問題解決のための最適化

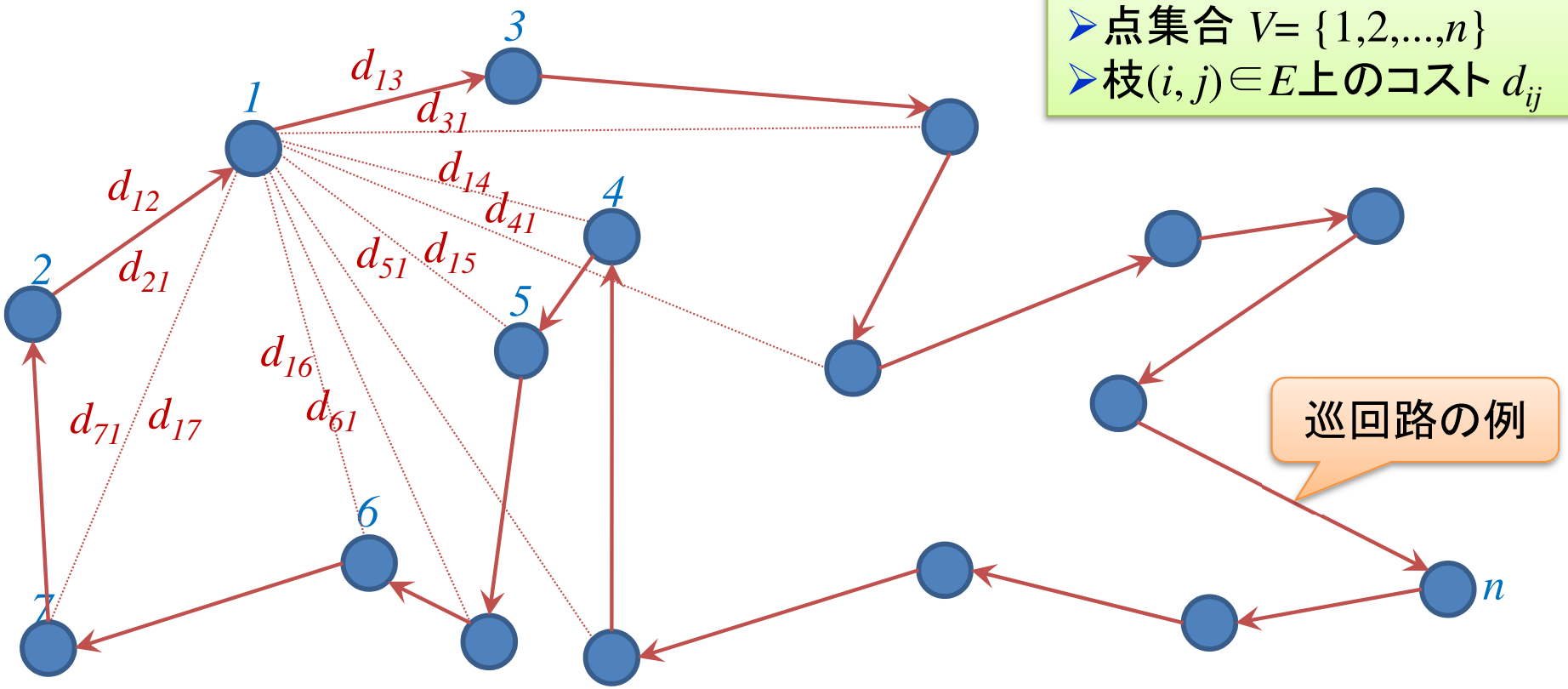
組合せ最適化と整数計画法
5. 巡回セールスマン問題

堀田 敬介

巡回セールスマン問題の最適化

- 巡回セールスマン問題 traveling salesman problem; TSP
  - n個の点があり，任意の2点間 i, j で相互に行き来する枝$(i, j), (j, i)$ を考える
  - 任意の2点i, j間にはコストd_{ij} がある（所与）
  - 全点を丁度1度ずつ回る巡回路を生成する（自己ループ (i, i) は除く）
  - コストの総和が最小の巡回路を求めたい

- 完全有向グラフ $G = (V, E)$
- 点集合 $V = \{1,2,...,n\}$
- 枝$(i, j) \in E$上のコスト d_{ij}



- $\forall i, j, d_{ij} = d_{ji}$ のとき対称巡回セールスマン問題
- $\exists i, j, d_{ij} \neq d_{ji}$ のとき非対称巡回セールスマン問題

※対称の場合，無向グラフで考えても良い

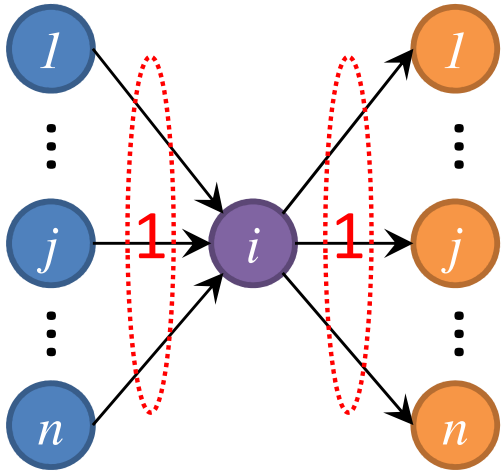
巡回セールスマン問題の最適化

- 最適化問題の定式化（変数設定・係数表記）
  - 0-1変数 $x_{ij} = \begin{cases} 1 & \cdots \text{枝}(i,j)\text{を通る} \\ 0 & \cdots \text{枝}(i,j)\text{を通らない} \end{cases}$

- 定式化の準備：巡回路となるための必要条件

点i へ**入る**枝は丁度1本

$$\sum_{j:j\neq i} x_{ji} = 1 \quad (\forall i \in \{1,\ldots,n\})$$



$$\sum_{j:j\neq i} x_{ij} = 1 \quad (\forall i \in \{1,\ldots,n\})$$

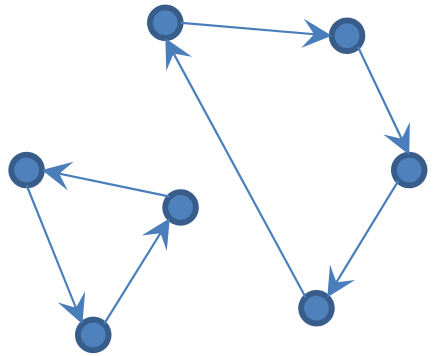
点i を**出る**枝は丁度1本

各点でみれば，巡回路とは，その点に接続する全枝の
- どれか丁度1本の枝を使って入ってきて，
- どれか丁度1本の枝を使って出て行く

ことになる．それを書いた，[入る制約]と[出る制約]の式

巡回セールスマン問題の最適化

部分巡回路の例



- TSPの定式化

min.

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

s. t.

$$\sum_{j:j\neq i} x_{ji} = 1(\forall i \in \{1,\dots,n\})$$

点iへ**入る**枝は丁度1本

$$\sum_{j:j\neq i} x_{ij} = 1(\forall i \in \{1,\dots,n\})$$

点iを**出る**枝は丁度1本

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

$$x_{ii} = 0\ (\forall i \in I)$$

しかし，この2種類の制約だけでは部分巡回路が含まれる

巡回路が1本につながらず，
部分的な巡回路が複数できてしまう

部分巡回路を回避する定式化

- ✓ 部分巡回路除去定式化
- ✓ ポテンシャル定式化
- ✓ 単一品種流定式化
- ✓ 多品種流定式化

巡回セールスマン問題をCPLEXで解く

- Traveling Salesman Problemの最適化（例1）
- 最適化問題の定式化1（変数設定・係数表記）
  - 距離行列 distance matrix $\boldsymbol{D}=[d_{ij}]$

コスト（距離）d_{ij}(km)

1	0.0	2.0	1.0	1.9	1.5	1.4	1.8	1.7
2	2.0	0.0	2.3	1.3	1.2	1.7	1.8	1.9
3	1.0	2.3	0.0	1.3	1.2	1.7	1.8	1.9
4	1.9	1.3	1.3	0.0	1.9	2.7	0.8	1.8
5	1.5	1.2	1.2	1.9	0.0	1.4	2.4	2.3
6	1.4	1.7	1.7	2.7	1.4	0.0	2.4	2.0
7	1.8	1.8	1.8	0.8	2.4	2.4	0.0	1.2
8	1.7	1.9	1.9	1.8	2.3	2.0	1.2	0.0

巡回セールスマン問題をCPLEXで解く

- 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：TSP）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 [TSP.mod] ※名称変更しない
  - ✓ データファイル [TSP.dat] → [TSPex1.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

巡回セールスマン問題をCPLEXで解く

- TSP.mod

```
int i_max = ...;// 頂点数|V|

range I = 1..i_max;

float d[I,I] = ...;// 距離行列D=[dij]

dvar int+ x[I,I] in 0..1;// 0-1変数

minimize
  sum(i in I) sum(j in I) d[i][j]*x[i][j];
subject to {
  forall (i in I) {
    sum(j in I) x[i][j] == 1;// 点iから出る枝1本
    sum(j in I) x[j][i] == 1;// 点iに入る枝1本
  }
  forall (i in I) {
    x[i][i] == 0;// 自己ループ(i→iの枝)はなし
  }

 // 部分巡回路除去制約(最初はなし)

};
```

巡回セールスマン問題をCPLEXで解く

- TSPex1.dat

```
i_max = 8;// 頂点数|V|

d = [
[ 0.0 2.0 1.0 1.9 1.5 1.4 1.8 1.7 ]
[ 2.0 0.0 2.3 1.3 1.2 1.7 1.8 1.9 ]
[ 1.0 2.3 0.0 1.3 1.2 1.7 1.8 1.9 ]
[ 1.9 1.3 1.3 0.0 1.9 2.7 0.8 1.8 ]
[ 1.5 1.2 1.2 1.9 0.0 1.4 2.4 2.3 ]
[ 1.4 1.7 1.7 2.7 1.4 0.0 2.4 2.0 ]
[ 1.8 1.8 1.8 0.8 2.4 2.4 0.0 1.2 ]
[ 1.7 1.9 1.9 1.8 2.3 2.0 1.2 0.0 ]
];
```

