問題解決

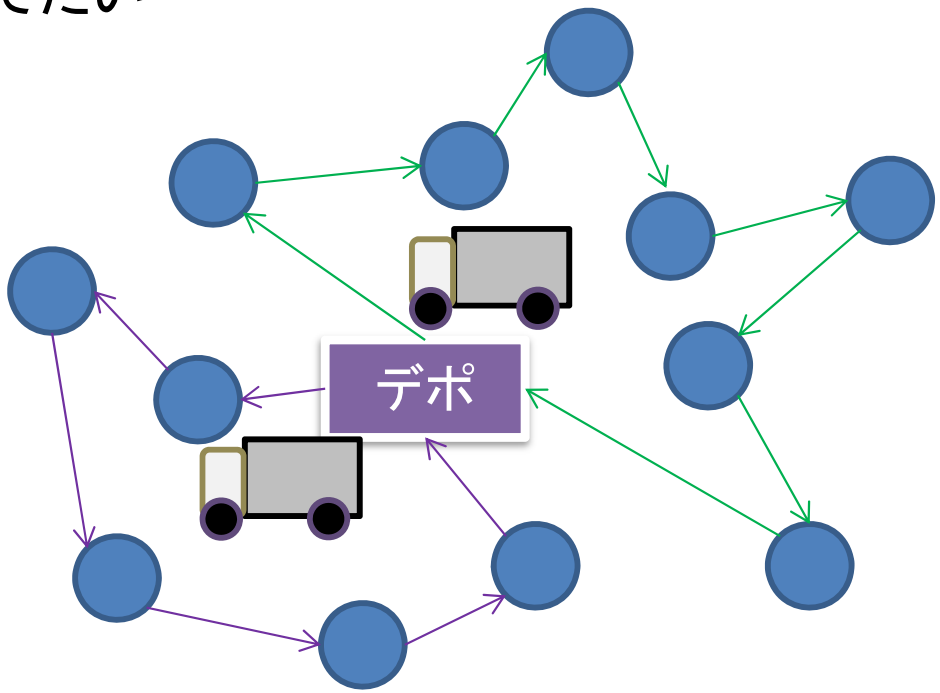
# 組合せ最適化と整数計画法
# 配送計画問題

堀田 敬介

# 配送計画問題の最適化

- 配送計画問題 vehicle routing problem; VRP
  - $n$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（ $i \neq j$ ）
  - デポ depotにある複数の運搬車が複数の顧客 customer を巡回しデポに戻る
  - コストの総和が最小の配送計画をたてたい

- 運搬車 vehicle … $m$台（$m \geqq 1$）
  （※$m$=1の場合はTSPとなる）




- 完全有向グラフ $G = (V, E)$
- 点集合 $V = V_1 \cup V_2 = \{1,2,...,n\}$
  - デポ集合 $V_1 = \{1\}$
  - 顧客集合 $V_2 = \{2,\ldots,n\}$
- 枝$(i, j) \in E$上のコスト $d_{ij}$

- $\forall i, j,\ d_{ij} = d_{ji}$ のとき対称VRP
- $\exists i, j,\ d_{ij} \neq d_{ji}$ のとき非対称VRP

# 配送計画問題の最適化

- 最適化問題の定式化（変数設定・係数表記）
  - 0-1変数 $x_{ij} = \begin{cases} 1 & \cdots \text{枝}(i,j)\text{を通る} \\ 0 & \cdots \text{枝}(i,j)\text{を通らない} \end{cases}$
- VRPの定式化（部分巡回路除去型）
  - $G = (V, E)$
  - $V = V_1 \cup V_2$
    - $V_1 = \{1\}$ … depot
    - $V_2 = \{2,3,\ldots,n\}$ … customer

min.

$$\sum_{(i,j)\in E} d_{ij} x_{ij}$$

s. t.

$$\sum_{j\in V_2} x_{1j} = m, \sum_{j\in V_2} x_{j1} = m$$

デポ（$V_1$={1}）の出る/入るのに使う枝はそれぞれ$m$本（運搬車 $m$ 台）

$$\sum_{j\in V} x_{ij} = 1, \sum_{j\in V} x_{ji} = 1 (\forall i \in V_2)$$

各顧客 $i \in V_2$={2,…,$n$} から出る/入るのに使う枝はそれぞれ各1本

$$\sum_{i,j\in S} x_{ij} \leq |S| - 1 \ (\forall S \in V_2, |S| \geq 2)$$

部分巡回路除去制約

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

# 配送計画問題をCPLEXで解く

- VRPの最適化（例1）
  - $G = (V, E)$
  - $V = V_1 \cup V_2$
    - $V_1 = \{1\}$ … depot
    - $V_2 = \{2,3,\ldots,10\}$ … customer
  - $m$=2 … number of vehicle

  - 距離行列 distance matrix $\boldsymbol{D}=[d_{ij}]$
  - 距離非対称（$\exists i, j,\ d_{ij} \neq d_{ji}$）

コスト（距離）$d_{ij}$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 7 | 2 | 3 | 8 | 8 | 3 | 8 | 4 | 1 |
| 2 | 9 | 0 | 2 | 8 | 8 | 6 | 4 | 8 | 5 | 3 |
| 3 | 6 | 1 | 0 | 9 | 1 | 6 | 2 | 7 | 3 | 7 |
| 4 | 3 | 6 | 5 | 0 | 5 | 7 | 1 | 4 | 8 | 7 |
| 5 | 2 | 9 | 6 | 7 | 0 | 8 | 1 | 4 | 5 | 8 |
| 6 | 6 | 7 | 1 | 6 | 6 | 0 | 1 | 2 | 7 | 7 |
| 7 | 5 | 9 | 7 | 9 | 6 | 8 | 0 | 9 | 9 | 3 |
| 8 | 3 | 5 | 4 | 3 | 6 | 7 | 8 | 0 | 7 | 9 |
| 9 | 1 | 2 | 1 | 8 | 8 | 1 | 5 | 8 | 0 | 6 |
| 10 | 4 | 9 | 8 | 6 | 6 | 5 | 8 | 8 | 7 | 0 |

