

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

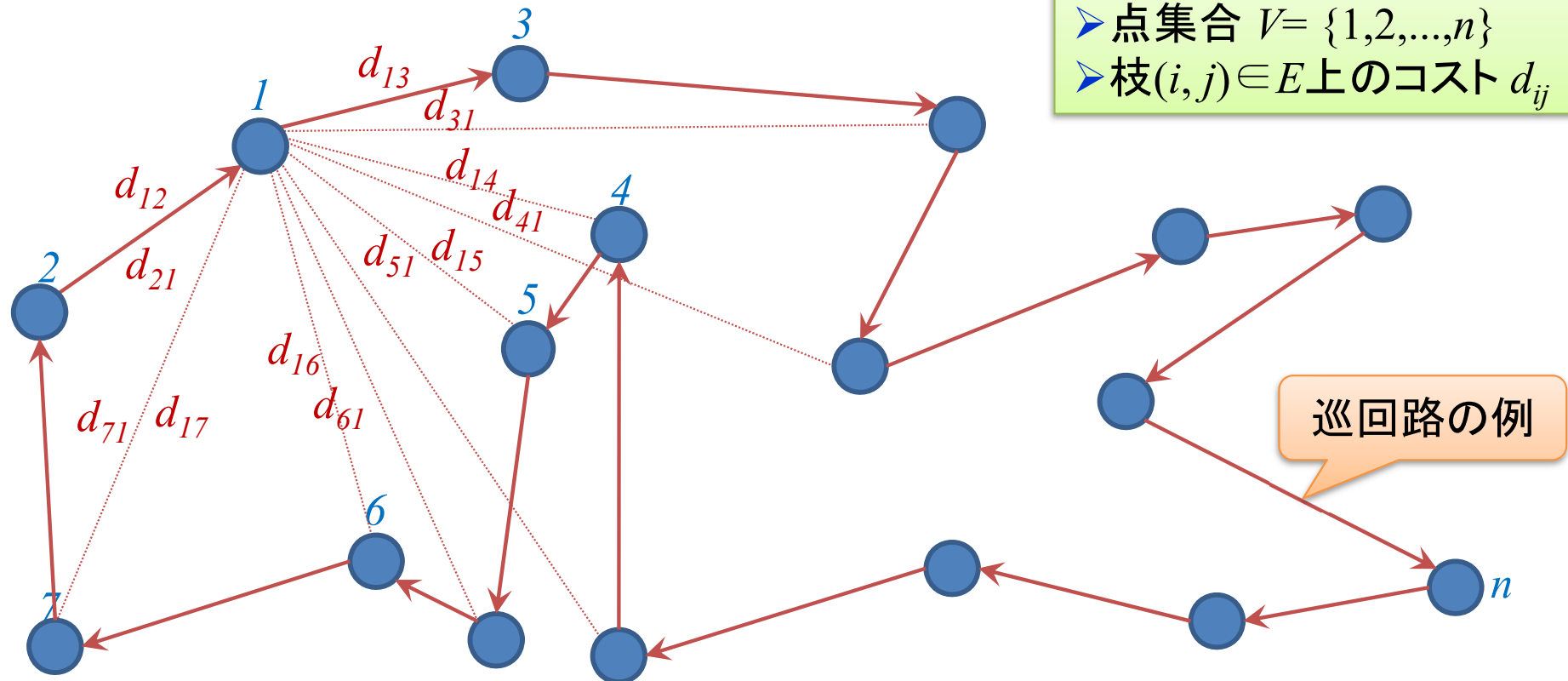
堀田 敬介

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 $V = \{1, 2, \dots, n\}$
- 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

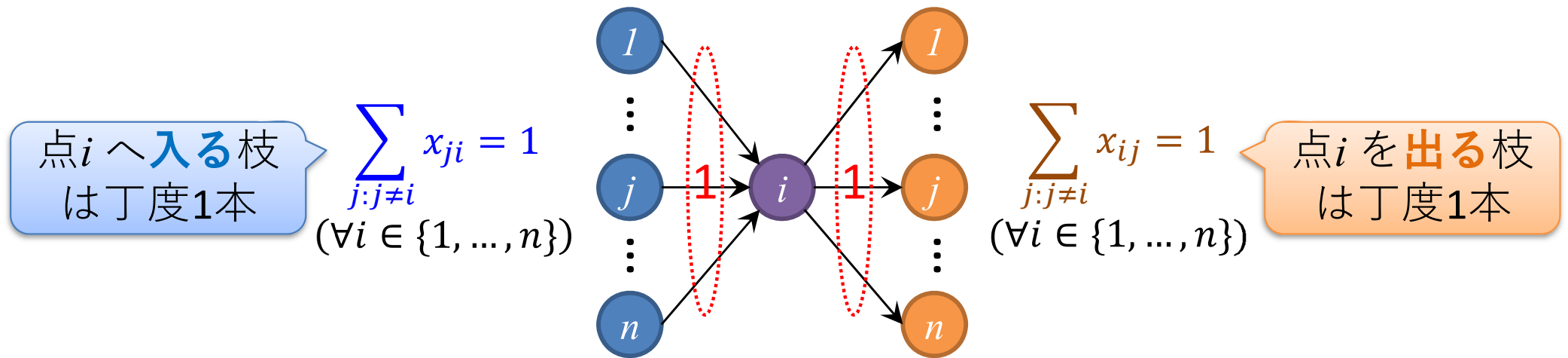
※対称の場合, 無向グラフで考えても良い

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 $x_{ij} = \begin{cases} 1 & \dots \text{枝}(i, j) \text{を通る} \\ 0 & \dots \text{枝}(i, j) \text{を通らない} \end{cases}$

