はじめての列生成法

宮本　裕一郎

この記事の目的は列生成法を平易に解説することである．列生成法は線形計画問題を解くための手法であるが，整数計画問題の高精度な近似解を得る役にも立つ．この記事では，ビンパッキング問題を例に，列生成法の設計と実行を平易に解説する．線形計画問題と双対問題は理解している程度の読者（例えば一通りの基礎を修めた学生）を想定している．

キーワード：列生成法（column generation），線形計画，双対，ビンパッキング

1. はじめに

私の後輩 M 氏は数理最適化（特に整数計画法）の研究者である．ある日の研究打合せにおいて，私が「そこは列を生成すればよいじゃないか」と言ったところ，M 氏は「わかりました，ではそれは先輩がやってください」と言うのであった．

- 名前は知られている．
- 実用において強力であることも知られている．
- でも，設計するのも，実装するのも面倒くさいと思われている．

そんな認識を持たれている手法が列生成法（column generation）である．列生成法を平易に解説し，負の認識を払拭する必要がある．そう痛感した私は筆をとり，この記事を書くことにした．

列生成法は，簡単に言ってしまえば，変数あるいは制約式が極端に多い線形計画問題を解くための枠組みである．解くための枠組みと言っても，実際に大きな問題を効率的に解けるかどうかは試してみなければわからない．

なお，この記事では以下の条件を満たす読者を対象としたい．

- 線形計画問題や整数計画問題の定式化は一応できる．
- 定式化に対する抵抗感はない．
- 線形計画問題の双対問題は作れる．
- 数理計画ソルバーを使うことはできる．
- しかし列生成法なるものは知らない．
- でも知りたい！

みやもと　ゆういちろう
上智大学理工学部情報理工学科
〒 102–8554 東京都千代田区紀尾井町 7–1

なお，列生成法をすでに知っている方，あるいはこの記事を読み終えた読者は当特集号の「船舶スケジューリング数理モデル作成の具体的手順」をお読みいただければ，さらに理解が進むことと思う．また，列生成法に的を絞った専門書 [1] も出版されている．以下では，ビンパッキング問題を例に話を進める．

2. ビンパッキング問題

さまざまな大きさのアイテムと均一な大きさのビンが与えられているとする．このとき，使うビンの個数をなるべく少なくするような，アイテムのビンへの詰め込み（割り当て）を決める問題をビンパッキング問題という[1]．

ビンパッキング問題を数理的に，ある程度厳密に，定義するため，以下に入力・出力・制約・目的を記す．

入力　ビンの容量 B，ビンに詰めるべきアイテムの集合 $I := \{1, \ldots, n\}$，各アイテム $i \in I$ の大きさ s_i

出力　使うビンの数を最小にするような，ビンへのアイテムの詰め込み（割り当て）

制約　各ビンに詰め込まれるアイテムの大きさの合計はビンの容量以下

目的　（アイテムを詰めるために）使うビンの数最小化

小さな問題例（$B = 8$，$I = \{1,2,3,4\}$，$s_1 = 1$，$s_2 = 3$，$s_3 = 6$，$s_4 = 7$）を図 1 に示す．

3. 整数計画問題として普通に定式化

まず，ビンパッキング問題を普通に整数計画問題として定式化してみる．もちろん何が普通かは人によっ

[1] 全くの余談だが，ここでのビンは瓶ではなく大きな箱（bin）である．私はかなり長い間勘違いしていた．

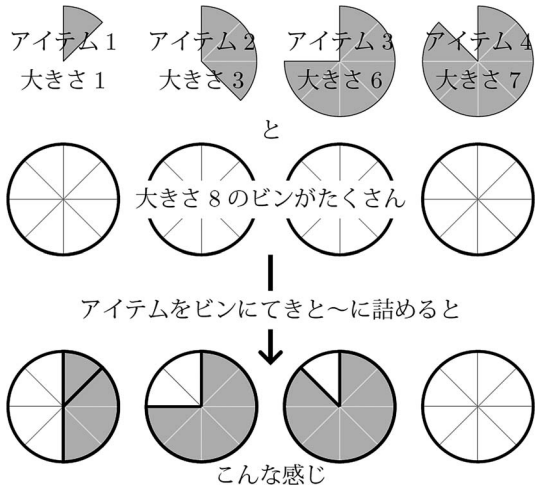

図 1 小さな問題例とその実行可能解．アイテムの部分集合 $\{1,2\},\{3\},\{4\}$ をそれぞれのビンに詰め込んでおり，使うビンの個数は 3 である．