巡回セールスマン問題をCPLEXで解く

- 結果([解]タブ)

```
// solution (optimal) with objective 9.8
// Quality Incumbent solution:
// MILP objective                                9.8000000000e+00
// MILP solution norm |x| (Total, Max)           8.00000e+00  1.00000e+00
// MILP solution error (Ax=b) (Total, Max)       0.00000e+00  0.00000e+00
// MILP x bound error (Total, Max)               0.00000e+00  0.00000e+00
// MILP x integrality error (Total, Max)         0.00000e+00  0.00000e+00
// MILP slack bound error (Total, Max)           0.00000e+00  0.00000e+00
//

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

最適値 = 9.8

最適解に部分巡回路がある

部分巡回路

①→③→①

②→④→②

⑤→⑥→⑤

⑦→⑧→⑦

TSPの部分巡回路除去(1)

- Traveling Salesman Problemの最適化(例1)
  - 完全グラフ $G = (V, E)$
  - 点集合 $V = \{1,2,\ldots,n\}$(例1はn=8)
  - 枝$(i, j) \in E$上のコスト(距離)d_{ij}
  - 全点を1度ずつ回る巡回路を生成
  - 最短の巡回路を求めたい

コスト(距離)d_{ij}(km)

1	0.0	2.0	1.0	1.9	1.5	1.4	1.8	1.7
2	2.0	0.0	2.3	1.3	1.2	1.7	1.8	1.9
3	1.0	2.3	0.0	1.3	1.2	1.7	1.8	1.9
4	1.9	1.3	1.3	0.0	1.9	2.7	0.8	1.8
5	1.5	1.2	1.2	1.9	0.0	1.4	2.4	2.3
6	1.4	1.7	1.7	2.7	1.4	0.0	2.4	2.0
7	1.8	1.8	1.8	0.8	2.4	2.4	0.0	1.2
8	1.7	1.9	1.9	1.8	2.3	2.0	1.2	0.0

- 最適化問題の定式化1(ベタ表記⇔Σ表記)

min. $d_{12}x_{12}+d_{13}x_{13}+\ldots+d_{78}x_{78}$ $\longleftrightarrow$ $\min.\sum_{i\neq j} d_{ij}x_{ij}$

s. t.
$x_{12}+x_{13}+\ldots+x_{18}=1$
$x_{21}+x_{23}+\ldots+x_{28}=1$
…
$x_{81}+x_{82}+\ldots+x_{87}=1$
$\longleftrightarrow$ s.t. $\sum_{j:j\neq i} x_{ij} = 1\ (\forall i \in V)$

点iを**出る**枝は丁度1本

$x_{21}+x_{31}+\ldots+x_{81}=1$
$x_{12}+x_{32}+\ldots+x_{82}=1$
…
$x_{18}+x_{28}+\ldots+x_{78}=1$
$\longleftrightarrow$ $\sum_{j:j\neq i} x_{ji} = 1\ (\forall i \in V)$

点iへ**入る**枝は丁度1本

部分巡回路除去制約

$x_{12}+x_{21} \leqq 2\text{-}1, \ldots$ $\longleftrightarrow$ $\sum_{(i,j)\in S} x_{ij} \leq |S| - 1\ (\forall S \subset E, |S| \geq 2)$

$x_{12}, x_{13}, \ldots, x_{78} \in \{0,1\}$ $\longleftrightarrow$ $x_{ij} \in \{0,1\}\ (\forall i, j: i \neq j)$

TSPの部分巡回路除去(1)をCPLEXで解く

- 結果([解]タブ)

```
// solution (optimal) with objective 9.8
// Quality Incumbent solution:
// MILP objective                                 9.8000000000e+00
// MILP solution norm |x| (Total, Max)            8.00000e+00  1.00000e+00
// MILP solution error (Ax=b) (Total, Max)        0.00000e+00  0.00000e+00
// MILP x bound error (Total, Max)                0.00000e+00  0.00000e+00
// MILP x integrality error (Total, Max)          0.00000e+00  0.00000e+00
// MILP slack bound error (Total, Max)            0.00000e+00  0.00000e+00
//

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

最適値 = 9.8

最適解に部分巡回路があるので除去制約を追加して解き直す

部分巡回路	除去制約
①→③→①	x[1,3] + x[3,1] <= 1;
②→④→②	x[2,4] + x[4,2] <= 1;
⑤→⑥→⑤	x[5,6] + x[6,5] <= 1;
⑦→⑧→⑦	x[7,8] + x[8,7] <= 1;

モデルファイル [TSP.mod]

```
  forall (i in I) {
    sum(j in I) x[i][j] == 1;// 点iから出る枝1本
    sum(j in I) x[j][i] == 1;// 点iに入る枝1本
  }
 // 部分巡回路除去制約
  x[1][3]+x[3][1] <= 1;
  x[2][4]+x[4][2] <= 1;
  x[5][6]+x[6][5] <= 1;
  x[7][8]+x[8][7] <= 1;
};
```

TSPの部分巡回路除去(1)をCPLEXで解く

- 結果([解]タブ) 2回目

```
// solution (optimal) with objective 10.1
// Quality Incumbent solution:
// MILP objective                                1.0100000000e+01
// MILP solution norm |x| (Total, Max)           8.00000e+00  1.00000e+00
// MILP solution error (Ax=b) (Total, Max)       0.00000e+00  0.00000e+00
// MILP x bound error (Total, Max)               0.00000e+00  0.00000e+00
// MILP x integrality error (Total, Max)         0.00000e+00  0.00000e+00
// MILP slack bound error (Total, Max)           0.00000e+00  0.00000e+00
//

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

最適値 = 10.1

最適解

最適解に部分巡回路がないので終了
得られた距離最小の巡回路
①→③→⑤→②→④→⑦→⑧→⑥→①

TSPの部分巡回路除去(1)をCPLEXで解く

- Traveling Salesman Problemの最適化(例2)
  - cplex: model file[TSP.mod]
  - cplex: data file[TSPex2.dat]

no	距離c_{ij}(km)	川崎	鹿島	セ大	柏	磐田	浦和	鳥栖	神戸	札幌	仙台
1	川崎	0.0	100.0	393.9	41.3	186.0	34.8	874.1	419.1	838.8	323.3
2	鹿島	100.0	0.0	489.0	63.1	287.9	84.9	968.7	517.6	783.7	257.9
3	セ大	393.9	489.0	0.0	429.0	217.2	406.8	480.4	32.3	1063.3	633.0
4	柏	41.3	63.1	429.0	0.0	228.1	25.1	907.6	454.5	805.3	288.6
5	磐田	186.0	287.9	217.2	228.1	0.0	214.5	694.7	247.8	974.2	482.0
6	浦和	34.8	84.9	406.8	25.1	214.5	0.0	886.7	435.9	803.0	288.1
7	鳥栖	874.1	968.7	480.4	907.6	694.7	886.7	0.0	450.1	1432.8	1086.1
8	神戸	419.1	517.6	32.3	454.5	247.8	435.9	450.1	0.0	1077.7	653.7
9	札幌	838.8	783.7	1063.3	805.3	974.2	803.0	1432.8	1077.7	0.0	523.7
10	仙台	323.3	257.9	633.0	288.6	482.0	288.1	1086.1	653.7	523.7	0.0

TSPの部分巡回路除去(2)

- Traveling Salesman Problemの最適化(例3)

コスト(距離)d_{ij}(km)

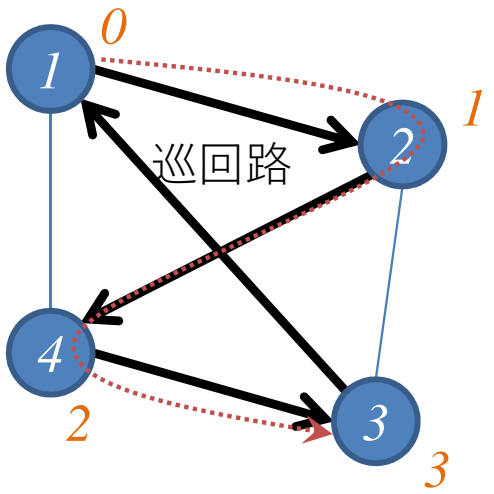
no	距離c_{ij}(km)	文教大	妙伝寺	宝蔵寺	来迎寺	白峰寺	正覚院	蓮妙寺	善谷寺
1	文教大学	0.0000	1.5000	0.5670	0.8464	0.7887	0.8142	0.8981	0.7531
2	妙伝寺（毘沙門天）	1.5000	0.0000	2.0173	2.2708	1.9024	1.0052	2.1164	1.3666
3	宝蔵寺（大黒天）	0.5670	2.0173	0.0000	0.7340	0.5411	1.1590	1.0315	1.2537
4	来迎寺（恵比寿神）	0.8464	2.2708	0.7340	0.0000	1.2711	1.6561	0.4210	1.0772
5	白峰寺（寿老人）	0.7887	1.9024	0.5411	1.2711	0.0000	0.9146	1.5277	1.5393
6	正覚院（布袋尊）	0.8142	1.0052	1.1590	1.6561	0.9146	0.0000	1.6881	1.2355
7	蓮妙寺（弁財天）	0.8981	2.1164	1.0315	0.4210	1.5277	1.6881	0.0000	0.7947
8	善谷寺（福禄寿）	0.7531	1.3666	1.2537	1.0772	1.5393	1.2355	0.7947	0.0000

- 部分巡回路除去(2) ポテンシャル定式化（変数設定・係数表記）
  - 距離行列 distance matrix $\boldsymbol{D}=[d_{ij}]$
  - 0-1変数 $x_{ij} = \begin{cases} 1 & \cdots \text{枝}(i,j)\text{を通る} \\ 0 & \cdots \text{枝}(i,j)\text{を通らない} \end{cases}$
  - 実数変数 $u_i \in [0, n-1]$　※点iに対するポテンシャル変数 potential variables

ポテンシャル定式化のための変数

TSPの部分巡回路除去(2)

<4点の例>
巡回路に訪問順の
ポテンシャルがつく



- Traveling Salesman Problemの最適化

- 最適化問題の定式化2(ベタ表記⇔Σ表記)

min. $d_{12}x_{12}+d_{13}x_{13}+\ldots+d_{78}x_{78}$ $\longleftrightarrow$ $\min.\sum_{i\neq j} d_{ij}x_{ij}$

s. t. $x_{12}+x_{13}+\ldots+x_{18}=1$
$\ldots$
$x_{81}+x_{82}+\ldots+x_{87}=1$
$\longleftrightarrow$ s.t. $\sum_{j:j\neq i} x_{ij} = 1\ (\forall i \in V)$

点iを**出る**枝は丁度1本

$x_{21}+x_{31}+\ldots+x_{81}=1$
$\ldots$
$x_{18}+x_{28}+\ldots+x_{78}=1$
$\longleftrightarrow$ $\sum_{j:j\neq i} x_{ji} = 1\ (\forall i \in V)$

点iへ**入る**枝は丁度1本

$u_1=0$ $\longleftrightarrow$ $u_1 = 0$

ポテンシャル制約

$u_1+1-(n-1)(1-x_{12}) \leqq u_2$
$\ldots$
$u_7+1-(n-1)(1-x_{78}) \leqq u_8$
$\longleftrightarrow$ $u_i + 1 - (n-1)(1-x_{ij}) \leq u_j$
$(\forall i \in V, \forall j \in V/\{1\}, i \neq j)$

$1 \leqq u_2 \leqq n-1$
$\ldots$
$1 \leqq u_2 \leqq n-1$
$\longleftrightarrow$ $1 \leq u_i \leq n-1\ (\forall i \in V/\{1\})$

$x_{12}, x_{13}, \ldots, x_{78} \in \{0,1\}$ $\longleftrightarrow$ $x_{ij} \in \{0,1\}\ (\forall i,j: i \neq j)$

TSPの部分巡回路除去(2)をCPLEXで解く

- 既存プロジェクト [TSP] を開く
- モデルファイルをコピー＆ペースト
  - ✓ プロジェクト内のモデルファイル [TSP.mod] を選択する
  - ✓ [Ctrl]+[c] でコピーし，そのまま [Ctrl]+[v] でペースト．名前を [TSPpot.mod]とする
- [TSPpot.mod]作成
  - ✓ ポテンシャル変数を追加
  - ✓ 部分巡回路除去制約を削除し，ポテンシャル制約を追加
- [TSPex3.dat]作成
- 解く

モデルファイル [TSPpot.mod]

