A_B steht hier für die Teilmatrix von A, die aus den *Spalten* $a^{(j)}$ mit $j \in B$ besteht – also nicht wie im Primalen eine Teilmatrix der Zeilen!) und $y_N = 0$ sowie $y \geq 0$ gilt. Wir wissen bereits, dass in einer Basislösung nur die dualen Basisvariablen y_B positive Werte annehmen können, während für die Nichtbasisvariablen gilt, dass $y_N = 0$. Die Spalten von A, die zu Indizes in N gehören, spielen also gar keine Rolle für die aktuelle Basislösung, insbesondere müssen wir die Spalten nicht kennen oder speichern.

4.1 Der duale Simplex-Algorithmus

Ein Simplexschritt im Dualen läuft ähnlich ab wie im Primalen: Wird in primaler Form eine Nebenbedingung aus der Basis entfernt und eine andere in die Basis hineingetauscht, so entfernt man im Dualen eine Variable aus der Basis und nimmt dafür eine Nichtbasisvariable neu in die Basis auf. Die Auswahl der Nichtbasisvariable muss dabei (analog zum primalen Simplex) so geschehen, dass der Tausch profitabel ist (geometrisch: eine Verbesserungskante).

Im Detail: Sei B eine Basis und $N := \{1, \dots, m\} \setminus B$ die Nichtbasis. Dann lässt sich die j-te Spalte $a^{(j)}$ der Matrix A als Linearkombination der Basisspalten darstellen, d. h., es gibt für jedes $j \in N$ Koeffizienten $\left\{\lambda_i^{(j)} : i \in B\right\}$ mit

$$a^{(j)} = \sum_{i \in B} \lambda_i^{(j)} a^{(i)}.$$

Das kann man folgendermaßen interpretieren: Erhöht man die Variable y_j um eine Einheit, so muss man dafür je $\lambda_i^{(j)}$ Einheiten von den Basisvariablen y_i abziehen, um weiter eine Lösung zu erhalten, für die $Ay = d$ gilt. Für so einen Erhöhungsschritt muss daneben auch weiterhin $y \geq 0$ gelten, diese Bedingung bestimmt also, wie stark y_j erhöht werden kann (und welche Basisvariable im Tausch gegen y_j die Basis verlassen wird). Außerdem wollen wir natürlich nur Austausschritte vornehmen, die eine bessere Lösung liefern. Eine Erhöhung von y_j um eine Einheit verändert die Zielfunktion um

$$1 - \sum_{i \in B} \lambda_i^{(j)},$$

das „Hineinpivotieren“ von y_j ist also dann vorteilhaft, wenn dieser Term negativ ist, d. h.,

$$1 - \sum_{i \in B} \lambda_i^{(j)} < 0.$$

4.2 Pricing

Nach den Ausführungen oben müssten wir in jedem dualen Simplex-Schritt jeweils $1 - \sum_{i \in B} \lambda_i^{(j)}$ für alle $j \in N$ berechnen. Mit $a^{(j)} = \sum_{i \in B} \lambda_i^{(j)} a^{(i)} = A_B \lambda^{(j)}$ ergibt sich $\lambda^{(j)} = A_B^{-1} a^{(j)}$, also suchen wir ein $j \in N$, das die *reduzierten Kosten*

$$\hat{b}^{(j)} := 1 - \mathbb{1}^T \left(A_B^{-1} a^{(j)}\right)$$

minimiert. Ist das Minimum negativ, so haben wir einen passenden Pivotschritt gefunden (sonst ist das aktuelle y bereits optimal).

Das Problem dabei: Wir kennen die $a^{(j)}$ ja gar nicht alle, außerdem sind es sehr viele. Ein einfaches Ausrechnen der reduzierten Kosten für alle Nichtbasiselemente (wie man es normalerweise machen würde, vgl. primaler Simplex) scheidet also aus, wir brauchen eine andere Methode, um eine geeignete Nichtbasisvariable zu finden. Diesen Schritt bezeichnet man auch als *Pricing*.

