02

第2章

施設配置問題

施設配置問題を例に、良い定式化と悪い定式化を学ぶ

この章の内容

1. 容量制約付き施設配置問題
2. 強い定式化と弱い定式化
3. k-メディアン問題
4. k-センター問題：標準定式化
5. k-センター問題：二分探索
6. k-センター問題：距離の候補

2.1

容量制約付き施設配置問題

どの倉庫を開設し、どの経路で輸送すれば、年間の総費用が最小になるか？

例題：スポーツ用品販売チェイン

- 5つの顧客（需要地点）に3つの施設（倉庫）から輸送する
- 施設の年間リース費用を開設費用と呼ぶ
- 施設の年間取り扱い可能量を容量と呼ぶ
- 顧客への輸送費用と各顧客の年間需要量は既知
- どの施設を開設し、どの輸送経路を選ぶか

! ポイント

開設費用と輸送費用の和（年間の総費用）を最小にしたい。輸送問題に「施設を開くか否か」の決定が加わった問題である。

施設配置問題のデータ

上段：顧客 i の年間需要量 d[i]、下段：輸送費用 c[i,j]・開設費用 f[j]・容量 M[j]

	顧客1	顧客2	顧客3	顧客4	顧客5	f[j]	M[j]
需要量 d[i]	80	270	250	160	180		
施設1	4	5	6	8	10	1000	500
施設2	6	4	3	5	8	1000	500
施設3	9	7	4	3	4	1000	500

施設の行の数値は、施設 j から各顧客へ1単位運ぶ輸送費用 $c[i,j]$

混合整数最適化による定式化

x[i,j]：顧客 i の需要を施設 j が満たす量（連続）、y[j]：施設 j を開設するとき 1

$$
\begin{aligned}
\text{minimize} \quad & \sum_{j \in J} f_j y_j + \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} & \\
\text{subject to} \quad & \sum_{j \in J} x_{ij} = d_i & \forall i \in I \\
& \sum_{i \in I} x_{ij} \leq M_j y_j & \forall j \in J \\
& x_{ij} \leq d_i y_j & \forall i \in I; j \in J \\
& x_{ij} \geq 0 & \forall i \in I; j \in J \\
& y_j \in \{0, 1\} & \forall j \in J
\end{aligned}
$$

- 目的：開設費用と輸送費用の和
- 第1制約：需要をすべて満たす
- 第2制約：施設の容量制約
- 第3制約：強制式（定式化を強化）

AMPLによるモデル

flp.mod

```
set I;                  # 顧客の集合
set J;                  # 施設の候補地の集合
param d {I} >= 0;       # 顧客 i の需要量
param M {J} >= 0;       # 施設 j の容量
param f {J} >= 0;       # 施設 j の開設固定費用
param c {I,J} >= 0;     # 顧客 i と施設 j の間の輸送費用

var x {I,J} >= 0;       # 施設 j から顧客 i への輸送量
var y {J} binary;       # 施設 j を開設するとき 1

minimize Cost:
   sum {j in J} f[j] * y[j]
   + sum {i in I, j in J} c[i,j] * x[i,j];

subject to Demand {i in I}:
   sum {j in J} x[i,j] = d[i];

subject to Capacity {j in J}:
   sum {i in I} x[i,j] <= M[j] * y[j];

subject to Strong {i in I, j in J}:
   x[i,j] <= d[i] * y[j];
```