てさまざまである．だが，ある程度定式化に慣れた者ならば，変数や制約式の個数が入力の多項式サイズの定式化をするであろう．

以上を念頭に置き，ここでは，アイテムと同じ個数のビンの集合 $J := \{1,\dots,n\}$ が用意されていると仮定する[2]．そして変数として

- アイテム $i \in I$ をビン $j \in J$ に詰めるとき 1，そうでないとき 0 となる変数 x_{ij}，と
- ビン $j \in J$ が使われるとき 1，そうでないとき 0 となる変数 y_j，

を用意すると，ビンパッキング問題は以下の整数計画問題として定式化できる．

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j\in J} y_j \\
\text{subject to} \quad & \sum_{j\in J} x_{ij} = 1 \quad (i \in I), \\
& \sum_{i\in I} s_i x_{ij} \le B y_j \quad (j \in J), \\
& x_{ij} \in \{0,1\} \quad (i \in I,\ j \in J), \\
& y_j \in \{0,1\} \quad (j \in J).
\end{aligned}
$$

先述の小さな問題例は以下の整数計画問題として表せる．

$$
\begin{aligned}
\text{minimize} \quad & y_1 + y_2 + y_3 + y_4 \\
\text{subject to} \quad & x_{11} + x_{12} + x_{13} + x_{14} = 1, \\
& x_{21} + x_{22} + x_{23} + x_{24} = 1, \\
& x_{31} + x_{32} + x_{33} + x_{34} = 1, \\
& x_{41} + x_{42} + x_{43} + x_{44} = 1, \\
& x_{11} + 3x_{21} + 6x_{31} + 7x_{41} \le 8y_1, \\
& x_{12} + 3x_{22} + 6x_{32} + 7x_{42} \le 8y_2, \\
& x_{13} + 3x_{23} + 6x_{33} + 7x_{43} \le 8y_3, \\
& x_{14} + 3x_{24} + 6x_{34} + 7x_{44} \le 8y_4, \\
& x_{11}, x_{12}, x_{13}, x_{14} \in \{0,1\}, \\
& x_{21}, x_{22}, x_{23}, x_{24} \in \{0,1\}, \\
& x_{31}, x_{32}, x_{33}, x_{34} \in \{0,1\}, \\
& x_{41}, x_{42}, x_{43}, x_{44} \in \{0,1\}, \\
& y_1, y_2, y_3, y_4 \in \{0,1\}.
\end{aligned}
$$

4. 普通の定式化でソルバーに入力

定式化すると，手元に整数計画ソルバーがあるならば，解かせたくなるのが人情である．試しに，アイテムを 10 個として $B = 10$，$s_i \in \{1,2,\dots,9\}$ のランダムな問題例をソルバーに入力してみる．すると，たちどころに最適解が出る[3]．2012 年現在のソルバーの性能はたいしたものである．次に，他の条件はそのままでアイテムを 100 個にした問題例をソルバーに入力してみる．すると，すぐに良さそうな実行可能解が見つかるものの，それが最適解であるかどうかはわからないまま，ソルバーは 1 時間以上計算し続ける[4]．

高精度な（最適解に近い）解を欲しい，あるいは得られた解の良さを知りたい，と思うものの「自分で専用のアルゴリズムを作り実装するのは面倒くさい」と思うのもまた人情である．このような場合に，ものぐさな人間が次に試すのは「定式化をちょっと変えるだけで解けるようにならないかな」というあたりであろう．

5. トリッキーな定式化

少しトリッキーだが，変数の個数が入力の指数サイズになるような定式化をしてみる．

ここでは，あらかじめビンに詰め込み可能なアイテムの詰合せ（パターン）を列挙したとし，その集合を J とする．行列 $A = \{a_{ij}\}$ を，

- アイテム $i \in I$ が詰合せ $j \in J$ に含まれるとき $a_{ij} = 1$，
- そうでないとき $a_{ij} = 0$，

と定める．そしてビンにアイテムを詰める際に詰合せ $j \in J$ を採用するとき 1，そうでないとき 0 となる変

[2] 整数計画問題としての定式化において「登場するものの個数はすべて固定する」のは基本的なテクニックの一つである．

[3] プロセッサ: 3.4GHz Intel Core i7,
メモリ：16 GB 1333 MHz DDR3,
ソルバー：Gurobi Optimizer 4.5.2（8 スレッド使用）．

