09

第9章

多目的最適化

複数目的のトレードオフを1目的の最適化に帰着して扱う

この章の内容

1. 多目的最適化の基礎理論
2. 多目的巡回セールスマン問題
3. 線形和によるスカラー化
4. 最小ノルムによるスカラー化
5. 制約を移動させることによる方法
6. スタッフスケジューリング

9.1 多目的最適化の基礎理論

複数の目的を同時に「最小」にするとはどういう意味か？

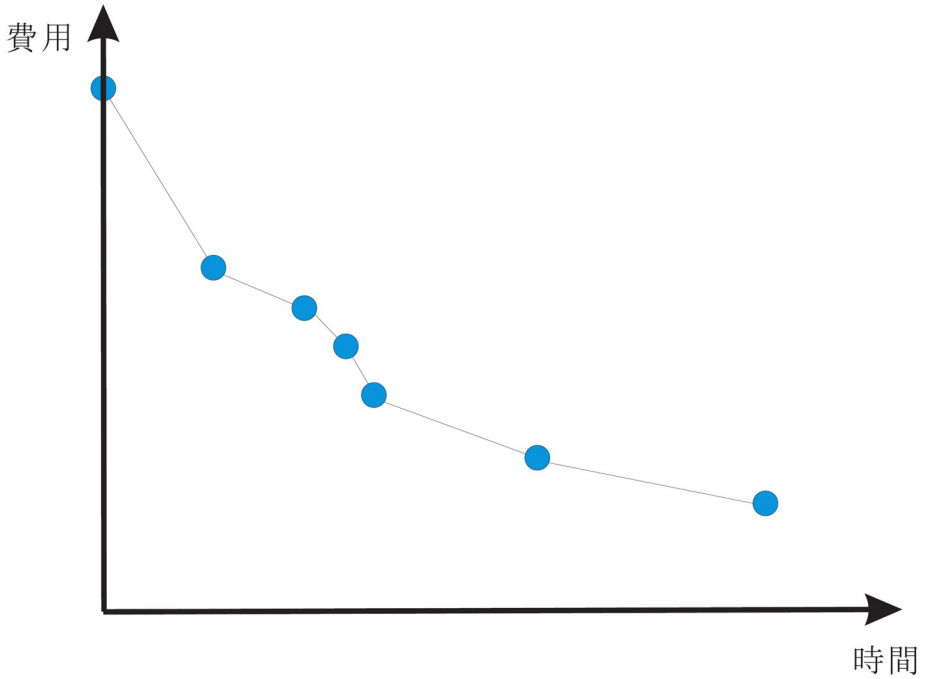
優越関係

解の集合 F と目的関数ベクトル $f\colon F \to \mathbb{R}^m$ が与えられる

$$f \prec g \;\Leftrightarrow\; f \neq g,\quad f_i \leq g_i \quad \forall i$$

- f と g が同じでなく、すべての要素で f が g 以下のとき「f は g に優越する」
- 例：f=(2,5,4), g=(2,6,8) なら f ≺ g
- f(x) ≺ f(y) のとき、解 x は解 y に優越する

非劣解（Pareto最適解）



非劣解（*Pareto*最適解）の集合

- $f(y) \prec f(x)$ となる解 y が存在しない解 x を非劣解（Pareto最適解）と呼ぶ
- 目的はすべての非劣解の集合を求めること
- その境界を有効フロンティアと呼ぶ
  - 一般には凸になるとは限らない

意思決定者と非劣解

- 非劣解の総数は非常に大きくなりうる
- すべてを列挙せず、好みに合う少数の非劣解を示すことが重要
- 選好を効用関数で表すのは実務的には極めて難しい
- 限られた入力で非劣解を提示し、入力を追加して好みの解に近づける

