

最適化シミュレーション演習 第6回

1. Dijkstra 法

以下の Dijkstra 法のアルゴリズムに従って、最短路を求めよ。なお、 π_i (現在の反復までに求められた点 i への最短距離) は空欄に記入し、 p_i (現在の反復までに求められた点 i への最短路の直前点) はネットワークの辺に記入せよ。

- ・ ステップ 1 : $\pi_0=0, \pi_i=\infty$ ($i \in V - \{0\}$), $M=\{1,2,\dots,n\}$, $i=0$
- ・ ステップ 2 : $M \neq \emptyset$ (空集合) でない場合 (i)(ii)(iii) を繰り返す
 - (i) $j \in M$ に到達する直前点を可能な限り i とする。 $j \in M$ に対し $\pi_j > \pi_i + c_{ij}$ ならば $\pi_j = \pi_i + c_{ij}$, $p_j = i$
 - (ii) $j \in M$ への暫定的最短距離から最小の $\min_{j \in M} \pi_j = \pi_k$ となる k を求める。
 - (iii) k を M から除く、 $i=k$ とする (k からの探索を行う)。

```
#dijkstra-kadai.run
```

```
param n; #ノード数
```

```
param k; #探索開始点
```

```
param jj; #先行点を表す
```

```
param c[i in 1..n, j in 1..n]; #ij 間距離
```

```
param tim[i in 1..n, j in 1..n]; #ij 間時間
```

```
param in_number[i in 1..n]; #i への入次数
```

```
param out_number[i in 1..n]; #i から出次数
```

```
param in_node[i in 1..n, j in 1..n]; #i への入ノード
```

```
param out_node[i in 1..n, j in 1..n]; #i から出ノード
```

```
param b[i in 1..n]; #minimum cost flow としての右辺定数
```

```
param pi[i in 1..n]; #点 i までの暫定的最短距離
```

```
param prev[i in 1..n]; #点 i への暫定的最短路における直前点
```

```
param fix[i in 1..n]; #点 i への最短路確定を示す
```

```
param pi_min; #暫定的最短距離
```

```
param mu;
```

```
data dijkstra.dat;
```

```
for {i in 1..n}
```

```
{
```

```
    let pi[i]:=1e10; #ステップ 1 初期設定  $\pi$ 
```

```
    let fix[i]:=0; #初期設定: 全点最短路未確定
```

```
}
```

```
let pi[1]:=0;
```

```
let prev[1]:=0;
```

```
let fix[1]:=1;
```

```
let k:=1; #探索開始点
```

```

repeat while(sum[i in 1..n]fix[i] < n) #すべての点が最短経路決定されるまで
    #Dijkstra 法全体の反復
{
    #ステップ 2(i)
    for[j in 1..out_number[k]] #探索開始点 k から接続する点 j すべてに対し
    {
        if(pi[out_node[k,j]]>pi[k]+c[k,out_node[k,j]] and out_node[k,j] != k and fix[out_node[k,j]]=0)then
        {
            #点 k から out_node[k,j]への距離が近い場合更新
            let pi[out_node[k,j]]:=pi[k]+c[k,out_node[k,j]];
            let prev[out_node[k,j]]:=k;
        }
    }
    #ステップ 2(ii)
    let pi_min:=10000;
    for[j in 1..n]#最短経路決定されていない点(fix[j]=0)で  $\pi$  の値が最小のものを選ぶ
    {
        if(pi[j]<pi_min and j != k and fix[j]=0)then
        {
            let pi_min:=pi[j];
            let k:=j;
        }
    }
    #ステップ 2(iii)
    let fix[k]:=1;
} #Dijkstra 法全体の反復
display pi;

for[j in 1..n] #すべての点への最短経路表示
{
    let jj:=j;
    printf:"node %d ",jj;
    repeat while( prev[jj]>0)
    {
        printf:"node %d ",prev[jj];
        let jj :=prev[jj];
    }
    printf:"¥n";
}

```

単純に点 i から n までの最短経路を求めたい場合は、 $\text{fix}[n]=1$ となった時点で終了すればよい。

課題：費用を $c_{ij} + \mu t_{ij}$ ($\mu=2$) とした場合の最短経路を Dijkstra 法で求めるように、run ファイルを修正せよ。

2. Lagrange 緩和法

```
#lag-lp.run
model lag-lp.mod;
data lag-lp.dat;

param theta; #ステップサイズ  $\theta$ 
param Lopt; #暫定目的関数値
param lambda; #係数  $\lambda$ 
param renzoku; #下界値の上昇が起こらない連続回数
param oldL; #前反復までの下界値
param tmpx{i in 1..n, j in 1..n}; #暫定解

let T:=14;
let Lopt:=24; #初期暫定値として、24 を与えた。
let mu:=0.0;
let lambda:=0.8;
let renzoku:=0;
let oldL:=-1e10;
printf:" k, mu, time-T, L(mu), lambda, theta %n" > lag-lp.sol;
for{iter in 0..200} #ラグランジュ緩和法の反復回数
{
    solve;

    if((sum{i in 1..n, j in 1..out_number[i]}tim[i,out_node[i,j]]*x[i,out_node[i,j]] <=T)
    and (sum{i in 1..n, j in 1..out_number[i]}c[i,out_node[i,j]]*x[i,out_node[i,j]] < Lopt))then
    {
        #実行可能 (時間制約) かつ目的関数値が暫定値より良い場合、暫定値更新
        let Lopt := (sum{i in 1..n, j in 1..out_number[i]}
                     c[i,out_node[i,j]]*x[i,out_node[i,j]]);
        for{i in 1..n}
        {
            for{j in 1..n}
            {
                let tmpx[i,j]:=x[i,j]; #暫定解更新
            }
        }
    }
}

if(distance <= oldL)then #前反復までの下界値が更新されない場合
{
    let renzoku:=renzoku+1; #下界値が更新されない回数を増加
    if(renzoku>=4)then #下界値が更新されない回数が4回連続なら
    {
        let renzoku:=0;
        let lambda:=lambda*1/2; # $\lambda$  を 1/2 倍
    }
}
}
```

```
else    #前反復までの下界値が更新される場合
```

```
{
    let oldL:=distance;
    let renzoku:=0;
}
```

```
if(sum{i in 1..n, j in 1..out_number[i]}tim[i,out_node[i,j]]*x[i,out_node[i,j]] <> T)then
#最適性チェック成立しない場合
```

```
{
    let theta:= lambda*(Lopt-distance)
    /(sum{i in 1..n, j in 1..out_number[i]}tim[i,out_node[i,j]]*x[i,out_node[i,j]]-T)**2;
    #ステップサイズ $\theta$ を決定
}
else
{
    break;
}
```

```
printf:"%3d, %6.6f, %3d, %10.6f, %8.6f, %8.6f¥n",
    iter, mu, (sum{i in 1..n, j in 1..out_number[i]}tim[i,out_node[i,j]]*x[i,out_node[i,j]]-T),
    distance, lambda, theta > lag-lp.sol;
```

```
let mu:=mu+theta*(sum{i in 1..n, j in 1..out_number[i]}
    tim[i,out_node[i,j]]*x[i,out_node[i,j]]-T);
#ラグランジュ乗数 $\mu$ の更新
```

```
}
```

```
display tmpx > lag-lp.sol;
display sum{i in 1..n, j in 1..out_number[i]}c[i,out_node[i,j]]*tmpx[i,out_node[i,j]] >
lag-lp.sol;
```

課題: $T=12$ のときラグランジュ緩和法により問題を解け。得られた解は最適解であるかどうか検討せよ。