05

第5章

巡回路問題

巡回セールスマン問題とその拡張の定式化をAMPLで比べる

この章の内容

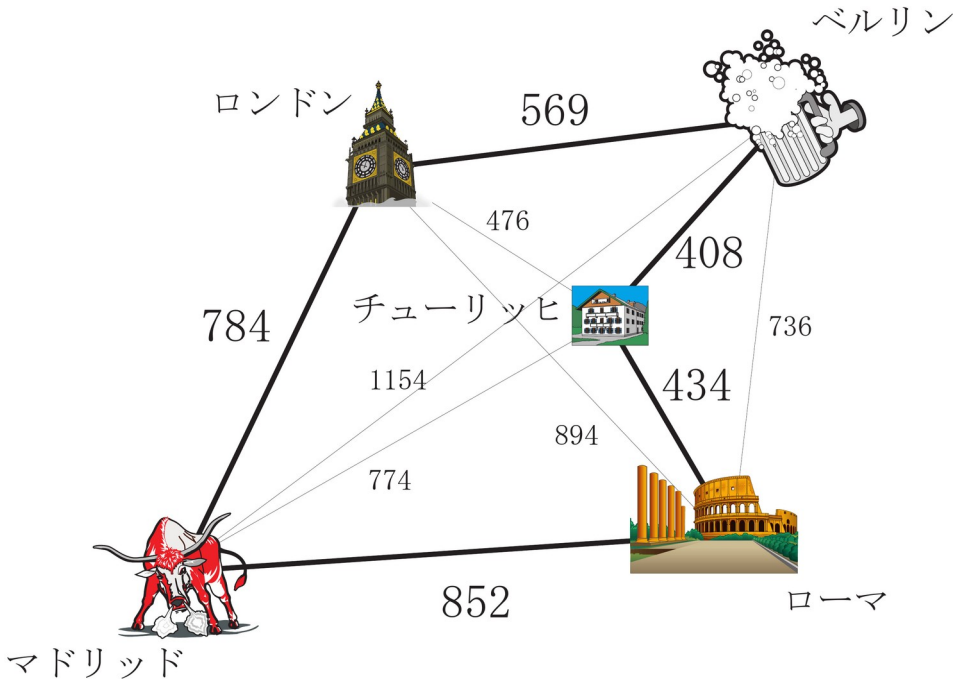
1. 巡回セールスマン問題
2. 時間枠付き巡回セールスマン問題
3. 容量制約付き配送計画問題

5.1

巡回セールスマン問題

チューリッヒを出て4都市を回り、また戻ってくる。最短の順序は？

ヨーロッパ旅行の問題



枝上の数値は距離（マイル）、太線が最適解

- チューリッヒを出発し、マドリッド・ロンドン・ローマ・ベルリンを回って戻る
- レンタルヘリコプターの料金は移動距離に比例
- 移動距離が最小になる訪問順序を求めたい

巡回セールスマン問題の定義

- n 個の点（都市）からなる無向グラフ G=(V,E)
- 枝 e 上の距離（重み、費用、移動時間）c[e]
- すべての点をちょうど1回ずつ経由する巡回路のうち、長さ最小のものを求める
- 無向グラフ上：対称巡回セールスマン問題
- 有向グラフ上（行きと帰りで距離が異なる）：非対称巡回セールスマン問題

