制約最適化ソルバーSCOP Pythonインターフェイス

LOG OPT Co., Ltd.

目次

- 制約最適化ソルバーSCOPとは
- 制約最適化ソルバーSCOPの特徴
- Pythonインターフェイス
- Pythonインターフェイスを用いた例題
  - 仕事割当問題1
  - 仕事割当問題2
  - 車の投入順決定問題
  - 人員割当問題

制約最適化とは

制約最適化：数理最適化とは異なり，組合せ最適化に特化したパラダイムで，（重み付き）制約充足問題を対象

制約充足問題（constraint satisfaction problem）
与えられた制約を満たす解が存在するかを判定する問題

重み付き制約充足問題（weighted constraint satisfaction problem）

- 単に制約を満たす解を求めるだけでなく，制約からの逸脱量の重み付き和（ペナルティ）を最小化する問題
  - 逸脱を許さない制約（絶対制約，ハード制約）
  - 逸脱を許す制約（考慮制約，ソフト制約）
- 目的関数は，考慮制約として処理

SCOPは重み付き制約充足問題を対象とする

制約最適化ソルバーSCOPとは

大規模な制約最適化問題を高速に解くためのソルバー

- 茨木先生（京都大学名誉教授）と野々部先生（法政大学）の開発したメタヒューリスティクスを基礎
- トライアルバージョン（１５変数まで）
  http://www.logopt.com/scop.htm

1. 簡易モデリング言語（テキスト）によるモデル作成
2. Pythonからの呼び出し
3. ライブラリ呼び出しによる利用が可能
   （C++, Visual Basic, C# ）

制約最適化ソルバーSCOPの特徴

- 誰でも効率の良い定式化が書ける
  - 混合整数最適化ソルバーと異なり定式化の強弱によらない
- 大規模組合せ最適化問題を解くのが得意
- 変数の領域を定義
  - 変数の数が少なくできる
- 制約の表現力大
  - 非凸2次制約（混合整数最適化ソルバーは凸のみ）
  - 相異制約
- 考慮（ソフト）制約
  - 実行不可能性の回避

制約最適化ソルバーSCOPモデルの構成

- 変数（variable）：最適化によって決めるもの.
  変数は，与えられた集合（領域）から1つの要素（値）を選択.
- 領域（domain；ドメイン）：変数ごとに決められた変数のとりえる値の集合.
  数でなくても良い⟶ 表現力が大⟶変数の数が少ない
- 制約（constraint）：幾つかの変数が同時にとることのできる値に制限を付加するための条件.
  - 制約の重み：正数（考慮制約を定義）
    inf（無限大;絶対制約を定義）
  - 制約の種類：Linear（線形制約），Alldiff（相異制約），
    Quadratic（非凸2次制約）

scop.pyモジュールとは

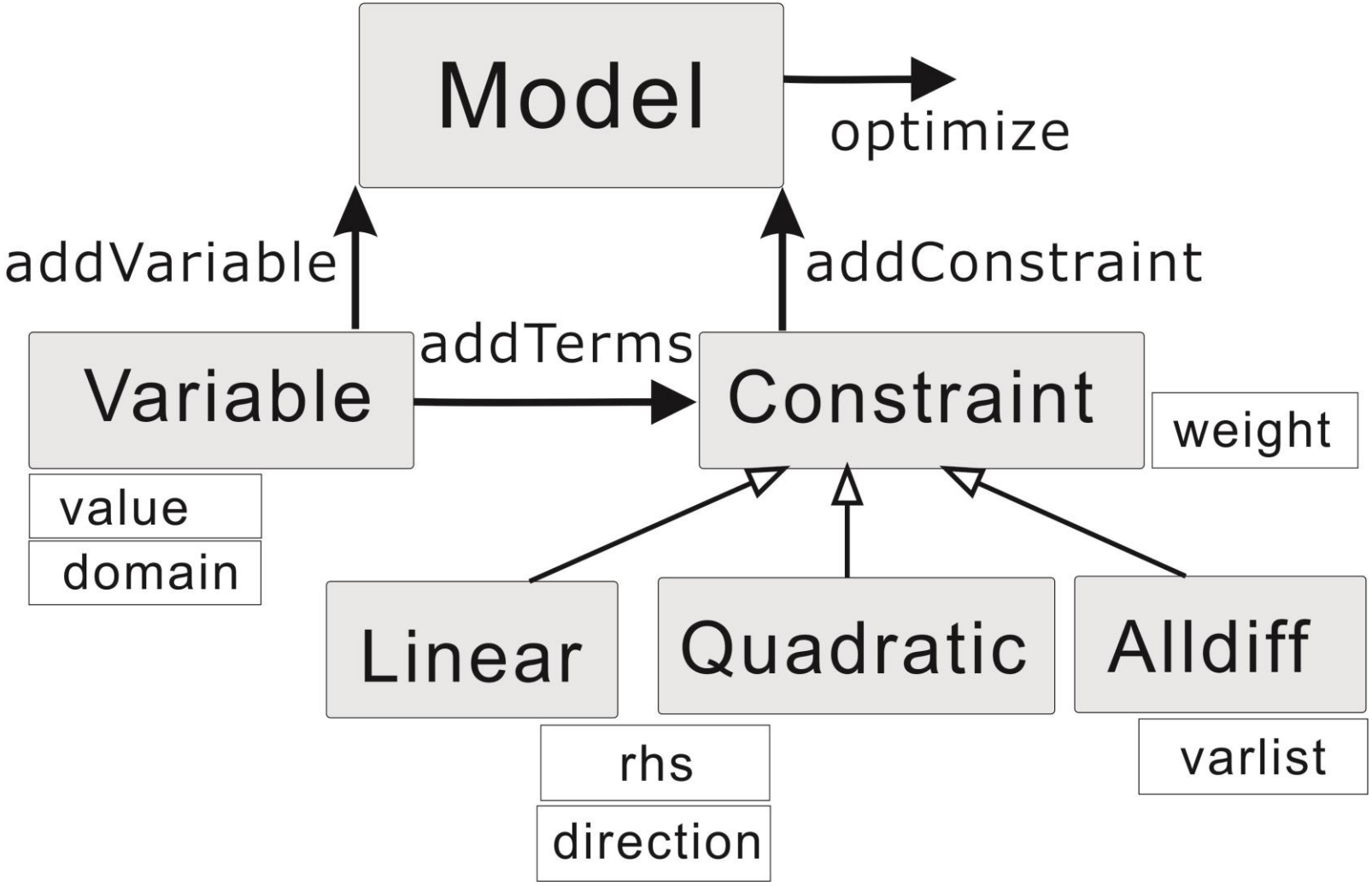
scop.py：制約最適化ソルバーSCOPをPythonから直接呼び出して，求解するためのモジュール

