

問題解決のための最適化

組合せ最適化と整数計画法

4. 施設配置問題

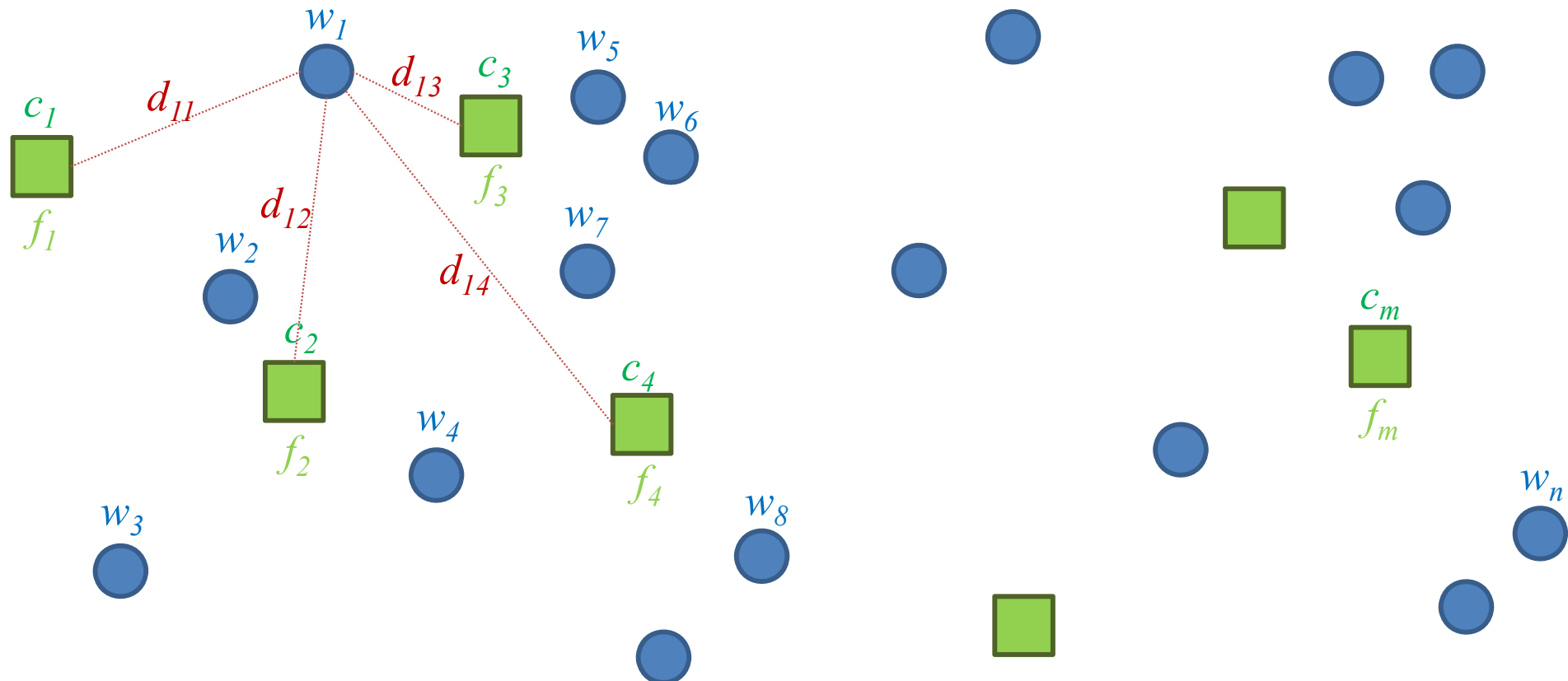
堀田 敬介

施設配置問題の最適化

● = 需要点 (n 個)
■ = 施設配置候補点 (m 個)

➤ 施設配置問題 facility location problem

- n 個の需要点と m 個の施設配置候補点がある
- 需要点 i には重み w_i が, 施設 j には容量 c_j と設置コスト f_j がある (所与)
- 需要点 i が施設 j を利用する場合の利用コスト d_{ij} が所与
- 候補点から k 個選んで施設を作りたい. k は所与



施設配置問題の最適化

➤ 最適化問題の定式化(変数設定・係数表記)

➤ 0-1変数 $x_{ij} = \begin{cases} 1 & \dots \text{需要点 } i \text{ は施設 } j \text{ を利用} \\ 0 & \dots \text{需要点 } i \text{ は施設 } j \text{ を利用せず} \end{cases}$

➤ 0-1変数 $y_j = \begin{cases} 1 & \dots \text{施設 } j \text{ を設置する} \\ 0 & \dots \text{施設 } j \text{ を設置しない} \end{cases}$

➤ 需要点集合 $I = \{1, \dots, n\}$, 需要点 i の重み w_i ($\mathbf{w} = (w_1, \dots, w_n)$)

