最近の混合整数計画ソルバーの進展について

藤江　哲也

アルゴリズムの進歩によって混合整数計画ソルバー（MIP ソルバー）の性能が向上を続けており，計算機能力向上との相乗効果によって解決可能な問題の規模が拡大し続けている．本稿では，MIP ソルバーの進歩について概説し，また，MIP ソルバーの利用方法について紹介する．次に，MIP ソルバーの利用例として，python による巡回セールスマン問題を解くプログラムとその実行結果を示す．

キーワード：混合整数計画問題，分枝カット法，ソフトウェア，巡回セールスマン問題

1. はじめに

混合整数計画問題（Mixed Integer Programming problem：MIP 問題）は，一部またはすべての変数に整数制約が付加された線形計画問題である．MIP は十数年前より実用性が向上している分野であり，以前その背景について解説した[7]．その後 MIP ソルバー（MIP 問題を解くソフトウエア）はさらに強力になり，また MIP に関する文献[9][11][20][21]等の著書・解説の増加によって，MIP が有効なツールであるという認識が，最近は定着しつつあるようにも感じられる．

MIP 問題は表現能力が高く[4][10][12][14][26][27]，数多くの NP 困難な組合せ最適化問題を特殊ケースとして含む．その意味で，MIP 問題は汎用的な問題であるということができ，これが実際に解けることには大きなインパクトがある．もちろん，bigM を含む問題など MIP ソルバーが苦手とする問題があり，MIP 問題として定式化できる範囲と MIP ソルバーで解ける範囲には依然として差がある．これは文献[8][16][20]等で指摘されており，つまり実際の問題解決には MIP ソルバーだけで十分とはいえない．しかし MIP ソルバーで解ける範囲が広がり続けているのは間違いなく，問題解決のためにとにかく MIP 問題として定式化し，MIP ソルバーで解いてみる価値は十分に高まっている．そしてそれに伴い，MIP 問題による定式化（モデル化）が必要とされる場面が増えているといえるであろう．実際，MIP ソルバーと同様に，商用・フリーのモデリング言語が充実してきている．そこで本稿では，MIP ソルバーそのものの進展とともに，MIP ソルバーの利用方法の進展にも重点をおき，それぞれについて概要を紹介する．

ふじえ　てつや
兵庫県立大学 大学院経営研究科
〒651-2197 神戸市西区学園西町 8-2-1

2. MIP ソルバー

線形計画（Linear Programming：LP）ベースの分枝限定法/分枝カット法[7][10][14]が MIP 問題に対する標準的な解法であり，本稿で取り上げる MIP ソルバーはこれらの解法に基づいている．商用ソフトウェアとして FICO Xpress Optimization（FICO Corporation），Gurobi Optimizer（Gurobi Optimization），IBM ILOG CPLEX（IBM），LINDO（LINDO Systems），NUOPT（数理システム），SOPT（Saitech, Inc.）等があり，また，表計算ソフト Excel の Solver（Frontline Systems）によって MIP を解くことができる．文献[3]は商用ソフトウェア CPLEX，LINDO，Xpress-MP を例に MIP ソルバーの解説を行っている．文献[15]はフリーソフトウェアの解説であり，ABACUS，BCP，BonsaiG，CBC，GLPK，lp_solve，MINTO，SYMPHONY を取り上げている．ここに挙げたもの以外では，SCIP[1]が有力な MIP ソルバーとして注目されている．以上の MIP ソルバーの情報は，関連の Web ページで見ることができる．

