使ってみよう GLPK

2017 年 5 月 11 日改訂版
慶應義塾大学　大学院理工学研究科
岡本　聡

1. GLPK とは

GLPK (GNU Linear Programming Kit) は線形計画問題を解いてくれるツールで、ロシアの A.O. Makhorin によって開発された、フリーソフトウェアである。
https://www.gnu.org/software/glpk/
よりダウンロードすることが可能であり、自分のパソコンにインストールすることもできる。
授業においては、ITC の Linux ワークステーションにインストール済のものを使って、自分でモデルファイルを作成して、GLPK に最適解を探索させる。
与えられた問題から、自分でモデルファイルを作成することが GLPK を使うということである。

> 第７回から第８回の授業は、この資料に沿って自主的に行ってください。授業の最初に説明はありません。仲間と組んで進めても OK です。先生は、巡回していますので、随時質問してください。

2. GLPK がインストールされていることの確認

2.1. which コマンドにより glpsol を探索する

ワークステーションにログインして、以下のコマンドを打つ ※”△”はスペースの意味※

```
$ which△glpsol
```

インストールされていれば、

```
$ which△glpsol
/usr/local/bin/glpsol
$
```

のように、探索したコマンドの絶対パスが表示される。
タイプミスや未インストール等により存在していなければ、

```
$ which△glpsol
glpsol: コマンドが見つかりません
$
```

のようにエラーメッセージが表示される。 glpsol が無いようであれば、インストール作業を行う。glpsol が存在しているならば、3 章に飛んでも良いが、2.2 のインストールを体験してみるのも一つの経験。

2.2. GLPK パッケージのインストール

sudo コマンドによって super-user 権限での作業を行うことができないため自分のホームに bin/ と lib/ を作成してインストール作業を行う。https://www.gnu.org/software/glpk/ から glpk-4.61.tar.gz をダウンロードします。

2.2.1. ~/bin/ と ~/lib/ の作成

$ cd　　←このコマンドで自分のホームディレクトリに戻ります
$ mkdir△bin　　← ~/bin の作成
$ mkdir△lib　　← ~/lib の作成

2.2.2. PATH と LD_LIBRARY_PATH の設定

※細かな点は環境依存で異なるかもしれません。
自分の shell が/bin/sh や /bin/bash の場合（多分こちらです）
~/.bashrc に以下を emacs 等のエディタを用いて追加記述します。
#================ ここから追加 ========================
export PATH=$HOME/bin :$PATH
export LD_LIBRARY_PATH=$HOME/lib :$LD_LIBRARY_PATH
#================ここまで追加=========================
LD_LIBRARY_PATH は、コマンド実行時に動的リンクでライブラリをリンクする際の探索パスの指示です。
以下のコマンドを実行して設定を有効にします。
$ source△~/.bashrc
※ source コマンドは、実行した Terminal Window でのみ有効。面倒な場合は Terminal Window を exit して、再度 Terminal Window を開くのが確実に反映させるコツ。
goto 2.2.3

自分の shell が /bin/csh や /bin/tcsh の場合
~/.cshrc に以下を emacs 等を用いて追加記述します。
#================ ここから追加 ========================
set△path=($HOME/bin△$path△)
setenv△LD_LIBRARY_PATH△$HOME/lib :$LD_LIBRARY_PATH
#================ここまで追加=========================
LD_LIBRARY_PATH は、コマンド実行時に動的リンクでライブラリをリンクする際の探索パスの指示です。
以下のコマンドを実行
$ source△~/.cshrc
※ source コマンドは、実行した Terminal Window でのみ有効。面倒な場合は Terminal Window を exit して、再度 Terminal Window を開くのが確実に反映させるコツ。

2.2.3. GLPK パッケージのコンパイル

例えば作業用ディレクトリを ~/GLPK とします。
$ cd
$ mkdir△GLPK
$ cd△GLPK
$ zcat△[glpk-4.61.tar.gz のダウンロード場所]△|tar△xvf△-

```
$ cd△glpk-4.61
$ ./configure△--prefix=$HOME
$ make
$ make△check
$ make△install
$ rehash                    ※ bash の場合 rehash は不要
```