（ソース公開しているため，ユーザが書き換え可能）

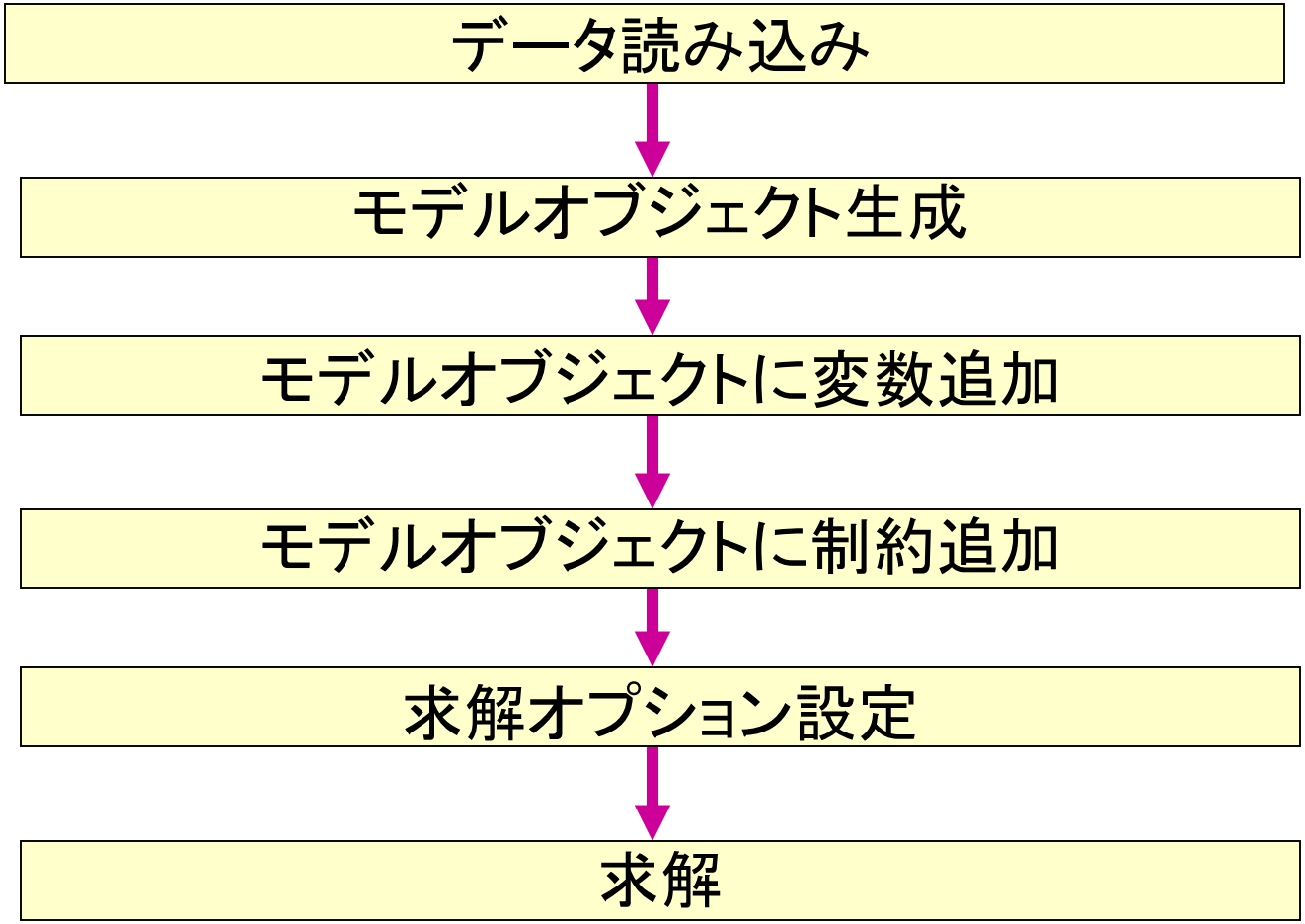
scop.pyモジュールの構成

- モデルクラス Model
- 変数クラス Variable
- 制約クラス Constraint
  - 線形制約クラス Linear
  - 2次制約クラス Quadratic
  - 相異制約クラス Alldiff

クラス間の関係と主要なメソッド・属性



scop.pyモジュールを用いての
モデル作成手順



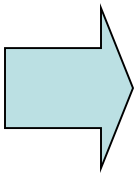
仕事割当問題1

- 3人の作業員 A,B,C を3つの仕事 0,1,2 に割り当てる.

（3人の作業員はそれぞれ異なる仕事に割り当てる必要がある. ）

- 割当費用＝

	0	1	2
A	15	20	30
B	7	15	12
C	25	10	13



割当は相異制約（Alldiff），割当費用は線形制約（Linear）

定式化
仕事割当問題1

混合整数最適化の定式化:

$$\text{minimize} \quad \sum_{ij} c_{ij} x_{ij}$$

作業員iが仕事jに割当られるときの割当費用c_{ij}の合計を最小化

$$\text{subject to} \quad \sum_{i} x_{ij} = 1 \qquad \forall j \quad (1)$$

各仕事にはいずれかの作業員を割り振る

$$\sum_{j} x_{ij} = 1 \qquad \forall i \quad (2)$$

各作業員はいずれかの仕事に割り振られる

$$x_{ij} \in \{0,1\} \qquad \forall i,j \quad (3)$$

作業員iの仕事jへ割り振られるとき1, その他0である変数

制約最適化の場合: 上記の式(2),(3)を下記のように簡略化できる.

$X_i \in \{D_i\} \quad \forall i$ 各作業員を表す変数 X_i は領域D_i（仕事0,1,2）から1つの値を選択.

変数の数が1/仕事数 になる

Python インターフェイスでのモデル作成1

（仕事割当問題1：assign1.py ）

無料配布
例題集の
ファイル名

```
from scop import *          #scop.pyモジュールを読み込む.

#データ作成
workers=['A','B','C']      #作業員のデータをリストで保管.