! ポイント

実際には、意思決定者との対話による反復過程で解を絞り込んでいく。

1目的ソルバーで多目的問題を扱う方法

複数の目的を何らかの方法で単一目的の最適化に帰着する

1 線形和スカラー化

重み λ（$\Sigma\,\lambda_i = 1$）を掛けて目的関数を足し合わせ、単一の目的関数 f_λ にする

2 理想点からの距離

意思決定者が示した理想点 f^* からの重み付き距離（の2乗）を最小化する

3 制約を移動させる方法

第1目的以外を「第 j 目的 $\leq b_j$」の制約にし、右辺 b_j を変えて最適化を繰り返す

スカラー化の目的関数

線形和：
$$f_\lambda(x) = \sum_{i=1}^{m} \lambda_i f_i(x), \quad \sum_{i=1}^{m} \lambda_i = 1$$

理想点からの距離：
$$f_\lambda(x) = \sum_{i=1}^{m} \lambda_i |f_i(x) - f_i^*|^2$$

- 距離は本来平方根をとるが、最適化するのは中身だけでよい
- スカラー化すると通常の1目的最適化問題になる

9.2

多目的巡回セールスマン問題

時間と費用のトレードオフ。無駄なく多くの代替スケジュールを作るには？

2目的巡回セールスマン問題

- n 個の点からなる有向グラフ G=(V,A)
- 枝 (i,j) 上に費用 c[i,j] と移動時間 t[i,j]
- すべての点をちょうど1回ずつ経由する巡回路
- （時間, 費用）を目的関数ベクトルとする
- すべての非劣解の集合を求める

! ポイント

製品の製造順序は巡回路、段取り時間・段取り費用は枝の時間・費用に対応する。

モデルファイル：目的関数を並べて書く

biobj_tsp.mod

```
param alpha default 0;           # 線形和スカラー化の傾き
param lambda default 0;          # 最小ノルムスカラー化の重み
param Tstar default 0;           # 理想点の時間
param Cstar default 0;           # 理想点の費用
param time_ub default Infinity;  # 時間の上限

var x {A} binary;
var u {V} >= 0, <= n-1;
var C >= 0;                      # 総移動費用
var T >= 0;                      # 総移動時間

minimize Time:   T;
minimize Cost:   C;
minimize Scalar: C + alpha * T;
minimize Ideal:  lambda*(T - Tstar)^2
              + (1-lambda)*(C - Cstar)^2;

subject to CostDef: C = sum {(i,j) in A} c[i,j] * x[i,j];
subject to TimeDef: T = sum {(i,j) in A} t[i,j] * x[i,j];
subject to TimeConstr: T <= time_ub;
```

- 目的関数を4つ書き、objective 命令で選ぶ
- 反復ごとに変わる値はすべてパラメータ
- C と T を変数で定義すると値の取り出しが簡単

Python側の準備

```python
def base_model(n, c, t):
    ampl = AMPL()
    ampl.read("biobj_tsp.mod")
    ampl.param["n"] = n
    ampl.param["c"] = c
    ampl.param["t"] = t
    ampl.option["solver"] = "highs"
    return ampl

def optimize(ampl, cand):
    ampl.solve(verbose=False)
    assert ampl.solve_result == "solved"
    cand.append((ampl.get_variable("T").value(),
                 ampl.get_variable("C").value()))
```

- base_model：モデルを読み込みデータを渡す
- optimize：解いて（時間, 費用）を候補 cand に追加
- 解プール（sol:stub）が使えるソルバーなら途中の解も候補になる