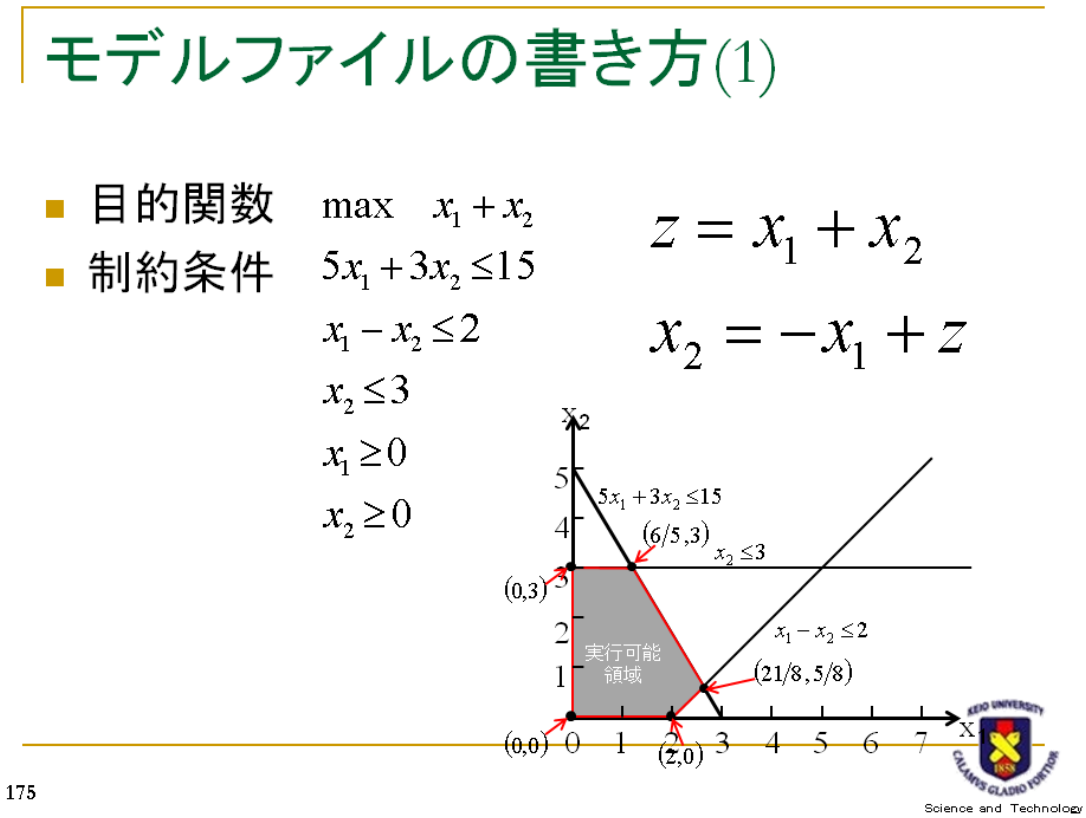
2.3. GLPL のバージョンの確認

```
$ glpsol△-v
4.61qqqqqqq でしょうか？
```

2.1 章に戻ります。

3. モデルファイルを記述してみましょう

3.1. 例題



ファイル名を lp-ex1.mod として、emacs 等のエディタで以下を記述します。
単純に打ち込む作業をするのではなく、問題の意味と記述の対応を学んで下さい。

```
/* lp-ex1.mod */

var x1 >= 0;
var x2 >= 0;

maximize z: x1 + x2 ;

s.t. st1: 5*x1 + 3*x2 <= 15 ;
s.t. st2: x1 - x2 <= 2;
s.t. st3: x2 <= 3 ;

end ;
```

モデルファイルを GLPK で解いてみます。

$ glpsol△-m△lp-ex1.mod△-o△lp-ex1.out

正しくモデルファイルが記述されていれば、画面に出力されるメッセージには

Model has been successfully generated

や

OPTICAL SOUTION FOUND

や

Writing basic solution to `lp-ex1.out'

というような記述が見られることでしょう。

実行結果を見てみましょう。

$ cat△lp-ex1.out

目的関数として、maximize z: x1 + x2 と記述したので、

Objective:　z = 4.2 (MAXimum)

という記述があることを確認しましょう。その時の x1 の値と x2 の値はどこを見れば良いか分かりますか？

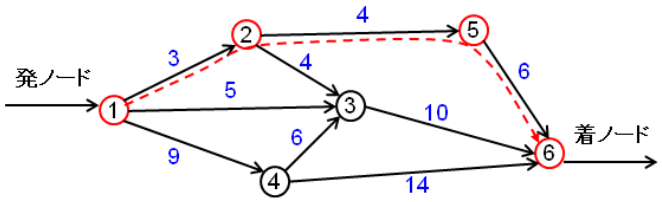
※練習：わざとモデルファイルに間違いを入れてみましょう。どんなエラーが出るのか、そのエラーからどこを修正すればよいのかを見つけることが大切です。

3.2.　最短経路問題

3.2.1.　与えられた問題のみを一生懸命に解いてみる

最短経路問題

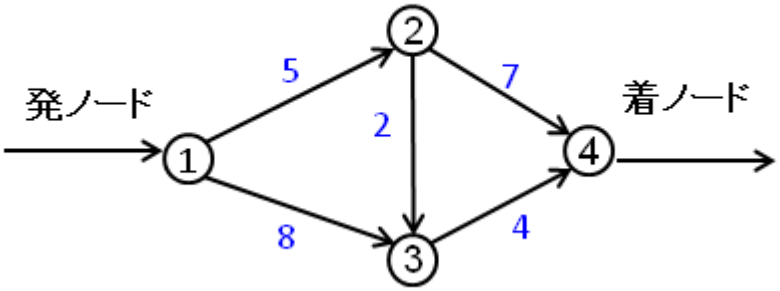
- ノード1からノード6まで、トラヒックを流す場合の最短経路を求める
  - 距離の和が最小になる経路



135

Science and Technology

まずは、簡単なモデルから。



目的関数は以下の通りになる。

$$\min \quad 5x_{12} + 8x_{13} + 2x_{23} + 7x_{24} + 4x_{34}$$

制約条件を考える。x_{ij}は、(i, j)を通ればトラヒック流量の割合、通らなければ0となる。

発ノードに流入するトラヒック量を1と仮定する。

$$0 \le x_{12} \le 1, \quad 0 \le x_{13} \le 1, \quad 0 \le x_{23} \le 1, \quad 0 \le x_{24} \le 1, \quad 0 \le x_{34} \le 1$$

