04

第4章

グラフ最適化問題

グラフ分割・最大安定集合・グラフ彩色をAMPLで解く

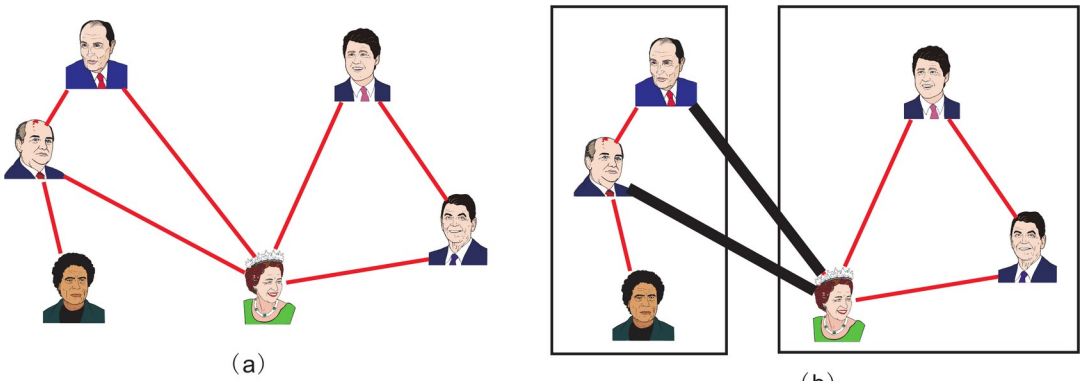
この章の内容

1. グラフ分割問題と二次の目的関数の線形化
2. 最大安定集合問題と最大クリーク問題
3. グラフ彩色問題の標準定式化と対称性の除去
4. 二分探索による解法
5. 半順序定式化と代表点定式化

4.1 グラフ分割問題

仲良しが別チームになるのを極力避けて、6人を3人ずつに分けるには？

ミニサッカーのチーム分け



- 友達を「点」、仲良しの対を「枝」で表す
- 点集合 V と枝集合 E からなるグラフ G=(V,E)
  - 枝の両端の点は互いに隣接する
  - 点に接続する枝の本数が次数
- VLSI設計など多くの応用をもつNP困難問題

(a) 線は仲良しを表す　*(b)* 等分割の最適解（またぐ枝は太線の2本）

グラフ分割問題の定式化（二次）

点 i が L 側に含まれるとき 1、R 側のとき 0 の 0-1 変数 x_i を用いる

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(i,j)\in E} x_i(1-x_j) + (1-x_i)x_j \\
\text{subject to} \quad & \sum_{i\in V} x_i = n/2 \\
& x_i \in \{0,1\} \qquad\qquad \forall i \in V
\end{aligned}
$$

- 等分割 (L,R)：$L\cap R=\emptyset$, $L\cup R=V$, $|L|=|R|=n/2$
- 枝 (i,j) が L と R をまたぐと $x_i(1-x_j)$ か $(1-x_i)x_j$ が 1
- 目的関数は凸でない二次関数で、多くのソルバーは扱えない

線形化した定式化

枝 (i,j) が L と R をまたぐとき 1 となる 0-1 変数 y_{ij} を導入する

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(i,j)\in E} y_{ij} \\
\text{subject to} \quad & \sum_{i\in V} x_i = n/2 \\
& x_i - x_j \le y_{ij} \quad \forall (i,j)\in E \\
& x_j - x_i \le y_{ij} \quad \forall (i,j)\in E \\
& x_i \in \{0,1\} \quad \forall i \in V \\
& y_{ij} \in \{0,1\} \quad \forall (i,j)\in E
\end{aligned}
$$

- 1番目の制約：等分割を規定
- 2番目：$i\in L$, $j\notin L$ のとき $y[i,j]=1$
- 3番目：$j\in L$, $i\notin L$ のとき $y[i,j]=1$