ポイント

対称な問題は非対称な問題の特殊形なので、非対称な問題の定式化はそのまま対称な問題にも適用できる。

本節で扱う4つの定式化

同じ問題でも定式化によって強さと解きやすさが異なる

1 部分巡回路除去

Dantzig–Fulkerson–Johnson。制約が膨大なので切除平面法で必要な分だけ追加する

2 ポテンシャル（MTZ）

Miller–Tucker–Zemlin。訪問順序の変数 u を使う。持ち上げで強化できる

3 単一品種フロー

点1から $n-1$ 単位の「もの」を運び、各点で1単位ずつ消費する

4 多品種フロー

点 k 行きの品種 k を区別して、1単位ずつ運ぶ

部分巡回路除去定式化

枝 e が巡回路に含まれるとき 1 となる 0-1 変数 x_e を用いる（対称TSP）

$$
\begin{aligned}
\text{minimize} \quad & \sum_{e \in E} c_e x_e \\
\text{subject to} \quad & \sum_{e \in \delta(\{i\})} x_e = 2 && \forall i \in V \\
& \sum_{e \in E(S)} x_e \leq |S| - 1 && \forall S \subset V, |S| \geq 2 \\
& x_e \in \{0, 1\} && \forall e \in E
\end{aligned}
$$

- E(S)：両端点が S に含まれる枝の集合
- δ(S)：一端だけが S に含まれる枝の集合
- 1本目：次数制約（各点に接続する枝は2本）
- 2本目：部分巡回路除去制約

カットセット制約と分離問題

部分巡回路除去制約を2倍し、S 内の点の次数制約を引くと得られる

$$\sum_{e \in \delta(S)} x_e \geq 2 \quad \forall S \subset V, |S| \geq 2$$

- カットセット制約：TSPでは部分巡回路除去制約と同じ強さ
- 制約が膨大なので、必要に応じて追加する切除平面法を使う
- 分離問題：緩和解を破っている制約をみつける問題
- 厳密には最小カット、本書では連結成分を求める簡便法

AMPLによるモデル（部分巡回路除去）

tsp.mod

```
set V;                                # 点の集合
set E = {i in V, j in V: i < j};  # 枝の集合
param c {E} >= 0;                     # 枝 (i,j) の距離

param nCut >= 0 integer, default 0; # 追加済みの本数
set CUTS = 1..nCut;
set S {CUTS} within V;    # k 番目の制約の点集合

var x {E} binary;         # 枝 (i,j) を使うとき 1
minimize Length:
   sum {(i,j) in E} c[i,j] * x[i,j];

subject to Degree {i in V}:
   sum {j in V: j < i} x[j,i]
     + sum {j in V: j > i} x[i,j] = 2;

subject to Subtour {k in CUTS}:
   sum {i in S[k], j in S[k]: j > i} x[i,j]
     <= card(S[k]) - 1;
```

- i < j の対だけで枝集合 E を定義（無向グラフの定石）
- S {CUTS} は添え字付き集合：k ごとに V の部分集合が1つ
- nCut を増やすだけで制約 Subtour が増える。モデルを作り直す必要はない

連結成分による部分巡回路除去制約の追加

addcut（Python）

```
def addcut(ampl, V, edges, nCut):
    G = networkx.Graph()
    G.add_nodes_from(V)
    for (i, j) in edges:
        G.add_edge(i, j)
    Components = list(networkx.connected_components(G))
    if len(Components) == 1:
        return nCut
    for S in Components:
        nCut += 1
        ampl.param["nCut"] = nCut
        ampl.get_set("S")[nCut].set_values(sorted(S))
    return nCut
```

