

知の探究

4. 最適化 optimization

堀田 敬介

Outline

1. 線形最適化とは？

1. 線形最適化問題(Linear Optimization Problem)
2. 線形最適化問題をソルバー(solver)を利用して解く
 - ✓ Excel solver, gurobi, cplex, python-MIP

2. 様々な最適化問題を線形最適化で解く

1. 輸送問題
2. 最大重みマッチング問題
3. 最短路問題
4. 最大流問題
5. 最小カット問題
6. 最小費用流問題

1. 線形最適化とは？

1. 線形最適化問題(Linear Optimization Problem)

- 線形(1次)等式・不等式系であらわされる条件のもとで, 線形(1次)の目的関数を最大・最小化する形式の最適化問題

$$\begin{array}{ll} \text{min.} & 2x_1 + x_2 + 2x_3 + x_4 + 3x_5 \\ \text{s.t.} & \begin{array}{l} x_1 + 2x_3 + x_5 \geq 5 \\ 9x_1 + 2x_2 + x_4 + 4x_5 \geq 1 \\ x_2 + 5x_3 + x_5 \geq 3 \\ x_1 + 3x_3 + x_5 \geq 2 \\ x_1, x_2, x_3, x_4, x_5 \geq 0 \end{array} \end{array}$$

目的関数 objective function

制約条件 constraints

非負条件 nonnegativity

- 線形計画問題を解くための主な手法 algorithm
 - 単体法 simplex method, G.B.Dantzig(1947)
 - 内点法 interior point method, N.Karmarkar (1984)
 - (楕円体法 ellipsoid method, Yudin, A.S.Nemirovskii(1976), Khachiyan(1979))

1. 線形最適化とは？

2. 線形最適化問題をソルバー(solver)を利用して解く

1. モデルをExcelに記述し, ソルバーを利用して解く
 2. モデルを LP file に記述し, gurobi で解く
 3. モデルを LP file に記述し, cplex で解く
 4. Google Colaboratory と Python-MIP を利用して解く
- etc.

1-2-1. Excelに記述しソルバーで解く

- 準備：Excelソルバーを使える状態にする設定方法

① メニューから[ファイル]-[オプション]を選択

→ [Excelのオプション]d-boxが開く (※d-box = dialog box)

Excelの初期状態では
ソルバーを使えない

The screenshot shows the 'Excelのオプション' (Excel Options) dialog box. The 'アドイン' (Add-ins) tab is selected in the left sidebar. The 'アドイン' (Add-ins) list shows 'Microsoft Solver Add-in' (ソルバー アドイン) with a checkmark. The '設定' (Settings) button at the bottom is highlighted. A secondary 'アドイン' (Add-ins) dialog box is shown, with 'ソルバー アドイン' (Solver Add-in) checked and the 'OK' button highlighted.

② [アドイン]を選択

③ [設定]をクリック
→ [アドイン]d-boxが開く

④ [ソルバーアドイン]にチェック ✓

⑤ [OK]クリック

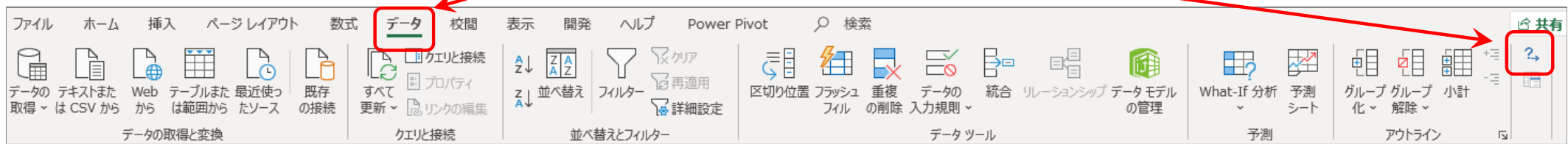
有効なアドイン(A):
☐ Euro Currency Tools
☒ ソルバー アドイン
☒ 分析ツール
☐ 分析ツール - VBA

最適化に関する数学的な手法を用いて、指定された範囲で最善の解を求めます

1-2-1. Excelに記述しソルバーで解く

・ソルバーの起動・設定・実行

起動 = メニューから[データ]-[ソルバー]を選択



→ [ソルバーの
パラメーター]
d-box が開く

The image shows the 'ソルバーのパラメーター' (Solver Parameters) dialog box. It contains fields for '目的セルの設定:(I)' (Set Objective), '目標値:' (To: Of Max Value (M), Min Value (N), Value Of (V)), '変数セルの変更:(B)' (Variable Cells), and '制約条件の対象:(U)' (Subject to the Constraints). There are also buttons for '追加(A)', '変更(C)', '削除(D)', 'すべてリセット(R)', and '読み込み/保存(L)'. At the bottom, there is a '解決方法の選択:(E)' (Select a Solving Method) section with a dropdown menu set to 'GRG 非線形' (Nonlinear GRG) and an 'オプション(P)' (Options) button. A green box highlights the '解決(S)' (Solve) button at the bottom right.

← 目的関数の設定

← 変数(セル)の設定

← 制約条件の設定

← 手法の選択
オプション設定

← 実行(計算開始)

1-2-1. Excelに記述しソルバーで解く

- 例) 線形最適化問題をExcelシートに記述

$$\min. 2x_1 + x_2 + 2x_3 + x_4 + 3x_5$$

$$\text{s.t.} \quad x_1 + 2x_3 + x_5 \geq 5$$

$$9x_1 + 2x_2 + x_4 + 4x_5 \geq 1$$

$$x_2 + 5x_3 + x_5 \geq 3$$

$$x_1 + 3x_3 + x_5 \geq 2$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0$$

$$\begin{aligned} \min. & c^T x \\ \text{s.t.} & Ax \geq b \\ & x \geq 0 \end{aligned}$$

行列・ベクトル
による定式化
の表記

$$c^T = (2 \quad 1 \quad 2 \quad 1 \quad 3)$$

$$A = \begin{pmatrix} 1 & 0 & 2 & 0 & 1 \\ 9 & 2 & 0 & 1 & 4 \\ 0 & 1 & 5 & 0 & 1 \\ 1 & 0 & 3 & 0 & 1 \end{pmatrix}, b = \begin{pmatrix} 5 \\ 1 \\ 3 \\ 2 \end{pmatrix}$$

	A	B	C	D	E	F	G	H		O	P
1	1. LP を解く										
2			x_1	x_2	x_3	x_4	x_5				
3											
4											
5		min	2	1	2	1	3	=	0		数式 [I5] = SUMPRODUCT(C\$3:G\$3, C5:G5)
6		s.t.	1	0	2	0	1	=	0	≧	5
7			9	2	0	1	4	=	0	≧	1
8			0	1	5	0	1	=	0	≧	3
9			1	0	3	0	1	=	0	≧	2
10											
									LHS	RHS	

変数（解）用のセル

定式化を記述した部分

1-2-1. Excelに記述しソルバーで解く

- Excelシート上の内容をソルバーへ設定する

The image shows an Excel spreadsheet and the Solver Parameters dialog box. The spreadsheet contains a linear programming problem with 5 variables (x_1 to x_5) and 4 constraints. The objective is to minimize the value in cell \$I\$5. The constraints are defined by the ranges \$C\$3:\$G\$3, \$I\$6:\$I\$9, and \$K\$6:\$K\$9. The Solver Parameters dialog box is configured with the following settings:

- 目的セルの設定:(I) $\$I\5
- 目標値: ☐ 最大値(M) ☒ 最小値(N) ☐ 指定値:(V) 0
- 変数セルの変更:(E) $\$C\$3:\$G\3
- 制約条件の対象:(L) $\$I\$6:\$I\$9 \geq \$K\$6:\$K\9
- 非負条件: ☒ 制約のない変数を非負数にする(K)
- 解決方法の選択:(E) シンプレックス LP
- 解決方法: シンプレックス LP

Annotations and steps:

- A green box at the top right states: "ソルバーの設定が全て終了した所" (The solver settings are all completed).
- A green box at the bottom left explains the steps to add constraints: "1. [追加]をクリック" (Click [Add]), "2. 制約条件を設定し[OK]クリック" (Set the constraint and click [OK]).
- A red box at the bottom right states: "[シンプレックスLP]を選ぶ" (Select [Simplex LP]).
- A red box at the bottom right states: "[解決]ボタンを押すと求解を開始" (Press the [Solve] button to start solving).

	x_1	x_2	x_3	x_4	x_5
min	2	1	2	1	3
s.t.	1	0	2	0	1
	9	2	0	1	4
	0	1	5	0	1
	1	0	3	0	1

LHS RHS

1-2-1. Excelに記述しソルバーで解く

- 結果がExcelシート上に反映される

求解が終わると、
[ソルバーの結果]d-boxが開き、
シート上に結果が反映される
(※前頁と比較せよ)

線形最適化問題										
	x_1	x_2	x_3	x_4	x_5					
	0.1	0.0	2.4	0.0	0.0					
min	2	1	2	1	3	=	obj. fn	5.1		
st	1	0	2	0	1	=	5.0	≥	5	
	9	2	0	1	4	=	1.0	≥	1	
	0	1	5	0	1	=	12.2	≥	3	
	1	0	3	0	1	=	7.4	≥	2	
								LHS	RHS	

最適解(Optimal Solution)
が見つかった場合の
メッセージ

オプションを設定して
[OK]で終了

ソルバーの結果

ソルバーによって解が見つかりました。すべての制約条件と最適化条件を満たしています。

☒ ソルバーの解の保持
☐ 計算前の値に戻す

☐ ソルバー パラメーターのダイアログに戻る
☐ アウトライン レポート

レポート
解答
感度
条件

OK キャンセル シナリオの保存...

レポート

指定した種類のレポートを作成し、各レポートをブックの各シートに配置します

1-2-2. LP file に記述し gurobi で解く

- 線形最適化問題の定式化(例)

$$\begin{array}{ll}\text{min.} & 2x_1 + x_2 + 2x_3 + x_4 + 3x_5 \\ \text{s.t.} & \begin{array}{l} x_1 + 2x_3 + x_5 \geq 5 \\ 9x_1 + 2x_2 + x_4 + 4x_5 \geq 1 \\ x_2 + 5x_3 + x_5 \geq 3 \\ x_1 + 3x_3 + x_5 \geq 2 \\ x_1, x_2, x_3, x_4, x_5 \geq 0 \end{array}\end{array}$$

目的関数 objective function

制約条件 constraints

非負条件 nonnegativity



- 定式化を LP file 形式で記述

```
minimize
  2 x1 + x2 + 2 x3 + x4 + 3 x5
subject to
  x1 + 2 x3 + x5 >= 5
  9 x1 + 2 x2 + x4 + 4 x5 >= 1
  x2 + 5 x3 + x5 >= 3
  x1 + 3 x3 + x5 >= 2
end
```

目的関数 objective function

制約条件 constraints

※LP file 形式では、非負条件は記述しない
(変数は自動的に全て非負と設定される)

非負条件 nonnegativity

1-2-2. LP file に記述し gurobi で解く

- 定式化をLP file 形式で記述

- ✓ マイドキュメント(K:ドライブ)に専用のフォルダ[LP]を作成する
- ✓ テキストエディタ(TeraPadやメモ帳)を起動する
- ✓ 定式化をLP file 形式で記述する

```
minimize
  2 x1 + x2 + 2 x3 + x4 + 3 x5
subject to
  x1 + 2 x3 + x5 >= 5
  9 x1 + 2 x2 + x4 + 4 x5 >= 1
  x2 + 5 x3 + x5 >= 3
  x1 + 3 x3 + x5 >= 2
end
```

注1) 数値・変数・記号の間に「半角スペース」が必要
(※空白のない文字を1つの単語と認識するため)
注2) 非負条件は記述しない(変数はデフォルトが非負のため). 非負条件のない変数(フリー変数)を使う場合は, 2つの非負変数の差に置き換える. 即ち, フリー変数の x は $x = x_p - x_m$ ($x_p, x_m \geq 0$) と置き換える

注) 「***」は半角英数で好きな名前

- ✓ フォルダ[LP]内に保存する. その際, ファイルの種類を「全て (*.*)」にし, ファイル名&拡張子を半角英数で「***.lp」と記述

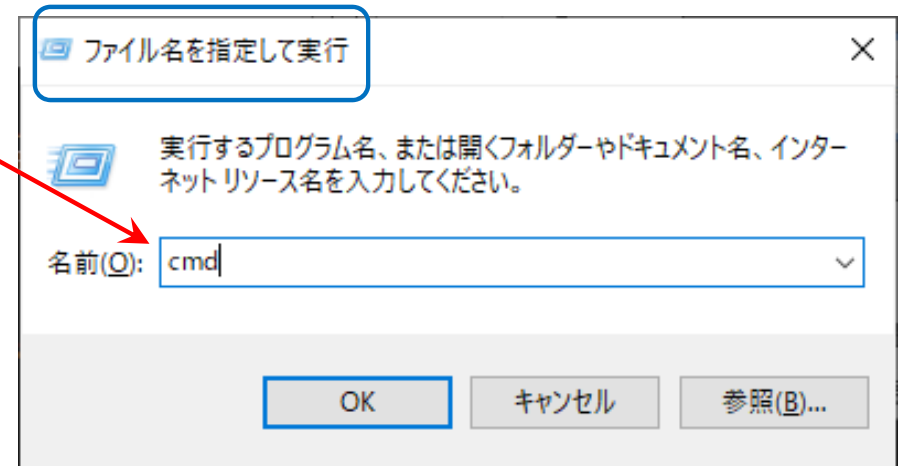
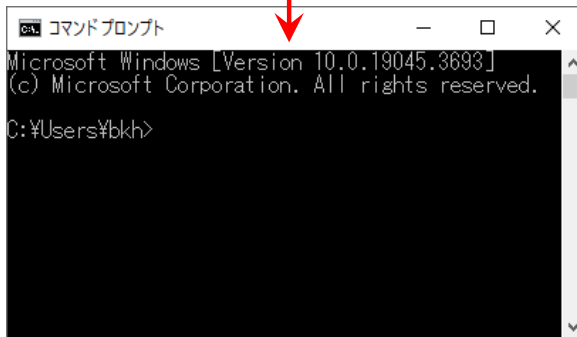
※ファイルの種類を全てに変更し忘れて「テキストファイル(*.txt)」で保存した場合, ファイル名が「***.lp.txt」となるので, ファイル名の変更で最後の部分「.txt」を削除し, 「***.lp」とする

1-2-2. LP file に記述し gurobi で解く

- LP file を gurobi で解く

- 「コマンド プロンプト command prompt」を起動する

- ✓ [Windows]+[R] キーを押し, [ファイル名を指定して実行] を起動する
- ✓ [cmd] と記述し [Enter] キーを押す
→ [コマンドプロンプト] が起動



- 「lpファイル」が保存されているフォルダへ移動する

- ✓ コマンドプロンプト上で [K:] と記述して [Enter] キーを押す
→ Kドライブ へ移動する
- ✓ コマンドプロンプト上で [cd LP] と記述して [Enter] キーを押す
→ [LP]フォルダへ移動する (※ cd = change directory)

1-2-2. LP file に記述し gurobi で解く

- gurobiで解く

※コマンドプロンプト上で[gurobi]と記述して [Enter]キー押す
→ gurobi が起動する

LP file の読込

gurobi> `m=read("xxx.lp")`

問題を解く

gurobi> `m.optimize()`

解の表示

gurobi> `m.printAttr('X')`

gurobi> `m.ObjVal`

解をファイル保存

gurobi> `m.write("xxx.sol")`

gurobiの終了

gurobi> `quit()`

```
コマンドプロンプト

Gurobi Interactive Shell (win32), Version 5.6.3
Copyright (c) 2013, Gurobi Optimization, Inc.
Type "help()" for help

gurobi> m = read("17knw_lp1.lp")
gurobi> m.optimize()
Optimize a model with 4 rows, 5 columns and 13 nonzeros
Presolve removed 2 rows and 3 columns
Presolve time: 0.02s
Presolved: 2 rows, 2 columns, 4 nonzeros

Iteration   Objective       Primal Inf.    Dual Inf.      Time
  0         1.4222222e+00   1.866667e+00   0.000000e+00   0s
  1         5.1111111e+00   0.000000e+00   0.000000e+00   0s

Solved in 1 iterations and 0.03 seconds
Optimal objective  5.111111111e+00
gurobi> m.printAttr('X')

Variable      X
-----
x1            0.111111
x3            2.444444
gurobi> m.ObjVal
5.111111111111112
gurobi> m.write("17knw_lp1.sol")
gurobi> quit()
```

1-2-3. LP file に記述し cplex で解く

- cplexで解く

※コマンドプロンプト上で[cplex]と記述して [Enter]キー押す
→ cplex が起動する

LP file の読込

CPLEX> read xxx.lp

問題の表示

CPLEX> d p a

問題を解く

CPLEX> opt

解の表示

CPLEX> d so v -

解をファイル保存

CPLEX> write xxx.sol

cplex の終了

CPLEX> quit

コマンドプロンプト

```
Welcome to IBM(R) ILOG(R) CPLEX(R) Interactive Optimizer 12.6.2.0  
with Simplex, Mixed Integer & Barrier Optimizers  
5725-A06 5725-A29 5724-Y48 5724-Y49 5724-Y54 5724-Y55 5655-Y21  
Copyright IBM Corp. 1988, 2015. All Rights Reserved.
```

```
Type 'help' for a list of available commands.  
Type 'help' followed by a command name for more  
information on commands.
```

```
CPLEX> read 17knw_lp1.lp  
Problem '17knw_lp1.lp' read.  
Read time = 0.06 sec. (0.00 ticks)
```

```
CPLEX> d p a  
Minimize  
obj: 2 x1 + x2 + 2 x3 + x4 + 3 x5  
Subject To  
c1: x1 + 2 x3 + x5 >= 5  
c2: 9 x1 + 2 x2 + x4 + x5 >= 1  
c3: x2 + 5 x3 + x5 >= 3  
c4: x1 + 3 x3 + x5 >= 2
```

```
Bounds  
All variables are >= 0.
```

```
CPLEX> opt  
Tried aggregator 1 time.  
LP Presolve eliminated 4 rows and 5 columns.  
All rows and columns eliminated.  
Presolve time = 0.02 sec. (0.00 ticks)
```

```
Dual simplex - Optimal: Objective = 5.111111111e+000  
Solution time = 0.02 sec. Iterations = 0 (0)  
Deterministic time = 0.01 ticks (0.35 ticks/sec)
```

```
CPLEX> d so v -  
Variable Name      Solution Value  
x1                  0.111111  
x3                  2.444444  
All other variables in the range 1-5 are 0.
```

```
CPLEX> write 17knw_lp1c.sol  
Solution written to file '17knw_lp1c.sol'.  
CPLEX> quit
```

d p a = display problem all

opt = optimize

d so v = display solution variables

1-2-4. Python-MIP で解く

- Google Colaboratory を開く

- 利用方法(初回)

- (1) google アカウントにログインし, google drive へ移動
- (2) 「新規」－「その他」－「アプリを追加」を選択
- (3) 「Google Colaboratory」を追加

- 利用方法(2回目以降)

- (1) google アカウントにログインし, google drive へ移動
- (2) 「新規」－「その他」－「Google Colaboratory」を選択

- ファイル名は default では [Untitled0.ipynb] となっている. 変更可.

