数理計画ソフト初体験

柳浦　睦憲

1. はじめに

I教授:「今年は科研費が結構あるやろ．うちにも数理計画ソフトが1つあると便利でいいんじゃないかと思うんやけど，柳浦君ちょっと調べておいてくれないかなぁ」

筆者:「ははー」

というわけで，急遽数理計画ソフトをいろいろ使ってみることになりました．この手のソフトは初体験だったのですが，使ってみてまず感じたのが，「おおっ，何と手軽!」ということです．これはひとえにモデル記述言語なるもののおかげなのですが，これを使うと，集合や数式を用いた入力が可能になるため，わざわざ行列の形に係数をならべ直すなどの必要がなく，人間が理解しやすい自然な形で問題を記述することができます．よって，数理計画ソフトを手軽に使えるようになるわけです．

線形計画問題 (LP) に限らず，整数計画問題や非線形計画問題が市販の数理計画ソフトで比較的大きなサイズまで解けるようになってきています．もちろん，整数，および非線形計画問題の場合は効率的に解けるケースが限られてくるので注意が必要ですが．「解きたい数理計画問題が手元にあるのだが，どうも数理計画法というのは難しそうで使う気が...」とか，「数理計画問題を解いてくれるソフトがあることは知っているのだが，どうも教科書に書いてあるような行列の形に定式化するのもめんどくさそうだし...」などと思っている方々が，この記事を読んで，「これなら使ってみようか」という気になって下されば幸いです．

以下では，モデル記述言語を利用した入力方法の一例と簡単な計算例を紹介します．数理計画ソフトを初めて使ってみた程度の筆者が解説記事を書くのは少々妙な感じですが，少なくとも，初心者でもすぐに使えるんだということはご理解いただけると思います．なお，モデル記述言語をすでに利用なさっている方にはとくに目新しい情報はないと思いますがご了承ください．

2. モデル記述言語

モデル記述言語に対応した数理計画ソフトとしては，例えば，SOPT (AMPL)，CPLEX (AMPL)，XPRESS-MP (MP-MODEL)，NUOPT (SIMPLE)，LINDO，GINO などがあります (括弧内はモデル記述言語名)．また，表計算ソフト Excel には独自のソルバーが付いています．ILOG や CHIP などは，制約充足問題を基本とするソフトですが，数理計画法の導入が予定されているようです．これらには論理的な制約を簡単に記述できるような様々な表現法が用意されています．モデル記述言語の詳しい解説は，もちろんソフトを購入すればセットで付いてきますが，例えば，以前の OR 学会誌に LINDO と GINO の解説 [9] が，また，問題のモデル化を含め，さらに詳しい解説書として [1, 11] などがありますのでご参照ください．また，毎年春と秋に行われる OR 学会の研究発表会場では，XPRESS-MP や NUOPT などのデモがあります．ソフトを販売している会社の連絡先は例えば最近の OR 学会誌の広告などに載っています．とりあえずちょっと試してみてから購入を検討したいという方のために，サンプル版を有効期限付きで貸してくれる会社もあります．少々古いですが，[7] にいろいろなソフトの紹介がありますので，こちらもご参照下さい．また，モデル記述言語に対応しているかどうかは確認していませんが，“タダ” のソフトも数多くあります．久保先生の記事 [6] や松井先生のホームページ[1]などをご覧ください．

上述したようなモデル記述言語は，それぞれに特色があり，また，文法も多少異なりますが，以下に述べるような基本的な記述能力は大抵どれにでも備わってい

やぎうら　むつのり　京都大学工学研究科
〒 606-01 京都市左京区吉田本町

[1] http://www.misojiro.t.u-tokyo.ac.jp/~tomomi/

ます. 以下では, SOPT+AMPL(AMPL については例えば [1] 参照) を用いた入力例を紹介します.