■ノード1でのフロー保存　⇒　入ったトラヒック(流量=1を仮定)は、(1,2)か(1,3)のどち

らかに全て出る。

$$x_{12} + x_{13} = 1$$

■ノード２でのフロー保存　⇒　(1,2)から流入したトラヒックは、(2,3)と(2,4)のどちらかに全て出る。

$$x_{12} - x_{23} - x_{24} = 0$$

■ノード 3 でのフロー保存　⇒ (1,3)と(2,3)から流入したトラヒックは、(3,4)から全部出る。

$$x_{13} + x_{23} - x_{34} = 0$$

■ノード４でのフロー保存　⇒ (2,4)と(3,4)から流入したトラヒックの流量和は 1。

$$x_{24} + x_{34} = 1$$

ただし、ノード 4 でのフロー保存条件は、ノード 1～3 でのフロー保存条件が満たされていれば自動的に成立するので、実際には制約条件に加えなくても良い。

※ 実際に計算して３個の式から４式目が導けることを確認して下さい。

ファイル名を sp-ex1.mod として、emacs 等のエディタで以下を記述します。
単純に打ち込む作業をするのではなく、問題の意味と記述の対応を学んで下さい。

```
/* sp-ex1.mod */

/* Decision variables */
var x12 <=1, >=0 ;
var x13 <=1, >=0 ;
var x23 <=1, >=0 ;
var x24 <=1, >=0 ;
var x34 <=1, >=0 ;

/* Objective function */
minimize PATH_COST: 5*x12 + 8*x13 + 2*x23 + 7*x24 + 4*x34 ;

/* Constraints */
s.t. NODE1: x12 + x13 = 1 ;
s.t. NODE2: x12 - x23 - x24 = 0 ;
s.t. NODE3: x13 + x23 - x34 = 0 ;

end ;
```

モデルファイルを GLPK で解いてみます。
$ glpsol△-m△sp-ex1.mod△-o△sp-ex1.out

sp-ex1.out を見てみましょう。Objective: PATH_COST = 11 (MINimum) という記述は入っていますか？その時の経路は見つかりましたか？

※ 応用練習　3.2.1 の最初にある６ノードのネットワークに対する最短経路問題を解いてみましょう。

3.2.2.　文字式による一般化にチャレンジ

最短経路問題の文字表記

目的関数　$\min \sum_{(i,j)\in E} cost_{ij} x_{ij}$

制約条件　$\sum_{j\in V} x_{ij} - \sum_{j\in V} x_{ji} = 1 \quad (i = p \text{のとき})$

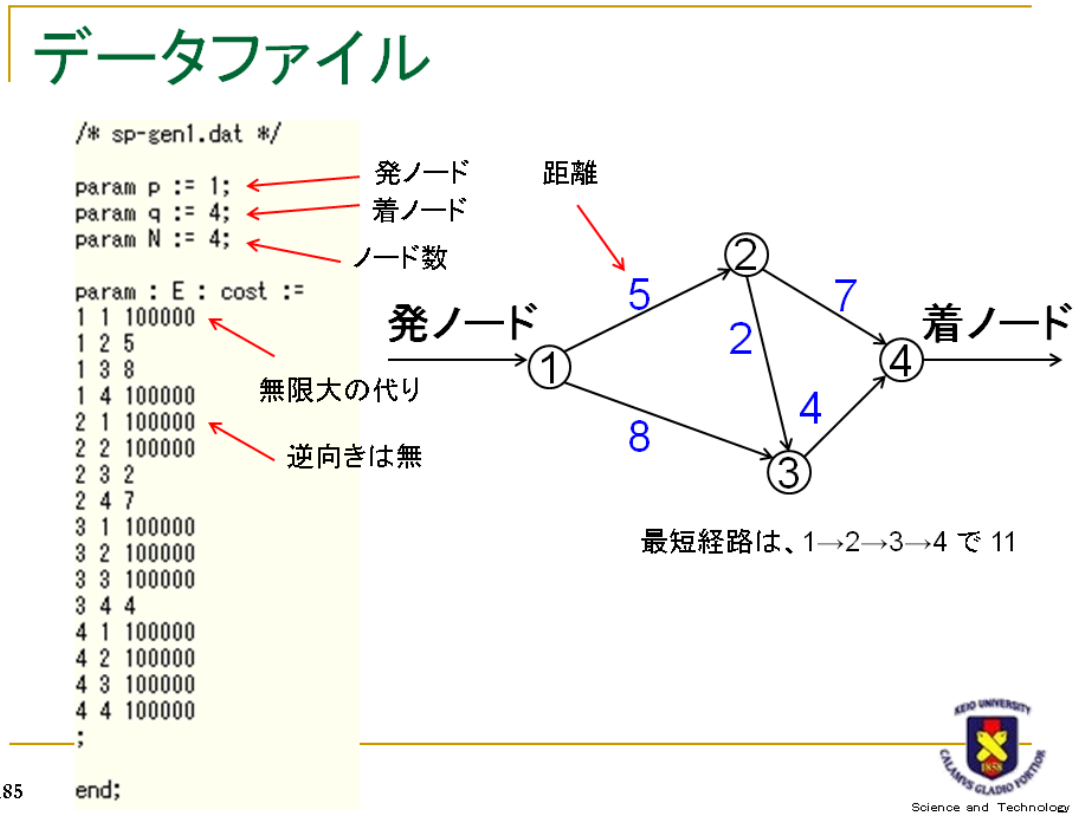
$\sum_{j\in V} x_{ij} - \sum_{j\in V} x_{ji} = 0 \quad (\forall i \neq p,q \in V)$