AMPLによるモデル

gpp.mod

```
set V;                    # 点の集合
set E within {V,V};       # 枝の集合

var x {V} binary;         # 点 i が部分集合 L に含まれるとき 1
var y {E} binary;         # 枝 (i,j) が両者を跨ぐとき 1

minimize CutEdges:
   sum {(i,j) in E} y[i,j];

subject to Balance:
   sum {i in V} x[i] = card(V) / 2;

subject to Cut1 {(i,j) in E}:
   x[i] - x[j] <= y[i,j];

subject to Cut2 {(i,j) in E}:
   x[j] - x[i] <= y[i,j];
```

- within {V,V}：枝は点の対の集合
- y は枝の集合 E の上で定義（存在しない枝の変数を作らない）
- card(V) は集合の要素数を返す

Pythonからの求解

gpp.py

```
def gpp(V, E):
    ampl = AMPL()
    ampl.read("gpp.mod")
    ampl.set["V"] = V
    ampl.set["E"] = E
    return ampl

def make_data(n, prob):
    V = list(range(1, n+1))
    E = [(i,j) for i in V for j in V
         if i < j and random.random() < prob]
    return V, E

V, E = make_data(30, .5)
ampl = gpp(V, E)
ampl.option["solver"] = "highs"
ampl.solve()
assert ampl.solve_result == "solved"
print("Opt.value=", ampl.get_objective("CutEdges").value())
```

- gpp：モデルを読み込み、点と枝のリストを流し込む
- make_data：ランダムグラフを生成
- solve_result で求解状態を確認

欄外ゼミナール：数理最適化と制約最適化

数理最適化

- 変数は実数や整数などの「数」
- 連続変数を扱える
- 非凸の二次関数は線形化が必要

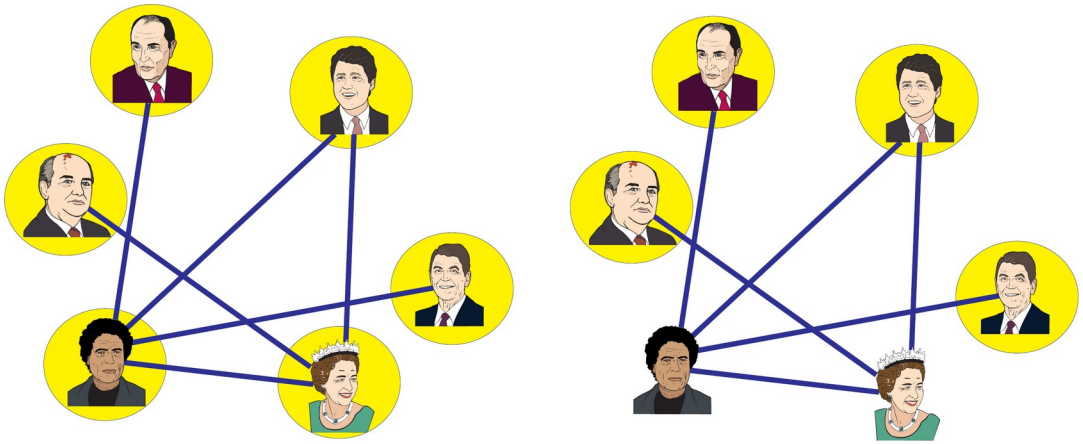
制約最適化

- 変数は領域から1つの要素を選ぶ
- 組合せ的な構造をもつ問題が得意
- 非凸の二次式も容易に定式化できる
- パズルのような問題も得意
  - 例：ソルバー SCOP