➤ 定式化の準備: 巡回路となるための必要条件



各点で見れば, 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

➤ TSPの定式化(共通部分)

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

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

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

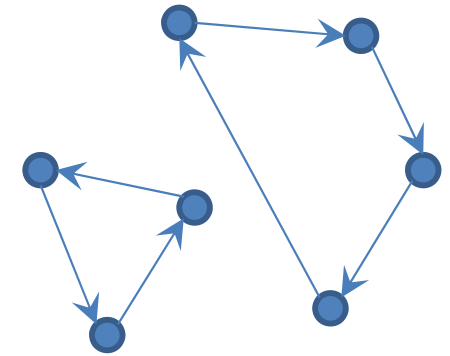
しかし、この2種類の制約だけでは**部分巡回路**が含まれる

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

部分巡回路の例



巡回セールスマン問題をCPLEXで解く

➤ TSPの最適化(例1)

- 完全グラフ $G = (V, E)$
- 点集合 $V = \{1, 2, \dots, n\}$ (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	2	3	4	5	6	7	8
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

➤ TSPの定式化1(変数設定・係数表記)

- 距離行列 distance matrix $\mathbf{D}=[d_{ij}]$

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)]－[新規(N)]－[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)

巡回セールスマン問題をCPLEXで解く

➤ cplex: model file [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

➤ Excel file [TSPex.xlsx]

	A	B	C	D	E	F	G	H	I	J
1	ex1)		コスト(距離)dij(km)							
2			1	2	3	4	5	6	7	8
3		1	0.0	2.0	1.0	1.9	1.5	1.4	1.8	1.7
4		2	2.0	0.0	2.3	1.3	1.2	1.7	1.8	1.9
5		3	1.0	2.3	0.0	1.3	1.2	1.7	1.8	1.9
6		4	1.9	1.3	1.3	0.0	1.9	2.7	0.8	1.8
7		5	1.5	1.2	1.2	1.9	0.0	1.4	2.4	2.3
8		6	1.4	1.7	1.7	2.7	1.4	0.0	2.4	2.0
9		7	1.8	1.8	1.8	0.8	2.4	2.4	0.0	1.2
10		8	1.7	1.9	1.9	1.8	2.3	2.0	1.2	0.0
11										
12		x	1	2	3	4	5	6	7	8
13		1								
14		2								
15		3								
16		4								
17		5								
18		6								
19		7								
20		8								
21										

Sheet1 Sheet2 J1 2017 Sheet3

➤ Cplex: data file [TSPex1.dat]

```
SheetConnection sheet("TSPex.xlsx");
```

```
i_max from SheetRead(sheet, "Sheet1!B10");  
d from SheetRead(sheet, "Sheet1!C3:J10");
```

```
x to SheetWrite(sheet, "Sheet1!C13:J20");
```

➤ Excelファイル [TSPex.xlsx] を作成し,
cplex の [TSP プロジェクト] に追加する

➤ cplex のデータファイル [TSPex1.dat] を上
記の通りに作成する

➤ [config1] にモデルファイル[TSP.mod] と
データファイル[TSPex1.dat] をセットし, 解く

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  
// MILP solution norm |x| (Total, Max)  
// MILP solution error (Ax=b) (Total, Max)  
// MILP x bound error (Total, Max)  
// MILP x integrality error (Total, Max)  
// MILP slack bound error (Total, Max)  
//
```

最適値 = 9.8

9.8000000000e+00

8.00000e+00 1.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

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 0 0 0 1]  
      [0 0 0 0 0 0 1 0]];
```

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

	A	B	C	D	E	F	G	H	I	J	K	L
1	ex1)	コスト(距離)dij(km)										
12		x	1	2	3	4	5	6	7	8		obj.fn.
13		1	0	0	1	0	0	0	0	0		9.8
14		2	0	0	0	1	0	0	0	0		
15		3	1	0	0	0	0	0	0	0		
16		4	0	1	0	0	0	0	0	0		
17		5	0	0	0	0	0	1	0	0		
18		6	0	0	0	0	1	0	0	0		
19		7	0	0	0	0	0	0	0	1		
20		8	0	0	0	0	0	0	1	0		
21												
Sheet1 Sheet2 J1 2017 Sheet3 ex4 J1 2017												

➤ 結果(Excelファイル

[TSPex.xlsx]) →

TSPの部分巡回路除去(1) 順次除去制約追加

コスト(距離) d_{ij} (km)

➤ TSPの最適化(例1)

- 完全グラフ $G = (V, E)$
- 点集合 $V = \{1, 2, \dots, n\}$ (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

	1	2	3	4	5	6	7	8
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

➤ TSP(1) 順次除去制約追加の定式化1 (ベタ表記 $\leftrightarrow$ Σ 表記)

$$\min. d_{12}x_{12} + d_{13}x_{13} + \dots + d_{78}x_{78}$$

$$\text{s. t. } x_{12} + x_{13} + \dots + x_{18} = 1$$

$$x_{21} + x_{23} + \dots + x_{28} = 1$$

...

$$x_{81} + x_{82} + \dots + x_{87} = 1$$

$$x_{21} + x_{31} + \dots + x_{81} = 1$$

$$x_{12} + x_{32} + \dots + x_{82} = 1$$

...

$$x_{18} + x_{28} + \dots + x_{78} = 1$$

$$x_{12} + x_{21} \leq 2 - 1, \dots$$

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

$$\longleftrightarrow \min. \sum_{i \neq j} d_{ij} x_{ij}$$

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

点 i を**出る**枝は丁度1本

点 i へ**入る**枝は丁度1本

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

部分巡回路除去制約

$$\longleftrightarrow \sum_{(i,j) \in S} x_{ij} \leq |S| - 1 \quad (\forall S \subset E, |S| \geq 2)$$