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

発ノードｐから着ノードｑまでの最短経路(最小コスト経路)を求める。
目的関数は、全リンクのコストの総和を最小化。通過リンクはフロー量 $x_{ij}=1$、不通過リンクは $x_{ij}=0$ となる。
最初の制約条件は、発ノードであるノードｐでのフロー保存。
二番目の制約条件は、発着ノード以外のノードでのフロー保存。
三番目の制約条件は、フロー量の制約である。
まずは、文字表記をモデルファイルに書き起こします。

```
/* sp-gen.mod */

/* Given parameters */
param N integer, >0;
param p integer, >0;
param q integer, >0;

set V := 1..N;
set E within {V,V};

param cost{E};

/* Decision variables */
var x{E} <=1, >=0;

/* Objective function */
minimize PATH_COST: sum{i in V} (sum{j in V} (cost[i,j]*x[i,j]) );

/* Constraints */
s.t. SOURCE{i in V: i = p && p != q}:
     sum{j in V} (x[i,j]) - sum{j in V} (x[j,i]) = 1;
s.t. INTERNAL{i in V: i != p && i != q && p != q}:
     sum{j in V} (x[i,j]) - sum{j in V} (x[j,i]) = 0;

end;
```

パラメータ N はノード数、p は発ノード、q は着ノード。変数 V を使って、i, j の組合せ範囲が 1～N の整数値であることを示しています。変数 E を使って、コストの二次元配列を定義しています。定義されていないリンクに対しては、一般的にコスト＝∞　やありえない大きな値を代入して使います。
制約条件の中の、 p != q は、発ノードと着ノードは違うノードであることを明示しています。
このモデルファイルに与えるデータファイルは次のようになります。



モデルファイルを GLPK で解いてみます。

$ glpsol△-m△sp-gen.mod△-d△sp-gen1.dat△-o△sp-gen1.out

sp-gen1.out を見てみましょう。Objective: PATH_COST = 11 (MINimum) という記述は入っていますか？その時の経路は見つかりましたか？それは、sp-ex1.out の経路と同じですか？

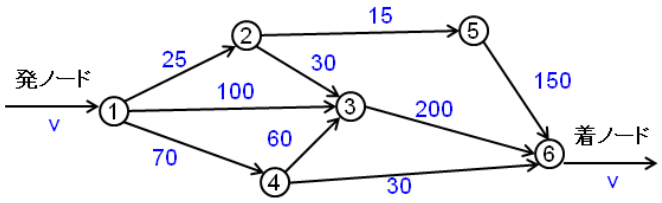
※ 応用練習　3.2.1 の最初にある 6 ノードのネットワークに対する dat ファイルを書いて最短経路問題を解いてみましょう。
※ 応用練習　(2,4)と(3,4) のリンクの値を 100000 に設定してみましょう。最短経路は存在しなくなります。どんな結果が出てくるでしょうか。

3.3.　最大流問題

3.3.1.　与えられた問題のみを一生懸命に解いてみる

最大流問題

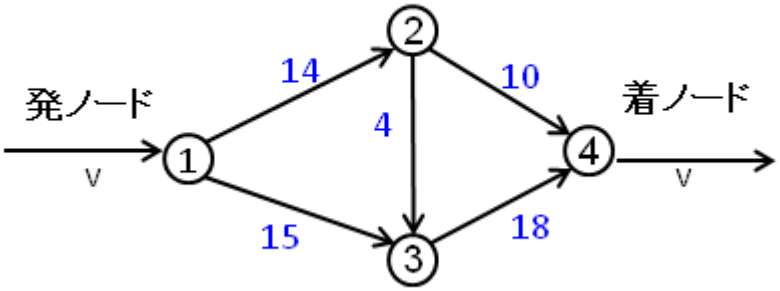
- リンクに容量を与える
  - リンク上のトラヒック量は容量以下　～　制約条件
- ノード1からノード6まで流すことが可能なトラヒック量vを最大化する経路と流量を求める



136

Science and Technology

まずは、簡単なモデルから。



目的関数は以下の通りになる。

$$\max \quad v$$

制約条件を考える。x_{ij} は、(i, j)を通ればトラヒック流量、通らなければ 0 となる。
発ノードに流入するトラヒック量が v で、各リンクの最大トラヒック量が与えられているので。

$$0 \le x_{12} \le 14, \quad 0 \le x_{13} \le 15, \quad 0 \le x_{23} \le 4, \quad 0 \le x_{24} \le 10, \quad 0 \le x_{34} \le 18$$

