ZIMPL言語とSCIPによる数理最適化

Mathematical Optimization with ZIMPL and SCIP

ネットワーク情報学部　高野 祐一
School of Network and Information　Yuichi TAKANO

Keywords：Mathematical optimization, Software, Modeling language, Transportation problem, Sudoku

1　はじめに

数理最適化とは，現実の問題を数理モデルとして定式化して解くことにより，意思決定を支援する数理技術である．設備計画・生産計画・スケジューリング・配送計画・資金運用などの問題に対して，実際に多くの企業で数理最適化の技術が活用され，業績の改善や業務の効率化を実現している．

近年は数理最適化のアルゴリズムとコンピュータの性能が飛躍的に向上し，コンピュータを利用して誰もが手軽に最適化問題を解くことができる環境が整ってきた [10]．小規模な最適化問題であればMicrosoft Excelのソルバー機能を使って解くこともできる [7]．一方で大規模な問題や，整数型の決定変数を含むような複雑な問題を扱う場合には，最適化問題を解くための専用ソフトウェアである最適化ソルバー（整数計画ソルバー）が有用だろう．

CPLEX[1]，Gurobi[2]，Xpress[3]は2016年現在いずれも世界最高レベルの性能を持つ最適化ソルバーである．CPLEXは大学に所属する研究者や学生であれば無料で使用できる，GurobiはPython言語との連携が良く用途が広い [9]，Xpressは初心者でも使いやすいなど，それぞれに特徴がある．また，Numerical Optimizer[4]はNTTデータ数理システムにより開発されており，日本語のマニュアルやサポートが利用できる．しかしながらこれらは商用の最適化ソルバーであり，一般には使用するためにソルバーを購入する必要がある．誰でも無料で利用できる最適化ソルバーとしてはCbc[5]，lp_solve[6]，GLPK[7]などがある．特にlp_solveとGLPKは操作も簡単だが，性能は上記の商用ソルバーと比較すると大きく劣る．

本稿では，モデリング言語ZIMPL [3] と最適化ソルバーSCIP（Solving Constraint Integer Programs）[1] の使用方法と応用例を解説する．SCIPはZuse Institute Berlinで開発されている最適化ソルバーであり，近年急速に性能を向上させている．また学術的使用に限り，SCIPは無料で利用できる．

2節では輸送問題を例にZIMPL言語とSCIPの使用方法を解説する．3節でZIMPL言語とSCIPによる数独の解法を紹介し，4節で結論を述べる．

[1] IBM ILOG CPLEX Optimization Studio, http://www-03.ibm.com/software/products/ja/ibmilogcpleoptistud/
[2] Gurobi Optimizer, http://www.octobersky.jp/products/gurobi/gurobi.html
[3] FICO Xpress, http://www.msi-jp.com/xpress/
[4] http://www.msi.co.jp/nuopt/
[5] http://projects.coin-or.org/Cbc
[6] http://lpsolve.sourceforge.net/
[7] http://www.gnu.org/software/glpk/

2　ZIMPL言語とSCIPの使用方法

本節では輸送問題を例にZIMPL言語とSCIPの使用方法を解説する．

2.1　輸送問題とは

まずは輸送問題について説明する．図1に示すように，工場 A_1，A_2 で生産した製品を取引先 B_1，B_2，B_3 に納入することを考える．ただし，各工場の生産量と各取引先からの注文量は決まっているものとする．また，工場と取引先の組に対して製品の輸送コストが定められており，例えば工場 A_1 から取引先 B_3 に製品を輸送するためには，製品1個あたり12のコストがかかる．このような状況で，総輸送コストが最小となる輸送計画を求める問題が輸送問題である．

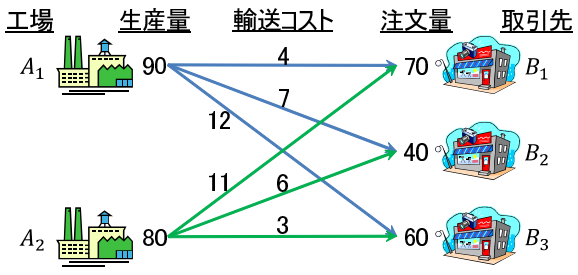

図 1: 輸送問題の例 [8]