➤ 施設候補集合 $J = \{1, \dots, m\}$, 施設候補点 j の容量 c_j ($\mathbf{c} = (c_1, \dots, c_m)$)

➤ 施設 j の設置にかかるコスト f_j ($\mathbf{f} = (f_1, \dots, f_m)$)

➤ 施設利用コスト行列 $\mathbf{D} = [d_{ij}]$

➤ 設置する施設の数 k (所与)

需要点の重みを用いず
変数 x_{ij} を用量とする(1
需要点が複数施設の利用可)定式化もあるが,
ここでは1需要点は1施設のみ利用とする

施設の容量 c_j を考慮しない定式化もあるが, 現実の事例では, 考慮しないことの方が少ないと思うので, ここでは含めたもののみ扱う

➤ 施設配置問題のバリエーション(一部)

➤ capacitated FLP ... f_j あり, k なし, 利用コスト+設置コスト最小化

➤ k -median ... f_j なし, k あり, 利用コスト加重和の最小化(min. w.sum.)

➤ k -center ... f_j なし, k あり, 利用コスト最大値の最小化(min. max.)

施設配置問題の最適化

➤ capacitated facility location(CFL)の定式化(f_j あり, k なし)

min.

$$\sum_{i=1}^n \sum_{j=1}^m d_{ij} x_{ij} + \sum_{j=1}^m f_j y_j$$

s. t.

$$\sum_{j=1}^m x_{ij} = 1 \quad (\forall i \in I)$$

$$\sum_{i=1}^n w_i x_{ij} \leq c_j y_j \quad (\forall j \in J)$$

$$x_{ij} \leq y_j \quad (\forall i \in I, \forall j \in J)$$

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

$$y_j \in \{0,1\} \quad (\forall j \in J)$$

目的:「利用コストの和」と「施設設置費用の和」の和を最小化

制約1: 任意の需要点 i は, $j=1, \dots, m$ ある施設のうち丁度1施設を利用する

制約2: 各施設 j の容量制約. その施設 j を利用する需要点の総和は容量を c_j 以下

制約3: 設置した施設 ($y_j=1$) のみ利用可. (定式化の強化用制約でもある)

施設配置問題の最適化

➤ Facility Location (k -median) の定式化 (f_j なし, k あり)

min.

$$\sum_{i=1}^n \sum_{j=1}^m d_{ij} x_{ij}$$

s. t.

$$\sum_{j=1}^m x_{ij} = 1 \quad (\forall i \in I)$$

$$\sum_{i=1}^n w_i x_{ij} \leq c_j y_j \quad (\forall j \in J)$$

$$x_{ij} \leq y_j \quad (\forall i \in I, \forall j \in J)$$

$$\sum_{j=1}^m y_j = k$$

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

$$y_j \in \{0,1\} \quad (\forall j \in J)$$

目的: 「利用コストの和」を最小化

制約1: 任意の需要点 i は, $j=1, \dots, m$ ある施設のうち丁度1施設を利用する

制約2: 各施設 j の容量制約. その施設 j を利用する需要点の総和は容量を c_j 以下

制約3: 設置した施設 ($y_j=1$) のみ利用可. (定式化の強化用制約でもある)

制約4: 設置する施設は k 個

施設配置問題の最適化

➤ Facility Location (k -center) の定式化 (f_j なし, k あり)

min. t

s. t.

$$\sum_{j=1}^m x_{ij} = 1 \quad (\forall i \in I)$$

$$\sum_{i=1}^n w_i x_{ij} \leq c_j y_j \quad (\forall j \in J)$$

$$x_{ij} \leq y_j \quad (\forall i \in I, \forall j \in J)$$

$$\sum_{j=1}^m y_j = k$$

$$d_{ij} x_{ij} \leq t \quad (\forall i \in I, \forall j \in J)$$

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

$$y_j \in \{0,1\} \quad (\forall j \in J)$$

目的: 最大利用コストを最小化

制約1: 任意の需要点 i は, $j=1, \dots, m$ ある施設のうち丁度1施設を利用する

制約2: 各施設 j の容量制約. その施設 j を利用する需要点の総和は容量を c_j 以下

制約3: 設置した施設 ($y_j=1$) のみ利用可. (定式化の強化用制約でもある)

制約4: 設置する施設は k 個

制約5: 全ての利用コストは t 以下
目的関数でこの t を最小化することで、
最大値最小 minimax を求められる

施設配置問題の最適化

➤ 施設配置問題 facility location problem (例1)

- 需要点 $n=10$, 施設配置候補点 $m=5$
- 需要点 i と候補点 j 間の距離 d_{ij} が所与 (右表)
- 需要点の重み w_i (人口)
- 施設容量 c_j (収容数)
- 施設設置コスト f_j
- 施設設置数 $k=3$ カ所