```
int i_max = ...;// 頂点数|V|
range I = 1..i_max;
float d[I,I] = ...;// 距離行列D=[dij]
dvar int+ x[I,I] in 0..1;// 0-1変数
dvar float+ u[I];// ポテンシャル変数

minimize
  sum(i in I) sum(j in I:i!=j) d[i][j]*x[i][j];
subject to {
  forall (i in I) {
    x[i][i] == 0;// 自己ループ(i→iの枝)はなし
  }
  forall (i in I) {
    sum(j in I) x[i][j] == 1;// 点iから出て行く枝は1本
    sum(j in I) x[j][i] == 1;// 点iに入ってくる枝は1本
  }
  // ポテンシャル制約
  forall(i in I) {
    forall(j in 2..i_max:i!=j) {
      u[i] + 1 - (i_max - 1)*(1 - x[i][j]) <= u[j];
    }
  }
  u[1]==0;
  forall(i in 2..i_max) {
    u[i] >= 1;
    u[i] <= i_max - 1;
  }
};
```

データファイル[TSPex3.dat]

```
i_max = 8;

d = [
[ 0.0000 1.5000 0.5670 0.8464 0.7887 0.8142 0.8981 0.7531 ]
[ 1.5000 0.0000 2.0173 2.2708 1.9024 1.0052 2.1164 1.3666 ]
[ 0.5670 2.0173 0.0000 0.7340 0.5411 1.1590 1.0315 1.2537 ]
[ 0.8464 2.2708 0.7340 0.0000 1.2711 1.6561 0.4210 1.0772 ]
[ 0.7887 1.9024 0.5411 1.2711 0.0000 0.9146 1.5277 1.5393 ]
[ 0.8142 1.0052 1.1590 1.6561 0.9146 0.0000 1.6881 1.2355 ]
[ 0.8981 2.1164 1.0315 0.4210 1.5277 1.6881 0.0000 0.7947 ]
[ 0.7531 1.3666 1.2537 1.0772 1.5393 1.2355 0.7947 0.0000 ]
];
```

TSPの部分巡回路除去(2)をCPLEXで解く

- 結果([解]タブ)

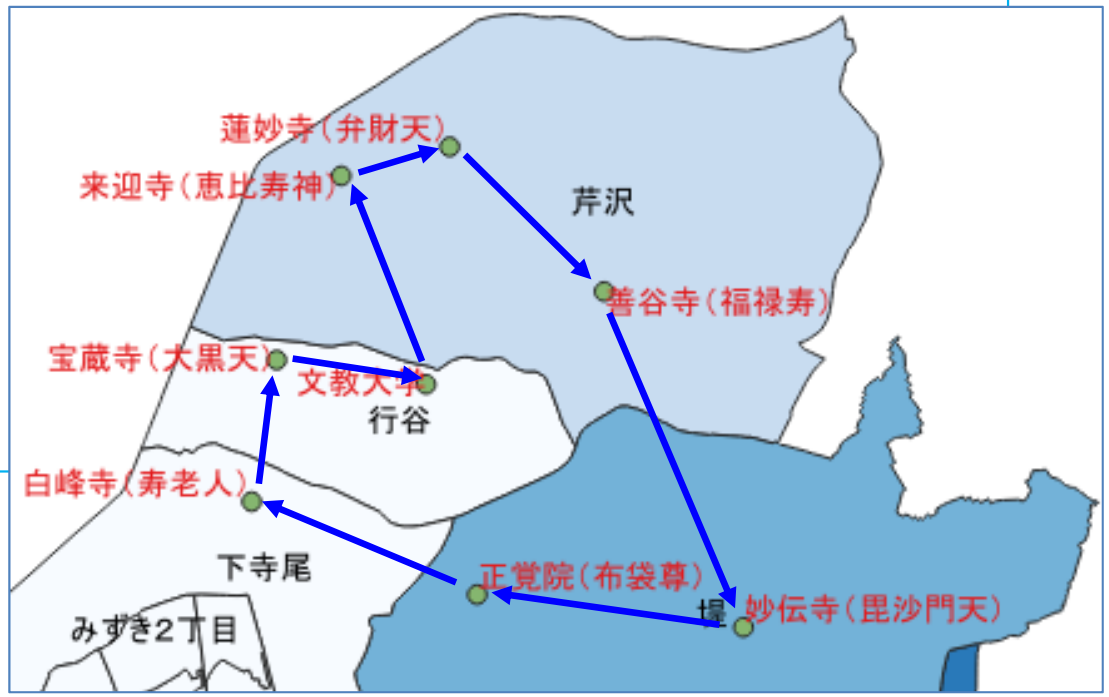
最適値 = 6.4566

```
// solution (optimal) with objective 6.4566
// Quality Incumbent solution:
// MILP objective                                 6.4566000000e+00
// MILP solution norm |x| (Total, Max)            3.60000e+01  7.00000e+00
// MILP solution error (Ax=b) (Total, Max)        0.00000e+00  0.00000e+00
// MILP x bound error (Total, Max)                0.00000e+00  0.00000e+00
// MILP x integrality error (Total, Max)          0.00000e+00  0.00000e+00
// MILP slack bound error (Total, Max)            0.00000e+00  0.00000e+00
//