[4] 正確には 10 回の試行中 8 回は 1 時間以上計算し続けた．

数を x_j とすると，ビンパッキング問題は以下の整数計画問題として定式化できる．

$$\begin{aligned} \text{minimize} \quad & \sum_{j\in J} x_j \\ \text{subject to} \quad & \sum_{j\in J} a_{ij}x_j \geq 1 \quad (i \in I), \\ & x_j \in \{0,1\} \quad (j \in J). \end{aligned}$$

先述の小さな問題例で可能な詰め合わせ（パターン）を列挙すると表 1 に挙げる 7 通りとなる．よって先程のトリッキーな定式化に当てはめると以下の整数計画問題となる．

表 1　小さな問題例における可能な詰合せ

詰合せ番号（名前）	詰合せ	大きさの合計
1	$\{1\}$	1
2	$\{2\}$	3
3	$\{3\}$	6
4	$\{4\}$	7
5	$\{1,2\}$	4
6	$\{1,3\}$	7
7	$\{1,4\}$	8

$$\begin{aligned} \text{minimize} \quad & x_1+x_2+x_3+x_4+x_5+x_6+x_7 \\ \text{subject to} \quad & x_1+x_5+x_6+x_7 \geq 1, \\ & x_2+x_5 \geq 1, \\ & x_3+x_6 \geq 1, \\ & x_4+x_7 \geq 1, \\ & x_1,x_2,x_3,x_4,x_5,x_6,x_7 \in \{0,1\}. \end{aligned}$$

6. トリッキーな定式化でソルバーに入力

定式化すると，手元に整数計画ソルバーがあるならば，解かせたくなるのが人情である．また，先程と同様に，大きさを一様にランダムに決めたアイテム 10 個の問題例をソルバーに入力してみる．すると，たちどころに最適解が出る．2012 年現在のソルバーの性能はやはりたいしたものである．次に，アイテムを 100 個にしてソルバーに入力してみる……と，その前に，そもそも「ビンに詰め込み可能なアイテムの詰め合わせの列挙」がかなり（計算の量的な意味で）困難であることに気がつく，どうしよう？

ここで一度，ソルバーで解いた結果を観察する．小さな問題例の最適値は 3 である．そしておもむろにその線形緩和問題の最適値を計算してみるとそれもまた 3 である．（ソルバーを簡単に使える方は試していただきたい．）すなわち，整数計画問題とその線形緩和問題の最適値が同じなのである．すると「整数計画問題を解く代わりに線形緩和問題を解いてもそれなりに何かの役に立つかな？」という気持ちになる．（なってください！）

よって次の一手として，トリッキーに定式化した整数計画問題の線形緩和問題（線形計画問題）を解くことにする．

7. 線形緩和問題を解いてみる

トリッキーに定式化した整数計画問題の線形緩和問題（線形計画問題）を書き下すと以下のとおりである．

$$\begin{aligned} \text{minimize} \quad & \sum_{j\in J} x_j & (1) \\ \text{subject to} \quad & \sum_{j\in J} a_{ij}x_j \geq 1 \quad (i \in I), & (2) \\ & 0 \leq x_j \leq 1 \quad (j \in J). & (3) \end{aligned}$$

ここで，目的関数 (1) を見ると変数 x_j の和の最小化になっている．よって各変数 x_j は小さいほど嬉しい．一方で制約式 (2) を見ると，係数 a_{ij} の値は 0 か 1 なので，x_j に 1 より大きい値を入れる必要はないことがわかる．よって制約式 (3) の $x_j \leq 1$ を省いても最適解は変わらない．以上の考察により，少しだけ簡単な以下の線形計画問題 (LP) を解けばよいことがわかる．

$$\begin{aligned} \text{(LP): minimize} \quad & \sum_{j\in J} x_j \\ \text{subject to} \quad & \sum_{j\in J} a_{ij}x_j \geq 1 \quad (i \in I), \\ & 0 \leq x_j \quad (j \in J). \end{aligned}$$

以降しばらくはこの線形計画問題 (LP) を解くことに注力する．

さて，この線形計画問題 (LP) もまた，たくさんの変数を含んでいる．よって，ビンに詰めるアイテムが 100 個程度でも，2012 年現在の整数計画ソルバーにそのまま入力するのは難しい．ここで少しだけ見方を変えてみる．