3. ナップサック問題

前回の特集「ユーザのための数理計画入門」の冒頭の茨木先生の記事 [3] で紹介されたナップサック問題の入力を考えてみましょう. 問題は, 予算 2000 円でコーヒー, 紅茶, 日本茶, ジュースをそれぞれ x_1, x_2, x_3, x_4 単位 (g, ml など) 購入して, 満足度を最大にするというもので, データは,

	コーヒー	紅茶	日本茶	ジュース
	x_1	x_2	x_3	x_4
嗜好係数 c_j:	2.0	2.3	2.5	0.3
値段 a_j:	2.0	2.5	3.0	0.5
最大必要量:	500g	300g	300g	1000ml

問題は

$$\begin{aligned} &\text{満足度:} && 2.0x_1 + 2.3x_2 + 2.5x_3 + 0.3x_4 \rightarrow \text{最大} \\ &\text{制約条件:} && 2.0x_1 + 2.5x_2 + 3.0x_3 + 0.5x_4 \leq 2000 \\ & && 0 \leq x_1 \leq 500,\ 0 \leq x_2 \leq 300 \\ & && 0 \leq x_3 \leq 300,\ 0 \leq x_4 \leq 1000 \end{aligned}$$

と与えられます (通常は整数制約が入るが簡単のため省略). これを AMPL で記述するわけですが, 以下では実際に入力する部分をタイプライターフォントで書くことにします. それ以外はコンピュータの出力です. まず, AMPL を立ち上げるために, コンピュータのコマンド入力プロンプトで,

```
prompt% ampl
```

と入力します. すると,

```
ampl:
```

という AMPL のプロンプトが出て入力待ち状態になります. ここで問題を 1 行 1 行入力していって, 最後に `solve` と入力すればソルバーが勝手に答えを出してくれます.

```
ampl: var x1 >= 0, <= 500;
ampl: var x2 >= 0, <= 300;
ampl: var x3 >= 0, <= 300;
ampl: var x4 >= 0, <= 1000;
ampl: maximize utility:
ampl: 2.0*x1 + 2.3*x2 + 2.5*x3 + 0.3*x4;
ampl: subject to budget:
ampl: 2.0*x1 + 2.5*x2 + 3.0*x3 + 0.5*x4
ampl: <= 2000;
ampl: solve;
    optimal solution found
    (obj = 1.8983e+03, time = 0.01 sec)
ampl: display utility;
    utility = 1898.3
ampl: display x1,x2,x3,x4;
    x1 = 500, x2 = 300, x3 = 83.3, x4 = 0
```

はじめの 4 行は変数の定義, および各変数の上下限制約を与えます. 次の 2 行は utility と名づけた目的関数の最大化が目標であることを指定します. 目的関数や制約式 1 本の定義はセミコロン「;」で区切られるので, この例のように間に改行が入ってもかまいません. 最後は予算制約です. `solve` という命令によりソルバーが稼動し, 最適解が求まります. `display` という命令により, 最適値や最適解など様々な情報を見ることができます. この例では, 求まった最適解は

$$\begin{aligned} x_1 &= 500 \quad \text{(コーヒー)} \\ x_2 &= 300 \quad \text{(紅茶)} \\ x_3 &= 83.3 \quad \text{(日本茶)} \\ x_4 &= 0 \quad \text{(ジュース)}, \end{aligned}$$

最適値は 1898.3 となっており, 得られた解が [3] で求めたものと一致していることが確認できます. C 言語の printf 文と同様の表記も受け付けますので, 出力を都合の良いフォーマットで行うこともできます. このように, 対話的なインターフェイスが用意されているので, 問題や解法のパラメータを変化させながら満足解が得られるまで対話的に作業を進めることが可能です.

通常, 問題を手で入力するのは大変なので, 何らかのプログラムを用いて問題を記述したファイルを生成し, それをソルバーに読ませるなどの方法をとります. これを行うには, 以下のようなファイルを, 例えば knapsack.mod と名前をつけて保存しておきます.

```
# ナップサック問題 (ファイル名 : knapsack.mod)
# 変数定義と上下限制約
var x1 >= 0, <=  500; # コーヒー
var x2 >= 0, <=  300; # 紅茶
var x3 >= 0, <=  300; # 日本茶
var x4 >= 0, <= 1000; # ジュース
```

```
# 目的関数 (満足度 utility の最大化)
maximize utility:
2.0*x1 + 2.3*x2 + 2.5*x3 + 0.3*x4;
# 制約条件 (予算制約)
subject to budget:
2.0*x1 + 2.5*x2 + 3.0*x3 + 0.5*x4 <= 2000;
```

そして, AMPL の中で,

```
ampl: include knapsack.mod;
ampl: solve;
```

とすればよいわけです. なお, 上の例のように,「#」(必ず半角) から改行までの間にはコメントを書くことができます.