2.1 分枝カット法の概略とソルバーの進展

LP ベースの分枝限定法/分枝カット法の概略を次の例題で説明する．

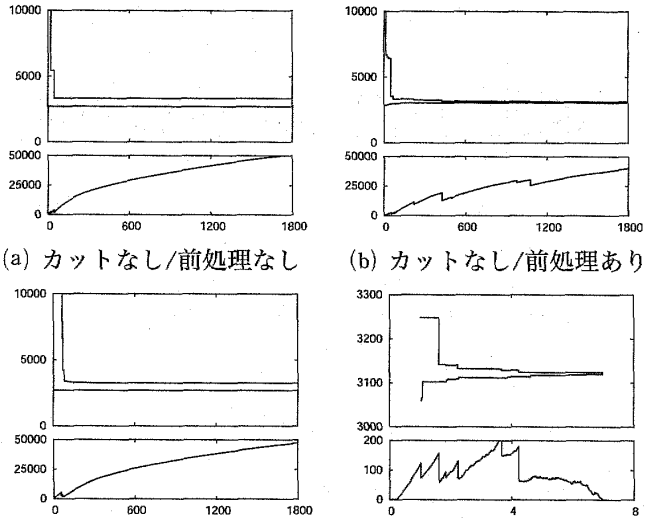
$$
\begin{aligned}
\text{(P)}\quad &\text{最小化 } z=-2x_1-3x_2\\
&\text{条件 } 2x_1+x_2\leqq 10\\
&\qquad 3x_1+6x_2\leqq 40\\
&\qquad x_1,x_2\geqq 0\\
&\qquad x_1,\ x_2:\text{整数}
\end{aligned}
$$

LP ベースの分枝限定法では，まず整数制約を除いた LP 問題（$\bar{\text{P}}$）を解く．（$\bar{\text{P}}$）の最適解は $(x_1, x_2)=$

(20/9, 50/9)，最適値は $\bar{z}=-190/9$ である．$\bar{z}$ は(P)の最適値以下になり，すなわち下界値を与える．一方，(P)の任意の実行可能解の目的関数値は上界値を与える．($\bar{\mathrm{P}}$)の最適解が整数であれば，それは(P)の最適解でもあるので終了する．ここではこれに該当しないため，整数値でない変数，例えば x_1 を選び，(P)に $x_1 \leqq \lfloor 20/9 \rfloor = 2$ を追加した問題を(P_1)，および，(P)に $x_1 \geqq \lceil 20/9 \rceil = 3$ を追加した問題を(P_2)として(P)を分割する．(P_1)と(P_2)は子問題とよばれ，これらに対して同様の操作を繰り返す（x_1 は分枝変数とよばれる）．実行が進むにつれて，上界値および下界値は最適値に近づく．

分枝カット法では，カット（切除平面）とよばれる不等式，すなわち，(P)の実行可能解は満たすが($\bar{\mathrm{P}}$)の最適解は満たさない不等式を追加する．例えば，$5x_1+3x_2 \leqq 27$ はカットであり，これを加えた($\bar{\mathrm{P}}$)の新しい最適解は $(x_1, x_2)=(2, 17/3)$, $\bar{z}=-21$ となって下界値が上昇する（なお，(P)の最適解は $(x_1, x_2)=(1, 6)$, $z=-20$ である）．