輸送問題は製品の輸送以外にも様々な応用例がある．例えば，2枚の画像の非類似度を測る指標の一つにEarth

Mover's Distance（EMD）[5, 6] があり，画像検索などで利用される．この EMD では 2 枚の画像をそれぞれ工場と取引先とみなして輸送問題を解き，その際の総輸送コストが小さいほど 2 枚の画像が類似していると評価する．

2.2 輸送問題の定式化

輸送問題では各工場から各取引先への製品の輸送量が決定変数であり，製品の総輸送コストが最小化すべき目的関数となる．具体的には，工場 A_i から取引先 B_j への輸送量を決定変数 x_{ij} とする．また，図 1 に示した輸送コストを考慮して，総輸送コストは

$$4x_{11} + 7x_{12} + 12x_{13} + 11x_{21} + 6x_{22} + 3x_{23}$$

と表せる．

輸送問題には 3 種類の制約条件がある．まず工場 A_1 から取引先 B_1，B_2，B_3 への輸送量の総和は，工場 A_1 の生産量 90 と一致しなければならない．工場 A_2 についても同様であり，これらの制約条件は以下のように表せる：

$$x_{11} + x_{12} + x_{13} = 90, \quad \cdots \text{工場 } A_1 \text{ の生産量}$$
$$x_{21} + x_{22} + x_{23} = 80. \quad \cdots \text{工場 } A_2 \text{ の生産量}$$

次に工場 A_1，A_2 から取引先 B_1 への輸送量の総和は取引先 B_1 の注文量 70 と一致しなければならない． 取引先 B_2，B_3 についても同様に考えると，これらの制約条件は以下のように表せる：

$$x_{11} + x_{21} = 70, \quad \cdots \text{工場 } B_1 \text{ の注文量}$$
$$x_{12} + x_{22} = 40, \quad \cdots \text{工場 } B_2 \text{ の注文量}$$
$$x_{13} + x_{23} = 60. \quad \cdots \text{工場 } B_3 \text{ の注文量}$$

最後に，工場から取引先への製品の輸送量が負の値となることを防ぐために，輸送量の非負条件

$$x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23} \geq 0$$

が課される．

以上をまとめると，輸送問題は以下のように定式化することができる：

$$\begin{array}{ll} \text{目的関数：} & 4x_{11} + 7x_{12} + 12x_{13} \\ & \quad + 11x_{21} + 6x_{22} + 3x_{23} \ \rightarrow \ \text{最小} \\ \text{制約条件：} & x_{11} + x_{12} + x_{13} = 90, \\ & x_{21} + x_{22} + x_{23} = 80, \\ & x_{11} + x_{21} = 70, \\ & x_{12} + x_{22} = 40, \\ & x_{13} + x_{23} = 60, \\ & x_{11}, x_{12}, x_{13}, x_{21}, x_{22}, x_{23} \geq 0. \end{array} \tag{1}$$

2.3 ZIMPL 言語による輸送問題の記述

モデリング言語 ZIMPL を使って輸送問題 (1) を記述する．まず以下の添え字集合を導入する：

$$I = \{1, 2\}, \quad \cdots \text{工場の添え字集合}$$
$$J = \{1, 2, 3\}. \quad \cdots \text{取引先の添え字集合}$$

これらは ZIMPL 言語では，集合を定義するコマンド `set` を利用して以下のように記述する：

輸送問題の添え字集合

```
set I := { 1, 2 };
set J := { 1 .. 3 };
```

集合 I は要素を直接書き下して定義しているが，集合 J は簡略化した定義になっており，ZIMPL 言語では例えば「`{ 1, 2, 3, 4, 5 }`」は「`{ 1 .. 5 }`」と書くことができる．

次に行列とベクトルを利用して，輸送問題のパラメータを以下のように表す：

$$(c_{ij})_{(i,j)\in I\times J} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \end{pmatrix} = \begin{pmatrix} 4 & 7 & 12 \\ 11 & 6 & 3 \end{pmatrix},$$
$$\cdots \text{工場 } A_i \text{ から取引先 } B_j \text{ への輸送コスト}$$
$$(p_i)_{i\in I} = (p_1, p_2) = (90, 80), \quad \cdots \text{工場 } A_i \text{ の生産量}$$
$$(o_j)_{j\in J} = (o_1, o_2, o_3) = (70, 40, 60).$$
$$\cdots \text{取引先 } B_j \text{ の注文量}$$