# 配送計画問題を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：VRP）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 [VRP.mod] ※名称変更しない
  - ✓ データファイル [VRP.dat] → [VRPex1.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

# 配送計画問題をCPLEXで解く

- VRP.mod

```
int v_max = ...;// 頂点数|V| V=V1&V2, V1={1}, V2={2,3,...,n}
int m = ...;// 運搬車の台数

range V = 1..v_max;// V={1,2,...,n}
range V2 = 2..v_max;// V2={2,3,...,n}

float d[V,V] = ...;// 距離行列D=[dij]

dvar int+ x[V,V] in 0..1;// 0-1変数

minimize
  sum(i in V) sum(j in V:i!=j) d[i][j]*x[i][j];
subject to {
  sum(j in V2) x[1][j] == m;// デポから出るのに使う枝は m本
  sum(j in V2) x[j][1] == m;// デポに入るのに使う枝は m本
  forall (i in V2) {
    sum(j in V) x[i][j] == 1;// 顧客iから出るのに使う枝は 1本
    sum(j in V) x[j][i] == 1;// 顧客iに入るのに使う枝は 1本
  }
  forall (i in V) {
    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)
};
```

# 配送計画問題をCPLEXで解く

- VRPex1.dat

```
v_max = 10;// 頂点数|V| V=V1&V2, V1={1}, V2={2,3,...,n}
m = 2;// 運搬車の台数

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

# 配送計画問題をCPLEXで解く

- 結果([解]タブ)

```
// solution (optimal) with objective 27
// Quality Incumbent solution:
// MILP objective                                2.7000000000e+01
// MILP solution norm |x| (Total, Max)           1.10000e+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 0 0 0 0 0 0 1 1]
     [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 0 1 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 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]
     [1 0 0 0 0 0 0 0 0 0]];
```

最適値 = 27

最適解に部分巡回路がある

- ✓ 運搬車1:①→⑨→⑥→⑧→④→⑦→⑤→①
- ✓ 運搬車2:①→⑩→①
- ✓ 部分巡回路:②→③→②

モデルファイル(VRP.mod)に部分巡回路除去制約

$$x[2][3] + x[3][2] <= 1;$$

を追加して解き直す

# 配送計画問題をCPLEXで解く

- 解き直し結果（[解]タブ）

```
// solution (optimal) with objective 28
// Quality Incumbent solution:
// MILP objective                                2.8000000000e+01
// MILP solution norm |x| (Total, Max)           1.10000e+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 0 1 0 0 0 0 1 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 1 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 0 0 0 0 1]
     [0 1 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 1 0 0 0 0]
     [1 0 0 0 0 0 0 0 0 0]];
```

最適値 = 28

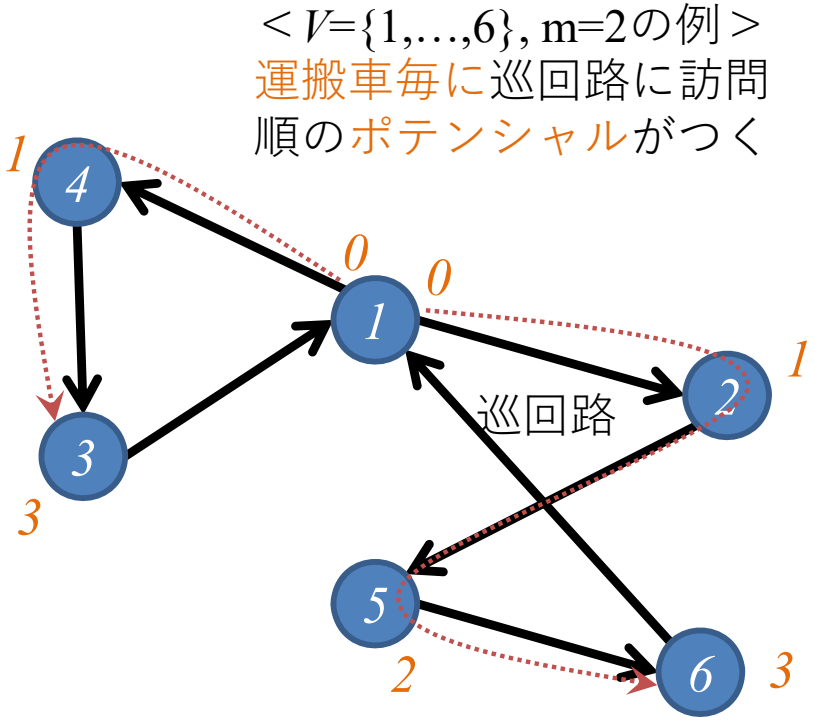
最適解に部分巡回路がない

- ✓ 運搬車1：①→④→⑦→⑩→①
- ✓ 運搬車2：①→⑨→⑥→⑧→②→③→⑤→①
- ✓ 部分巡回路：なし

最適解発見！終了

# 配送計画問題の最適化

- 部分巡回路除去(2) ポテンシャル定式化（変数設定）
  - 0-1変数 $x_{ij}^k = \begin{cases} 1 & \ldots \text{運搬車} k \text{ が枝 } (i,j) \text{ を通る} \\ 0 & \ldots \text{運搬車} k \text{が枝} (i,j) \text{を通らない} \end{cases}$
  - 実数変数 $u_i^k \in [0,p]$ … 運搬車 $k$, 点 $i$ に対するポテンシャル変数
    ※ $p$ … 各運搬車 $k$ が担当する点（顧客）の最大数　$1 \leqq p \leqq n\text{-}1$




- $G = (V, E)$
- $V = V_1 \cup V_2$
- $V_1 = \{1\}$ … depot
- $V_2 = \{2,\ldots,n\}$…customer

# 配送計画問題の最適化

- VRPの定式化（部分巡回路除去(2) ポテンシャル定式化）

min.

$$\sum_{i\in V}\sum_{j\in V;j\neq i}\sum_{k\in K} d_{ij}x_{ij}^{k}$$

- $G=(V,E)$
- $V=V_1\cup V_2$
- $V_1=\{1\}$ … depot
- $V_2=\{2,\ldots,n\}$…customer

s. t.

$$\sum_{j\in V_2} x_{1j}^{k}=1,\ \sum_{j\in V_2} x_{j1}^{k}=1\ (\forall k\in K)$$

各運搬車$k$がデポ（$V_1=\{1\}$）を出る/入るのに使う枝はそれぞれ1本

$$\sum_{j\in V;j\neq i}\sum_{k\in K} x_{ij}^{k}=1,\ \sum_{j\in V;j\neq i}\sum_{k\in K} x_{ji}^{k}=1(\forall i\in V_2)$$

各顧客 $i\in V_2=\{2,\ldots,n\}$ から出る/入るのに使う枝はそれぞれ1本

$$\sum_{j\in V;j\neq i} x_{ij}^{k}=\sum_{j\in V;j\neq i} x_{ji}^{k}\ (\forall i\in V_2,\forall k\in K)$$

顧客 $i$ に出入りするのは1台の運搬車 $k$

$$x_{ij}^{k}\in\{0,1\}\ (\forall i,j\in V,\forall k\in K)$$

部分巡回路除去制約

$$u_1^{k}=0\ (\forall k\in K)$$

$$1\leq u_i^{k}\leq p\ (\forall i\in V_2,\forall k\in K)$$

$$u_i^{k}+1-(n-1)\left(1-x_{ij}^{k}\right)\leq u_j^{k}\ (\forall i\in V,\forall j\in V_2;j\neq i,\forall k\in K)$$

**ポテンシャル制約**

# 配送計画問題をCPLEXで解く

- VRPpot.mod

①