分枝限定法/分枝カット法の性能を向上させるには，上界値・下界値を急速に最適値に近づける工夫や，無駄な子問題の生成を避ける工夫などが必要である．上界値を下げる工夫としてヒューリスティック解法の開発，下界値を上げる工夫としてカット生成，分枝変数選択，未処理の子問題選択などが挙げられる．また最近は，前処理（不必要な変数や制約の除去，変数の上下限や制約式の強化等の処理）[19]，解の対称性の検出[17]，制約プログラミングとの融合[1]等様々な処理が加えられている（最近の解説として文献[1][6][9][21]がある）．そして，このような処理を加えることにより，しばしば劇的に効率が向上する．一例として，GNU GLPK（version 4.44）によって p2756（MIPLIB[2]の一問）を解いた結果を示す．p2756 は，変数の数 2756（すべて 0-1 変数），制約数 755 の問題である．GNU GLPK（version 4.44）には，前処理，カット（Gomory's mixed integer cut, mixed integer rounding cut, cover cut, clique cut），ヒューリスティック解法（feasibility pump）が実装されている．図1は CPU : Intel Core 2 Duo E6600（2.40 GHz），RAM : 2 GB の計算機を使って，30 分タイムアウトで実行した結果を示している．それぞれ横軸を実行時間（秒）とし，図1(a)-(d)の各上部は上界値と下界値の時間変化を，下部は未処理の子問題数の時間変化を表示している．ヒューリスティック解法による効果は見られず，ここではヒューリスティック解法を実行しない結果を示している．また，カットは Gomory's mixed integer cut のみを加えた．図1(a)-(c)が示すように，前処理とカットを両方加えない場合 30 分で最適解を得ることができず，未処理の子問題数が 50000 前後まで増加した．ところが，両方加えると約 7 秒で最適解を得ることができた（図1(d)）．また，図1(b)では下界値の上昇が見られるが，(a)および(c)では見られない．どの処理が有効に働くのか，あるいはどの処理の組合せが有効であるのかは，定式化や実装されているアルゴリズムによって異なるが，p2756 には前処理が有効に働いたということができる．それと同時に，オリジナルの p2756 にはカットがさほど有効ではなかったが，前処理を施された問題（新しい MIP 問題）にはカットが非常に効果的であったことがわかる．このような処理（工夫）を加えることにより，計算時間が大幅に短縮されたり未解決の MIP 問題が解けるようになったわけである．



(a) カットなし/前処理なし　(b) カットなし/前処理あり

(c) カットあり/前処理なし　(d) カットあり/前処理あり

図 1　p2756 に対する結果（(a)-(d)：上下界の時間変化（上）と未処理子問題数の時間変化（下））

2.2　MIP ソルバーの利用方法

MIP 問題（あるいは MIP 問題として定式化される問題）をコンピュータで解く方法として，次に挙げるアプローチが考えられる．

(I) MIP 問題を解く部分を含め，ソフトウエアを一から作成する．

(II) MIP ソルバーが API（Application Programming Interface）として与えられ，それを利用

したソフトウェアを作成する．

(Ⅲ) 実行ファイルとなっているMIPソルバーを利用する．

(Ⅲ') GUIアプリケーションを利用する．

番号が小さいほど，プログラミングやアルゴリズムに関する知識が必要である．問題の構造を生かしたアルゴリズムを一から実装する場合などに(Ⅰ)のアプローチがとられる．ただこの場合でも，LPソルバーをAPIとして利用することが多く見られる．(Ⅱ)のアプローチは，MIPソルバーが進化した結果意味が出てきたといえる．LPソルバーおよびMIPソルバーは，多くの場合C/C++でAPIとして利用可能である．また，ソルバーによっては，C#/Java/python/matlab/Excel等でも利用可能である．次節ではpythonによるプログラム作成例を紹介する．

(Ⅲ)のアプローチでは，入力ファイルを用意しソルバーを実行するのが一般的である．入力ファイルの形式としては，LP-format，MPS format，モデリング言語の利用が挙げられる．図2は，前節の例題(P)をCPLEX LP-formatで記述したものである．そして図3はMPS formatによる記述である．MPS formatは1960年代にIBMによって導入された形式で，現在も標準的に使われているが，可読性はLP-formatの方が高いといえる．図4はGNU MathProg（GLPK packageに含まれている）による記述である．図4ではモデル部とデータ部が一つのファイルに書かれているが，これを2つのファイルに切り離すこともできる．商用のモデリング言語としてAIMMS（Paragon Decision Technology），AMPL（AMPL Optimization），GAMS（GAMS Development Corporation），LINGO（LINDO Systems），MPL（Maximal Software），Simple（数理システム）等があり，フリーではGNU MathProg，ZIMPL等がある．

ソルバーの実行は，コマンドライン（Windowsではコマンドプロンプト）でコマンドをタイプして行うのが一般的であるが，(Ⅲ')では問題の記述/実行/解の出力が容易に行えるようになる．(Ⅲ')については，商用ソフトウェアでは充実しているが，フリーでも例えばGLPKのIDE（Integrated Development Environment）としてGUSEK（GLPK Under Scite Extended Kit）がある．Excelを使った実行も(Ⅲ')に分類されるであろう[25]．

```
minimize
 obj: -2 x1 - 3 x2
subject to
 r1:   2 x1 +   x2 <= 10
 r2:   3 x1 + 6 x2 <= 40
general
  x1 x2
end
```