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

TSP(1) 順次除去制約追加をCPLEXで解く

➤ 結果([解]タブ)

```
// solution (optimal) with objective 9.8
// Quality Incumbent solution:
// MILP objective
// MILP solution norm |x| (Total, Max)
// MILP solution error (Ax=b) (Total, Max)
// MILP x bound error (Total, Max)
// MILP x integrality error (Total, Max)
// MILP slack bound error (Total, Max)
//
```

最適値 = 9.8

9.8000000000e+00

8.00000e+00 1.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

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

$x[1,3] + x[3,1] \leq 1;$

②→④→②

$x[2,4] + x[4,2] \leq 1;$

⑤→⑥→⑤

$x[5,6] + x[6,5] \leq 1;$

⑦→⑧→⑦

$x[7,8] + x[8,7] \leq 1;$

モデルファイル[TSP.mod]
に部分巡回路除去制約を
4つ追記して解きなす

```
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(1) 順次除去制約追加をCPLEXで解く

➤ 結果([解]タブ) 2回目

```
// solution (optimal) with objective 10.1
// Quality Incumbent solution:
// MILP objective
// MILP solution norm |x| (Total, Max)
// MILP solution error (Ax=b) (Total, Max)
// MILP x bound error (Total, Max)
// MILP x integrality error (Total, Max)
// MILP slack bound error (Total, Max)
//
```

最適値 = 10.1

1.0100000000e+01

8.00000e+00 1.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

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

最適解

	A	B	C	D	E	F	G	H	I	J	K	L
1	ex1)		コスト(距離)dij(km)									
12		x	1	2	3	4	5	6	7	8		obj.fn.
13		1	0	0	1	0	0	0	0	0		10.1
14		2	0	0	0	1	0	0	0	0		
15		3	0	0	0	0	1	0	0	0		
16		4	0	0	0	0	0	0	1	0		
17		5	0	1	0	0	0	0	0	0		
18		6	1	0	0	0	0	0	0	0		
19		7	0	0	0	0	0	0	0	1		
20		8	0	0	0	0	0	1	0	0		
21												
			Sheet1	Sheet2	J1 2017	ex3茅ヶ崎七福神	ex4 J					

➤ 結果(Excelファイル

[TSPex.xlsx]) →

➤ Excelファイル[TSPex.xlsx] の Sheet2

- TSPex2.dat

```
x to SheetWrite(sheet, "Sheet2!J17:S26");
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	ex2)	J1 [2017年より10チーム]							本拠地	1	2	3	4	5	6	7	8	9	10
2										川崎	鹿島	セ大	柏	磐田	浦和	鳥栖	神戸	札幌	仙台
3									1	2.44	2.45	2.37	2.44	2.41	2.44	2.28	2.36	2.47	2.46
4	no	名称	本住	lat					2	0.62	0.63	0.60	0.63	0.61	0.63	0.58	0.60	0.75	0.67
5	1	川崎フロンターレ	等禰						[	0.0	100.0	391.9	41.3	188.0	35.8	873.1	421.1	840.8	323.3
6	2	鹿島アントラーズ	県彦						[	100.0	0.0	490.0	62.1	287.9	83.9	970.7	518.6	784.7	259.9
7	3	セレッソ大阪	ヤナ						[	391.9	490.0	0.0	428.0	216.2	407.8	481.4	32.3	1065.3	633.0
8	4	柏レイソル	日千						[	41.3	62.1	428.0	0.0	228.1	24.1	908.6	456.5	807.3	286.6
9	5	ジュビロ磐田	ヤ薮						[	188.0	287.9	216.2	228.1	0.0	212.5	694.7	247.8	972.2	482.0
10	6	浦和レッズ	堀堀						[	35.8	83.9	407.8	24.1	212.5	0.0	887.7	435.9	805.0	288.1
11	7	サガン鳥栖	ヘ佐						[	873.1	970.7	481.4	908.6	694.7	887.7	0.0	452.1	1432.8	1084.1
12	8	ヴィッセル神戸	ノ兵						[	421.1	518.6	32.3	456.5	247.8	435.9	452.1	0.0	1075.7	653.7
13	9	北海道コンサドーレ札幌	村北						[	840.8	784.7	1065.3	807.3	972.2	805.0	1432.8	1075.7	0.0	524.7
14	10	ベガルタ仙台	コ宮						[	323.3	259.9	633.0	286.6	482.0	288.1	1084.1	653.7	524.7	0.0
15																			
16								x		1	2	3	4	5	6	7	8	9	10
17								1											
								2											
								3											
								4											
								5											
								6											
								7											
								8											
								9											
								#											

(("TSPex.xlsx"));

sheet, "Sheet2!A14");

t, "Sheet2!J5:S14");

, "Sheet2!J17:S26");

TSP部分巡回路除去(2)ポテンシャル定式化

➤ TSP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 (福祿寿)	0.7531	1.3666	1.2537	1.0772	1.5393	1.2355	0.7947	0.0000

➤ TSP(2)ポテンシャル定式化(変数設定・係数表記)

➤ 距離行列 distance matrix $D=[d_{ij}]$

➤ 0-1変数 $x_{ij} = \begin{cases} 1 & \dots \text{枝}(i,j) \text{を通る} \\ 0 & \dots \text{枝}(i,j) \text{を通らない} \end{cases}$

➤ 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(2)ポテンシャル定式化

➤ TSPの最適化

➤ TSP(2)ポテンシャル定式化 (ベタ表記 $\leftrightarrow$ Σ 表記)

$$\min. d_{12}x_{12} + d_{13}x_{13} + \dots + d_{78}x_{78}$$

$$\text{s. t. } x_{12} + x_{13} + \dots + x_{18} = 1$$

...