```
int v_max = ...;// 頂点数|V| V=V1&V2, V1=
int m = ...;    // 運搬車の台数
int p_max = ...;// 1台の運搬車が受け持つ最

range V = 1..v_max; // V={1,2,...,n} デポ
range V2 = 2..v_max;// V2={2,3,...,n} デ
range K = 1..m;     // K={1,2,...,m} 運搬

float d[V,V] = ...; // 距離行列D=[dij]

dvar int+ x[V,V,K] in 0..1;// 0-1変数
dvar float+ u[V,K] in 0..p_max;// k毎のポ
```

①②の順にモデルファイル「VRPpot.mod」に記述

②

```
minimize
  sum(i in V) sum(j in V:i!=j) sum(k in K) d[i,j]*x[i,j,k];
subject to {
  forall(k in K) {
    sum(j in V2) x[1,j,k] == 1;// デポから出るのに使う枝は各kで1本
    sum(j in V2) x[j,1,k] == 1;// デポに入るのに使う枝は各kで1本
  }
  forall(i in V2) {
    sum(j in V:j!=i) sum(k in K) x[i,j,k] == 1;// 顧客iから出枝1本
    sum(j in V:j!=i) sum(k in K) x[j,i,k] == 1;// 顧客iに入枝1本
  }
  forall(i in V2) {
    forall(k in K) {
      sum(j in V:j!=i) x[i,j,k] == sum(j in V:j!=i) x[j,i,k];
    }// 点 i に運搬車 k が来たら，kが出て行く
  }
  forall (k in K) { // ポテンシャル制約
    u[1,k] == 0;
    forall(i in V) {
      forall(j in V2:i!=j) {
        u[i,k] + 1 - (v_max - 1)*(1 - x[i,j,k]) <= u[j,k];
      }
    }
  }
  forall (i in V) {
    forall (k in K) {
      x[i,i,k] == 0;// 自己ループ(i→iの枝)はなし
    }
  }
};
```

# 配送計画問題をCPLEXで解く

- VRPex1.dat

```
v_max = 10;// 頂点数|V| V=V1&V2, V1={1}, V2={2,3,...,n}
m = 2;      // 運搬車の台数
p_max = 5; // 1台の運搬車が受け持つ最大点(顧客)数 ←―― 追加

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

# 配送計画問題をCPLEXで解く

- 結果([解]タブ(一部))

```
// solution (optimal) with objective 28
// Quality Incumbent solution:
// MILP objective                                2.8000000000e+01
// MILP solution norm |x| (Total, Max)           5.20000e+01  5.00000e+00
// MILP solution error (Ax=b) (Total, Max)       0.00000e+00  0.00000e+00
// MILP x bound error (Total, Max)               3.77476e-15  3.55271e-15
// MILP x integrality error (Total, Max)         2.22045e-16  2.22045e-16
// MILP slack bound error (Total, Max)           2.66454e-15  1.77636e-15
//
```

最適値 = 28

最適解

$x_{ij}^k$

| i | j | k | val |
|---|---|---|---|
| 1 | 9 | 1 | 1 |
| 9 | 2 | 1 | 1 |
| 2 | 3 | 1 | 1 |
| 3 | 5 | 1 | 1 |
| 5 | 1 | 1 | 1 |
| 1 | 10 | 2 | 1 |
| 10 | 6 | 2 | 1 |
| 6 | 8 | 2 | 1 |
| 8 | 4 | 2 | 1 |
| 4 | 7 | 2 | 1 |
| 7 | 1 | 2 | 1 |

$u_i^k$

| 1 | 2 |
|---|---|
| 0 | 0 |
| 2 | 0 |
| 3 | 5 |
| 0 | 4 |
| 5 | 0 |
| 0 | 2 |
| 5 | 5 |
| 5 | 3 |
| 1 | 0 |
| 0 | 1 |