x = [[0 0 0 1 0 0 0 0]
     [0 0 0 0 0 1 0 0]
     [1 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 1 0]
     [0 0 1 0 0 0 0 0]
     [0 0 0 0 1 0 0 0]
     [0 0 0 0 0 0 0 1]
     [0 1 0 0 0 0 0 0]];
u = [0 4 7 1 6 5 2 3];
```

最適解

1	文教大学
2	妙伝寺（毘沙門天）
3	宝蔵寺（大黒天）
4	来迎寺（恵比寿神）
5	白峰寺（寿老人）
6	正覚院（布袋尊）
7	蓮妙寺（弁財天）
8	善谷寺（福禄寿）



得られた距離最小の巡回路
①→④→⑦→⑧→②→⑥→⑤→③→①

なお，巡回路は
0-1変数 x[i][j] =1 をつないでいく，x[1][4] → x[4][7] → x[7][8] → x[8][2] →… か
ポテンシャル変数 u[i] の値の小さい順(0,1,2,3,…,7)をたどれば得られる
※問題が対称TSPなので，最適巡回路は逆順でも可

TSPの部分巡回路除去(2)をCPLEXで解く

- Traveling Salesman Problemの最適化(例2)
  - cplex: model file[TSPpot.mod]
  - cplex: data file[TSPex2.dat]

no	距離c_{ij}(km)	川崎	鹿島	セ大	柏	磐田	浦和	鳥栖	神戸	札幌	仙台
1	川崎	0.0	100.0	393.9	41.3	186.0	34.8	874.1	419.1	838.8	323.3
2	鹿島	100.0	0.0	489.0	63.1	287.9	84.9	968.7	517.6	783.7	257.9
3	セ大	393.9	489.0	0.0	429.0	217.2	406.8	480.4	32.3	1063.3	633.0
4	柏	41.3	63.1	429.0	0.0	228.1	25.1	907.6	454.5	805.3	288.6
5	磐田	186.0	287.9	217.2	228.1	0.0	214.5	694.7	247.8	974.2	482.0
6	浦和	34.8	84.9	406.8	25.1	214.5	0.0	886.7	435.9	803.0	288.1
7	鳥栖	874.1	968.7	480.4	907.6	694.7	886.7	0.0	450.1	1432.8	1086.1
8	神戸	419.1	517.6	32.3	454.5	247.8	435.9	450.1	0.0	1077.7	653.7
9	札幌	838.8	783.7	1063.3	805.3	974.2	803.0	1432.8	1077.7	0.0	523.7
10	仙台	323.3	257.9	633.0	288.6	482.0	288.1	1086.1	653.7	523.7	0.0

TSPの部分巡回路除去(2)をCPLEXで解く

- 結果([解]タブ)

最適値 = 3223

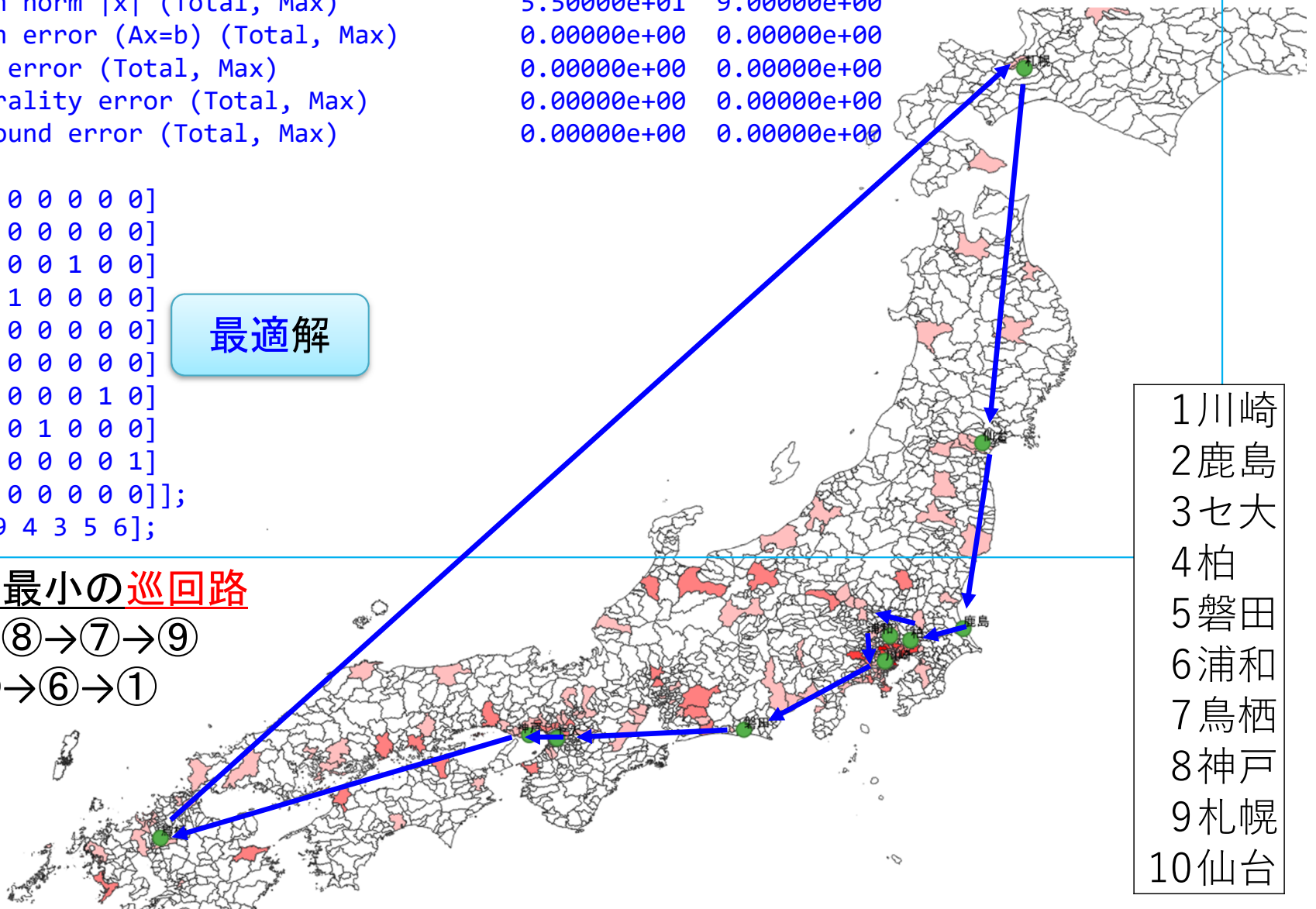
```
// solution (optimal) with objective 3223
// Quality Incumbent solution:
// MILP objective                                3.2230000000e+03
// MILP solution norm |x| (Total, Max)           5.50000e+01  9.00000e+00
// MILP solution error (Ax=b) (Total, Max)       0.00000e+00  0.00000e+00
// MILP x bound error (Total, Max)               0.00000e+00  0.00000e+00
// MILP x integrality error (Total, Max)         0.00000e+00  0.00000e+00
// MILP slack bound error (Total, Max)           0.00000e+00  0.00000e+00

x = [[0 0 0 0 1 0 0 0 0 0]
     [0 0 0 1 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 1 0 0]
     [0 0 0 0 0 1 0 0 0 0]
     [0 0 1 0 0 0 0 0 0 0]
     [1 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 1 0]
     [0 0 0 0 0 0 1 0 0 0]
     [0 0 0 0 0 0 0 0 0 1]
     [0 1 0 0 0 0 0 0 0 0]];
u = [0 7 2 8 1 9 4 3 5 6];
```

最適解

得られた距離最小の巡回路

①→⑤→③→⑧→⑦→⑨
→⑩→②→④→⑥→①



1川崎
2鹿島
3セ大
4柏
5磐田
6浦和
7鳥栖
8神戸
9札幌
10仙台

TSPの部分巡回路除去(3)

- Traveling Salesman Problemの最適化(例3)

コスト(距離)d_{ij}(km)

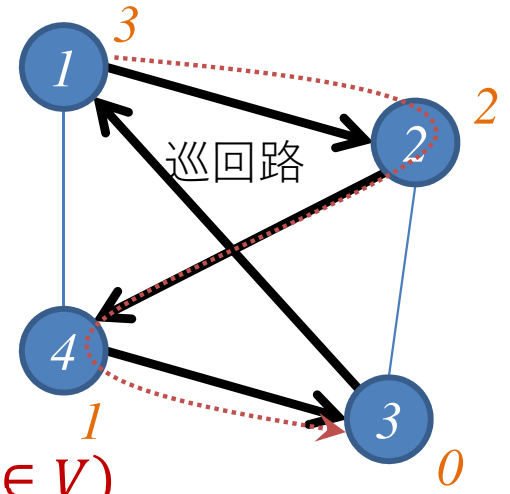
no	距離c_{ij}(km)	文教大	妙伝寺	宝蔵寺	来迎寺	白峰寺	正覚院	蓮妙寺	善谷寺
1	文教大学	0.0000	1.5000	0.5670	0.8464	0.7887	0.8142	0.8981	0.7531
2	妙伝寺（毘沙門天）	1.5000	0.0000	2.0173	2.2708	1.9024	1.0052	2.1164	1.3666
3	宝蔵寺（大黒天）	0.5670	2.0173	0.0000	0.7340	0.5411	1.1590	1.0315	1.2537
4	来迎寺（恵比寿神）	0.8464	2.2708	0.7340	0.0000	1.2711	1.6561	0.4210	1.0772
5	白峰寺（寿老人）	0.7887	1.9024	0.5411	1.2711	0.0000	0.9146	1.5277	1.5393
6	正覚院（布袋尊）	0.8142	1.0052	1.1590	1.6561	0.9146	0.0000	1.6881	1.2355
7	蓮妙寺（弁財天）	0.8981	2.1164	1.0315	0.4210	1.5277	1.6881	0.0000	0.7947
8	善谷寺（福禄寿）	0.7531	1.3666	1.2537	1.0772	1.5393	1.2355	0.7947	0.0000

- 部分巡回路除去(3)単一品種流定式化（変数設定・係数表記）
  - 距離行列 distance matrix $\boldsymbol{D}=[d_{ij}]$
  - 0-1変数 $x_{ij} = \begin{cases} 1 & \cdots \text{枝}(i,j)\text{を通る} \\ 0 & \cdots \text{枝}(i,j)\text{を通らない} \end{cases}$
  - 実数変数 $f_{ij} \in [0, n-1]$　※点$i \to j$に対するフロー変数 flow variables

単一品種流定式化のためのフロー変数

TSPの部分巡回路除去(3)

<4点の例>
点1に流量3のフローを流す
各点は1ずつ消費



- Traveling Salesman Problemの最適化

- 最適化問題の定式化3(ベタ表記⇔Σ表記)

$\min.\ d_{12}x_{12}+d_{13}x_{13}+\ldots+d_{78}x_{78} \longleftrightarrow \min.\sum_{i\neq j} d_{ij}x_{ij}$

s. t.
$x_{12}+x_{13}+\ldots+x_{18}=1$
$\ldots$
$x_{81}+x_{82}+\ldots+x_{87}=1$
$\longleftrightarrow$ s.t. $\sum_{j:j\neq i} x_{ij} = 1\ (\forall i \in V)$

点iを出る枝は丁度1本

$x_{21}+x_{31}+\ldots+x_{81}=1$
$\ldots$
$x_{18}+x_{28}+\ldots+x_{78}=1$
$\longleftrightarrow \sum_{j:j\neq i} x_{ji} = 1\ (\forall i \in V)$

点iへ入る枝は丁度1本

$f_{12}+f_{13}+\ldots+f_{18}=7 \longleftrightarrow \sum_{j=2}^{n} f_{1j} = n-1$

単一フロー制約

$(f_{12}+f_{32}+\ldots+f_{82}) - (f_{21}+f_{23}+\ldots+f_{28})=1$
$\ldots$
$(f_{18}+f_{28}+\ldots+f_{78}) - (f_{81}+f_{82}+\ldots+f_{87})=1$
$\longleftrightarrow \sum_{j:j\neq i} f_{ji} - \sum_{j:j\neq i} f_{ij} = 1\ (\forall i = 2,\ldots,n)$

$f_{12}\leqq 7x_{12},\ldots,f_{18}\leqq 7x_{18} \longleftrightarrow f_{1j} \leq (n-1)x_{1j}\ (\forall j \neq 1)$

$f_{23}\leqq 6x_{23},\ldots,f_{28}\leqq 6x_{28}$
$\ldots$
$f_{82}\leqq 6x_{82},\ldots,f_{87}\leqq 6x_{87}$
$\longleftrightarrow f_{ij} \leq (n-2)x_{ij}\ (\forall i \neq j, i \neq 1, j \neq 1)$

$f_{12}, f_{13}, \ldots, f_{87}\geqq 0 \longleftrightarrow f_{ij} \geq 0\ (\forall i \neq j)$

$x_{12}, x_{13}, \ldots, x_{78} \in \{0,1\} \longleftrightarrow x_{ij} \in \{0,1\}\ (\forall i,j: i \neq j)$

TSPの部分巡回路除去(3)をCPLEXで解く

- 既存プロジェクト [TSP] を開く
- モデルファイルをコピー＆ペースト
  - ✓ プロジェクト内のモデルファイル [TSP.mod] を選択する
  - ✓ [Ctrl]+[c] でコピーし，そのまま [Ctrl]+[v] でペースト．名前を [TSPpot.mod]とする
- [TSPflo.mod]作成
  - ✓ フロー変数を追加
  - ✓ 部分巡回路除去制約を削除し，単一品種流制約を追加
- [TSPex3.dat]を利用
- 解く

モデルファイル [TSPflo.mod]