8. 線形緩和問題の双対を解いてみる

線形計画問題とその双対問題の関係を知っている者は，（最適解が存在する問題ならば）どちらを解いても本質的には同じ，ということを知っている．よって，線形計画問題を見て行き詰まった場合にはその双対問題を考えてみる，というのは比較的素直な考え方である．

先ほどのトリッキーな定式化の線形緩和問題 (LP) の双対問題 (DUAL) を書き下すと以下のとおりである．

$$\begin{aligned}\text{(DUAL): maximize} \quad & \sum_{i\in I} y_i \\ \text{subject to} \quad & \sum_{i\in I} a_{ij} y_i \le 1 \quad (j \in J), \\ & y_i \ge 0 \quad (i \in I).\end{aligned}$$

小さな問題例を当てはめると以下のとおりである．（実際に自分の手で変形して試していただきたい．）

$$\begin{aligned}\text{maximize} \quad & y_1 + y_2 + y_3 + y_4 \\ \text{subject to} \quad & y_1 \le 1, \\ & y_2 \le 1, \\ & y_3 \le 1, \\ & y_4 \le 1, \\ & y_1 + y_2 \le 1, \\ & y_1 + y_3 \le 1, \\ & y_1 + y_4 \le 1, \\ & y_1, y_2, y_3, y_4 \ge 0.\end{aligned}$$

この問題 (DUAL) ならば簡単に解けるだろうか？この双対問題 (DUAL) の変数の個数はアイテムの個数（入力の多項式サイズ）である．しかし一方で制約式の本数が多くなっている（一般に入力の指数サイズである）．よって，やはりソルバーに入力することすら難しい．

でも，冷静に考えてほしい．例えば，$n = 10$ として，10 変数（10 次元）の問題で $2^{10} = 1000$ 以上の制約式がすべて必要なことなどあるだろうか？図 2 に 2 変数（2 次元）で制約式がたくさんある実行可能領域を描いてみた．よほど巧妙に問題や入力データを作らない限りは，必要のない制約式ばかりに違いない！

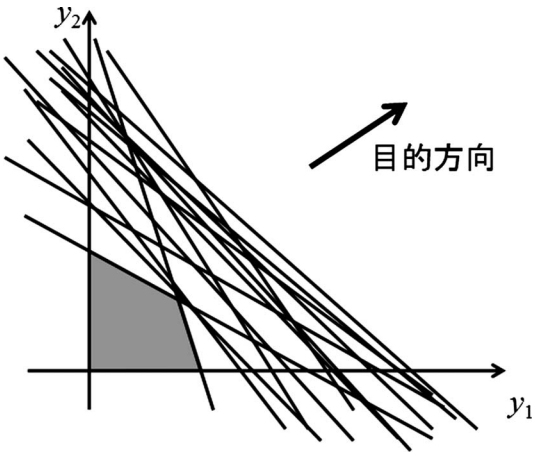

図 2　2 次元の問題にこんなに制約が必要なわけがない．

そこで，自然な発想として「必要な制約式だけつかえばいいじゃないか」という考えに至る．でも，どうすればよいだろうか？最小限必要な制約式だけをいきなり選び出すのは難しそうなので，段階的に見つける方法を模索する．

9. 制約式の段階的な抽出を絵でイメージ

この節では，少しだけビンパッキング問題のことを忘れて，一般的な線形計画問題へのアプローチとして考えてみる．ここで考える線形計画問題の制約式の集合を J とする．そしてこの線形計画問題はものすごくたくさんの制約式を持つとする．（すなわち $|J|$ がものすごく大きいとする．）

まず，すべての制約式 J のうち，いくつかの制約式 $J_1 \subset J$ を適当に選んで暫定的に線形計画問題 (TEMP-1) を作る．TEMP-1の最適解を $\boldsymbol{y}^{*1}$ とする（図 3）．