最適解に部分巡回路がない

- ✓ 運搬車1:①→⑨→②→③→⑤→①
- ✓ 運搬車2:①→⑩→⑥→⑧→④→⑦→①
- ✓ 部分巡回路:なし

# 配送計画問題をCPLEXで解く

- VRPex1.dat

```
v_max = 10;// 頂点数|V| V=V1&V2, V1={1}, V2={2,3,...,n}
m = 3;      // 運搬車の台数
p_max = 3; // 1台の運搬車が受け持つ最大点(顧客)数

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

運搬車数3
受け持ち最大数3
に変更して解いてみる

# 配送計画問題をCPLEXで解く

- 結果([解]タブ(一部))

```
// solution (optimal) with objective 32
// Quality Incumbent solution:
// MILP objective                                3.2000000000e+01
// MILP solution norm |x| (Total, Max)           4.80000e+01   3.00000e+00
// MILP solution error (Ax=b) (Total, Max)       1.31006e-14   6.66134e-16
// MILP x bound error (Total, Max)               3.55271e-15   3.55271e-15
// MILP x integrality error (Total, Max)         1.88738e-15   3.33067e-16
// MILP slack bound error (Total, Max)           3.99680e-15   3.33067e-16
//
```

最適値 = 32

最適解

$x_{ij}^k$

| i | j | k | val |
|---|---|---|---|
| 1 | 4 | 1 | 1 |
| 4 | 7 | 1 | 1 |
| 7 | 5 | 1 | 1 |
| 5 | 1 | 1 | 1 |
| 1 | 3 | 2 | 1 |
| 3 | 2 | 2 | 1 |
| 2 | 9 | 2 | 1 |
| 9 | 1 | 2 | 1 |
| 1 | 10 | 3 | 1 |
| 10 | 6 | 3 | 1 |
| 6 | 8 | 3 | 1 |
| 8 | 1 | 3 | 1 |

$u_i^k$

| 1 | 2 | 3 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 2 | 3 |
| 0 | 1 | 0 |
| 1 | 3 | 3 |
| 3 | 0 | 0 |
| 0 | 0 | 2 |
| 2 | 3 | 0 |
| 0 | 3 | 3 |
| 0 | 3 | 3 |
| 0 | 0 | 1 |

最適解に部分巡回路がない

- ✓ 運搬車1:①→④→⑦→⑤→①
- ✓ 運搬車2:①→③→②→⑨→①
- ✓ 運搬車3:①→⑩→⑥→⑧→①
- ✓ 部分巡回路:なし

# VRP（ポテンシャル定式化）をgurobiで解く

- 問題（ex1）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_ex1():
    V = [1,2,3,4,5,6,7,8,9,10]
    K = [1,2,3]
    P = 3
    d = {(1,1):0,(1,2):7,(1,3):2,(1,4):3,(1,5):8,(1,6):8,(1,7):3,(1,8):8,(1,9):4,(1,10):1,
         (2,1):9,(2,2):0,(2,3):2,(2,4):8,(2,5):8,(2,6):6,(2,7):4,(2,8):8,(2,9):5,(2,10):3,
         (3,1):6,(3,2):1,(3,3):0,(3,4):9,(3,5):1,(3,6):6,(3,7):2,(3,8):7,(3,9):3,(3,10):7,
         (4,1):3,(4,2):6,(4,3):5,(4,4):0,(4,5):5,(4,6):7,(4,7):1,(4,8):4,(4,9):8,(4,10):7,
         (5,1):2,(5,2):9,(5,3):6,(5,4):7,(5,5):0,(5,6):8,(5,7):1,(5,8):4,(5,9):5,(5,10):8,
         (6,1):6,(6,2):7,(6,3):1,(6,4):6,(6,5):6,(6,6):0,(6,7):1,(6,8):2,(6,9):7,(6,10):7,
         (7,1):5,(7,2):9,(7,3):7,(7,4):9,(7,5):6,(7,6):8,(7,7):0,(7,8):9,(7,9):9,(7,10):3,
         (8,1):3,(8,2):5,(8,3):4,(8,4):3,(8,5):6,(8,6):7,(8,7):8,(8,8):0,(8,9):7,(8,10):9,
         (9,1):1,(9,2):2,(9,3):1,(9,4):8,(9,5):8,(9,6):1,(9,7):5,(9,8):8,(9,9):0,(9,10):6,
         (10,1):4,(10,2):9,(10,3):8,(10,4):6,(10,5):6,(10,6):5,(10,7):8,(10,8):8,(10,9):7,(10,10):0,
    } # 距離
    return V,K,P,d
```

# VRP（ポテンシャ

- VRP

  potential定式化

  （ex1）

1つのファイル
「VRPpot.py」に
①②③の順に記述して
保存

③

```
# ####### 実行 #######
if __name__=="__main__":
    V,K,P,d = make_data_ex1()          # テ
    mod = VRPpot(V,K,P,d)
    mod.write("VRPpotex1.lp")          # lp
    mod.optimize()                     # 最
    print("¥n optimal value = ", mod.Obj
    mod.printAttr('X')                 # 最
    mod.write("VRPpotex1.sol")         # 最
```

②

```
# ####### 定式化 #######
def VRPpot(V,K,P,d):
    mod = Model("VRP:potential")

    V2 = V[1:]                  # V2 = [2,3,...,10]
    # 変数設定
    x,u = {},{}
    for k in K:
        for i in V:
            u[i,k] = mod.addVar(vtype="C", lb=0, ub=P, name="u(%s,%s)" % (i,k))
            for j in V:
                if j != i:
                    x[i,j,k] = mod.addVar(vtype="B", name="x(%s,%s,%s)" % (i,j,k))
    mod.update()

    # 制約条件の設定
    for k in K:
        mod.addConstr(quicksum(x[1,j,k] for j in V2) == 1)
        mod.addConstr(quicksum(x[j,1,k] for j in V2) == 1)
    for i in V2:
        mod.addConstr(quicksum(x[i,j,k] for j in V if j!=i for k in K) == 1)
        mod.addConstr(quicksum(x[j,i,k] for j in V if j!=i for k in K) == 1)
        for k in K:
            mod.addConstr(quicksum(x[i,j,k] for j in V if j!=i) == quicksum(x[j,i,k] for j in V if j!=i))
    for k in K:
        mod.addConstr(u[1,k] == 0)
        for i in V:
            for j in V2:
                if j != i:
                    mod.addConstr(u[i,k]+1-(len(V)-1)*(1-x[i,j,k]) <= u[j,k])

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

# VRP（ポテンシャル定式化）をgurobiで解く

- 実行結果

  VRP

  potential定式化

  （ex1）