Betrachten wir das Problem mal etwas genauer:

$$\begin{aligned}
&\min_{j \in N} \; 1 - \mathbb{1}^T \left(A_B^{-1} a^{(j)} \right) \\
\Leftrightarrow &\max_{j \in N} \; \underbrace{\left(\mathbb{1}^T A_B^{-1} \right)}_{=:v^T} a^{(j)} \\
\Leftrightarrow &\max_{j \in N} \; v^T a^{(j)}
\end{aligned}$$

Den Vektor v können wir dabei aus der aktuellen Basis berechnen, die $a^{(j)}$ haben wir immer noch nicht in der Hand. Es gibt aber eine implizite Beschreibung dieser Vektoren. Wir erinnern uns: $a^{(j)}$ steht für ein Schnittmuster, d. h. für einen Vektor $a^{(j)} \in \mathbb{N}_0^k$, der $\sum_{i=1}^k b_i a_i^{(j)} \leq B$ erfüllt. Da jeder solche Vektor eine Nichtbasisspalte von A sein kann, müssen wir also nur folgendes Problem lösen:

$$\begin{aligned}
\max \; & v^T a^{(j)} \\
& \sum_{i=1}^{k} b_i a_i^{(j)} \leq B \\
& a^{(j)} \in \mathbb{N}_0^k
\end{aligned}$$

Das ist aber ein Knapsack-Problem! Im Unterschied zum „klassischen“ Knapsack darf hier jeder Gegenstand auch mehrfach eingepackt werden, die bekannten Algorithmen (z. B. dynamische Optimierung) lassen sich aber einfach auf diesen Fall verallgemeinern (überlegen Sie mal, wie das geht!).

Dieses Problem können wir also lösen, *ohne* alle Schnittmuster explizit aufzählen zu müssen! Dann bleibt nur noch zu prüfen, ob die Lösung einen Wert größer als 1 besitzt, denn nur in diesem Fall ist der Basistausch sinnvoll (weil nur dann die reduzierten Kosten negativ sind). Wenn wir also eine Lösung mit Zielfunktionswert > 1 gefunden haben, tauschen wir die zugehörige y-Variable in die Basis hinein und führen den zugehörigen Simplex-Schritt aus. Es kann übrigens nicht passieren, dass wir als Optimallösung versehentlich eine Basisvariable finden (es sei denn, die Lösung wäre schon optimal), da die reduzierten Kosten aller Basisvariablen immer 0 betragen (das entspräche also einer Knapsack-Lösung mit Zielfunktionswert 1).

4.3 Allgemeine Anwendung

Das beschriebene Column Generation-Verfahren lässt sich natürlich ähnlich auch für viele andere schwere Optimierungsprobleme durchführen, bei denen uns sehr viele Variablen das Leben schwer machen. Voraussetzung ist dabei immer, dass wir in der Lage sind, den Pricing-Schritt effizient durchzuführen. Häufig stößt man dabei auf Probleme, für die polynomielle oder (wie hier) pseudopolynomielle Algorithmen bekannt sind. Findet man nichts derartiges, kann man sich mit einer Pricing-Heuristik behelfen. Man muss ja nicht unbedingt die Nichtbasisvariable mit den *geringsten* reduzierten Kosten finden, es genügt, wenn diese negativ sind.

Bei einer Heuristik kann es natürlich auch passieren, dass keine entsprechende Nichtbasisvariable gefunden wird, obwohl eine existiert – in dem Fall würde man einen möglicherweise profitablen Simplex-Schritt nicht mehr durchführen und erreicht so auch nicht unbedingt das Optimum des Ausgangsproblems. Die Grundprobleme der Heuristik übertragen sich also in gewissem Maße auf das Optimierungsproblem, auch wenn die Heuristik nur als Pricing-Verfahren

eingesetzt wird.

Column Generation ist ein Ansatz, der in reiner Form nur für LP-Probleme geeignet ist. Für die Lösung eines ILP verwendet man Column Generation daher häufig in Kombination mit den bekannten ILP-Verfahren wie Branch & Bound (man spricht von *Branch & Price-Methoden*) oder Branch & Bound und Schnittebenen-Verfahren (sogenannte *Branch & Cut & Price-Methoden*).

Referenzen

[Hei07] Susanne Heipcke. *Embedding Optimization Algorithms*. Techn. Ber. Dash Optimization, 2007. URL: `http://www.dashoptimization.com/home/downloads/pdf/embed.pdf`.

[PS98] Christos H. Papadimitriou und Kenneth Steiglitz. *Combinatorial optimization: Algorithms and complexity*. Corrected reprint of the 1982 original. Dover Publications, Inc., Mineola, NY, 1998.