Pythonからモデルを解く

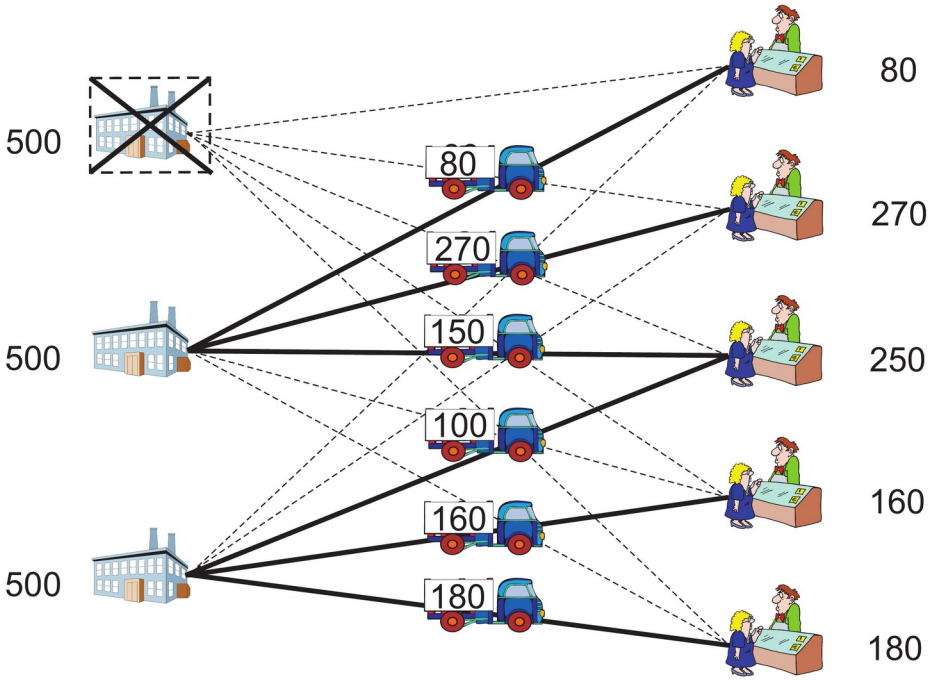
flp.py

```
def flp(customer_df, facility_df, c):
    ampl = AMPL()
    ampl.read("flp.mod")
    ampl.set_data(customer_df, "I")
    ampl.set_data(facility_df, "J")
    ampl.param["c"] = c
    return ampl

if __name__ == "__main__":
    customer_df, facility_df, c = make_data()
    ampl = flp(customer_df, facility_df, c)
    ampl.option["solver"] = "highs"
    ampl.solve()
    assert ampl.solve_result == "solved"
    print("Opt.value=",
          ampl.get_objective("Cost").value())
```

- set_data で集合とパラメータを同時に渡す
  - set_index の列が集合、残りがパラメータ
- 変数・制約・目的関数は flp.mod 側にある
- ソルバーは HiGHS

例題の最適解



施設配置問題の例題と最適解

- 最適費用は 5610
- 施設 2, 3 を開設、施設 1 は閉鎖
- 顧客3の需要は施設2と3で分担

最適輸送量

施設	顧客1	顧客2	顧客3	顧客4	顧客5	開設の可否
1	0	0	0	0	0	閉鎖
2	80	270	150	0	0	開設
3	0	0	100	160	180	開設

施設2は 500、施設3は 440 を出荷し、どちらも容量 500 以内に収まる

2.2

強い定式化と弱い定式化

同じ問題でも、定式化によってソルバーの解きやすさが大きく変わる

Big M の落とし穴

- 容量制約なし施設配置問題：容量が実質的にない
  - 容量制約の右辺を非常に大きな数 M にする
- 強制式を除いて大きな M を用いると、規模が大きくなると急に解けなくなる
- Big M は論理条件を楽に書けるので乱発しがち
- 線形最適化による限界値はソルバーの命綱

! モデリングのコツ

大きな数 M はなるべく小さな値に設定しなければならない。M=9 999 999 などと設定するのは論外である。

強い定式化の定義

線形最適化緩和問題の実行可能領域の包含関係によって強弱を定義する

P_A, P_B : 定式化 A, B の線形最適化緩和問題の実行可能領域

$P_A \subset P_B \implies$ 定式化 A は B より強い（B は A より弱い）

定式化 A（強い）: $x_{ij} \leq d_i y_j \qquad \forall i \in I; j \in J$

定式化 B（弱い）: $\sum_{i \in I} x_{ij} \leq \left(\sum_{i \in I} d_i \right) y_j \qquad \forall j \in J$

A の制約の和 $\implies P_A \subseteq P_B$ （真に強い : $\exists (x, y) \in P_B \setminus P_A$）

- 実行可能領域が狭い方が、より強い限界値（最小化なら下界）を与える
- 定式化 B の制約は A の制約を顧客について足し合わせたもの
- 真に強いことは、B に含まれ A に含まれない緩和解を示せばよい

どちらの定式化を使うべきか

強い定式化

- 線形緩和の限界値が良い
- 列挙木の増大を抑えられる
- 制約が多い（nm 本）
- 緩和問題を解く時間は長い

弱い定式化

- 制約が少ない（m 本）
- 緩和問題を速く解ける
- 限界値が悪く列挙木が増大しがち
- 小規模ならソルバーの機能で十分な場合も

2.3 k-メディアン問題

顧客から最も近い施設への距離の「合計」を最小にする

k-メディアン問題の定式化

x[i,j]：顧客 i を施設 j に割り当てるとき 1、y[j]：施設 j を開設するとき 1

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} & \\
\text{subject to} \quad & \sum_{j \in J} x_{ij} = 1 & \forall i \in I \\
& \sum_{j \in J} y_j = k & \\
& x_{ij} \leq y_j & \forall i \in I; j \in J \\
& x_{ij} \in \{0, 1\} & \forall i \in I; j \in J \\
& y_j \in \{0, 1\} & \forall j \in J
\end{aligned}
$$

