《論　文》

容量制約のないネットワーク設計問題の Lagrange緩和法とソースコード

片　山　直　登

1　はじめに

ネットワークデザイン問題は，適切なネットワークの形状と多品種のフローを求める問題であり，通信ネットワーク設計，交通ネットワーク設計や輸送・配送ネットワーク設計などに様々な応用分野が存在する基本的な問題である．

ネットワークを設計する際に考慮すべき基本的な費用は，アークを選択するときに発生する費用であるデザイン費用とモノがアーク上を移動するときに発生するフロー費用である．本研究では，アーク容量の制限を考慮しない条件のもとで，これら二種類の費用の合計を最小にするような多品種のネットワーク設計問題を対象とする．アーク容量の制限を考慮しないため，この問題は容量制約のないネットワーク設計問題（Uncapacitated Network Design Problem: *UND*）とよばれる．*UND*に対しては数多くの研究が行われており，研究のサーベイとしてはMagnanti–Wong（1984）[6]，Wong（1984）[8]，Balakrishnan–Magnanti–Mirchandani（1997）[1]，片山直登（2002）[2]および片山直登（2008）[3]などがある．

本研究では，*UND*の定式化，Lagrange緩和法およびLagrangeヒューリスティックを示し，これらの解法のためのＣ言語によるソースコードを示す．

2　問題の定式化

*UND*は，ノード集合，デザイン費用とフロー費用をもつ向きをもたないアーク集合，品種の需要をもつ品種集合が与えられているとき，フロー費用とデザイン費用の合計を最小にするアーク集合とフローを求める問題である．

はじめに，前提条件を示す．

・ノード集合が与えられている.

・アーク集合が与えられている.

・アークは向きをもたない.

・アークの処理量には，容量である上限を考慮しない.

・アークには，非負のデザイン費用が与えられている.

・アークには，単位当たりの非負のフロー費用が与えられている.

・複数の品種からなる品種集合が与えられている.

・各品種の需要は 1 である.

・各品種の需要は，始点から終点までのパス上を移動する.

・ネットワークは連結する.

次に，記号の定義を示す.

N：ノード集合

A：アーク集合

K：品種集合

N_n：ノード n に接続するアークの他方の端点の集合

K_n^o：ノード n を始点とする品種の集合

O^k：品種 k の始点

D^k：品種 k の終点

$K_{O^k}^o$：品種 k の始点O^kを始点とする品種の集合

c_{ij}^k：アーク（i,j）上を $i \rightarrow j$ 向きに移動する品種 k の単位当たりの非負のフロー費用

f_{ij}：アーク（i,j）の非負のデザイン費用

x_{ij}^k：品種 k のフローがアーク（i,j）上を $i \rightarrow j$ 向きに移動する量を表すアークフロー変数；非負の連続変数

y_{ij}：アーク（i,j）を選択するとき 1，そうでないとき 0 であるデザイン変数；0−1変数

Y：ネットワークが連結するようなデザイン変数の集合

v_n^k：品種 k，ノード n に関するフロー保存式に対する双対変数；連続変数

s_n^k：品種 k，ノード n に関するフロー保存式に対する劣勾配；整数変数

次に，*UND*の定式化を示す.

$$\text{最小化} \quad \sum_{(i,j)\in A}\sum_{k\in K}(c_{ij}^k x_{ij}^k + c_{ji}^k x_{ji}^k) + \sum_{(i,j)\in A} f_{ij} y_{ij} \tag{1}$$

$$\text{条件} \quad \sum_{i\in N_n} x_{in}^k - \sum_{j\in N_n} x_{nj}^k = \begin{cases} -1 & if\ n = O^k \\ 1 & if\ n = D^k \\ 0 & otherwise \end{cases} \quad \forall n \in N, k \in K \tag{2}$$

$$x_{ij}^k + x_{ji}^h \leq y_{ij} \quad \forall h \in K_{O^k}^o, k \in K, (i,j) \in A \tag{3}$$

$$\boldsymbol{y} \in Y \tag{4}$$
$$x_{ij}^k \geq 0,\ x_{ji}^k \geq 0 \quad \forall k \in K, (i,j) \in A \tag{5}$$
$$y_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{6}$$

⑴式は目的関数であり，フロー費用とデザイン費用の総和を最小化する．⑵式はフロー保存式であり，ノードに流入するフローと流出するフローの差が，品種 k の始点であれば－１，終点であれば１，その他のノードであれば０であること表す．この式は，各品種について，必ず始点から終点まで需要が移動することを保証する．⑶式は，非逆流不等式である．左辺は品種 k の $i \rightarrow j$ 向きのフローと k と始点が同じである品種 h の $j \rightarrow i$ 向きのフローの和であり，これがアーク（i, j）が選択されるときに最大で１，選択されないときに０となることを表す．⑷式は，y_{ij}=１からなるアークで構成されるネットワークが連結することを表す．⑸式はフロー変数の非負条件，⑹式はデザイン変数の0－1条件である．

3　Lagrange緩和法

*UND*に対して，Lagrange乗数 $\boldsymbol{v}$ を用いて，フロー保存式⑵をLagrange緩和した問題 LGを作成する．LGは次のように表される．

（LG）

最小化
$$\sum_{k\in K}(v_{D^k}^k - v_{O^k}^k) + \sum_{(i,j)\in A}\sum_{k\in K}\{(c_{ij}^k - v_j^k + v_i^k)x_{ij}^k + (c_{ji}^k - v_i^k + v_j^k)x_{ji}^k\} + \sum_{(i,j)\in A} f_{ij}y_{ij} \tag{7}$$

条件
$$x_{ij}^k + x_{ji}^h \leq y_{ij} \quad \forall k \in K_{O^k}^o, k \in K, (i,j) \in A \tag{8}$$
$$x_{ij}^k \geq 0,\ x_{ji}^k \geq 0 \quad \forall k \in K, (i,j) \in A \tag{9}$$
$$\boldsymbol{y} \in Y \tag{10}$$
$$y_{ij} \in \{0,1\} \quad \forall (i,j) \in A \tag{11}$$

はじめに，連結条件⑽を取り除いた問題LG'を考える．適当な $\boldsymbol{v}$ を与えると目的関数の第一項$\sum_{k\in K}(v_{D^k}^k - v_{O^k}^k)$は定数項として扱えるため，$LG'$はアーク（$i, j$）毎の独立した次のような問題$LG_{ij}$ に分割できる．

（LG_{ij}）

最小化
$$\sum_{k\in K}\{(c_{ij}^k - v_j^k + v_i^k)x_{ij}^k + (c_{ji}^k - v_i^k + v_j^k)x_{ji}^k\} + f_{ij}y_{ij} \tag{12}$$

条件
$$x_{ij}^k + x_{ji}^h \leq y_{ij} \quad \forall h \in K_{O^k}^o, k \in K \tag{13}$$
$$x_{ij}^k \geq 0,\ x_{ji}^k \geq 0 \quad \forall k \in K \tag{14}$$

$$y_{ij} \in \{0,1\} \tag{15}$$

始点を n とする品種集合K_n^oを用いると，k∈Kはn∈N，k∈K_n^oと表現でき，h∈$K_{O^k}^o$はh∈K_n^oと表現できるので，LG_{ij}は次のような問題LG'_{ij}に書き直すことができる．

（LG'_{ij}）

$$\text{最小化} \quad \sum_{n\in N}\sum_{k\in K_n^o}\{(c_{ij}^k - v_j^k + v_i^k)x_{ij}^k + (c_{ji}^k - v_i^k + v_j^k)x_{ji}^k\} + f_{ij}y_{ij} \tag{16}$$

$$\text{条件} \quad x_{ij}^k + x_{ji}^h \le y_{ij} \quad \forall h \in K_n^o, k \in K_n^o, n \in N \tag{17}$$

$$x_{ij}^k \ge 0,\ x_{ji}^k \ge 0 \quad \forall k \in K_n^o, n \in N \tag{18}$$

$$y_{ij} \in \{0,1\} \tag{19}$$

LG'_{ij}において，y_{ij}は 0 または 1 のどちらかである．そこで，y_{ij}= 1 とy_{ij}= 0 の場合に分けて考える．

y_{ij} = 1 である場合，$f_{ij}y_{ij}$は定数項となるので，LG'_{ij}は次のようなノード n 毎の独立した問題LG_{ij}^{n1}に分割できる．

（LG_{ij}^{n1}）

$$\text{最小化} \quad \sum_{k\in K_n^o}\{(c_{ij}^k - v_j^k + v_i^k)x_{ij}^k + (c_{ji}^k - v_i^k + v_j^k)x_{ji}^k\} \tag{20}$$

$$\text{条件} \quad x_{ij}^k + x_{ji}^h \le 1 \quad \forall k \in K_n^o,\ h \in K_n^o \tag{21}$$

$$x_{ij}^k \ge 0,\ x_{ji}^k \ge 0 \quad \forall k \in K_n^o \tag{22}$$

(21)式は，高々一方向のフローを許す非逆流制約式となる．$i \to j$ 方向のフローが存在するとき，x_{ij}^kの上限は 1 であり，かつ最小化問題であるため，目的関数値の最適値は $\sum_{k\in K_n^o}\min(0, c_{ij}^k - v_j^k + v_i^k)$ となる．一方，$j \to i$ 方向のフローが存在するとき，目的関数値の最適値は $\sum_{k\in K_n^o}\min(0, c_{ji}^k - v_i^k + v_j^k)$ となる．最小化問題であるため，これらの小さい方が選ばれる．このため，LG_{ij}^{n1}の最適値 $\tilde{\phi}_{ij}^{n1}$は，

$$\tilde{\phi}_{ij}^{n1} = \min\Big\{\sum_{k\in K_n^o}\min(0, c_{ij}^k - v_i^k + v_j^k),\ \sum_{k\in K_n^o}\min(0, c_{ij}^k - v_i^k + v_j^k)\Big\} \quad n \in N, (i,j) \in A \tag{23}$$

となり，最適解 $\tilde{x}$ は次のようになる．

$$\tilde{x}_{ij}^k = \begin{cases} 1 & if\ \sum_{k\in K_n^o}\min(0, c_{ij}^k - v_j^k + v_i^k) \le \sum_{k\in K_n^o}\min(0, c_{ji}^k - v_i^k + v_j^k) \\ & \quad and\ c_{ij}^k - v_i^k + v_j^k < 0 \\ 0 & otherwise \end{cases} \quad \forall k \in K, (i,j) \in A \tag{24}$$

$$\tilde{x}_{ji}^k = \begin{cases} 1 & if\ \sum_{k\in K_n^o}\min(0, c_{ji}^k - v_i^k + v_j^k) < \sum_{k\in K_n^o}\min(0, c_{ij}^k - v_j^k + v_i^k) \\ & \quad and\ c_{ji}^k - v_j^k + v_i^k < 0 \\ 0 & otherwise \end{cases} \quad \forall k \in K, (i,j) \in A \tag{25}$$

一方，$y_{ij}=0$ である場合，LG_{ij}の最適値は 0 である．したがって，LG_{ij}はフロー変数 $\boldsymbol{x}$ を用いない次のような問題LGY_{ij}に帰着できる．

(LGY_{ij})

$$\text{最小化}\quad \Big[\sum_{n\in N}\min\Big\{\sum_{k\in K_n^o}\min(0,c_{ij}^k-v_j^k+v_i^k),\ \sum_{k\in K_n^o}\min(0,c_{ji}^k-v_i^k+v_j^k)\Big\}+f_{ij}\Big]y_{ij} \tag{26}$$

$$\text{条件}\quad y_{ij}\in\{0,1\} \tag{27}$$

以上のことから，連結条件⑽も考慮すると，LGはフロー変数 $\boldsymbol{x}$ を用いない次のような問題LGYに帰着できる．

(LGY)

$$\begin{aligned}\text{最小化}\quad &\sum_{k\in K}(v_{D^k}^k-v_{O^k}^k)\\ &+\sum_{(i,j)\in A}\Big[\sum_{n\in N}\min\Big\{\sum_{k\in K_n^o}\min(0,c_{ij}^k-v_j^k+v_i^k),\ \sum_{k\in K_n^o}\min(0,c_{ji}^k-v_i^k+v_j^k)\Big\}\\ &\qquad+f_{ij}\Big]y_{ij}\end{aligned} \tag{28}$$

$$\text{条件}\quad \boldsymbol{y}\in Y \tag{29}$$

$$y_{ij}\in\{0,1\}\quad \forall(i,j)\in A \tag{30}$$

アークの重みWを⑿式の $\boldsymbol{y}$ の係数とし，次のようにおく．

$$W_{ij}=\sum_{n\in N}\min\Big\{\sum_{k\in K_n^o}\min(0,c_{ij}^k-v_j^k+v_i^k),\ \sum_{k\in K_n^o}\min(0,c_{ji}^k-v_i^k+v_j^k)\Big\}+f_{ij}\quad \forall(i,j)\in A \tag{31}$$

LGYは，アークの重みWの合計が最小となる連結を求める問題となる．ただし，重みが負となる場合があることに注意する．この問題の解は，最小木問題の解と重みが負であるアークを併せたものとなる．