Jobs  =  [0,1,2]           #作業のデータをリストで保管.

Cost={('A',0):15, ('A',1):20, ('A',2):30,
      ('B',0): 7, ('B',1):15, ('B',2):12,
      ('C',0):25, ('C',1):10, ('C',2):13 }
```

#割当費用を辞書で保管.
例えば，('A',0):15は作業員Aが仕事0に割り当てられるときの費用が15であることを表す.

```
#モデル作成
m=Model()      #モデルオブジェクト(インスタンス)mを生成.
```

Python インターフェイスでのモデル作成2

（仕事割当問題1：assign1.py ）

作業員をいずれかの作業に割り振ることを表す変数：

$$X_i \in \{D_i\} \quad \forall i$$

モデルテキスト：

予約語

```
variable A in { 0,1,2 }
variable B in { 0,1,2 }
variable C in { 0,1,2 }
```

変数名　　領域

Python入力：

```
x={}    #変数を保管するための辞書を作成.
for i in workers:
    x[i]=m.addVariable(name=i,domain=Jobs) #モデルに変数を追加.
```

変数追加メソッドaddVariable(変数名,リスト形式の変数の領域)

Python インターフェイスでのモデル作成3

（仕事割当問題1：assign1.py ）

定式化：

$$\sum_i x_{ij} = 1 \qquad \forall j \quad (1)$$

モデルテキスト：

AD: weight= inf type=alldiff B C A ;

制約名

重み：必ず守る必要のある場合，inf

制約タイプ

変数リスト

Python記述：

```
#すべての作業員が異なる作業に割り当てられることを表す相異制約に追加するための変数リストを作成.
xlist=[]
for i in x:
    xlist.append(x[i])
#すべての作業員が異なる作業に割り当てられることを表す制約.
con1=Alldiff('AD',xlist,weight='inf')
m.addConstraint(con1)       #作成した制約をモデルに追加する.
```