$$x_{81} + x_{82} + \dots + x_{87} = 1$$

$$x_{21} + x_{31} + \dots + x_{81} = 1$$

...

$$x_{18} + x_{28} + \dots + x_{78} = 1$$

$$u_1 = 0$$

$$u_1 + 1 - (n-1)(1-x_{12}) \leq u_2$$

...

$$u_7 + 1 - (n-1)(1-x_{78}) \leq u_8$$

$$1 \leq u_2 \leq n-1$$

...

$$1 \leq u_2 \leq n-1$$

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

$$\longleftrightarrow \min. \sum_{i \neq j} d_{ij} x_{ij}$$

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

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

$$\longleftrightarrow u_1 = 0$$

$$\longleftrightarrow u_i + 1 - (n-1)(1-x_{ij}) \leq u_j$$

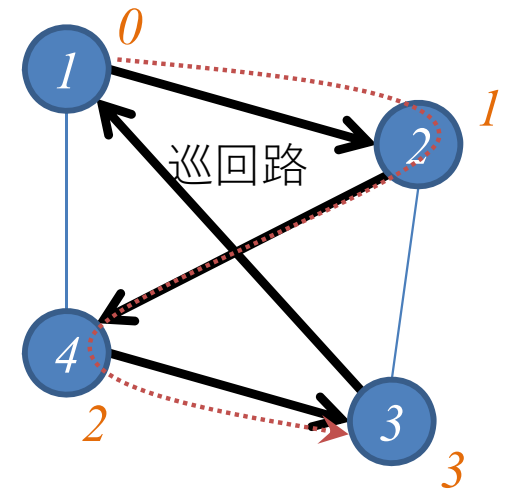
$$(\forall i \in V, \forall j \in V/\{1\}, i \neq j)$$

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

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

<4点の例>

巡回路に訪問順の
ポテンシャルがつく



点*i* を出る枝は丁度1本

点*i* へ入る枝は丁度1本

ポテンシャル制約

TSP(2)ポテンシャル定式化を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
- データは [TSPex.xlsx] の [Sheet3]
- [TSPex3.dat] 作成
- [config1] へ [TSPpot.mod][TSPex3.dat] をセット

データファイル [TSPex3.dat]

```
SheetConnection sheet("TSPex.xlsx");
```

```
i_max from SheetRead(sheet, "Sheet3!A12");  
d from SheetRead(sheet, "Sheet3!J5:Q12");
```

```
x to SheetWrite(sheet, "Sheet3!J15:Q22");
```

```
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
```

モデルファイル
[TSPpot.mod]

```
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;  
    }  
};
```

TSP(2)ポテンシャル定式化をCPLEXで解く

➤ 結果([解]タブ)

```
// solution (optimal) with objective 6.4566
// Quality Incumbent solution:
// MILP objective
// MILP solution norm |x| (Total, Max)
// MILP solution error (Ax=b) (Total, Max)
// MILP x bound error (Total, Max)
// MILP x integrality error (Total, Max)
// MILP slack bound error (Total, Max)
//
```

```
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

最適値 = 6.4566

6.4566000000e+00

3.60000e+01 7.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

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 (福祿寿)



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, $x[1][4] \rightarrow x[4][7] \rightarrow x[7][8] \rightarrow x[8][2] \rightarrow \dots$ 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

➤ Excelファイル[TSPex.xlsx] の Sheet2

```
SheetConnection sheet("TSPex.xlsx");

i_max from SheetRead(sheet, "Sheet2!A14");
d from SheetRead(sheet, "Sheet2!J5:S14");

x to SheetWrite(sheet, "Sheet2!J17:S26");
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	ex2)	J1 [2017年より10チーム]							本拠地	1	2	3	4	5	6	7	8	9	10
2										川崎	鹿島	セ大	柏	磐田	浦和	鳥栖	神戸	札幌	仙台
3									1	2.44	2.45	2.37	2.44	2.41	2.44	2.28	2.36	2.47	2.46
4	no	名称	本住	lat					2	0.62	0.63	0.60	0.63	0.61	0.63	0.58	0.60	0.75	0.67
5	1	川崎フロンターレ	等神						[	0.0	100.0	391.9	41.3	188.0	35.8	873.1	421.1	840.8	323.3
6	2	鹿島アントラーズ	県茨						[	100.0	0.0	490.0	62.1	287.9	83.9	970.7	518.6	784.7	259.9
7	3	セレッソ大阪	ヤ大						[	391.9	490.0	0.0	428.0	216.2	407.8	481.4	32.3	1065.3	633.0
8	4	柏レイソル	日千						[	41.3	62.1	428.0	0.0	228.1	24.1	908.6	456.5	807.3	286.6
9	5	ジュビロ磐田	ヤ静						[	188.0	287.9	216.2	228.1	0.0	212.5	694.7	247.8	972.2	482.0
10	6	浦和レッズ	埼埼						[	35.8	83.9	407.8	24.1	212.5	0.0	887.7	435.9	805.0	288.1
11	7	サガン鳥栖	ベ佐						[	873.1	970.7	481.4	908.6	694.7	887.7	0.0	452.1	1432.8	1084.1
12	8	ヴィッセル神戸	ノ兵						[	421.1	518.6	32.3	456.5	247.8	435.9	452.1	0.0	1075.7	653.7
13	9	北海道コンサドーレ札幌	札北						[	840.8	784.7	1065.3	807.3	972.2	805.0	1432.8	1075.7	0.0	524.7
14	10	ベガルタ仙台	コ宮						[	323.3	259.9	633.0	286.6	482.0	288.1	1084.1	653.7	524.7	0.0
15																			
16								x		1	2	3	4	5	6	7	8	9	10
17									1										
									2										
									3										
									4										
									5										
									6										
									7										
									8										
									9										
									#										