4. 一般化割当問題

ナップサック問題の例で簡単なルールはご理解いただけたと思います. しかし, 上の例では, 表現がなじみやすいということの他にはあまりモデル記述言語のメリットが見えません. ここでは, もう少し複雑な一般化割当問題を考えてみましょう. この問題は, ナップサック問題に似ているのですが, 少し拡張した問題になっています. 問題を図 1の例で説明します. 仕事

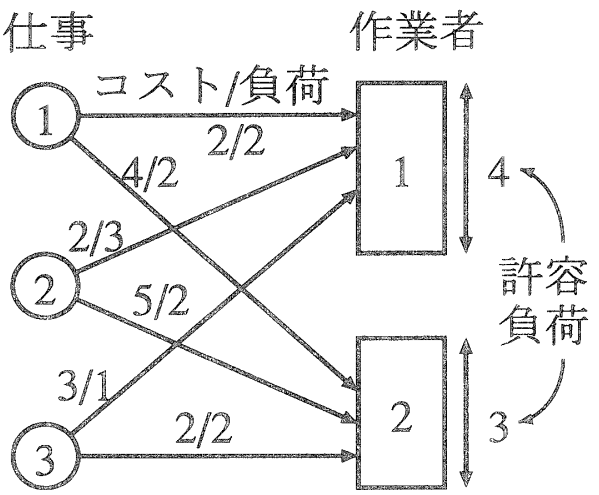

図 1: 一般化割当問題の一例

を作業者に割当てる際, 仕事 1 単位につき枝上の数字分のコスト (賃金など) を要し, また, 負荷 (労働時間など) が作業者にかかるものとします. この時, 各作業者の負荷が許容量を超えないようにした上で, 総コストが最小となる仕事の配分を求めるという問題です. 図 1の問題例は,

$$
\begin{aligned}
\text{目的関数:}\quad & 2y_{11}+4y_{12}+2y_{21} \\
& +5y_{22}+3y_{31}+2y_{32} \rightarrow \text{最小} \\
\text{制約条件:}\quad & 2y_{11}+3y_{21}+1y_{31} \leq 4 \\
& 2y_{12}+2y_{22}+2y_{32} \leq 3 \\
& y_{11}+y_{12}=1 \\
& y_{21}+y_{22}=1 \\
& y_{31}+y_{32}=1 \\
& y_{11} \geq 0, y_{12} \geq 0 \\
& y_{21} \geq 0, y_{22} \geq 0 \\
& y_{31} \geq 0, y_{32} \geq 0
\end{aligned}
\tag{1}
$$

と表せます. 各仕事の総量を 1 として, 変数 y_{ij} は仕事 i を作業者 j に割当てる割合を意味します (通常は整数制約が入るが簡単のため省略). 目的関数は総コスト, 制約条件のはじめの 2 つは作業者 1 と 2 の負荷許容量に対する制約, 次の 3 つは各仕事がすべて割当てられなければならないこと, 最後は割当てる量は負になってはならないことを表します.

説明の都合上, 記号を使ってもう少し一般的に書くと, 仕事数を n, 作業者数を m, 仕事 i を作業者 j に割当てたときのコストを d_{ij}, 負荷を r_{ij}, 作業者 j の負荷許容量を s_j として,

$$
\begin{aligned}
\text{目的関数:}\quad & \sum_{i=1}^{n}\sum_{j=1}^{m} d_{ij}y_{ij} \rightarrow \text{最小} \\
\text{制約条件:}\quad & \sum_{i=1}^{n} r_{ij}y_{ij} \leq s_j,\ j=1,\ldots,m \\
& \sum_{j=1}^{m} y_{ij} = 1,\ i=1,\ldots,n \\
& y_{ij} \geq 0,\ i=1,\ldots,n,\ j=1,\ldots,m
\end{aligned}
$$