絶対制約

相異制約：Alldiff ('制約名',変数リスト，制約重み)

Python インターフェイスでのモデル作成4

（仕事割当問題1：assign1.py ）

定式化： minimize $\sum_{ij} c_{ij} x_{ij}$

制約名

重み：目的関数など考慮
制約の場合，1など整数

制約タイプ

モデルテキスト：

linear_constraint: weight= 1 type=linear 15(A,0) 20(A,1) 30(A,2) 7(B,0) 15(B,1) 12(B,2) 25(C,0) 10(C,1) 13(C,2) <=0

sum [費用（変数，値）]<=0：例えば，15(A,0)は，作業員Aが仕事0に割り振られるときの費用が15であることを表す．

Python記述：

```
#作業員の各作業への割当費用を表す制約.
con2=Linear('linear_constraint',weight=1,rhs=0,direction='<=')
```

絶対制約

線形制約：Linear('制約名', 制約重み，右辺値，制約向き)

```
for i in workers:
    for j in Jobs:
        con2.addTerms(Cost[i,j],x[i],j)

m.addConstraint(con2)        #作成した制約をモデルに追加する.
```

左辺追加メソッドaddTerms(係数，変数，値)

Python インターフェイスでのモデル作成5

（仕事割当問題1：assign1.py ）

```
print (m)                                    #モデルをプリントする.
```

出力結果：

```
Model:
number of variables = 3
number of constraints= 2
variable A:['0', '1', '2'] = None
variable B:['0', '1', '2'] = None
variable C:['0', '1', '2'] = None
AD: weight= inf type=alldiff  B C A ;  :LHS =0
linear_constraint: weight= 1 type=linear 15(A,0) 20(A,1) 30(A,2) 7(B,0)
     15(B,1) 12(B,2) 25(C,0) 10(C,1) 13(C,2) <=0 :LHS =0
```

Python インターフェイスでのモデル作成6

（仕事割当問題1：assign1.py ）

```
m.Params.TimeLimit=1        #求解時間を1秒に設定する.
sol,violated=m.optimize()   #解と違反制約をsolとviolatedへ保存.

print ('solution‘)
for x in sol:                          #結果のプリント
    print (x,sol[x])
print ('violated constraint(s)')
for v in violated:                     #違反制約のプリント
    print (v,violated[v])
```

出力結果：

```
solution
C 1
A 0
B 2
violated constraint(s)
linear_constraint 37
```

Python インターフェイスでのモデル作成7
（仕事割当問題1：assign1.py ）

求解後自動的に出力される結果ファイルscop_out.txtの一部:

```
[best solution]
A:0
C:1
B:2
penalty: 0/37 (hard/soft)
[Violated constraints]
obj: 37
```

#作業員Aが作業0，作業員Cが作業1，
作業員Bが作業2に割り振られたことを表す．

#hardは重みinfの絶対制約，
softは正数重みの考慮制約の逸脱値

#割当費用が37であることを表す．

割当費用：37 =15+12+10

	0	1	2
A	15	20	30
B	7	15	12
C	25	10	13

仕事割当問題2

- ５人の作業員 A,B,C,D,E を3つの仕事 0,1,2 に割当
- 各仕事の必要人数は(1,2,2)人
- 作業員AとCは仲が悪いので，異なる仕事に割り振る
- 割当費用＝

	0	1	2
A	15	20	30
B	7	15	12
C	25	10	13
D	15	18	3
E	5	12	17

Python インターフェイスでのモデル作成1
（仕事割当問題2 : assign3.py）

```
from scop import *
m=Model()                                  #モデル作成
workers=['A','B','C','D','E']              #作業員のデータをリストで保管.
Jobs  =  [0,1,2]                           #作業のデータをリストで保管.

Cost={ ('A',0):15, ('A',1):20, ('A',2):30,
       ('B',0): 7, ('B',1):15, ('B',2):12,
       ('C',0):25, ('C',1):10, ('C',2):13,
       ('D',0):15, ('D',1):18, ('D',2): 3,
       ('E',0): 5, ('E',1):12, ('E',2):17
       }                                   #割当費用を辞書で保管.

LB={0: 1,
    1: 2,
    2: 2}                                  #各作業の必要人数下限
```

Python インターフェイスでのモデル作成2
（仕事割当問題2 :assign3.py）

モデルテキスト
variable A in { 0,1,2 }
variable B in { 0,1,2 }
variable C in { 0,1,2 }
variable D in { 0,1,2 }
variable E in { 0,1,2 }

Python入力

```
#作業を領域とする作業員変数作成.
x={}
for i in workers:
    x[i]= m.addVariable(i,Jobs)
```