図2 CPLEX LP-formatによる記述

```
NAME          sample
ROWS
 N  obj
 L  r1
 L  r2
COLUMNS
    MO000000  'MARKER'                 'INTORG'
    x1        obj                 -2   r1                  2
    x1        r2                   3
    x2        obj                 -3   r1                  1
    x2        r2                   6
    MO000000  'MARKER'                 'INTEND'
RHS
    RHS       r1                  10   r2                 40
BOUNDS
 PL BOUND     x1
 PL BOUND     x2
ENDATA
```

図3 MPS formatによる記述

```
param n, integer;
param m, integer;
param c{j in 1..n};
param a{i in 1..m, j in 1..n};
param b{i in 1..m};
var x{j in 1..n}, integer >= 0;

minimize z: sum{j in 1..n} c[j]*x[j];
s.t. con{i in 1..m}: sum{j in 1..n} a[i,j]*
x[j] <= b[i];

data;
param n := 2;
param m := 2;
param c := 1 -2, 2 -3;
param a: 1  2 :=
      1  2  1
      2  3  6;
param b := 1 10, 2 40;
end;
```

図4 GNU MathProgによる記述

3. 巡回セールスマン問題

前節(Ⅱ)のアプローチによる一例として，文献[23]で提案されている，非対称巡回セールスマン問題（Asymmetric Traveling Salesman Problem：ATSP）

の解法を紹介する（TSPの詳細は文献[28]を参照されたい）．頂点集合（都市の集合）を$V=\{1, ..., n\}$とし，頂点iからjへの費用をc_{ij}とする．ATSPは完全有向グラフ$D=(V, A)$において，総費用最小の巡回路（tour），すなわち，すべての頂点（都市）をちょうど一度ずつ通る閉路を求めることに等しい．

D中の閉路を部分巡回路（subtour）とよぶ．このときATSPは次のように定式化することができる．

(ATSP)

$$\text{最小化} \quad z=\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$\text{条　件} \quad \sum_{i=1}^{n} x_{ij}=1 \quad (j=1, ..., n) \tag{1}$$

$$\sum_{j=1}^{n} x_{ij}=1 \quad (i=1, ..., n) \tag{2}$$

$$\{(i, j) : x_{ij}=1\}\text{は2つ以上の部分巡回路を含まない} \tag{3}$$

$$x_{ij}=0\text{または}1 \quad (i, j=1, ..., n) \tag{4}$$

(1)-(4)を満たす実行可能解(x_{ij})に対して$\{(i, j) : x_{ij}=1\}$は巡回路であり逆も成り立つ．(3)は部分巡回路除去制約とよばれている．(3)には様々な表現方法が知られている．Dantzig-Fulkerson-Johnson[5]による先駆的な論文では次の表現が与えられている：

$$\sum_{i\in S}\sum_{j\in S} x_{ij}\leqq|S|-1 \ (S\subseteq V\backslash\{1\}, |S|\geqq 2). \tag{5}$$

これらは巡回セールスマン多面体のファセットを定める強い不等式である．一方，不等式の数が$O(2^n)$と膨大であり，(5)を書き下して直接MIPソルバーに解かせることは現実的でない．そこで，新たな変数を導入して不等式の数を抑える方法が多く提案されている[22]．文献[23]では，Miller-Tucker-Zemlin[18]による表現を取り上げている：

$$u_i-u_j+(n-1)x_{ij}\leqq n-2 \ (i, j=2, ..., n). \tag{6}$$

$u_i(i=2, ..., n)$が新たに導入された変数である．(6)が満たされているとき，頂点1を含まない部分巡回路$C=\{(i_1, i_2), ..., (i_{p-1}, i_p), (i_p, i_1)\}$の各要素に関する(6)を足し合わせると

$$\sum_{(i,j)\in C} x_{ij}\leqq\frac{n-2}{n-1}|C|=\left(1-\frac{1}{n-1}\right)|C| \tag{7}$$