これらは ZIMPL 言語では，パラメータを定義するコマンド `param` を利用して以下のように記述する：

輸送問題のパラメータ

```
param c[I*J] :=
      |  1,  2,  3 |
  | 1 |  4,  7, 12 |
  | 2 | 11,  6,  3 |;
param p[I] := <1> 90, <2> 80;
param o[J] := <1> 70, <2> 40, <3> 60;
```

最後に輸送問題の決定変数

$$(x_{ij})_{(i,j)\in I\times J} = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{pmatrix}$$
$$\cdots \text{工場 } A_i \text{ から取引先 } B_j \text{ への輸送量}$$

は，ZIMPL 言語では決定変数を定義するコマンド `var` を使って以下のように記述する：

輸送問題の決定変数

```
var x[I*J];
```

ここで決定変数は自動的に下限が 0 に設定され，非負変数として定義されることに注意してほしい．自由変数（非負条件の無い決定変数）など，変数の上下限を変更する場合は以下のように記述する：

```
var y real >= -infinity;  # y は自由変数
var z real >= -5 <= 5;    # -5 <= z <= 5
```

0-1 変数や整数変数を記述する場合は，実数変数を表す `real` を 0-1 変数 `binary` や整数変数 `integer` で置き換えれば良い．

以上の表記法を用いると，輸送問題 (1) は以下のように書き換えることができる：

$$
\begin{array}{ll}
\text{目的関数：} & \displaystyle\sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} \;\rightarrow\; \text{最小} \\
\text{制約条件：} & \displaystyle\sum_{j\in J} x_{ij} = p_i \quad (\forall i \in I), \\
& \displaystyle\sum_{i\in I} x_{ij} = o_j \quad (\forall j \in J), \\
& x_{ij} \geq 0 \quad (\forall (i,j) \in I \times J).
\end{array}
\tag{2}
$$

上記の定式化は ZIMPL 言語では，最小化する目的関数を定義するコマンド `minimize`（最大化する場合は `maximize`）と制約条件を定義するコマンド `subto` を使って，以下のように記述する：

輸送問題の定式化

```
minimize obj:
  sum <i,j> in I*J: c[i,j]*x[i,j];
subto con1: forall <i> in I do
  sum <j> in J: x[i,j] == p[i];
subto con2: forall <j> in J do
  sum <i> in I: x[i,j] == o[j];
```

`sum` は総和記号（$\sum$）を表すコマンドであり，`forall` は全称記号（$\forall$）を表すコマンドである．問題 (2) と見比べると記述方法も理解できるだろう．また，`obj`，`con1`，`con2` は目的関数と制約条件の名前であり，名前は自由に付けて良い．決定変数は非負変数として定義されているため，非負条件を記述する必要は無い．不等式条件を記述する場合は，等号「`==`」を不等号「`<=`」,「`>=`」で置き換えれば良い．

2.4 SCIP による輸送問題の求解

輸送問題を解くために，文献 [11] の手順に従って最適化ソルバー SCIP のインストールを行なう．Windows 環境では以下の手順で簡単にインストールができる[8]：

1. SCIP のウェブページ（`http://scip.zib.de/`）を開く．

[8]ただし，インストール時には Visual C++が必要となる．

```
set I := { 1, 2 };
set J := { 1 .. 3 };
param c[I*J] :=
      |  1,  2,  3 |
  | 1 |  4,  7, 12 |
  | 2 | 11,  6,  3 |;
param p[I] := <1> 90, <2> 80;
param o[J] := <1> 70, <2> 40, <3> 60;
var x[I*J];
minimize obj:
  sum <i,j> in I*J: c[i,j]*x[i,j];
subto con1: forall <i> in I do
  sum <j> in J: x[i,j] == p[i];
subto con2: forall <j> in J do
  sum <i> in I: x[i,j] == o[j];
```