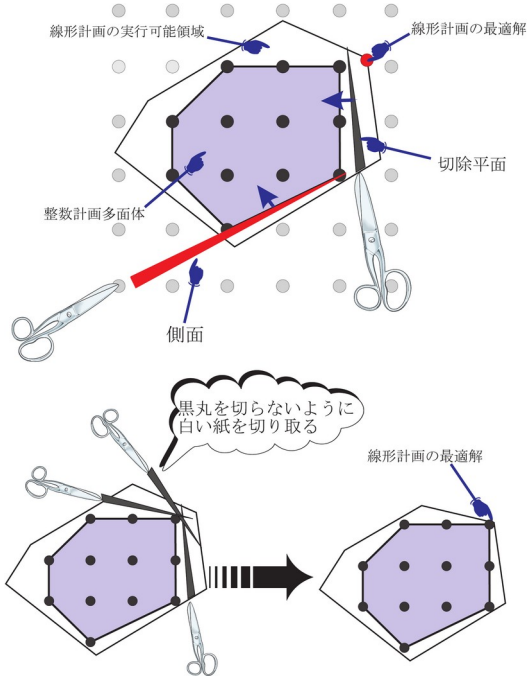
反復による切除平面法

- relax_integrality=1 として、部分巡回路除去制約のない線形緩和問題から出発
- 解いて使われている枝を取り出し、連結成分ごとに制約を追加
- 連結成分が1つになったら relax_integrality=0 として整数条件を付加
- 再び制約を追加し、整数解の連結成分が1つになったら最適解
- 分枝限定法の途中で制約を追加する方法が分枝切除法

! ポイント

分枝切除法にはコールバックと遅延制約が必要で、amplpy とは別に ampls を使う。本書は反復による方法を標準として推奨する。

欄外ゼミナール：切除平面法



切除平面法の概念図

- 妥当不等式：実行可能解（整数解）を除かない不等式
- 切除平面：妥当不等式のうち線形緩和問題の解を除くもの
- 側面（facet）：最も強い妥当不等式
- 切除平面を追加しながら線形最適化問題を繰り返し解く

Miller–Tucker–Zemlin（ポテンシャル）定式化

非対称TSP。x_{ij}：点 i の次に点 j を訪問するとき 1、u_i：点 i の訪問順序（$u_1 = 0$）

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \neq j} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j : j \neq i} x_{ij} = 1 && \forall i \\
& \sum_{j : j \neq i} x_{ji} = 1 && \forall i \\
& u_i + 1 - (n-1)(1 - x_{ij}) \leq u_j && \forall i \neq j,\ j \geq 2 \\
& 1 \leq u_i \leq n - 1 && \forall i = 2, \ldots, n \\
& x_{ij} \in \{0, 1\} && \forall i \neq j
\end{aligned}
$$

- 出次数制約・入次数制約：出る枝・入る枝はちょうど1本
- ポテンシャル制約：$x_{ij} = 1$ なら $u_j \geq u_i + 1$
- 係数 n−1 が Big M の働きをするため、部分巡回路除去定式化よりはるかに弱い

持ち上げによる強化

式に変数の項を加え、実行可能解を除かない範囲で係数をなるべく大きくする

$$u_i + 1 - (n-1)(1 - x_{ij}) + (n-3)\,x_{ji} \le u_j$$

$$1 + (1 - x_{1i}) + (n-3)\,x_{i1} \le u_i$$

$$u_i \le (n-1) - (1 - x_{i1}) - (n-3)\,x_{1i}$$

- ポテンシャル制約に $\alpha\, x_{ji}$ を追加：$x_{ji} = 1$ なら $u_j + 1 = u_i$ より $\alpha \le n-3$
- 下限制約：$x_{1i} = 0$ なら点 i は2番目以降なので $u_i \ge 2$、$x_{i1} = 1$ なら $u_i = n-1$
- 上限制約も同様に持ち上げる

AMPLによるモデル（持ち上げ版MTZ）

mtz.mod

```
param n > 0 integer;                # 点の数
set V = 1..n;
set A = {i in V, j in V: i <> j};   # 有向枝の集合
param c {A} >= 0;

var x {A} binary;
var u {i in V} >= (if i = 1 then 0 else 1),
               <= (if i = 1 then 0 else n-1);

minimize Length: sum {(i,j) in A} c[i,j] * x[i,j];

subject to OutDeg {i in V}: sum {j in V: j <> i} x[i,j] = 1;
subject to InDeg {i in V}:  sum {j in V: j <> i} x[j,i] = 1;

subject to MTZ {i in V, j in 2..n: i <> j}:
   u[i] - u[j] + (n-1)*x[i,j] + (n-3)*x[j,i] <= n-2;
subject to LiftLB {i in 2..n}:
   -x[1,i] - u[i] + (n-3)*x[i,1] <= -2;
subject to LiftUB {i in 2..n}:
   -x[i,1] + u[i] + (n-3)*x[1,i] <= n-2;
```