```
int i_max = ...;// 頂点数|V|
range I = 1..i_max;
float d[I,I] = ...;// 距離行列D=[dij]
dvar int+ x[I,I] in 0..1;// 0-1変数
dvar float+ flow[I,I];// フロー変数

minimize
  sum(i in I) sum(j in I:i!=j) d[i][j]*x[i][j];
subject to {
  forall (i in I) {
    x[i][i] == 0;// 自己ループ(i→iの枝)はなし
    flow[i][i] == 0.0;
  }
  forall (i in I) {
    sum(j in I) x[i][j] == 1;// 点iから出て行く枝は1本
    sum(j in I) x[j][i] == 1;// 点iに入ってくる枝は1本
  }
  // フロー制約
  sum(j in 2..i_max) flow[1][j] == i_max - 1;
  forall(i in 2..i_max) {
    sum(j in I:i!=j) flow[j][i] - sum(j in I:i!=j) flow[i][j] ==1;
  }
  forall(j in 2..i_max) {
    flow[1][j] <= (i_max - 1)*x[1][j];
  }
  forall(i in 2..i_max) {
    forall(j in 2..i_max) {
      flow[i][j] <= (i_max - 2)*x[i][j];
    }
  }
};
```

データファイル[TSPex3.dat](既出)

```
i_max = 8;

d = [
[ 0.0000 1.5000 0.5670 0.8464 0.7887 0.8142 0.8981 0.7531 ]
[ 1.5000 0.0000 2.0173 2.2708 1.9024 1.0052 2.1164 1.3666 ]
[ 0.5670 2.0173 0.0000 0.7340 0.5411 1.1590 1.0315 1.2537 ]
[ 0.8464 2.2708 0.7340 0.0000 1.2711 1.6561 0.4210 1.0772 ]
[ 0.7887 1.9024 0.5411 1.2711 0.0000 0.9146 1.5277 1.5393 ]
[ 0.8142 1.0052 1.1590 1.6561 0.9146 0.0000 1.6881 1.2355 ]
[ 0.8981 2.1164 1.0315 0.4210 1.5277 1.6881 0.0000 0.7947 ]
[ 0.7531 1.3666 1.2537 1.0772 1.5393 1.2355 0.7947 0.0000 ]
];
```

TSPの部分巡回路除去(3)をCPLEXで解く

- 結果([解]タブ)

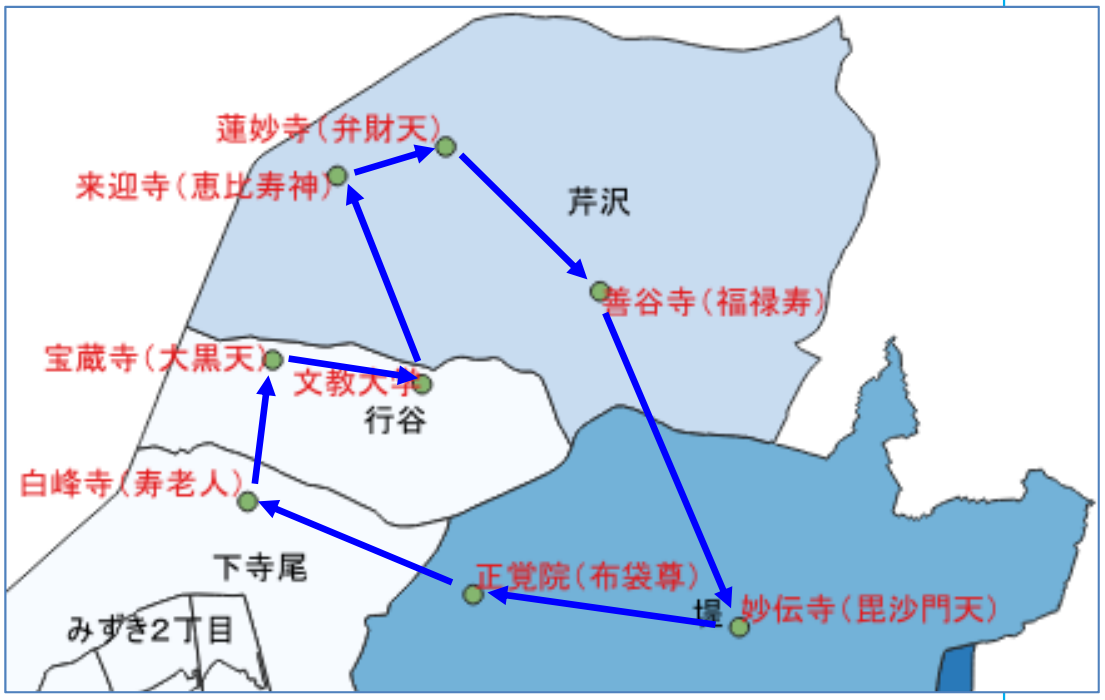
最適値 = 6.4566

```
// solution (optimal) with objective 6.4566
// Quality Incumbent solution:
// MILP objective                                6.4566000000e+00
// MILP solution norm |x| (Total, Max)           3.60000e+01  7.00000e+00
// MILP solution error (Ax=b) (Total, Max)       0.00000e+00  0.00000e+00
// MILP x bound error (Total, Max)               0.00000e+00  0.00000e+00
// MILP x integrality error (Total, Max)         0.00000e+00  0.00000e+00
// MILP slack bound error (Total, Max)           0.00000e+00  0.00000e+00

x = [[0 0 0 1 0 0 0 0]
     [0 0 0 0 0 1 0 0]
     [1 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 1 0]
     [0 0 1 0 0 0 0 0]
     [0 0 0 0 1 0 0 0]
     [0 0 0 0 0 0 0 1]
     [0 1 0 0 0 0 0 0]];
flow = [[0 0 0 7 0 0 0 0]
        [0 0 0 0 0 3 0 0]
        [0 0 0 0 0 0 0 0]
        [0 0 0 0 0 0 6 0]
        [0 0 1 0 0 0 0 0]
        [0 0 0 0 2 0 0 0]
        [0 0 0 0 0 0 0 5]
        [0 4 0 0 0 0 0 0]];
```

最適解

1	文教大学
2	妙伝寺（毘沙門天）
3	宝蔵寺（大黒天）
4	来迎寺（恵比寿神）
5	白峰寺（寿老人）
6	正覚院（布袋尊）
7	蓮妙寺（弁財天）
8	善谷寺（福禄寿）



得られた距離最小の巡回路:①→④→⑦→⑧→②→⑥→⑤→③→①

なお，巡回路は0-1変数 x[i][j] =1 をつないでいくか，

フロー変数 flow[i][j] の値を降順(7,6,5,…,0)に行番号をたどると得られる

TSPの部分巡回路除去(3)をCPLEXで解く

- Traveling Salesman Problemの最適化(例4)
  - cplex: model file[TSPflo.mod]
  - cplex: data file[TSPex4.dat]

川崎	0.0	100.0	391.9	41.3	188.0	35.8	873.1	421.1	840.8	323.3	14.3	673.8	96.3	261.1	36.8
鹿島	100.0	0.0	490.0	62.1	287.9	83.9	970.7	518.6	784.7	259.9	106.9	769.3	189.6	253.1	91.1
セ大	391.9	490.0	0.0	428.0	216.2	407.8	481.4	32.3	1065.3	633.0	383.3	287.0	301.5	483.3	401.2
柏	41.3	62.1	428.0	0.0	228.1	24.1	908.6	456.5	807.3	286.6	45.4	707.4	127.7	240.7	31.7
磐田	188.0	287.9	216.2	228.1	0.0	212.5	694.7	247.8	972.2	482.0	183.1	503.2	119.2	367.2	207.6
浦和	35.8	83.9	407.8	24.1	212.5	0.0	887.7	435.9	805.0	288.1	31.6	685.7	106.5	228.0	7.6
鳥栖	873.1	970.7	481.4	908.6	694.7	887.7	0.0	452.1	1432.8	1084.1	864.3	210.1	781.3	920.9	880.9
神戸	421.1	518.6	32.3	456.5	247.8	435.9	452.1	0.0	1075.7	653.7	412.1	255.5	329.4	500.8	429.2
札幌	840.8	784.7	1065.3	807.3	972.2	805.0	1432.8	1075.7	0.0	524.7	834.2	1233.0	858.0	605.0	804.9
仙台	323.3	259.9	633.0	286.6	482.0	288.1	1084.1	653.7	524.7	0.0	319.1	874.1	362.8	166.9	289.5
東京	14.3	106.9	383.3	45.4	183.1	31.6	864.3	412.1	834.2	319.1	0.0	664.0	84.9	250.4	29.6
広島	673.8	769.3	287.0	707.4	503.2	685.7	210.1	255.5	1233.0	874.1	664.0	0.0	579.7	710.8	678.7
甲府	96.3	189.6	301.5	127.7	119.2	106.5	781.3	329.4	858.0	362.8	84.9	579.7	0.0	255.1	99.8
新潟	261.1	253.1	483.3	240.7	367.2	228.0	920.9	500.8	605.0	166.9	250.4	710.8	255.1	0.0	224.8
大宮	36.8	91.1	401.2	31.7	207.6	7.6	880.9	429.2	804.9	289.5	29.6	678.7	99.8	224.8	0.0

TSPの部分巡回路除去(3)をCPLEXで解く

- 結果([解]タブ)※一部省略

最適値 = 3360.1