*LGY*の解法

［ステップ 1］$W_{ij}<0$ であるアーク $(i,j)\in A$について$\hat{y}_{ij}:=1$ とし，それ以外を$\hat{y}_{ij}:=0$ とする．

［ステップ 2］$\hat{\boldsymbol{y}}$ により連結される成分をノードに縮約する．連結成分に含まれるノード集合Mの縮約ノードmとGの縮約ノードg間のアークの重みを$W'_{mg}:=\min_{i\in M,j\in G}W_{ij}$

［ステップ 3］アークの重みをW'とした縮約したネットワーク上で，最小木を求める．

［ステップ 4］最小木に含まれるアーク (i,j) について，$\hat{y}_{ij}:=1$ とする．

ステップ3において，最小木はKruskal法[3]などを用いて求めることができる．

LGYの最適解$\hat{\boldsymbol{y}}$を用いると，LGの最適解$\hat{\boldsymbol{x}}$は次のようになる．

$$\hat{x}_{ij}^k = \begin{cases} \tilde{x}_{ij}^k & if\ \hat{y}_{ij} = 1 \\ 0 & otherwise \end{cases}, \qquad \hat{x}_{ji}^k = \begin{cases} \tilde{x}_{ji}^k & if\ \hat{y}_{ij} = 1 \\ 0 & otherwise \end{cases} \qquad \forall k \in K, (i,j) \in A \tag{32}$$

$\hat{\boldsymbol{y}}$ をLGYの目的関数に代入すれば，LGの最適値，すなわちUNDの下界値を求めることができる．

$\hat{\boldsymbol{y}}$，$\hat{\boldsymbol{x}}$が求められたときに，LGにおいてLagrange乗数 $\boldsymbol{v}$ を変数と見なした最大化問題をLDとすると，LDは次のように表される．

（LD）

$$\begin{aligned} \text{最大化} \quad & \sum_{(i,j)\in A} \sum_{k\in K} (c_{ij}^k \hat{x}_{ij}^k + c_{ji}^k \hat{x}_{ji}^k) + \sum_{(i,j)\in A} f_{ij} \hat{y}_{ij} \\ & + \sum_{k\in K} v_{O^k}^k \Big(-1 - \sum_{i\in N_{O^k}} \hat{x}_{iO^k}^k + \sum_{j\in N_{O^k}} \hat{x}_{O^k j}^k \Big) + \sum_{k\in K} v_{D^k}^k \Big(1 - \sum_{i\in N_{D^k}} \hat{x}_{iD^k}^k + \sum_{j\in N_{D^k}} \hat{x}_{D^k j}^k \Big) \\ & + \sum_{k\in K} \sum_{n\in N\setminus\{O^k, D^k\}} v_n^k \Big(- \sum_{i\in N_n} \hat{x}_{in}^k + \sum_{j\in N_n} \hat{x}_{nj}^k \Big) \end{aligned} \tag{33}$$

LDは陽的な制約条件をもたない問題である．しかし，$\boldsymbol{v}$が変化するとLGの最適解である$\hat{\boldsymbol{x}}$が離散的に変化し，LDにおける$\boldsymbol{v}$の係数も離散的に変化することになる．このため，LDは微分不可能な項を含む最適化問題となる．そこで，LDに対して劣勾配法[5]を適用する．

$\boldsymbol{v}$に関する劣勾配$\boldsymbol{s}$を$\boldsymbol{v}$の係数，すなわち，

$$s_n^k = \begin{cases} -1 - \sum_{i\in N_n} \hat{x}_{in}^k + \sum_{j\in N_n} \hat{x}_{nj}^k & if\ n = O^k \\ 1 - \sum_{i\in N_n} \hat{x}_{in}^k + \sum_{j\in N_n} \hat{x}_{nj}^k & if\ n = D^k \\ - \sum_{i\in N_n} \hat{x}_{in}^k + \sum_{j\in N_n} \hat{x}_{nj}^k & otherwise \end{cases} \qquad \forall k \in K, n \in N \tag{34}$$

とする．

この劣勾配を用いて次のように乗数を更新する．

$$v_n^k := v_n^k + \theta_l s_n^k \quad \forall k \in K, n \in N \tag{35}$$

ここで，θ_lはl回目の繰り返しにおけるステップサイズである．適当なUNDの最良の上界値をUB，l回目の繰り返しにおけるUNDの下界値をLB_l，パラメータをρ（$0 < \rho < 2$）としたときに，θ_lは次式で与えられる．

$$\theta_l = \frac{\rho(UB - LB_l)}{\sum_{k\in K} \sum_{n\in N} (v_n^k)^2} \tag{36}$$

*UND*に対するLagrange緩和法の流れをまとめておく．

Lagrange緩和法

［ステップ1］適当な $\boldsymbol{v}$ の初期値を与える．繰返し回数の上限を l_{max}，収束判定基準を ϵ とする．劣勾配法のパラメータを ρ，ρ の変更率を λ，変更周期を l_ρ とする．最良の上界値を*UB*，最良の下界値を*LB*とする．$UB:=\infty$，$LB:=0$，$\rho:=1$，$l:=1$ とする．

［ステップ2］*LG*を解き，l 回目の下界値 LB_l および最適解 $\hat{\boldsymbol{y}}$，$\hat{\boldsymbol{x}}$ を求める．$LB_l>LB$ であれば $LB:=LB_l$ とする．

［ステップ3］$\hat{\boldsymbol{y}}$ を用いて適当なLagrangeヒューリスティックを行い，l 回目の上界値 UB_l を求める．$UB_l<UB$ であれば $UB:=UB_l$ とする．

［ステップ4］$l=l_{max}$ または $(UB-LB)/LB<\epsilon$ であれば終了する．

［ステップ5］$l \bmod l_\rho=0$ であれば，$\rho:=\rho\times\lambda$ とする．

［ステップ6］$\hat{\boldsymbol{y}}$，$\hat{\boldsymbol{x}}$ より劣勾配 $\boldsymbol{s}$ を求め，劣勾配法により $\boldsymbol{v}$ を更新する．$l:=l+1$ として，ステップ2へ戻る．

4　Lagrange ヒューリスティック

Lagrange緩和解を用いた三種類のLagrangeヒューリスティックを示す．Lagrange緩和法では，劣勾配法の繰返し毎に緩和解が得られる．Lagrangeヒューリスティックは，Lagrange緩和解を改良して近似解および上界値を求める解法である．劣勾配法の繰返し毎に近似解を求めることができ，これらの内の最良解を最終的な近似解として採用する．

4．1．初期解

Lagrange緩和問題ではフロー保存式を緩和しているため，フロー変数がフロー保存式を満たしているとは限らない．しかし，デザイン変数解は実行可能解となる．Lagrange 緩和問題のデザイン変数解を $\hat{\boldsymbol{y}}$ とする．*UND*において $\boldsymbol{y}$ を $\hat{\boldsymbol{y}}$ に固定した問題 $AFY(\hat{\boldsymbol{y}})$ は次のようになる．

$AFY(\hat{\boldsymbol{y}})$

$$\text{最小化}\quad \sum_{(i,j)\in A}\sum_{k\in K}(c_{ij}^k x_{ij}^k + c_{ji}^k x_{ji}^k) + \sum_{(i,j)\in A} f_{ij}\hat{y}_{ij} \tag{37}$$

$$\text{条件}\quad \sum_{i\in N_n} x_{in}^k - \sum_{j\in N_n} x_{nj}^k = \begin{cases} -1 & if\ n = O^k \\ 1 & if\ n = D^k \\ 0 & otherwise \end{cases} \quad \forall n \in N, k \in K \tag{38}$$

$$x_{ij}^k + x_{ji}^k \le \hat{y}_{ij} \quad \forall k \in K, (i,j) \in A \tag{39}$$

$$x_{ji}^k \le \hat{y}_{ij} \quad \forall k \in K, (i,j) \in A \tag{40}$$

$$x_{ij}^k \ge 0,\ x_{ji}^k \ge 0 \quad \forall k \in K, (i,j) \in A \tag{41}$$

目的関数の末項は定数項であるので，$AFY(\hat{\boldsymbol{y}})$ は品種 k 毎の独立した問題$AFY^k(\hat{\boldsymbol{y}})$に分解することができる．$AFY^k(\hat{\boldsymbol{y}})$は，$\hat{y}_{ij}=1$ であるアークにより構成されたネットワーク上において，アーク（i, j）の長さをc_{ij}^k，アーク（j, i）の長さをc_{ji}^kとした品種 k の始点・終点間の最短路問題となるため，Dijkstra法[3]などを用いて解くことができる．これらの問題を解けば，UNDのフロー変数の実行可能解$\hat{\boldsymbol{x}}$を求めることができる．

$\hat{\boldsymbol{y}}$はUNDの実行可能解であり，$\hat{\boldsymbol{y}}$を用いて実行可能なフロー変数が求まれば，これらはUNDの実行可能解となる．さらに，デザイン変数解を初期解としたヒューリスティック解法を適用すれば，より良い解を求めることができる．

4．2　限定したMinouxタイプのフォワード法

フォワード法は，初期解より，目的関数値が減少するようなアークを逐次加えていく貪欲法[3]である．しかし，問題の規模が大きくなると大きな計算時間を必要とする．このため，付加の対象となるアークを限定したMinouxタイプのリストを用いたフォワード法を提案する．

LGYは最小化問題であるため，LGYにおけるy_{ij}の係数$W_{ij}\le 0$であれば，$y_{ij}=1$が最適解となる．また，W_{ij}が小さい方が最適木に含まれる可能性が高くなる．そこで，W_{ij}を昇順に並べ換え，W_{ij}の小さい順にκ本を選択し，これらのアークのデザイン変数を付加するアークの候補とする．κを大きくすれば，最適解において1となるアークを近似解に含めることができる可能性が高くなる．このように，付加するアークを限定することにより，計算量を抑えることができる．

各アークに対して，Lagrange緩和解による初期解においてアークを付加したときの目的関数値の減少量を計算し，目的関数値の減少量とアークを目的関数値の減少量の降順にリストに格納しておく．なお，目的関数値の減少量はMurchland法[7]を用いると，効率的に求めることができる．

目的関数値の減少量の大きい順にアークの付加を検討する．アークを付加する対象となったときに，現在のネットワークにおいて，目的関数値の減少量を再計算する．この減少量が正でリスト内で最大であるときのみアークを付加する．この減少量が正でリスト内で最大でないときには，リストの適切な位置に目的関数値の減少量とアークを格納

する．

Minouxタイプのフォワード法

［ステップ1］Lagrange緩和解より，初期解を求める．付加対象基準κにより，付加対象のアーク候補を選定する．

［ステップ2］すべての付加対象アーク（i, j）について，このアークを付加した場合の目的関数値の減少量Δ_{ij}を計算する．

［ステップ3］$\Delta_{ij}>0$であるすべてのアーク（i, j）について，Δ_{ij}の降順に該当するアークとΔ_{ij}をリストに格納する．

［ステップ4］リスト内に$\Delta_{ij}>0$であるアーク（i, j）がなければ，終了する．そうでなければ，リストからΔ_{ij}が最大であるアークを取り出す．このアークを（p, q）とする．

［ステップ5］アーク（p, q）に対して，現在のネットワークにおけるΔ_{pq}を再計算する．

・$\Delta_{pq}>0$で，リスト内で最大であれば，ステップ6へ．

・$\Delta_{pq}>0$であれば，Δ_{pq}の値にしたがって，アーク（p, q）とΔ_{pq}をリスト内の適切な位置に挿入し，ステップ4へ戻る．

・$\Delta_{pq}\leq 0$であれば，ステップ4へ戻る．

［ステップ6］アーク（p, q）をネットワークに付加し，ステップ4へ戻る．

4．3　限定したMinoux法の改良法

Minoux法の改良法[3]は，目的関数値が減少するようなアークを削除していく方法である．ここでは，初期解に含むアークを限定した方法を用いる．

LGYにおけるy_{ij}の係数W_{ij}を昇順に並べ換え，係数の小さい順にκ本を選択し，これらのアークのデザイン変数を1とする．加えて，$\hat{y}_{ij}=1$であるアークのデザイン変数を1としたものを初期解とする．κを大きくすれば，最適解において1となるアークを初期解に含めることができる可能性が高くなる．

続いて，Minoux法の改良法を用いて，初期解を改善する．Minoux法の改良法は高速な解法であり，初期解に含まれるアーク数は高々$\kappa+|N-1|$本と限定されるため，短時間で改善した解を求めることができる．

ネットワークに含まれるアークに対して，アークを取り除いたときの目的関数値の減少量を計算し，目的関数値の減少量とアークを目的関数値の減少量の降順にリストに格納しておく．なお，目的関数値の減少量はMurchland法を用いると，効率的に求めることができる．

目的関数値の減少量の大きい順にアークの削除を検討する．アークが削除の対象と

なったときに，現在のネットワークにおいて，目的関数値の減少量を再計算する．この減少量が正でリスト内で最大であるときのみアークを取り除く．この減少量が正でリスト内で最大でないときには，リストの適切な位置に目的関数値の減少量とアークを格納する．