■ノード 1 でのフロー保存　⇒　入ったトラヒック(流量=v を仮定)は、(1,2)か(1,3)のどち

らかに全て出る。

$$x_{12} + x_{13} = v$$

■ノード２でのフロー保存　⇒　(1,2)から流入したトラヒックは、(2,3)と(2,4)のどちらかに全て出る。

$$x_{12} - x_{23} - x_{24} = 0$$

■ノード 3 でのフロー保存　⇒ (1,3)と(2,3)から流入したトラヒックは、(3,4)から全部出る。

$$x_{13} + x_{23} - x_{34} = 0$$

■ノード４でのフロー保存　⇒ (2,4)と(3,4)から流入したトラヒックの流量和は v。

$$x_{24} + x_{34} = v$$

ただし、ノード 4 でのフロー保存条件は、ノード 1～3 でのフロー保存条件が満たされていれば自動的に成立するので、実際には制約条件に加えなくても良い。

※ 実際に計算して３個の式から４式目が導けることを確認して下さい。

ファイル名を mf-ex1.mod として、emacs 等のエディタで以下を記述します。
単純に打ち込む作業をするのではなく、問題の意味と記述の対応を学んで下さい。

```
/* mf-ex1.mod */

/* Decision variables */
var v ;

var x12 <=14, >=0 ;
var x13 <=15, >=0 ;
var x23 <= 4, >=0 ;
var x24 <=10, >=0 ;
var x34 <=18, >=0 ;

/* Objective function */
maximize TRAFFIC: v ;

/* Constraints */
s.t. NODE1: x12 + x13 = v ;
s.t. NODE2: x12 - x23 - x24 = 0 ;
s.t. NODE3: x13 + x23 - x34 = 0 ;

end ;
```

モデルファイルを GLPK で解いてみます。
$ glpsol△-m△mf-ex1.mod△-o△mf-ex1.out

mf-ex1.out を見てみましょう。Objective: TRAFFIC = 28 (MAXimum) という記述は入っていますか？ 問題は最大流量 v だけを求めることであるため、最大流量 v=28 を与えるときのそれぞれのリンクの流量値 x_{12}=13、x_{13}=15、x_{23}=3、x_{24}=10、x_{34}=18 (x_{12}=14、x_{13}=14、x_{23}=4、x_{24}=10、x_{34}=18 の場合もある)は読みとれますが、経路(３経路あります)に関しては自分で調べる必要があります。

※ 応用練習　3.3.1 の最初にある６ノードのネットワークに対する最大流問題を解いてみましょう。

3.3.2.　文字式による一般化にチャレンジ

最大流問題の文字表記

目的関数　$\max v$

制約条件

$$\sum_{j\in V} x_{ij} - \sum_{j\in V} x_{ji} = v \quad (i = p \text{のとき})$$

$$\sum_{j\in V} x_{ij} - \sum_{j\in V} x_{ji} = 0 \quad (\forall i \neq p, q \in V)$$

$$0 \le x_{ij} \le c_{ij} \quad (\forall (i,j) \in E)$$

発ノードｐから着ノードｑまでの最大流量 ｖ を求める。
目的関数は、ｖ の最大化である。
最初の制約条件は、発ノードであるノードｐでのフロー保存。
二番目の制約条件は、発着ノード以外のノードでのフロー保存。
三番目の制約条件は、フロー量の制約であり、c_{ij} は、各リンクにおける最大容量(capacity)である。
まずは、文字表記をモデルファイルに書き起こします。

```
/* mf-gen.mod */

/* Given parameters */
param N integer, >0;
param p integer, >0;
param q integer, >0;

set V := 1..N;
set E within {V,V};

param capa{E};

/* Decision variables */
var TRAFFIC >= 0;
var x{E} >=0;

/* Objective function */
maximize FLOW: TRAFFIC ;

/* Constraints */
s.t. SOURCE{i in V: i = p && p != q}:
     sum{j in V} (x[i,j]) - sum{j in V} (x[j,i]) = TRAFFIC ;
s.t. INTERNAL{i in V: i != p && i != q && p != q}:
     sum{j in V} (x[i,j]) - sum{j in V} (x[j,i]) = 0 ;
s.t. CAPACITY{(i,j) in E}:
     x[i,j] <= capa[i,j] ;

end;
```

パラメータ N はノード数、p は発ノード、q は着ノード。変数 V を使って、i, j の組合せ範囲が 1～N の整数値であることを示しています。変数 E を使って、容量の二次元配列を定義しています。定義されていないリンクに対しては、容量=0 を代入して使います。制約条件の中の、 p != q は、発ノードと着ノードは違うノードであることを明示しています。

このモデルファイルに与えるデータファイルは次のようになります。

```
/* mf-gen1.dat */

param p := 1;
param q := 4;
param N := 4;

param : E : capa :=
1 1 0
1 2 14
1 3 15
1 4 0
2 1 0
2 2 0
2 3 4
2 4 10
3 1 0
3 2 0
3 3 0
3 4 18
4 1 0
4 2 0
4 3 0
4 4 0
;

end;
```