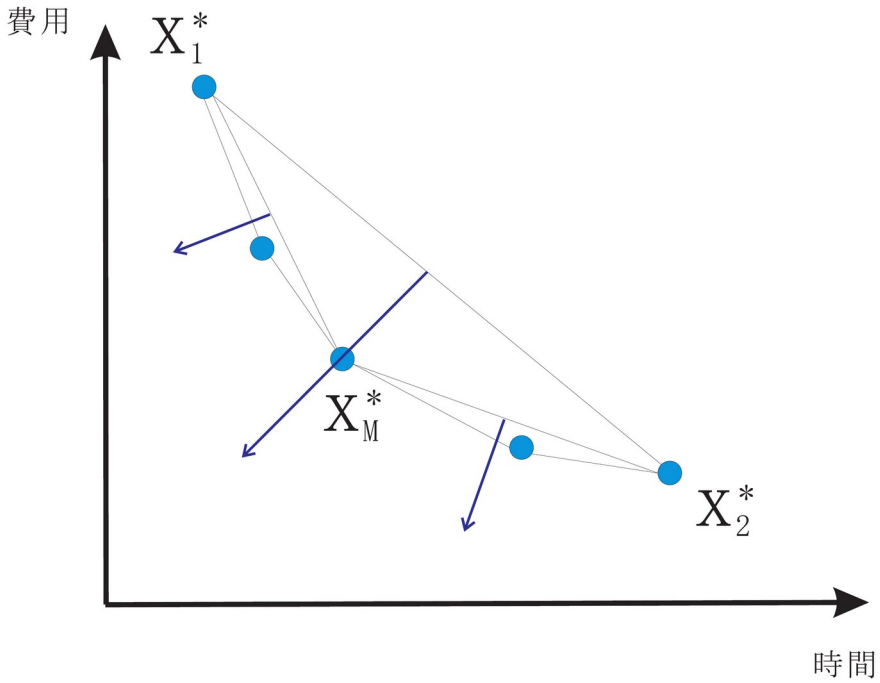
線形和によるスカラー化

最短時間巡回路 x^*_1 と最小費用巡回路 x^*_2 を結ぶ直線の傾きを使う

$$\alpha = \frac{C(x_1^*) - C(x_2^*)}{T(x_2^*) - T(x_1^*)} \quad \Longrightarrow \quad c_{ij} + \alpha\, t_{ij} \text{ を距離として巡回路 } x_M^* \text{ を求める}$$

- x^*_M が x^*_1, x^*_2 に優越されなければ新たな非劣解
- (x^*_1, x^*_M) と (x^*_M, x^*_2) の組で同じ操作を繰り返す

スカラー化による非劣解の生成



スカラー化による非劣解の部分集合の生成

- 両端点を結ぶ直線の傾きで重みを決める
- 新しい非劣解が出たら区間を2つに分けて再帰
- すべての非劣解を生成する保証はない
  - 簡単に多くの非劣解が得られる

再帰的な探索 explore

```python
def explore(ampl, C1, T1, C2, T2, front):
    if T2 <= T1:
        return front
    alpha = (C1 - C2) / (T2 - T1)
    init = list(front)
    ampl.param["alpha"] = alpha
    ampl.eval("objective Scalar;")
    optimize(ampl, front)
    front = pareto_front(front)
    if front == init:
        return front
    CM = ampl.get_variable("C").value()
    TM = ampl.get_variable("T").value()
    if TM > T1:
        front = explore(ampl, C1, T1, CM, TM, front)
    if T2 > TM:
        front = explore(ampl, CM, TM, C2, T2, front)
    return front
```

- 傾き alpha を計算してパラメータに代入
- objective Scalar; で目的関数を切り替え
- 新しい非劣解 (CM, TM) で左右を再帰探索

最小ノルムによるスカラー化

理想点 (T*, C*)：時間だけ・費用だけを最小化したときの値

$$\lambda \left(\sum t_{ij} x_{ij} - T^* \right)^2 + (1 - \lambda) \left(\sum c_{ij} x_{ij} - C^* \right)^2$$

- λ は各目的の重要性を表すパラメータ
- λ を 0 から 1 まで区分数 segments で変化させる
- モデルの目的関数 Ideal に対応

二次の目的関数とソルバー

- Ideal は凸二次関数で、変数に整数条件がある
- 混合整数二次最適化（MIQP）問題になる
- HiGHS は LP と QP に対応するが MIQP には非対応
- Gurobi、CPLEX、COPT、Xpress、Mosek、SCIP を使う