図 2: 輸送問題の問題ファイル `tp01.zpl`

2. 左側のタブから【Download】を選択する．
3. 【Binaries:】の中から PC 環境に合わせて，【Windows/PC, 32bit, msvc···】もしくは【Windows/PC, 64bit, msvc···】を選択する．
4. ライセンス条項に同意したら【I certify that ···】にチェックし，【Start Download】を選択する．
5. ダウンロードした zip ファイルを解凍すると，SCIP の実行ファイル（scip-····.exe）が作成される．

図 2 のようにテキストファイルを作成し，`tp01.zpl` と名前を付けて SCIP の実行ファイルと同じフォルダに保存する．SCIP の実行ファイルをダブルクリックして SCIP を起動し，SCIP の画面に以下のコマンドを入力する：

輸送問題求解のコマンド

```
SCIP> read tp01.zpl
SCIP> optimize
SCIP> write solution tp01.sol
```

上記のコマンドはそれぞれ

- 問題ファイル `tp01.zpl` の読み込み
- 最適化問題の求解
- 求解結果ファイル `tp01.sol` の作成

を表す．

特に不具合が無ければ，SCIP の実行ファイルと同じフォルダには以下のような求解結果ファイル `tp01.sol` が作成される：

輸送問題の求解結果ファイル `tp01.sol`

```
solution status: optimal solution found
objective value:           720
x#1#1                      70  (obj:4)
x#1#2                      20  (obj:7)
x#2#2                      20  (obj:6)
x#2#3                      60  (obj:3)
```

上記の 1 行目は最適解が見つかったことを示している．2 行目は目的関数の値を示しており，総輸送コストの最小値は 720 である．3 行目以降に最適解が表示されており，値が 0 の場合は表示が省略される．したがって輸送問題の最適解は

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{pmatrix} = \begin{pmatrix} 70 & 20 & 0 \\ 0 & 20 & 60 \end{pmatrix}$$

となる．図 1 と見比べると，工場 A_1 から取引先 B_1 に 70 個輸送し，工場 A_2 から取引先 B_3 に 60 個輸送するといったように，輸送コストが低い工場と取引先の組で製品を多く輸送していることが分かる．

3 数独の解法

本節では ZIMPL 言語と SCIP を用いた数独の解法を紹介する．

3.1 数独とは

数独は 9×9 のマスに 1〜9 の数字を入れるパズルであり，表 1 のようにいくつかのマスに数字が記入された表が問題として与えられる．行列の場合と同様に，マスの横の並びを行，縦の並びを列と呼ぶこととし，i 行 j 列（上から i 番目，左から j 番目）のマスをマス (i, j) と呼ぶこととする．数独では 3×3 のマスによって構成される 9 個のブロックがあり，表 1 では 2 重線によってブロックが区切られている．上から p 個目かつ左から q 個目のブロックをブロック $[p, q]$ と呼ぶこととする．

数字を入れる際には，各行，各列，各ブロック内に同じ数字が入ってはならない（数字は独身に限る：数独）．例えば，表 1 ではマス $(1, 3)$ に数字「2」が記入されているため，1 行目と 3 列目の空欄のマスには「2」を入れることはできないし，左上のブロック $[1, 1]$ 内の空欄のマスにも「2」を入れることはできない．

表 1: 数独の問題例

		2		9			6	
	4				1			8
	7		4	2				3
5						3		
		1		6		5		
		3						6
1				5	7		4	
6			9				2	
	2			8		1		

3.2 数独の定式化

まず，マスやブロックの位置や数字を指定するための添え字集合を以下のように定義する：

$$N = \{1, 2, \ldots, 9\}, \quad B = \{1, 2, 3\}.$$

また，記入済みのマスと数字の集合を

$$G = \{(1, 3, 2), (1, 5, 9), \ldots, (9, 7, 1)\}$$

のように定義する．例えば $(1, 3, 2)$ は，マス $(1, 3)$ に数字「2」が記入されていることを表す．

決定変数は以下のように定義する：

$$x_{ijk} = \begin{cases} 1, & \text{マス } (i, j) \text{ に数字 } k \text{ を入れる場合,} \\ 0, & \text{それ以外.} \end{cases}$$