重み1，右辺0
の考慮制約

モデルテキスト
obj: weight= 1 type=linear 15(A,0)
20(A,1) 30(A,2) 7(B,0) 15(B,1) 12(B,2)
25(C,0) 10(C,1) 13(C,2) 15(D,0)
18(D,1) 3(D,2) 5(E,0) 12(E,1) 17(E,2)
<=0

Python入力

```
#作業員の各作業への割当費用を表す制約.
obj=Linear('obj',1,0,'<=')
for i in workers:
    for j in Jobs:
        obj.addTerms(Cost[i,j],x[i],j)
m.addConstraint(obj)
```

Python インターフェイスでのモデル作成3
（仕事割当問題2 :assign3.py）

モデルテキスト

```
LB0: weight= inf type=linear 1(A,0)
1(B,0) 1(C,0) 1(D,0) 1(E,0) >=1
LB1: weight= inf type=linear 1(A,1)
1(B,1) 1(C,1) 1(D,1) 1(E,1) >=2
LB2: weight= inf type=linear 1(A,2)
1(B,2) 1(C,2) 1(D,2) 1(E,2) >=2
```

Python入力

```
#各仕事に割り振られる作業員の数≦　作業の必要人数下限
LBC={}
for j in Jobs:
    LBC[j]=Linear(
                'LB{0}'.format(j),'inf',LB[j],'>=')
    for i in workers:
        LBC[j].addTerms(1,x[i],j)
    m.addConstraint(LBC[j])
```

右辺がLB[j]の絶対制約

Python インターフェイスでのモデル作成4
（仕事割当問題2 :assign3.py）

モデルテキスト

sum [費用（変数1，値1）（変数2，値2）]=0：例えば，1(A,0)(C,0)は，AとCが仕事0に割り振られるときの費用が1であることを表す．

conflict: weight=100 type=quadratic 1(A,0)(C,0) 1(A,1)(C,1) 1(A,2)(C,2) =0

- 制約名
- 重み：目的関数など考慮制約の場合，1など整数
- 制約タイプ

重み100，右辺0の考慮制約

Python入力

```
#作業員AとCを同じ仕事に割り振ることができない

conf=Quadratic('conflict',100,0,'=')
for j in Jobs:
    conf.addTerms(1,x['A'],j,x['C'],j)
m.addConstraint(conf)
```

2次制約：Quadratic
('制約名', 制約重み，右辺値，制約向き)

左辺追加メソッドaddTerms
(係数1，変数1，係数2，変数2，領域値)

Python インターフェイスでのモデル作成5
（仕事割当問題2 : assign3.py）

```
m.Params.TimeLimit=1        #求解時間を1秒に設定する.
sol,violated=m.optimize()   #解と違反制約をsolとviolatedへ保存.
```

求解後自動的に出力される結果ファイルscop_out.txtの一部:

```
[best solution]
A: 0
B: 2
C: 1
D: 2
E: 1
penalty: 0/52 (hard/soft)

[Violated constraints]
obj: 52
```

割当費用:52=15+12+10+3+12

	0	1	2
A	15	20	30
B	7	15	12
C	25	10	13
D	15	18	3
E	5	12	17

車の投入順決定問題

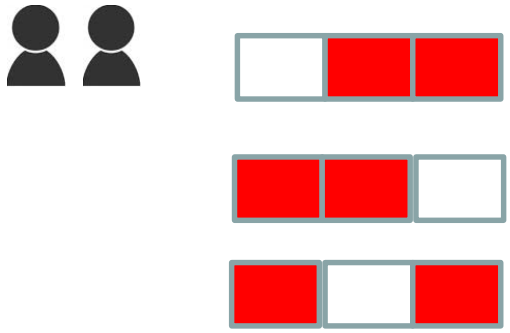
6種類の車(A, B, C, D, E, F)を同じ生産ラインで組み立てる.

需要量(D): A:1台, B :1台, C :2台, D :2台, E :2台, F :2台

各Optionを付ける必要のある車の種類とOptionの処理能力は下記の通り.

Option	車の種類	処理能力 Capacity/Length
0	A E F	1/2
1	C D F	2/3
2	A E	1/3
3	A B D	2/5
4	C	1/5

e.g.:処理能力=2/3は, 連続する3つの作業スペースで2つしか処理できないことを表す.



line

上記の制約の下でどの順序で車を生産ラインに投入すれば良いか.