- 顧客はいずれかの施設に割り当てる
- 開設する施設はちょうど k 個
- 開設した施設からのみサービス
  - n 本の和の式で代用すると弱い

AMPLによるモデル

kmedian.mod

```
set I;                    # 顧客の集合
set J;                    # 施設の候補地の集合
param c {I,J} >= 0;       # 顧客 i と施設 j の間の距離
param k > 0 integer;      # 開設する施設の数

var x {I,J} binary;
var y {J} binary;

minimize Cost:
   sum {i in I, j in J} c[i,j] * x[i,j];

subject to Assign {i in I}:
   sum {j in J} x[i,j] = 1;

subject to Open {i in I, j in J}:
   x[i,j] <= y[j];

subject to Facilities:
   sum {j in J} y[j] = k;
```

- k は解くたびに変えたいのでパラメータにする
- param ... integer は値の検査
  - 変数の整数条件とは別物
- Python 側は set と param を渡すだけ

2.4 k-センター問題

顧客から最も近い施設への距離の「最大値」を最小にする

標準定式化

最も遠い施設でサービスを受ける顧客の移動距離を表す連続変数 z を導入する

$$
\begin{aligned}
\text{minimize} \quad & z \\
\text{subject to} \quad & \sum_{j \in J} x_{ij} = 1 && \forall i \in I \\
& \sum_{j \in J} y_j = k \\
& x_{ij} \leq y_j && \forall i \in I; j \in J \\
& c_{ij} x_{ij} \leq z && \forall i \in I; j \in J \\
& x_{ij} \in \{0, 1\} && \forall i \in I; j \in J \\
& y_j \in \{0, 1\} && \forall j \in J
\end{aligned}
$$

- 第1〜3制約は k-メディアン問題と同じ
- 第4制約：割り当てた顧客の移動距離は z 以下
- z を最小化して「最大値の最小化」を実現
  - ソルバーが苦手とする目的関数

k-センター問題のモデル

kcenter.mod

```
set I;
set J;
param c {I,J} >= 0;
param k > 0 integer;

var x {I,J} binary;
var y {J} binary;
var z >= 0;             # 最も遠い顧客の移動距離

minimize MaxDist: z;

subject to Assign {i in I}:
   sum {j in J} x[i,j] = 1;

subject to Open {i in I, j in J}:
   x[i,j] <= y[j];

subject to Dist {i in I, j in J}:
   c[i,j] * x[i,j] <= z;

subject to Facilities:
   sum {j in J} y[j] = k;
```

欄外ゼミナール：線形最適化の小技

新しい変数を導入して、通常の線形最適化に帰着する

1 最大値の最小化

$3x_1+4x_2 \le z$, $2x_1+7x_2 \le z$ を加えて z を最小化。AMPL では max(...) と書くだけ

2 絶対値（2変数）

$x = y - z$（$y, z \ge 0$）と置き、目的関数の $|x|$ を $y + z$ で置き換える

3 絶対値（1変数）

$z \ge x$ と $z \ge -x$ を加えて z を最小化すれば、z は $|x|$ と一致する

4 AMPL の abs

z = abs(x) と書ける。ただし z の下界をあらかじめ 0 にしておく

最大値を最小化する目的関数は避けよ

- 整数最適化問題は分枝限定法で解かれる
- 最大値の最小化（最小値の最大化）では双対ギャップが大きくなりがち
  - 双対ギャップ＝下界と上界の差
- 求解時間が非常に長くなる
- 途中で中断したときの暫定解の質が悪い

! モデリングのコツ

実際問題をモデル化する際には、最大値を最小化するような定式化はできるだけ避けた方が良い。

二分探索の部分問題：k点被覆問題

距離が θ 以下の枝からなるグラフ上で、サイズ k の被覆が存在するかを判定する

$$a_{ij} = 1\ (c_{ij} \le \theta\ \text{のとき}),\quad 0\ (\text{それ以外})$$

$$\begin{aligned}
\text{minimize} \quad & \sum_{i \in I} z_i \\
\text{subject to} \quad & \sum_{j \in J} a_{ij} y_j + z_i \ge 1 \quad \forall i \in I \\
& \sum_{j \in J} y_j = k \\
& z_i \in \{0,1\} \qquad \forall i \in I \\
& y_j \in \{0,1\} \qquad \forall j \in J
\end{aligned}$$