が得られる．よって(6)は部分巡回路除去制約になっている．(6)は$O(n)$個の新しい変数を導入し，不等式の数を$O(n^2)$に抑えている．ただし，(5)では頂点部分集合S内の任意のペア(i, j)に関する変数x_{ij}が現れるのに対し，(7)では閉路C内の(i, j)のみが現れる．これに加えて，(7)より推測されるように，(6)は(5)よりも弱い不等式であることが知られている[22]．つまり，いずれも正しい定式化を与えるが，整数制約を除去したLP問題が異なる．そして弱い定式化では最適値から遠く離れた下界値しか得られず，分枝限定法/分枝カット法の実行時間に影響を与える．

文献[23]では，(5)を使うATSPの強い定式化と(6)を使う弱い定式化を比較する練習問題として，次のように2つの定式化を組み合わせる方法を提案した（Kはパラメータである）．

Step 1. $\mathcal{S}=\emptyset, k=0$とする．

Step 2. $k\leqq K$である間，次を実行する．

Step 2.1（ATSP）の(3)を

$$\sum_{i\in S}\sum_{j\in S} x_{ij}\leqq|S|-1 \ (S\in\mathcal{S})$$

に置き換えたMIP問題を解く．

Step 2.2 最適解の部分巡回路数が1なら終了．さもなければ，各部分巡回路の頂点集合を$\mathcal{S}$に加え，$k=k+1$とする．

Step 3. (6)を追加して（ATSP）の最適解を得る．

これはMIP問題を繰り返し解く方法であり，最終的に(6)を追加して解く(Step 3)ため，必ず最適解が得られる．この方法の利点として，Step 2.1で部分巡回路集合を容易に見つけられることが挙げられる．図5にpythonコードを示す（文献[23]ではmatlabコードが与えられている）．(2)より，各$i\in V$は$x_{ij}=1$となるjを唯一つ持つ．図5では，このjをdest[i]としている．なお，Step 2.2は文献[23]と弱干異なる．

次にpythonでCPLEX（version 12.2）のAPIを利用する例を図6に示す．

この方法でTSPLIB[24]の問題att48（$n=48$）を

```
. . . (略) . . .
V = range(n)
F = []
while V:
    S = []
    start = V[0]
    S.append(start)
    V.remove(start)
    next = start
    while True:
        next = dest[next]
        if next == start: break
        S.append(next)
        V.remove(next)
    F.append(S)
return F
```

図5　部分巡回路集合を計算するpythonコード

```
. . . . .

model = cplex.Cplex()

. . . . .

for i in range(n):
    model.variables.add(
      obj = dist[i],
      lb = [0] * n,
      ub = [1] * n,
      types = ["B"] * n,
      names = ["x" + str(i) + "_" + str(j)])

. . . . .

for i in range(n):
    con = cplex.SparsePair(
            ind = [indx[i][j] for j in range(n)],
            val = [1.0] * (n-1))
    model.linear_constraints.add(
      val lin_expr = [con],
      senses = ["E"],
      rhs = [1],
      names = ["ACrow_" + str(i)])

. . . . .

model.solve()

tlen = model.solution.get_objective_value()
print 'Tour Length = ' + tlen

. . . . .
```