ポテンシャル制約の2つの書き方

持ち上げで強化した式

- $u[i] - u[j] + (n-1)x[i,j] + (n-3)x[j,i] \leq n-2$
- Gurobi の実験では速い
- 速度重視ならこちら

指示演算子を使う式

- $x[i,j] = 1 \Longrightarrow u[j] >= u[i] + 1$
- 枝 (i,j) を使えば u[j] は u[i] より1以上大きい
- 等式にすると僅かに遅くなる
- 分かりやすさ重視ならこちら

単一品種フロー定式化

点1に置いた n−1 単位の「もの」をセールスマンが運び、各点で1単位ずつ消費する

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \neq j} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j : j \neq i} x_{ij} = 1, \ \sum_{j : j \neq i} x_{ji} = 1 \quad \forall i \\
& \sum_{j} f_{1j} = n - 1 \\
& \sum_{j} f_{ji} - \sum_{j} f_{ij} = 1 \qquad \forall i = 2, \ldots, n \\
& f_{1j} \leq (n-1) x_{1j} \qquad \forall j \neq 1 \\
& f_{ij} \leq (n-2) x_{ij} \qquad \forall i \neq j;\ i, j \neq 1 \\
& x_{ij} \in \{0,1\},\ f_{ij} \geq 0 \qquad \forall i \neq j
\end{aligned}
$$

- f_{ij}：枝 (i,j) を通過する「もの」の量
- 容量制約：セールスマンが移動しない枝には流れない
- 点1以外から出る枝は n−2 以下（多少強化した式）

AMPLによるモデル（単一品種フロー）

scf.mod

```
param n > 0 integer;
set V = 1..n;
set A = {i in V, j in V: i <> j};
param c {A} >= 0;

var x {A} binary;
var f {(i,j) in A} >= 0, <= (if i = 1 then n-1 else n-2);

minimize Length: sum {(i,j) in A} c[i,j] * x[i,j];
# OutDeg, InDeg は mtz.mod と同じ

subject to Source:
   sum {j in 2..n} f[1,j] = n-1;

subject to FlowBalance {i in 2..n}:
   sum {j in V: j <> i} f[j,i]
     - sum {j in V: j <> i} f[i,j] = 1;

subject to Cap1 {j in 2..n}: f[1,j] <= (n-1) * x[1,j];
subject to Cap2 {i in 2..n, j in 2..n: i <> j}:
   f[i,j] <= (n-2) * x[i,j];
```

多品種フロー定式化

点1から点 k 行きの品種 k を1単位ずつ出荷し、点 k で消費する

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \neq j} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j:j\neq i} x_{ij} = 1, \ \sum_{j:j\neq i} x_{ji} = 1 && \forall i \\
& \sum_{j:j\neq i} f_{ji}^{k} - \sum_{j:j\neq i} f_{ij}^{k} = \begin{cases} -1 & i = 1 \\ 0 & i \neq 1, k \\ 1 & i = k \end{cases} && \forall k \\
& f_{ij}^{k} \leq x_{ij} && \forall k, i, j \\
& x_{ij} \in \{0,1\}, \ f_{ij}^{k} \geq 0
\end{aligned}
$$

- f_{ij}^{k}：枝 (i,j) を通過する品種 k の量
- 点ごとに運ばれる「もの」が区別される点が単一品種フローとの違い
- 容量制約 f ≤ x：移動しない枝には流れない

AMPLによるモデル（多品種フロー）

mcf.mod

```
param n > 0 integer;
set V = 1..n;
set A = {i in V, j in V: i <> j};
set K = 2..n;                                   # 品種
set F = {(i,j) in A, k in K: j <> 1 and i <> k};
param c {A} >= 0;

var x {A} binary;
var f {F} >= 0, <= 1;

minimize Length: sum {(i,j) in A} c[i,j] * x[i,j];
# OutDeg, InDeg は mtz.mod と同じ

subject to Ship {k in K}:
   sum {i in 2..n: (1,i,k) in F} f[1,i,k] = 1;
subject to Receive {k in K}:
   sum {i in V: (i,k,k) in F} f[i,k,k] = 1;
subject to Conserve {k in K, i in 2..n: i <> k}:
   sum {j in V: (j,i,k) in F} f[j,i,k]
     = sum {j in V: (i,j,k) in F} f[i,j,k];
subject to Cap {(i,j,k) in F}: f[i,j,k] <= x[i,j];
```