- a[i,j]：距離が θ 以下のとき 1、それ以外 0（隣接関係）
- z[i]：点 i が被覆されないとき 1
- 目的：被覆されない顧客の数を最小化
- 最適値が 0 なら k-センターの最適値は θ 以下

k点被覆問題のモデル

kcover.mod

```
set I;
set J;
param c {I,J} >= 0;
param k > 0 integer;
param theta default 0;   # 被覆とみなす距離のしきい値

var x {i in I, j in J} integer >= 0,
                       <= (if c[i,j] <= theta then 1 else 0);
var y {J} binary;        # 施設 j を選ぶとき 1
var z {I} binary;        # 顧客 i が被覆されないとき 1

minimize Uncovered:
   sum {i in I} z[i];

subject to Cover {i in I}:
   sum {j in J} x[i,j] + z[i] >= 1;

subject to Open {i in I, j in J}:
   x[i,j] <= y[j];

subject to Facilities:
   sum {j in J} y[j] = k;
```

二分探索法

kcenter.py

```
def solve_kcenter(I, J, c, k):
    ampl = AMPL()
    ampl.read("kcover.mod")
    ampl.set["I"] = I
    ampl.set["J"] = J
    ampl.param["c"] = c
    ampl.param["k"] = k
    ampl.option["solver"] = "highs"
    LB = 0
    UB = max(c[i,j] for (i,j) in c)
    while UB - LB > 1.e-4:
        theta = (UB + LB) / 2.
        ampl.param["theta"] = theta
        ampl.solve(verbose=False)
        assert ampl.solve_result == "solved"
        infeas = ampl.get_objective("Uncovered").value()
        if infeas > 0:
            LB = theta
        else:
            UB = theta
    return UB
```

- UB:=距離の最大値、LB:=0 から始める
- θ:=(UB+LB)/2 で k点被覆問題を解く
- 最適値が 0 なら UB:=θ、そうでなければ LB:=θ
- 反復内では theta を代入して解き直すだけ

標準定式化はなぜ遅いか

- 原因は線形緩和が極端に弱いこと
- $x[i,j]$ を小数にすれば、各顧客をすべての施設に少しずつ割り当てられる
  - $c[i,j]\,x[i,j] \le z$ の左辺はいくらでも小さくなる
  - 線形緩和の最適値はほぼ 0
- 最適値はいずれかの距離 $c[i,j]$ に一致する
- この性質を定式化そのものに組み込む

! ポイント

二分探索法が利用した性質を、定式化に直接組み込むことで、最大値の最小化を避けられる。

距離の候補を用いた定式化

異なる距離の候補を昇順にソートし、半径がその値以上かを表す 0-1 変数 u[q] を導入する

$D_0 < D_1 < \cdots < D_K$: 異なる距離の候補を昇順に整列

$u_q \in \{0,1\}$: 半径が D_q 以上のとき 1

$$
\begin{aligned}
\text{minimize} \quad & D_0 + \sum_{q=1}^{K} (D_q - D_{q-1}) u_q \\
\text{subject to} \quad & 1 \le \sum_{j \in J} y_j \le k \\
& u_q + \sum_{j : c_{ij} < D_q} y_j \ge 1 && \forall i \in I; q = 1, \ldots, K \\
& u_q \ge u_{q+1} && \forall q = 1, \ldots, K-1 \\
& y_j \in \{0,1\} && \forall j \in J \\
& u_q \in \{0,1\} && \forall q = 1, \ldots, K
\end{aligned}
$$

- 割当変数 x は不要、変数は |J|+K 個に削減
- Elloumi–Labbé–Pochet：第3制約を除いたモデル
- Ales–Elloumi：第3制約を加えると多くの制約が冗長
  - 各顧客で必要な候補添字集合 S[i] ∪ {K} だけ残せば十分

距離の候補を用いたモデル

kcenter_cp1.mod

```
set I;
set J;
param c {I,J} >= 0;
param k > 0 integer;
param K > 0 integer;          # 異なる距離の数 - 1
param D {0..K};               # 異なる距離を昇順に並べたもの
set S {i in I} =
   {q in 1..K-1: exists {j in J} c[i,j] = D[q]} union {K};

var y {J} binary;
var u {1..K} binary;          # 半径が D[q] 以上のとき 1

minimize Radius:
   D[0] + sum {q in 1..K} (D[q] - D[q-1]) * u[q];
subject to Facilities:
   1 <= sum {j in J} y[j] <= k;
subject to Order {q in 1..K-1}:
   u[q] >= u[q+1];
subject to Cover {i in I, q in S[i]}:
   u[q] + sum {j in J: c[i,j] < D[q]} y[j] >= 1;
```

まとめ

- ✓ 容量制約付き施設配置問題は混合整数最適化で書ける
- ✓ 強弱は線形緩和の実行可能領域の包含関係で定義する
- ✓ Big M はなるべく使わず、使うなら小さな値にする
- ✓ k-メディアン問題は距離の合計、k-センター問題は最大値を最小化
- ✓ 最大値の最小化はソルバーが苦手なので避ける
- ✓ 二分探索や距離の候補を用いた定式化で回避できる