Minoux法の改良法

［ステップ1］Lagrange緩和解より，初期解を求める．対象基準κを満たすアークからなるネットワークを作成する．

［ステップ2］ネットワークに含まれているすべてのアーク(i, j)について，このアークを取り除いた場合の目的関数値の減少量Δ_{ij}を計算する．

［ステップ3］$\Delta_{ij} > 0$であるすべてのアーク(i, j)について，Δ_{ij}の降順に該当するアークとΔ_{ij}をリストに格納する．

［ステップ4］リスト内に$\Delta_{ij} > 0$であるアーク(i, j)がなければ，終了する．そうでなければ，リストからΔ_{ij}が最大であるアークを取り出す．このアークを(p, q)とする．

［ステップ5］アーク(p, q)に対して，現在のネットワークにおけるΔ_{pq}を再計算する．

・$\Delta_{pq} > 0$で，リスト内で最大であれば，ステップ6へ．

・$\Delta_{pq} > 0$であれば，Δ_{pq}の値にしたがって，アーク(p, q)とΔ_{pq}をリスト内の適切な位置に挿入し，ステップ4へ戻る．

・$\Delta_{pq} \leq 0$であれば，ステップ4へ戻る．

［ステップ6］アーク(p, q)をネットワークから取り除き，ステップ4へ戻る．

4．4　ω近傍局所探索法

前述の二つの近似解法によって，現在までの最良の上界値とψ%以内の差の上界値が求まった場合，アーク交換にもとずく局所探索法により，これらを初期解として解を改善する．

ここで用いる局所探索法は，現在の解に含まれていないアークを付加し，現在の解に含まれているアークを削除したときの目的関数値の変化量を計算し，目的関数値が最も減少するようなアークの付加と削除を同時に行うアーク交換を行い，この操作を目的関数値が減少する間，繰り返す方法である．

アーク交換の組合せ数は非常に多いため，付加を検討するアークは，削除するアークの端点の近傍にあるω個のノードを端点とするアークに限定する．このように限定しても，アーク交換の組合せ数は多い．そこで，アーク交換をアーク削除とアーク付加に分離し，アーク削除時の目的関数の変化量を計算し，続いてアーク付加時の目的関数の変

化量を計算し，その和をアーク交換時の目的関数の変化量とする．これにより，計算量を軽減することができる．アーク付加時および削除時の目的関数の変化量は，それぞれMurchland法を用いることができる．ただし，アーク削除時に連結性が失われる場合は，二つの部分グラフ間のカット上のアークを付加の対象として，アーク交換時の目的関数の変化量を計算する．

ω近傍局所探索法

［ステップ1］現在の解を用いたネットワークを作成する．

［ステップ2］各アークの端点の近傍にあるω個のノードを端点とするアークをそのアークに関する付加対象アークとする．

［ステップ3］ネットワークに含まれているすべてのアーク(i, j)について，このアークを取り除いた場合の目的関数値の変化量δ_{ij}^{d}を計算する．アークを取り除いたときに，ネットワークが非連結になればステップ5へ．

［ステップ4］アーク(i, j)に関するすべての付加対象のアーク(q, r)に対して，このアークを加えた場合の目的関数値の変化量δ_{qr}^{a}を計算する．$\Delta_{ij}^{qr} := \delta_{ij}^{d}+\delta_{qr}^{a}$とする．ステップ6へ．

［ステップ5］アーク(i, j)を取り除き，非連結となるネットワーク上のカットに含まれるアーク(q, r)を付加したときの目的関数値の変化量を計算し，Δ_{ij}^{qr}とする．

［ステップ6］Δ_{ij}^{qr}が最小となるアーク対(i^*, j^*)，(q^*, r^*)を求める．

［ステップ7］$\Delta_{i^*j^*}^{q^*r^*} \geq 0$であれば，終了する．

［ステップ8］ネットワークからアーク(i^*, j^*)を取り除き，アーク(q^*, r^*)を加える．ステップ3へ戻る．

4．5　Lagrangeヒューリスティック

前節までに示したLagrangeヒューリスティックをまとめると，次のようになる．

Lagrangeヒューリスティック

［ステップ1］アークの選択基準をκ，ωとし，局所探索法実施基準をψとする．$\bar{\boldsymbol{y}} := \boldsymbol{0}$とする．

［ステップ2］Lagrange緩和問題LGを解き，最適解$\hat{\boldsymbol{y}}$を求める．アーク(i, j)において，$\hat{y}_{ij}=1$であれば$\bar{y}_{ij}=1$とする．

［ステップ3］LGYにおけるy_{ij}の係数W_{ij}を昇順に並べ換える．W_{ij}の小さい順にκ本のアーク(i, j)を選択し，$\bar{y}_{ij}=1$とする．

［ステップ４］ $\hat{y}_{ij}=1$ であるアーク（i, j）で構成されるネットワークを初期解とする．$\hat{y}_{ij}=0$ かつ $\bar{y}_{ij}=1$ であるアークに対して限定したMinouxタイプのフォワード法を適用し，近似解を求める．

［ステップ５］ $\bar{\boldsymbol{y}}$ を初期解とした限定したMinoux法の改良法を適用する．

［ステップ６］ ステップ４またはステップ５で求められた上界値が，現在までの最良の上界値と ψ％以内であれば，これらの近似解を初期解とした ω 近傍局所探索法を行う．

［ステップ７］ ステップ４～６で求められた内の最良解を近似解とする．

5　C言語によるソースコード

全体のフローチャートを図１に示す．同一規模の複数の問題を解くため，初めに規模ごとの設定を行っている．problemは問題数であり，問題ごとにFor problemとNext problemの間を繰り返す．初めに問題ごとの設定を行っている．For iterationとNext iterationは劣勾配法の繰り返しである．劣勾配法の繰り返しの中で，被約費用の計算，最小木問題の求解，劣勾配の算出，限定したフォワード法，限定したMinouxの改良法，限定した局所探索法を行った後で，終了判定を行っている．続いて，Lagarange 乗数の更新，中間結果の表示を行っている．全問が終了と判定された場合に集計結果の表示を行い，計算を終了する．

次に，関数ごとに，関数名，入力，出力，呼び出し関数および概要を記述する．

int main（void） Lagrange 緩和法

入力　node,fixed_cost,fixed_lev,od_lev,arc_lev,od_level,arc_level

出力　x,upper_bound,lower_bound

呼び出し関数　read_data_1,clear_data_1,clear_data_2,read_data_2,vm_init,
adjacent_node,calc_ct,min_span_tree,calc_lower_bound,lower_bound_update,
calc_subgrad,rest_forward,rest_backward,adjacent_local_main,
etc,decide_exit,calc_v,inter_result_1,inter_result_2,result_main

Lagrange緩和法により，実行可能解，上界値および下界値を算出する．

void adjacent_local_main（void） ω 近傍局所探索法

入力　x

出力　x,upper_bound

呼び出し関数　dijk,calc_object,short_init,short_delete,connect,marge_node,
short_add,object_update,connect_node

現在の実行可能解に対して ω 近傍の局所探索を行い，上界値と実行可能解を算出

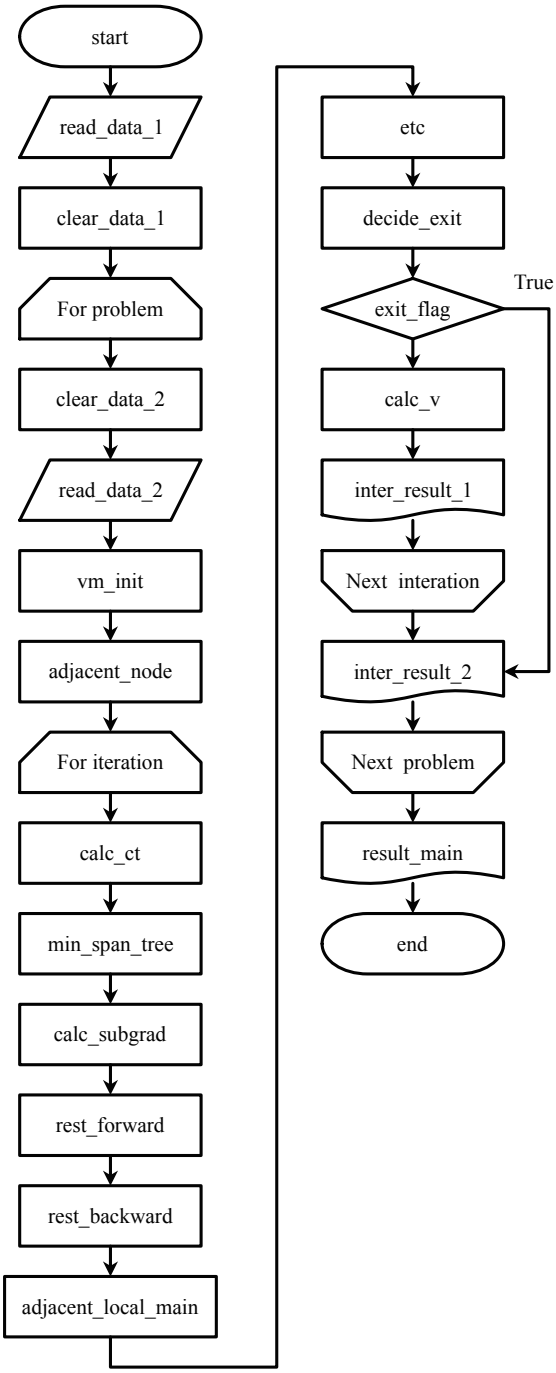
start
read_data_1
clear_data_1
For problem
clear_data_2
read_data_2
vm_init
adjacent_node
For iteration
calc_ct
min_span_tree
calc_subgrad
rest_forward
rest_backward
adjacent_local_main
etc
decide_exit
exit_flag
True
calc_v
inter_result_1
Next interation
inter_result_2
Next problem
result_main
end

図 1　フローチャート

する．

void adjacent_node（void） 近傍にあるノードリストの算出

入力　fixed_cost

出力　adj_node

各ノードについて，固定費用が昇順となるような隣接ノードリストを算出する．

void calc_ct（void） 被約費用の算出

入力　flow_cost,vm

出力　ct

各アークについて，Lagrange緩和問題におけるデザイン変数の係数である被約費用の算出する．

void calc_flow（void） アークフローの算出

入力　path

出力　flow

最短パスから，各アークのアークフローを算出する．

int calc_length（iis,iie） 最短距離の算出

入力　iis,iie,x,flow_cost

出力　length

フロー費用をアークの長さとして，２ノード間の最短距離を算出する．

float calc_lower_bound（void） 下界値と緩和解の算出

入力　vm,fixed_cost,ct

出力　　curr_lower_bound,x

Lagarange乗数から下界値と緩和解を算出する．

int calc_min_flow（i,j） 最小フローの算出

入力　i,j,path,flow

出力　minimun flow

パス上の最小フローを算出する．

float calc_object（distance） 上界値の算出

入力　distance,x,fixed_cost

出力　current_object

実行可能解と実行可能解におけるノード間距離から上界値を算出する．

void calc subgrad（void） 劣勾配の算出

入力　flow_cost,vm

出力　wn

フロー費用とLagarange乗数から，アークと品種に関する劣勾配を算出する．

void calc_v(void) Lagarange乗数の更新

入力　theta,vm,wn

出力　vm

劣勾配とLagarange乗数により，Lagarange乗数を更新する．

void clear_data_1(void) 初期化

全体で使用する集計のための変数の初期値を設定する．

void clear_data_2(void) 初期化

品種数，アーク数を算出し，被約費用を初期化する．

int connect(distance2) 連結性の判定

入力　distance2

出力　graph_connect

現在のグラフが連結しているか否かを判定する．

void connect_node(n1,n2,use_node,distance3) 連結性の判定の初期化

入力　use_node,distance3

出力　n1,n2,n1_num,n2_num

連結グラフに含まれるノード集合と，含まれないノード集合に分割する．

void decide_exit(void) 終了判定

入力　pres,_total_wa,lower_bound,upper_bound,eps

出力　flag

高精度の解が求められたき，劣勾配の二乗和が0，上界値=下界値の場合に，終了と判定する．

void dijk(distance) Dijkstra法

入力　x,flow_cost

出力　distance,path

フロー費用をアークの長さとして，Dijkstra 法により全ノード間の最短距離と最短パスを算出する．

void etc(void) 各種計算

入力　wn,alpha,alpha_param,lower_bound,upper_bound

出力　total_wn,alpha,diff,theta,pres

各種パラメータの更新，劣勾配の二乗和および精度の計算を行う．

float lower_bound_update(void) 下界値の更新

入力　lower_bound,current_lower_bound

出力　new_lower_bound

最良の下界値が得られた場合に下界値を更新する．

void init_upper_bound(void) 初期上界値の算出

入力　fixed_cost,flow_cost

出力　upper_bound,x

呼び出し関数　minoux_mod,dijk,calc_object,object_update,m_forw_mod

Minoux法の改良法と限定したフォワード法により，上界値と近似解を算出する．

void inter_result_1(void) 中間結果の表示

繰り返し中の中間の計算結果を表示する．

void inter_result_2(void) 中間結果の表示

１問ごとの計算結果を表示する．

void m_forw_init_1(x_i,x_j,addobj_k,xin) 限定したフォワード法の初期設定

入力　x,xin,fixed_cost,flow_cost

出力　x_i,x_j,addobj_k

呼び出し関数　calc_length,calc_min_flow

限定したフォワード法における修正アーク費用を昇順に算出する．

void m_forw_main(x_i,x_j,point,addobj_k) 限定したフォワード法

入力　x_i,x_j,addobj_k

出力　x,current_object

呼び出し関数　dijk,calc_object,object_update,short_init.short_add

修正アーク費用を用いて，限定したフォワード法を行い，上界値を算出する．

void m_forw_mod(xin) 限定したフォワード法

入力　x

出力　object,upper_bound

呼び出し関数　warshall,calc_object,object_update,m_forw_init_1,minoux_init_2,

m forw main

限定したフォワード法を行う．

void marge_node(adj_node_2,i,j,use_node) グラフの連結

入力　adj_node,i,j

出力　use_node,adj_node_2

ノード集合を連結する．

void min_span_tree(void) Prim法

入力　fixed_cost,ct

出力　x,total_length

呼び出し関数　min_span_tree_init,mst_main,min_span_tree_solution

Prim法により最小木を算出する．

void min_span_tree_init(start_node,end_node,length_ct) Prim法の初期設定

入力　fixed_cost,ct

出力　length_ct

修正アーク費用を求める.