例えば，$x_{247} = 1$ はマス $(2, 4)$ に数字「7」を入れることを表す．

数独の制約条件は以下のように表される：

$$\sum_{i \in N} x_{ijk} = 1 \quad (\forall (j, k) \in N \times N),$$

‥数字 k は j 列に一つだけ

$$\sum_{j \in N} x_{ijk} = 1 \quad (\forall (i, k) \in N \times N),$$

‥数字 k は i 行に一つだけ

$$\sum_{i=3(p-1)+1}^{3p} \sum_{j=3(q-1)+1}^{3q} x_{ijk} = 1 \quad (\forall (k, p, q) \in N \times B \times B),$$

‥数字 k はブロック $[p, q]$ に一つだけ

$$\sum_{k \in N} x_{ijk} = 1 \quad (\forall (i, j) \in N \times N),$$

‥マス (i, j) に数字を必ず一つ入れる

$x_{ijk} = 1 \quad (\forall (i,j,k) \in G),$

‥記入済みの数字は固定する

$x_{ijk} \in \{0,1\} \quad (\forall (i,j,k) \in N \times N \times N).$

‥決定変数の 0-1 制約

以上をまとめて，数独は以下のように定式化することができる [2]：

$$
\begin{array}{ll}
\text{目的関数：} & \text{無し} \\
\text{制約条件：} & \displaystyle\sum_{i \in N} x_{ijk} = 1 \quad (\forall (j,k) \in N \times N), \\
& \displaystyle\sum_{j \in N} x_{ijk} = 1 \quad (\forall (i,k) \in N \times N), \\
& \displaystyle\sum_{i=3(p-1)+1}^{3p} \sum_{j=3(q-1)+1}^{3q} x_{ijk} = 1 \\
& \qquad (\forall (k,p,q) \in N \times B \times B), \\
& \displaystyle\sum_{k \in N} x_{ijk} = 1 \quad (\forall (i,j) \in N \times N), \\
& x_{ijk} = 1 \quad (\forall (i,j,k) \in G), \\
& x_{ijk} \in \{0,1\} \\
& \qquad (\forall (i,j,k) \in N \times N \times N).
\end{array}
\tag{3}
$$

3.3 数独の求解

ZIMPL 言語によって問題 (3) を記述すると図 3 のようになる [4]．この問題ファイルを `sd01.zpl` と名前を付けて SCIP の実行ファイルと同じフォルダに保存し，SCIP の画面に

数独求解のコマンド

```
SCIP> read sd01.zpl
SCIP> optimize
SCIP> write solution sd01.sol
```

と入力すると，以下のような結果が得られる：

数独の求解結果ファイル `sd01.sol` の一部

```
solution status: optimal solution found
objective value:            0
x#1#1#3                     1  (obj:0)
x#1#2#1                     1  (obj:0)
x#1#3#2                     1  (obj:0)
x#1#4#5                     1  (obj:0)
x#1#5#9                     1  (obj:0)
```

```
set N := { 1 .. 9 };
set B := { 1 .. 3 };
set G := {<1,3,2>, <1,5,9>, <1,8,6>,
  <2,2,4>, <2,6,1>, <2,9,8>, <3,2,7>,
  <3,4,4>, <3,5,2>, <3,9,3>, <4,1,5>,
  <4,7,3>, <5,3,1>, <5,5,6>, <5,7,5>,
  <6,3,3>, <6,9,6>, <7,1,1>, <7,5,5>,
  <7,6,7>, <7,8,4>, <8,1,6>, <8,4,9>,
  <8,8,2>, <9,2,2>, <9,5,8>, <9,7,1>};
var x[N*N*N] binary;
subto con1: forall <j,k> in N*N do
  sum <i> in N: x[i,j,k] == 1;
subto con2: forall <i,k> in N*N do
  sum <j> in N: x[i,j,k] == 1;
subto con3: forall <k,p,q> in N*B*B do
  sum <i,j> in B*B:
  x[3*(p-1)+i,3*(q-1)+j,k] == 1;
subto con4: forall <i,j> in N*N do
  sum <k> in N: x[i,j,k] == 1;
subto con5: forall <i,j,k> in G do
  x[i,j,k] == 1;
```