! モデルとソルバーの分離

モデルファイルは一切変更せず、ampl.option["solver"] = "gurobi" と1行書き換えるだけでよい。

制約を移動させることによる方法

```python
def solve_epsilon(n, c, t, segments):
    ampl = base_model(n, c, t)
    cand = []
    ampl.eval("objective Time;")
    optimize(ampl, cand)
    ampl.eval("objective Cost;")
    optimize(ampl, cand)
    times = [ti for (ti, ci) in cand]
    max_time = max(times)
    min_time = min(times)
    delta = (max_time - min_time) / segments
    for i in range(segments + 1):
        ampl.param["time_ub"] = max_time - delta*i
        optimize(ampl, cand)
    return cand
```

- 費用を目的関数、時間を制約にする
- 上限 time_ub を delta ずつ小さくして解く
- 制約 TimeConstr には手を触れない
  - 右辺の変更はパラメータの代入だけ

2つの方法の違い

線形和によるスカラー化

- フロンティアの凹んだ部分の解（非支持解）は見つからない
- $n=8$ のある例で非劣解 4 個

制約を移動させる方法

- 非支持解も見つけられる
- 同じ例で 4 個＋1 個を発見
- 区分数だけ最適化を繰り返す
- 刻みが粗いと解を見落とすこともある

9.3 スタッフスケジューリング

「人」がからむ複雑な制約のトレードオフをとり、みんなが納得する解を

24時間営業の弁当屋の例

- 8人のスタッフ、朝・昼・夜の3シフト（各8時間）
- 1日に高々1シフト、1週間に5日勤務・2日休み
- 日給 $7000+1000\times i$ 円、夜勤2倍、週末1.5倍
- 各シフトに2人以上を割り当てる
- 異なるシフトに移るときは必ず休日を入れる
- 昼・夜のシフトは少なくとも2日連続

ポイント

休日をシフト0とし、シフト集合 $\{0,1,2,3\}$、スタッフ $1\sim8$、期 $1\sim7$ で表す。

スタッフスケジューリングの定式化

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{8}\sum_{t=1}^{7}\sum_{j=1}^{3} c_{itj}\,x_{itj} \\
\text{subject to} \quad & \sum_{j=0}^{3} x_{itj} = 1 && \forall i,t \\
& \sum_{t=1}^{7}\sum_{j=1}^{3} x_{itj} = 5 && \forall i \\
& \sum_{i=1}^{8} x_{itj} \geq 2 && \forall t;\ j=1,2,3 \\
& x_{itj} + \sum_{k\neq j,k\geq 1} x_{i,t+1,k} \leq 1 && \forall i;\ t\leq 6;\ j=1,2,3 \\
& x_{i,t-1,j} + x_{i,t+1,j} \geq x_{itj} && \forall i,t;\ j=2,3
\end{aligned}
$$

- 各期に1つのシフト（休日含む）
- 出勤日数は5日
- 各シフト2人以上
- 翌日に異なるシフトは不可
- 昼・夜は2日以上連続

最適解が得られない！

そのまま解くと最適解が得られない。主な理由は3つ

1 **非有界（unbounded）**

制約が不完全で、目的関数値が無限に良くなれてしまう

2 **実行不可能（infeasible）**

制約がきつすぎて、すべての制約を満たす解が存在しない

3 **その他の終了**

計算時間の上限を超えた、数値的に不安定になった など

制約の逸脱を第2の目的に

「各シフト2人以上」を破ってもよいとし、不足人数 y_{tj} を導入する

$$\sum_{i=1}^{8} x_{itj} \geq 2 - y_{tj} \qquad \forall t = 1, \ldots, 7;\ j = 1, 2, 3$$