(("TSPex.xlsx"));

sheet, "Sheet2!A14");

t, "Sheet2!J5:S14");

, "Sheet2!J17:S26");

【演習】ポテンシャル定式化で例2を解く

➤ 結果([解]タブ)

```
// solution (optimal) with objective 3223
// Quality Incumbent solution:
// MILP objective
// MILP solution norm |x| (Total, Max)
// MILP solution error (Ax=b) (Total, Max)
// MILP x bound error (Total, Max)
// MILP x integrality error (Total, Max)
// MILP slack bound error (Total, Max)
```

```
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

最適値 = 3223

3.2230000000e+03

5.50000e+01 9.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

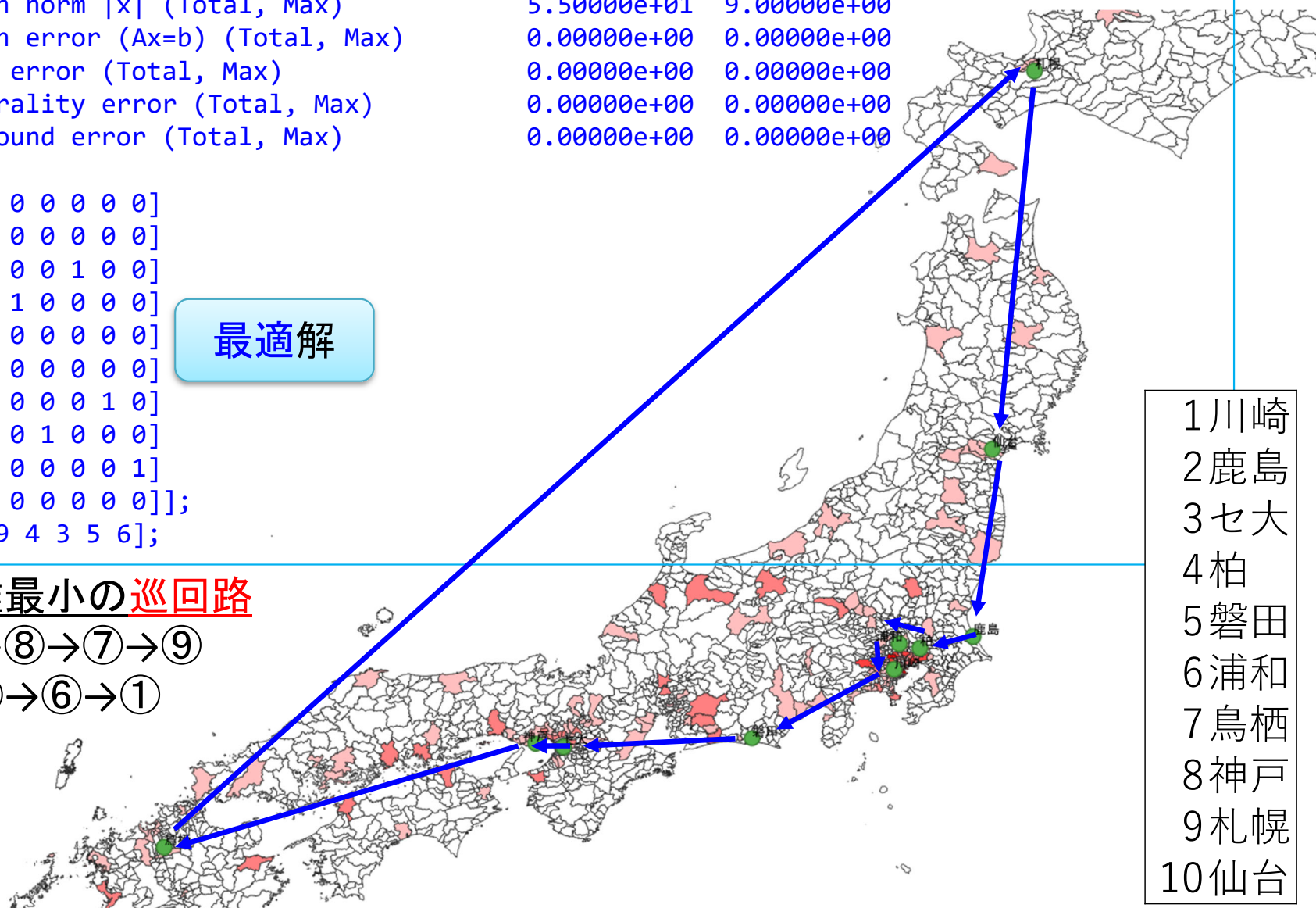
0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

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部分巡回路除去(3)単一品種流定式化

➤ TSP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 (福祿寿)	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 $D=[d_{ij}]$

➤ 0-1変数 $x_{ij} = \begin{cases} 1 & \dots \text{枝}(i,j) \text{を通る} \\ 0 & \dots \text{枝}(i,j) \text{を通らない} \end{cases}$

➤ 実数変数 $f_{ij} \in [0, n-1]$ ※点 $i \rightarrow j$ に対するフロー変数 flow variables

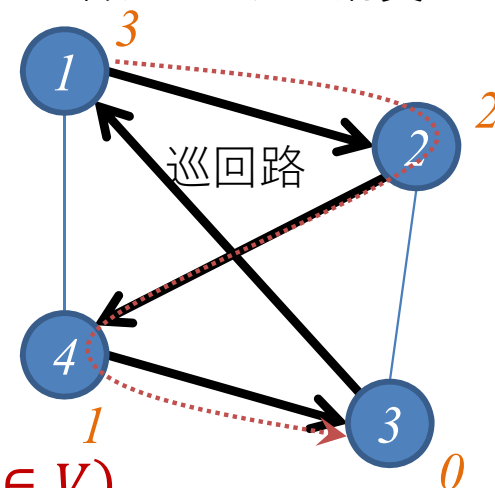
単一品種流定式化のためのフロー変数

TSP(3) 単一品種流定式化

➤ TSPの最適化

➤ TSP(3) 単一品種流定式化 (ベタ表記 $\leftrightarrow$ Σ 表記)

<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



$$\min. d_{12}x_{12} + d_{13}x_{13} + \dots + d_{78}x_{78}$$

$$\text{s. t. } x_{12} + x_{13} + \dots + x_{18} = 1$$

...

$$x_{81} + x_{82} + \dots + x_{87} = 1$$

$$x_{21} + x_{31} + \dots + x_{81} = 1$$

...

$$x_{18} + x_{28} + \dots + x_{78} = 1$$

$$f_{12} + f_{13} + \dots + f_{18} = 7$$

$$(f_{12} + f_{32} + \dots + f_{82}) - (f_{21} + f_{23} + \dots + f_{28}) = 1$$

...

$$(f_{18} + f_{28} + \dots + f_{78}) - (f_{81} + f_{82} + \dots + f_{87}) = 1$$

$$f_{12} \leq 7x_{12}, \dots, f_{18} \leq 7x_{18}$$

$$f_{23} \leq 6x_{23}, \dots, f_{28} \leq 6x_{28}$$

...

$$f_{82} \leq 6x_{82}, \dots, f_{87} \leq 6x_{87}$$

$$f_{12}, f_{13}, \dots, f_{87} \geq 0$$

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

$$\longleftrightarrow \min. \sum_{i \neq j} d_{ij} x_{ij}$$

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

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

$$\longleftrightarrow \sum_{j=2}^n f_{1j} = n - 1$$

$$\longleftrightarrow \sum_{j: j \neq i} f_{ji} - \sum_{j: j \neq i} f_{ij} = 1 \quad (\forall i = 2, \dots, n)$$

$$\longleftrightarrow f_{1j} \leq (n - 1)x_{1j} \quad (\forall j \neq 1)$$

$$\longleftrightarrow f_{ij} \leq (n - 2)x_{ij} \quad (\forall i \neq j, i \neq 1, j \neq 1)$$

$$\longleftrightarrow f_{ij} \geq 0 \quad (\forall i \neq j)$$

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

点*i*を**出る**枝は丁度1本

点*i*へ**入る**枝は丁度1本

単一フロー制約

TSP(3) 単一品種流定式化を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 [TSPflo.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
- データは [TSPex.xlsx] の [Sheet3]
- [TSPex3.dat] 利用 (既出)
- [config1] へ [TSPflo.mod][TSPex3.dat] をセット

データファイル [TSPex3.dat] (既出)