モデルファイルを GLPK で解いてみます。

$ glpsol△-m△mf-gen.mod△-d△mf-gen1.dat△-o△mf-gen1.out

mf-gen1.out を見てみましょう。Objective: FLOW = 28 (MAXimum) という記述は入っていますか？ 問題は最大流量 v だけを求めることであるため、最大流量 v=28 を与えるときのそれぞれのリンクの流量値 x_{12}=13、x_{13}=15、x_{23}=3、x_{24}=10、x_{34}=18 は読みとれますか？　先ほどは、x_{12}=14、x_{13}=14、x_{23}=4、x_{24}=10、x_{34}=18 と求まったはずですが、最大流量 28 を与える解は、この問題では二通りあるので、どちらが求まってもよいのです。

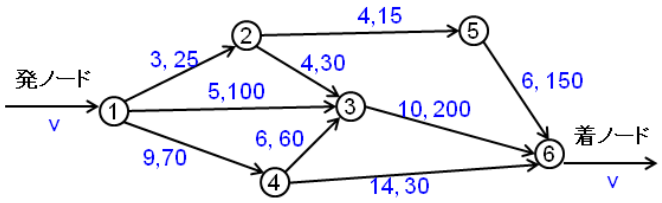
※ 応用練習　3.3.1 の最初にある６ノードのネットワークに対する dat ファイルを書いて最大流問題を解いてみましょう。

3.4.　最小費用流問題

3.4.1.　与えられた問題のみを一生懸命に解いてみる

最小費用流問題

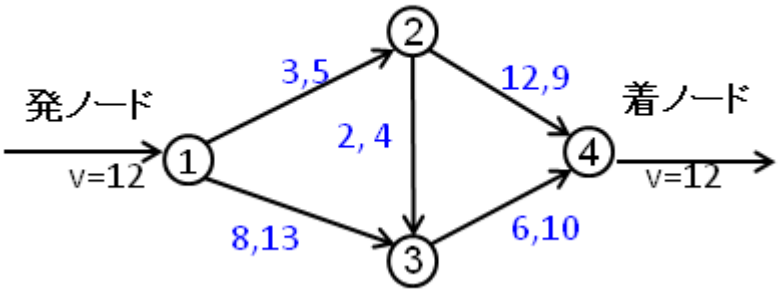
- リンクに距離と容量を与える
  - リンク上のトラヒック量は容量以下　～　制約条件
- ノード1からノード6までトラヒック量v=180が与えられたときに、最小の費用でトラヒックを流す経路と流量を求める
  - 費用＝距離×リンクを通過するトラヒック量



138

Science and Technology

まずは、簡単なモデルから。



目的関数は以下の通りになる。

$$\min \quad 3x_{12} + 8x_{13} + 2x_{23} + 12x_{24} + 6x_{34}$$

制約条件を考える。x_{ij} は、(i, j)を通ればトラヒック流量、通らなければ 0 となる。
発ノードに流入するトラヒック量が v で、各リンクの最大トラヒック量が与えられているので。

$$0 \le x_{12} \le 5, \quad 0 \le x_{13} \le 13, \quad 0 \le x_{23} \le 4, \quad 0 \le x_{24} \le 9, \quad 0 \le x_{34} \le 10$$

■ノード 1 でのフロー保存　⇒　入ったトラヒック(流量=12 を仮定)は、(1,2)か(1,3)のどちらかに全て出る。

$$x_{12} + x_{13} = 12$$

■ノード２でのフロー保存　⇒　(1,2)から流入したトラヒックは、(2,3)と(2,4)のどちらかに全て出る。

$$x_{12} - x_{23} - x_{24} = 0$$

■ノード 3 でのフロー保存　⇒ (1,3)と(2,3)から流入したトラヒックは、(3,4)から全部出る。

$$x_{13} + x_{23} - x_{34} = 0$$

■ノード４でのフロー保存　⇒ (2,4)と(3,4)から流入したトラヒックの流量和は 12。

$$x_{24} + x_{34} = 12$$

ただし、ノード 4 でのフロー保存条件は、ノード 1～3 でのフロー保存条件が満たされていれば自動的に成立するので、実際には制約条件に加えなくても良い。

※　実際に計算して３個の式から４式目が導けることを確認して下さい。

ファイル名を mcf-ex1.mod として、emacs 等のエディタで以下を記述します。
単純に打ち込む作業をするのではなく、問題の意味と記述の対応を学んで下さい。

```
/* mcf-ex1.mod */

/* Decision variables */
var x12 <=5, >=0 ;
var x13 <=13, >=0 ;
var x23 <= 4, >=0 ;
var x24 <= 9, >=0 ;
var x34 <=10, >=0 ;

/* Objective function */
minimize COSTFLOW: 3*x12 + 8*x13 + 2*x23 + 12*x24 + 6*x34 ;

/* Constraints */
s.t. NODE1: x12 + x13 = 12 ;
s.t. NODE2: x12 - x23 - x24 = 0 ;
s.t. NODE3: x13 + x23 - x34 = 0 ;

end ;
```

モデルファイルを GLPK で解いてみます。
$ glpsol△-m△mcf-ex1.mod△-o△mcf-ex1.out

mcf-ex1.out を見てみましょう。Objective: COSTFLOW = 161 (MINimum) という記述は入っていますか？ 問題は最小費用流だけを求めることであるため、最小費用流量 161 を与えるときのそれぞれのリンクの流量 x_{12}=5、x_{13}=7、x_{23}=3、x_{24}=2、x_{34}=10 は読みとれますが、経路(３経路あります)に関しては自分で調べる必要があります。