定式化
車の投入順決定問題

定式化：

$$\text{subject to} \quad \sum_{j} x_{ij} = \text{Number}_i \qquad \forall i \quad (1)$$

各種類の車の生産量は需要量と同じ

$$\sum_{k}^{k+\text{Length}_i} x_{ik} \le \text{Capacity}_i \qquad \forall i,k \quad (2)$$

各Optionの作業効率制約

$$\sum_{i} x_{ij} = 1 \qquad \forall j \quad (3)$$

各順番jにはいずれかの種類の車を割り当てる

$$x_{ij} \in \{0,1\} \qquad \forall i,j \quad (4)$$

種類iの車をj番目に投入するときに1，その他のとき0であることを表す変数

制約最適化の場合：$X_j \in \{\text{車の種類の集合 } A, B, \ldots, F\} \qquad \forall j$ （各順番 j）

Python インターフェイスでのモデル作成1

（車投入順序決定問題：car_small.py ）

データ作成

```
from scop import *
m=Model()
Type=['A','B','C','D','E','F']                    #車の種類
Number={'A':1,'B':1,'C':2,'D':2,'E':2,'F':2}      #車の種類ごとの需要量
n=sum(Number[i] for i in Number)                  #車の全体の生産必要な台数

Option=[['A','E','F'],                            # Optionごとの車の種類
        ['C','D','F'],
        ['A','E'],
        ['A','B','D'],
        ['C']]

Length=[2,3,3,5,5]                                # Optionごとの処理能力：Capacity/Length
Capacity=[1,2,1,2,1]
```

Python インターフェイスでのモデル作成2

（車投入順序決定問題：car_small.py ）

定式化　$X_j \in \{車の種類（Type）\}$　$\forall j$

モデルテキスト

```
variable seq[0] in { A,B,C,D,E,F }
variable seq[1] in { A,B,C,D,E,F }
…
variable seq[9] in { A,B,C,D,E,F }
```

Python入力

```
#何番目にどの種類の車を投入するかを表す変数.
X={}
for j in range(n):
    X[j]=m.addVariable('seq[{0}] '.format(j),Type)
```

Python インターフェイスでのモデル作成3
（車投入順序決定問題：car_small.py ）

定式化 $\sum_j x_{ij} = Number_i \quad \forall i \quad (1)$

モデルテキスト

```
req[A]: weight= 1 type=linear 1(seq[0],A) 1(seq[1],A) 1(seq[2],A)
1(seq[3],A) 1(seq[4],A) 1(seq[5],A) 1(seq[6],A) 1(seq[7],A) 1(seq[8],A)
1(seq[9],A) =1
…
req[F]:  …
```

Python入力

```
for i in Type:        #各種類の車の生産量制約
    L1=Linear(
                'req[{0}] '.format(i),direction='=',rhs=Number[i])
    for j in range(n):
        L1.addTerms(1,X[j],i)
    m.addConstraint(L1)
```

Python インターフェイスでのモデル作成4

（車投入順序決定問題：car_small.py ）

定式化

$$\sum_{k}^{k+Length_i} x_{ik} \leq Capacity_i \quad \forall i,k \quad (2)$$

モデルテキスト

ub[0_1]: weight= 1 type=linear 1(seq[1],A) 1(seq[1],E) 1(seq[1],F) 1(seq[2],A) 1(seq[2],E) 1(seq[2],F) <=1