- ファイルは google drive に自動保存される. 一度作成したら, 次回以降は, google drive 内のファイル [***.ipynb] を選択して, 開くことができる

※この拡張子名は IPython Notebook の略で, Jupyter Notebook 専用のファイルということ. IPython は Python を対話的に実行する環境の1つで, Jupyter Notebook とは, それをブラウザ上で動かす実行環境

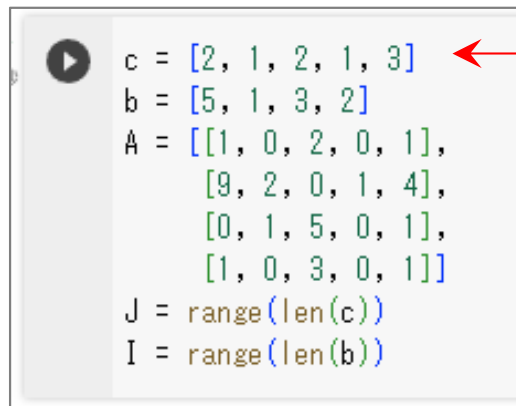
1-2-4. Python-MIP で解く

- Python-mip インストール



[コード]の欄にこのように記入
左の三角ボタンを押して「実行」
(このコードの下に、メッセージが
表示されるので終わるまで待つ)

- 線形最適化問題の記述1(係数の設定)
 - 左上の[+コード]ボタンを押して、次の記述欄([コード]欄)を追加する
 - 以下の通りに記述し、左の三角ボタンを押して「実行」する



例題の線形最適化問題について

$c = [\dots]$: 目的関数の係数ベクトル

$b = [\dots]$: 制約条件の右辺ベクトル

$A = [\dots]$: 制約条件の左辺係数行列

$J = \text{range}(\text{len}(c))$: 列の添え字の範囲

$I = \text{range}(\text{len}(b))$: 行の添え字の範囲

※ $\text{len}(\dots)$ は $\dots$ のサイズlengthを返す関数
を設定しているところ

1-2-4. Python-MIP で解く

• 線形最適化問題の記述2 (定式化と求解)

- 左上の [+コード] をクリックして, 以下のコードを記述する欄を追加する
- そこに以下の通りにコードを記述した後, 左三角ボタンで**実行**する

※ #より右は**コメント**
(プログラムとは関係ない,
人間用の記述)

定式化

```
from mip.model import *  
  
m = Model("lpex1") # モデルの設定  
  
x = [m.add_var(var_type="C", lb=0) for j in J] # 変数宣言: モデル m に変数を追加  
m.objective = minimize(xsum(c[j] * x[j] for j in J)) # 目的関数の設定: モデル m に目的関数を追加  
for i in I:  
    m += xsum(A[i][j] * x[j] for j in J) >= b[i] # 制約条件の設定: モデル m に制約条件を追加  
  
m.optimize() # 最適化 (求解) の実行
```

最適解
と
最適値
の表示

```
if m.status.value==0: # もし, 最適解が求まったなら  
    print("最適解:") # 最適解を表示  
    for j in J:  
        print(" x[" + str(j) + "] = ", x[j].x)  
    print("最適値:", m.objective_value, "=", m.objective) # 目的関数値を表示  
else:  
    # もし, 最適解が求まらなかったなら  
    print("error:最適解は求まりませんでした") # エラーメッセージを表示
```

※ "C" = **continuous** (連続)
連続変数とするということ
※ lb = **lower bound** (下限)
非負条件に該当する.
default は lb=0 なので, 記
述しなくても可

```
最適解:  
x[ 0 ] = 0.11111111111111111  
x[ 1 ] = 0.0  
x[ 2 ] = 2.4444444444444446  
x[ 3 ] = 0.0  
x[ 4 ] = 0.0  
最適値: 5.1111111111111112 = + 2.0var(0) + var(1) + 2.0var(2) + var(3) + 3.0var(4)
```

実行すると, 結果を表示

2. 多様な最適化問題

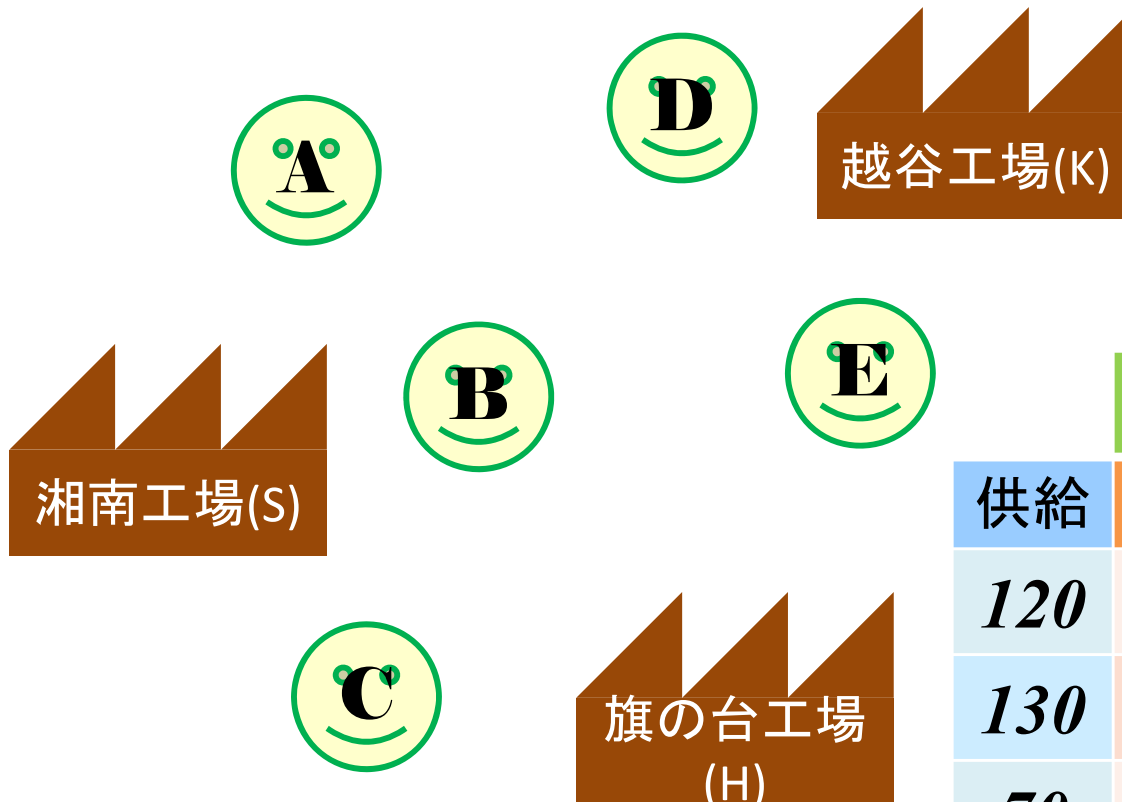
- 様々な最適化問題を線形最適化で解く
 1. 輸送問題
 2. 最大重みマッチング問題
 3. 最短路問題
 4. 最大流問題
 5. 最小カット問題
 6. 最小費用流問題

輸送問題

問) 文教重工には3工場(湘南・越谷・旗の台)あり, 製品を供給(製品生産量)できる

顧客は5人いて, 需要(製品を欲しい量)がある

3工場から5人の顧客それぞれへの単位あたり輸送コストは表の通り
輸送コストが最小となる配送計画をたてよ



- ✓ 工場の供給量
- ✓ 顧客の需要量
- ✓ 工場から顧客へ製品を1単位配送するのにかかる輸送コスト表

		需要	50	80	60	70	40
供給	工場\顧客	A	B	C	D	E	
120	湘南(S)	3	2	4	5	8	
130	越谷(K)	5	6	5	3	2	
70	旗の台(H)	7	3	1	2	3	

輸送問題の定式化

- 線形最適化 Linear Optimization
- 問題のモデル化(定式化)
 - 目的: 輸送コストを最小
 - 条件1: 顧客の需要を満たす
 - 条件2: 工場の出荷量は供給量まで
 - 条件3: 輸送量は非負

目的関数 objective function

制約条件 constraints

非負条件 nonnegativity

変数設定

x_{ij} : 工場 i → 顧客 j への輸送量

ex) $x_{SB} = 30$: 湘南工場(S)から
顧客Bへ製品を30輸送する
その輸送コスト: $2 \times 30 = 60$

	需要	50	80	60	70	40
供給	工場\顧客	A	B	C	D	E
120	湘南(S)	3	2	4	5	8
130	越谷(K)	5	6	5	3	2
70	旗の台(H)	7	3	1	2	3

輸送問題

• 問題の定式化

需要		50	80	60	70	40
供給	工場\顧客	A	B	C	D	E
120	湘南(S)	3	2	4	5	8
130	越谷(K)	5	6	5	3	2
70	旗の台(H)	7	3	1	2	3

minimize

$$\begin{aligned} & 3x_{SA} + 2x_{SB} + 4x_{SC} + 5x_{SD} + 8x_{SE} \\ & + 5x_{KA} + 6x_{KB} + 5x_{KC} + 3x_{KD} + 2x_{KE} \\ & + 7x_{HA} + 3x_{HB} + 1x_{HC} + 2x_{HD} + 3x_{HE} \end{aligned}$$

subject to

$$\begin{aligned} x_{SA} + x_{KA} + x_{HA} &= 50 \\ x_{SB} + x_{KB} + x_{HB} &= 80 \\ x_{SC} + x_{KC} + x_{HC} &= 60 \\ x_{SD} + x_{KD} + x_{HD} &= 70 \\ x_{SE} + x_{KE} + x_{HE} &= 40 \\ x_{SA} + x_{SB} + x_{SC} + x_{SD} + x_{SE} &\leq 120 \\ x_{KA} + x_{KB} + x_{KC} + x_{KD} + x_{KE} &\leq 130 \\ x_{HA} + x_{HB} + x_{HC} + x_{HD} + x_{HE} &\leq 70 \end{aligned}$$

end

目的関数 objective function

制約条件 constraints

非負条件 nonnegativity

※LPファイル形式では書かない

→ ファイル名「ex.lp」で保存(LPファイル)

輸送問題の求解

- Excelで解く(セル記述&ソルバー設定)

ソルバーの設定が
全て終わった状態

	A	B	C	D	E	F	G	H	I	J	K
1	3. 輸送問題										
2		輸送コスト									
3		工場\顧客	A	B	C	D	E				
4		湘南(S)	3	2	4	5	8				
5		越谷(K)	5	6	5	3	2				
6		旗の台(H)	7	3	1	2	3				
7											
8		x_{ij}	A	B	C	D	E		輸送量		供給
9		S							0	≦	120
10		K							0	≦	130
11		H							0	≦	70
12											
13		輸送量	0	0	0	0	0				
14			∥	∥	∥	∥	∥				
15		需要	50	80	60	70	40				
16											
17		[I9] = SUM(C9:G9)									
18		→[I9]をコピーし、[I10:I11]へ貼り付け									
19		[C13] = SUM(C9:C11)									
20		→[C13]をコピーし、[D13:G13]へ貼り付け									
21		[I6] = SUMPRODUCT(C4:G6, C9:G11)									
22											

ソルバーのパラメーター

目的セルの設定:(I)

\$I\$6

目標値: ☐ 最大値(M) ☒ 最小値(N) ☐ 指定値:(V)

0

変数セルの変更:(B)

\$C\$9:\$G\$11

制約条件の対象:(U)

\$C\$13:\$G\$13 = \$C\$15:\$G\$15
\$I\$9:\$I\$11 <= \$K\$9:\$K\$11

追加(A)

変更(C)

削除(D)

すべてリセット(R)

読み込み/保存(L)

☒ 制約のない変数を非負数にする(K)

解決方法の選択:
(E)

シンプレックス LP

オプション(P)

解決方法

滑らかな非線形を示すソルバー問題には GRG 非線形エンジン、線形を示すソルバー問題には LP シンプレックス エンジン、滑らかではない非線形を示すソルバー問題にはエボリューションナリー エンジンを選択してください。

ヘルプ(H)

解決(S)

閉じる(O)

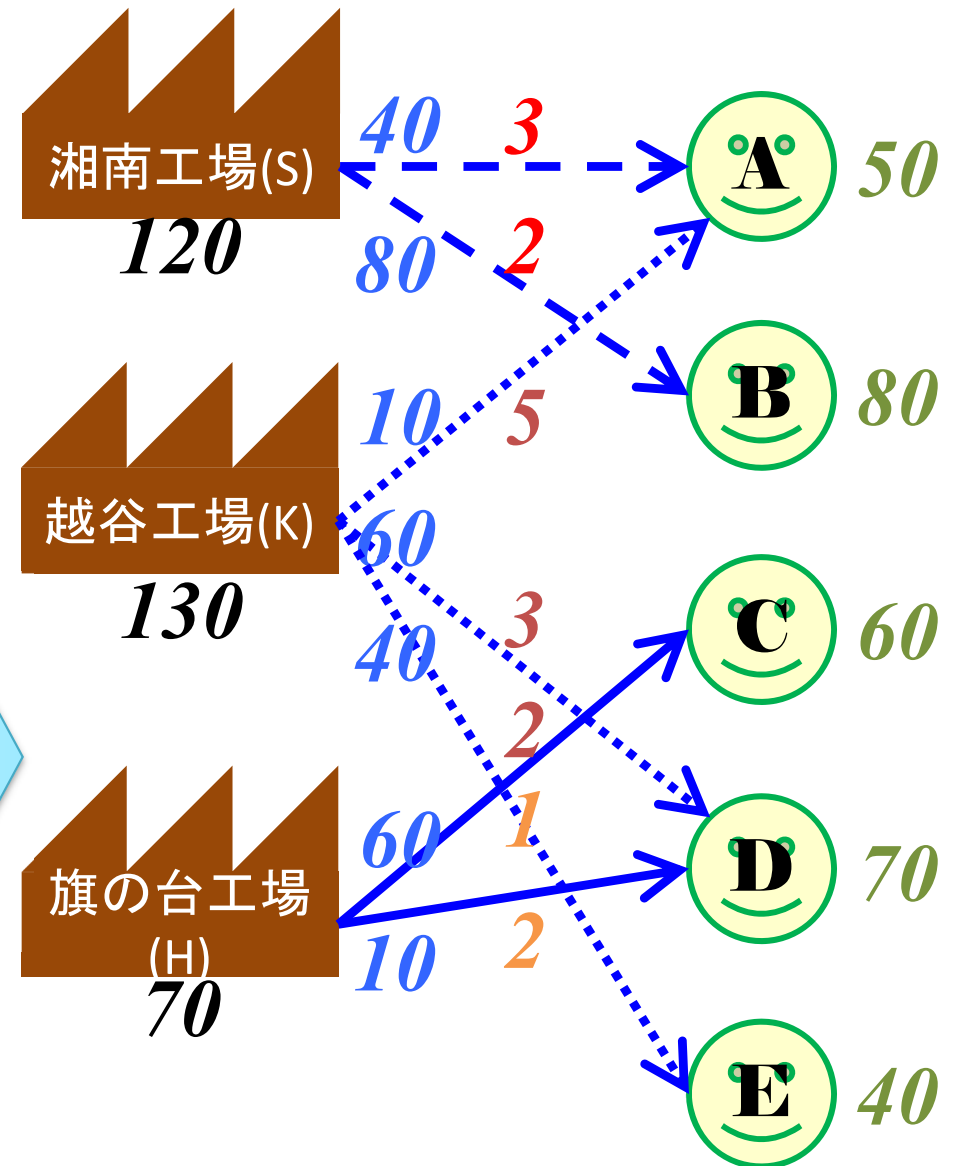
輸送問題の求解

- Excelソルバーで解いた結果

	A	B	C	D	E	F	G	H	I	J	K
1	輸送問題										
2		輸送コスト									
3		工場\顧客	A	B	C	D	E				
4		湘南(S)	3	2	4	5	8				
5		越谷(K)	5	6	5	3	2		総コスト		
6		旗の台(H)	7	3	1	2	3		670		
7											
8		x_{ij}	A	B	C	D	E		輸送量		供給
9		S	40	80	0	0	0		120	≡	120
10		K	10	0	0	60	40		110	≡	130
11		H	0	0	60	10	0		70	≡	70
12											
13		輸送量	50	80	60	70	40				
14			Ⅱ	Ⅱ	Ⅱ	Ⅱ	Ⅱ				
15		需要	50	80	60	70	40				

最適解・最適
の評価・検

最適解・最適値
の評価・検証



輸送問題の求解

- gurobiで解く

Y:¥LP>gurobi [Enter]

LPファイル読込

gurobi> m=read("ex.lp")

最適化(解く)

gurobi> m.optimize()

解の表示

gurobi> m.printAttr('X')

gurobi> m.ObjVal

```
コマンドプロンプト - gurobi
gurobi> m = read("ex.lp")
gurobi> m.optimize()
Optimize a model with 8 rows, 15 columns and 30 nonzeros
Presolve time: 0.00s
Presolved: 8 rows, 15 columns, 30 nonzeros

Iteration   Objective       Primal Inf.    Dual Inf.      Time
     0      5.9000000e+02   7.000000e+01   0.000000e+00     0s
     2      6.7000000e+02   0.000000e+00   0.000000e+00     0s

Solved in 2 iterations and 0.01 seconds
Optimal objective  6.7000000000e+02
gurobi> m.printAttr('X')

Variable      X
-----
xSA            50
xSB            70
xKD            70
xKE            40
xHB            10
xHC            60
gurobi> m.ObjVal
670.0
gurobi>
```

輸送問題の求解

- gurobiで解いた結果

```
Optimal objective 6.700000000e+02
gurobi> m.printAttr('X')

Variable      X
-----
xSA           50
xSB           70
xKD           70
xKE           40
xHB           10
xHC           60

gurobi> m.ObjVal
670.0
```