図6 CPLEX API を利用する python コード

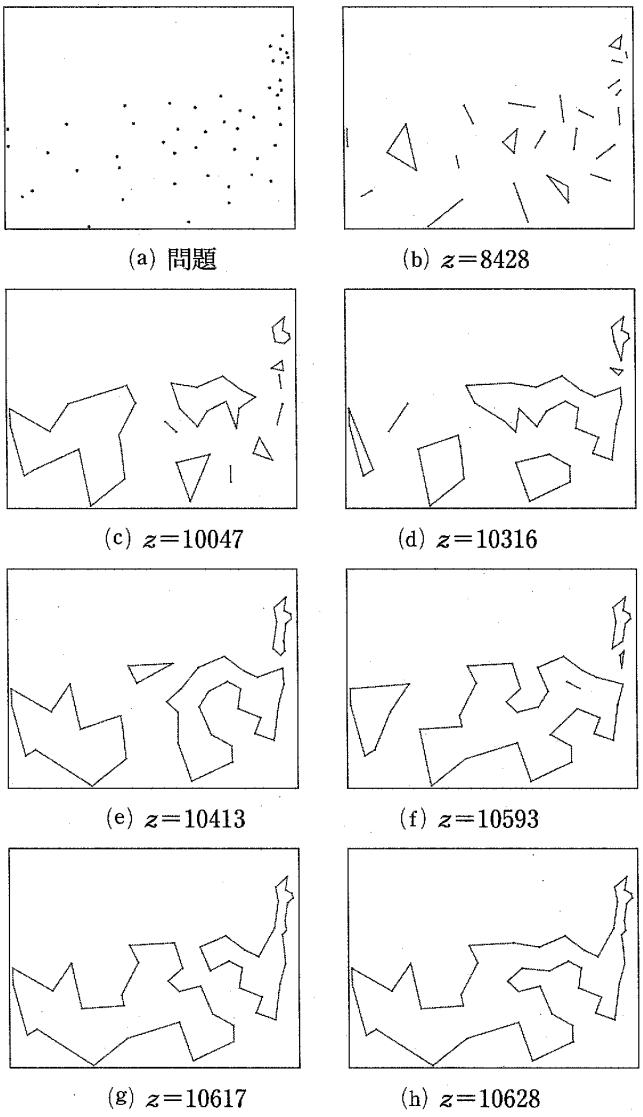


図7 att48 に対する実行結果

解いた結果を図7に示す（z は目的関数値である）．K は十分大きい値に設定している．図の描画は Matplotlib を利用した．この問題は，図7(a)のように，V が平面上の点として与えられ，c_{ij} は i, j 間の距離を整数で表したものである．$c_{ij}=c_{ji}$ であるため att48 は対称 TSP 問題である．図7(b)-(h)は実行の過程を示している．まず，(ATSP) から(3)を除いた MIP 問題（これは割当問題になる[12]）を解いた結果，図7(b)が得られる．長さ2の部分巡回路 $\{(i_1, i_2), (i_2, i_1)\}$ は i_1 と i_2 を結ぶ直線で表示されている．目的関数値は z=8428 で22の部分巡回路がある．次に，これら部分巡回路に関する(5)を追加した MIP 問題を解く．その結果が図7(c)である．これを続けていくと，最終的に図7(h)の最適解が得られた．トータルの計算時間は，図1と同じ計算機で1.41秒であった．一方，(6)を使う定式化では，LP 問題の目的関数値は $\bar{z}$=8490.11 であり，30分タイムアウトで最適解を得ることはできなかった．このように，定式化によって計算結果が大きく異なることがわかる．他の非対称の問題でも同様であった．Step 1, 2 によって gr120（n=120，対称 TSP）を3.38秒で解くことができたが，n が300程度以上になると，効率よく最適解に収束することはなかった．

python API が利用できるようになったことにより，プログラミングで MIP ソルバーを利用するハードルが低くなったと感じる（文献[13]においてもこの点が強調されている）．また，ここでは変数の数が数千～一万を超える MIP 問題を繰り返し，かつ，高速に解いており，MIP ソルバーの進歩を示す一例となっている．

4. おわりに

本稿で紹介したように，MIP ソルバーは使いやすいツールになったといえる．本稿が MIP モデルを試してみるきっかけとなれば幸いである．なお，MIP モデルに興味をもたれた読者には文献[20]の一読を勧めたい．

参考文献

[1] T. Achterberg, *Constraint Integer Programming*, Ph. D. Thesis, Technische Universität Berlin, 2007.