```
optimal value =  32.0

   Variable            X
-----------------------
    x(1,3,1)           1
      u(2,1)           2
   x(2,10,1)           1
      u(3,1)           1
    x(3,2,1)           1
     u(10,1)           3
   x(10,1,1)           1
    x(1,4,2)           1
      u(4,2)           1
    x(4,7,2)           1
      u(5,2)           3
    x(5,1,2)           1
      u(7,2)           2
    x(7,5,2)           1
    x(1,9,3)           1
      u(6,3)           2
    x(6,8,3)           1
      u(8,3)           3
    x(8,1,3)           1
      u(9,3)           1
    x(9,6,3)           1
gurobi>
```

```
gurobi> exec(open("VRPpot.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 297 rows, 300 columns and 1758 nonzeros
Model fingerprint: 0x40b0d5e5
Variable types: 30 continuous, 270 integer (270 binary)
Coefficient statistics:
  Matrix range     [1e+00, 9e+00]
  Objective range  [1e+00, 9e+00]
  Bounds range     [1e+00, 3e+00]
  RHS range        [1e+00, 8e+00]
Presolve removed 3 rows and 3 columns
Presolve time: 0.01s
Presolved: 294 rows, 297 columns, 3456 nonzeros
Variable types: 27 continuous, 270 integer (270 binary)

Root relaxation: objective 3.000000e+01, 76 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   30.00000    0    4          -   30.00000      -     -    0s
H    0     0                       32.0000000   30.00000  6.25%     -    0s
     0     0   30.25000    0    6   32.00000   30.25000  5.47%     -    0s
     0     0     cutoff    0        32.00000   32.00000  0.00%     -    0s

Explored 1 nodes (149 simplex iterations) in 0.04 seconds (0.02 work units)
Thread count was 20 (of 20 available processors)