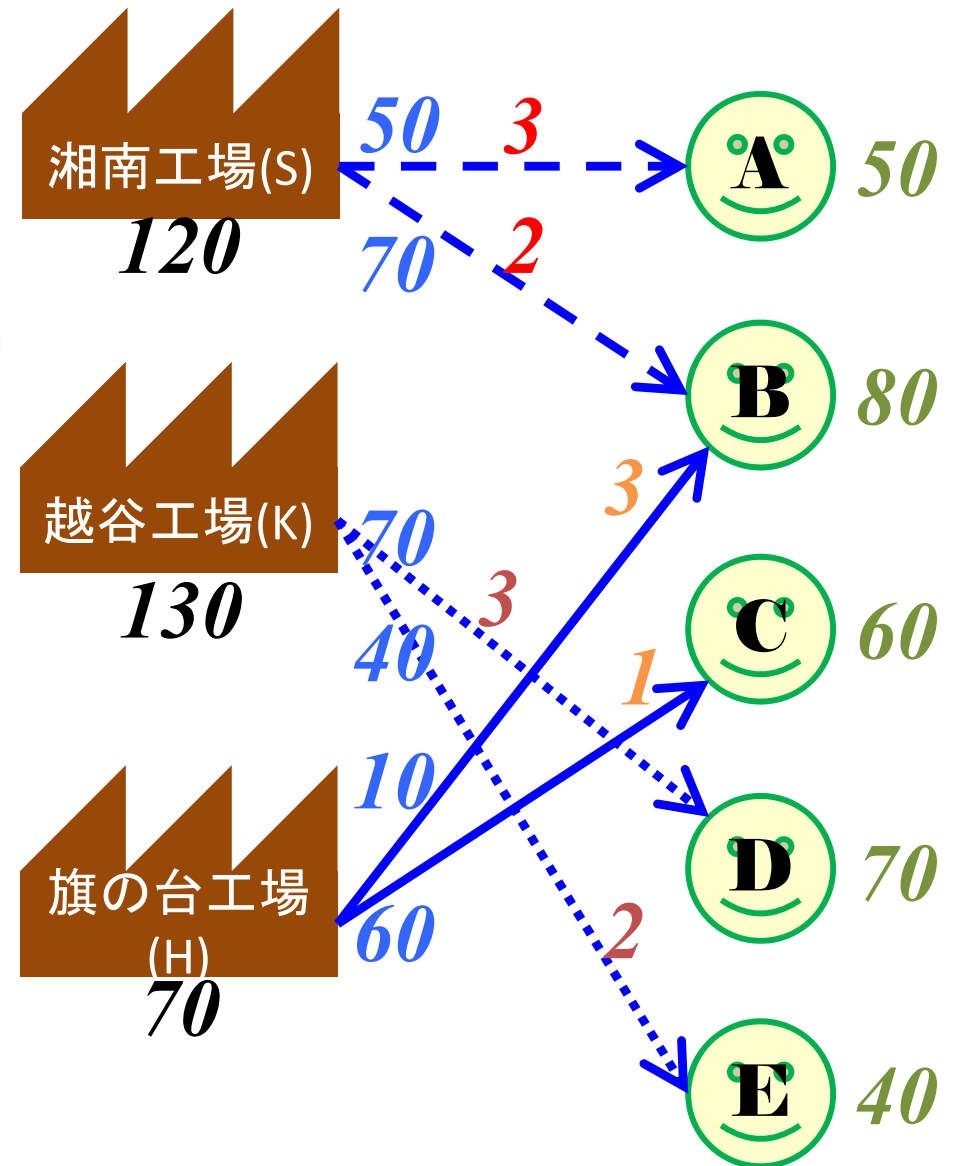
最適解・最適値
の評価・検証

※最適値の表記について

$6.700000000e+02$

$= 6.7 \times 10^2$

$= 670$



輸送問題の求解

- **cplex**で解く

Y:¥LP>**cplex** [Enter]

LPファイル読込

CPLEX> **read ex.lp**

問題の表示

CPLEX> **d p a**

最適化(解く)

CPLEX> **opt**

解の表示

CPLEX> **d so v -**

```
コマンドプロンプト - cplex
C:\> read ex.lp
Problem "ex.lp" read.
Presolve time = 0.00 sec. (0.00 ticks)
CPLEX> d p a
Minimize
  obj: 3 xSA + 2 xSB + 4 xSC + 5 xSD + 8 xSE + 5 xKA + 6 xKB + 5 xKC + 3 xKD
        + 2 xKE + 7 xHA + 3 xHB + xHC + 2 xHD + 3 xHE
Subject To
  c1: xSA + xKA + xHA = 50
  c2: xSB + xKB + xHB = 80
  c3: xSC + xKC + xHC = 60
  c4: xSD + xKD + xHD = 70
  c5: xSE + xKE + xHE = 40
  c6: xSA + xSB + xSC + xSD + xSE <= 120
  c7: xKA + xKB + xKC + xKD + xKE <= 130
  c8: xHA + xHB + xHC + xHD + xHE <= 70
Bounds
  All variables are >= 0.
CPLEX> opt
Tried aggregator 1 time.
No LP presolve or aggregator reductions.
Presolve time = -0.00 sec. (0.01 ticks)

Iteration log . . .
Iteration:    1    Dual objective      =          160.000000

Dual simplex - Optimal: Objective = 6.7000000000e+002
Solution time =    0.00 sec. Iterations = 7 (0)
Deterministic time = 0.01 ticks (13.28 ticks/sec)

CPLEX> d so v -
Variable Name      Solution Value
xSA                40.000000
xSB                80.000000
xKA                10.000000
xKD                60.000000
xKE                40.000000
xHC                60.000000
xHD                10.000000
All other variables in the range 1-15 are 0.
CPLEX>
```

d p a = display problem all

opt = optimize

d so v = display solution variables

輸送問題の求解

- **cplex**で解いた結果

```
Dual simplex - Optimal: Objective = 6.7000000000e+002  
Solution time = 0.00 sec. Iterations = 7 (0)  
Deterministic time = 0.01 ticks (13.28 ticks/sec)
```

```
CPLEX> d so v -
```

Variable Name	Solution Value
xSA	40.000000
xSB	80.000000
xKA	10.000000
xKD	60.000000
xKE	40.000000
xHC	60.000000
xHD	10.000000

```
All other variables in the range 1-15 are 0.
```

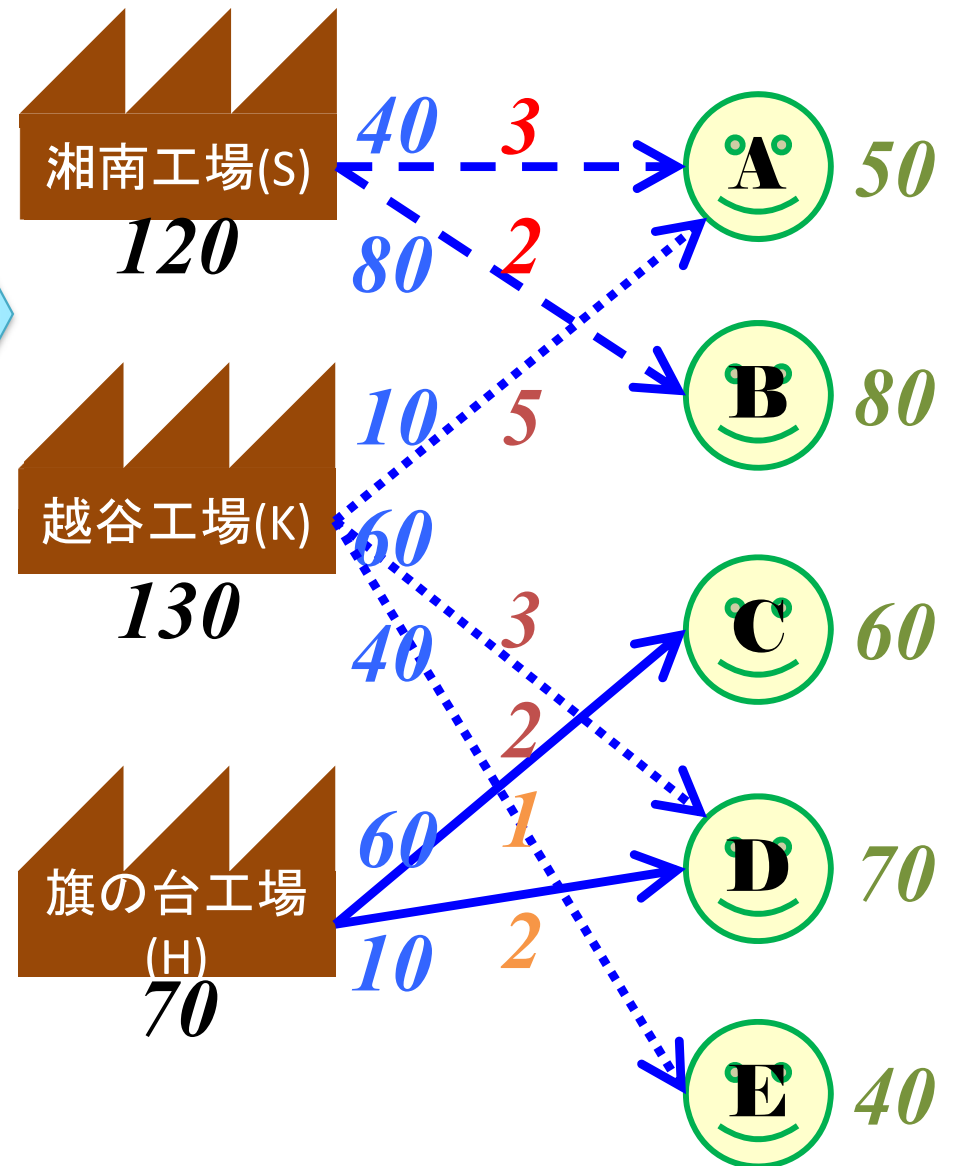
最適解・最適値
の評価・検証

※最適値の表記について

$6.7000000000e+02$

$= 6.7 \times 10^2$

$= 670$



輸送問題の求解

- Python-MIP で解く
 - Python-mip インストール



[コード]の欄にこのように記入
左の三角ボタンを押して「実行」
(このコードの下に、メッセージが
表示されるので終わるまで待つ)

– 輸送問題の記述1 (係数の設定)

```
d = [50, 80, 60, 70, 40] # 需要
s = [120, 130, 70] # 供給
C = [[3, 2, 4, 5, 8], # 輸送コスト
      [5, 6, 5, 3, 2],
      [7, 3, 1, 2, 3]]
J = range(len(d))
I = range(len(s))
```

輸送問題の最適化について

$d = [\dots]$: 顧客の需要ベクトル

$s = [\dots]$: 工場の供給ベクトル

$C = [\dots]$: 輸送コスト行列

$J = \text{range}(\text{len}(d))$: 列の添え字の範囲

$I = \text{range}(\text{len}(s))$: 行の添え字の範囲

※ $\text{len}(\dots)$ は $\dots$ のサイズlengthを返す関数を設定しているところ

輸送問題の求解

– 輸送問題の記述2(定式化)

定式化

```
from mip.model import *

m = Model("trex1") # モデルの設定 (輸送問題)

x = [[m.add_var(var_type="C", lb=0) for j in J] for i in I] # 変数宣言: 輸送量
m.objective = minimize(xsum(C[i][j] * x[i][j] for i in I for j in J)) # 目的関数: 輸送コスト最小化
for j in J:
    m += xsum(x[i][j] for i in I) == d[j] # 制約条件1: 需要
for i in I:
    m += xsum(x[i][j] for j in J) <= s[i] # 制約条件2: 供給

m.optimize() # 最適化 (求解) の実行

if m.status.value==0: # もし, 最適解が求まったなら
    print("最適解:") # 最適解を表示
    for i in I:
        for j in J:
            print(" x[",i,"","j,""] = ", x[i][j].x)

    print(" 目的関数値: ", m.objective_value, "=", m.objective) # 目的関数値を表示
else:
    # もし, 最適解が求まらなかったなら
    print(" error: 最適解は求まりませんでした") # エラーメッセージを表示
```

最適解
と
最適値
の表示

<結果>

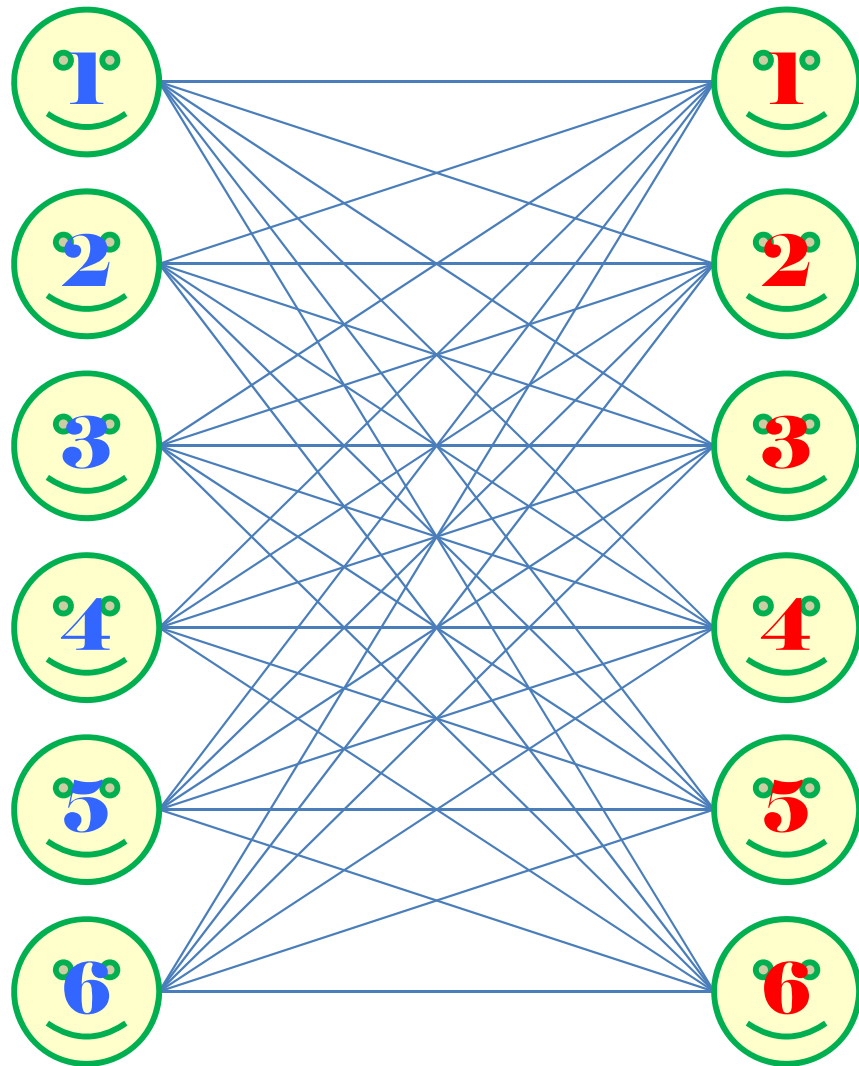
最適解:

```
x[ 0 , 0 ] = 40.0
x[ 0 , 1 ] = 80.0
x[ 0 , 2 ] = 0.0
x[ 0 , 3 ] = 0.0
x[ 0 , 4 ] = 0.0
x[ 1 , 0 ] = 10.0
x[ 1 , 1 ] = 0.0
x[ 1 , 2 ] = 0.0
x[ 1 , 3 ] = 60.0
x[ 1 , 4 ] = 40.0
x[ 2 , 0 ] = 0.0
x[ 2 , 1 ] = 0.0
x[ 2 , 2 ] = 60.0
x[ 2 , 3 ] = 10.0
x[ 2 , 4 ] = 0.0
```

目的関数値: 670.0 = + 3.0var(0) + 2.0var(1) + 4.0var(2) + 5.0var(3) + 8.0var(4) + 5.0var(5) + 6.0var(6) + 5.0var(7) + 3.0var(8) + 2.0var(9) + 7.0var(10) + 3.0var(11) + var(12) + 2.0var(13) + 3.0var(14)

最大重みマッチング

問) 6人の男女がいて、ペアを組む. 互いにペアを組む場合の相性を数値化した. 相性が最大になるマッチングを求めたい



ペアの相性(重み)

w_{ij}	1	2	3	4	5	6
1	3	1	2	5	6	4
2	1	3	5	4	6	2
3	3	6	1	5	4	2
4	4	6	3	2	5	1
5	1	6	2	5	4	3
6	3	6	5	1	2	4

最大重みマッチングの定式化

- 0-1整数最適化 0-1 Integer Optimization
- 問題のモデル化(定式化)

- 目的: 重み和の最大化
- 条件1: 男が組む相手は1人以下
- 条件2: 女が組む相手は1人以下

- 変数設定

- 0-1変数 $x_{ij} = \begin{cases} 1 & \dots \text{枝}(i,j) \text{を使う} \\ 0 & \dots \text{枝}(i,j) \text{使わない} \end{cases}$

ペアの相性(重み)

w_{ij}	1	2	3	4	5	6
1	3	1	2	5	6	4
2	1	3	5	4	6	2
3	3	6	1	5	4	2
4	4	6	3	2	5	1
5	1	6	2	5	4	3
6	3	6	5	1	2	4

最大重みマッチングの定式化

- 0-1整数最適化 0-1 Integer Optimization
- 問題のモデル化(定式化)

maximize

$$\begin{aligned} & 3x_{11} + 1x_{12} + 2x_{13} + 5x_{14} + 6x_{15} + 4x_{16} \\ & + 1x_{21} + 3x_{22} + 5x_{23} + 4x_{24} + 6x_{25} + 2x_{26} \\ & + \dots \\ & + 3x_{61} + 6x_{62} + 5x_{63} + 1x_{64} + 2x_{65} + 4x_{66} \end{aligned}$$

subject to

$$\begin{aligned} & x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} \leq 1 \\ & \dots \\ & x_{61} + x_{62} + x_{63} + x_{64} + x_{65} + x_{66} \leq 1 \end{aligned} \quad \left. \vphantom{\begin{aligned} & x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} \leq 1 \\ & \dots \\ & x_{61} + x_{62} + x_{63} + x_{64} + x_{65} + x_{66} \leq 1 \end{aligned}} \right\} \text{条件1}$$

$$\begin{aligned} & x_{11} + x_{21} + x_{31} + x_{41} + x_{51} + x_{61} \leq 1 \\ & \dots \\ & x_{16} + x_{26} + x_{36} + x_{46} + x_{56} + x_{66} \leq 1 \end{aligned} \quad \left. \vphantom{\begin{aligned} & x_{11} + x_{21} + x_{31} + x_{41} + x_{51} + x_{61} \leq 1 \\ & \dots \\ & x_{16} + x_{26} + x_{36} + x_{46} + x_{56} + x_{66} \leq 1 \end{aligned}} \right\} \text{条件2}$$

$$x_{ij} \in \{0, 1\} \quad (i=1, \dots, 6, j=1, \dots, 6)$$

ペアの相性(重み)

w_{ij}	1	2	3	4	5	6
1	3	1	2	5	6	4
2	1	3	5	4	6	2
3	3	6	1	5	4	2
4	4	6	3	2	5	1
5	1	6	2	5	4	3
6	3	6	5	1	2	4

最大重みマッチングの求解

- Excelソルバーで解く(セル記述)

[illegible]

最大重みマッチングの求解

- Excelで解く
(ソルバー設定)

The image shows the Excel Solver interface with the Solver Parameters dialog box open. The main dialog has the following settings:

- 目的セルの設定:(I)
- 目標値: ☒ 最大値(M) ☐ 最小値(N) ☐ 指定値:(V)
- 変数セルの変更:(B)
- 制約条件の対象:(U)
- 解決方法の選択:
- ボタン: 追加(A), 変更(C), 削除(D), すべてリセット(R), 読み込み/保存(L), オプション(P), 解決(S), 閉じる(O)

Two sub-dialogs for adding constraints are also shown:

制約条件の追加 (Top):

- セル参照:(E)
- 制約条件:(N)
- 比較演算子:
- ボタン: OK, 追加(A), キャンセル(C)

制約条件の追加 (Bottom):

- セル参照:(E)
- 制約条件:(N)
- 比較演算子:
- ボタン: OK, キャンセル(C)