```
// solution (optimal) with objective 3360.1

x = [[0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
     [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
     [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]
     [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
     [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
     [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
     [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
     [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
     [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]];
flow = [[0 0 0 0 0 0 0 0 0 0 0 0 0 0 14]
        [0 0 0 0 0 0 0 0 0 10 0 0 0 0 0]
        [0 0 0 0 3 0 0 0 0 0 0 0 0 0 0]
        [0 11 0 0 0 0 0 0 0 0 0 0 0 0 0]
        [0 0 0 0 0 0 0 0 0 0 0 0 2 0 0]
        [0 0 0 12 0 0 0 0 0 0 0 0 0 0 0]
        [0 0 0 0 0 0 0 5 0 0 0 0 0 0 0]
        [0 0 4 0 0 0 0 0 0 0 0 0 0 0 0]
        [0 0 0 0 0 0 0 0 0 0 0 0 0 8 0]
        [0 0 0 0 0 0 0 0 9 0 0 0 0 0 0]
        [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
        [0 0 0 0 0 0 6 0 0 0 0 0 0 0 0]
        [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
        [0 0 0 0 0 0 0 0 0 0 0 7 0 0 0]
        [0 0 0 0 0 13 0 0 0 0 0 0 0 0 0]];
```

最適解

得られた距離最小の巡回路

①川崎→⑮大宮→⑥浦和→④柏→②鹿島
→⑩仙台→⑨札幌→⑭新潟→⑫広島→⑦鳥栖
→⑧神戸→③大坂→⑤磐田→⑬甲府→⑪東京→①

1 川崎
2 鹿島
3 セ大
4 柏
5 磐田
6 浦和
7 鳥栖
8 神戸
9 札幌
10 仙台
11 東京
12 広島
13 甲府
14 新潟
15 大宮

TSPの部分巡回路除去(4)

- Traveling Salesman Problemの最適化(例3)

コスト(距離)d_{ij}(km)

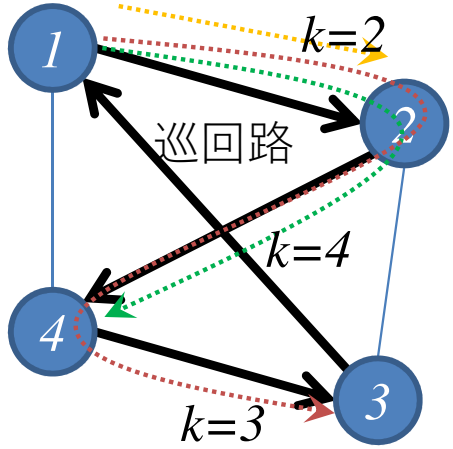
no	距離c_{ij}(km)	文教大	妙伝寺	宝蔵寺	来迎寺	白峰寺	正覚院	蓮妙寺	善谷寺
1	文教大学	0.0000	1.5000	0.5670	0.8464	0.7887	0.8142	0.8981	0.7531
2	妙伝寺（毘沙門天）	1.5000	0.0000	2.0173	2.2708	1.9024	1.0052	2.1164	1.3666
3	宝蔵寺（大黒天）	0.5670	2.0173	0.0000	0.7340	0.5411	1.1590	1.0315	1.2537
4	来迎寺（恵比寿神）	0.8464	2.2708	0.7340	0.0000	1.2711	1.6561	0.4210	1.0772
5	白峰寺（寿老人）	0.7887	1.9024	0.5411	1.2711	0.0000	0.9146	1.5277	1.5393
6	正覚院（布袋尊）	0.8142	1.0052	1.1590	1.6561	0.9146	0.0000	1.6881	1.2355
7	蓮妙寺（弁財天）	0.8981	2.1164	1.0315	0.4210	1.5277	1.6881	0.0000	0.7947
8	善谷寺（福禄寿）	0.7531	1.3666	1.2537	1.0772	1.5393	1.2355	0.7947	0.0000

- 部分巡回路除去(4)多品種流定式化（変数設定・係数表記）
  - 距離行列 distance matrix $\boldsymbol{D}=[d_{ij}]$
  - 0-1変数 $x_{ij} = \begin{cases} 1 & \cdots \text{枝}(i,j)\text{を通る} \\ 0 & \cdots \text{枝}(i,j)\text{を通らない} \end{cases}$
  - 実数変数 $f_{ij}^{k} \geq 0$ ※品種kを点$i \to j$への多品種フロー変数 multi-flow variables

多品種流定式化のための多品種フロー変数

TSPの部分巡回路除去(4)

<4点の例>
点1に3種のフローを流す
各点は対応品種を受領



- Traveling Salesman Problemの最適化

- 最適化問題の定式化4(ベタ表記⇔Σ表記)

$\min.\ d_{12}x_{12}+d_{13}x_{13}+\ldots+d_{78}x_{78} \quad \Leftrightarrow \quad \min.\sum_{i\neq j} d_{ij}x_{ij}$

s. t.
$x_{12}+x_{13}+\ldots+x_{18}=1$
$\ldots$
$x_{81}+x_{82}+\ldots+x_{87}=1$

$\Leftrightarrow$ s.t. $\sum_{j:j\neq i} x_{ij} = 1\ (\forall i \in V)$

点iを出る枝は丁度1本

$x_{21}+x_{31}+\ldots+x_{81}=1$
$\ldots$
$x_{18}+x_{28}+\ldots+x_{78}=1$

$\Leftrightarrow \quad \sum_{j:j\neq i} x_{ji} = 1\ (\forall i \in V)$

点iへ入る枝は丁度1本

多品種フロー制約

$(f^2_{21}+f^2_{31}+\ldots+f^2_{81}) - (f^2_{12}+f^2_{13}+\ldots+f^2_{18}) = -1$
$\ldots$
$(f^2_{13}+f^2_{23}+\ldots+f^2_{83}) - (f^2_{31}+f^2_{32}+\ldots+f^2_{38}) = 0$
$\ldots$
$(f^2_{12}+f^2_{32}+\ldots+f^2_{82}) - (f^2_{21}+f^2_{23}+\ldots+f^2_{28}) = 1$
$\ldots$

$\Leftrightarrow \quad \sum_{j:j\neq i} f^k_{ji} - \sum_{j:j\neq i} f^k_{ij} = \begin{cases} -1 (i=1) \\ 0 (i\neq 1, i\neq k) \\ 1 (i=k) \end{cases} (\forall k = 2, \ldots, n)$

$f^1_{12} \leqq x_{12}, f^1_{13} \leqq x_{13}, \ldots, f^1_{18} \leqq x_{18}$
$\ldots$
$f^7_{81} \leqq x_{81}, f^7_{82} \leqq x_{82}, \ldots, f^7_{87} \leqq x_{87}$

$\Leftrightarrow \quad f^k_{ij} \leq x_{ij}\ (\forall k, i, j)$

$f_{12}, f_{13}, \ldots, f_{87} \geqq 0$

$\Leftrightarrow \quad f^k_{ij} \geq 0\ (\forall k, i, j: i \neq j, j \neq 1)$

$x_{12}, x_{13}, \ldots, x_{78} \in \{0,1\}$

$\Leftrightarrow \quad x_{ij} \in \{0,1\}\ (\forall i, j: i \neq j)$

TSPの部分巡回路

- 既存プロジェクト [TSP] 開く
- モデルをコピー＆ペースト
  - ✓ プロジェクト内のモデルファイル [TSP.mod] を選択する
  - ✓ [Ctrl]+[c] でコピーし，そのまま[Ctrl]+[v] でペースト．名前を[TSPmfl.mod]とする
- [TSPmfl.mod]作成
  - ✓ フロー変数を追加
  - ✓ 部分巡回路除去制約を削除し，多品種流制約を追加
- データは[TSPex3.dat]を利用
- 解く

モデルファイル
[TSPmfl.mod]

データファイル
[TSPex3.dat]

```
int i_max = ...;// 頂点数|V|
range I = 1..i_max;
range K = 2..i_max;
float d[I,I] = ...;// 距離行列D=[dij]
dvar int+ x[I,I] in 0..1;// 0-1変数
dvar float+ mflow[K,I,I];// 多品種フロー変数

minimize
  sum(i in I) sum(j in I:i!=j) d[i][j]*x[i][j];
subject to {
  forall (i in I) {
    x[i][i] == 0;// 自己ループ(i→iの枝)はなし
    forall (k in K) {
      mflow[k][i][i] == 0.0;
    }
  }
  forall (i in I) {
    sum(j in I) x[i][j] == 1;// 点iから出て行く枝は1本
    sum(j in I) x[j][i] == 1;// 点iに入ってくる枝は1本
  }
  // 多品種フロー制約
  forall(k in K) {
    sum(j in K) mflow[k][j][1] - sum(j in K) mflow[k][1][j] == -1;
    forall(i in K:i!=k) {
     sum(j in I:j!=i) mflow[k][j][i] - sum(j in I:j!=i) mflow[k][i][j] == 0;
    }
   sum(j in I:j!=k) mflow[k][j][k] - sum(j in I:j!=k) mflow[k][k][j] == 1;
  }
  forall(k in K) {
    forall(i in I) {
      forall(j in I) {
        mflow[k][i][j] <= x[i][j];
      }
    }
  }
};
```

TSPの部分巡回路除去(4)をCPLEXで解く

➢ 結果([解]タブ)

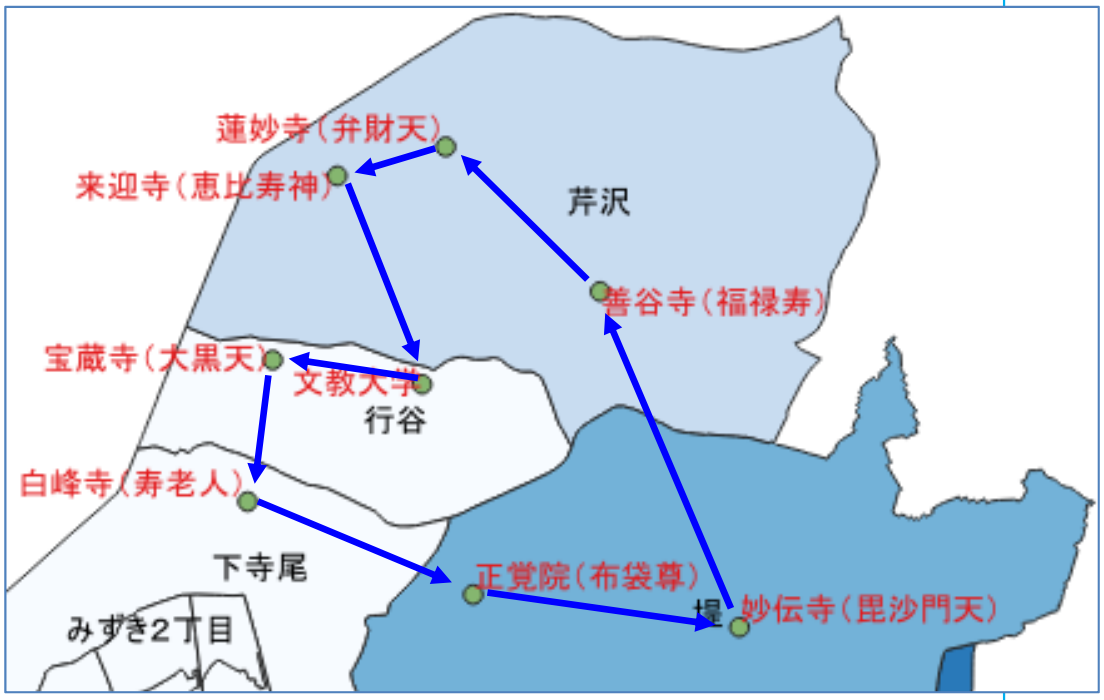
最適値 = 6.4566