図 3: 数独の問題ファイル `sd01.zpl`

上記の求解結果にしたがって，

$x_{113} = 1 \quad \Rightarrow \quad$ マス $(1,1)$ に「3」

$x_{121} = 1 \quad \Rightarrow \quad$ マス $(1,2)$ に「1」

$x_{132} = 1 \quad \Rightarrow \quad$ マス $(1,3)$ に「2」

と数字を入れていけば，表 2 のように解答が完成する．

表 2: 数独の解答

3	1	2	5	9	8	7	6	4
9	4	6	7	3	1	2	5	8
8	7	5	4	2	6	9	1	3
5	6	7	8	4	2	3	9	1
4	8	1	3	6	9	5	7	2
2	9	3	1	7	5	4	8	6
1	3	8	2	5	7	6	4	9
6	5	4	9	1	3	8	2	7
7	2	9	6	8	4	1	3	5

4 おわりに

本稿では，輸送問題を例にモデリング言語 ZIMPL と最適化ソルバー SCIP の使用方法を解説し，それらを用いた数独の解法を紹介した．ZIMPL 言語の詳細を知りたい読者は，文献 [3] や ZIMPL User Guide[9]を参照してほしい．最適化ソルバーの機能や使用方法についてより詳しく知りたい読者は，東京農工大学の宮代隆平氏により運営されているウェブページ「整数計画法メモ[10]」や，日本オペレーションズ・リサーチ学会機関誌 2012 年 4 月号「はじめよう整数計画」を参照してほしい．

数理最適化では問題の性質や解法などの理論的な側面が非常に重要である．一方で，コンピュータを利用して実用規模の問題を解くことができるようになった現在では，現実の問題を数理モデルとして定式化し，最適化ソルバーを使って解くことの重要性が格段に増していると感じる．このような背景から自分が担当している「数理計画法」の授業では，講義 2 回分を使って最適化ソルバーの実習を行なっており，本稿はその内容の一部をまとめたものである．

最後に，本号は石原秀男教授の追悼号である．自分はある委員会で石原先生と一緒に仕事をしていたが，石原先生は思考が速く仕事の進め方も効率的であり，その気さくな人柄も相まって自分はとても快適に仕事をすることができた．学部の中でも最も尊敬する先生の一人であったが，そのような方が若くして亡くなってしまったことは痛恨の極みである．謹んで哀悼の意を表すとともに，今後は石原先生の分まで学部の運営と教育に尽力していきたい．

謝辞

本稿の執筆にあたり，有益な助言をいただいた東京農工大学の宮代隆平氏，筑波大学の佐藤俊樹氏に感謝いたします．

[9] http://zimpl.zib.de/download/zimpl.pdf
[10] http://www.tuat.ac.jp/~miya/ipmemo.html

参考文献

[1] Achterberg, T. (2009). SCIP: Solving constraint integer programs. *Mathematical Programming Computation, 1*, 1–41.

[2] Bartlett, A., Chartier, T. P., Langville, A. N., & Rankin, T. D. (2008). An integer programming model for the Sudoku problem. *Journal of Online Mathematics and its Applications, 8*.

[3] Koch, T. (2004). Rapid mathematical programming. ZIB Report 04-58, Konrad-Zuse-Zentrum für Informationstechnik Berlin.

[4] Koch, T. (2005). Rapid mathematical programming or how to solve Sudoku puzzles in a few seconds. ZIB Report 05-51, Konrad-Zuse-Zentrum für Informationstechnik Berlin.

[5] Rubner, Y., Tomasi, C., & Guibas, L. J. (2000). The earth mover's distance as a metric for image retrieval. *International Journal of Computer Vision, 40*, 99–121.

[6] Takano, Y., & Yamamoto, Y. (2010). Metric-preserving reduction of earth mover's distance. *Asia-Pacific Journal of Operational Research, 27*, 39–54.

[7] 藤澤克樹, 後藤順哉, 安井雄一郎 (2011). 『Excel で学ぶ OR』オーム社.

[8] 福島雅夫 (2011). 『新版 数理計画入門』朝倉書店.

[9] 久保幹雄, J.P. ペドロソ, 村松正和, A. レイス (2012). 『あたらしい数理最適化：Python 言語と Gurobi で解く』近代科学社.

[10] 宮代隆平, 松井知己 (2006). 「ここまで解ける整数計画」『システム/制御/情報』 50, 363–368.

[11] 宮代隆平 (2012). 「整数計画ソルバー入門」『オペレーションズ・リサーチ：経営の科学』 57, 183–189.