4.2 最大安定集合問題

仲の悪い人同士を同時に誘わずに、なるべくたくさんの仲間でピクニックに行くには？

ピクニックに誰を誘うか

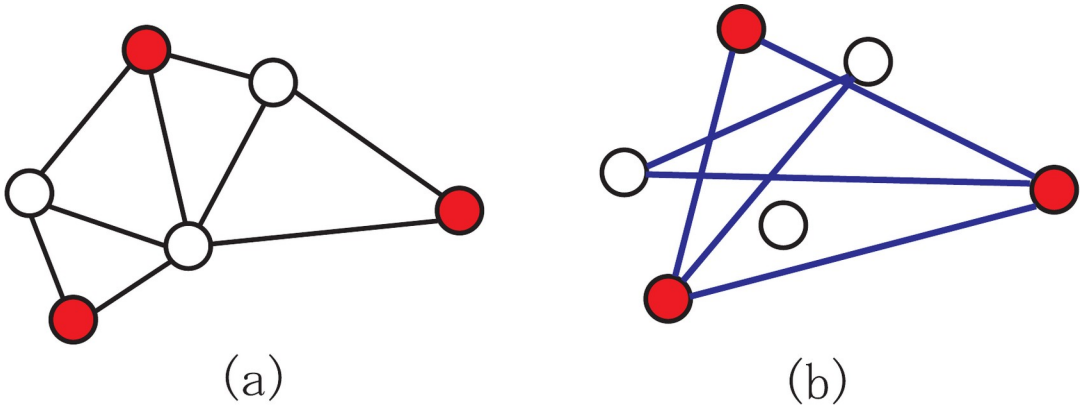


(a)　　(b)

(a) 線は仲が悪いことを表す　*(b)* 丸で囲んだ4人が最適解

- 点の部分集合 S 内のどの2点間にも枝がないとき、S を安定集合と呼ぶ
- 要素数（位数）|S| が最大の安定集合を求める
- 6人の例では4人を誘うのが最適

補グラフ上の最大クリーク



(a) 最大安定集合　*(b)* 補グラフ上の最大クリーク

- 補グラフ：枝の有無を反転させたグラフ
- クリーク：誘導部分グラフが完全グラフになる点集合 C
- 最大安定集合と最大クリークは同値
- 符号理論・信頼性・遺伝学・考古学・VLSI設計などに応用

0-1整数最適化による定式化

点 i が安定集合 S に含まれるとき 1 となる 0-1 変数 x_i を用いる

$$
\begin{aligned}
\text{maximize} \quad & \sum_{i \in V} x_i \\
\text{subject to} \quad & x_i + x_j \leq 1 \quad \forall (i,j) \in E \\
& x_i \in \{0,1\} \quad \forall i \in V
\end{aligned}
$$

- 目的：安定集合に含まれる点の数を最大化
- 制約：枝の両端点を同時には選べない

AMPLによるモデル

ssp.mod

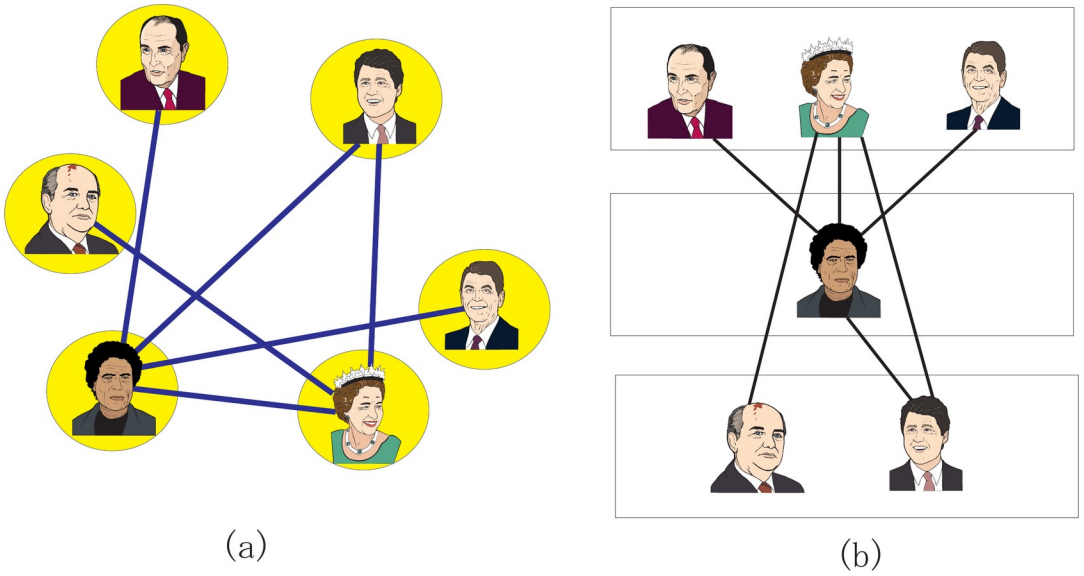
```
set V;                    # 点の集合
set E within {V,V};       # 枝の集合

var x {V} binary;         # 点 i を安定集合に入れるとき 1

maximize Size:
   sum {i in V} x[i];

subject to Edge {(i,j) in E}:
   x[i] + x[j] <= 1;
```