```
SheetConnection sheet("TSPex.xlsx");
```

```
i_max from SheetRead(sheet, "Sheet3!A12");  
d from SheetRead(sheet, "Sheet3!J5:Q12");
```

```
x to SheetWrite(sheet, "Sheet3!J15:Q22");
```

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];  
        }  
    }  
};
```

TSP(3) 単一品種流定式化をCPLEXで解く

➤ 結果([解]タブ)

```
// solution (optimal) with objective 6.4566
// Quality Incumbent solution:
// MILP objective
// MILP solution norm |x| (Total, Max)
// MILP solution error (Ax=b) (Total, Max)
// MILP x bound error (Total, Max)
// MILP x integrality error (Total, Max)
// MILP slack bound error (Total, Max)
```

```
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

最適値 = 6.4566

6.4566000000e+00

3.60000e+01 7.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

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 (福祿寿)



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

【演習】単一品種流定式化で例4を解く

➤ TSPの最適化(例4)

➤ Excelファイル[TSPex.xlsx] の Sheet4

➤ cplex: model file
[TSPflo.mod]

➤ cplex: data file
[TSPex4.dat]

➤ Excel file
[TSPex.xlsx]
の[Sheet4]

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
1	ex4) J1 [2017年より15チーム]								本拠地	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2										川崎	鹿島	セ大	柏	磐田	浦和	鳥栖	神戸	札幌	仙台	東京	広島	甲府	新潟	大宮
3										2.44	2.45	2.37	2.44	2.41	2.44	2.28	2.36	2.47	2.46	2.44	2.31	2.42	2.43	2.44
4	no	名称	本拠地	lat	2					0.62	0.63	0.60	0.63	0.61	0.63	0.58	0.60	0.75	0.67	0.62	0.60	0.62	0.66	0.63
5	1	川崎フロンターレ	埼玉							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
6	2	鹿島アントラーズ	千葉							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
7	3	セレッソ大阪	大阪							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
8	4	柏レイソル	千葉							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
9	5	ジュビロ磐田	静岡							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
10	6	浦和レッズ	埼玉							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
11	7	サガン鳥栖	福岡							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
12	8	ヴィッセル神戸	兵庫							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
13	9	北海道コンサドーレ札幌	北海道							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
14	10	ベガルタ仙台	宮城							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
15	11	FC東京	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
16	12	サンフレッチェ広島	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
17	13	ヴァンフォーレ甲府	山梨							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
18	14	アルビレックス新潟	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
19	15	大宮アルディージャ	埼玉							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
20																								
21									x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
22									1															
23									2															
35									14															
36									15															

➤ TSPex4.dat

```
SheetConnection sheet("TSPex.xlsx");
```

```
i_max from SheetRead(sheet, "Sheet4!A19");
```

```
d from SheetRead(sheet, "Sheet4!J5:X19");
```

```
x to SheetWrite(sheet, "Sheet4!J22:X36");
```

【演習】単一品種流定式化で例4を解く

➤ 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 0 1]
      [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 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 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 0 1 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 1 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 1 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 1 0 0]
      [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]];
flow = [[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 14]
        [0 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]
        [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 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 0 5 0 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 0 8 0 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 0 0 6 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 7 0 0 0 0]
        [0 0 0 0 0 13 0 0 0 0 0 0 0 0 0 0]];
```