```
// solution (optimal) with objective 6.4566
// Quality Incumbent solution:
// MILP objective                                  6.4566000000e+00
// MILP solution norm |x| (Total, Max)             3.60000e+01  1.00000e+00
// MILP solution error (Ax=b) (Total, Max)         0.00000e+00  0.00000e+00
// MILP x bound error (Total, Max)                 0.00000e+00  0.00000e+00
// MILP x integrality error (Total, Max)           0.00000e+00  0.00000e+00
// MILP slack bound error (Total, Max)             0.00000e+00  0.00000e+00
//

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

最適解

1	文教大学
2	妙伝寺（毘沙門天）
3	宝蔵寺（大黒天）
4	来迎寺（恵比寿神）
5	白峰寺（寿老人）
6	正覚院（布袋尊）
7	蓮妙寺（弁財天）
8	善谷寺（福禄寿）



距離最小巡回路:①→③→⑤→⑥→②→⑧→⑦→④→①

なお，巡回路は0-1変数 x[i][j] =1 をつないでいくか，多品種フロー変数 mflow[k][i][j] の値1が増えていく[k]順となる

TSPの部分巡回路除去(4)をCPLEXで解く

- Traveling Salesman Problemの最適化(例4)
  - cplex: model file[TSPmfl.mod]
  - cplex: data file[TSPex4.dat]

川崎	0.0	100.0	391.9	41.3	188.0	35.8	873.1	421.1	840.8	323.3	14.3	673.8	96.3	261.1	36.8
鹿島	100.0	0.0	490.0	62.1	287.9	83.9	970.7	518.6	784.7	259.9	106.9	769.3	189.6	253.1	91.1
セ大	391.9	490.0	0.0	428.0	216.2	407.8	481.4	32.3	1065.3	633.0	383.3	287.0	301.5	483.3	401.2
柏	41.3	62.1	428.0	0.0	228.1	24.1	908.6	456.5	807.3	286.6	45.4	707.4	127.7	240.7	31.7
磐田	188.0	287.9	216.2	228.1	0.0	212.5	694.7	247.8	972.2	482.0	183.1	503.2	119.2	367.2	207.6
浦和	35.8	83.9	407.8	24.1	212.5	0.0	887.7	435.9	805.0	288.1	31.6	685.7	106.5	228.0	7.6
鳥栖	873.1	970.7	481.4	908.6	694.7	887.7	0.0	452.1	1432.8	1084.1	864.3	210.1	781.3	920.9	880.9
神戸	421.1	518.6	32.3	456.5	247.8	435.9	452.1	0.0	1075.7	653.7	412.1	255.5	329.4	500.8	429.2
札幌	840.8	784.7	1065.3	807.3	972.2	805.0	1432.8	1075.7	0.0	524.7	834.2	1233.0	858.0	605.0	804.9
仙台	323.3	259.9	633.0	286.6	482.0	288.1	1084.1	653.7	524.7	0.0	319.1	874.1	362.8	166.9	289.5
東京	14.3	106.9	383.3	45.4	183.1	31.6	864.3	412.1	834.2	319.1	0.0	664.0	84.9	250.4	29.6
広島	673.8	769.3	287.0	707.4	503.2	685.7	210.1	255.5	1233.0	874.1	664.0	0.0	579.7	710.8	678.7
甲府	96.3	189.6	301.5	127.7	119.2	106.5	781.3	329.4	858.0	362.8	84.9	579.7	0.0	255.1	99.8
新潟	261.1	253.1	483.3	240.7	367.2	228.0	920.9	500.8	605.0	166.9	250.4	710.8	255.1	0.0	224.8
大宮	36.8	91.1	401.2	31.7	207.6	7.6	880.9	429.2	804.9	289.5	29.6	678.7	99.8	224.8	0.0

TSPの部分巡回路除去(4)をCPLEXで解く

- 結果([解]タブ)※一部省略

最適値 = 3360.1

```
// solution (optimal) with objective 3360.1
// Quality Incumbent solution:
// MILP objective                                 3.3601000000e+03
// MILP solution norm |x| (Total, Max)            1.20000e+02  1.00000e+00
// MILP solution error (Ax=b) (Total, Max)        0.00000e+00  0.00000e+00
// MILP x bound error (Total, Max)                0.00000e+00  0.00000e+00
// MILP x integrality error (Total, Max)          0.00000e+00  0.00000e+00
// MILP slack bound error (Total, Max)            0.00000e+00  0.00000e+00
//

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

最適解

得られた距離最小の巡回路

①川崎→⑪東京→⑬甲府→⑤磐田→③大坂
→⑧神戸→⑦鳥栖→⑫広島→⑭新潟→⑨札幌
→⑩仙台→②鹿島→④柏→⑥浦和→⑮大宮→①

1 川崎
2 鹿島
3 セ大
4 柏
5 磐田
6 浦和
7 鳥栖
8 神戸
9 札幌
10 仙台
11 東京
12 広島
13 甲府
14 新潟
15 大宮

TSPの部分巡回路除去(2)をgurobiで解く

- 問題(ex3)を python & gurobi で記述(データ生成部分①)

①

```
# coding: Shift_JIS
from gurobipy import *

# ######## 例題設定 #######
def make_data_ex3():
    I = [1,2,3,4,5,6,7,8]
    d = {(1,1):0,(1,2):1.5,(1,3):0.6,(1,4):0.8,(1,5):0.8,(1,6):0.8,(1,7):0.9,(1,8):0.8,
         (2,1):1.5,(2,2):0,(2,3):2,(2,4):2.3,(2,5):1.9,(2,6):1,(2,7):2.1,(2,8):1.4,
         (3,1):0.6,(3,2):2,(3,3):0,(3,4):0.7,(3,5):0.5,(3,6):1.2,(3,7):1,(3,8):1.3,
         (4,1):0.8,(4,2):2.3,(4,3):0.7,(4,4):0,(4,5):1.3,(4,6):1.7,(4,7):0.4,(4,8):1.1,
         (5,1):0.8,(5,2):1.9,(5,3):0.5,(5,4):1.3,(5,5):0,(5,6):0.9,(5,7):1.5,(5,8):1.5,
         (6,1):0.8,(6,2):1,(6,3):1.2,(6,4):1.7,(6,5):0.9,(6,6):0,(6,7):1.7,(6,8):1.2,
         (7,1):0.9,(7,2):2.1,(7,3):1,(7,4):0.4,(7,5):1.5,(7,6):1.7,(7,7):0,(7,8):0.8,
         (8,1):0.8,(8,2):1.4,(8,3):1.3,(8,4):1.1,(8,5):1.5,(8,6):1.2,(8,7):0.8,(8,8):0,
    } # 距離
    return I,d
```

※3種の定式化(TSP(2),(3),(4))で共通で使う

	1	2	3	4	5	6	7	8
	文教大	妙伝寺	宝蔵寺	来迎寺	白峰寺	正覚院	蓮妙寺	善谷寺
1	0.0	1.5	0.6	0.8	0.8	0.8	0.9	0.8
2	1.5	0.0	2.0	2.3	1.9	1.0	2.1	1.4
3	0.6	2.0	0.0	0.7	0.5	1.2	1.0	1.3
4	0.8	2.3	0.7	0.0	1.3	1.7	0.4	1.1
5	0.8	1.9	0.5	1.3	0.0	0.9	1.5	1.5
6	0.8	1.0	1.2	1.7	0.9	0.0	1.7	1.2
7	0.9	2.1	1.0	0.4	1.5	1.7	0.0	0.8
8	0.8	1.4	1.3	1.1	1.5	1.2	0.8	0.0

TSPの部分巡回路除去(2)をgurobiで解く

- TSP(2)

  potential定式化(ex3)

1つのファイル「TSPpot.py」に
①②③の順に記述して保存

②

```
# ####### 定式化 #######
def TSPpot(I,d):
    mod = Model("TSP:potential")

    # 変数設定
    x,u = {},{}
    for i in I:
        u[i] = mod.addVar(vtype="C", lb=0, ub=len(I)-1, name="u(%s)" % i)
            for j in I:
                if j != i:
                    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 I:
        mod.addConstr(quicksum(x[i,j] for j in I if j!=i) == 1)
        mod.addConstr(quicksum(x[j,i] for j in I if j!=i) == 1)
    for i in I:
        for j in range(1,len(I)):
            if j != i:
                mod.addConstr(u[i]+1-(len(I)-1)*(1-x[i,j]) <= u[j])
    mod.addConstr(u[0] == 0)

    # 目的関数の設定
    mod.setObjective(d[i,j]*x[i,j] for (i,j) in x), GRB.MINIMIZE)
    mod.update()
    mod.__data = x,u
    return mod
```

③

```
# ####### 実行 #######
if __name__=="__main__":
    I,d = make_data_ex3()          # デー
    mod = TSPpot(I,d)              # モデ
    mod.write("TSPpotex3.lp")      # lpフ
    mod.optimize()                 # 最適
    print("¥n optimal value = ", mod.ObjVal
    mod.printAttr('X')             # 最適
    mod.write("TSPpotex3.sol")     # 最適
```

TSPの部分巡回路除去(2)をgurobiで解く

- 実行結果

  TSP(2)

  potential定式化