- 制約 Edge は枝の集合 E の上で定義
- 「すべての枝の両端点を同時には選べない」が1行で書ける
- Python側はグラフ分割問題と同様にデータを流し込む

4.3 グラフ彩色問題

仲が悪い友達を同じクラスに入れずに、なるべく少ないクラスに分けるには？

クラス分けとグラフ彩色



(a) 線は仲が悪いことを表す　*(b) 3*クラスに分けるのが最適

- K 分割：点集合 V を K 個の色クラス $V_1, \ldots, V_K$ に分ける
- すべての色クラスが安定集合なら K 彩色
- 最小の K（彩色数）を導く K 彩色を求める
- 時間割作成、周波数割当などに応用

標準定式化

x_{ik}：点 i を色 k で塗るとき 1、y_k：色 k を使うとき 1（$K_{\max}$ は色数の上界）

$$
\begin{aligned}
\text{minimize} \quad & \sum_{k=1}^{K_{\max}} y_k \\
\text{subject to} \quad & \sum_{k=1}^{K_{\max}} x_{ik} = 1 && \forall i \in V \\
& x_{ik} + x_{jk} \le y_k && \forall (i,j) \in E;\ k = 1, \dots, K_{\max} \\
& x_{ik} \in \{0,1\} && \forall i \in V;\ k = 1, \dots, K_{\max} \\
& y_k \in \{0,1\} && k = 1, \dots, K_{\max}
\end{aligned}
$$

- 1番目：各点に必ず1つの色を塗る
- 2番目：隣接点の同色禁止と、x と y の繋ぎ条件を同時に表す

解の対称性とその除去

- 分枝限定法では色クラスはすべて無記名に扱われる
  - $V_1 = \{1,2,3\}, V_2 = \{4,5\}$ と $V_1 = \{4,5\}, V_2 = \{1,2,3\}$ は同じ解
  - しかし異なるベクトル x, y で表される
- 変数で分枝しても下界が改良されない
- 対称性を除く制約 $y[k] \geq y[k+1]$ を付加
  - 添字の小さい色クラスを優先して使う

! モデリングのコツ

解が対称性をもつ場合には、対称性を除く制約を付加せよ。0-1変数を用いた簡単な制約の方がうまく働くことが多い。

対称性除去制約付きのモデル

gcp.mod

```
set V;                      # 点の集合
set E within {V,V};         # 枝の集合
param K > 0 integer;        # 使用可能な色数の上限
set COLORS = 1..K;          # 色の集合

var x {V,COLORS} binary;  # 点 i を色 k で塗るとき 1
var y {COLORS} binary;    # 色 k を使うとき 1

minimize NumColors:
   sum {k in COLORS} y[k];

subject to Assign {i in V}:
   sum {k in COLORS} x[i,k] = 1;

subject to Conflict {(i,j) in E, k in COLORS}:
   x[i,k] + x[j,k] <= y[k];

subject to Symmetry {k in COLORS: k < K}:
   y[k] >= y[k+1];
```

- 1..K：パラメータから集合をその場で定義
- K を変えれば COLORS も自動的に変わる
- {k in COLORS: k < K} で k=1,...,K-1 だけ生成