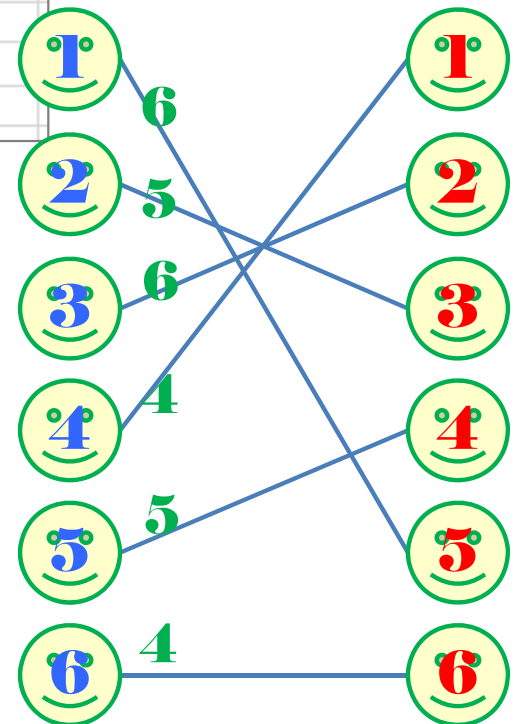
最大重みマッチングの求解

Excelソルバーで解いた結果

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	完全2部グラフの最大重みマッチング																			
2																				
3		重み								0-1変数										
4		w_{ij}	1	2	3	4	5	6		x_{ij}	1	2	3	4	5	6		和		
5		1	3	1	2	5	6	4		1	0	0	0	0	1	0		1	≦	1
6		2	1	3	5	4	6	2		2	0	0	1	0	0	0		1	≦	1
7		3	3	6	1	5	4	2		3	0	1	0	0	0	0		1	≦	1
8		4	4	6	3	2	5	1		4	1	0	0	0	0	0		1	≦	1
9		5	1	6	2	5	4	3		5	0	0	0	1	0	0		1	≦	1
10		6	3	6	5	1	2	4		6	0	0	0	0	0	1		1	≦	1
11																				
12										和	1	1	1	1	1	1		重み和		
13										IIA	IIA	IIA	IIA	IIA	IIA	IIA		30		
14										1	1	1	1	1	1	1				

最適解・最適値
の評価・検証

Objective Value:
6+5+6+4+5+4=30



最大重みマッチング

- gurobi & cplex で
解く準備

– lpファイル

[mwm_ex1.lp]

binary変数(0-1変数)設定

maximize

$3x_{11} + 1x_{12} + 2x_{13} + 5x_{14} + 6x_{15} + 4x_{16}$
 $+ 1x_{21} + 3x_{22} + 5x_{23} + 4x_{24} + 6x_{25} + 2x_{26}$
 $+ 3x_{31} + 6x_{32} + 1x_{33} + 5x_{34} + 4x_{35} + 2x_{36}$
 $+ 4x_{41} + 6x_{42} + 3x_{43} + 2x_{44} + 5x_{45} + 1x_{46}$
 $+ 1x_{51} + 6x_{52} + 2x_{53} + 5x_{54} + 4x_{55} + 3x_{56}$
 $+ 3x_{61} + 6x_{62} + 5x_{63} + 1x_{64} + 2x_{65} + 4x_{66}$

subject to

$x_{11} + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} \leq 1$
 $x_{21} + x_{22} + x_{23} + x_{24} + x_{25} + x_{26} \leq 1$
 $x_{31} + x_{32} + x_{33} + x_{34} + x_{35} + x_{36} \leq 1$
 $x_{41} + x_{42} + x_{43} + x_{44} + x_{45} + x_{46} \leq 1$
 $x_{51} + x_{52} + x_{53} + x_{54} + x_{55} + x_{56} \leq 1$
 $x_{61} + x_{62} + x_{63} + x_{64} + x_{65} + x_{66} \leq 1$

条件1

$x_{11} + x_{21} + x_{31} + x_{41} + x_{51} + x_{61} \leq 1$
 $x_{12} + x_{22} + x_{32} + x_{42} + x_{52} + x_{62} \leq 1$
 $x_{13} + x_{23} + x_{33} + x_{43} + x_{53} + x_{63} \leq 1$
 $x_{14} + x_{24} + x_{34} + x_{44} + x_{54} + x_{64} \leq 1$
 $x_{15} + x_{25} + x_{35} + x_{45} + x_{55} + x_{65} \leq 1$
 $x_{16} + x_{26} + x_{36} + x_{46} + x_{56} + x_{66} \leq 1$

条件2

binary

$x_{11} x_{12} x_{13} x_{14} x_{15} x_{16}$
 $x_{21} x_{22} x_{23} x_{24} x_{25} x_{26}$
 $x_{31} x_{32} x_{33} x_{34} x_{35} x_{36}$
 $x_{41} x_{42} x_{43} x_{44} x_{45} x_{46}$
 $x_{51} x_{52} x_{53} x_{54} x_{55} x_{56}$
 $x_{61} x_{62} x_{63} x_{64} x_{65} x_{66}$

end

最大重みマッチングの求解

- gurobiで解く

```
gurobi> m = read('mwm_ex1.lp')
Read LP format model from file mwm_ex1.lp
Reading time = 0.00 seconds
: 12 rows, 36 columns, 72 nonzeros
gurobi> m.optimize()
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)
Thread count: 10 physical cores, 20 logical processors, using up to 20 threads
Optimize a model with 12 rows, 36 columns and 72 nonzeros
Model fingerprint: 0xdb4f615a
Variable types: 0 continuous, 36 integer (36 binary)
Coefficient statistics:
  Matrix range    [1e+00, 1e+00]
  Objective range [1e+00, 6e+00]
  Bounds range    [1e+00, 1e+00]
  RHS range       [1e+00, 1e+00]
Found heuristic solution: objective 17.0000000
Presolve time: 0.00s
Presolved: 12 rows, 36 columns, 72 nonzeros
Variable types: 0 continuous, 36 integer (36 binary)
Found heuristic solution: objective 24.0000000

Root relaxation: objective 3.000000e+01, 6 iterations, 0.00 seconds (0.00 work units)

   Nodes      |   Current Node   |   Objective Bounds   |   Work
  Expl Unexpl |  Obj  Depth IntInf | Incumbent  BestBd  Gap   | It/Node Time
*    0       0 |                0 | 30.0000000  30.00000  0.00% | -     0s

Explored 1 nodes (6 simplex iterations) in 0.02 seconds (0.00 work units)
Thread count was 20 (of 20 available processors)

Solution count 3: 30 24 17

Optimal solution found (tolerance 1.00e-04)
Best objective 3.000000000000e+01, best bound 3.000000000000e+01, gap 0.0000%
```

最大重みマッチングの求解

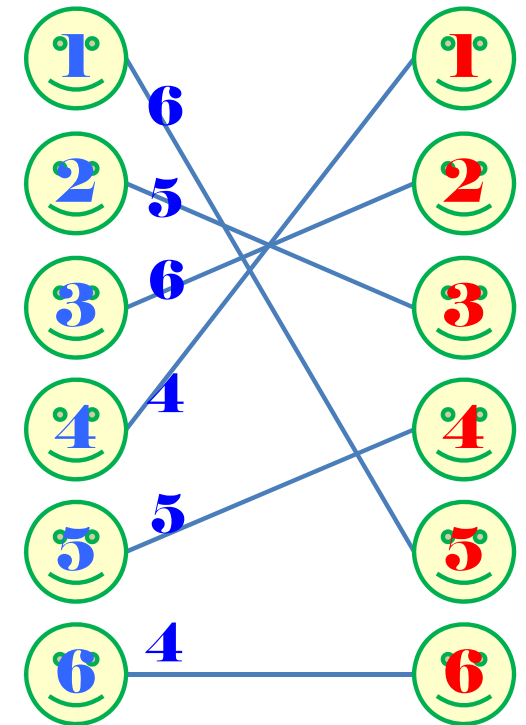
- gurobiで解いた結果

```
gurobi> m.printAttr('X')  
  
Variable      X  
-----  
x15           1  
x23           1  
x32           1  
x41           1  
x54           1  
x66           1  
gurobi> m.ObjVal  
30.0
```

ペアの相性(重み)

w_{ij}	1	2	3	4	5	6
1	3	1	2	5	6	4
2	1	3	5	4	6	2
3	3	6	1	5	4	2
4	4	6	3	2	5	1
5	1	6	2	5	4	3
6	3	6	5	1	2	4

最適解・最適値
の評価・検証



Objective Value:
6+5+6+4+5+4=30

最大重みマッチング

- cplexで解く

```
CPLEX> read mwm_ext1.lp
Problem mwm_ext1.lp read.
Read time = 0.02 sec. (0.00 ticks)
CPLEX> opt
Version identifier: 20.1.0.0 | 2020-11-10 | 9bedb6d68
Found incumbent of value 0.000000 after 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
Reduced MIP has 12 rows, 36 columns, and 72 nonzeros.
Reduced MIP has 36 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.04 ticks)
Probing time = 0.00 sec. (0.02 ticks)
Tried aggregator 1 time.
Detecting symmetries...
Reduced MIP has 12 rows, 36 columns, and 72 nonzeros.
Reduced MIP has 36 binaries, 0 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.05 ticks)
Probing time = 0.00 sec. (0.02 ticks)
Clique table members: 12.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 20 threads.
Root relaxation solution time = 0.00 sec. (0.03 ticks)

      Nodes
      Node Left   Objective  IInf  Best Integer    Cuts/
                                     Best Bound   ItCnt   Gap
*      0+    0                0.0000    126.0000
*      0+    0                17.0000    126.0000      641.18%
*      0+    0                23.0000    126.0000      447.83%
*      0    0      integral    0        30.0000    30.0000      12    0.00%
Elapsed time = 0.02 sec. (0.22 ticks, tree = 0.00 MB, solutions = 4)

Root node processing (before b&c):
  Real time                =    0.02 sec. (0.22 ticks)
Parallel b&c, 20 threads:
  Real time                =    0.00 sec. (0.00 ticks)
  Sync time (average)      =    0.00 sec.
  Wait time (average)      =    0.00 sec.
-----
Total (root+branch&cut) =    0.02 sec. (0.22 ticks)

Solution pool: 4 solutions saved.

MIP - Integer optimal solution: Objective = 3.0000000000e+01
Solution time =    0.02 sec. Iterations = 12 Nodes = 0
Deterministic time = 0.22 ticks (14.63 ticks/sec)
```

最大重みマッチングの求解

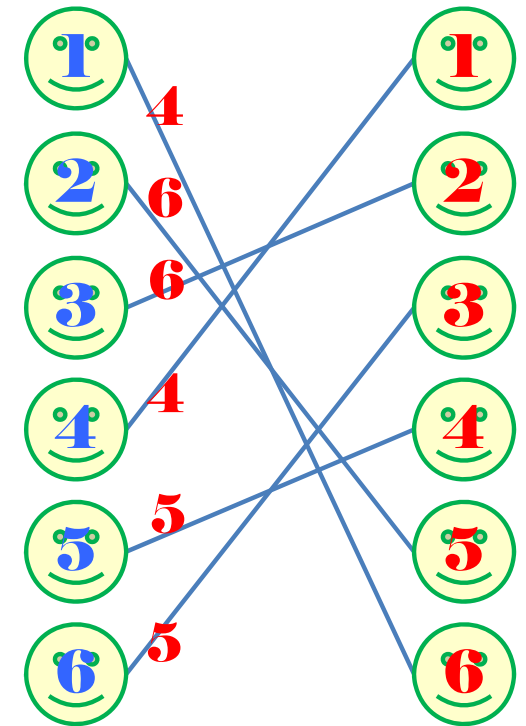
- **cplex**で解いた結果

```
CPLEX> d so v -  
Incumbent solution  
Variable Name      Solution Value  
x16                1.000000  
x25                1.000000  
x32                1.000000  
x41                1.000000  
x54                1.000000  
x63                1.000000  
All other variables in the range 1-36 are 0.
```

ペアの相性(重み)

w_{ij}	1	2	3	4	5	6
1	3	1	2	5	6	4
2	1	3	5	4	6	2
3	3	6	1	5	4	2
4	4	6	3	2	5	1
5	1	6	2	5	4	3
6	3	6	5	1	2	4

最適解・最適値
の評価・検証



Objective Value:
 $4+6+6+4+5+5=30$

最大重みマッチングの求解

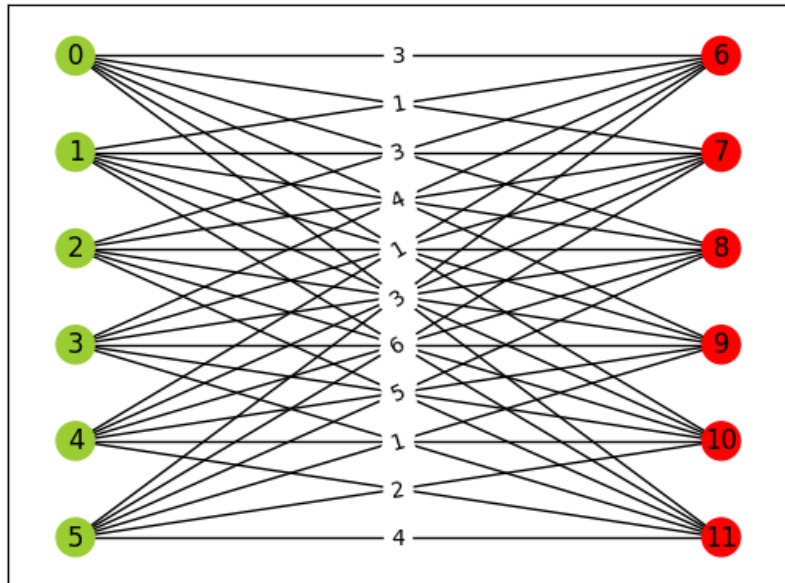
- Python-MIP で解く
 - Python-mip インストール

```
pip install mip
```

– MWMの記述1

- グラフ定義
- 係数設定

```
nodes: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
edges: [(0, 6), (0, 7), (0, 8), (0, 9), (0, 10), (0, 11), (1, 6), (1, 7), (1,
```



```
%matplotlib inline
import matplotlib.pyplot as plt
import networkx as nx

male = female = 6    # male=男数, female=女数
I = J = range(male)  # 添え字範囲設定
G = nx.complete_bipartite_graph(male, female) # 完全2部グラフ K_{6,6}
print("nodes: ", G.nodes()) # 点集合表示 (確認)
print("edges: ", G.edges()) # 枝集合表示 (確認)

W = [[3, 1, 2, 5, 6, 4], # 枝の重み行列 W 作成
      [1, 3, 5, 4, 6, 2],
      [3, 6, 1, 5, 4, 2],
      [4, 6, 3, 2, 5, 1],
      [1, 6, 2, 5, 4, 3],
      [3, 6, 5, 1, 2, 4]]

for i, j in G.edges(): # K_{6,6} なので, i=0..5, j=6..11 であることに注意
    G.adj[i][j]['weight'] = W[i][j-6] # 枝に重み'weight'を付加 (値は重み行列)

pos = {0:(0,3), 1:(0,2), 2:(0,1), 3:(0,0), 4:(0,-1), 5:(0,-2), # 各点の位置図
        6:(6,3), 7:(6,2), 8:(6,1), 9:(6,0), 10:(6,-1), 11:(6,-2)}

ncol = [] # 点の色用ベクトル
for i in range(G.number_of_nodes()):
    ncol.append('yellowgreen')
for i in range(6, 12):
    ncol[i] = 'red'
edge_labels = nx.get_edge_attributes(G, 'weight') # 枝の属性の'weight'を取得

nx.draw_networkx_nodes(G, pos, node_color=ncol) # 描画: 点, 点の位置, 点の色
nx.draw_networkx_labels(G, pos)                # 描画: 点のラベル
nx.draw_networkx_edges(G, pos)                 # 描画: 枝
nx.draw_networkx_edge_labels(G, pos, edge_labels) # 描画: 枝の重み
plt.show()
```

最大重みマッチングの求解

– MWMの記述2(定式化)

定式化

最適解
と
最適値
の表示

```
from mip.model import *

m = Model("mwmex1") # モデルの設定 (最大マッチング問題)

x = [[m.add_var(var_type="B", lb=0, ub=1) for j in J] for i in I] # 変数宣言: マッチング枝[0/1]
m.objective = maximize(xsum(W[i][j] * x[i][j] for i in I for j in J)) # 目的関数: 重み最大化
for j in J:
    m += xsum(x[i][j] for i in I) <= 1 # 制約条件1: 男が組む相手は1人以下
for i in I:
    m += xsum(x[i][j] for j in J) <= 1 # 制約条件2: 女が組む相手は1人以下

m.optimize() # 最適化 (求解) の実行

if m.status.value==0: # もし、最適解が求まったなら
    print("最適解:") # 最適解を表示
    for i in I:
        for j in J:
            if x[i][j].x==1:
                print(" x[" + i + "," + j + "] = ", x[i][j].x)
    print(" 目的関数値: ", m.objective_value, "=", m.objective) # 目的関数値を表示
else: # もし、最適解が求まらなかったなら
    print(" error: 最適解は求まりませんでした") # エラーメッセージを表示
```

<結果>



最適解:

```
x[ 0 , 4 ] = 1.0
x[ 1 , 2 ] = 1.0
x[ 2 , 3 ] = 1.0
x[ 3 , 0 ] = 1.0
x[ 4 , 1 ] = 1.0
x[ 5 , 5 ] = 1.0
```

目的関数値: 30.0 = + 3.0var(0) + var(1) + 2.0var(2) + 5.0var(3) + 6.0var(4) + 4.0var(5) + var(6) + 3.0var(7) + 5.0var(8) + 4.0var(

※注: Python-MIPを利用して0-1整数線形最適化に定式化して解いているが, NetworkX の maximum_matching(G) 等で解いても良い

最大重みマッチングの求解

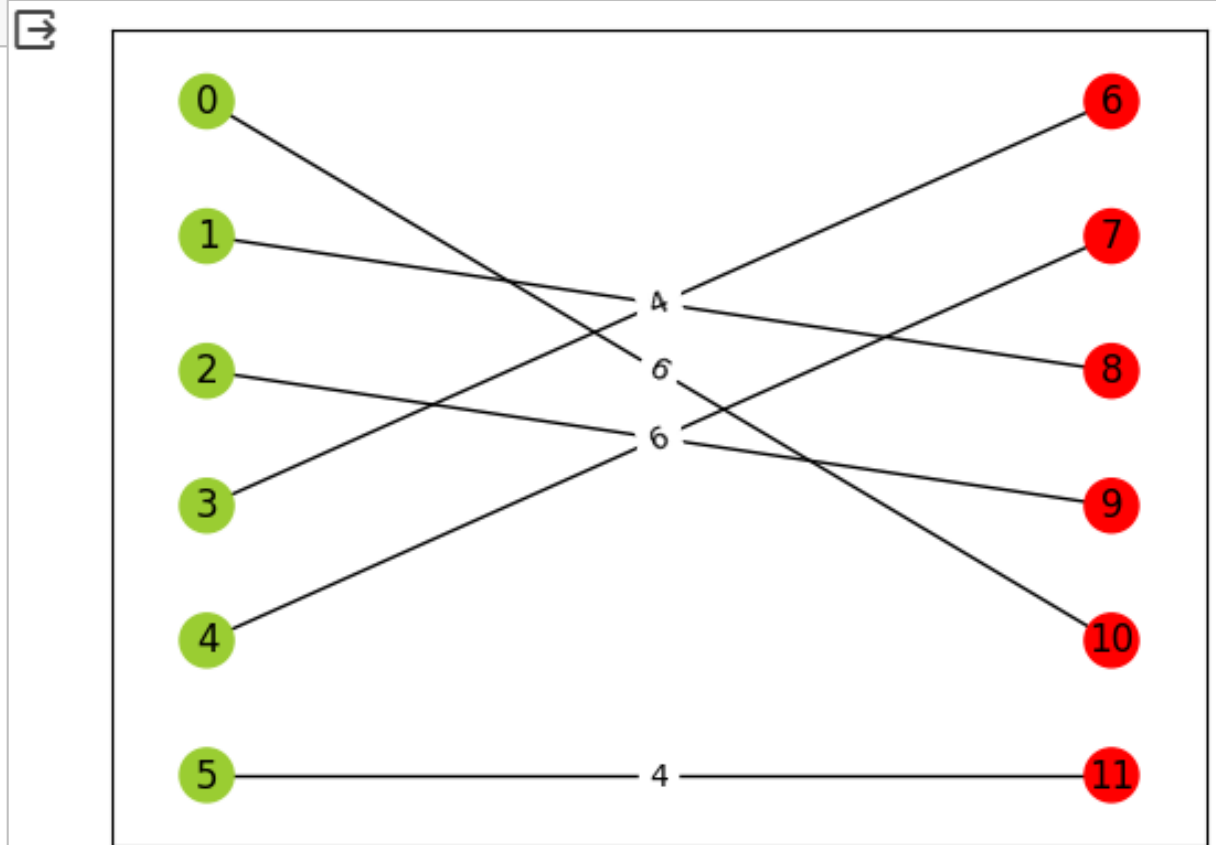
– MWMの記述3 (結果の図示)

```
GR = nx.Graph()
GR.add_nodes_from(G.nodes())
rwedges = [(0,4+6,W[0][4]),(1,2+6,W[1][2]),(2,3+6,W[2][3]),(3,0+6,W[3][0]),(4,1+6,W[4][1]),(5,5+6,W[5][5])]
GR.add_weighted_edges_from(rwedges)
nx.draw_networkx_nodes(GR, pos, node_color=ncol) # 描画: 点, 点の位置, 点の色
nx.draw_networkx_labels(GR, pos)                # 描画: 点のラベル
nx.draw_networkx_edges(GR, pos)                  # 描画: 枝
edge_labels = nx.get_edge_attributes(GR, 'weight')
nx.draw_networkx_edge_labels(GR, pos, edge_labels) # 描画: 枝の重み
plt.show()
```