5.2 時間枠付き巡回セールスマン問題

各点を出発する時刻に、最早時刻と最遅時刻の枠があったら？

時間枠付き巡回セールスマン問題

- 点1を時刻0に出発する非対称TSP
- 移動距離 $c[i,j]$ を移動時間とみなす（$c[i,j] > 0$ と仮定）
- 点 i の出発時刻は最早時刻 $e[i]$ と最遅時刻 $l[i]$ の間
- $e[i]$ より早く着いたら、$e[i]$ まで待てる
- 変数 $t[i]$：点 i を出発する時刻（$e[1]=0$, $l[1]=\infty$）

! ポイント

$c[i,j] > 0$ なら、時刻の制約だけで部分巡回路も除去できる。$c[i,j]=0$ だと $t[i]=t[j]$ となり部分巡回路ができうる。

ポテンシャル定式化

$x_{ij} = 0$ のとき $M \geq t_i + c_{ij} - t_j$ が成り立てばよいので、$M = [\ell_i + c_{ij} - e_j]^+$ とできる

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \neq j} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j:j \neq i} x_{ij} = 1, \ \sum_{j:j \neq i} x_{ji} = 1 && \forall i \\
& t_i + c_{ij} - [\ell_i + c_{ij} - e_j]^+ (1 - x_{ij}) \leq t_j && \forall i, j : j \neq 1, i \neq j \\
& x_{ij} \in \{0, 1\} && \forall i \neq j \\
& e_i \leq t_i \leq \ell_i && \forall i
\end{aligned}
$$

- 素朴な $t_i + c_{ij} - M(1-x_{ij}) \leq t_j$ の Big M を時間枠で小さくする
- $[\cdot]^+$ は $\max\{\cdot, 0\}$。M はなるべく小さい方が強い

持ち上げによるさらなる強化

ポテンシャル制約と上下限制約を、TSPと同様に持ち上げる

$$
\begin{aligned}
&t_i + c_{ij} - [\ell_i + c_{ij} - e_j]^+(1 - x_{ij}) \\
&\quad + [\ell_i - e_j + \min\{-c_{ji},\, e_j - e_i\}]^+ x_{ji} \le t_j && \forall i, j : j \neq 1, i \neq j \\
&e_i + \sum_{j \neq i} [e_j + c_{ji} - e_i]^+ x_{ji} \le t_i && \forall i = 2, \ldots, n \\
&t_i \le \ell_i - \sum_{j \neq 1, i} [\ell_i - \ell_j + c_{ij}]^+ x_{ij} && \forall i = 2, \ldots, n
\end{aligned}
$$

- 1本目：逆向きの枝 x_{ji} の項を加えたポテンシャル制約
- 2・3本目：前後の点との関係で時刻の下限・上限を強化

AMPLによるモデル（時間枠付き）

mtz2tw.mod

```
param c {A} >= 0;            # 枝 (i,j) の移動時間
param e {V};                 # 時間枠の下限
param l {i in V} >= e[i];    # 時間枠の上限

var x {A} binary;
var t {i in V} >= e[i], <= l[i];   # 訪問時刻

minimize Length: sum {(i,j) in A} c[i,j] * x[i,j];
# OutDeg, InDeg は mtz.mod と同じ

subject to MTZ {i in V, j in 2..n: i <> j}:
   t[i] + c[i,j]
   - max(l[i] + c[i,j] - e[j], 0) * (1 - x[i,j])
   + max(l[i] + min(-c[j,i], e[j] - e[i]) - e[j], 0)
     * x[j,i] <= t[j];
subject to TightLB {i in 2..n}:
   e[i] + sum {j in V: j <> i}
     max(e[j] + c[j,i] - e[i], 0) * x[j,i] <= t[i];
subject to TightUB {i in 2..n}:
   t[i] <= l[i] - sum {j in 2..n: j <> i}
     max(l[i] - l[j] + c[i,j], 0) * x[i,j];
```