```
optimal value =  6.4

   Variable            X
-----------------------
      x(0,4)           1
        u(1)           6
      x(1,5)           1
        u(2)           2
      x(2,3)           1
        u(3)           3
      x(3,6)           1
        u(4)           1
      x(4,2)           1
        u(5)           7
      x(5,0)           1
        u(6)           4
      x(6,7)           1
        u(7)           5
      x(7,1)           1
gurobi>
```

```
gurobi> exec(open("TSPpot.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, 64 columns and 260 nonzeros
Model fingerprint: 0x479e2480
Variable types: 8 continuous, 56 integer (56 binary)
Coefficient statistics:
  Matrix range     [1e+00, 7e+00]
  Objective range  [4e-01, 2e+00]
  Bounds range     [1e+00, 7e+00]
  RHS range        [1e+00, 6e+00]
Presolve removed 8 rows and 1 columns
Presolve time: 0.00s
Presolved: 58 rows, 63 columns, 238 nonzeros
Variable types: 7 continuous, 56 integer (56 binary)
Found heuristic solution: objective 6.6000000

Root relaxation: objective 5.685714e+00, 27 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    5.68571    0    9    6.60000    5.68571  13.9%     -    0s
H    0     0                       6.4000000    5.68571  11.2%     -    0s
     0     0     cutoff    0         6.40000    6.40000  0.00%     -    0s

Cutting planes:
  Learned: 3
  Gomory: 1
  Implied bound: 4
  Relax-and-lift: 2
  PSD: 1

Explored 1 nodes (53 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 20 (of 20 available processors)

Solution count 2: 6.4 6.6

Optimal solution found (tolerance 1.00e-04)
Best objective 6.400000000000e+00, best bound 6.400000000000e+00, gap 0.0000%
```

TSPの部分巡回

- TSP(3)
  - 単一品種流
  - 定式化（ex3）

1つのファイル
「TSPflo.py」に
①②③の順に記述して
保存

②

```
# ####### 定式化 #######
def TSPflo(I,d):
    mod = Model("TSP:flow")

    L = I[1:]
    # 変数設定
    x,flow = {},{}
    for i in I:
        for j in I:
            if j != i:
                x[i,j] = mod.addVar(vtype="B", name="x(%s,%s)" % (i,j))
                flow[i,j] = mod.addVar(vtype="C", lb=0, ub=len(L), name="flow(%s,%s)" % (i,j))
    mod.update()

    # 制約条件の設定
    for i in I:
        mod.addConstr(quicksum(x[i,j] for j in I if j!=i) == 1)
        mod.addConstr(quicksum(x[j,i] for j in I if j!=i) == 1)
    for i in L:
        mod.addConstr(quicksum(flow[j,i] for j in I if i!=j) - quicksum(flow[i,j] for j in I if i!=j) == 1)
    for j in L:
        mod.addConstr(flow[0,j] <= (len(I)-1)*x[0,j])
    for i in L:
        for j in L:
            if j != i:
                mod.addConstr(flow[i,j] <= (len(I)-2)*x[i,j])

    # 目的関数の設定
    mod.setObjective(quicksum(d[i,j]*x[i,j] for (i,j) in x), GRB.MINIMIZE)
    mod.update()
    mod.__data = x,flow
    return mod
```

③

```
# ####### 実行 #######
if __name__=="__main__":
    I,d = make_data_ex3()
    mod = TSPflo(I,d)
    mod.write("TSPfloex3.lp")
    mod.optimize()
    print("¥n optimal value = ", mod.
    mod.printAttr('X')
    mod.write("TSPfloex3.sol")
```

TSPの部分巡回路除去(2)をgurobiで解く

- 実行結果

  TSP(3)

  単一品種流

  定式化

```
optimal value =  6.4

   Variable            X
-----------------------
      x(0,3)           1
   flow(0,3)           7
      x(1,5)           1
   flow(1,5)           3
      x(2,0)           1
      x(3,6)           1
   flow(3,6)           6
      x(4,2)           1
   flow(4,2)           1
      x(5,4)           1
   flow(5,4)           2
      x(6,7)           1
   flow(6,7)           5
      x(7,1)           1
   flow(7,1)           4
gurobi>
```

```
gurobi> exec(open("TSPflo.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 73 rows, 112 columns and 315 nonzeros
Model fingerprint: 0x8bfeea14
Variable types: 56 continuous, 56 integer (56 binary)
Coefficient statistics:
  Matrix range     [1e+00, 7e+00]
  Objective range  [4e-01, 2e+00]
  Bounds range     [1e+00, 7e+00]
  RHS range        [1e+00, 7e+00]
Found heuristic solution: objective 10.1000000
Presolve removed 8 rows and 7 columns
Presolve time: 0.00s
Presolved: 65 rows, 105 columns, 294 nonzeros
Variable types: 49 continuous, 56 integer (56 binary)

Root relaxation: objective 5.759184e+00, 54 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    5.75918    0   12   10.10000    5.75918  43.0%     -    0s
H    0     0                       7.4000000    5.75918  22.2%     -    0s
H    0     0                       6.4000000    6.40000  0.00%     -    0s
     0     0          -    0         6.40000    6.40000  0.00%     -    0s

Cutting planes:
  Gomory: 1
  Implied bound: 5
  MIR: 4
  Flow cover: 2
  Network: 3
  RLT: 1

Explored 1 nodes (103 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 20 (of 20 available processors)

Solution count 3: 6.4 7.4 10.1

Optimal solution found (tolerance 1.00e-04)
Best objective 6.400000000000e+00, best bound 6.400000000000e+00, gap 0.0000%
```

TSPの部分

- TSP(4)
  多品種流
  定式化（ex3）

1つのファイル
「TSPflo.py」に
①②③の順に
記述して保存

②

```
# ####### 定式化 #######
def TSPmfl(I,d):
        mod = Model(“TSP:multi-flow")
        K = I[1:]
        # 変数設定
        x, mflow = {},{}
        for i in I:
            for j in I:
                    if j != i:
                        x[i,j] = mod.addVar(vtype="B", name="x(%s,%s)" % (i,j))
                            for k in K:
                                        mflow[k,i,j] = mod.addVar(vtype="C", lb=0, ub=1, name="mflow(%s,%s,%s)" % (k,i,j))
        mod.update()
        # 制約条件の設定
        for i in I:
            mod.addConstr(quicksum(x[i,j] for j in I if j!=i) == 1)
            mod.addConstr(quicksum(x[j,i] for j in I if j!=i) == 1)
        for k in K:
            mod.addConstr(quicksum(mflow[k,j,0] for j in K) - quicksum(mflow[k,0,j] for j in K) == -1)
            for i in K:
                    if i != k:
                        mod.addConstr(quicksum(mflow[k,j,i] for j in I if j!=i) - quicksum(mflow[k,i,j] for j in I if j!=i) == 0)
                        mod.addConstr(quicksum(mflow[k,j,k] for j in I if j!=k) - quicksum(mflow[k,k,j] for j in I if j!=k) == 1)
        for k in K:
            for i in I:
                    for j in I:
                        if j != i:
                            mod.addConstr(mflow[k,i,j] <= x[i,j])
        # 目的関数の設定
        mod.setObjective(quicksum(d[i,j]*x[i,j] for (i,j) in x), GRB.MINIMIZE)
        mod.update()
        mod.__data = x,mflow
        return mod
```

③

```
# ####### 実行 #######
if __name__=="__main__":
        I,d = make_data_ex3()
        mod = TSPmfl(I,d)
        mod.write(“TSPmflex3.lp")          # lpファイルを出力
        mod.optimize()                     # 最適化実行
        print("¥n optimal value = ", mod.ObjVal)       # 最適値の表示
        mod.printAttr('X')                 # 最適解の表示
        mod.write(“TSPmflex3.sol")         # 最適解をsolファイルに出力
```

TSPの部分巡回路除去(4)をgurobiで解く

- 実行結果

TSP(4) 多品種流定式化

```
gurobi> exec(open("TSPmfl.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 464 rows, 448 columns and 1680 nonzeros
Model fingerprint: 0x9d1b9dfb
Variable types: 392 continuous, 56 integer (56 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [4e-01, 2e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 1e+00]
Found heuristic solution: objective 9.5000000
Presolve removed 196 rows and 182 columns
Presolve time: 0.00s
Presolved: 268 rows, 266 columns, 1036 nonzeros
Variable types: 210 continuous, 56 integer (56 binary)

Root relaxation: objective 6.400000e+00, 115 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

H    0     0                       6.4000000    0.00000   100%     -    0s
     0     0          -    0         6.40000    6.40000  0.00%     -    0s

Explored 1 nodes (146 simplex iterations) in 0.01 seconds (0.01 work units)
Thread count was 20 (of 20 available processors)

Solution count 2: 6.4 9.5

Optimal solution found (tolerance 1.00e-04)
Best objective 6.400000000000e+00, best bound 6.400000000000e+00, gap 0.0000%
```

```
optimal value =  6.4

    Variable            X
-----------------------
        x(0,3)          1
   mflow(1,0,3)         1
   mflow(2,0,3)         1
   mflow(3,0,3)         1
   mflow(4,0,3)         1
   mflow(5,0,3)         1
   mflow(6,0,3)         1
   mflow(7,0,3)         1
        x(1,5)          1
   mflow(2,1,5)         1
   mflow(4,1,5)         1
   mflow(5,1,5)         1
        x(2,0)          1
        x(3,6)          1
   mflow(1,3,6)         1
   mflow(2,3,6)         1
   mflow(4,3,6)         1
   mflow(5,3,6)         1
   mflow(6,3,6)         1
   mflow(7,3,6)         1
        x(4,2)          1
   mflow(2,4,2)         1
        x(5,4)          1
   mflow(2,5,4)         1
   mflow(4,5,4)         1
        x(6,7)          1
   mflow(1,6,7)         1
   mflow(2,6,7)         1
   mflow(4,6,7)         1
   mflow(5,6,7)         1
   mflow(7,6,7)         1
        x(7,1)          1
   mflow(1,7,1)         1
   mflow(2,7,1)         1
   mflow(4,7,1)         1
   mflow(5,7,1)         1
gurobi>
```