※　応用練習　3.3.1 の最初にある６ノードのネットワークに対する最小費用流問題を解いてみましょう。

3.4.2.　文字式による一般化にチャレンジ

最小費用流問題の文字表記

目的関数　$\min \sum_{(i,j)\in E} d_{ij}x_{ij}$

制約条件

$$\sum_{j\in V} x_{ij} - \sum_{j\in V} x_{ji} = v \quad (i = p\text{のとき})$$

$$\sum_{j\in V} x_{ij} - \sum_{j\in V} x_{ji} = 0 \quad (\forall i \neq p,q \in V)$$

$$0 \le x_{ij} \le c_{ij} \quad (\forall (i,j) \in E)$$

発ノード p から着ノード q までの流量 v を流す。(i, j)の距離 d_{ij}、(I, j)の容量 c_{ij} とする。
目的関数は、距離×流量の和の最小化である。
最初の制約条件は、発ノードであるノード p でのフロー保存。
二番目の制約条件は、発着ノード以外のノードでのフロー保存。
三番目の制約条件は、フロー量の制約である。

※文字表記をモデルファイルに書き起こしましょう。
※データファイルを書いてみましょう。
ヒント
param capa{E}；
param cost{E}；
と宣言してしまうと、データファイル中で
param：E：capa :=　データの並び；
param: E：cost :=　データの並び；
と記述することになり、実行時に、
mcf-gen1.dat:28: E already defined
の様にエラーとなってしまいます。そのため、別の変数を使って cost の方を宣言してあげる必要があります。
※実行結果は、mcf-ex1.out と一致しましたか？

※応用練習　3.4.1 の最初にある 6 ノードのネットワークに対する dat ファイルを書いて最小費用流問題を解いてみましょう。v=180 を与えます。

4. 整数線形計画法(ILP)にするには
3 まで利用したのは、変数を整数と限定していないため、線形計画問題となる。コスト(距離)や、トラヒック流量等は、整数値と限定されないからである。
整数線形計画法は、変数の定義を
param K integer, >0 ;
set V := 1..K ;
に見られるように、該当変数が整数であることを宣言してあげればよい。

演習7

整数線形計画問題

目的関数 $\max \quad 10x_1 + 12x_2$

制約条件 $2x_1 + 3x_2 \le 30$

$3x_1 + 2x_2 \le 24$

$x_1 = 0,1,\cdots$ (整数値)

$x_2 = 0,1,\cdots$ (整数値)

187

Science and Technology

※ モデルファイルを書いて演習７を解いてみましょう。

x1=0, x2=10 の時に、最大値 120 となります。

5. GLPK でパズルを解いてみよう

5.1. 数独に挑戦

数独（すうどく、Sudoku）とは、3×3 のブロックに区切られた 9×9 の正方形の枠内に 1～9 までの数字を入れるペンシルパズルの一つである。 by Wikipedia

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

ルール

1. 空いているマスに 1～9 のいずれかの数字を入れる。
2. 縦・横の各列及び、太線で囲まれた 3×3 のブロック内に同じ数字が複数入ってはいけない。

初期配置の最小個数

数独の初期配置の最少個数は、17 個である。2012 年 1 月 6 日、アイルランドの数学者 Gary McGuire は「数独においてヒントが 16 個以下のものは解法を持ちえない」ということを証明した。証明にあたっては「hitting-set algorithm」を用いて単純化し、2 年間で 700 万 CPU 時間をかけ、答えにたどり着いた。

5.2. 数独の線形計画法による定式化

数独を一般化すると、n×n のマトリックスに対して $n=m^2$ を満たす m×m のサブマトリックスが構成され、整数 1～m を 1 個ずつ n×n マトリックスの各行、列、サブマトリックスに配置する。

数独は、目的関数を最大化又は最小化するような問題ではないので、目的関数は定義しない。つまり、制約条件を定めていく。

5.2.1. 9×9 の数独の定式化

決定変数 x_{ijk} を次のように定義する。

$$x_{ijk} = \begin{cases} 1 & \text{if } (i,j) \text{ contains integer } k \\ 0 & \text{otherwise} \end{cases}$$

ここで、$i,j,k \in D,\ D = \{1,\cdots,9\}$である。$(i,\ j)$ は、9×9 マトリックスの i 行 j 列を意味する。

G を(i, j, k) 全体集合とする。k は(i, j)が保有する値のことである。従って、G に所属する(i, j, k)に対しては、$x_{ijk} = 1$となる。