と書けます. 上の例題を AMPL で書くと, 例えば,

```
# パラメータ定義
param n; param m;    # 仕事数と作業者数
param d{1..n,1..m}; # 各割当のコスト
param r{1..n,1..m}; # 各割当の必要負荷
param s{1..m};       # 各作業者の許容負荷
# 変数定義と下限
var y{1..n,1..m} >= 0;
# 目的関数
minimize cost:
sum{i in 1..n} sum{j in 1..m} d[i,j]*y[i,j];
# 負荷許容量制約
subject to resource {j in 1..m}:
```

```
sum{i in 1..n} r[i,j]*y[i,j] <= s[j];
# 割当て制約
subject to assign {i in 1..n}:
sum{j in 1..m} y[i,j] = 1;
data; # 以下は問題例のデータ
param n := 3; param m := 2;
param d :  1   2 :=
      1    2   4
      2    2   5
      3    3   2;
param r :  1   2 :=
      1    2   2
      2    3   2
      3    1   2;
param s :=
      1    4
      2    3;
```

のようになります. 前半で問題の一般的な定義を行い, 後半で具体的なデータを入力しています. コメントを含む最初の 5 行で問題を定めるパラメータを定義し, 次の 2 行で変数を定義します. そして, これらの文字を使って目的関数と制約式の定義を行います. AMPL の中では 1..n というのは, 集合 $\{1,2,\ldots,n\}$ を意味し, sum{i in 1..n} は$\sum_{i=1}^{n}$や$\sum_{i\in\{1,\ldots,n\}}$と同じ意味になります. また, subject to resource {j in 1..m}:は, 各 $j=1,2,\ldots,m$ に対してこの制約が定義されることを意味します. パラメータ d と r の入力部分では, 第1行目と第1列目は行と列の番号になっています. 例えば, d[1,1]=2, d[1,2]=4, d[2,1]=2, d[2,2]=5...などという情報がまとめて書いてあるわけです. パラメータ s の第1列目も同様です.

データ入力の部分が, 図 1の枝に付いている数値をそのまま順序良く並べただけの, 直感的にわかりやすい形でコンパクトに入力されており, (1) の表記をそのまま書き下すよりも少々賢い入力法になっていることがお分かりいただけると思います. また, このような書き方をしておくと, 前半と後半を別々のファイルに分けておき, 新たな例題を入力する際には後半のデータ部分のみを生成して入力を行うことができるので, 便利です. さらに, このような入力方法をとると, 書かれている数字の意味がすぐに分かるので, 修正やデバッグが容易であるというメリットもあります.

さて, 次に, モデル記述言語がない場合の入力がどれだけ大変か(ちょっと大袈裟かも)を簡単に説明したいと思います. 線形計画問題は一般的には

$$
\begin{aligned}
&\text{目的関数:} && \sum_{j=1}^{N} c_j x_j \rightarrow \text{最小} \\
&\text{制約条件:} && \sum_{j=1}^{N} a_{ij} x_j \le b_i,\ i=1,\ldots,M_1 \qquad (2)\\
& && \sum_{j=1}^{N} a_{ij} x_j = b_i,\ i=M_1+1,\ldots,M \\
& && x_j \ge 0,\ j=1,\ldots,N
\end{aligned}
$$

と書けますが, モデル記述言語を用いない場合は, 通常, この形に問題を記述した上で, a_{ij}, b_i, および c_j のゼロでないもの (非ゼロ要素) を指定された順序で添字とともに入力していきます. 一般化割当問題の場合, 変数 y_{ij}には 2 つの添字がついているので,

$$y_{ij} \leftrightarrow x_k \ (k := m(i-1)+j)$$

のような対応づけを行い, 問題が (2) の形になるように添字の変換を行う必要があります. これは, 別に難しいことではないのですが, 良く考えながら慎重に作業を進めないと, 思わぬバグの原因になりがちな作業です. 上の一般化割当問題を (2) の形に書き直すと,

$$
\begin{aligned}
&\text{目的関数:} && 2x_1+4x_2+2x_3 \\
& && +5x_4+3x_5+2x_6 \rightarrow \text{最小} \\
&\text{制約条件:} && 2x_1+3x_3+1x_5 \le 4 \\
& && 2x_2+2x_4+2x_6 \le 3 \\
& && x_1+x_2=1 \\
& && x_3+x_4=1 \\
& && x_5+x_6=1 \\
& && x_1 \ge 0, x_2 \ge 0 \\
& && x_3 \ge 0, x_4 \ge 0 \\
& && x_5 \ge 0, x_6 \ge 0
\end{aligned}
$$