ペアの相性(重み)

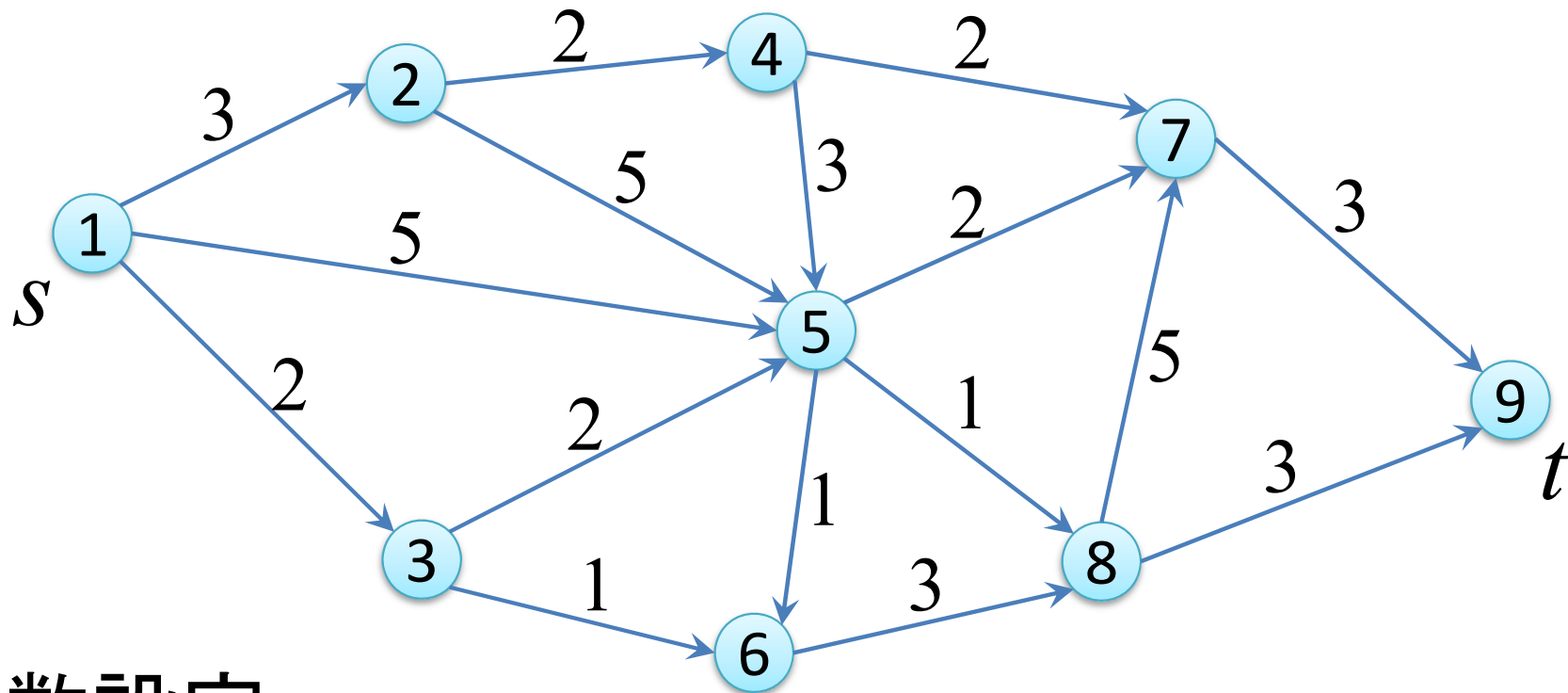
w_{ij}	1	2	3	4	5	6
1	3	1	2	5	6	4
2	1	3	5	4	6	2
3	3	6	1	5	4	2
4	4	6	3	2	5	1
5	1	6	2	5	4	3
6	3	6	5	1	2	4

Objective Value:
6+5+5+4+6+4=30



最短路問題 shortest path problem

- 問) グラフ $G=(V,E)$ と枝上のコスト(cost)が与えられている. スタート地点(点1)からゴール地点(点9)まで, コストの総和が最小となる路(最短路)を求めたい



変数設定

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

最短路問題の定式化

- 0-1整数最適化法によるモデル化(定式化)

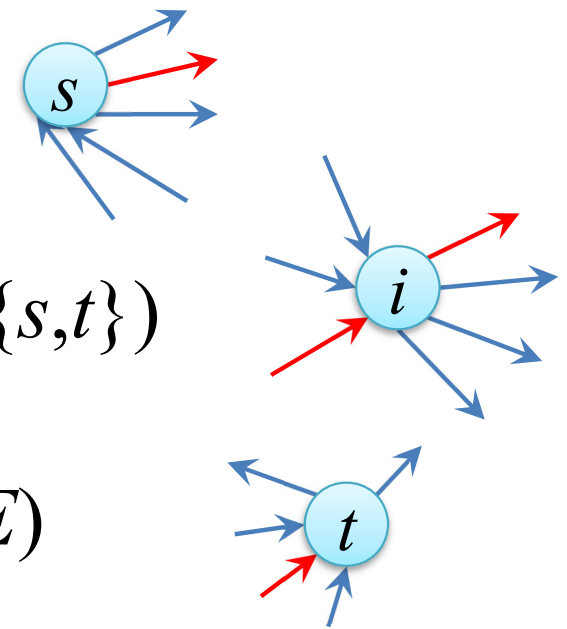
0-1変数 x_{ij} は、枝 $(i,j) \in E$ について、経路として使うなら1、使わないなら0

$$\min. \sum_{(i,j) \in E} c_{ij} x_{ij}$$

$$s.t. \quad \sum_{j \in V} x_{ij} - \sum_{j \in V} x_{ji} = \begin{cases} 1 & (i=s) \\ 0 & (\forall i \in V \setminus \{s,t\}) \\ -1 & (i=t) \end{cases}$$
$$x_{ij} \in \{0,1\} \quad (\forall (i,j) \in E)$$

点iからの流出量の和

点iへの流入量の和



制約式は、「点iからの流出量の和」と「点iへの流入量の和」との差に関するもので点iがスタート地点($i=s$)なら1, ゴール地点($i=t$)なら-1, それ以外なら0とする
(スタート点は流出のみで $1-0=1$, 途中点は通る時 $1-1=0$, 通らない時 $0-0=0$, ゴール点は流入のみで $0-1=-1$ ということ. ただし, この制約だけだとスタート点 $3-2=1$, 途中点 $3-3=0$, ゴール点 $2-3=-1$ 等も実行可能となるが, 目的関数が最小化であることより排除される)

最短路問題の求解

- Excelソルバーで解く(セル記述)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	最短路問題				経路長		点集合							
2	枝集合 E			min.	0		V	流出和	流入和		流出和-流入和			
3		i	j	cost	x_{ij}		i	$\sum_j x_{ij}$	$\sum_j x_{ji}$		$\sum_j x_{ij} - \sum_j x_{ji}$			
4		1	2	3		start	1	0	0		0 = 1			
5		1	3	2			2	0	0		0 = 0			
6		1	5	5			3	0	0		0 = 0			
7		2	4	2			4	0	0		0 = 0			
8		2	5	5			5	0	0		0 = 0			
9		3	5	2			6	0	0		0 = 0			
10		3	6	1			7	0	0		0 = 0			
11		4	5	3			8	0	0		0 = 0			
12		4	7	2		goal	9	0	0		0 = -1			
13		5	6	1										
14		5	7	2			【入力する数式】							
15		5	8	1			<1>	[E2] = SUMPRODUCT(D4:D19, E4:E19)						
16		6	8	3										
17		7	9	3			<2>	[H4] = SUMIF(B\$4:B\$19, \$G4, \$E\$4:\$E\$19)						
18		8	7	5				→[H4]をコピーし, [H4:I12]へ貼り付け						
19		8	9	3										
20							<3>	[K4] = H4 - I4						
21								→[K4]をコピーし, [K5:K12]へ貼り付け						

最短路問題の求解

- Excelで解く
(ソルバー設定)

ソルバーのパラメーター

目的セルの設定:(I)

目標値: ☐ 最大値(M) ☒ 最小値(N) ☐ 指定値:(V)

変数セルの変更:(B)

制約条件の対象:(U)

☒ 制約のない変数を非負数にする(K)

解決方法の選択:(E)

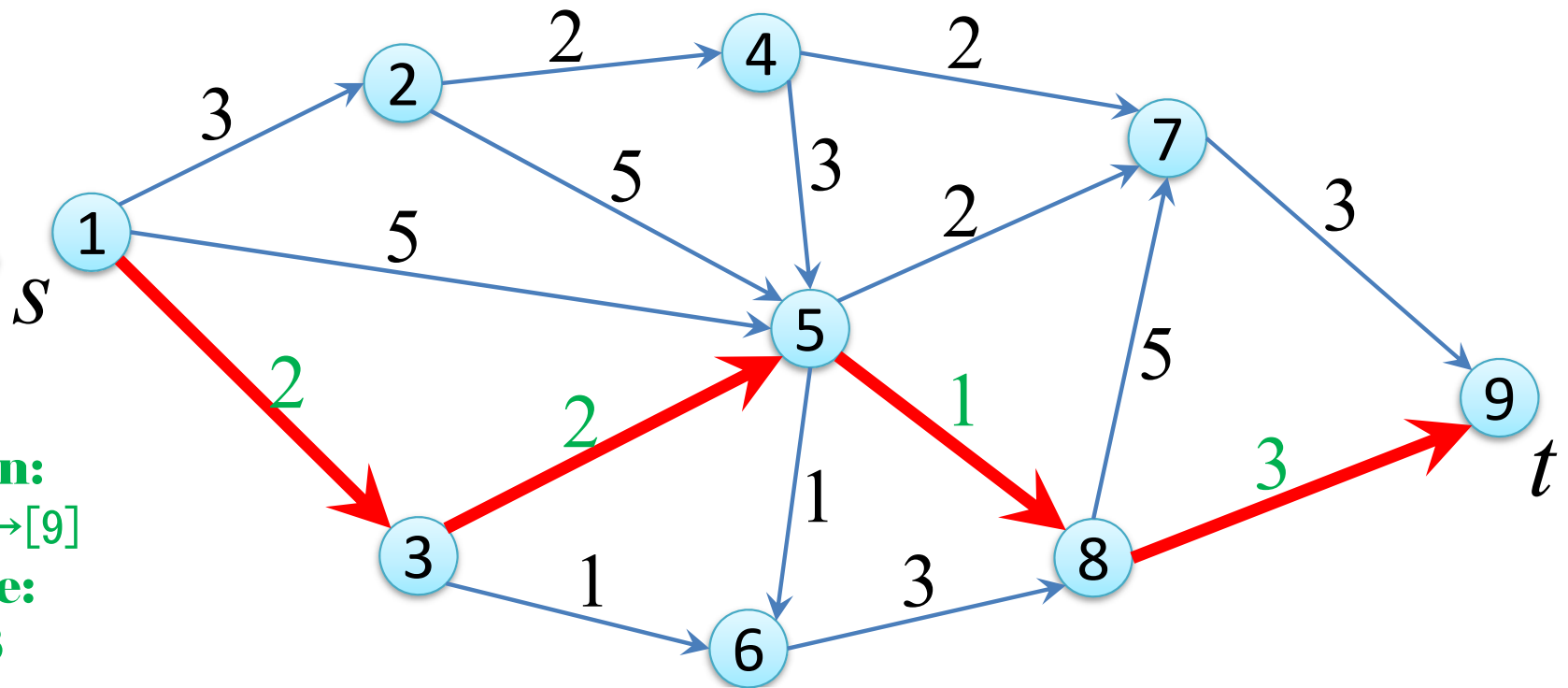
解決方法
滑らかな非線形を示すソルバー問題には GRG 非線形エンジン、線形を示すソルバー問題には LP シンプレックス エンジン、滑らかではない非線形を示すソルバー問題にはエボリューションナリー エンジンを選択してください。

最短路問題

- Excelソルバー
で解いた結果

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	最短路問題				経路長		点集合						
2	枝集合 E			min.	8		V	流出和	流入和		流出和-流入和		
3		i	j	cost	x_{ij}		i	$\sum_j x_{ij}$	$\sum_j x_{ji}$		$\sum_j x_{ij} - \sum_j x_{ji}$		
4		1	2	3	0	start	1	1	0		1 = 1		
5		1	3	2	1		2	0	0		0 = 0		
6		1	5	5	0		3	1	1		0 = 0		
7		2	4	2	0		4	0	0		0 = 0		
8		2	5	5	0		5	1	1		0 = 0		
9		3	5	2	1		6	0	0		0 = 0		
10		3	6	1	0		7	0	0		0 = 0		
11		4	5	3	0		8	1	1		0 = 0		
12		4	7	2	0	goal	9	0	1		-1 = -1		
13		5	6	1	0								
14		5	7	2	0								
15		5	8	1	1		【入力する数式】						
16		6	8	3	0		<1>	[E2] = SUMPRODUCT(D4:D19, E4:E19					
17		7	9	3	0		<2>	[H4] = SUMIF(B\$4:B\$19, \$G4, \$E\$4:\$E					
18		8	7	5	0			→[H4]をコピーし, [H4:I12]へ貼り付け					
19		8	9	3	1								

最適解・最適値
の評価・検証



optimal solution:
[1]→[3]→[5]→[8]→[9]
Objective Value:
2+2+1+3=8

最短路問題の求解

- gurobi & cplex で
解く準備

– lpファイル
[sp_ex1.lp]

minimize

$$\begin{aligned} & 3 x_{12} + 2 x_{13} + 5 x_{15} + 2 x_{24} + 5 x_{25} + 2 x_{35} \\ & + 1 x_{36} + 3 x_{45} + 2 x_{47} + 1 x_{56} + 2 x_{57} + 1 x_{58} \\ & + 3 x_{68} + 3 x_{79} + 5 x_{87} + 3 x_{89} \end{aligned}$$

subject to

$$x_{12} + x_{13} + x_{15} = 1$$

$$x_{24} + x_{25} - x_{12} = 0$$

$$x_{35} + x_{36} - x_{13} = 0$$

$$x_{45} + x_{47} - x_{24} = 0$$

$$x_{56} + x_{57} + x_{58} - x_{15} - x_{25} - x_{35} - x_{45} = 0$$

$$x_{68} - x_{36} - x_{56} = 0$$

$$x_{79} - x_{47} - x_{57} - x_{87} = 0$$

$$x_{87} + x_{89} - x_{58} - x_{68} = 0$$

$$-x_{79} - x_{89} = -1$$

binary

$$x_{12} \ x_{13} \ x_{15}$$

$$x_{24} \ x_{25}$$

$$x_{35} \ x_{36}$$

$$x_{45} \ x_{47}$$

$$x_{56} \ x_{57} \ x_{58}$$

$$x_{68}$$

$$x_{79}$$

$$x_{87} \ x_{89}$$

end

最短路問題の求解

- gurobiで解く

- 解いた結果

optimal solution:

[1]→[3]→[5]→[8]→[9]

Objective Value:

2+2+1+3=8

```
gurobi> m = read('sp_ex1.lp')
Read LP format model from file sp_ex1.lp
Reading time = 0.00 seconds
: 9 rows, 16 columns, 31 nonzeros
gurobi> m.optimize()
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)
Thread count: 10 physical cores, 20 logical processors, using up to 20 threads
Optimize a model with 9 rows, 16 columns and 31 nonzeros
Model fingerprint: 0x283c1908
Variable types: 0 continuous, 16 integer (16 binary)
Coefficient statistics:
  Matrix range    [1e+00, 1e+00]
  Objective range [1e+00, 5e+00]
  Bounds range    [1e+00, 1e+00]
  RHS range       [1e+00, 1e+00]
Found heuristic solution: objective 10.00000000
Presolve removed 9 rows and 16 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)
Thread count was 1 (of 20 available processors)

Solution count 2: 8 10

Optimal solution found (tolerance 1.00e-04)
Best objective 8.000000000000e+00, best bound 8.000000000000e+00, gap 0.0000%
gurobi> m.printAttr('X')

  Variable      X
-----
      x13       1
      x35       1
      x58       1
      x89       1
gurobi> m.ObjVal
8.0
```

最短路問題の求解

- **cplex**で解く

- 解いた結果

optimal solution:

[1]→[3]→[5]→[8]→[9]

Objective Value:

2+2+1+3=8

```
CPLEX> read sp_ex1.lp
Problem 'sp_ex1.lp' read.
Read time = 0.00 sec. (0.00 ticks)
CPLEX> opt
Version identifier: 20.1.0.0 | 2020-11-10 | 9bedb6d68
Tried aggregator 5 times.
MIP Presolve eliminated 1 rows and 5 columns.
MIP Presolve added 1 rows and 1 columns.
Aggregator did 7 substitutions.
Reduced MIP has 2 rows, 4 columns, and 6 nonzeros.
Reduced MIP has 3 binaries, 1 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.02 sec. (0.06 ticks)
Found incumbent of value 8.000000 after 0.02 sec. (0.07 ticks)

Root node processing (before b&c):
  Real time                =    0.02 sec. (0.07 ticks)
Parallel b&c, 20 threads:
  Real time                =    0.00 sec. (0.00 ticks)
  Sync time (average)      =    0.00 sec.
  Wait time (average)      =    0.00 sec.
-----
Total (root+branch&cut) =    0.02 sec. (0.07 ticks)

Solution pool: 1 solution saved.

MIP - Integer optimal solution: Objective = 8.000000000000e+00
Solution time =    0.02 sec. Iterations = 0 Nodes = 0 (1)
Deterministic time = 0.07 ticks (4.43 ticks/sec)

CPLEX> d so v -
Incumbent solution
Variable Name      Solution Value
x13                1.000000
x35                1.000000
x58                1.000000
x89                1.000000
All other variables in the range 1-16 are 0.
```