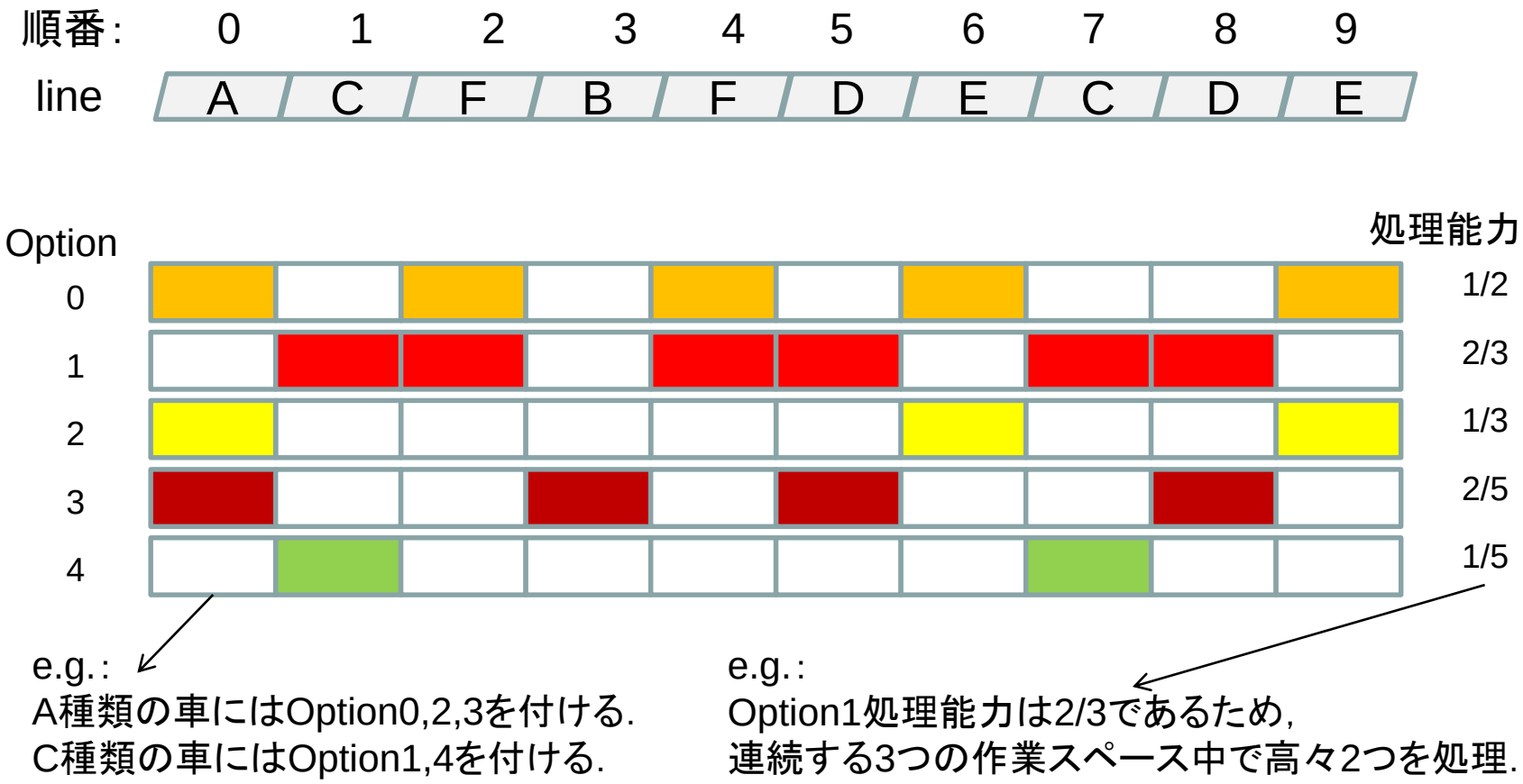
…

Option0の処理能力が1/2であるため，Option0を付ける必要のある車（A，E，F）を1番目と2番目に連続投入することができないことを表す．

Python入力

```
for i in range(len(Length)):          #各Optionの処理能力の制約
    for k in range(n-Length[i]+1):
        L2=Linear('ub[{0}_{1}] '.format(i,k),direction='<=',rhs=Capacity[i])
        for t in range(k,k+Length[i]):
            for j in range(len(Option[i])):
                L2.addTerms(1,X[t],Option[i][j])
        m.addConstraint(L2)
```

車の投入順決定問題(結果)



人員割当問題

計画期間

	1	2	3	4	5	6	7
A							
B				?			
C							
D							
朝	1	1	1	1	1	1	1
昼	1	1	1	1	1	1	1
夜	1	1	1	1	1	1	1

3人の
スタッフ

各シフトの
必要人数

出勤日数min	出勤日数max
3	6
3	6
3	6
3	6

シフトの種類

朝 昼 夜 休

- 各スタッフは1日高々1つのシフトしか行うことができない.
- 異なるシフトに移るときには，必ず休日を入れる.
- シフト「昼」,「夜」は，最低2日間は連続で行う.
- シフト「夜」は4日以上連続で行うことができない.

Python インターフェイスでのモデル作成1
（人員割当問題：staff2.py ）

```
#データ設定
periods=[1,2,3,4,5,6,7]          #期
shifts=[0,1,2,3]                 #シフト
staffs=["A","B","C","D"]         #スタッフ

var={}                           #期tのスタッフiを変数，シフトの集合をドメインに設定
for i in staffs:
    for t in periods:
        var[i,t]=m.addVariable(name=i+str(t),domain=shifts)
```

数理最適化定式化での変数：　$x_{its} \in \{0,1\}$　　$\forall i, t, s$

SCOPの定式化の変数：　var[i,t]　∈　{シフトsの集合}

Python インターフェイスでのモデル作成2

（人員割当問題：staff2.py ）

```
#各スタッフは最低3日以上出勤する必要がある.（「休」を除き，各スタッフに割り当てられるシフトの合計が3以上である制約）
LB={}
for i in staffs:
    LB[i]=Linear("LB_{0}".format(i),rhs=3,direction=">=")
    for t in periods:
        for s in range(1,len(shifts)):
            LB[i].addTerms(1,var[i,t],shifts[s])
    m.addConstraint(LB[i])
```

$$\sum_{t,s} x_{its} \geqq 3 \qquad \forall i$$

```
#各スタッフは1日以上休みを取る必要がある.（「休」を除き，各スタッフに割り当てられるシフトの合計が6以下である制約）
UB={}
for i in staffs:
    UB[i]=Linear("UB_{0}".format(i),rhs=6,direction="<=")
    for t in periods:
        for s in range(1,len(shifts)):
            UB[i].addTerms(1,var[i,t],shifts[s])
    m.addConstraint(UB[i])
```

$$\sum_{t,s} x_{its} \leqq 6 \qquad \forall i$$

Python インターフェイスでのモデル作成2
（人員割当問題：staff2.py ）

#各スタッフの夜勤回数は最大4回まで可能.