void min_span_tree_solution(start_node,end_node,i_path) Prim法の解の算出

入力　start_node,end_node,i_path

出力　x

最小木の解を算出する.

void minoux_init_1(x_i,x_j,delobj_k) minouxの改良法の初期化

入力　flow,flow_cost,x

出力　x_i,x_j,delobj_k

呼び出し関数　calc_length

アーク削除時の目的関数の変化量を降順に算出する.

void minoux_init_2(point) minouxの改良法の初期化

入力　link

出力　point

変化量リストのポインタを初期化する.

void minoux_main(x_i,x_j,point,delobj_k) minouxの改良法

入力　point,x_i,x_j,delobj_k

出力　current_object

呼び出し関数　dijk,calc_object,object_update,short_init,short_delete

minouxの改良法により実行可能解と上界値を算出する.

void minoux_mod(void) minouxの改良法

入力　point,x_i,x_j,delobj_k

出力　current_object

呼び出し関数　warshall,calc_object,object_update,calc_flow,minoux_init_1,minoux_init_2,minoux_main,dijk

minouxの改良法を実行する.

void mst_main(start_node,end_node,i_path,float_length_ct) 最小木の解の算出

入力　length_ct,start_node,end_node

出力　i_path,total_length

最小木および最小木の重みの合計を算出する.

float object_update(void) 上界値の更新

入力　object,upper_bound

出力　new_upper_bound

最良の下界値が得られた場合に下界値を更新する.

void read_data_1(void) データの読み込み

問題の属性，問題数，固定費用比率，ノード数を読み込む.

void read_data_2(void) データの読み込み

フロー費用を読み込む，固定費用などを設定する.

void rest_backward(void) 限定したフォワード法

入力　x,ct,fixed_cost

出力　current_object

呼び出し関数　warshall,calc_object,uobject_update,mioux_mod,dijk

限定したフォワード法を実行する.

void rest_forward(void) 限定したフォワード法の初期設定

入力　x,ct,fixed_cost

出力　current_object,xin

呼び出し関数　m_forw_mod,dijk,calc_object,uobject_update

限定したフォワード法の初期設定を行う.

void result_main(void) 結果の表示

全体の計算結果の集計を表示する.

void short_add(add_i,add_j,distance2) アーク付加時の最短距離の算出

入力　add_i,add_j

出力　distance2

アークを付加したとき全ノード間の最短距離を算出する.

void short_delete(del_i,del_j,distance2) アーク削除時の最短距離の算出

入力　del_i,del_jdistance2,flow_cost

出力　distance2

アークを削除したとき全ノード間の最短距離を算出する.

void short_init(distance,distance2) 最短距離の保存

入力　distance

出力　distance2

最短距離を一時保存する.

void vm_init(void) Lagrange乗数の初期化

入力　flow_cost

出力　vm

フロー費用を用いた最短路問題を解くことによって，Lagrange 乗数の初期値を設定する.

void warshall（distance）floyd-warshall 法

入力　flow_cost,x

出力　distance,path

Floyd-Warshall 法により，現在の解における全ノード間の最短距離を算出する．

図２に，Ｃ言語によるソースコードを記述する．ページ数の制限から，一行に複数の文を記述している場合があり，またインデントが１文字となっているが，実際のソースコードでは一行は一文，インデントは４文字としている．

ソースコードで使用する主な配列，変数および定数と，定式化，解法における変数およびパラメータの関係を示す．

problem_num：問題数

node：ノード数 $|N|$

arc_num：アーク数 $|A|$

od_num：品種数 $|K|$

flow：フロー費用 $\boldsymbol{c}$

fixed：デザイン費用 $\boldsymbol{f}$

flow：フロー解 $\boldsymbol{x}$

x：デザイン解 $\boldsymbol{y}$

vm：Lagrange 乗数 $\boldsymbol{v}$

wn：劣勾配 $\boldsymbol{s}$

ct：修正アーク費用 W

ite_limit：繰り返し回数の上限 l_{max}

eps：収束判定基準 ϵ

alpha：劣勾配のパラメータ ρ

alpha_param：ρ の変更率 l_{λ}

param_int：ρ の変更周期 l_{ρ}

rest_param：アーク選択基準 κ

adj_param：アーク選択基準 ω

local_param：局所探索法実施基準 ψ

図３および図４に，ノード数が10である問題の２つデータファイルを記載する．param.dat には，デザイン費用とフロー費用の比率，アーク基準の上限，品種基準の上限が含まれている．und.datには，問題数，ノード数，フロー費用，アーク基準，品種基準およびノードのxy座標が含まれている．アーク基準の上限が５であればアーク基準が５以下であるアークをアーク集合とし，品集基準の上限が５であれば品集基準が５

以下であるノード対間を品種集合としている．

6　おわりに

本研究では，*UND*の定式化，Lagrange緩和法およびLagrangeヒューリスティックを示し，これらの解法のためのC言語によるソースコードを示した．ソースコードは，十分に整理されたものとは言えないかもしれないが，プログラム開発において参考になれば幸いである．なお，計算結果については，片山の研究[4]に詳細が記載されている，

参考文献

[1] A. Balakrishnan, T. L. Magnanti and P. Mirchandani. Network design. In M. Dell' Amico, F. Maffioli and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pp. 311–334. John Wiley & Sons, New York, 1997.

[2] 片山直登. ネットワークデザイン問題. 久保幹雄, 田村明久, 松井知己（編）, 応用数理計画ハンドブック, pp. 1184–1282. 朝倉書店, 2002.

[3] 片山直登. ネットワーク設計問題. 朝倉書店, 2008.

[4] 片山直登. 多品種を考慮したロジスティクスネットワーク設計問題の数理的解法に関する研究. 博士論文, 流通経済大学, 2010.

[5] 久保幹雄, 田村明久, 松井知己. 応用数理計画ハンドブック. 朝倉書店, 2002.

[6] T. L. Magnanti and R. T. Wong. Network design and transportation planning: Models and algorithms. *Transportation Science*, Vol. 18, pp. 1–55, 1984.

[7] P. A. Steenbrink. *Optimization of Transport Networks*. John Wiley & Sons, New York, 1974.

[8] R. T. Wong. Introduction and recent advances in network design: Models and algorithms. In M. Florian, editor, *Transportation Planning Models*, pp. 187–225. Elsevier Science, North Holland, Amsterdam, 1984.