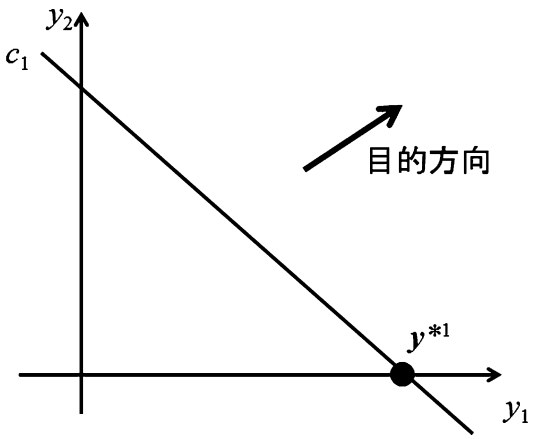

図 3　とりあえず少ない制約式で解いてみる．ここでは制約式は c_1 だけ（$J_1 = \{c_1\}$）である．

次にその最適解 $\boldsymbol{y}^{*1}$ が満たさないような制約式を，J 全体の中から見つけて J_1 に追加したものを J_2 とする．この J_2 を制約式集合として暫定的に作った問題を TEMP-2とし，その最適解を $\boldsymbol{y}^{*2}$ とする（図 4）．次にその最適解 $\boldsymbol{y}^{*2}$ が満たさない制約式を J 全体の中から見つけて J_2 に追加したものを J_3 とする．この J_3 を制約式集合として暫定的に線形計画問題 (TEMP-3) を作る．これを繰り返していけば，いつかは暫定的な問題 TEMP-Kの最適解 $\boldsymbol{y}^{*K}$ が J のすべての制約式を満たす時が来る（図 5）．もちろんここでの K や $|J_K|$ がいくつになるかはやってみなければわからない．最悪の場合は，J のすべての制約式を加えなければこの作

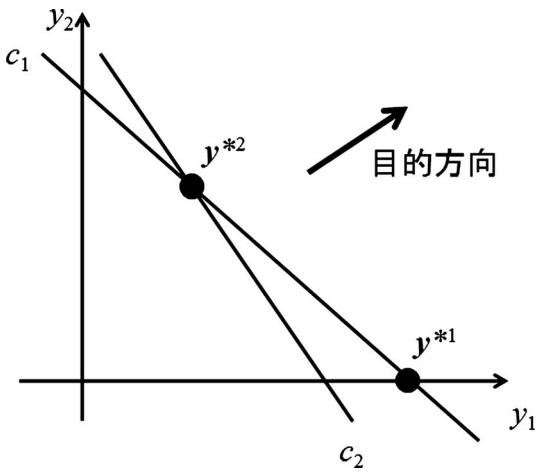

図 4　さらに制約式を加えて解いてみる．ここでは制約式は c_1, c_2 の 2 本だけ（$J_2 = \{c_1, c_2\}$）である．

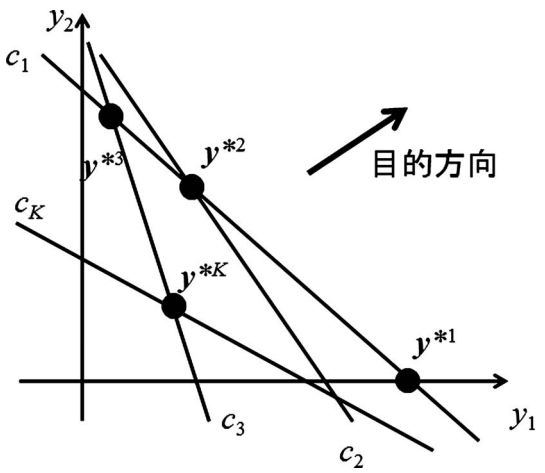

図 5　$\boldsymbol{y}^{*K}$ がすべての制約式を満たした！

業は終わらないかもしれない．しかし，もしかしたら問題例によっては意外と早く終わるかもしれない．よし，これでいこう！

しかし，絵に描くと簡単に見えるが，暫定的な問題 TEMP-kの最適解 $\boldsymbol{y}^{*k}$ が満たさない制約式を J の中から機械的かつ効率的に見つけるにはどうしたらよいだろうか？

10.　ビンパッキング問題の構造を利用

一般には，先ほどの「暫定的な最適解が満たさない制約式」を効率的に見つけるのは難しい．しかし，対象としている問題，今はビンパッキング問題，の構造を利用すれば何かできるかもしれない．ここで，今解きたい線形計画問題 (DUAL) の制約式のうち何本かだけを集めたものを $J_k \subset J$ とする．そしてその制約式 J_k だけで作った以下の問題 (TEMP-k) を考える．

(TEMP-k):

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in I} y_i \\ \text{subject to} \quad & \sum_{i \in I} a_{ij} y_i \leq 1 \quad (j \in J_k \subset J), \\ & y_i \geq 0 \quad (i \in I). \end{aligned}$$