$$\sum_{t,s=3} x_{its} \leqq 4 \qquad \forall i$$

```
UB_night={}
for i in staffs:
    UB_night[i]=Linear("UB_night_{0}".format(i),rhs=4,direction="<=")
    for t in periods:
        UB_night[i].addTerms(1,var[i,t],shifts[-1])
    m.addConstraint(UB_night[i])
```

#各シフトには1人のスタッフを割り当てる必要がある.

$$\sum_{i} x_{its} = 1 \qquad \forall t, s$$

```
UB_shift={}
for t in periods:
    for s in range(1,len(shifts)):
        UB_shift[t,s]=Linear(“UBshift_{0}_{1}”.format(t,s),rhs=1,direction=“=”)
        for i in staffs:
            UB_shift[t,s].addTerms(1,var[i,t],shifts[s])
        m.addConstraint(UB_shift[t,s])
```

Python インターフェイスでのモデル作成2

（人員割当問題：staff2.py ）

#異なるシフトに移る場合は休みを入れる必要がある.（異なるシフトが2日間連続で行うのを禁止する制約. ）

$$x_{its} + \sum_{k \neq s} x_{i,t+1,k} \leqq 1 \quad \forall i, t, s$$

```
Forbid={}
for i in staffs:
    for t in periods:
        for s in range(1,len(shifts)):
          Forbid[(i,t,s)]=Linear("Forbid_{0}_{1}_{2}".format(i,t,s),rhs=1)
          Forbid[(i,t,s)].addTerms(1,var[i,t],shifts[s])
          for k in range(1,len(shifts)):
              if k!=s:
                  if t==periods[-1]:
                      Forbid[(i,t,s)].addTerms(1,var[i,1],shifts[k])
                  else:
                      Forbid[(i,t,s)].addTerms(1,var[i,t+1],shifts[k])
          m.addConstraint(Forbid[(i,t,s)])
```

Python インターフェイスでのモデル作成2
（人員割当問題：staff2.py ）

$$-x_{its} + x_{i,t-1,s} + x_{i,t+1,s} \geqq 0 \quad \forall i, t, s \in \{2,3\}$$

```
#シフト「昼」,「夜」は，最低2日間は連続で行う.
Cons={}
for i in staffs:
    for t in periods:
      for s in range(2,len(shifts)):
          Cons[(i,t)]=Linear("Cons_{0}_{1}".format(i,t),direction=">=")
          Cons[(i,t)].addTerms(-1,var[i,t],shifts[s])
          if t==1:
              Cons[(i,t)].addTerms(1,var[i,periods[-1]],shifts[s])
          else:
              Cons[(i,t)].addTerms(1,var[i,t-1],shifts[s])
          if t==periods[-1]:
              Cons[(i,t)].addTerms(1,var[i,1],shifts[s])
          else:
              Cons[(i,t)].addTerms(1,var[i,t+1],shifts[s])
          m.addConstraint(Cons[(i,t)])
```

人員割当問題結果

	1	2	3	4	5	6	7
A	休	休	昼	昼	昼	昼	昼
B	朝	朝	朝	朝	休	朝	朝
C	昼	昼	休	夜	夜	夜	休
D	夜	夜	夜	休	朝	休	夜

出勤日数min	出勤日数max
3	6
3	6
3	6
3	6

朝	1	1	1	1	1	1	1
昼	1	1	1	1	1	1	1
夜	1	1	1	1	1	1	1

初期解の設定方法

- 初期解を設定せずに求解成功すると自動的にscop_best_data.txtというファイルが生成される.
  このファイルが初期解ファイルである.

- scop_best_data.txtがある環境で，モデルの属性Params.initialをTrueに設定する.
  e.g.：モデル名.Params.Initial=True

例題は次のページ

初期解設定例

下記のファイルに初期解が保存される

scop_best_data.txt

```
X_3_2_1: 1
X_4_1_1: 1
X_3_1_1: 0
X_3_2_2: 0
X_4_1_2: 1
X_3_1_2: 0
y_f4: 4_1_2
y_f2: 3_2_1
y_f1: 3_2_1
y_f3: 4_1_1
z_f4: 4_1_2
z_f2: 3_2_1
z_f1: 0
z_f3: 4_1_1
```

モデルファイル.py

```
print (model)
model.Params.Initial=True
model.Params.TimeLimit=1
sol,violated= model.optimize()

print ('solution')
for x in sol:
    print (x,sol[x])
print ('violated constraint(s)')
for v in violated:
    print (v,violated[v])
```