Solution count 1: 32

Optimal solution found (tolerance 1.00e-04)
Best objective 3.200000000000e+01, best bound 3.200000000000e+01, gap 0.0000%
```

- ✓ 運搬車1:①→③→②→⑩→①
- ✓ 運搬車2:①→④→⑦→⑤→①
- ✓ 運搬車3:①→⑨→⑥→⑧→①

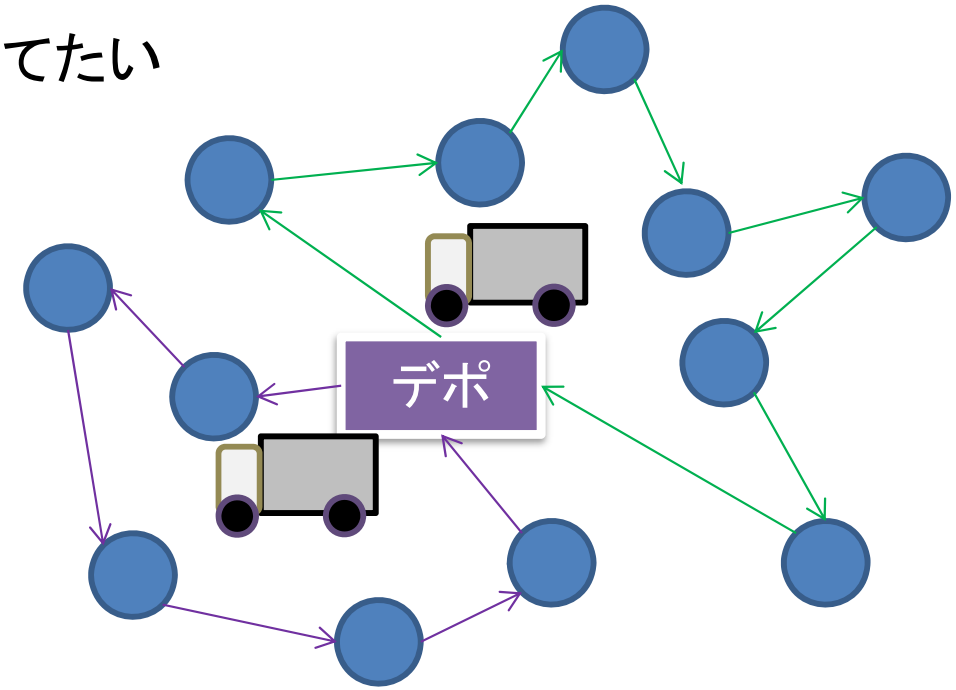
# 【演習】配送計画問題を解く

- VRPの最適化（ex2）
  - V={1,…,10}（川崎=depot, 鹿島～仙台=customer）, $V_1$={1}, $V_2$={2,…,10}
  - 運搬車 m = 3
  - Vの点間の距離行列は以下

| 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 |

# 容量制約付き配送計画問題の最適化

- 容量制約付き配送計画問題 capacitated vehicle routing problem
  - $n$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（ $i \neq j$ ）
  - デポ depotにある複数の運搬車が複数の顧客 customer の需要を満たすよう顧客を巡回し，デポに戻る
  - コストの総和が最小の配送計画をたてたい




- 運搬車 vehicle … $m$台（$m \geqq 1$）
- 運搬車の最大積載量 … $Q$
- 顧客 $i$ の需要 $q_i$ …（$0 \leqq q_i \leqq Q$）

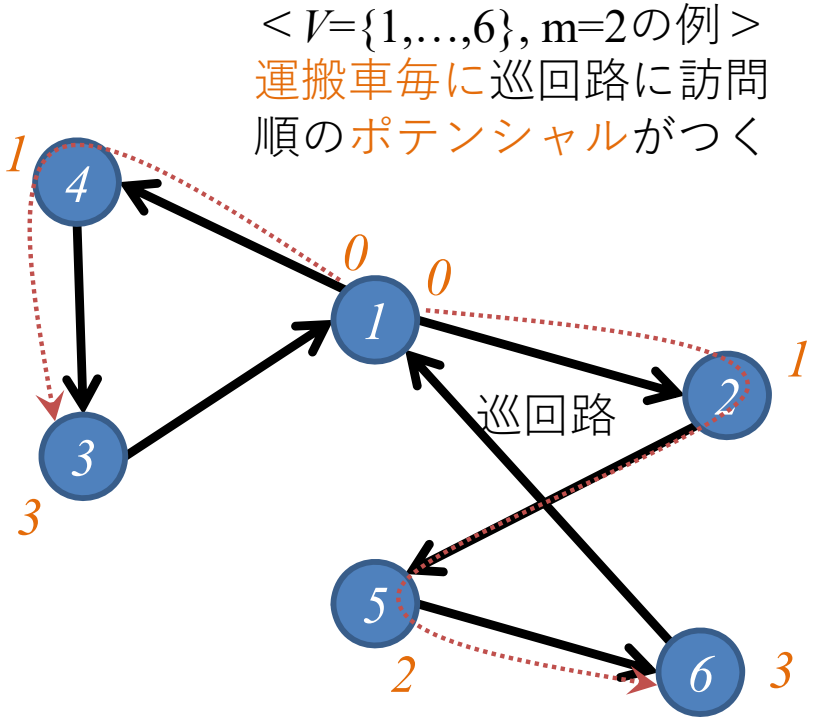
※もし $q_i > Q$ なら，$i$ を分割して，同じ場所に居る別の顧客と考えれば良いので，常に $q_i \leqq Q$ と仮定して良い

- 完全有向グラフ $G = (V, E)$
- 点集合 $V = V_1 \cup V_2 = \{1,2,...,n\}$
  - デポ集合 $V_1 = \{1\}$
  - 顧客集合 $V_2 = \{2,\ldots,n\}$
- 枝$(i, j) \in E$上のコスト $d_{ij}$

- $\forall i, j,\ d_{ij} = d_{ji}$ のとき対称VRP
- $\exists i, j,\ d_{ij} \neq d_{ji}$ のとき非対称VRP

# 容量制約付き配送計画問題の最適化

- CVRPの定式化（ポテンシャル定式化:変数設定）
  - 0-1変数 $x_{ij}^k = \begin{cases} 1 & \ldots \text{運搬車} k \text{ が枝 } (i,j) \text{ を通る} \\ 0 & \ldots \text{運搬車} k \text{が枝} (i,j) \text{を通らない} \end{cases}$
  - 実数変数 $u_i^k \in [0,p]$ … 運搬車 $k$, 点 $i$ に対するポテンシャル変数
    ※ $p$ … 各運搬車 $k$ が担当する点（顧客）の最大数　$1 \leqq p \leqq n\text{-}1$




- $G = (V, E)$
- $V = V_1 \cup V_2$
- $V_1 = \{1\}$ … depot
- $V_2 = \{2,\ldots,n\}$…customer

# 容量制約付き配送計画問題の最適化

- CVRPの定式化（ポテンシャル定式化）

min.

$$\sum_{i\in V}\sum_{j\in V;j\neq i}\sum_{k\in K} d_{ij}x_{ij}^{k}$$

- $G=(V,E)$
- $V=V_1\cup V_2$
- $V_1=\{1\}$ … depot
- $V_2=\{2,\ldots,n\}$…customer

s. t.

$$\sum_{j\in V_2} x_{1j}^{k}=1,\ \sum_{j\in V_2} x_{j1}^{k}=1\ (\forall k\in K)$$

各運搬車$k$がデポ（$V_1=\{1\}$）を出る/入るのに使う枝はそれぞれ1本

$$\sum_{j\in V;j\neq i}\sum_{k\in K} x_{ij}^{k}=1,\ \sum_{j\in V;j\neq i}\sum_{k\in K} x_{ji}^{k}=1(\forall i\in V_2)$$

各顧客 $i\in V_2=\{2,\ldots,n\}$ から出る/入るのに使う枝はそれぞれ1本

$$\sum_{j\in V;j\neq i} x_{ij}^{k}=\sum_{j\in V;j\neq i} x_{ji}^{k}\ (\forall i\in V_2,\forall k\in K)$$

顧客 $i$ に出入りするのは1台の運搬車 $k$

$$\sum_{i\in V_2}\sum_{j\in V;j\neq i} q_i x_{ij}^{k}\le Q\ (\forall k\in K)$$

運搬車$k$ 毎に積載量が最大容量以内

$$x_{ij}^{k}\in\{0,1\}\ (\forall i,j\in V,\forall k\in K)$$

部分巡回路除去制約

$$u_1^{k}=0\ (\forall k\in K)$$

$$1\le u_i^{k}\le p\ (\forall i\in V_2,\forall k\in K)$$

$$u_i^{k}+1-(n-1)\left(1-x_{ij}^{k}\right)\le u_j^{k}\ (\forall i\in V,\forall j\in V_2;j\neq i,\forall k\in K)$$

**ポテンシャル制約**

# 容量制約付き配送計画問題をCPLEXで解く

- CVRPpot.mod

①