最短路問題の求解

- Python-MIP で解く
 - Python-mip インストール

```
pip install mip
```

– SPの記述1

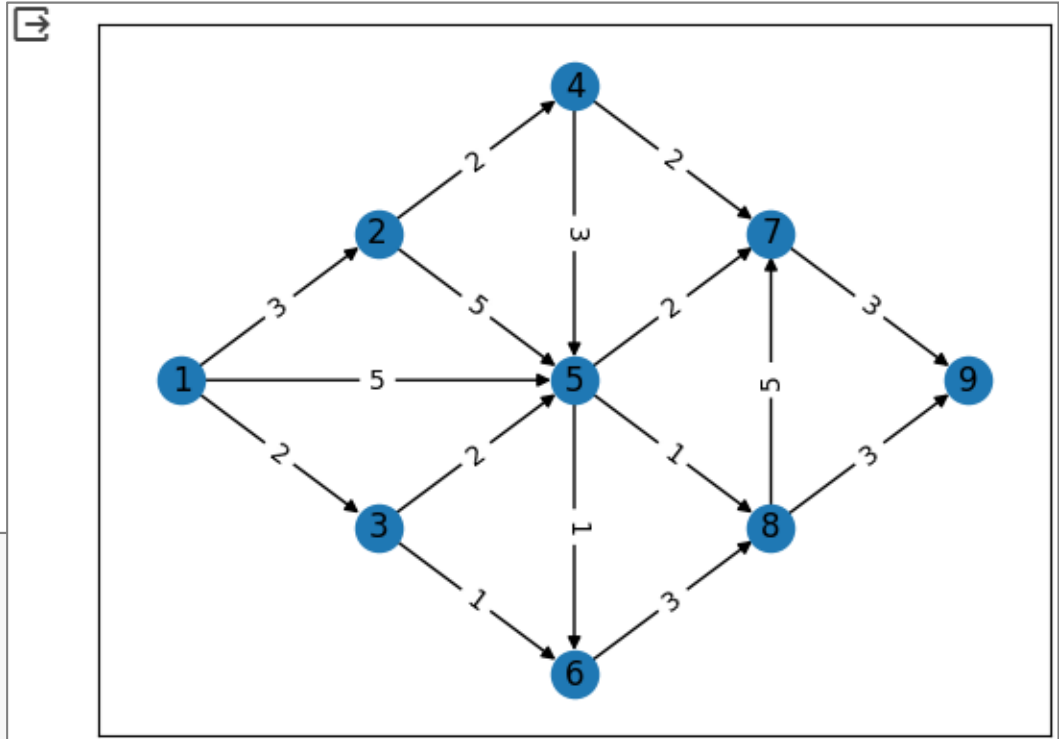
- グラフ定義&係数設定

```
%matplotlib inline
import matplotlib.pyplot as plt
import networkx as nx

G = nx.DiGraph()
G.add_nodes_from([1,2,3,4,5,6,7,8,9])
G.add_weighted_edges_from([(1,2,3),(1,3,2),(1,5,5),(2,4,2),(2,5,5),(3,5,2),(3,6,1),
    (4,5,3),(4,7,2),(5,6,1),(5,7,2),(5,8,1),(6,8,3),(7,9,3),(8,7,5),(8,9,3)])

pos = {1:(0,3), 2:(1,4), 3:(1,2), 4:(2,5), 5:(2,3), 6:(2,1), 7:(3,4), 8:(3,2), 9:(4,3)}
edge_labels = nx.get_edge_attributes(G, 'weight')

nx.draw_networkx_nodes(G, pos)
nx.draw_networkx_labels(G, pos)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_edge_labels(G, pos, edge_labels) # 描画: 枝の重み
plt.show()
```



最短路問題の求解

– SPの記述2(定式化)

※注: Python-MIPを利用して0-1整数線形最適化に定式化して解いているが, NetworkX の `shortest_path(G,...)` 等で解いても良い

定式化

最適解
と
最適値
の表示

<結果>

```
from mip.model import *

I,J = [],[]
for (i,j) in G.edges():
    I.append(i), J.append(j)
b = [1,0,0,0,0,0,0,0,-1]

m = Model("spex1") # モデルの設定 (最短路問題)

x = [[m.add_var(var_type="B", lb=0, ub=1) for j in J] for i in I] # 変数宣言: 最短路枝[0/1]
m.objective = minimize(xsum(G.adj[i][j]['weight'] * x[i][j] for (i,j) in G.edges())) # 目的関数: 最短路
for k in G.nodes():
    m += xsum(x[i][j] for (i,j) in G.edges() if i==k) - xsum(x[i][j] for (i,j) in G.edges() if j==k) <= b[k-1]

m.optimize() # 最適化 (求解) の実行

if m.status.value==0: # もし, 最適解が求まったなら
    spath = [] # 最短路(最適解)のリスト初期化
    print("最適解:") # 最適解を表示
    for (i,j) in G.edges():
        if x[i][j].x==1:
            print(" x[" + str(i) + "," + str(j) + "] = ", x[i][j].x)
            spath.append((i,j)) # 最短路のリスト取得
    print("最適値:", m.objective_value, "=", m.objective) # 目的関数値を表示
else: # もし, 最適解が求まらなかったなら
    print(" error: 最適解は求まりませんでした") # エラーメッセージを表示
```



最適解:

```
x[ 1 , 3 ] = 1.0
x[ 3 , 5 ] = 1.0
x[ 5 , 8 ] = 1.0
x[ 8 , 9 ] = 1.0
```

最適値: 8.0 = + 3.0var(18) + 2.0var(19) + 5.0var(21) + 2.0var(36) + 5.0var(37) + 2.0var(53) + var(54) + 3.0var(69) + 2.0var(71) + va

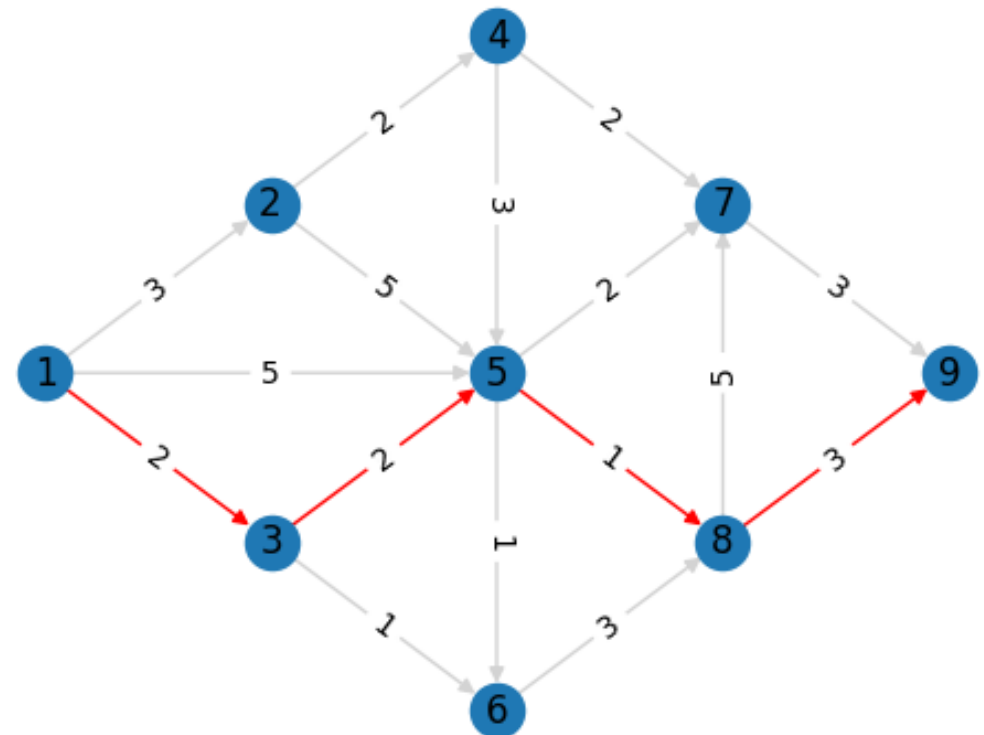
最短路問題の求解

– SPの記述3 (結果の図示)

```
GR = G.copy()

GR.add_edges_from(spath, color='red') # 最短路の枝を赤色に
ecol_dict = nx.get_edge_attributes(GR, "color") # 枝の色属性を取得
ecol = [ecol_dict[p] if p in ecol_dict else 'lightgray' for p in G.edges()] # 最短路以外の枝を薄灰色に

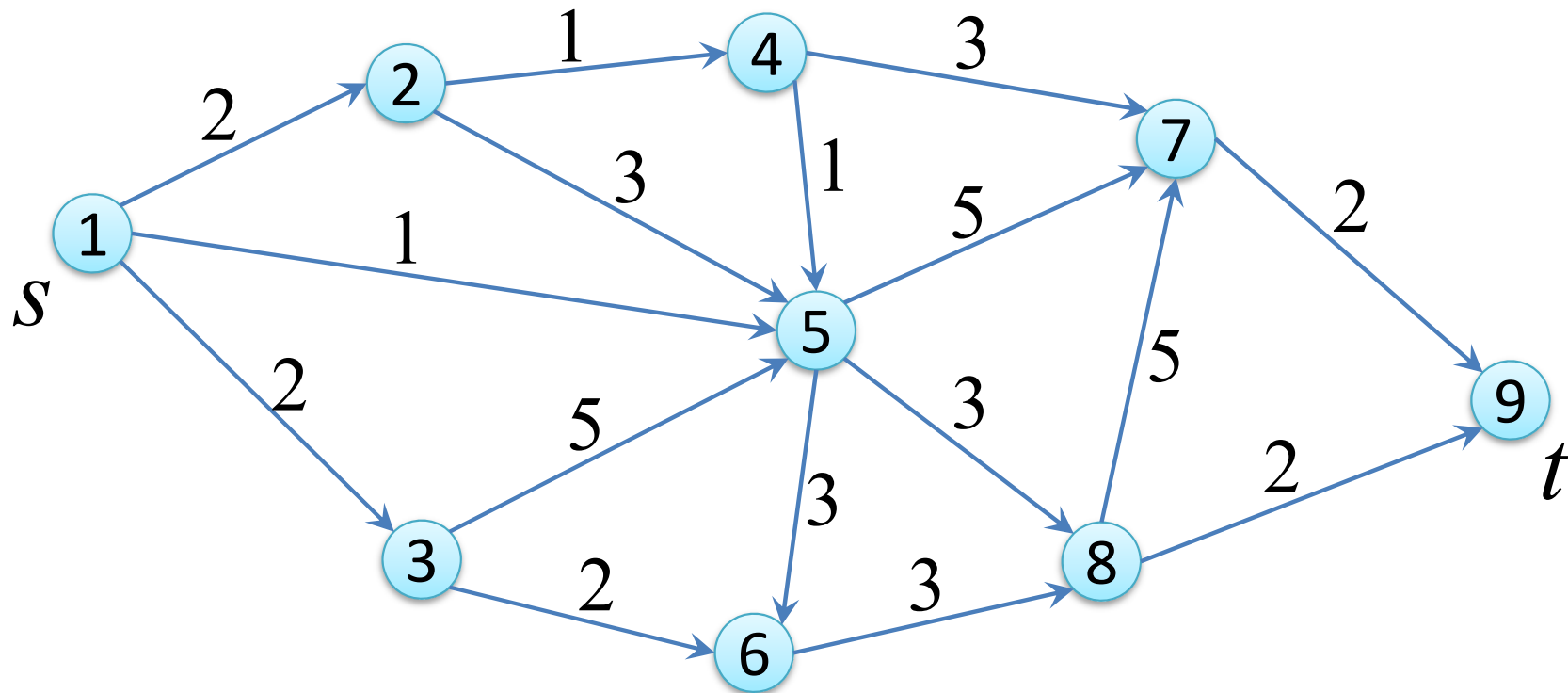
nx.draw_networkx_nodes(GR, pos) # 描画: 点, 点の位置
nx.draw_networkx_labels(GR, pos) # 描画: 点のラベル
nx.draw_networkx_edges(GR, pos, edge_color=ecol) # 描画: 枝, 枝の色
nx.draw_networkx_edge_labels(GR, pos, edge_labels) # 描画: 枝の重み
plt.show()
```



Objective Value:
2+2+1+3=8

最大流問題 maximum flow problem

- 問) グラフ $G=(V,E)$ と枝 $(i,j) \in E$ 上の容量 (capacity) u_{ij} が与えられている. スタート地点 (点1) からゴール地点 (点9) までものを流すとき, 流量が最大となる流れ (最大流) を求めたい



- 変数設定
 - 実数変数 x_{ij} : 枝 (i,j) に流す流量

最大流問題の定式化

- 線形最適化法によるモデル化(定式化)

$$\begin{aligned} \max. \quad & \sum_{j \in V} x_{sj} - \sum_{j \in V} x_{js} \\ \text{s.t.} \quad & \sum_{j \in V} x_{ij} - \sum_{j \in V} x_{ji} = 0 \quad (\forall i \in V \setminus \{s, t\}) \\ & 0 \leq x_{ij} \leq u_{ij} \quad (\forall (i, j) \in E) \end{aligned}$$

実数変数 x_{ij} は、
枝 $(i, j) \in E$ に流れる流量

点 s からの流出量の和

点 i への流入量の和

1つ目の制約式は、流量保存則を表す。即ち、start/goal以外の任意の点 i について「点 i からの流出量の和」と「点 i への流入量の和」との差が0(流量保存)である
(s [start]/ t [goal]は流量保存制約から除外されることに注意)

目的関数は点 s [start]の「流出量の和と流入量の和の差」を最大化することとなる

最大流問題の求解

- Excelソルバーで解く(セル記述)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	最大流問題 Maximum Flow Problem							点集合						
2	枝集合 E					容量		V	流出和	流入和		流出和-流入和		
3		i	j	x_{ij}		u_{ij}		i	$\sum_j x_{ij}$	$\sum_j x_{ji}$		$\sum_j x_{ij} - \sum_j x_{ji}$		
4		1	2		$\leq$	2	start	1	0	0	max.	0		
5		1	3		$\leq$	2		2	0	0		0 =	0	
6		1	5		$\leq$	1		3	0	0		0 =	0	
7		2	4		$\leq$	1		4	0	0		0 =	0	
8		2	5		$\leq$	3		5	0	0		0 =	0	
9		3	5		$\leq$	5		6	0	0		0 =	0	
10		3	6		$\leq$	2		7	0	0		0 =	0	
11		4	5		$\leq$	1		8	0	0		0 =	0	
12		4	7		$\leq$	3	goal	9	0	0				
13		5	6		$\leq$	3								
14		5	7		$\leq$	5		【入力する数式】						
15		5	8		$\leq$	3		<1>	[I4] = SUMIF(B\$4:B\$19, \$H4, \$D\$4:\$D\$19)					
16		6	8		$\leq$	3			→ [I4]をコピーし, [I4:J12]へ貼り付け					
17		7	9		$\leq$	2								
18		8	7		$\leq$	5		<2>	[L4] = I4 - J4					
19		8	9		$\leq$	2			→ [L4]をコピーし, [L5:L12]へ貼り付け					

最大流問題の求解

- Excelで解く
(ソルバー設定)

ソルバーのパラメーター

目的セルの設定:(I)

目標値: ☒ 最大値(M) ☐ 最小値(N) ☐ 指定値:(V)

変数セルの変更:(B)

制約条件の対象:(U)

☐ 制約のない変数を非負数にする(K)

解決方法の選択:(E)

解決方法
滑らかな非線形を示すソルバー問題には GRG 非線形エンジン、線形を示すソルバー問題には LP シンプレックス エンジン、滑らかではない非線形を示すソルバー問題にはエボリューションエンジンを選択してください。

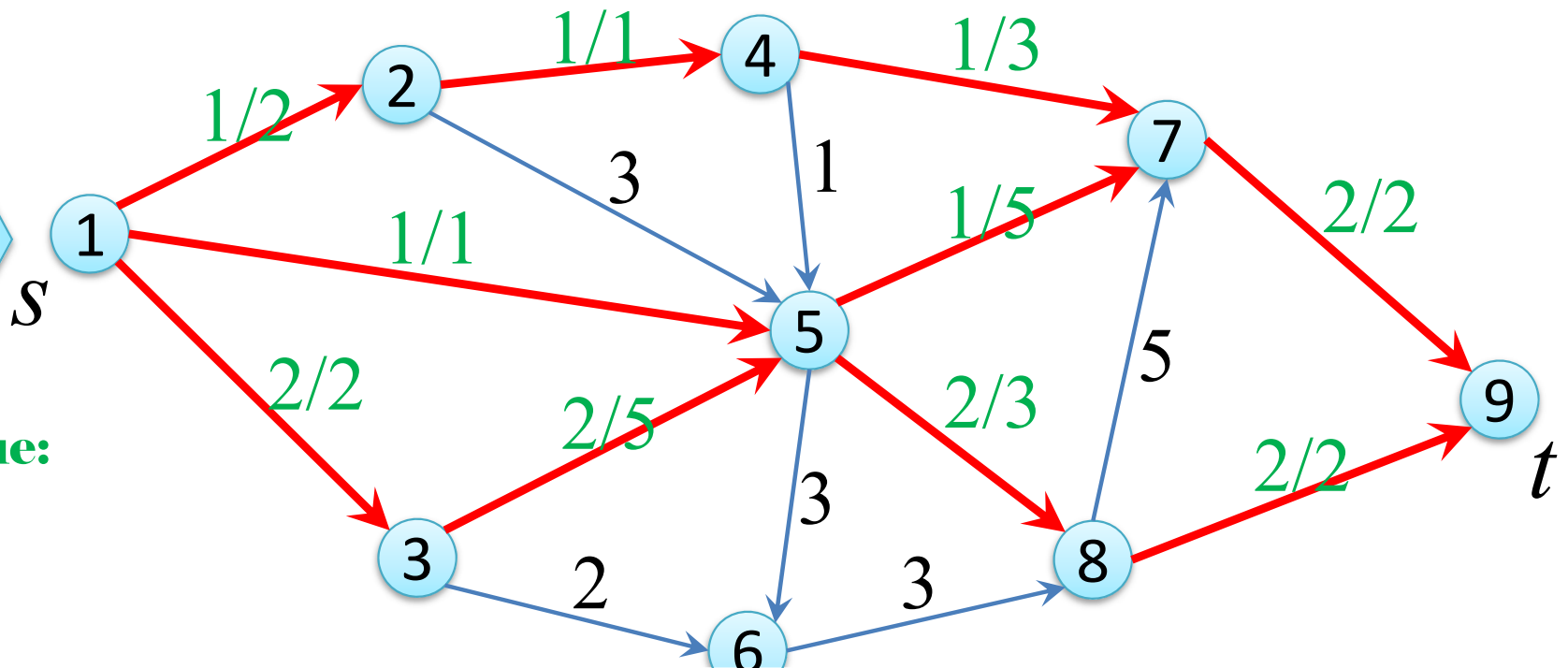
最大流問題

- Excelソルバー
で解いた結果

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	最大流問題 Maximum Flow Problem							点集合					
2	枝集合 E					容量		V	流出和	流入和		流出和-流入和	
3		i	j	x_{ij}		u_{ij}		i	$\sum_j x_{ij}$	$\sum_j x_{ji}$		$\sum_j x_{ij} - \sum_j x_{ji}$	
4		1	2	1	≤	2	start	1	4	0	max.	4	
5		1	3	2	≤	2		2	1	1		0 =	
6		1	5	1	≤	1		3	2	2		0 =	
7		2	4	1	≤	1		4	1	1		0 =	
8		2	5	0	≤	3		5	3	3		0 =	
9		3	5	2	≤	5		6	0	0		0 =	
10		3	6	0	≤	2		7	2	2		0 =	
11		4	5	0	≤	1		8	2	2		0 =	
12		4	7	1	≤	3	goal	9	0	4			
13		5	6	0	≤	3							
14		5	7	1	≤	5		【入力する数式】					
15		5	8	2	≤	3		<1>	[I4] = SUMIF(B\$4:B\$19, \$H4, \$D\$4:\$I				
16		6	8	0	≤	3			→ [I4]をコピーし, [I4:J12]へ貼り付け				
17		7	9	2	≤	2							
18		8	7	0	≤	5		<2>	[L4] = I4 - J4				
19		8	9	2	≤	2			→ [L4]をコピーし, [L5:L12]へ貼り付け				