- 目的1：総費用
- 目的2：不足人数の合計 Σ y[t,j]
- 2つを目的関数とした多目的最適化問題

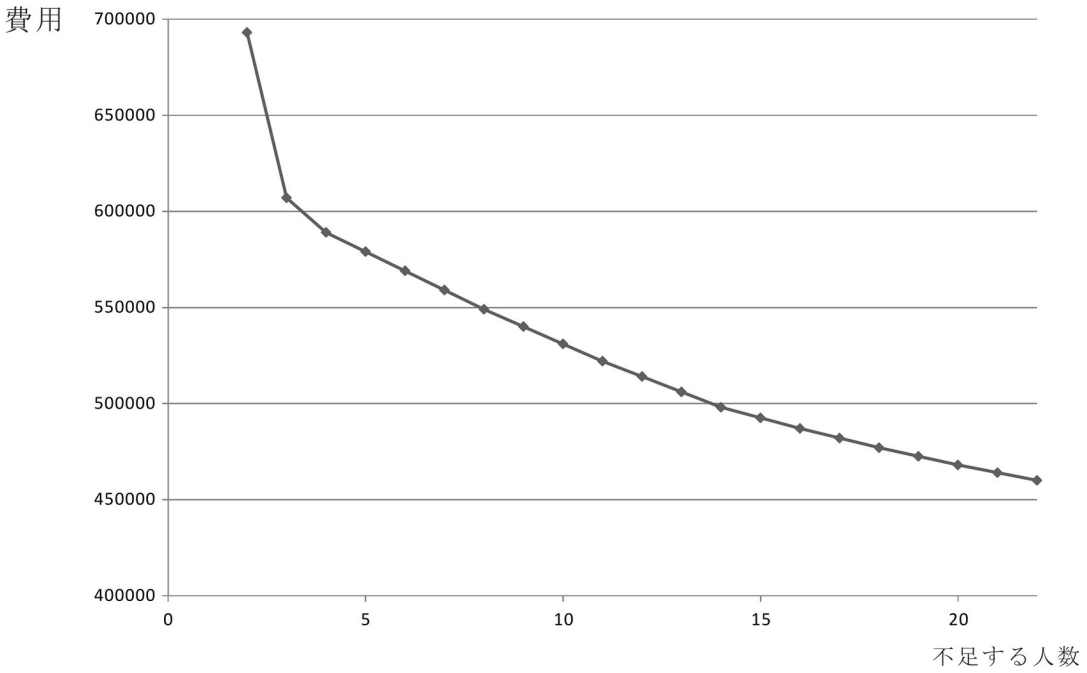
AMPLモデル（抜粋）

staff_mo.mod

```
var x {I,Ts,J} binary;       # 割り当てるとき 1
var y {Ts,J} >= 0;           # 不足するスタッフの数
var C >= 0;                  # 総費用
var U >= 0;                  # 不足するスタッフの合計
param cost_ub default Infinity;

minimize Cost:      C;
minimize Uncovered: U;

subject to CostLimit: C <= cost_ub;

subject to Cover {t in Ts, j in J: j <> 0}:
   sum {i in I} x[i,t,j] >= b[t,j] - y[t,j];

subject to SameShift {i in I, t in Ts, j in S}:
   (if t > 1 then x[i,t-1,j] else 0)
   + (if t < T then x[i,t+1,j] else 0)
      >= x[i,t,j];
```

- 費用の上限 cost_ub を動かして解く
- if-then-else で端の期の項を 0 にする
- 端の場合分けが1本の制約にまとまる

費用と不足人数の有効フロンティア



多目的スタッフスケジューリング問題の非劣解（有効フロンティア）

- 制約を移動させる方法と同様に求める
- 費用の上限を動かして不足人数を最小化
- 意思決定者はトレードオフを見て解を選べる

まとめ

- ✓ 多目的最適化の目的は非劣解（Pareto最適解）の集合を求めること
- ✓ 線形和・理想点からの距離でスカラー化して1目的に帰着する
- ✓ 制約を移動させる方法は非支持解も見つけられる
- ✓ AMPLでは目的関数を並べて書き objective 命令で切り替える
- ✓ 実行不可能な問題は制約の逸脱を第2目的にして扱える