- CFL 定式化 (容量ありFL) で解く
 - f_j あり, k なし
- k -median 定式化 (距離総和を最小化) で解く
 - f_j なし, k あり
- k -center 定式化 (距離最長を最小化) で解く
 - f_j なし, k あり

需要点重み w_i
 施設容量 c_j , 設置コスト f_j
 需要点 $\leftrightarrow$ 候補点 距離 d_{ij}

	候補	1	2	3	4	5
需要	c_j	120	130	125	135	125
	f_j	90	50	70	80	60
1	30	7	6	6	9	9
2	25	3	5	7	8	5
3	40	3	7	8	5	4
4	35	2	7	1	7	9
5	25	8	7	6	7	9
6	20	3	4	2	8	4
7	30	9	9	6	5	7
8	45	2	9	7	1	5
9	30	8	9	7	7	4
10	50	8	6	6	2	1

施設配置問題をCPLEXで解く

CFL定式化
(f_j あり, k なし)

➤ 新規プロジェクトの作成

- ① [ファイル(F)]ー[新規(N)]ー[OPLプロジェクト]を選択
- ② [プロジェクト名] を記入 (例: **FacilityLocation**) し, 3カ所にチェックする
 - ☒ デフォルトの実行構成の追加
 - ☒ モデルの作成
 - ☒ データの作成
- ③ [終了]をクリック

プロジェクト名は自由だが, **半角英数**で何の問題を解こうとしているのかが分かる名前が良い

➤ プロジェクト内のいくつかの名前を変更

- ✓ [構成1] → [**config1**] ※日本語を英語に変更しないと実行時エラーになる
- ✓ モデルファイル [FacilityLocation.mod] → [**cfl.mod**] ※CFLP定式化用
- ✓ データファイル [FacilityLocation.dat] → [**flex1.dat**]

➤ モデルファイル・データファイルを記述し保存 (次ページ参照)

➤ [config1]にモデルファイルとデータファイルをセットし, 解く

施設配置問題を CPLEXで解く

➤ cfl.mod

3行目「`int k = ...;`」
は必要ない(使わない)が、
datファイルを共通化
したいので書いてある

```
int i_max = ...; // 需要点の数
int j_max = ...; // 施設候補点の数
int k = ...; // 配置したい施設数
```

```
range I = 1..i_max; // 需要点集合の範囲
range J = 1..j_max; // 施設候補地集合の範囲
```

```
float d[I,J] = ...; // 需要点i が施設j を使う場合の利用コスト
int w[I] = ...; // 需要点i の重み
int c[J] = ...; // 施設j の容量
int f[J] = ...; // 施設j の設置コスト
```

```
dvar int x[I,J] in 0..1; // 需要点i が施設j を利用するかどうかの0-1変数
dvar int y[J] in 0..1; // 候補地j に施設を配置するかどうかの0-1変数
```

```
minimize
    sum(i in I) sum(j in J) d[i,j]*x[i,j] + sum(j in J) f[j]*y[j];
subject to {
    forall(i in I) {
        sum(j in J) x[i,j] == 1;
    }
    forall(j in J) {
        sum(i in I) w[i]*x[i,j] <= c[j]*y[j];
    }
    forall(i in I) {
        forall(j in J) {
            x[i,j] <= y[j];
        }
    }
}
```

CFL定式化
(f_j あり, k なし)

施設配置問題をCPLEXで解く

➤ flex1.dat

```
SheetConnection sheet("flex.xlsx");  
i_max from SheetRead(sheet, "Sheet1!B1");  
j_max from SheetRead(sheet, "Sheet1!B2");  
k from SheetRead(sheet, "Sheet1!J1");  
  
w from SheetRead(sheet, "Sheet1!B4:B13");  
c from SheetRead(sheet, "Sheet1!C2:G2");  
f from SheetRead(sheet, "Sheet1!C3:G3");  
d from SheetRead(sheet, "Sheet1!C4:G13");  
  
y to SheetWrite(sheet, "Sheet1!J3:N3");  
x to SheetWrite(sheet, "Sheet1!J4:N13");
```

データファイル(*.dat)は、3種類の定式化で共通で使いたいので、モデルによっては使わない定数も全て記述してあることに注意

施設配置問題をCPLEXで解く

➤ flex.xlsx [Sheet1]

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	需要数	10	1	2	3	4	5	施設候補	設置数	3						0	= P3+P14	目的関数
2	施設数	5	120	130	125	135	125	容量cj		1	2	3	4	5			設置コスト合計	
3	需要	重みwi	90	50	70	80	60	設置コストfj	y =							0	= SUMPRODUCT(C3:G3, J3:N3)	
4	1	30	7	6	6	9	9	距離dij	x =						1	0	↓ [P4:P13]へコピー	需要毎の距離
5	2	25	3	5	7	8	5								2	0		
6	3	40	3	7	8	5	4								3	0		
7	4	35	2	7	1	7	9								4	0		
8	5	25	8	7	6	7	9								5	0		
9	6	20	3	4	2	8	4								6	0		
10	7	30	9	9	6	5	7								7	0		
11	8	45	2	9	7	1	5								8	0		
12	9	30	8	9	7	7	4								9	0		
13	10	50	8	6	6	2	1								10	0		
14			0	0	0	0	0	施設ごと重み和								0	= SUM(P4:P13)	距離合計
15			[C14] = SUMPRODUCT(\$B4:\$B13, J4:J13)															
16			→[D14:G14]へコピー															