最適解・最適値
の評価・検証

Objective Value:
1+2+1=4



最大流問題の求解

- gurobi & cplex で
解く準備

– lpファイル
[mf_ex1.lp]

maximize

$$x_{12} + x_{13} + x_{15}$$

subject to

$$x_{24} + x_{25} - x_{12} = 0$$

$$x_{35} + x_{36} - x_{13} = 0$$

$$x_{45} + x_{47} - x_{24} = 0$$

$$x_{56} + x_{57} + x_{58} - x_{15} - x_{25} - x_{35} - x_{45} = 0$$

$$x_{68} - x_{36} - x_{56} = 0$$

$$x_{79} - x_{47} - x_{57} - x_{87} = 0$$

$$x_{87} + x_{89} - x_{58} - x_{68} = 0$$

bound

$$x_{12} \leq 2$$

$$x_{13} \leq 2$$

$$x_{15} \leq 1$$

$$x_{24} \leq 1$$

$$x_{25} \leq 3$$

$$x_{35} \leq 5$$

$$x_{36} \leq 2$$

$$x_{45} \leq 1$$

$$x_{47} \leq 3$$

$$x_{56} \leq 3$$

$$x_{57} \leq 5$$

$$x_{58} \leq 3$$

$$x_{68} \leq 3$$

$$x_{79} \leq 2$$

$$x_{87} \leq 5$$

$$x_{89} \leq 2$$

end

最大流問題の求解

- gurobiで解く

- 解いた結果

```
gurobi> m = read('mf_ex1.lp')
Read LP format model from file mf_ex1.lp
Reading time = 0.00 seconds
: 7 rows, 16 columns, 27 nonzeros
gurobi> m.optimize()
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)
Thread count: 10 physical cores, 20 logical processors, using up to 20 threads
Optimize a model with 7 rows, 16 columns and 27 nonzeros
Model fingerprint: 0x1f836c96
Coefficient statistics:
  Matrix range      [1e+00, 1e+00]
  Objective range   [1e+00, 1e+00]
  Bounds range      [1e+00, 5e+00]
  RHS range         [0e+00, 0e+00]
Presolve removed 0 rows and 1 columns
Presolve time: 0.00s
Presolved: 7 rows, 15 columns, 26 nonzeros

Iteration    Objective          Primal Inf.    Dual Inf.      Time
     0      5.0000000e+00    6.000000e+00    0.000000e+00     0s
     7      4.0000000e+00    0.000000e+00    0.000000e+00     0s

Solved in 7 iterations and 0.00 seconds (0.00 work units)
Optimal objective 4.000000000e+00
gurobi> m.printAttr('X')

  Variable      X
-----
    x12         1
    x13         2
    x15         1
    x24         1
    x35         2
    x45         1
    x57         2
    x58         2
    x79         2
    x89         2

gurobi> m.ObjVal
4.0
```

最大流問題の求解

- **cplex**で解く
- 解いた結果

```
CPLEX> read mf_ex1.lp
Problem 'mf_ex1.lp' read.
Read time = 0.00 sec. (0.00 ticks)
CPLEX> opt
Version identifier: 20.1.0.0 | 2020-11-10 | 9bedb6d68
Tried aggregator 1 time.
LP Presolve eliminated 0 rows and 1 columns.
Reduced LP has 7 rows, 15 columns, and 26 nonzeros.
Presolve time = 0.02 sec. (0.01 ticks)
Initializing dual steep norms . . .

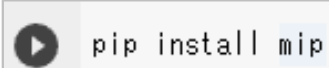
Iteration log . . .
Iteration:    1    Dual objective      =          4.000000

Dual simplex - Optimal: Objective = 4.000000000000e+00
Solution time =    0.02 sec. Iterations = 2 (0)
Deterministic time = 0.02 ticks (1.18 ticks/sec)

CPLEX> d so v -
Variable Name      Solution Value
x12                2.000000
x13                2.000000
x25                2.000000
x35                2.000000
x56                2.000000
x57                2.000000
x68                2.000000
x79                2.000000
x89                2.000000
All other variables in the range 1-16 are 0.
```

最大流問題の求解

- Python-MIP で解く
 - Python-mip インストー

 pip install mip

– MFの記述1

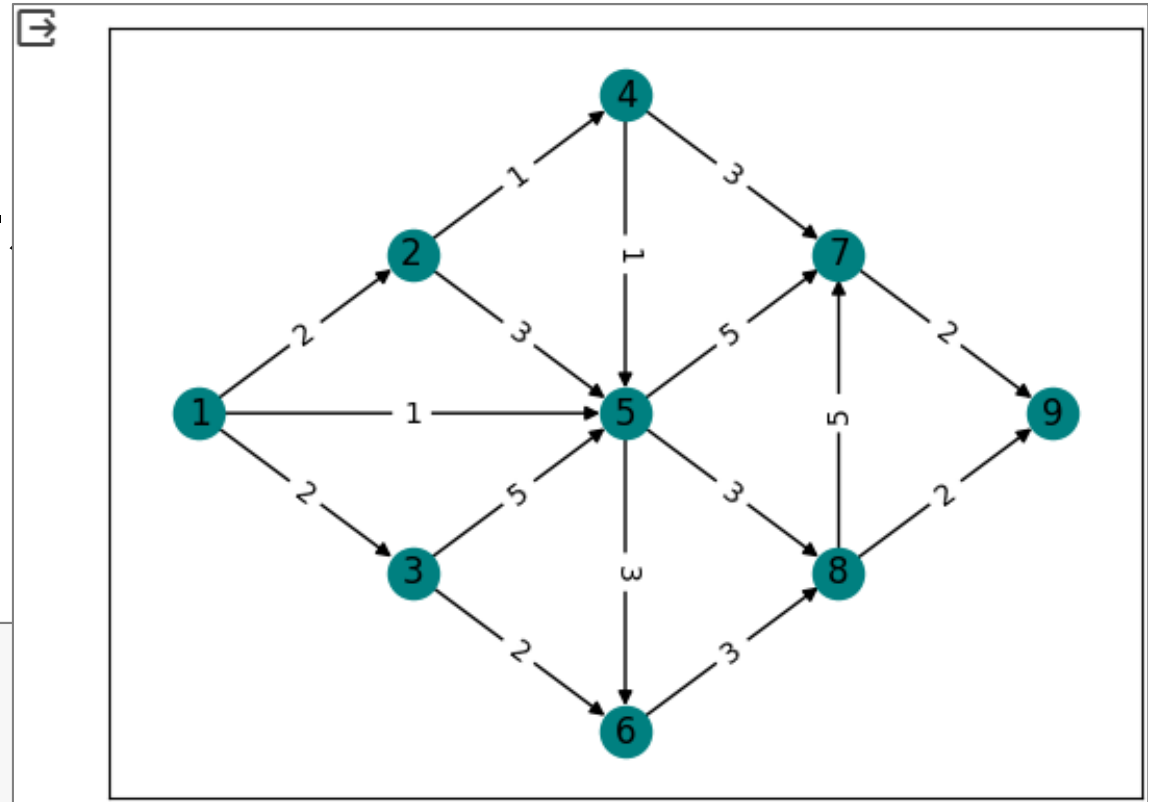
- グラフ定義&係数設定

```
%matplotlib inline
import matplotlib.pyplot as plt
import networkx as nx

G = nx.DiGraph()
G.add_nodes_from([1,2,3,4,5,6,7,8,9])
G.add_weighted_edges_from([(1,2,2),(1,3,2),(1,5,1),(2,4,1),(2,5,3),(3,5,5),(3,6,2),
(4,5,1),(4,7,3),(5,6,3),(5,7,5),(5,8,3),(6,8,3),(7,9,2),(8,7,5),(8,9,2)])

pos = {1:(0,3), 2:(1,4), 3:(1,2), 4:(2,5), 5:(2,3), 6:(2,1), 7:(3,4), 8:(3,2), 9:(4,3)}
edge_labels = nx.get_edge_attributes(G, 'weight')

nx.draw_networkx_nodes(G, pos, node_color='teal') # 描画: 点, 点の位置, 点の色
nx.draw_networkx_labels(G, pos)                 # 描画: 点のラベル
nx.draw_networkx_edges(G, pos)                   # 描画: 枝
nx.draw_networkx_edge_labels(G, pos, edge_labels) # 描画: 枝の重み(容量)
plt.show()
```



最大流問題の求解

※注: Python-MIPを利用して0-1整数線形最適化に定式化して解いているが, NetworkX の `maximum_flow(G,...)` 等で解いても良い

– MFの記述2(定式化)

定式化

```
from mip.model import *

I,J = [],[]
for (i,j) in G.edges():
    I.append(i), J.append(j)

m = Model("mfex1") # モデルの設定 (最大流問題)

x = [[m.add_var(var_type="C", lb=0) for j in J] for i in I] # 変数宣言: 最大流(非負)
m.objective = maximize(xsum(x[i][j] for (i,j) in G.edges() if i==1) - xsum(x[i][j] for (i,j) in G.edges() if j==1))
for k in G.nodes()-{1,9}:
    m += xsum(x[i][j] for (i,j) in G.edges() if i==k) - xsum(x[i][j] for (i,j) in G.edges() if j==k) == 0 # 制約1: 流
for (i,j) in G.edges():
    m += x[i][j] <= G.adj[i][j]["weight"] # 制約2: 各枝 流量<=容量

m.optimize() # 最適化 (求解) の実行
```

最適解
と
最適値
の表示

```
if m.status.value==0: # もし, 最適解が求まったなら
    mflow = [] # 最大流(最適解)のリスト初期化
    print("最適解:") # 最適解を表示
    for (i,j) in G.edges():
        if x[i][j].x>0:
            print(" x[" + str(i) + "," + str(j) + "] = ", x[i][j].x)
            mflow.append((i,j)) # 最大流のリスト取得
    print("最適値:", m.objective_value, "=", m.objective) # 目的関数値
else: # もし, 最適解が求まらなかったなら
    print(" error: 最適解は求まりませんでした") # エラーメッセージを
```

<結果>

```
最適解:
x[ 1 , 2 ] = 1.0
x[ 1 , 3 ] = 2.0
x[ 1 , 5 ] = 1.0
x[ 2 , 4 ] = 1.0
x[ 3 , 6 ] = 2.0
x[ 4 , 5 ] = 1.0
x[ 5 , 7 ] = 2.0
x[ 6 , 8 ] = 2.0
x[ 7 , 9 ] = 2.0
x[ 8 , 9 ] = 2.0
最適値: 4.0 = + var(18) + var(19) + var(21)
```

最大流問題の求解

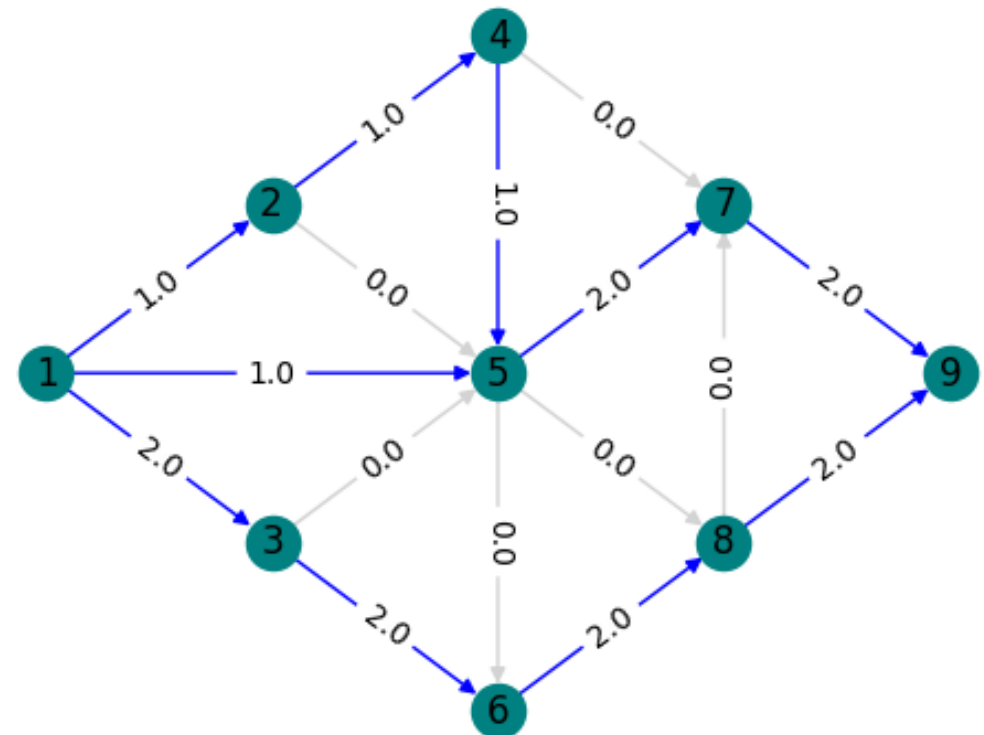
– MFの記述3(結果の図示)

```
GR = G.copy()

GR.add_edges_from(mflow, color='blue') # 最大流の枝を青色に
ecol_dict = nx.get_edge_attributes(GR, "color") # 枝の色属性を取得
ecol = [ecol_dict[p] if p in ecol_dict else 'lightgray' for p in G.edges()] # 最大流以外の枝を薄灰色に

for (i,j) in GR.edges():
    GR.adj[i][j]["weight"] = x[i][j].x
edge_labels = nx.get_edge_attributes(GR, "weight")

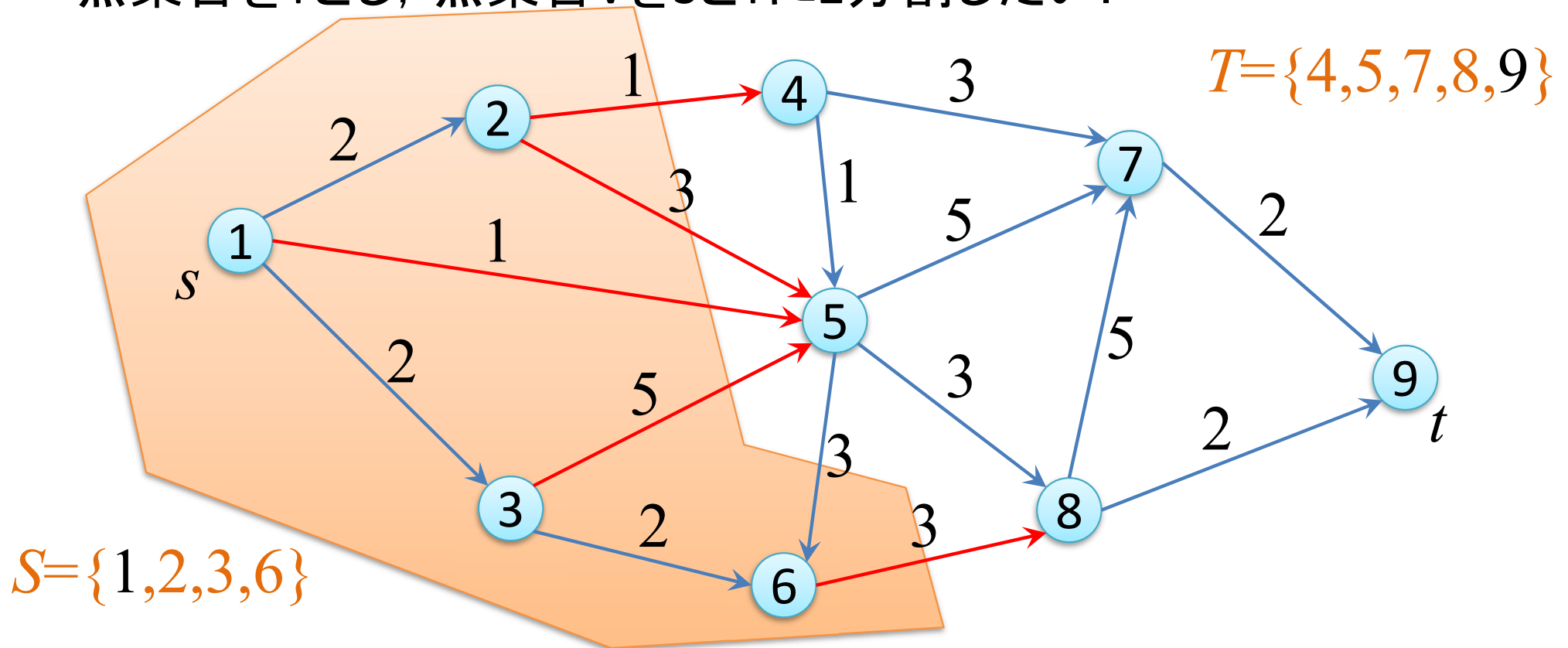
nx.draw_networkx_nodes(GR, pos, node_color='teal') #
nx.draw_networkx_labels(GR, pos) #
nx.draw_networkx_edges(GR, pos, edge_color=ecol) #
nx.draw_networkx_edge_labels(GR, pos, edge_labels) #
plt.show()
```



Objective Value:
1+1+2=4

最小カット問題 minimum cut problem

- 問) グラフ $G=(V,E)$ と枝 $(i,j) \in E$ 上の容量 (capacity) u_{ij} が与えられている. スタート点 (点1) を含む点集合を S , ゴール点 (点9) を含む点集合を T とし, 点集合 V を S と T に2分割したい.