図２　C言語のソースコード

```
//*********************************************
//  容量制約のないネットワーク設計問題
//  Uncapacitated Netowrk Design Problem
//  Lagrange緩和法
//*********************************************
#include <stdio.h>
#include <time.h>
const int n_num=101,icomm_num=9901,link_num=9901;
const int ite_limit=10000,inf=999999,param_int=100;
const int adj_param=10;
const float big=99999999.9,alpha_param=0.98,local_param=1.002;
const float rest_param=4.0,eps=0.01;
int arc_lev[n_num][n_num],od_lev[n_num][n_num],
int flow_cost[n_num][n_num], fixed_cost[n_num][n_num];
int x[n_num][n_num],wn[n_num][icomm_num];
int adj_node[n_num][n_num],path[n_num][n_num];
int flow[n_num][n_num];
int od_level,arc_level,fixed_lev,arc_num,od_num,ex_arc,link;
int prob_num,node,total_iteration,comm;
int total_wn,exit_flag,problem,ite,n_adj_node,n_adj_node_org;
int length_ij,min_flow,n1_num,n2_num,g_connect,c_point;
float ct[n_num][n_num],vm[n_num][icomm_num];
float total_lower_bound,total_upper_bound,object;
float upper_bound, lower_bound,total_vm,current_lower_bound;
float alpha,pres,theta,current_object,old_upper_bound;
clock_t start_time,end_time;
FILE *INP1,*INP2,*OUTP1,*OUTP2;
static void adjacent_local_main(void);
static void adjacent_node(void);
static void calc_ct(void);
static void calc_flow(void);
static int calc_length(int iis,int iie);
static float calc_lower_bound(void);
static float lower_bound_update(void);
static int calc_min_flow(int i,int j);
static float calc_object(int distance[][n_num]);
static void calc_subgrad(void);
static void calc_v(void);
static void clear_data_1(void);
```

```
static void clear_data_2(void);
static int connect(int distance2[][n_num]);
static void connect_node(int n1[],int n2[],int use_node[],int
distance3[][n_num]);
static void dijk(int distance[][n_num]);
static void etc(void);
static void init_upper_bound(void);
static void inter_result_1(void);
static void inter_result_2(void);
static void m_forw_init_1(int x_i[],int x_j[],int
addobj_k[],int xin[][n_num]);
static void m_forw_main(int x_i[],int x_j[],int point[],int
addobj_k[]);
static void m_forw_mod(int xin[][n_num]);
static void marge_node(int adj_node_2[],int i,int j,int
use_node[]);
static void min_span_tree(void);
static void min_span_tree_init(int start_node[],int
end_node[],float length_ct[]);
static void min_span_tree_solution(int start_node[],int
end_node[],int i_path[]);
static void minoux_init_1(int x_i[],int x_j[],int delobj_k[]);
static void minoux_init_2(int point[]);
static void minoux_main(int x_i[],int x_j[],int point[],int
delobj_k[]);
static void minoux_mod(void);
static void mst_main(int start_node[],int end_node[],int
i_path[],float length_ct[]);
static float object_update(void);
static void read_data_1(void);
static void read_data_2(void);
static void rest_backward(void);
static void rest_forward(void);
static void result_main(void);
static void short_add(int add_i,int add_j,int
distance2[][n_num]);
static void short_delete(int del_i,int del_j,int distance2[][n
_num]);
static void short_init(int distance[][n_num],int distance2[][n
_num]);
```

```
static void vm_init(void);
static void warshall(int distance[][n_num]);
static int decide_exit(void);
//*********************************************
int main(){
 read_data_1(); clear_data_1();
 for(problem=1;problem<=prob_num;problem++){
  start_time=clock();
  clear_data_2(); read_data_2(); vm_init();
  adjacent_node();
  n_adj_node=adj_param;
  if(node<=n_adj_node) n_adj_node=node-1;
  for(ite=1;ite<=ite_limit;ite++){
   old_upper_bound=upper_bound;
   calc_ct(); min_span_tree();
   current_lower_bound=calc_lower_bound();
   lower_bound=lower_bound_update();
   calc_subgrad(); rest_forward(); rest_backward();
   if(current_object<upper_bound*local_param)
    adjacent_local_main();
   etc(); exit_flag=decide_exit();
   if(exit_flag==1) break;
   calc_v(); inter_result_1();
  }//for
  end_time=clock(); inter_result_2();
 }//for
 end_time=clock(); result_main();
 fclose(INP1); fclose(INP2)   fclose(OUTP1); fclose(OUTP2);
 return(0);
}// main
//*********************************************
void read_data_1(void){
 long ndummy;
 fopen_s(&INP1,"und.dat","r");
 fopen_s(&INP2,"param.dat","r");
 fscanf_s(INP2,"%ld",&ndummy);
 fscanf_s(INP2,"%5d",&fixed_lev);
 fscanf_s(INP2,"%d",&arc_level);
 fscanf_s(INP2,"%d",&od_level);
 fscanf_s(INP1,"%5d",&prob_num);
```

```
 fscanf_s(INP1,"%5d",&node);
 printf("ndummy=%ld¥n",ndummy);
 fopen_s(&OUTP2,"undlagout2.out","a");
}// read_data_1
//***********************************************
void clear_data_1(void){
 total_lower_bound=0.0; total_upper_bound=0.0;
 total_iteration=0;
 fopen_s(&OUTP1,"undlagout.out","a");
}// clear_data_1
//***********************************************
void clear_data_2(void){
 int i,j;
 upper_bound=big*2.0; lower_bound=-big*2.0;
 alpha=1.0; comm=0;
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   if(i!=j&&od_lev[i][j]<=od_level){
    comm=comm+1;
   }//if
  }//for
 }//for
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   ct[i][j]=0.0; ct[j][i]=0.0;
  }//for
 }//for
 ex_arc=0;
}// clear_data_2
//***********************************************
void read_data_2(void){
 int i,j,dummy1,dummy2;
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   fscanf_s(INP1,"%5d",&flow_cost[i][j]);
  }//for
 }//for
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   fscanf_s(INP1,"%5d",&arc_lev[i][j]);
  }//for
 }//for
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   fscanf_s(INP1,"%5d",&od_lev[i][j]);
  }//for
 }//for
 for(i=1;i<=node;i++){
  fscanf_s(INP1,"%5d%5d",&dummy1,&dummy2);
 }//for
 for(i=1;i<=node;i++){
  flow_cost[i][i]=0;
 }//for
 arc_num=0;
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   if(i!=j&&arc_lev[i][j]<=arc_level){
    fixed_cost[i][j]=flow_cost[i][j]*fixed_lev;
    x[i][j]=1;
   }else{
    fixed_cost[i][j]=inf; flow_cost[i][j]=inf; x[i][j]=0;
   }//if
   if(i<j&&arc_lev[i][j]<=arc_level){
    arc_num=arc_num+1;
   }//if
  }//for
 }//for
 od_num=0;
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   if(i!=j&&od_lev[i][j]<=od_level){
    od_num=od_num+1;
   }//if
  }//for
 }//for
}// read_data_2
//***********************************************
void vm_init(void){
 int distance[n_num][n_num];
 int i,j,l,newl,m;
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   path[i][j]=i; distance[i][j]=flow_cost[i][j];
   if(i==j){
    distance[i][j]=0;
   }//if
  }//for
 }//for
 for(l=1;l<=node;l++){
  for(i=1;i<=node;i++){
   for(j=1;j<=node;j++){
    newl=distance[i][l]+distance[l][j];
    if(distance[i][j]>newl){
     distance[i][j]=newl; path[i][j]=path[l][j];
    }//if
   }//for
  }//for
 }//for
 m=0;
 for(l=1;l<=node;l++){
  for(j=1;j<=node;j++){
   if(l!=j&&od_lev[l][j]<=od_level){
    m=m+1;
    for(i=1;i<=node;i++){
     vm[i][m]=distance[l][i];
    }//for
   }//if
  }//for
 }//for
}// vm_init
//***********************************************
int decide_exit(void){
 int flag;
 flag=0;
 if(pres>=eps)  flag=1;
 if(total_wn==0) flag=1;
 if(lower_bound==upper_bound) flag=1;
 return(flag);
}// decide_exit
//***********************************************
```

```
void calc_ct(void){
 int i,j,m,k,l;
 float s1,s2,cbijm,cbjim;
 for(i=1;i<=node;i++){
  for(j=i+1;j<=node;j++){
   if(fixed_cost[i][j]<inf){
    m=0; ct[i][j]=0.0;
    for(l=1;l<=node;l++){
     s1=0.0; s2=0.0;
     for(k=1;k<=node;k++){
      if(l!=k&&od_lev[l][k]<=od_level){
       m=m+1;
       cbijm=(float)flow_cost[i][j]-(vm[j][m]-vm[i][m]);
       if(cbijm<0.0){
        s1=s1+cbijm;
       }else{
        cbjim=(float)flow_cost[j][i]-(vm[i][m]-vm[j][m]);
        if(cbjim<0.0){
         s2=s2+cbjim;
        }//if
       }//if
      }//for
      if(s1<=s2){
       ct[i][j]=ct[i][j]+s1;
      }else{
       ct[i][j]=ct[i][j]+s2;
      }//if
     }//for
    }//if
   }//for
  }//for
}// calc_ct
//***********************************************
void calc_subgrad(void){
 int i,j,m,k,l,m1,m2;
 float p1,p2,cbijm,cbjim;
 m=0;
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   if(i!=j&&od_lev[i][j]<=od_level){
    m=m+1;
    for(l=1;l<=node;l++){
     wn[l][m]=0;
    }//for
    wn[i][m]=-1; wn[j][m]=1;
   }//if
  }//for
 }//for
 for(i=1;i<=node;i++){
  for(j=i+1;j<=node;j++){
   if(fixed_cost[i][j]<inf){
    m=0;
    for(l=1;l<=node;l++){
     p1=0.0; p2=0.0; m1=m; m2=m;
     for(k=1;k<=node;k++){
      if(l!=k&&od_lev[l][k]<=od_level){
       m=m+1;
       if(x[i][j]==1){
        cbijm=(float)flow_cost[i][j]-(vm[j][m]-vm[i][m]);
        if(cbijm<0.0){
         p1=p1+cbijm;
        }else{
        cbjim=(float)flow_cost[j][i]-(vm[i][m]-vm[j][m]);
         if(cbjim<0.0){
          p2=p2+cbjim;
         }//if
        }//if
       }//if
      }//for
      if(x[i][j]==1){
       if(p1<=p2&&p1<0.0){
        for(k=1;k<=node;k++){
         if(l!=k&&od_lev[l][k]<=od_level){
          m1=m1+1;
          cbijm=(float)flow_cost[i][j]-(vm[j][m1]-vm[i][m1]);
          if(cbijm<0.0){
           wn[i][m1]=wn[i][m1]+1; wn[j][m1]=wn[j][m1]-1;
          }//if
         }//if
        }//for
       }else{
        if(p1>p2&&p2<0.0){
         for(k=1;k<=node;k++){
          if(l!=k&&od_lev[l][k]<=od_level){
           m2=m2+1;
           cbjim=(float)flow_cost[j][i]-(vm[i][m2]-vm[j][m2]);
           if(cbjim<0.0){
            wn[i][m2]=wn[i][m2]-1; wn[j][m2]=wn[j][m2]+1;
           }//if
          }//if
         }//for
        }//if
       }//if
      }//for
     }//if
    }//for
   }//if
  }//for
 }// calc_subgrad
 //***********************************************
 float calc_lower_bound(void){
  int m=0,i,j;
  float total_ct,curr_lower_bound;
  total_vm=0.0;
  for(i=1;i<=node;i++){
   for(j=1;j<=node;j++){
    if(i!=j&&od_lev[i][j]<=od_level){
     m=m+1;
     total_vm=total_vm-vm[i][m]; total_vm=total_vm+vm[j][m];
    }//if
   }//for
  }//for
  for(i=1;i<=node;i++){
   for(j=i+1;j<=node;j++){
    if(fixed_cost[i][j]+ct[i][j]<0.0){
     x[i][j]=1; x[j][i]=1;
    }//if
   }//for
```

```
  }//for
  total_ct=0.0;
  for(i=1;i<=node;i++){
   for(j=i+1;j<=node;j++){
    total_ct=total_ct+ct[i][j]*x[i][j]+fixed_cost[i][j]*x[i][j]
;
   }//for
  }//for
  curr_lower_bound=total_vm+total_ct+.99;
  return(curr_lower_bound);
}// calc_lower_bound
//*********************************************
float lower_bound_update(void){
  float new_lower_bound;
  if(lower_bound<current_lower_bound){
   new_lower_bound=current_lower_bound;
  }else{
   new_lower_bound=lower_bound;
  }//if
  return(new_lower_bound);
}// olower_bound_update
//*********************************************
void etc(void){
  int m;
  float diff,tw;
  total_wn=0;
  for(m=1;m<=comm;m++){
   for(i=1;i<=node;i++){
    total_wn=total_wn+wn[i][m]*wn[i][m];
   }//for
  }//for
  if(ite% param_int ==0) alpha=alpha*alpha_param;
  if(total_wn==0) total_wn=1;
  diff=upper_bound-current_lower_bound; tw=total_wn;
  theta=alpha*(upper_bound-current_lower_bound)/tw;
  pres=100.0*( upper_bound -lower_bound)/ lower_bound;
}// etc
//*********************************************
void calc_v(void){
  int m,i;
  for(m=1;m<=comm;m++){
   for(i=1;i<=node;i++){
    vm[i][m]=vm[i][m]+theta*wn[i][m];
   }//for
  }//for
}// calc_v
//*********************************************
void inter_result_1(void){
  if(ite%200==0){
   printf("  %3d p=%3d i=%5d ub=%7.0f lb=%7.0f c_lb=%7.0f
 err=%6.2f¥n",node,problem,ite,upper_bound,lower_bound,curren
 t_lower_bound,  100.0*(upper_bound-lower_bound)/lower_bound);
  }//if
}// inter_result_1
//*********************************************
void inter_result_2(void){
  printf(" n=%5d p=%5d ub=%7.0f lb=%7.0f err=%6.2f ite=%6dex_ar
 c=%5d¥n",node,problem,upper_bound,lower_bound,100.0*(upper_bo
 und-lower_bound)/lower_bound,ite,ex_arc);
  fprintf(OUTP2," node=%4d fixed-lev=%2d arc=%4d od=%4d ub=%10.
 1f lb=%10.1f err=%6.2f time=%15.2f¥n",
 node,fixed_lev,arc_num,od_num,upper_bound,lower_bound,100.0*
  (upper_bound-lower_bound)/lower_bound,  (end_time-start_time)
  /(float)CLOCKS_PER_SEC);
  total_lower_bound=total_lower_bound+lower_bound;
  total_upper_bound=total_upper_bound+upper_bound;
  total_iteration=total_iteration+ite-1;
}// inter_result_2
//*********************************************
void result_main(void){
  printf(" n=%4d total_ub=%8.0f total_lb=%8.0f err=%6.2f ite=%6
 .1f¥n",  node,total_upper_bound,total_lower_bound,100.0*(total_
 upper_bound-total_lower_bound)/total_lower_bound,total_itera
 tion/(float)prob_num);
  fprintf(OUTP1," node=%4d fixed-lev=%2d arc=%4d od=%4d ub=%10
 .1f lb=%10.1f err=%6.2f¥n",node,fixed_lev,arc_num,od_num,total
 _upper_bound,total_lower_bound,100.0*(total_upper_bound-total
 _lower_bound)/total_lower_bound);
}// result_main
//*********************************************
void min_span_tree(void){
  int end_node[link_num],start_node[link_num],i_path[n_num];
  float length_ct[link_num];
  min_span_tree_init(start_node,end_node,length_ct);
  mst_main(start_node,end_node,i_path,length_ct);
  min_span_tree_solution(start_node,end_node,i_path);
 }// min_span_tree
//*********************************************
void min_span_tree_init(int start_node[],int end_node[],float
length_ct[]){
  int k=0,i,j;
  for(i=1;i<=node;i++){
   for(j=i+1;j<=node;j++){
    k=k+1; end_node[k]=0; start_node[k]=0; length_ct[k]=0;
   }//for
  }//for
  link=0;
  for(i=1;i<=node;i++){
   for(j=i+1;j<=node;j++){
    if(fixed_cost[i][j]>=inf){
     fixed_cost[i][j]=inf; fixed_cost[j][i]=inf;
    }else{
     if(i!=j){
      link=link+1; end_node[link]=i; start_node[link]=j;
      length_ct[link]=fixed_cost[i][j]+ct[i][j];
     }//if
    }//if
   }//for
  }//for
}// min_span_tree_init
//*********************************************
void mst_main(int start_node[],int end_node[],int
i_path[],float length_ct[]){
  int k_end_node[n_num],node_label[n_num];
  int m_end_node[link_num],m_start_node[link_num];
  int s[link_num],t[link_num],scan_node[link_num];
  float v[link_num];
  int n,i,j,k,kk,l,scanned_node,s_node;
  float min_v,total_length;
  total_length=0.0; n=0; min_v=0.0;
```

```
for(i=1;i<=node;i++){
 k_end_node[i]=0; node_label[i]=0; i_path[i]=0;
 for(j=i+1;j<=node;j++){
  n=n+1; s[n]=0; m_end_node[n]=0; m_start_node[n]=0;
  t[n]=0; v[n]=0.0; scan_node[n]=0;
 }//for
}//for
for(k=1;k<=link;k++){
 if(k_end_node[end_node[k]]<=0){
  k_end_node[end_node[k]]=k; s[end_node[k]]=k;
 }else{
  kk=s[end_node[k]]; m_end_node[kk]=k; s[end_node[k]]=k;
 }//if
 if(node_label[start_node[k]]<=0){
  node_label[start_node[k]]=k; t[start_node[k]]=k;
 }else{
  kk=t[start_node[k]]; m_start_node[kk]=k; [start_node[k]]=k;
 }//if
}//for
for(i=1;i<=node;i++){
 v[i]=big; s[i]=i; t[i]=i;
}//for
s_node=1; i=s_node; v[i]=0; s[i]=0; scanned_node=1;
while(1){
 k=k_end_node[i];
 if(k!=0){
  do{
   if(scan_node[k]==0){