以上より、9×9 数独パズルは以下のように定式化できる。
制約条件

$$x_{ijk} = 1, \qquad \forall i,j,k \in G$$

$$\sum_{k=1}^{9} x_{ijk} = 1, \qquad \forall i,j \in D$$

$$\sum_{i=1}^{9} x_{ijk} = 1, \qquad \forall j,k \in D$$

$$\sum_{j=1}^{9} x_{ijk} = 1, \qquad \forall i,k \in D$$

$$\sum_{i=I}^{I+2}\sum_{j=J}^{J+2} x_{ijk} = 1, \qquad \forall k \in G, \qquad I = 1,4,7, \qquad J = 1,4,7$$

$$x_{ijk} \in \{0,1\}, \qquad \forall i,j,k \in D$$

一つ目の制約条件が、数独の全数字を使うということを表している。
二つ目の制約条件が、(i, j) には、k の値候補 1～9 のうちただ一つのみが入ることを表している。
三つめの制約条件が、各行に k の値を持つものはただ一つであることを表している。
四つ目の制約条件が、各列に k の値を持つものはただ一つであることを表している。
五つ目の制約条件が、各 3×3 のサブマトリックスに k の値を持つものはただ一つであることを表している。

モデルファイルは以下のようになる。今まで紹介していないテクニックが使われているので注意して下さい。

```
/* sudoku9x9.mod */

/* Decision variables */
        var x {i in 1..9, j in 1..9, k in 1..9}, binary ;
        /* x [i, j, k] = 1 means cell [i, j] is assigned number k */

/* Initialization */
        param input_problem {1..9, 1..9}, integer, >=0, <=9, default 0 ;
        /* input problem */

        s.t. pre_defined {i in 1..9, j in 1..9, k in 1..9 : input_problem[i, j] != 0} :
                x[i, j, k] = ( if input_problem[i, j] = k then 1 else 0) ;
        /* assign pre-defined number */
```

```
/* No objective function */

/* Constraints */
          s.t. constr_fill {i in 1..9, j in 1..9} : sum{k in 1..9} x[i, j, k] = 1;
          /* constraint #1 : every cell must be filled by exactly one number */

          s.t. constr_col {j in 1..9, k in 1..9} : sum{i in 1..9} x[i, j, k] = 1;
          /* constraint #2 : only one k in each column */

          s.t. constr_row {i in 1..9, k in 1..9} : sum{j in 1..9} x[i, j, k] = 1;
          /* constraint #3 : only one k in each row */

          s.t. constr_sub {I in 1..9 by 3, J in 1..9 by 3, k in 1..9} :
                    sum{i in I..I+2, j in J..J+2} x[i, j, k] = 1;
          /* constraint #4 : only one k in each submatrix */

solve ;

printf "This is the solution of Sudoku 9x9¥n" ;

for {i in 1..9}
{         for {0..0: i = 1 or i = 4 or i = 7}
                    printf " +-------+-------+-------+ ¥n" ;
          for {j in 1..9}
          {         for {0..0: j = 1 or j = 4 or j = 7}
                    printf ( " |" );
                    printf " %d", sum{ k in 1..9} x[i, j, k] * k ;
                    for {0..0: j = 9} printf( " |¥n" );
          }
          for {0..0: i = 9}
                    printf " +-------+-------+-------+ ¥n" ;
}

param TXT, symbolic, := "sudoku9x9.txt" ;     ← ここで書込みファイルを指定！！

printf "This is the solution of Sudoku 9x9¥n" > TXT ;

for {i in 1..9}
{         for {0..0: i = 1 or i = 4 or i = 7}
                    printf " +-------+-------+-------+ ¥n"  >> TXT ;
          for { j in 1..9}
          {         for {0..0: j = 1 or j = 4 or j = 7}
```

```
                        printf ( " |" ) >> TXT ;
                        printf  " %d", sum{ k in 1..9} x[i, j, k] * k >> TXT ;
                        for {0..0: j = 9} printf( "  |¥n" ) >> TXT ;
        }
        for {0..0: i = 9}
                printf  " +-------+-------+-------+ ¥n"   >> TXT ;
}
end;
```

必要となるデータファイルは、例えば

```
/* FILE NAME : sudoku9x9.dat */
/* This is the Sudoku 9x9 problem sample */

data;

param input_problem : 1 2 3 4 5 6 7 8 9 :=
                    1 5 3 . . 7 . . . .
                    2 6 . . 1 9 5 . . .
                    3 . 9 8 . . . . 6 .
                    4 8 . . . 6 . . . 3
                    5 4 . . 8 . 3 . . 1
                    6 7 . . . 2 . . . 6
                    7 . 6 . . . . 2 8 .
                    8 . . . 4 1 9 . . 5
                    9 . . . . 8 . . 7 9
;
end ;
```

として与える。"." が値の入っていない場所を示している。

これは、Initialization の

```
s.t. pre_defined {i in 1..9, j in 1..9, k in 1..9 : input_problem[i, j] != 0} :
                    x[i, j, k] = ( if input_problem[i, j] = k then 1 else 0) ;
```

の行で、x[i,j,k]に 0～9 以外のもの、つまり"."、が入っていたら 0 で初期化を行うという形で利用されている。

mod ファイルでは、画面への出力以外に固定ファイル名 sudoku9x9.txt への出力が毎回行われるように設定されている。不要な場合は、param TXT, の行以下をコメントアウトすると良い。

※一番ヒントの数が少ない問題です。GLPK 使わないで人力で解けたらすごい!!

			8		1			
						4	3	
5								
				7		8		
						1		
	2			3				
6							7	5
		3	4					
			2			6		

5.3.　時間の余った人へ

９×９の数独を拡張して 16×16 の数独の定式化にチャレンジしましょう。

授業の第９回からは、12-102 に戻ります。

以上