この問題の最適解 $\boldsymbol{y}^{*k}$ が制約式 $l \in J \setminus J_k$ を満たしていないとする．すなわち

$$a_{1l} y_1 + a_{2l} y_2 + \cdots + a_{nl} y_n \leq 1 \tag{4}$$

であるとする．この式の変数 $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)$ に値 $\boldsymbol{y}^{*k} = (y_1^{*k}, y_2^{*k}, \ldots, y_n^{*k})$ を代入すると

$$a_{1l} y_1^{*k} + a_{2l} y_2^{*k} + \cdots + a_{nl} y_n^{*k} > 1 \tag{5}$$

となるはずである．制約式 (4) は，実質的には係数 $a_{1l}, a_{2l}, \ldots, a_{nl}$ で特徴づけられている．よって $\boldsymbol{y}^{*k}$ を入力定数として，式 (5) を満たす $a_{1l}, a_{2l}, \ldots, a_{nl} \in \{0, 1\}$ を効率的な計算により見つけられれば嬉しい．

今一度思い出すと，$a_{ij} \in \{0, 1\}$ は

- アイテム $i \in I$ が詰合せ $j \in J$ に含まれるとき $a_{ij} = 1$,
- そうでないとき $a_{ij} = 0$,

となるような定数であった．よって詰合せ $l \in J$ に関しては

$$s_1 a_{1l} + s_2 a_{2l} + \cdots + s_n a_{nl} \leq B \tag{6}$$

が満たされなければいけない．（そうでないと，その詰合せではビンがあふれてしまう！）以上の考察により，以下の最適化問題 (KS) を解けばよい．

入力　定数値 $\boldsymbol{y}^{*k} = (y_1^{*k}, y_2^{*k}, \ldots, y_n^{*k})$
出力　$\alpha_1, \alpha_2, \ldots, \alpha_n \in \{0, 1\}$
制約　$s_1 \alpha_1 + s_2 \alpha_2 + \cdots + s_n \alpha_n \leq B$
目的　$y_1^{*k} \alpha_1 + y_2^{*k} \alpha_2 + \cdots + y_n^{*k} \alpha_n$ の最大化

そして問題 (KS) の最適値が 1 より大きいならば，求める制約式が見つかったことになる．なお，この問題 (KS) は 0-1 ナップサック問題という名で知られている．

ここまでの考察により，以下の手続き（`Step 1` から `Step 4`）によって，トリッキーな定式化の線形緩和問題の双対問題 (DUAL) を解けることがわかる．

`Step 1`　少なくとも問題 (DUAL) の実行可能解は得られる程度の詰合せ（パターン）を用意する．な

お，詰合せは問題 (DUAL) の制約に対応している．これらを $J_1 \subset J$ とする．k を 1 とする．

Step 2　制約 $J_k \subset J$ のみで暫定的線形計画問題（TEMP-k）を作る．

Step 3　問題 (TEMP-k) の最適解 $\boldsymbol{y}^{*k}$ を得る．

Step 4　その最適解 $\boldsymbol{y}^{*k}$ を入力として 0-1 ナップサック問題 (KS) を解く．

Step 4-a　問題 (KS) の最適値が 1 以下ならば，その時点での暫定的問題 (TEMP-k) の最適解 $\boldsymbol{y}^*$ が元の問題 (DUAL) の最適解なので，終了する．

Step 4-b　そうでないならば，その問題 (KS) の最適解が加えるべき制約式の係数になっているはずなので，J_k にその制約式を加えたものを J_{k+1} とし，k を 1 増やし，Step 2 へ移る．

この手続きが列生成法とよばれるものになっている．

11. 数値実験してみる

例によってランダムに問題例を生成し，上記の手続きを試してみる．手続きを実行する際には，ナップサック問題を解く必要がある．ナップサック問題は NP 困難問題として知られているが，実際にはソルバーにそのまま入力してもかなり早く最適解が得られることが多い[5]．ここでは動的計画法などは使わず，整数計画ソルバーを用いて解くことにする．

これまでと同様に，アイテム 100 個，$B = 10$, $s_i \in \{1, \ldots, 9\}$ のランダムな問題例を生成する．そのような問題例 100 個に対して先ほどの手続きを実行すると，手続きが完了するまでの実行時間は平均で約 3 秒，完了時の制約式＝詰合せ（パターン）の個数は平均で 300 個程度であった[6]．可能な詰合せを列挙すると膨大な数になることと比べると，実際に生成される制約式の本数は意外と少ない．