- 時間枠は変数 t の上下限として宣言に書く
- max, min の引数はパラメータだけなので、展開時に数値に確定する
- 持ち上げの係数を Python で計算せずモデルにそのまま書ける

2添え字ポテンシャル定式化

Big M を除くため、y_{ij} = 点 i の発時刻（$x_{ij} = 1$ のとき）, 0（それ以外）を導入する

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i \neq j} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j:j \neq i} x_{ij} = 1, \ \sum_{j:j \neq i} x_{ji} = 1 && \forall i \\
& \sum_{i:i \neq j} y_{ij} + \sum_{i:i \neq j} c_{ij} x_{ij} \leq \sum_{k:k \neq j} y_{jk} && \forall j = 2, \ldots, n \\
& e_i x_{ij} \leq y_{ij} \leq \ell_i x_{ij} && \forall i \neq j \\
& x_{ij} \in \{0, 1\} && \forall i \neq j
\end{aligned}
$$

- 3本目：点 i の発時刻＋移動時間 $c_{ij} \leq$ 点 j の発時刻
- 4本目：$x_{ij} = 1$ のときだけ y_{ij} が正になれることと時間枠を同時に表す

AMPLによるモデル（2添え字）

tsptw2.mod

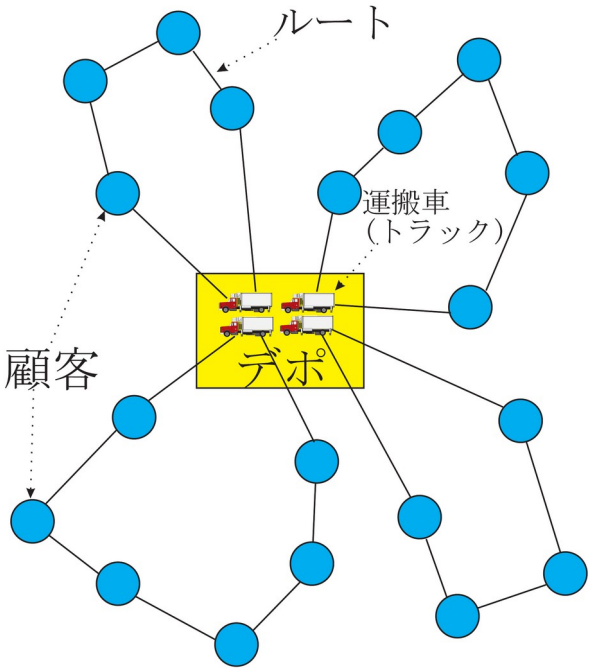
```
param n > 0 integer;
set V = 1..n;
set A = {i in V, j in V: i <> j};
param c {A} >= 0;
param e {V};
param l {i in V} >= e[i];

var x {A} binary;
var y {A} >= 0;     # 枝 (i,j) に付随する時刻変数

minimize Length: sum {(i,j) in A} c[i,j] * x[i,j];
# OutDeg, InDeg は mtz.mod と同じ

subject to Time {j in 2..n}:
   sum {i in V: i <> j} (y[i,j] + c[i,j] * x[i,j])
   - sum {k in V: k <> j} y[j,k] <= 0;

subject to WindowLB {(i,j) in A}: e[i] * x[i,j] <= y[i,j];
subject to WindowUB {(i,j) in A}: y[i,j] <= l[i] * x[i,j];
```