作成・保存後, [プロジェクト]へ追加

施設配置問題をCPLEXで解く

CFL定式化
(f_j あり, k なし)

➤ 結果([解]タブ)

```
// solution (optimal) with objective 223 ←
// Quality Incumbent solution:
// MILP objective
// MILP solution norm |x| (Total, Max)
// MILP solution error (Ax=b) (Total, Max)
// MILP x bound error (Total, Max)
// MILP x integrality error (Total, Max)
// MILP slack bound error (Total, Max)
//
```

```
x = [[0 1 0 0 0]
      [0 1 0 0 0]
      [0 1 0 0 0]
      [0 0 1 0 0]
      [0 0 1 0 0]
      [0 0 1 0 0]
      [0 0 1 0 0]
      [0 0 0 0 1]
      [0 0 0 0 1]
      [0 0 0 0 1]];
y = [0 1 1 0 1];
```

最適値 = 223

2.2300000000e+02

1.30000e+01 1.00000e+00

0.00000e+00

0.00000e+00

0.00000e+00

0.00000e+00

候補		1	2	3	4	5
需要	c_j	120	130	125	135	125
	f_j	90	50	70	80	60
1	30	7	6	6	9	9
2	25	3	5	7	8	5
3	40	3	7	8	5	4
4	35	2	7	1	7	9
5	25	8	7	6	7	9
6	20	3	4	2	8	4
7	30	9	9	6	5	7
8	45	2	9	7	1	5
9	30	8	9	7	7	4
10	50	8	6	6	2	1

CFL定式化 (f_j あり, k なし)

[illegible]

施設配置問題をCPLEXで解く

k -median定式化
(f_j なし, k あり)

➤ 2つ目のモデルファイルをコピーで作成 (k -median)

- ✓ モデルファイル [cfl.mod]をコピー ([cfl.mod]を選択し [ctrl]+[c]キー押す)
- ✓ ペースト(そのまま[ctrl]+[v]キー押す)
- ✓ ファイル名を [fl-km.mod] として保存
- ✓ 中身を修正する

➤ 解く

- ✓ データファイルは共通: [flex1.dat]を使う

k -center定式化
(f_j なし, k あり)

➤ 3つ目のモデルファイルをコピーで作成 (k -center)

- ✓ モデルファイル [fl-km.mod]をコピー ([fl-km.mod]を選択し [ctrl]+[c]押す)
- ✓ ペースト(そのまま [ctrl]+[v] キー押す)
- ✓ ファイル名を [fl-cm.mod] として保存
- ✓ 中身を修正する

➤ 解く

- ✓ データファイルは共通: [flex1.dat]を使う

施設配置問題を CPLEXで解く

➤ fl-km.mod

```
int i_max = ...; // 需要点の数
int j_max = ...; // 施設候補点の数
int k = ...; // 配置したい施設数
```

```
range I = 1..i_max; // 需要点集合の範囲
range J = 1..j_max; // 施設候補地集合の範囲
```

```
float d[I,J] = ...; // 需要点i が施設j を使う場合の利用コスト
int w[I] = ...; // 需要点i の重み
int c[J] = ...; // 施設j の容量
int f[J] = ...; // 施設j の設置コスト
```

```
dvar int x[I,J] in 0..1; // 需要点i が施設j を利用するかどうかの0-1変数
dvar int y[J] in 0..1; // 候補地j に施設を配置するかどうかの0-1変数
```

```
minimize
    sum(i in I) sum(j in J) d[i,j]*x[i,j];
subject to {
    forall(i in I) {
        sum(j in J) x[i,j] == 1;
    }
    forall(j in J) {
        sum(i in I) w[i]*x[i,j] <= c[j]*y[j];
    }
    forall(i in I) {
        forall(j in J) {
            x[i,j] <= y[j];
        }
    }
    sum(j in J) y[j] == k;
}
```

k -median定式化
(f なし, k あり)

CFLと異なる記述

施設配置問題をCPLEXで解く

k -median定式化
(f_j なし, k あり)