12. 列生成法一般論

先ほどの手続きをより一般的に書くと以下のとおりになる．これが列生成法（を双対問題の側から見たもの）の枠組みである．

Step A　少なくとも実行可能解は得られる程度の制約式を用意し，それで暫定的に線形計画問題を作る．

Step B　その暫定的線形計画問題の最適解を得る．

Step C　暫定的問題の最適解を入力定数とみなして，その最適解が満たしていない制約式を探索する．

Step C-1　そのような制約式がないならば，手続きを終了する．

Step C-2　そうでないならば，その制約式を暫定的線形計画問題に加え Step B へ移る．

この記事では，列生成法の設計と実行をわかりやすく説明するために，双対問題の観点から，制約式を増やす手法であると説明した．しかし，双対問題は主問題と一対一に対応するので，実は双対問題を経由する必要はない．先ほどの手続きを，主問題の観点から書き直すと，変数を生成していることになる．線形計画問題を行列形式で記述すると，変数の生成は行列の列生成になる．これが列生成法とよばれる所以であろう．

ここまで読み進められた読者は，列生成法を直感的につかめているに違いない．列生成法を正確に（しかも日本語で）記述した書籍 [2] にあたるとより正確に把握できると思われる．

13. 線形緩和問題から整数計画問題へ

ここまで苦労をして，得られたものといえば，線形緩和問題の最適解と最適値だけである．これにどのような価値があるだろうか？

まず，最小化問題の緩和問題の最適値が得られたので，元の問題の最適値の下界が得られた．これはヒューリスティックな解の目的関数値の精度評価に使える．

次に，ここで得られた詰合せ（パターン）を用いたヒューリスティクスの構築が考えられる．「緩和問題の最適解を得るために取りそろえた詰合せは，元の整数計画問題の解を構成するうえでも優秀なのではないか？」と考えるのである．実際にそれをお手軽に試すには，以下を実行すればよい．

Step 5　列生成法終了後に，生成された詰合せ（パターン）だけを用いて（再度）整数計画問題として定式化し，ソルバーに入力して解を得る．

こうして得られた解が，元の問題の最適解である保証

[5] TSP「ナップサックがやられたようだな……」，SAT「ククク……，奴は NP 困難問題の中でも弱……」，MIP「ソルバーごときに負けるとは NP 困難問題の面汚しよ……」

[6] なお，最初に用意するべき「少なくとも実行可能解は得られる程度の詰合せ」として，各アイテム単独の詰合せ $\{1\}, \{2\}, \ldots, \{n\}$ を用いた（今回は $n = 100$）．よって新たに生成された詰合せは $300 - 100 = 200$ 個程度である．

はない．ただのヒューリスティックな解である．しかし，先ほどの問題例で試してみると，このヒューリスティックな解の目的関数値と緩和問題の最適値の差（ギャップ）が 1 未満であり，最適解であった[7]．

これはトリッキーな定式化のおかげである．理論的な根拠には乏しいが，一般に，実行可能解の部品を列挙してその被覆（あるいは分割，あるいは充填）問題として定式化したものは，その線形緩和問題とのギャップが小さいことが経験的に知られている．

14. おわりに

この記事を書く機会を与えてくれた M 氏に感謝したい．M 氏がこの記事を読んだならば，彼の列生成法への負の認識も払拭されるであろう．いや，自ら列生成法の実装をやりたがるに違いない．そんな想いにふけっていると，早速 M 氏から電話がかかってきた．「やあ先輩，『はじめての列生成法』を読みましたよ．さすが先輩です．これだけ良い解説を書けるのだから，さぞかし実装も素晴らしいんでしょうね．ぜひ先輩の実装を見てみたいなあ．」

参考文献

[1] Desaulniers, G., Solomon, M. M. and Desrosiers, J. eds.: *Column Generation*, Springer, 2005.
[2] 久保幹雄，田村明久，松井知己（編）『応用数理計画ハンドブック』，朝倉書店，2002.

[7] 「ギャップが 1 未満ならば最適解である」というのは，トリッキーな定式化の整数性によるものである．なお正確には，100 回の試行におけるギャップの平均は約 0.74 であり，63 回はギャップが 1 未満，すなわち最適解である保証が得られた．