    scan_node[k]=1; j=start_node[k];
    if(v[j]>length_ct[k]){
     v[j]=length_ct[k]; i_path[j]=k;
    }//if
   }//if
   k=m_end_node[k];
  }while(k>0);
 }//if
 k=node_label[i];
 if(k!=0){
  do{
   if(scan_node[k]==0){
    scan_node[k]=1; j=end_node[k];
    if(v[j]>length_ct[k]){
     v[j]=length_ct[k]; i_path[j]=k;
    }//if
   }//if
   k=m_start_node[k];
  }while(k>0);
 }//if
 if(scanned_node==node-1)break;
 min_v=big;
 for(l=scanned_node+1;l<=node;l++){
  j=t[l];
  if(v[j]<min_v){
   min_v=v[j]; i=j;
  }//if
 }//for
 scanned_node=scanned_node+1;
 l=s[i]; j=t[scanned_node]; s[j]=l; t[l]=j; s[i]=0;
}//while
for(i=1;i<=node;i++){
 if(i!=s_node){
  total_length=total_length+length_ct[i_path[i]];
 }//if
}//for
}// mst_main
//*********************************************
void min_span_tree_solution(int start_node[],int
end_node[],int i_path[]){
 int i,j,k;
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   x[i][j]=0;
  }//for
 }//for
 for(i=1;i<=node;i++){
  if(i!=1){
   k=i_path[i];
   if(end_node[k]>0&&start_node[k]>0){
    x[start_node[k]][end_node[k]]=1;
    x[end_node[k]][start_node[k]]=1;
   }//if
  }//if
 }//for
}// min_span_tree_solution
//*********************************************
float calc_object(int distance[][n_num]){
 int i,j;
 float current_object; current_object=0;
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   if(i!=j){
    if(od_lev[i][j]<=od_level){
     current_object=current_object+distance[i][j];
    }//if
   }//if
  }//for
 }//for
 for(i=1;i<=node;i++){
  for(j=i+1;j<=node;j++){
   current_object=current_object+fixed_cost[i][j]*x[i][j];
  }//for
 }//for
 return(current_object);
}// calc_object
//*********************************************
float object_update(void){
 float new_upper_bound;
 if(object<upper_bound){
  new_upper_bound=object;
}else{
  new_upper_bound=upper_bound;
 }//if
 return(new_upper_bound);
}// object_update
//*********************************************
void init_upper_bound(void){
 int distance[n_num][n_num];
 int xin[n_num][n_num];
 int i,j;
 for(i=1;i<=node;i++){
```

```
    for(j=1;j<=node;j++){
     x[i][j]=0;
     if(fixed_cost[i][j]<inf){
      if(i!=j){
       x[i][j]=1;
      }//if
     }//if
    }//for
   }//for
   minoux_mod();
   dijk(distance);
   object=calc_object(distance);
   upper_bound=object_update();
   for(i=1;i<=node;i++){
    for(j=i+1;j<=node;j++){
     if(fixed_cost[i][j]<inf){
      xin[i][j]=1; xin[j][i]=1;
     }else{
      xin[i][j]=0; xin[j][i]=0;
     }//if
    }//for
   }//for
   m_forw_mod(xin); dijk(distance);
   object=calc_object(distance); upper_bound=object_update();
  }// init_upper_bound
  //*********************************************
  void minoux_mod(void){
   int delobj_k[link_num],x_i[link_num],x_j[link_num];
   int point[link_num];
   int distance[n_num][n_num];
   warshall(distance);
   object=calc_object(distance);  upper_bound=object_update();
   if(object<big){
    calc_flow(); minoux_init_1(x_i,x_j,delobj_k);
    minoux_init_2(point); minoux_main(x_i,x_j,point,delobj_k);
    dijk(distance);
    object=calc_object(distance); upper_bound=object_update();
   }//if
  }// minoux_mod
  //*********************************************
```

```
  void minoux_init_1(int x_i[],int x_j[],int delobj_k[]){
   int i,j,l;
   delobj_k[0]=inf;
   link=0;
   for(i=1;i<=node;i++){
    for(j=i+1;j<=node;j++){
     if(x[i][j]==1){
      link=link+1; x[i][j]=0; x[j][i]=0;
      length_ij=calc_length(i,j);
      x[i][j]=1; x[j][i]=1; x_i[link]=i; x_j[link]=j;
      delobj_k[link]=flow[i][j]*length_ij-(flow[i][j]*flow_cost
      [i][j]+fixed_cost[i][j]);
     }//if
    }//for
   }//for
   for(i=1;i<=link-1;i++){
    for(j=i+1;j<=link;j++){
     if(delobj_k[i]>delobj_k[j]){
      l=delobj_k[i]; delobj_k[i]=delobj_k[j]; delobj_k[j]=l;
      l=x_i[i]; x_i[i]=x_i[j]; x_i[j]=l;
      l=x_j[i]; x_j[i]=x_j[j]; x_j[j]=l;
     }//if
    }//for
   }//for
  }// minoux_init_1
  //*********************************************
  void minoux_init_2(int point[]){
   int i;
   point[0]=1;
   for(i=1;i<=link-1;i++){
    point[i]=i+1;
   }//for
   point[link]=0;
   c_point=2;
  }// minoux_init_2
  //*********************************************
  void minoux_main(int x_i[],int x_j[],int point[],int delobj_k
  []){
   int distance[n_num][n_num];
   int distance2[n_num][n_num];
```

```
   int del_link,next_point,i,j;
   float del_object;
   dijk(distance); object=calc_object(distance);
   short_init(distance,distance2); current_object=object;
   do{
    del_link=point[0]; i [del_link];
    x[i][j]=0; x[j][i]=0;
    short_init(distance,distance2); short_delete(i,j,distance2);
    object=calc_object(distance2); upper_bound=object_update();
    short_init(distance,distance2);
    x[i][j]=1; x[j][i]=1; del_object=object-current_object;
    if(del_object>=0){
     point[0]=point[del_link];
     point[del_link]=0;
    }else{
     if(del_object<=delobj_k[point[del_link]]){
      x[i][j]=0; x[j][i]=0;
      dijk(distance);
      object=calc_object(distance); upper_bound=object_update();
      current_object=object;
      point[0]=point[del_link]; point[del_link]=0;
     }else{
      c_point=del_link; next_point=point[c_point];
      while(1){
       if(del_object<=delobj_k[next_point]){
        point[0]=point[del_link]; point[c_point]=del_link;
        point[del_link]=next_point;
        delobj_k[del_link]=del_object;
        break;
       }//if
       c_point=next_point; next_point=point[c_point];
      }//while
     }//if
    }//if
   }while(delobj_k[c_point]<=0&&point[0]!=0&&current_object
    <big);
  }// minoux_main
  //*********************************************
  void short_init(int distance[][n_num],int distance2[][n_num]){
   int i,j;
```

```
  for(i=1;i<=node;i++){
   for(j=1;j<=node;j++){
    distance2[i][j]=distance[i][j];
   }//for
  }//for
}// short_init
//*********************************************
void calc_flow(void){
  int i,j,k,l;
  for(i=1;i<=node;i++){
   for(j=1;j<=node;j++){
    flow[i][j]=0;
   }//for
  }//for
  for(i=1;i<=node;i++){
   for(j=1;j<=node;j++){
    if(i!=j&&od_lev[i][j]<=od_level){
     if(i!=j){
      l=j;
      do{
       k=calc¥_length [i][l];
       if(k!=0){
        flow[k][l]=flow[k][l]+1; l=k;
        if(i==l){
         break;
        }//if
       }//if
      }while(k!=0);
     }//if
    }//if
   }//for
  }//for
  for(i=1;i<=node;i++){
   for(j=i+1;j<=node;j++){
    flow[i][j]=flow[i][j]+flow[j][i]; flow[j][i]=0;
   }//for
  }//for
}// calc_flow
//*********************************************
int calc_length(int iis,int iie){
  int arc_length[link_num],start_node[link_num],k_end_node[link
_num];
  int v[link_num],m_end_node[link_num],t[link_num],ip[n_num];
  int i,j,n,k,min_v,l,scanned_node,length;
  n=0; m_end_node[0]=0;
  for(i=1;i<=node;i++){
   for(j=1;j<=node;j++){
    if(i!=j){
     if(x[i][j]==1||x[j][i]==1){
      n=n+1; start_node[n]=j; arc_length[n]=flow_cost[i][j];
     }//if
    }//if
   }//for
   m_end_node[i]=n;
  }//for
  for(i=1;i<=node;i++){
   ip[i]=0;
  }//for
  if(iis!=0){
   for(i=1;i<=node;i++){
    v[i]=inf; k_end_node[i]=i; t[i]=i;
   }//for
   k_end_node[1]=iis; t[iis]=1; i=iis;
   v[i]=0; k_end_node[i]=0; scanned_node=1;
   while(1){
    l=m_end_node[i-1]+1;
    if(l<=m_end_node[i]){
     for(k=l;k<=m_end_node[i];k++){
      j=start_node[k];
      if(k_end_node[j]!=0){
       if(v[j]>v[i]+arc_length[k]){
        v[j]=v[i]+arc_length[k]; ip[j]=i;
       }//if
      }//if
     }//for
    }//if
    if(j>node){
     v[iie]=inf;
     break;
    }//if
    if(ip[j]==iie)break;
    if(scanned_node==node-1) break;
    min_v=inf;
    for(l=scanned_node+1;l<=node;l++){
     j=t[l];
     if(min_v>v[j]){
      min_v=v[j]; i=j;
     }//if
    }//for
    if(k_end_node[i]==0){
     v[iie]=inf;
     break;
    }//if
    scanned_node=scanned_node+1;
    l=k_end_node[i]; j=t[scanned_node];
    k_end_node[j]=l; t[l]=j; k_end_node[i]=0;
   }//while
  }//if
  length=v[iie];
  return(length);
}// calc_length
//*********************************************
void dijk(int distance[][n_num]){
  int arc_length[link_num],start_node[link_num];
  int k_end_node[n_num],v[n_num],m_end_node[n_num];
  int t[link_num],i_path[n_num];
  int n=0,k,i,j,iis,l,scanned_node,min_v;
m_end_node[0]=n;
  for(i=1;i<=node;i++){
   for(j=1;j<=node;j++){
    if(i!=j){
     if(x[i][j]==1||x[j][i]==1){
      n=n+1; start_node[n]=j; arc_length[n]=flow_cost[i][j];
     }//if
    }//if
   }//for
   m_end_node[i]=n;
  }//for
  for(iis=1;iis<=node;iis++){
   for(i=1;i<=node;i++){
```

```
    i_path[i]=0;
   }//for
   if(iis!=0){
    for(i=1;i<=node;i++){
     v[i]=inf; k_end_node[i]=i; t[i]=i;
    }//for
    k_end_node[1]=iis; t[iis]=1; i=iis;
    v[i]=0; k_end_node[i]=0; scanned_node=1;
    while(1){
     l=m_end_node[i-1]+1;
     if(l<=m_end_node[i]){
      for(k=l;k<=m_end_node[i];k++){
       j=start_node[k];
       if(k_end_node[j]!=0){
        if(v[j]>v[i]+arc_length[k]){
         v[j]=v[i]+arc_length[k]; i_path[j]=i;
        }//if
       }//if
      }//for
     }//if
     if(scanned_node==node-1) break;
     min_v=inf;
     for(l=scanned_node+1;l<=node;l++){
      j=t[l];
      if(min_v>v[j]){
       min_v=v[j]; i=j;
      }//if
     }//for
     scanned_node=scanned_node+1;
     l=k_end_node[i]; j=t[scanned_node];
     k_end_node[j]=l; t[l]=j; k_end_node[i]=0;
    }//while
   }//if
   for(j=1;j<=node;j++){
    distance[iis][j]=v[j]; path[iis][j]=i_path[j];
   }//for
  }//for
 }// dijk
 //*********************************************
 void warshall(int distance[][n_num]){
  int newl;
  int i,j,l;
  for(i=1;i<=node;i++){
   for(j=1;j<=node;j++){
    path[i][j]=i;
    if(x[i][j]==1||x[j][i]==1){
     distance[i][j]=flow_cost[i][j];
    }else{
     distance[i][j]=inf;
    }//if
    if(i==j){
     distance[i][j]=0;
    }//if
   }//for
  }//for
  for(l=1;l<=node;l++){
   for(i=1;i<=node;i++){
    for(j=1;j<=node;j++){
     newl=distance[i][l]+distance[l][j];
     if(distance[i][j]>newl){
      distance[i][j]=newl; path[i][j]=path[l][j];
     }//if
    }//for
   }//for
  }//for
 }// warshall
 //*********************************************
 void short_delete(int del_i,int del_j,int distance2[][n_num]){
  int node1[n_num],node2[n_num],pi_i[icomm_num],pi_l[icomm_num]
  ;
  int node1_number,node2_number,node3_number,i,l,m,ii,ll,il;
  node1_number=0; node2_number=0; node3_number=0;
  for(i=1;i<=node;i++){
   node1[i]=0; node2[i]=0;
  }//for
  for(i=1;i<=node;i++){
  if(distance2[i][del_j]==distance2[i][del_i]+flow_cost[del_i][
   del_j]){
    node1_number=node1_number+1; node1[node1_number]=i;
   }//if
  }//for
  for(l=1;l<=node;l++){
   if(distance2[del_i][l]==flow_cost[del_i][del_j]+distance2[
   del_j][l]){
    node2_number=node2_number+1; node2[node2_number]=l;
   }//if
  }//for
  for(ii=1;ii<=node1_number;ii++){
   for(ll=1;ll<=node2_number;ll++){
    i=node1[ii]; l=node2[ll];
    if(distance2[i][l]==distance2[i][del_i]+flow_cost[del_i][de
    l_j]+distance2[del_j][l]){
     if(x[i][l]==1){
      distance2[i][l]=flow_cost[i][l];
      distance2[l][i]=flow_cost[i][l];
     }else{
      distance2[i][l]=inf; distance2[l][i]=inf;
     }//if
     node3_number=node3_number+1;
     pi_i[node3_number]=i; pi_l[node3_number]=l;
    }//if
   }//for
  }//for
  for(m=1;m<=node;m++){
   for(il=1;il<=node3_number;il++){
    i=pi_i[il]; l=pi_l[il];
    if(distance2[i][m]+distance2[m][l]<distance2[i][l]){
     distance2[i][l]=distance2[i][m]+distance2[m][l];
     distance2[l][i]=distance2[i][l];
    }//if
   }//for
  }//for
 }// short_delete
 //*********************************************
 void rest_backward(void){
  int distance[n_num][n_num];
  int i,j,rnode;
  float ct_fixed[link_num],p,ct_fixed_flag;
  link=0;
  for(i=1;i<=node;i++){
```

```
  for(j=i+1;j<=node;j++){
   link=link+1;
   ct_fixed[link]=ct[i][j]+fixed_cost[i][j];
  }//for
 }//for
 for(i=1;i<=link-1;i++){
  for(j=i+1;j<=link;j++){
   if(ct_fixed[i]>ct_fixed[j]){
    p=ct_fixed[i]; ct_fixed[i]=ct_fixed[j]; ct_fixed[j]=p;
   }//if
  }//for
 }//for
 rnode=node*rest_param; ct_fixed_flag=ct_fixed[rnode];
 for(i=1;i<=node;i++){
  for(j=i+1;j<=node;j++){
   x[i][j]=0; x[j][i]=0;
   if(ct[i][j]+fixed_cost[i][j]<ct_fixed_flag){
    x[i][j]=1; x[j][i]=1;
   }//if
  }//for
 }//for
 warshall(distance);
 object=calc_object(distance); upper_bound=object_update();
 if(object<big){
  minoux_mod(); dijk(distance);
  object=calc_object(distance); upper_bound=object_update();
 }//if
 current_object=object;
}// rest_backward
//*********************************************
void rest_forward(void){
 int distance[n_num][n_num],xin[n_num][n_num];
 int i,j,rnode;
 float ct_fixed[link_num],p,ct_fixed_flag;
 link=0;
 for(i=1;i<=node;i++){
  for(j=i+1;j<=node;j++){
   link=link+1; ct_fixed[link]=ct[i][j]+fixed_cost[i][j];
  }//for
 }//for
 for(i=1;i<=link-1;i++){
  for(j=i+1;j<=link;j++){
   if(ct_fixed[i]>ct_fixed[j]){
    p=ct_fixed[i]; ct_fixed[i]=ct_fixed[j]; ct_fixed[j]=p;
   }//if
  }//for
 }//for
 rnode=node*rest_param; ct_fixed_flag=ct_fixed[rnode];
 for(i=1;i<=node;i++){
  for(j=i+1;j<=node;j++){
   xin[i][j]=0; xin[j][i]=0;
   if(fixed_cost[i][j]<inf){
    if(ct[i][j]+fixed_cost[i][j]<=ct_fixed_flag){
     xin[i][j]=1; xin[j][i]=1;
    }//if
   }//if
  }//for
 }//for
 m_forw_mod(xin); dijk(distance);
 object=calc_object(distance); upper_bound=object_update();
 current_object=object;
}// rest_forward
//*********************************************
void m_forw_mod(int xin[][n_num]){
 int distance[n_num][n_num];
 int addobj_k[link_num],x_i[link_num],x_j[link_num];
 int point[link_num];
 c_point=0;
 warshall(distance);
 object=calc_object(distance); upper_bound=object_update();
 if(object<big){
  m_forw_init_1(x_i,x_j,addobj_k,xin); minoux_init_2(point);
  if(link>0){
   m_forw_main(x_i,x_j,point,addobj_k);
   object=calc_object(distance); upper_bound=object_update();
  }//if
 }//if
}// m_forw_mod
//*********************************************
void m_forw_init_1(int x_i[],int x_j[],int addobj_k[],int
xin[][n_num]){
 int i,j,l;
 calc_flow();
 addobj_k[0]=inf; link=0;
 for(i=1;i<=node;i++){
  for(j=i+1;j<=node;j++){
   if(x[i][j]==0&&fixed_cost[i][j]<inf){
    if(xin[i][j]==1){
     link=link+1; length_ij=calc_length(i,j);
     min_flow=calc_min_flow(i,j); x_i[link]=i; x_j[link]=j;
     addobj_k[link]=min_flow*flow_cost[i][j]+fixed_cost[i][j]
      -min_flow*length_ij;
    }//if
   }//if
  }//for
 }//for
 for(i=1;i<=link-1;i++){
  for(j=i+1;j<=link;j++){
   if(addobj_k[i]>addobj_k[j]){
    l=addobj_k[i]; addobj_k[i]=addobj_k[j]; addobj_k[j]=l;
    l=x_i[i]; [i]=x_i[j]; x_i[j]=l;
    l=x_j[i]; x_j[i]=x_j[j]; x_j[j]=l;
   }//if
  }//for
 }//for
}// m_forw_init_1
//*********************************************