➤ 結果([解]タブ)

```
// solution (optimal) with objective 35  
// Quality Incumbent solution:  
// MILP objective  
// MILP solution norm |x| (Total, Max)  
// MILP solution error (Ax=b) (Total, Max)  
// MILP x bound error (Total, Max)  
// MILP x integrality error (Total, Max)  
// MILP slack bound error (Total, Max)  
//
```

```
x = [  
  0  0  1  0  0  
  1  0  0  0  0  
  1  0  0  0  0  
  0  0  1  0  0  
  0  0  1  0  0  
  0  0  1  0  0  
  0  0  0  0  1  
  1  0  0  0  0  
  0  0  0  0  1  
  0  0  0  0  1  
];  
y = [  
  1  0  1  0  1  
];
```

最適値 = 35

3.5000000000e+01

1.30000e+01 1.00000e+00

0.00000e+00

0.00000e+00

0.00000e+00

0.00000e+00

	候補	1	2	3	4	5
需要	c_j	120	130	125	135	125
	f_j	90	50	70	80	60
1	30	7	6	6	9	9
2	25	3	5	7	8	5
3	40	3	7	8	5	4
4	35	2	7	1	7	9
5	25	8	7	6	7	9
6	20	3	4	2	8	4
7	30	9	9	6	5	7
8	45	2	9	7	1	5
9	30	8	9	7	7	4
10	50	8	6	6	2	1

施設配置問題を CPLEXで解く

➤ fl-kc.mod

k -center定式化
(f_j なし, k あり)

```
int i_max = ...; // 需要点の数
int j_max = ...; // 施設候補点の数
int k = ...; // 配置したい施設数
range I = 1..i_max; // 需要点集合の範囲
range J = 1..j_max; // 施設候補地集合の範囲
float d[I,J] = ...; // 需要点i が施設j を使う場合の利用コスト
int w[I] = ...; // 需要点i の重み
int c[J] = ...; // 施設j の容量
int f[J] = ...; // 施設j の設置コスト

dvar int x[I,J] in 0..1; // 需要点i が施設j を利用するかどうかの0-1変数
dvar int y[J] in 0..1; // 候補地j に施設を配置するかどうかの0-1変数
dvar float t; // minimax 用変数

minimize
  t;
subject to {
  forall(i in I) {
    sum(j in J) x[i,j] == 1;
  }
  forall(j in J) {
    sum(i in I) w[i]*x[i,j] <= c[j]*y[j];
  }
  forall(i in I) {
    forall(j in J) {
      x[i,j] <= y[j];
      d[i,j]*x[i,j] <= t;
    }
  }
  sum(j in J) y[j] == k;
}
```

k -medianと
異なる記述

施設配置問題をCPLEXで解く

k -center定式化
(f_j なし, k あり)

➤ 結果([解]タブ)

```
// solution (optimal) with objective 6  
// Quality Incumbent solution:  
// MILP objective  
// MILP solution norm |x| (Total, Max)  
// MILP solution error (Ax=b) (Total, Max)  
// MILP x bound error (Total, Max)  
// MILP x integrality error (Total, Max)  
// MILP slack bound error (Total, Max)  
//
```

```
t = 6;  
x = [  
  0 0 1 0 0  
  1 0 0 0 0  
  1 0 0 0 0  
  1 0 0 0 0  
  0 0 1 0 0  
  1 0 0 0 0  
  0 0 1 0 0  
  0 0 0 0 1  
  0 0 0 0 1  
  0 0 0 0 1  
  1 0 1 0 1  
];  
y = [  
  1 0 1 0 1  
];
```

最適値 = 35

6.0000000000e+00

1.90000e+01 6.00000e+00

0.00000e+00

0.00000e+00

0.00000e+00

0.00000e+00

	候補	1	2	3	4	5
需要	c_j	120	130	125	135	125
	f_j	90	50	70	80	60
1	30	7	6	6	9	9
2	25	3	5	7	8	5
3	40	3	7	8	5	4
4	35	2	7	1	7	9
5	25	8	7	6	7	9
6	20	3	4	2	8	4
7	30	9	9	6	5	7
8	45	2	9	7	1	5
9	30	8	9	7	7	4
10	50	8	6	6	2	1

施設配置問題をgurobiで解く

➤ 問題(ex1)を python & gurobi で記述(データ生成部分①)

```
# coding: Shift_JIS
```

```
from gurobipy import *
```

①

```
# ##### 例題設定 #####
```

```
def make_data_ex1():
```

```
    k = 3
```

```
    w = [30,25,40,35,25,20,30,45,30,50]
```

```
    c = [120,130,125,135,125]
```

```
    f = [90,50,70,80,60]
```

```
    d = {(0,0):7,(0,1):6,(0,2):6,(0,3):9,(0,4):9,  
         (1,0):3,(1,1):5,(1,2):7,(1,3):8,(1,4):5,  
         (2,0):3,(2,1):7,(2,2):8,(2,3):5,(2,4):4,  
         (3,0):2,(3,1):7,(3,2):1,(3,3):7,(3,4):9,  
         (4,0):8,(4,1):7,(4,2):6,(4,3):7,(4,4):9,  
         (5,0):3,(5,1):4,(5,2):2,(5,3):8,(5,4):4,  
         (6,0):9,(6,1):9,(6,2):6,(6,3):5,(6,4):7,  
         (7,0):2,(7,1):9,(7,2):7,(7,3):1,(7,4):5,  
         (8,0):8,(8,1):9,(8,2):7,(8,3):7,(8,4):4,  
         (9,0):8,(9,1):6,(9,2):6,(9,3):2,(9,4):1,
```

```
    } # 需要点iが施設jを使う場合の利用コスト
```

```
    return k,w,c,f,d
```

```
# 必要施設数(設置施設数)  
# 需要点iの重み  
# 施設jの容量  
# 施設jの設置コスト
```