となりますから,

$$
\begin{aligned}
&N=6, \quad M_1=2, \quad M=5 \\
&c_1=2, \quad c_2=4, \quad c_3=2 \\
&c_4=5, \quad c_5=3, \quad c_6=2 \\
&a_{11}=2, \quad a_{13}=3, \quad a_{15}=1 \\
&a_{22}=2, \quad a_{24}=2, \quad a_{26}=2 \\
&a_{31}=1, \quad a_{32}=1 \\
&a_{43}=1, \quad a_{44}=1 \\
&a_{55}=1, \quad a_{56}=1 \\
&b_1=4, \quad b_2=3, \quad b_3=1 \\
&b_4=1, \quad b_5=1
\end{aligned}
$$

という情報を入力するわけです．例えば，[4] (p. 30) のフォーマットに従うと，この入力は以下のようになります．(この本には，線形計画問題に対するシンプレックス法をはじめとする様々なアルゴリズムの解説があり，その中で紹介されたプログラムの入ったフロッピーディスクも安価で販売されているので，そういう意味では手軽に利用できるのですが，モデル記述言語とは対応しておらず，入力は少々面倒になります．また，著者曰く「分かりやすさを重視して教育用に開発した」プログラムなので，大きいサイズの問題を解くのには適さないようです．)

```
6 5     3 2 1   3 1   2 4
1 1 2   4 3 1   4 1   3 2
1 3 3   4 4 1   5 1   4 5
1 5 1   5 5 1   0 0   5 3
2 2 2   5 6 1   1 1   6 2
2 4 2   0 0 0   2 1   0 0
2 6 2   1 4     0 0   0 0
3 1 1   2 3     1 2   0 0
```

実際の入力ファイルはこの数字の列が1列に並ぶのですが，場所の都合上，4列に分けて書いてあります．フォーマットを一応簡単に説明しておきますと，一行目は変数の数と制約の数，2行目以降「0 0 0」までが，「i, j, a_{ij}」，次の「0 0」までが，「i, b_i」，次の「0 0」までが，各行の制約タイプ (第i番目の制約が不等式 ($\leq$) の場合は「i, 1」と書き，等式ならば記述不要)，次の「0 0」までが，「j, c_j」，最後に各変数の上下限制約を書きます (今回は不要なので「0 0」のみを入力)．これらの数字の並びを見てもピンと来ず，一目では意味が分からない上に，このような形式に従って入力を行うことが少々やっかいな作業であることが分かると思います．

5. 一般化割当問題に対する計算例

それでは，入力した問題が市販のソフトでどの程度の時間で解けるかを見てみたいと思います．使用した計算機はSun Ultra 1 Model 170E，利用したソフトはサイテック社のSOPTです．一般化割当問題をSOPTに入力したときの計算時間を表1に示します．表中，GAPは一般化割当問題 (generalized assignment problem) を意味します．作業者数50，仕事数2000の場合は，線形計画問題の変数の数は10万，制約を記述するa_{ij}の非ゼロ要素数は20万にもなり，かなり大規模な問題になりますが，普通のワークステーションで2分程度で解けています．計算時間は，通常，変数の数よりもむしろ非ゼロ要素数に依存します．また，線形計画問題の場合はこのように大規模な問題が効率的に解けるのですが，整数性の制約や非線形の式が加わると，いつもこのように効率よく問題が解けるわけではありませんので，ご注意ください．いずれにしても，このような大規模な問題を効率よく解いてくれるソフトが意外と手軽に使えることは大変心強いことと思うのですが，いかがでしょうか．

表 1: 一般化割当問題に対する計算例

GAPのサイズ		線形計画問題のサイズ			計算
作業者	仕事	変数	制約	非ゼロ要素	時間(秒)
5	1000	5000	1005	10000	3.31
5	2000	10000	2005	20000	10.00
10	1000	10000	1010	20000	7.02
10	2000	20000	2010	40000	17.44
20	1000	20000	1020	40000	16.58
20	2000	40000	2020	80000	41.00
30	1000	30000	1030	60000	26.58
30	2000	60000	2030	120000	61.86
40	1000	40000	1040	80000	35.79
40	2000	80000	2040	160000	84.77
50	1000	50000	1050	100000	47.27
50	2000	100000	2050	200000	110.74

6. 入力あれこれ

ここでは，問題に整数制約や非線形関数が加わるときの入力方法や，規則的なデータや疎なデータの入力方法についてふれておきます．

まず，変数に整数制約を加える場合は，変数の定義のところで，

```
var x1 integer >= 0, <= 500;
var y{1..n,1..m} integer >= 0;
```