```
int v_max = ...;// 頂点数|V| V=V1&V2, V1=
int m = ...;    // 運搬車の台数
int p_max = ...;// 1台の運搬車が受け持つ最
int Q = ...;    // 運搬車の最大積載量

range V = 1..v_max; // V={1,2,...,n} デポ
range V2 = 2..v_max;// V2={2,3,...,n} デ
range K = 1..m;     // K={1,2,...,m} 運搬

float d[V,V] = ...; // 距離行列D=[dij]
float q[V] = ...;   // 顧客需要

dvar int+ x[V,V,K] in 0..1;// 0-1変数
dvar float+ u[V,K] in 0..p_max;// k毎のポ
```

①②の順にモデルファイル「CVRPpot.mod」に記述

②

```
minimize
  sum(i in V) sum(j in V:i!=j) sum(k in K) d[i,j]*x[i,j,k];
subject to {
  forall(k in K) {
    sum(j in V2) x[1,j,k] == 1;// デポから出るのに使う枝は各kで1本
    sum(j in V2) x[j,1,k] == 1;// デポに入るのに使う枝は各kで1本
  }
  forall(i in V2) {
    sum(j in V:j!=i) sum(k in K) x[i,j,k] == 1;// 顧客iから出枝1本
    sum(j in V:j!=i) sum(k in K) x[j,i,k] == 1;// 顧客iに入枝1本
  }
  forall(i in V2) {
    forall(k in K) {
      sum(j in V:j!=i) x[i,j,k] == sum(j in V:j!=i) x[j,i,k];
    }// 点 i に運搬車 k が来たら，kが出て行く
  }
  forall (k in K) {
    sum(i in V2) sum(j in V:i!=j) q[i]*x[i,j,k] <= Q;
  }
  forall (k in K) { // ポテンシャル制約
    u[1,k] == 0;
    forall(i in V) {
      forall(j in V2:i!=j) {
        u[i,k] + 1 - (v_max - 1)*(1 - x[i,j,k]) <= u[j,k];
      }
    }
  }
  forall (i in V) {
    forall (k in K) {
      x[i,i,k] == 0;// 自己ループ(i→iの枝)はなし
    }
  }
};
```

# 容量制約付き配送計画問題をCPLEXで解く

- CVRPex1.dat

```
v_max = 10;// 頂点数|V| V=V1&V2, V1={1}, V2={2,3,...,n}
m = 3;      // 運搬車の台数
p_max = 4; // 1台の運搬車が受け持つ最大点(顧客)数
Q = 53;     // 運搬車の最大積載量

d = [
[ 0 7 2 3 8 8 3 8 4 1 ]
[ 9 0 2 8 8 6 4 8 5 3 ]
[ 6 1 0 9 1 6 2 7 3 7 ]
[ 3 6 5 0 5 7 1 4 8 7 ]
[ 2 9 6 7 0 8 1 4 5 8 ]
[ 6 7 1 6 6 0 1 2 7 7 ]
[ 5 9 7 9 6 8 0 9 9 3 ]
[ 3 5 4 3 6 7 8 0 7 9 ]
[ 1 2 1 8 8 1 5 8 0 6 ]
[ 4 9 8 6 6 5 8 8 7 0 ]
];
q = [ 0 13 20 18 16 17 20 13 15 20 ];
```

追加

# 容量制約付き配送計画問題をCPLEXで解く

- 結果([解]タブ(一部))

```
// solution (optimal) with objective 36
// Quality Incumbent solution:
// MILP objective                                3.6000000000e+01
// MILP solution norm |x| (Total, Max)           3.80000e+01  4.00000e+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
//
```

最適値 = 36

最適解

$x_{ij}^k$

| i | j | k | val |
|---|---|---|---|
| 1 | 3 | 1 | 1 |
| 3 | 2 | 1 | 1 |
| 2 | 10 | 1 | 1 |
| 10 | 1 | 1 | 1 |
| 1 | 4 | 2 | 1 |
| 4 | 5 | 2 | 1 |
| 5 | 8 | 2 | 1 |
| 8 | 1 | 2 | 1 |
| 1 | 9 | 3 | 1 |
| 9 | 6 | 3 | 1 |
| 6 | 7 | 3 | 1 |
| 7 | 1 | 3 | 1 |

$u_i^k$

| 1 | 2 | 3 |
|---|---|---|
| 0 | 0 | 0 |
| 2 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 2 | 0 |
| 0 | 0 | 2 |
| 0 | 4 | 3 |
| 0 | 3 | 4 |
| 0 | 0 | 1 |
| 3 | 0 | 0 |

最適解に部分巡回路がない

- ✓ 運搬車1:①→③→②→⑩→①
- ✓ 運搬車2:①→④→⑤→⑧→①
- ✓ 運搬車3:①→⑨→⑥→⑦→①
- ✓ 部分巡回路:なし

積載量

- ✓ 運搬車1:20+13+20=53 <=53
- ✓ 運搬車2:18+16+13=47 <=53
- ✓ 運搬車3:15+17+20=52 <=53

# 容量制約付き配送計画問題をgurobiで解く

- 問題（ex1）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_ex1():
    V = [1,2,3,4,5,6,7,8,9,10]
    K = [1,2,3]
    P = 4
    Q = 53
    q = [0,13,20,18,16,17,20,13,15,20]
    d = {(1,1):0,(1,2):7,(1,3):2,(1,4):3,(1,5):8,(1,6):8,(1,7):3,(1,8):8,(1,9):4,(1,10):1,
            (2,1):9,(2,2):0,(2,3):2,(2,4):8,(2,5):8,(2,6):6,(2,7):4,(2,8):8,(2,9):5,(2,10):3,
            (3,1):6,(3,2):1,(3,3):0,(3,4):9,(3,5):1,(3,6):6,(3,7):2,(3,8):7,(3,9):3,(3,10):7,
            (4,1):3,(4,2):6,(4,3):5,(4,4):0,(4,5):5,(4,6):7,(4,7):1,(4,8):4,(4,9):8,(4,10):7,
            (5,1):2,(5,2):9,(5,3):6,(5,4):7,(5,5):0,(5,6):8,(5,7):1,(5,8):4,(5,9):5,(5,10):8,
            (6,1):6,(6,2):7,(6,3):1,(6,4):6,(6,5):6,(6,6):0,(6,7):1,(6,8):2,(6,9):7,(6,10):7,
            (7,1):5,(7,2):9,(7,3):7,(7,4):9,(7,5):6,(7,6):8,(7,7):0,(7,8):9,(7,9):9,(7,10):3,
            (8,1):3,(8,2):5,(8,3):4,(8,4):3,(8,5):6,(8,6):7,(8,7):8,(8,8):0,(8,9):7,(8,10):9,
            (9,1):1,(9,2):2,(9,3):1,(9,4):8,(9,5):8,(9,6):1,(9,7):5,(9,8):8,(9,9):0,(9,10):6,
            (10,1):4,(10,2):9,(10,3):8,(10,4):6,(10,5):6,(10,6):5,(10,7):8,(10,8):8,(10,9):7,(10,10):0,
    } # 距離
    return V,K,P,Q,q,d
```