※3種の定式化で共通
で使うので, それぞれで
使うデータと使わない
データがあることに注意

施設配置問題をgurobiで解く

CFL定式化
(f_j あり, k なし)

➤ 問題(ex1)をpython &

定式化

```
def cfl(w,c,f,d):
```

```
    mod = Model("facility location problem:CFL")
```

変数設定

```
x,y = {},{}
```

```
for j in range(len(c)):
```

```
    y[j] = mod.addVar(vtype="B", name="y(%s)" % j)
```

```
    for i in range(len(w)):
```

```
        x[i,j] = mod.addVar(vtype="B", name="x(%s,%s)" % (i,j))
```

```
mod.update()
```

制約条件の設定

```
for i in range(len(w)):
```

```
    mod.addConstr(quicksum(x[i,j] for j in range(len(c))) == 1)
```

```
for j in range(len(c)):
```

```
    mod.addConstr(quicksum(w[i]*x[i,j] for i in range(len(w))) <= c[j]*y[j])
```

```
for j in range(len(w)):
```

```
    for i in range(len(c)):
```

```
        mod.addConstr(x[i,j] <= y[j])
```

目的関数の設定

```
mod.setObjective(quicksum(d[i,j]*x[i,j] for i in range(len(w)) for j in  
range(len(c))) + quicksum(f[j]*y[j] for j in range(len(c))), GRB.MINIMIZE)
```

```
mod.update()
```

```
mod.__data = x,y
```

```
return mod
```

1つのファイル「cfl.py」に
①②③の順に記述して保存

実行

```
if __name__=="__main__":
```

```
    k,w,c,f,d = make_data_ex1() # データ生成
```

```
    mod = cfl(w,c,f,d) # モデル生成
```

```
    mod.write("cflex1.lp") # lpファイルに出力
```

```
    mod.optimize() # 最適化
```

```
    print("¥n optimal value = ", mod.ObjVal)
```

```
    mod.printAttr('X') # 最適解をsolファイルに出力
```

```
    mod.write("cflex1.sol") # 最適解をsolファイルに出力
```

③

②

施設配置問題をgurobiで解く

➤ Pythonファイル(cfl.py)をgurobi上で実行し, 解く

➤ [Win]+[R] キー で [ファイル名を指定して実行] d-boxを起動する

➤ 枠内で `cmd` `[Enter]`

➤ コマンドプロンプト command prompt のウィンドウ(黒い画面)が起動する

➤ コマンドプロンプト内でコマンド(命令文)を打って順次命令を実行する

(1) 実行ファイルがあるフォルダに移動する

```
cd [フォルダへのパス] [Enter]
```

(2) 以下の命令文を打って gurobi を起動する

```
gurobi [Enter]
```

➤ 起動した gurobi 内で, 以下の命令文を打って問題を解く

```
gurobi> exec( open("cfl.py").read() ) [Enter]
```

※python3系の場合

※python2系の場合の命令文は以下

```
gurobi> execfile("cfl.py") [Enter]
```

施設配置問題をgurobiで解く

CFL定式化
(f_j あり, k なし)

実行結果

```
optimal value = 223.0
```

Variable	X
y(1)	1
x(0,1)	1
x(1,1)	1
x(2,1)	1
y(2)	1
x(3,2)	1
x(4,2)	1
x(5,2)	1
x(6,2)	1
y(4)	1
x(7,4)	1
x(8,4)	1
x(9,4)	1

```
gurobi> _
```

```
gurobi> exec(open("cfl.py").read())
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)
Thread count: 10 physical cores, 20 logical processors, using up to 20 threads
Optimize a model with 65 rows, 55 columns and 205 nonzeros
Model fingerprint: 0x4333b926
Variable types: 0 continuous, 55 integer (55 binary)
Coefficient statistics:
  Matrix range      [1e+00, 1e+02]
  Objective range   [1e+00, 9e+01]
  Bounds range      [1e+00, 1e+00]
  RHS range         [1e+00, 1e+00]
Found heuristic solution: objective 404.0000000
Presolve time: 0.00s
Presolved: 65 rows, 55 columns, 205 nonzeros
Variable types: 0 continuous, 55 integer (55 binary)
Found heuristic solution: objective 385.0000000

Root relaxation: objective 1.995250e+02, 41 iterations, 0.00 seconds (0.00 work)

   Nodes      |   Current Node   |   Objective Bounds   |         Work
  Expl Unexpl |  Obj Depth IntInf | Incumbent BestBd   Gap | It/Node Time
-----
    0     0   199.52500   0   13   385.00000   199.52500   48.2%    -    0s
H    0     0             cutoff   0             223.00000   199.52500   10.5%    -    0s
    0     0             cutoff   0             223.00000   223.00000   0.00%    -    0s

Cutting planes:
  Gomory: 5
  MIR: 1
  StrongCG: 1
  Zero half: 4

Explored 1 nodes (48 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 20 (of 20 available processors)

Solution count 3: 223 385 404

Optimal solution found (tolerance 1.00e-04)
Best objective 2.2300000000000e+02, best bound 2.2300000000000e+02, gap 0.0000%
```