[2] T. Achterberg, T. Koch and A. Martin, "MIPLIB 2003," *Operations Research Letters*, **34** (2006), 361-372.

[3] A. Atamtürk and M.W.P. Savelsbergh, "Integer-Programming Software Systems," *Annals of Operations Research*, **140** (2005), 67-124.

[4] D.S. Chen, R.G. Baston and Y. Dang, *Applied Integer Programing : Modeling and Solution*, Wiley, 2010.

[5] G.B. Dantzig, D.R. Fulkerson and S.M. Johnson, "Solution of a Large Scale Traveling Salesman Problem," *Operations Research*, **2** (1954), 393-410.

[6] M. Fischetti, A. Lodi and D. Salvagnin, "Just MIP it!," in *Metaheuristics : Hybridizing Metaheuristics and Mathematical Programming*, V. Maniezzo et al., eds., Springer, 2010.

[7] 藤江哲也，「整数計画問題に対する分枝カット法とカットの理論」，『オペレーションズ・リサーチ』，**48** (2003), 935-940.

[8] 茨木俊秀，「「問題解決エンジン」群とモデリング」，『オペレーションズ・リサーチ』，**50** (2005), 229-232.

[9] M. Jünger et al. (eds.), *50 Years of Integer Programming*, Springer, 2010.

[10] 今野浩，『整数計画法』，産業図書，1981.

[11] 今野浩，『役に立つ一次式—整数計画法「きまぐれな王女」の 50 年』，日本評論社，2005.

[12] 今野浩，鈴木久敏（編集），『整数計画法と組合せ最適化』，日科技連出版社，1982.

[13] 久保幹雄，「サプライ・チェインにおける様々な最適化問題を解くための統一言語」，OR 学会中部支部講演会，2010 年 9 月 22 日.

[14] 久保幹雄，田村明久，松井知己（編集），『応用数理計画ハンドブック』，朝倉書店，2002.

[15] J.T. Linderoth and T.K. Ralphs, "Noncommercial Software for Mixed-Integer Linear Programming," in *Integer Programming—Theory and Practice*, J.K. Karlof, ed., Taylor & Francis, 2006.

[16] A. Lodi, "Mixed Integer Programming Computation," in Ref. [9], 619-645, 2010.

[17] F. Margot, "Symmetry in Integer Linear Programming," in Ref. [9], 647-686, 2010.

[18] C. Miller, A. Tucker and R. Zemlin, "Integer Programming Formulations and Traveling Salesman Problems," *Journal of the Association for Computing Machinery*, **7** (1960), 326-329.

[19] W.P. Savelsbergh, "Preprocessing and Probing Techniques for Mixed Integer Programming Problems," *ORSA Journal on Computing*, **6** (1994), 445-454.

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

[21] 宮代隆平，「ここまで解ける整数計画—近年の発展—」，『第 20 回 RAMP シンポジウム報告集』，1-21, 2008.

[22] T. Öncan, İ. K. Altınel and G. Laporte, "A Comparative Analysis of Several Asymmetric Traveling Salesman Problem Formulations," *Computers & Operations Research*, **36** (2009), 637-654.

[23] G. Pataki, "Teaching Integer Programming Formulations Using the Traveling Salesman Problem," *SIAM Review*, **45** (2003), 116-123.

[24] G. Reinelt, "TSPLIB—A Traveling Salesman Problem Library," *ORSA Journal on Computing*, **3** (1991), 376-384.

[25] 高井英造，真鍋龍太郎（編集），『問題解決のためのオペレーションズ・リサーチ入門：Excel の活用と実務的例題』，日本評論社，2000.

[26] H.P. Williams, *Model Building in Mathematical Programming*, John Wiley and Sons, 1993. 前田英次郎監訳，小林英三訳，『数理計画モデルの作成法』，産業図書，1995.

[27] L. Wolsey, *Integer Programming*, Wiley, 1998.

[28] 山本芳嗣，久保幹雄，『巡回セールスマン問題への招待』，朝倉書店，1997.