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

最適解

1 川崎
2 鹿島
3 大坂
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) 多品種流定式化

➤ TSP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 $D=[d_{ij}]$

➤ 0-1変数 $x_{ij} = \begin{cases} 1 & \dots \text{枝}(i,j) \text{を通る} \\ 0 & \dots \text{枝}(i,j) \text{を通らない} \end{cases}$

➤ 実数変数 $f_{ij}^k \geq 0$ ※品種 k を点 $i \rightarrow j$ 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(4) 多品種流定式化

➤ TSPの最適化

➤ TSP(4) 多品種流定式化 (ベタ表記 $\leftrightarrow$ Σ 表記)

$$\min. d_{12}x_{12} + d_{13}x_{13} + \dots + d_{78}x_{78}$$

$$\text{s. t. } x_{12} + x_{13} + \dots + x_{18} = 1$$

...

$$x_{81} + x_{82} + \dots + x_{87} = 1$$

$$x_{21} + x_{31} + \dots + x_{81} = 1$$

...

$$x_{18} + x_{28} + \dots + x_{78} = 1$$

$$(f_{21}^2 + f_{31}^2 + \dots + f_{81}^2) - (f_{12}^2 + f_{13}^2 + \dots + f_{18}^2) = -1$$

...

$$(f_{13}^2 + f_{23}^2 + \dots + f_{83}^2) - (f_{31}^2 + f_{32}^2 + \dots + f_{38}^2) = 0$$

...

$$(f_{12}^2 + f_{32}^2 + \dots + f_{82}^2) - (f_{21}^2 + f_{23}^2 + \dots + f_{28}^2) = 1$$

...

$$f_{12}^1 \leq x_{12}, f_{13}^1 \leq x_{13}, \dots, f_{18}^1 \leq x_{18}$$

...

$$f_{81}^7 \leq x_{81}, f_{82}^7 \leq x_{82}, \dots, f_{87}^7 \leq x_{87}$$

$$f_{12}, f_{13}, \dots, f_{87} \geq 0$$

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

$$\longleftrightarrow \min. \sum_{i \neq j} d_{ij} x_{ij}$$

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

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

多品種フロー制約

$$\longleftrightarrow \sum_{j: j \neq i} f_{ji}^k - \sum_{j: j \neq i} f_{ij}^k = \begin{cases} -1 & (i = 1) \\ 0 & (i \neq 1, i \neq k) \\ 1 & (i = k) \end{cases} \quad (\forall k = 2, \dots, n)$$

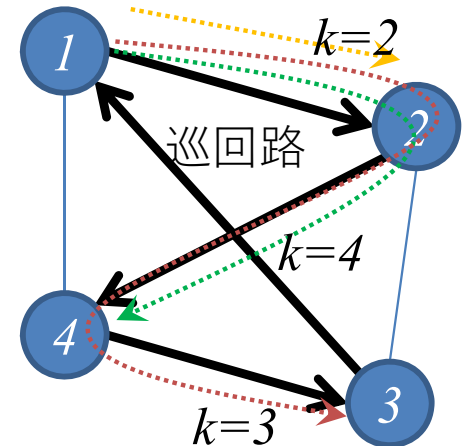
$$\longleftrightarrow f_{ij}^k \leq x_{ij} \quad (\forall k, i, j)$$

$$\longleftrightarrow f_{ij}^k \geq 0 \quad (\forall k, i, j: i \neq j, j \neq 1)$$

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

< 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



点*i*を**出る**枝は丁度1本

点*i*へ**入る**枝は丁度1本

TSP(4) 多品種流

モデルファイル
[TSPmfl.mod]

- 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 [TSPex.xlsx] の [Sheet3] を利用
- [TSPex3.dat]利用(既出)
- [config1]へ
[TSPmfl.mod][TSPex3.dat]セット

データファイル[TSPex3.dat](既出)

```
SheetConnection sheet("TSPex.xlsx");
```

```
i_max from SheetRead(sheet, "Sheet3!A12")  
d from SheetRead(sheet, "Sheet3!J5:Q12")
```

```
x to SheetWrite(sheet, "Sheet3!J15:Q22");
```

```
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(4) 多品種流定式化をCPLEXで解く

➤ 結果([解]タブ)

```
// solution (optimal) with objective 6.4566
// Quality Incumbent solution:
// MILP objective
// MILP solution norm |x| (Total, Max)
// MILP solution error (Ax=b) (Total, Max)
// MILP x bound error (Total, Max)
// MILP x integrality error (Total, Max)
// MILP slack bound error (Total, Max)
//
```

```
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]
           [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

最適値 = 6.4566

6.4566000000e+00

3.60000e+01 1.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

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 (福祿寿)



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

【演習】多品種流定式化で例4を解く

➤ TSPの最適化(例4)

➤ Excelファイル[TSPex.xlsx] の Sheet4

➤ cplex: model file
[TSPmfl.mod]

➤ cplex: data file
[TSPex4.dat]

➤ Excel file
[TSPex.xlsx]
の[Sheet4]