void m_forw_main(int x_i[],int x_j[],int point[],int
addobj_k[]){
 int distance[n_num][n_num],distance2[n_num][n_num];
 int add_link,next_point,i,j;
 float add_object;
 dijk(distance);
 object=calc_object(distance); upper_bound=object_update();
 short_init(distance,distance2);
 current_object=object;
 do{
  add_link=point[0]; i=x_i[add_link]; j=x_j[add_link];
  x[i][j]=1; x[j][i]=1;
  short_init(distance,distance2); short_add(i,j,distance2);
```

```
    for(j=i+1;j<=node;j++){
      link=link+1;
      ct_fixed[link]=ct[i][j]+fixed_cost[i][j];
    }//for
  }//for
  for(i=1;i<=link-1;i++){
    for(j=i+1;j<=link;j++){
      if(ct_fixed[i]>ct_fixed[j]){
       p=ct_fixed[i]; ct_fixed[i]=ct_fixed[j]; ct_fixed[j]=p;
      }//if
    }//for
  }//for
  rnode=node*rest_param; ct_fixed_flag=ct_fixed[rnode];
  for(i=1;i<=node;i++){
    for(j=i+1;j<=node;j++){
      x[i][j]=0; x[j][i]=0;
      if(ct[i][j]+fixed_cost[i][j]<ct_fixed_flag){
       x[i][j]=1; x[j][i]=1;
      }//if
    }//for
  }//for
  warshall(distance);
  object=calc_object(distance); upper_bound=object_update();
  if(object<big){
   minoux_mod(); dijk(distance);
   object=calc_object(distance); upper_bound=object_update();
  }//if
  current_object=object;
}// rest_backward
//***********************************************
void rest_forward(void){
  int distance[n_num][n_num],xin[n_num][n_num];
  int i,j,rnode;
  float ct_fixed[link_num],p,ct_fixed_flag;
  link=0;
  for(i=1;i<=node;i++){
    for(j=i+1;j<=node;j++){
      link=link+1; ct_fixed[link]=ct[i][j]+fixed_cost[i][j];
    }//for
  }//for
  for(i=1;i<=link-1;i++){
    for(j=i+1;j<=link;j++){
      if(ct_fixed[i]>ct_fixed[j]){
       p=ct_fixed[i]; ct_fixed[i]=ct_fixed[j]; ct_fixed[j]=p;
      }//if
    }//for
  }//for
  rnode=node*rest_param; ct_fixed_flag=ct_fixed[rnode];
  for(i=1;i<=node;i++){
    for(j=i+1;j<=node;j++){
      xin[i][j]=0; xin[j][i]=0;
      if(fixed_cost[i][j]<inf){
       if(ct[i][j]+fixed_cost[i][j]<=ct_fixed_flag){
        xin[i][j]=1; xin[j][i]=1;
       }//if
      }//if
    }//for
  }//for
  m_forw_mod(xin); dijk(distance);
  object=calc_object(distance); upper_bound=object_update();
  current_object=object;
}// rest_forward
//***********************************************
void m_forw_mod(int xin[][n_num]){
  int distance[n_num][n_num];
  int addobj_k[link_num],x_i[link_num],x_j[link_num];
  int point[link_num];
  c_point=0;
  warshall(distance);
  object=calc_object(distance); upper_bound=object_update();
  if(object<big){
   m_forw_init_1(x_i,x_j,addobj_k,xin); minoux_init_2(point);
   if(link>0){
    m_forw_main(x_i,x_j,point,addobj_k);
    object=calc_object(distance); upper_bound=object_update();
   }//if
  }//if
}// m_forw_mod
//***********************************************
void m_forw_init_1(int x_i[],int x_j[],int addobj_k[],int
xin[][n_num]){
  int i,j,l;
  calc_flow();
  addobj_k[0]=inf; link=0;
  for(i=1;i<=node;i++){
    for(j=i+1;j<=node;j++){
      if(x[i][j]==0&&fixed_cost[i][j]<inf){
        if(xin[i][j]==1){
          link=link+1; length_ij=calc_length(i,j);
          min_flow=calc_min_flow(i,j); x_i[link]=i; x_j[link]=j;
          addobj_k[link]=min_flow*flow_cost[i][j]+fixed_cost[i][j]
            -min_flow*length_ij;
        }//if
      }//if
    }//for
  }//for
  for(i=1;i<=link-1;i++){
    for(j=i+1;j<=link;j++){
      if(addobj_k[i]>addobj_k[j]){
        l=addobj_k[i]; addobj_k[i]=addobj_k[j]; addobj_k[j]=l;
        l=x_i[i]; [i]=x_i[j]; x_i[j]=l;
        l=x_j[i]; x_j[i]=x_j[j]; x_j[j]=l;
      }//if
    }//for
  }//for
}// m_forw_init_1
//***********************************************
void m_forw_main(int x_i[],int x_j[],int point[],int
addobj_k[]){
  int distance[n_num][n_num],distance2[n_num][n_num];
  int add_link,next_point,i,j;
  float add_object;
  dijk(distance);
  object=calc_object(distance); upper_bound=object_update();
  short_init(distance,distance2);
  current_object=object;
  do{
    add_link=point[0]; i=x_i[add_link]; j=x_j[add_link];
    x[i][j]=1; x[j][i]=1;
    short_init(distance,distance2); short_add(i,j,distance2);
```

```
   object=calc_object(distance2); upper_bound=object_update();
   short_init(distance,distance2);
   x[i][j]=0; x[j][i]=0;
   add_object=object-current_object;
   if(add_object>=0){
    point[0]=point[add_link]; point[add_link]=0;
   }else{
    if(add_object<=addobj_k[point[add_link]]){
     x[i][j]=1;  x[j][i]=1;
     dijk(distance);
     object=calc_object(distance); upper_bound=object_update();
     current_object=object;
     point[0]=point[add_link]; point[add_link]=0;
    }else{
     c_point=add_link; next_point=point[c_point];
     while(1){
      if(add_object<=addobj_k[next_point]){
       point[0]=point[add_link]; point[c_point]=add_link;
       point[add_link]=next_point;
       addobj_k[add_link]=add_object;
       break;
      }//if
      c_point=next_point; next_point=point[c_point];
     }//while
    }//if
   }//if
  }while(addobj_k[c_point]<=0&&point[0]!=0);
 }// m_forw_main
 //*********************************************
 int calc_min_flow(int i,int j){
  int k,l,minimun_flow;
  if(i!=j){
   minimun_flow=inf; l=j;
   while(1){
    k=path[i][l];
    if(k!=0){
     if(flow[k][l]+flow[l][k]<min_flow){
      minimun_flow=flow[k][l]+flow[l][k];
     }//if
     l=k;
     if(i==l) break;
    }else{
     break;
    }//if
   }//while
  }else{
   minimun_flow=0;
  }//if
  return(minimun_flow);
 }// calc_min_flow
 //*********************************************
 void short_add(int add_i,int add_j,int distance2[][n_num]){
  int node1[n_num],node2[n_num],node1_number,node2_number;
  int i,l,ii,ll;
  node1_number=0; node2_number=0;
  for(i=1;i<=node;i++){
   node1[i]=0; node2[i]=0;
  }//for
  for(i=1;i<=node;i++){
   if(distance2[i][add_j]>distance2[i][add_i]+flow_cost[add_i]
   [add_j]){
    node1_number=node1_number+1; node1[node1_number]=i;
    distance2[i][add_j]=distance2[i][add_i]+flow_cost[add_i][ad
    d_j];
    distance2[add_j][i]=distance2[i][add_j];
   }//if
  }//for
  for(l=1;l<=node;l++){
   if(distance2[add_i][l]>flow_cost[add_i][add_j]+distance2[ad
   d_j][l]){
    node2_number=node2_number+1; node2[node2_number]=l;
    distance2[add_i][l]=flow_cost[add_i][add_j]+distance2[add_
    j][l];
    distance2[l][add_i]=distance2[add_i][l];
   }//if
  }//for
  for(ii=1;ii<=node1_number;ii++){
   for(ll=1;ll<=node2_number;ll++){
    i=node1[ii]; l=node2[ll];
    if(distance2[i][l]>distance2[i][add_i]+distance2[add_i][l])
     {
      distance2[i][l]=distance2[i][add_i]+distance2[add_i][l];
      distance2[l][i]=distance2[i][l];
     }//if
    }//for
   }// short_add
   //*********************************************
   void adjacent_local_main(void){
    int distance[n_num][n_num],distance2[n_num][n_num];
    int distance3[n_num][n_num];
    int adj_node_2[n_num];
    int n1[n_num],n2[n_num],use_node[n_num];
    int i,j,k,l,k1,l1,x_flag[n_num][n_num];
    float current_upper_bound;
    for(i=1;i<=node;i++){
     for(j=1;j<=node;j++){
      x_flag[i][j]=0;
     }//for
    }//for
    dijk(distance);
    object=calc_object(distance); current_object=object;
    do{
     current_upper_bound=current_object;
     for(i=1;i<=node;i++){
      for(j=i+1;j<=node;j++){
       if(x[i][j]==1){
        x[i][j]=0; x[j][i]=0; x_flag[i][j]=1; x_flag[j][i]=1;
        short_init(distance,distance2);
        short_delete(i,j,distance2);
        short_init(distance2,distance3);
        g_connect=connect(distance2);
        marge_node(adj_node_2,i,j,use_node);
        if(g_connect==1){
         for(k1=1;k1<=n_adj_node;k1++){
          k=adj_node_2[k1];
          for(l1=k1+1;l1<=n_adj_node;l1++){
           l=adj_node_2[l1];
           if(fixed_cost[k][l]!=inf&&x[k][l]==0){
            if(x_flag[k][l]==0){
```

```
   object=calc_object(distance2); upper_bound=object_update();
   short_init(distance,distance2);
   x[i][j]=0; x[j][i]=0;
   add_object=object-current_object;
   if(add_object>=0){
    point[0]=point[add_link]; point[add_link]=0;
   }else{
    if(add_object<=addobj_k[point[add_link]]){
     x[i][j]=1;  x[j][i]=1;
     dijk(distance);
     object=calc_object(distance); upper_bound=object_update();
     current_object=object;
     point[0]=point[add_link]; point[add_link]=0;
    }else{
     c_point=add_link; next_point=point[c_point];
     while(1){
      if(add_object<=addobj_k[next_point]){
       point[0]=point[add_link]; point[c_point]=add_link;
       point[add_link]=next_point;
       addobj_k[add_link]=add_object;
       break;
      }//if
      c_point=next_point; next_point=point[c_point];
     }//while
    }//if
   }//if
  }while(addobj_k[c_point]<=0&&point[0]!=0);
}// m_forw_main
//*********************************************
int calc_min_flow(int i,int j){
 int k,l,minimun_flow;
 if(i!=j){
  minimun_flow=inf; l=j;
  while(1){
   k=path[i][l];
   if(k!=0){
    if(flow[k][l]+flow[l][k]<min_flow){
     minimun_flow=flow[k][l]+flow[l][k];
    }//if
    l=k;
    if(i==l) break;
   }else{
    break;
   }//if
  }//while
 }else{
  minimun_flow=0;
 }//if
 return(minimun_flow);
}// calc_min_flow
//*********************************************
void short_add(int add_i,int add_j,int distance2[][n_num]){
 int node1[n_num],node2[n_num],node1_number,node2_number;
 int i,l,ii,ll;
 node1_number=0; node2_number=0;
 for(i=1;i<=node;i++){
  node1[i]=0; node2[i]=0;
 }//for
 for(i=1;i<=node;i++){
  if(distance2[i][add_j]>distance2[i][add_i]+flow_cost[add_i]
  [add_j]){
   node1_number=node1_number+1; node1[node1_number]=i;
   distance2[i][add_j]=distance2[i][add_i]+flow_cost[add_i][ad
   d_j];
   distance2[add_j][i]=distance2[i][add_j];
  }//if
 }//for
 for(l=1;l<=node;l++){
  if(distance2[add_i][l]>flow_cost[add_i][add_j]+distance2[ad
  d_j][l]){
   node2_number=node2_number+1; node2[node2_number]=l;
   distance2[add_i][l]=flow_cost[add_i][add_j]+distance2[add_
   j][l];
   distance2[l][add_i]=distance2[add_i][l];
  }//if
 }//for
 for(ii=1;ii<=node1_number;ii++){
  for(ll=1;ll<=node2_number;ll++){
   i=node1[ii]; l=node2[ll];
   if(distance2[i][l]>distance2[i][add_i]+distance2[add_i][l])
   {
    distance2[i][l]=distance2[i][add_i]+distance2[add_i][l];
    distance2[l][i]=distance2[i][l];
   }//if
  }//for
 }//for
}// short_add
//*********************************************
void adjacent_local_main(void){
 int distance[n_num][n_num],distance2[n_num][n_num];
 int distance3[n_num][n_num];
 int adj_node_2[n_num];
 int n1[n_num],n2[n_num],use_node[n_num];
 int i,j,k,l,k1,l1,x_flag[n_num][n_num];
 float current_upper_bound;
 for(i=1;i<=node;i++){
  for(j=1;j<=node;j++){
   x_flag[i][j]=0;
  }//for
 }//for
 dijk(distance);
 object=calc_object(distance); current_object=object;
 do{
  current_upper_bound=current_object;
  for(i=1;i<=node;i++){
   for(j=i+1;j<=node;j++){
    if(x[i][j]==1){
     x[i][j]=0; x[j][i]=0; x_flag[i][j]=1; x_flag[j][i]=1;
     short_init(distance,distance2);
     short_delete(i,j,distance2);
     short_init(distance2,distance3);
     g_connect=connect(distance2);
     marge_node(adj_node_2,i,j,use_node);
     if(g_connect==1){
      for(k1=1;k1<=n_adj_node;k1++){
       k=adj_node_2[k1];
       for(l1=k1+1;l1<=n_adj_node;l1++){
        l=adj_node_2[l1];
        if(fixed_cost[k][l]!=inf&&x[k][l]==0){
         if(x_flag[k][l]==0){
```

```
            x[k][l]=1; x[l][k]=1;
            short_init(distance3,distance2);
            short_add(k,l,distance2);
            object=calc_object(distance2);
            if(object<current_object){
             dijk(distance);
             current_object=object; upper_bound=object_update();
             continue;
            }//if
            x[k][l]=0; x[l][k]=0;
           }//if
          }//if
         }//for
        }//for
       }else{
        connect_node(n1,n2,use_node,distance3);
        for(k1=1;k1<=n1_num;k1++){
         k=n1[k1];
         for(l1=1;l1<=n2_num;l1++){
          l=n2[l1];
          if(fixed_cost[k][l]!=inf&&x[k][l]==0){
           if(x_flag[k][l]==0){
            x[k][l]=1; x[l][k]=1;
            short_init(distance3,distance2);
            short_add(k,l,distance2);
            object=calc_object(distance2);
            if(object<current_object){
             dijk(distance);
             current_object=object; upper_bound=object_update();
             continue;
            }//if
            x[k][l]=0;
            x[l][k]=0;
           }//if
          }//if
         }//for
        }//for
       }//if
       x[i][j]=1; x[j][i]=1; x_flag[i][j]=0; x_flag[j][i]=0;
      }//if
```

```
     }//for
    }//for
   }while(current_object<current_upper_bound);
   dijk(distance);
   upper_bound=object_update();
  }// adjacent_local_main
  //*********************************************
  void adjacent_node(void){
   int x_j[link_num],x_length[link_num];
   int i,j,h,l;
   for(i=1;i<=node;i++){
    for(j=1;j<=node;j++){
     x_j[j]=j; x_length[j]=fixed_cost[i][j];
    }//for
    for(h=1;h<=node-1;h++){
     for(j=h+1;j<=node;j++){
      if(x_length[h]>x_length[j]){
       l=x_length[h]; x_length[h]=x_length[j]; x_length[j]=l;
       l=x_j[h]; x_j[h]=x_j[j]; x_j[j]=l;
      }//if
     }//for
    }//for
    for(j=1;j<=node;j++){
     adj_node[i][j]=x_j[j];
    }//for
   }//for
  }// adjacent_node
  //*********************************************
  int connect(int distance2[][n_num]){
   int i,j,graph_connect;
   graph_connect=1;
   for(i=1;i<=node;i++){
    for(j=i+1;j<=node;j++){
     if(distance2[i][j]>=inf){
      graph_connect=0;
      return(graph_connect);
     }//if
    }//for
   }//for
   return(graph_connect);
```

```
  }// connect
  //*********************************************
  void marge_node(int adj_node_2[],int i,int j,int use_node[]){
   int k,l,m;
   for(k=1;k<=node;k++){
    use_node[k]=0;
   }//for
   k=0; l=1; m=1;
   do{
    if(fixed_cost[i][adj_node[i][l]]<fixed_cost[j][adj_node[j]
    [m]]){
     if(use_node[adj_node[i][l]]==0){
      k=k+1; adj_node_2[k]=adj_node[i][l];
      use_node[adj_node[i][l]]=1;
     }//if
     l=l+1;
    }else{
     if(use_node[adj_node[j][m]]==0){
      k=k+1; adj_node_2[k]=adj_node[j][m];
      use_node[adj_node[j][m]]=1;
     }//if
     m=m+1;
    }//if
   }while(k!=n_adj_node);
  }// marge_node
  //*********************************************
  void connect_node(int n1[],int n2[],int use_node[],int distan
  ce3[][n_num]){
   int j;
   n1_num=0; n2_num=0;
   if(use_node[1]==1){
    n1_num=n1_num+1; n1[n1_num]=use_node[1];
   }//if
   for(j=2;j<=node;j++){
    if(use_node[j]==1){
     if(distance3[1][j]<inf){
      n1_num=n1_num+1; n1[n1_num]=j;
     }else{
      n2_num=n2_num+1; n2[n2_num]=j;
     }//if
```

```
  }//if
 }//for
}// connect_node
//************************************************
```