# 容量制約付き配

- CVRP

  potential定式化

  （ex1）

1つのファイル
「CVRPpot.py」に
①②③の順に記述して
保存

②

```
# ####### 定式化 #######
def CVRPpot(V,K,P,Q,q,d):
    mod = Model("capacitated VRP:potential")

    V2 = V[1:]     # V2 = [2,3,...,10]
    # 変数設定
    x,u = {},{}
    for k in K:
        for i in V:
            u[i,k] = mod.addVar(vtype="C", lb=0, ub=P, name="u(%s,%s)" % (i,k))
            for j in V:
                if j != i:
                    x[i,j,k] = mod.addVar(vtype="B", name="x(%s,%s,%s)" % (i,j,k))
    mod.update()

    # 制約条件の設定
    for k in K:
        mod.addConstr(quicksum(x[1,j,k] for j in V2) == 1)
        mod.addConstr(quicksum(x[j,1,k] for j in V2) == 1)
    for i in V2:
        mod.addConstr(quicksum(x[i,j,k] for j in V if j!=i for k in K) == 1)
        mod.addConstr(quicksum(x[j,i,k] for j in V if j!=i for k in K) == 1)
        for k in K:
            mod.addConstr(quicksum(x[i,j,k] for j in V if j!=i) == quicksum(x[j,i,k] for j in V if j!=i))
    for k in K:
        mod.addConstr(quicksum(q[i-1]*x[i,j,k] for i in V2 for j in V if j!=i) <= Q)
        mod.addConstr(u[1,k] == 0)
        for i in V:
            for j in V2:
                if j != i:
                    mod.addConstr(u[i,k]+1-(len(V)-1)*(1-x[i,j,k]) <= u[j,k])

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

③

```
# ####### 実行 #######
if __name__=="__main__":
    V,K,P,Q,q,d = make_data_ex1()    # デ
    mod = CVRPpot(V,K,P,Q,q,d)       # モ
    mod.write("CVRPpotex1.lp")       # lp
    mod.optimize()                   # 最
    print("¥n optimal value = ", mod.Obj
    mod.printAttr('X')               # 最
    mod.write("CVRPpotex1.sol")      # 最
```

# 容量制約付

- 実行結果

  VRP

  potential定式化

```
optimal value =  36.0

   Variable            X
-----------------------
    x(1,3,1)           1
      u(2,1)           2
   x(2,10,1)           1
      u(3,1)           1
    x(3,2,1)           1
      u(6,1)           4
     u(10,1)           3
   x(10,1,1)           1
    x(1,4,2)           1
      u(2,2)           4
      u(3,2)           4
      u(4,2)           1
    x(4,5,2)           1
      u(5,2)           3
    x(5,8,2)           1
      u(7,2)           4
      u(8,2)           4
    x(8,1,2)           1
     u(10,2)           4
    x(1,9,3)           1
      u(6,3)           3
    x(6,7,3)           1
      u(7,3)           4
    x(7,1,3)           1
      u(9,3)           1
    x(9,6,3)           1
gurobi>
```

```
gurobi> exec(open("CVRPpot.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 300 rows, 300 columns and 2001 nonzeros
Model fingerprint: 0x4f2fb06c
Variable types: 30 continuous, 270 integer (270 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+01]
  Objective range  [1e+00, 9e+00]
  Bounds range     [1e+00, 4e+00]
  RHS range        [1e+00, 5e+01]
Found heuristic solution: objective 68.0000000
Presolve removed 3 rows and 3 columns
Presolve time: 0.01s
Presolved: 297 rows, 297 columns, 3909 nonzeros
Variable types: 27 continuous, 270 integer (270 binary)

Root relaxation: objective 3.000000e+01, 101 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   30.00000    0   16   68.00000   30.00000  55.9%     -    0s
H    0     0                       37.00000   33.00000  10.8%     -    0s
H    0     0                       36.0000000 34.66667  3.70%  20.3    0s
     0     0
H    0     0
     0     0
     0     0
H    0     0

Cutting planes:
  Gomory: 3
  Cover: 1
  MIR: 17
  StrongCG: 4
  GUB cover: 5
  Inf proof: 3
  Zero half: 2
  RLT: 7
  Relax-and-lift: 3

Explored 43 nodes (2171 simplex iterations) in 0.25 seconds (0.22 work units)
Thread count was 20 (of 20 available processors)

Solution count 6: 36 37 39 ... 68

Optimal solution found (tolerance 1.00e-04)
Best objective 3.600000000000e+01, best bound 3.600000000000e+01, gap 0.0000%
```

最適解

- ✓ 運搬車1:①→③→②→⑩→①
- ✓ 運搬車2:①→④→⑤→⑧→①
- ✓ 運搬車3:①→⑨→⑥→⑦→①

積載量

- ✓ 運搬車1:20+13+20=53 <=53
- ✓ 運搬車2:18+16+13=47 <=53
- ✓ 運搬車3:15+17+20=52 <=53