のように書きます．また，目的関数が，

$$\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}x_ix_j \rightarrow \text{最小}$$

のように2次の場合は，

```
minimize obj:sum{i in 1..n} sum{j in 1..n}
a[i,j] * x[i] * x[j];
```

のように入力できます．このように，+, -, *, /, ** (四則演算とべき乗) が利用できる上，abs(), exp(), log() などの組込み関数も用意されていますので，さらに複雑な関数も入力できます．ただし，整数制約や

非線形関数については, ソルバーが対応していない場合がありますので, 注意が必要です.

次に, データ入力についてですが,

$$\begin{pmatrix} 3 & 0 & 0 \\ 0 & 0 & 5 \\ 0 & 0 & 7 \end{pmatrix}$$

のように0が多い行列を入力するときには,

```
param n; param a{1..n,1..n};
data;
param n := 3;
param a default 0 :=
   1  1   3
   2  3   5
   3  3   7;
```

のように, デフォルト値を0と定めた上で, 0以外の要素について「行番号, 列番号, 要素の値」の順に入力することができます. このような入力方法は, 大規模で疎な行列のデータを入力する際に便利です. また,

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

のように, 対角要素のみが1であるような行列を定義する場合には,

```
param n;
param a {i in 1..n, j in 1..n} :=
 if i=j then 1 else 0;
```

のように, 規則を書くだけでコンパクトに入力できます. これら以外にも便利な入力方法は多数用意されていますが, 詳しくは [1, 11] などを参照下さい.

7. むすび

数理計画ソフトの簡単な紹介, およびモデル記述言語を用いた手軽な利用法を紹介いたしました.

最近では, 企業においてスケジューリング問題などを解く際に, ここで紹介したような数理計画ソフトを利用したという報告が見受けられるようになってきました [2, 5, 8, 10]. スケジューリング問題の場合は, どうしても組合せ的な制約があるために, 線形計画ではなく, 混合整数計画問題や制約充足問題として定式化することのほうが多いようですが. いずれにしても, 数理計画法が現実の問題にどんどん利用されるようになってきたことは, 大変喜ばしいことと思います.

さて, 数理計画ソフトは「手軽に使えそうだ!」と思っていただけたでしょうか? モデルを記述する具体例を説明するため, かなり数式を使ってしまいましたが, そのせいで「何だ, やっぱり数式がいっぱい出てきて難しいじゃないか」ということにならなければいいのですが.

参考文献

[1] R. Fourer, D.M. Gay and B.W. Kernighan, *AMPL A Modeling Language for Mathematical Programming*, boyd & fraser publishing compagy, 1993.

[2] 羽鳥, "制約論理プログラミングによるフィルム・紙の裁断計画," 生産スケジューリングシンポジウム'97講演論文集 (1997) 157–160.

[3] 茨木, "数理計画: 問題解決への広き門," オペレーションズ・リサーチ, 41 (1996) 313–319.

[4] 茨木, 福島, FORTRAN 77 最適化プログラミング, 岩波書店, 1991.

[5] N. Konishi, Y. Nakagawa, H. Nishida and N. Sakai, "A MIP-Based Approach to the Cutting Stock Problem for Roll and Long Strip Materials with Minimum Production Amount Constraint," *Proc. Advances in Production Management Systems* (1996) 603–608.

[6] 久保, "タダより安い数理計画入門," オペレーションズ・リサーチ, 41 (1996) 337–342.

[7] J.J. Moré and S.J. Wright, *Optimization Software Guide*, SIAM, 1993.

[8] 西田, 山田, 今井, 熊本, 清家, 神谷, 中村, "列生成法による鋳型設置場所割当問題の解法," 日本OR学会秋季研究発表会アブストラクト集 (1996) 112–113.

[9] 新村, "数理計画法のパッケージ, 中間言語, ライブラリー," オペレーションズ・リサーチ, 38 (1993) 124–130.

[10] 高田, 寺野, "作業者のスキルとローテーションを考慮したスケジューリングシステム," 生産スケジューリングシンポジウム'97講演論文集 (1997) 31–36.

[11] H.P. ウィリアムス 著 (前田 監訳, 小林 訳), 数理計画モデルの作成法, 産業図書, 1995.