特殊順序集合（SOS）

1つ（連続する2つ）が正の値をとる変数群には特殊順序集合を使え

1 タイプ1

集合内の変数のうち、たかだか1つが0でない値をとる。$x \le M(1-y)$ も $\{x,y\}$ で表現できる

2 タイプ2

順序集合内の変数のうち、連続するたかだか2つが0でない値をとる。区分的線形近似で威力を発揮

3 グラフ彩色では

各点の色の変数にタイプ1を宣言できる。常に改善するとは限らないが試す価値はある

彩色数 K を固定した定式化

z_{ij}：枝 (i,j) の両端点が同じ色（悪い枝）のとき 1

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(i,j)\in E} z_{ij} & \\
\text{subject to} \quad & \textstyle\sum_{k=1}^{K} x_{ik} = 1 & \forall i \in V \\
& x_{ik} + x_{jk} \le 1 + z_{ij} & \forall (i,j) \in E;\ k = 1, \ldots, K \\
& x_{ik} \in \{0,1\} & \forall i \in V;\ k = 1, \ldots, K \\
& z_{ij} \in \{0,1\} & \forall (i,j) \in E
\end{aligned}
$$

- 悪い枝の数を最小化し、最適値が 0 なら K 彩色可能
- K を変えながら解けば最小の彩色数が求まる
- より大規模な問題の求解をねらう

二分探索による解法

solve_gcp.py

```
def solve_gcp(V, E):
    ampl = AMPL()
    ampl.read("gcp_fixed_k.mod")
    ampl.set["V"] = V
    ampl.set["E"] = E
    ampl.option["solver"] = "highs"

    LB = 0
    UB = len(V)
    while UB - LB > 1:
        K = (UB + LB) // 2
        ampl.param["K"] = K
        ampl.solve(verbose=False)
        if (ampl.solve_result == "solved" and
                ampl.get_objective("BadEdges").value() < 0.5):
            UB = K
        else:
            LB = K
    return UB
```

- UB：彩色可能な K、LB：彩色不能な K
- K を変えると COLORS と変数 x の個数もAMPLが自動で作り直す
- モデルの読み込みは反復の外で1度きり

探索の打ち切りと判定の注意

- 知りたいのは「悪い枝を0本にできるか」だけ
- 目的関数値がある値を超えた時点で打ち切るオプション
  - Gurobi の cutoff、HiGHS の lim:objectivebound など
  - {ソルバー名}_options で渡す
- 打ち切ると solve_result は "solved" にならない
- solved かつ悪い枝 0.5 未満のときだけ K 彩色可能と判定

! ポイント

打ち切った場合、変数には前回の求解の値が残っていることがある。目的関数値だけでなく solve_result も確かめる。

半順序定式化

点 i の色の番号を $c(i)$ とし、y_{ik}：$c(i) > k$ のとき 1、z_{ik}：$c(i) < k$ のとき 1

$$
\begin{aligned}
\text{minimize} \quad & 1 + \sum_{k=1}^{K_{\max}} y_{1k} \\
\text{subject to} \quad & x_{ik} = 1 - y_{ik} - z_{ik} && \forall i \in V;\ k = 1, \ldots, K_{\max} \\
& z_{i1} = 0, \quad y_{iK_{\max}} = 0 && \forall i \in V \\
& y_{ik} \geq y_{i,k+1} && \forall i \in V;\ k = 1, \ldots, K_{\max} - 1 \\
& y_{ik} + z_{i,k+1} = 1 && \forall i \in V;\ k = 1, \ldots, K_{\max} - 1 \\
& y_{1k} \geq y_{ik} && \forall i \in V \setminus \{1\};\ k = 1, \ldots, K_{\max} \\
& x_{ik} + x_{jk} \leq 1 && \forall (i,j) \in E;\ k = 1, \ldots, K_{\max} \\
& x_{ik}, y_{ik}, z_{ik} \in \{0, 1\} && \forall i \in V;\ k = 1, \ldots, K_{\max}
\end{aligned}
$$