施設配置問題をgurobiで解く

k -median定式化
(f なし, k あり)

➤ 問題(ex1)をpython &

定式化

def flkm(k,w,c,d):

mod = Model("facility location problem: FL_k-median")

変数設定

x,y = {},{}

for j in range(len(c)):

y[j] = mod.addVar(vtype="B", name="y(%s)" % j)

for i in range(len(w)):

x[i,j] = mod.addVar(vtype="B", name="x(%s,%s)" % (i,j))

mod.update()

制約条件の設定

for i in range(len(w)):

mod.addConstr(quicksum(x[i,j] for j in range(len(c))) == 1)

for j in range(len(c)):

mod.addConstr(quicksum(w[i]*x[i,j] for i in range(len(w))) <= c[j]*y[j])

for j in range(len(w)):

for i in range(len(c)):

mod.addConstr(x[i,j] <= y[j])

mod.addConstr(quicksum(y[j] for j in range(len(c))) == k)

目的関数の設定

mod.setObjective(quicksum(d[i,j]*x[i,j] for i in range(len(w)) for j in

range(len(c))), GRB.MINIMIZE)

mod.update()

mod.__data = x,y

return mod

1つのファイル「fl-km.py」に
①②③の順に記述して保存

実行

if __name__=="__main__":

k,w,c,f,d = make_data_ex1() # データ生成

mod = flkm(k,w,c,d) # モデルの生成

mod.write("fl-kmex1.lp") # lpファイルの書き出し

mod.optimize() # 最適化の実行

print("¥n optimal value = ", mod.ObjVal)

mod.printAttr('X') # 最適解の出力

mod.write("fl-kmex1.sol") # 最適解の保存

③

施設配置問題をgurobiで解く

k -median定式化
(f_j なし, k あり)

実行結果

optimal value = 35.0

Variable	X
y(0)	1
x(1,0)	1
x(2,0)	1
x(7,0)	1
y(2)	1
x(0,2)	1
x(3,2)	1
x(4,2)	1
x(5,2)	1
y(4)	1
x(6,4)	1
x(8,4)	1
x(9,4)	1

gurobi>

```
gurobi> exec(open("fl-km.py").read())
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)
Thread count: 10 physical cores, 20 logical processors, using up to 20 threads
Optimize a model with 66 rows, 55 columns and 210 nonzeros
Model fingerprint: 0x980ec5f3
Variable types: 0 continuous, 55 integer (55 binary)
Coefficient statistics:
  Matrix range      [1e+00, 1e+02]
  Objective range   [1e+00, 9e+00]
  Bounds range      [1e+00, 1e+00]
  RHS range         [1e+00, 3e+00]
Found heuristic solution: objective 54.0000000
Presolve time: 0.00s
Presolved: 66 rows, 55 columns, 210 nonzeros
Variable types: 0 continuous, 55 integer (55 binary)
Found heuristic solution: objective 51.0000000

Root relaxation: objective 3.445238e+01, 23 iterations, 0.00 seconds (0.00 work units)

   Nodes      |   Current Node   |   Objective Bounds   |   Work
  Expl Unexpl |  Obj  Depth IntInf | Incumbent  BestBd   Gap   | It/Node Time
-----
    0     0    34.45238    0   4    51.00000    34.45238   32.4%   -    0s
    H     0     0             42.0000000    34.45238   18.0%   -    0s
    H     0     0             36.0000000    34.45238    4.30%   -    0s
    H     0     0             35.0000000    34.45238    1.56%   -    0s
    0     0    34.45238    0   4    35.00000    34.45238    1.56%   -    0s

Explored 1 nodes (23 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 20 (of 20 available processors)

Solution count 5: 35 36 42 ... 54

Optimal solution found (tolerance 1.00e-04)
Best objective 3.500000000000e+01, best bound 3.500000000000e+01, gap 0.0000%
```

施設配置問題をgurobiで解く

k -center定式化
(f_j なし, k あり)