図 3. param.dat

```
//問題の分類
100505
//デザイン費用比率
10
//アーク基準の上限
05
//品種基準の上限
05
```

図 4. und.dat

```
//問題数
  1
//ノード数
  10
//フロー費用
  0 745 738 261 519 395 230 450 286 565
745   0 181 629 331 841 921 357 579 213
738 181   0 685 237 915 941 443 639 178
261 629 685   0 523 237 317 276   50 509
519 331 237 523   0 760 735 356 486 125
395 841 915 237 760   0 282 484 278 742
230 921 941 317 735 282   0 584 365 764
450 357 443 276 356 484 584   0 226 287
286 579 639   50 486 278 365 226   0 465
565 213 178 509 125 742 764 287 465   0
//アーク基準
  0  4  4  1  3  3  1  3  2  3
  4  0  1  4  2  5  5  2  4  1
  4  1  0  4  1  5  5  3  4  1
  1  4  4  0  3  1  2  2  1  3
  3  2  1  3  0  4  4  2  3  1
  3  5  5  1  4  0  2  3  2  4
  1  5  5  2  4  2  0  4  2  5
  3  2  3  2  2  3  4  0  1  1
  2  4  4  1  3  2  2  1  0  3
  3  1  1  3  1  4  5  1  3  0
//品種基準
  0  2  1  3  8  6  4  7  2  6
  3  0  2  2  5  1  5  9  4  7
  3  1  0  6  5  9  9  4  6  2
  4  3  3  0  5  7  1  9  4  6
  7  8  7  7  0  7  5  1  2  8
  4  2  1  8  6  0  6  4  3  5
  2  1  8  7  9  9  0  4  8  8
  3  6  3  8  8  7  1  0  9  8
  7  9  3  4  9  5  1  9  0  4
  5  2  2  1  6  3  6  5  5  0
//x, y 座標
 764 357
 106 707
  48 536
 728 616
 248 408
 947 707
 983 427
 463 692
 680 630
 226 531
```