- Big M を含まない
- 2つの定式化は同じ問題を表すので最適値は一致
- n=8 のランダムな例（60例）で、どちらも全数列挙と同じ最適値

5.3 容量制約付き配送計画問題

容量に限りのある複数台の運搬車で、すべての顧客をどう回るか？

配送計画問題の仮定



配送計画問題の概念図

- デポを出た運搬車が顧客を回ってデポに戻る（ルート）
- 運搬車の容量、顧客の位置と需要量、移動費用は既知
- 各顧客はちょうど1回訪問される
- ルート上の需要量の合計は容量以下（容量制約）
- 運搬車は1種類で台数 m は決まっている

容量制約付き配送計画問題の定式化

x_{ij}：点 i, j 間（$i < j$）を運搬車が移動する回数。ピストン輸送では $x_{1j} = 2$

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i,j} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j} x_{1j} = 2m \\
& \sum_{j} x_{ij} = 2 && \forall i = 2, \dots, n \\
& \sum_{i,j \in S} x_{ij} \leq |S| - N(S) && \forall S \subset \{2, \dots, n\}, |S| \geq 2 \\
& x_{1j} \in \{0, 1, 2\} && \forall j = 2, \dots, n \\
& x_{ij} \in \{0, 1\} && \forall i < j : i \neq 1
\end{aligned}
$$

- 1本目：デポに出入りする枝は 2m 本
- 2本目：各顧客に1台が訪問
- 3本目：容量制約と部分巡回路除去を同時に表す
- N(S)：S の需要を運ぶのに必要な台数。下界 ⌈Σ q[i] / Q⌉ で代用

AMPLによるモデル（配送計画）

vrp.mod

```
param n > 0 integer;
set V = 1..n;                       # 1 がデポ
set E = {i in V, j in V: i < j};
param c {E} >= 0;
param q {V} >= 0;                   # 需要量
param Q > 0;                        # 運搬車の容量
param m > 0 integer;                # 運搬車の台数

param nCut >= 0 integer, default 0;
set CUTS = 1..nCut;
set S {CUTS} within V;
param NS {CUTS} >= 0 default 0;   # 必要台数の下界

var x {(i,j) in E} integer >= 0,
                   <= (if i = 1 then 2 else 1);

minimize Length: sum {(i,j) in E} c[i,j] * x[i,j];
subject to Depot: sum {j in V: j > 1} x[1,j] = 2*m;
subject to Degree {i in V: i > 1}:
   sum {j in V: j < i} x[j,i] + sum {j in V: j > i} x[i,j] = 2;
subject to Capacity {k in CUTS}:
   sum {i in S[k], j in S[k]: j > i} x[i,j] <= card(S[k]) - NS[k];
```

反復による切除平面法（配送計画）

solve_vrp（抜粋）

```
while True:
    ampl.solve(verbose=False)
    xs = ampl.get_variable("x").get_values().to_pandas()
    edges = [(i, j) for (i, j), r in xs.iterrows()
             if r["x.val"] > .5 and i != 1 and j != 1]
    G = networkx.Graph()
    G.add_edges_from(edges)
    added = 0
    for S in networkx.connected_components(G):
        S_card = len(S)
        NS = int(math.ceil(sum(q[i] for i in S) / Q))
        S_edges = [(i, j) for i in S for j in S
                   if i < j and (i, j) in edges]
        if S_card >= 2 and (len(S_edges) >= S_card or NS > 1):
            nCut += 1
            ampl.param["nCut"] = nCut
            ampl.get_set("S")[nCut].set_values(sorted(S))
            ampl.param["NS"] = {nCut: NS}
            added += 1
    if added == 0:
        break
```

まとめ

- ✓ TSPには部分巡回路除去、MTZ、単一品種・多品種フローの定式化がある
- ✓ 膨大な制約は nCut で添え字付けし、切除平面法で必要な分だけ追加する
- ✓ Big M を含む弱い式は持ち上げで強化できる
- ✓ 時間枠付きTSPは時間枠で M を小さくし、2添え字定式化で M を除く
- ✓ 配送計画は容量と部分巡回路を $|S|-N(S)$ の切除平面で同時に扱う