➤ 問題(ex1)を python &

定式化

```
def flkc(k,w,c,d):  
    mod = Model("facility location problem: FL_k-center")
```

変数設定

```
x,y = {},{}  
for j in range(len(c)):  
    y[j] = mod.addVar(vtype="B", name="y(%s)" % j)  
    for i in range(len(w)):  
        x[i,j] = mod.addVar(vtype="B", name="x(%s,%s)" % (i,j))  
t = mod.addVar(vtype="C", name="t")  
mod.update()
```

制約条件の設定

```
for i in range(len(w)):  
    mod.addConstr(quicksum(x[i,j] for j in range(len(c))) == 1)  
for j in range(len(c)):  
    mod.addConstr(quicksum(w[i]*x[i,j] for i in range(len(w))) <= c[j]*y[j])  
for j in range(len(w)):  
    for i in range(len(c)):  
        mod.addConstr(x[i,j] <= y[j])  
        mod.addConstr(d[i,j]*x[i,j] <= t)  
mod.addConstr(quicksum(y[j] for j in range(len(c))) == k)
```

目的関数の設定

```
mod.setObjective(t, GRB.MINIMIZE)  
mod.update()  
mod.__data = x,y  
return mod
```

1つのファイル「fl-kc.py」に
①②③の順に記述して保存

実行

```
if __name__=="__main__":
```

③

```
    k,w,c,f,d = make_data_ex1() # データの読み込み  
    mod = flkc(k,w,c,d) # モデルの生成  
    mod.write("fl-kcex1.lp") # lpファイルの書き出し  
    mod.optimize() # 最適化  
    print("¥n optimal value = ", mod.ObjVal) # 最適値の出力  
    mod.printAttr('X') # 最適解の出力  
    mod.write("fl-kcex1.sol") # 最適解の書き出し
```

施設配置問題をgurobiで解く

k -center定式化
(f_j なし, k あり)

実行結果

```
optimal value = 6.0
```

Variable	X
y(0)	1
x(1,0)	1
x(3,0)	1
x(7,0)	1
y(2)	1
x(0,2)	1
x(4,2)	1
x(5,2)	1
x(6,2)	1
y(4)	1
x(2,4)	1
x(8,4)	1
x(9,4)	1
t	6

```
gurobi> _
```

```
gurobi> exec(open("fl-kc.py").read())
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)
Thread count: 10 physical cores, 20 logical processors, using up to 20 threads
Optimize a model with 116 rows, 56 columns and 310 nonzeros
Model fingerprint: 0xf99b88f6
Variable types: 1 continuous, 55 integer (55 binary)
Coefficient statistics:
  Matrix range      [1e+00, 1e+02]
  Objective range   [1e+00, 1e+00]
  Bounds range      [1e+00, 1e+00]
  RHS range         [1e+00, 3e+00]
Found heuristic solution: objective 9.0000000
Presolve removed 30 rows and 0 columns
Presolve time: 0.00s
Presolved: 86 rows, 56 columns, 255 nonzeros
Variable types: 0 continuous, 56 integer (55 binary)

Root relaxation: objective 6.000000e+00, 23 iterations, 0.00 seconds (0.00 work units)

   Nodes      |   Current Node   |   Objective Bounds   |   Work
  Expl Unexpl |  Obj  Depth IntInf | Incumbent  BestBd   Gap   | It/Node Time
-----
    0     0    6.00000    0   2    9.00000    6.00000   33.3%   -    0s
H    0     0           7.0000000    6.00000   14.3%   -    0s
H    0     0    6.0000000    6.00000   0.00%   -    0s
    0     0    6.00000    0   2    6.00000    6.00000   0.00%   -    0s

Explored 1 nodes (23 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 20 (of 20 available processors)

Solution count 3: 6 7 9

Optimal solution found (tolerance 1.00e-04)
Best objective 6.000000000000e+00, best bound 6.000000000000e+00, gap 0.0000%
```

【演習】施設配置問題を解く

➤ 施設配置問題 facility location problem (ex2)

- 需要点 $n=30$, 施設配置候補点 $m=10$
 - 需要点 i と候補点 j 間の距離 d_{ij} が所与 (右表)
 - 需要点の重み w_i (人口)
 - 施設容量 c_j (収容数)
 - 施設設置コスト f_j
 - 施設設置数 $k=4$ カ所
-
- CFL 定式化 (容量あり FL) で解く
 - f_j あり, k なし
 - k -median 定式化 (距離総和を最小化) で解く
 - f_j なし, k あり
 - k -center 定式化 (距離最長を最小化) で解く
 - f_j なし, k あり

需要点重み w_i
施設容量 c_j , 設置コスト f_j
需要点 $\leftrightarrow$ 候補点 距離 d_{ij}

	候補	1	2	3	...	10
需要	c_j					
	f_j					
1						
2						
3						
4						
5						
6						
7						
8						
...						
30						