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
1	ex4) J1 [2017年より15チーム]								本拠地	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2										川崎	鹿島	セ大	柏	磐田	浦和	鳥栖	神戸	札幌	仙台	東京	広島	甲府	新潟	大宮
3										2.44	2.45	2.37	2.44	2.41	2.44	2.28	2.36	2.47	2.46	2.44	2.31	2.42	2.43	2.44
4	no	名称	本拠地	lat	2					0.62	0.63	0.60	0.63	0.61	0.63	0.58	0.60	0.75	0.67	0.62	0.60	0.62	0.66	0.63
5	1	川崎フロンターレ	埼玉							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
6	2	鹿島アントラーズ	千葉							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
7	3	セレッソ大阪	大阪							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
8	4	柏レイソル	千葉							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
9	5	ジュビロ磐田	静岡							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
10	6	浦和レッズ	埼玉							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
11	7	サガン鳥栖	福岡							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
12	8	ヴィッセル神戸	兵庫							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
13	9	北海道コンサドーレ札幌	北海道							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
14	10	ベガルタ仙台	宮城							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
15	11	FC東京	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
16	12	サンフレッチェ広島	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
17	13	ヴァンフォーレ甲府	山梨							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
18	14	アルビレックス新潟	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
19	15	大宮アルディージャ	埼玉							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
20																								
21									x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
22									1															
23									2															
35									14															
36									15															

➤ TSPex4.dat

```
SheetConnection sheet("TSPex.xlsx");
```

```
i_max from SheetRead(sheet, "Sheet4!A19");
```

```
d from SheetRead(sheet, "Sheet4!J5:X19");
```

```
x to SheetWrite(sheet, "Sheet4!J22:X36");
```

【演習】多品種流定式化で例4を解く

➤ 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
// MILP solution norm |x| (Total, Max)
// MILP solution error (Ax=b) (Total, Max)
// MILP x bound error (Total, Max)
// MILP x integrality error (Total, Max)
// MILP slack bound error (Total, Max)
//
```

3.3601000000e+03

1.20000e+02 1.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

0.00000e+00 0.00000e+00

```
x = [[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]
      [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 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 1]
      [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 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 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 0 0 1 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]
      [1 0 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 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 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 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 大坂
- 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

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

TSP部分巡回路除去(1)～(4)を GUROBIで解く

TSP部分巡回路除去(1)～(4)をgurobiで解く

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

```
# coding: Shift_JIS
```

```
from gurobipy import *
```

①

```
# ##### 例題設定 #####
```

```
def make_data_ex3():
```

```
    l = [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 l,d
```

※4種の定式化(TSP(1)～(4)) 共通で使う

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

TSP(2)ポテンシャル定式化を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

実行

③

```
if __name__ == "__main__":  
    l,d = make_data_ex3()          # データ生成  
    mod = TSPpot(l,d)              # モデル作成  
    mod.write("TSPpotex3.lp")      # lpファイルの書き出し  
    mod.optimize()                 # 最適化  
    print("¥n optimal value = ", mod.ObjVal)  # 最適値の出力  
    mod.printAttr('X')             # 最適解の出力  
    mod.write("TSPpotex3.sol")     # 最適解の書き出し
```

定式化

②

```
def TSPpot(l,d):  
    mod = Model("TSP:potential")  
  
    # 変数設定  
    x,u = {},{}  
    for i in l:  
        u[i] = mod.addVar(vtype="C", lb=0, ub=len(l)-1, name="u(%s)" % i)  
        for j in l:  
            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 l:  
    mod.addConstr(quicksum(x[i,j] for j in l if j!=i) == 1)  
    mod.addConstr(quicksum(x[j,i] for j in l if j!=i) == 1)  
for i in l:  
    for j in range(1,len(l)):  
        if j != i:  
            mod.addConstr(u[i]+1-(len(l)-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
```

TSP(2)ポテンシャル定式化を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             cutoff   0    6.40000    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(3) 単一品

➤ 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

定式化

```
def TSPflo(l,d):  
    mod = Model("TSP:flow")  
  
    L = l[1:]  
    # 変数設定  
    x,flow = {},{}  
    for i in l:  
        for j in l:  
            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 l:  
    mod.addConstr(quicksum(x[i,j] for j in l if j!=i) == 1)  
    mod.addConstr(quicksum(x[j,i] for j in l if j!=i) == 1)  
for i in L:  
    mod.addConstr(quicksum(flow[j,i] for j in l if i!=j) - quicksum(flow[i,j] for j in l if i!=j) == 1)  
for j in L:  
    mod.addConstr(flow[0,j] <= (len(l)-1)*x[0,j])  
for i in L:  
    for j in L:  
        if j != i:  
            mod.addConstr(flow[i,j] <= (len(l)-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
```

実行

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

③

②

TSP(3) 単一口種流定式化をgurobiで解く

実行結果

TSP(3)

単一口種流

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     7.4000000    5.75918  22.2%   -    0s
    H     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(4) 多品

➤ 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

実行

```
if __name__ == "__main__":  
    l,d = make_data_ex3()  
    mod = TSPmfl(l,d)  
    mod.write("TSPmflex3.lp")  
    mod.optimize()  
    print("¥n optimal value = ", mod.ObjVal)  
    mod.printAttr('X')  
    mod.write("TSPmflex3.sol")
```

定式化

```
def TSPmfl(l,d):  
    mod = Model("TSP:multi-flow")  
    K = l[1:]  
    # 変数設定  
    x, mflow = {},{}  
    for i in l:  
        for j in l:  
            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 l:  
        mod.addConstr(quicksum(x[i,j] for j in l if j!=i) == 1)  
        mod.addConstr(quicksum(x[j,i] for j in l 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 l if j!=i) - quicksum(mflow[k,i,j] for j in l if j!=i) == 0)  
                mod.addConstr(quicksum(mflow[k,j,k] for j in l if j!=k) - quicksum(mflow[k,k,j] for j in l if j!=k) == 1)  
    for k in K:  
        for i in l:  
            for j in l:  
                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
```

lpファイルを出力

最適化実行

最適値の表示

最適解の表示

最適解をsolファイルに出力

②

③

TSP(4) 多品種流定式化を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           -    0      6.400000    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>
```