半順序定式化のモデル（抜粋）

gcp_partial_order.mod

```
param n;
set V := 1..n;
set E within {V,V};
param K > 0 integer;          # 使用可能な色数の上限
set COLORS = 1..K;
var x {V,COLORS} binary;      # 点 i の色が k のとき 1
var y {V,COLORS} binary;      # 点 i の色が k より大きいとき 1
var z {V,COLORS} binary;      # 点 i の色が k より小さいとき 1

minimize NumColors: 1 + sum {k in COLORS} y[1,k];
subject to Define {i in V, k in COLORS}:
   x[i,k] = 1 - y[i,k] - z[i,k];
subject to First {i in V}: z[i,1] = 0;
subject to Last {i in V}: y[i,K] = 0;
subject to Order {i in V, k in COLORS: k < K}:
   y[i,k] >= y[i,k+1];
subject to Link {i in V, k in COLORS: k < K}:
   y[i,k] + z[i,k+1] = 1;
subject to Max {i in V, k in COLORS: i <> 1}:
   y[1,k] >= y[i,k];
subject to Conflict {(i,j) in E, k in COLORS}:
   x[i,k] + x[j,k] <= 1;
```

代表点定式化

色クラス内で番号最小の点を代表点とし、x_{ij}：点 j が代表点 i の色クラスに入るとき 1

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \in V} x_{ii} \\
\text{subject to} \quad & x_{jj} + \sum_{i \in N_j} x_{ij} \geq 1 && \forall j \in V \\
& x_{ij} + x_{ik} \leq x_{ii} && \forall i \in V;\ (j,k) \in A_i \\
& x_{ij} \leq x_{ii} && \forall j \in V;\ i \in N_j \\
& x_{ij} \in \{0,1\} && \forall j \in V;\ i \in N_j \cup \{j\}
\end{aligned}
$$

- N[j]：j より番号が小さく j に隣接しない点（代表点の候補）
- A[i]：i に隣接しない2点を結ぶ枝で、両端点の番号が i より大きいもの
- 色の番号を用いないので付け替えによる対称性が生じない

代表点定式化のモデル

gcp_representative.mod

```
param n;
set V := 1..n;
set E within {V,V};           # 枝 (i,j) は i < j とする

set N {j in V} = {i in V: i < j and (i,j) not in E};
set A {i in V} = {(j,k) in E: i < j and
                  (i,j) not in E and (i,k) not in E};

var x {i in V, j in V: i = j or i in N[j]} binary;

minimize NumColors:
   sum {i in V} x[i,i];

subject to Assign {j in V}:
   x[j,j] + sum {i in N[j]} x[i,j] >= 1;

subject to Conflict {i in V, (j,k) in A[i]}:
   x[i,j] + x[i,k] <= x[i,i];

subject to Represent {j in V, i in N[j]}:
   x[i,j] <= x[i,i];
```

- 集合 N と A はAMPLが枝の集合 E から計算
- not in：組が集合に含まれないことを表す
- i = j or i in N[j] で意味のある組だけ変数を生成

グラフ彩色問題の4つの定式化

定式化	主な変数	目的関数	対称性への対処
標準定式化	$x[i,k]$, $y[k]$	使う色の数	$y[k] \geq y[k+1]$ を付加
K 固定＋二分探索	$x[i,k]$, $z[i,j]$	悪い枝の数	K を二分探索で調整
半順序定式化	x, y, z（色の大小）	点1の色の番号	点1の色を最大にして一部除去
代表点定式化	$x[i,j]$（代表点）	代表点の数	色の番号を使わず対称性なし

半順序定式化と代表点定式化は解の対称性を避けており、比較的高速に解を得られる（問題例やソルバーによる）

まとめ

- ✓ グラフ分割は二次の目的関数を $y[i,j]$ で線形化して解く
- ✓ 最大安定集合は枝ごとに $x[i]+x[j] \le 1$ を課す
- ✓ AMPLでは枝の集合 E をそのまま添字集合に使える
- ✓ グラフ彩色では解の対称性を除く制約が有効
- ✓ 彩色数を固定した定式化と二分探索で大規模化に対応
- ✓ 半順序・代表点定式化は対称性を避けて高速に解く