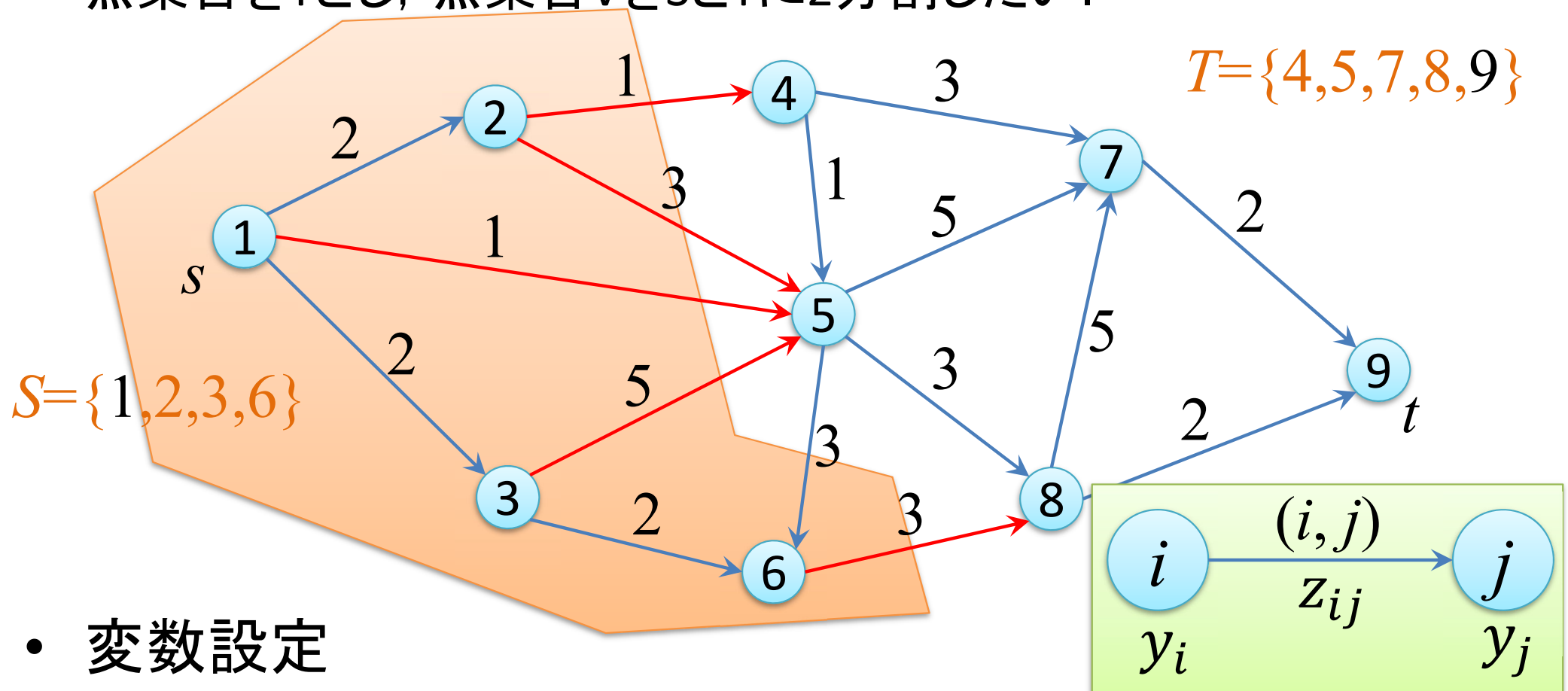


S と T をまたぐ枝 (S の点 $\rightarrow$ T の点への出枝) の枝集合をSTカットとよぶ
容量が最小となるSTカットを求める問題を最小カット問題とよぶ

$ST\text{カット} = \{(2,4), (2,5), (1,5), (3,5), (6,8)\}$, $ST\text{カットの容量} = 13$

最小カット問題 minimum cut problem

- 問) グラフ $G=(V,E)$ と枝 $(i,j) \in E$ 上の容量 (capacity) u_{ij} が与えられている. スタート点 (点1) を含む点集合を S , ゴール点 (点9) を含む点集合を T とし, 点集合 V を S と T に2分割したい.



- 変数設定

- 0-1変数 y_i : 点 i が集合 S に含まれるとき1, T に含まれるとき0
- 0-1変数 z_{ij} : 枝 (i,j) が ST カットに含まれる枝なら1, 違うなら0

最小カット問題の定式化

- 0-1整数最適化法によるモデル化(定式化)

$$\min. \sum_{(i,j) \in E} u_{ij} z_{ij}$$

$$s.t. \quad y_i - y_j \leq z_{ij} \quad (\forall (i,j) \in E) \quad \dots \textcircled{1}$$

$$y_s = 1, y_t = 0 \quad \dots \textcircled{2}$$

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

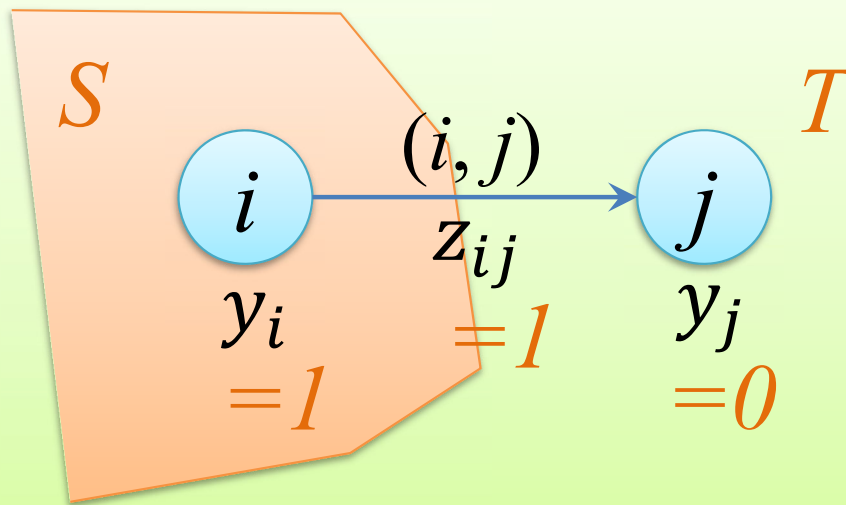
$$y_i \in \{0,1\} \quad (\forall i \in V)$$

$$y_i = 1, y_j = 0$$

のときだけ $z_{ij} = 1$ にせよ
それ以外は自由(ただし、
目的関数より $z_{ij} = 0$ に)

制約①の意味

枝 (i,j) が
STカット
の場合



y_i	y_j		z_{ij}
1	1	→	0 or 1
1	0	→	1
0	1	→	0 or 1
0	0	→	0 or 1

最小カット問題の定式化

- Excelへの記述

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q				
1	最小カット問題 minimum cut problem							STカット容量													
2	点集合 V			枝集合 E			容量	min.					カット制約		スタート/ゴール制約						
3		i	y_i		i	j	u_{ij}	c_{ij}					$y_i - y_j$								
4		1			1	2	2					1				=	1				
5		2			1	3	1					2				=	0				
6		3			1	5	2					3									
7		4			2	4	1					4									
8		5			2	5	3					5									
9		6			3	5	5					6									
10		7			3	6	2					7									
11		8			4	5	1					8									
12		9			4	7	3					9									
13					5	6	3					10									
14	スタート点	1			5	7	5					11									
15	ゴール点	9			5	8	3					12									
16					6	8	3					13									
17					7	9	2					14									
18					8	7	5					15									
19					8	9	2					16									
20																					
21	【入力する数式】																				
22	制約① [M4] = VLOOKUP(E4, B\$4:C\$12, 2, FALSE) – VLOOKUP(F4, B\$4:C\$12, 2, FALSE)																				
23	→[M4]をコピーし、[M5:M19]へ貼り付け																				
24																					
25	制約② [O4] = VLOOKUP(B14, \$B\$4:\$C\$12, 2, FALSE)																				
26	→[O4]をコピーし、[O5]へ貼り付け																				
27																					
28	目的関数 [J2] = SUMPRODUCT(G4:G19, H4:H19)																				

最小カット問題の定式化

- ソルバー設定

ソルバーのパラメーター

目的セルの設定:(I)

目標値: ☐ 最大値(M) ☒ 最小値(N) ☐ 指定値:(V)

変数セルの変更:(B)

制約条件の対象:(U)

☐ 制約のない変数を非負数にする(K)

解決方法の選択:
(E)

解決方法
滑らかな非線形を示すソルバー問題には GRG 非線形エンジン、線形を示すソルバー問題には LP シンプレックス エンジン、滑らかではない非線形を示すソルバー問題にはエボリューションナリー エンジンを選択してください。

最小カット問題

- Excelソルバーで解いた結果

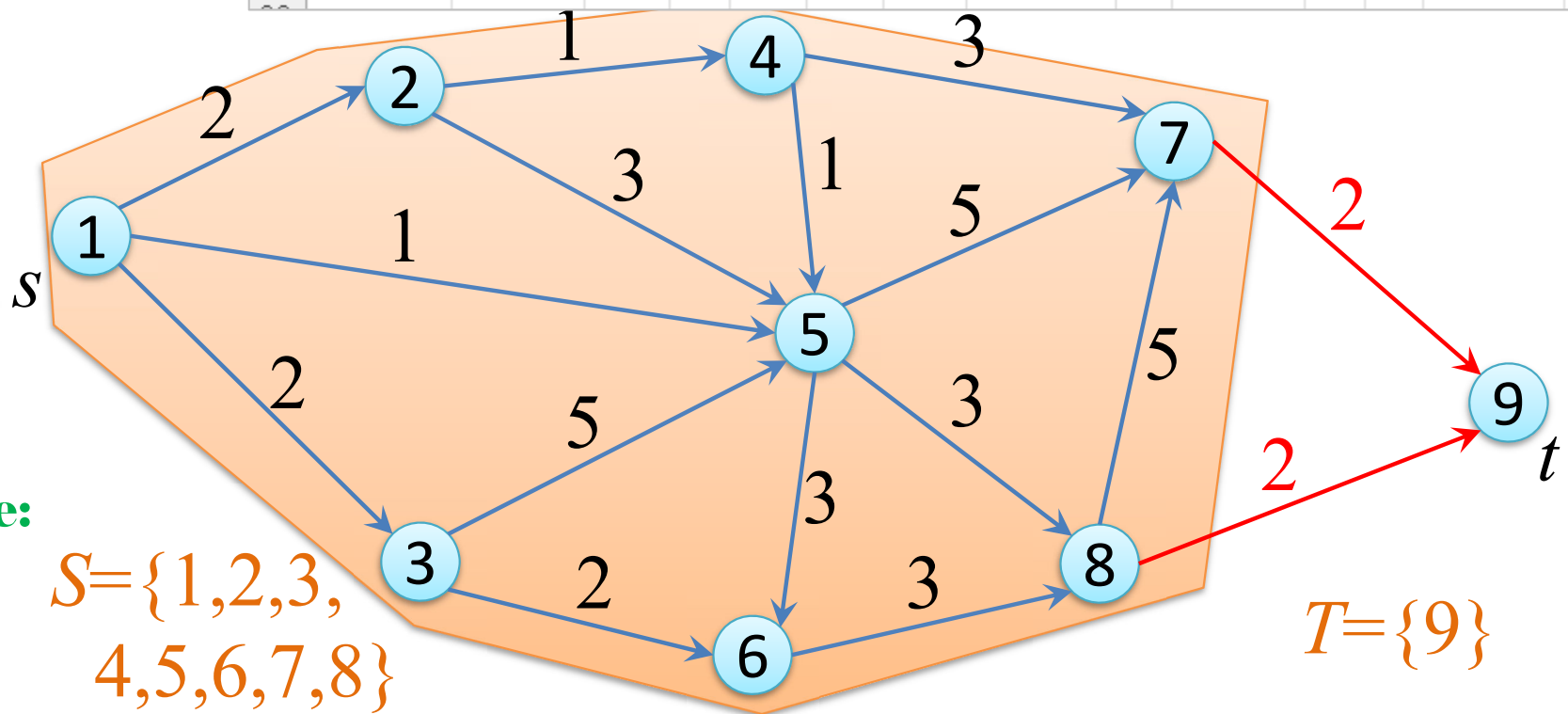
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	最小カット問題 minimum cut problem							STカット 容量					
2		点集合 V			枝集合 E		容量		min.	4		カット 制約	
3		i	y_i		i	j	u_{ij}	z_{ij}					$y_i - y_j$
4		1	1		1	2	2	0				1	0
5		2	1		1	3	2	0				2	0
6		3	1		1	5	1	0				3	0
7		4	1		2	4	1	0				4	0
8		5	1		2	5	3	0				5	0
9		6	1		3	5	5	0				6	0
10		7	1		3	6	2	0				7	0
11		8	1		4	5	1	0				8	0
12		9	0		4	7	3	0				9	0
13					5	6	3	0				10	0
14	スタート点	1			5	7	5	0				11	0
15	ゴール点	9			5	8	3	0				12	0
16					6	8	3	0				13	0
17					7	9	2	1				14	1
18					8	7	5	0				15	0
19					8	9	2	1				16	1

最適解・最適値
の評価・検証

Objective Value:
 $2+2=4$

$S = \{1, 2, 3, 4, 5, 6, 7, 8\}$

$T = \{9\}$



最小カット問題

- gurobi & cplex で
解く準備

– lpファイル
[mc_ex1.lp]

minimize

$$2 x_{12} + 2 x_{13} + x_{15} + x_{24} + 3 x_{25} + 5 x_{35} + 2 x_{36} + x_{45} \\ + 3 x_{47} + 3 x_{56} + 5 x_{57} + 3 x_{58} + 3 x_{68} + 2 x_{79} + 5 x_{87} + 2 x_{89}$$

subject to

$$y_1 - y_2 - x_{12} \leq 0$$

$$y_1 - y_3 - x_{13} \leq 0$$

$$y_1 - y_5 - x_{15} \leq 0$$

$$y_2 - y_4 - x_{24} \leq 0$$

$$y_2 - y_5 - x_{25} \leq 0$$

$$y_3 - y_5 - x_{35} \leq 0$$

$$y_3 - y_6 - x_{36} \leq 0$$

$$y_4 - y_5 - x_{45} \leq 0$$

$$y_4 - y_7 - x_{47} \leq 0$$

$$y_5 - y_6 - x_{56} \leq 0$$

$$y_5 - y_7 - x_{57} \leq 0$$

$$y_5 - y_8 - x_{58} \leq 0$$

$$y_6 - y_8 - x_{68} \leq 0$$

$$y_7 - y_9 - x_{79} \leq 0$$

$$y_8 - y_7 - x_{87} \leq 0$$

$$y_8 - y_9 - x_{89} \leq 0$$

$$y_1 = 1$$

$$y_9 = 0$$

binary

$$x_{12} \ x_{13} \ x_{15} \ x_{24} \ x_{25} \ x_{35} \ x_{36} \ x_{45} \ x_{47}$$

$$x_{56} \ x_{57} \ x_{58} \ x_{68} \ x_{79} \ x_{87} \ x_{89}$$

$$y_1 \ y_2 \ y_3 \ y_4 \ y_5 \ y_6 \ y_7 \ y_8 \ y_9$$

end

最小カット問題の求解

- gurobiで解く

- 解いた結果

```
gurobi> m.printAttr('X')
-----X-----
Variable
-----
x79      1
x89      1
y1       1
y2       1
y3       1
y5       1
y4       1
y6       1
y7       1
v8       1
gurobi> m.ObjVal
4.0
```

```
gurobi> m = read('mc_ex1.lp')
Read LP format model from file mc_ex1.lp
Reading time = 0.00 seconds
: 18 rows, 26 columns, 50 nonzeros
gurobi> m.optimize()
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)
Thread count: 10 physical cores, 20 logical processors, using up to 20 threads
Optimize a model with 18 rows, 26 columns and 50 nonzeros
Model fingerprint: 0xbe696f14
Variable types: 1 continuous, 25 integer (25 binary)
Coefficient statistics:
  Matrix range      [1e+00, 1e+00]
  Objective range   [1e+00, 5e+00]
  Bounds range      [1e+00, 1e+00]
  RHS range         [1e+00, 1e+00]
Found heuristic solution: objective 5.0000000
Presolve removed 11 rows and 18 columns
Presolve time: 0.00s
Presolved: 7 rows, 8 columns, 17 nonzeros
Variable types: 0 continuous, 8 integer (8 binary)

Root relaxation: objective 4.000000e+00, 2 iterations, 0.00 seconds (0.00 work units)

   Nodes      |   Current Node   |   Objective Bounds   |   Work
  Expl Unexpl |  Obj  Depth IntInf | Incumbent   BestBd   Gap   | It/Node Time
*    0       0 |       0         0   | 4.00000000   4.00000   0.00% | -     0s

Explored 1 nodes (2 simplex iterations) in 0.01 seconds (0.00 work units)
Thread count was 20 (of 20 available processors)

Solution count 2: 4 5

Optimal solution found (tolerance 1.00e-04)
Best objective 4.000000000000e+00, best bound 4.000000000000e+00, gap 0.0000%
```

最小カット問題

- cplexで解く
- 解いた結果

```
CPLEX> d so v -
Incumbent solution
Variable Name      Solution Value
x79                1.000000
x89                1.000000
y1                 1.000000
y2                 1.000000
y3                 1.000000
y5                 1.000000
y4                 1.000000
y6                 1.000000
y7                 1.000000
y8                 1.000000
All other variables in the range 1-26 are 0.
CPLEX> _
```

```
CPLEX> read mc_ext1.lp
Problem 'mc_ext1.lp' read.
Read time = 0.00 sec. (0.00 ticks)
CPLEX> opt
Version identifier: 12.10.0.0 | 2019-11-26 | 843d4de2ae
Tried aggregator 2 times.
MIP Presolve eliminated 5 rows and 11 columns.
MIP Presolve added 1 rows and 1 columns.
Aggregator did 5 substitutions.
Reduced MIP has 9 rows, 11 columns, and 23 nonzeros.
Reduced MIP has 10 binaries, 1 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.04 ticks)
Found incumbent of value 5.000000 after 0.00 sec. (0.05 ticks)
Probing time = 0.00 sec. (0.00 ticks)
Tried aggregator 1 time.
Detecting symmetries...
MIP Presolve eliminated 1 rows and 1 columns.
MIP Presolve added 1 rows and 1 columns.
Reduced MIP has 9 rows, 11 columns, and 23 nonzeros.
Reduced MIP has 10 binaries, 1 generals, 0 SOSs, and 0 indicators.
Presolve time = 0.00 sec. (0.02 ticks)
Probing time = 0.00 sec. (0.00 ticks)
Clique table members: 5.
MIP emphasis: balance optimality and feasibility.
MIP search method: dynamic search.
Parallel mode: deterministic, using up to 4 threads.
Root relaxation solution time = 0.00 sec. (0.02 ticks)
```

	Nodes				Cuts/		
	Node	Left	Objective	IInf	Best Integer	Best Bound	ItCnt
*	0+	0			5.0000	0.0000	100.00%
*	0+	0			4.0000	0.0000	100.00%
	0	0	cutoff		4.0000	4.0000	5
	0	0	cutoff		4.0000	4.0000	5

Elapsed time = 0.05 sec. (0.12 ticks, tree = 0.01 MB, solutions = 2)

Root node processing (before b&c):

Real time = 0.05 sec. (0.12 ticks)

Parallel b&c, 4 threads:

Real time = 0.00 sec. (0.00 ticks)

Sync time (average) = 0.00 sec.

Wait time (average) = 0.00 sec.

Total (root+branch&cut) = 0.05 sec. (0.12 ticks)

Solution pool: 2 solutions saved.

MIP - Integer optimal solution: Objective = 4.0000000000e+00

Solution time = 0.05 sec. Iterations = 5 Nodes = 0

Deterministic time = 0.12 ticks (2.55 ticks/sec)

最小カット問題の求解

- Python-MIP で解く
 - Python-mip インストー

```
pip install mip
```

– MCの記述1

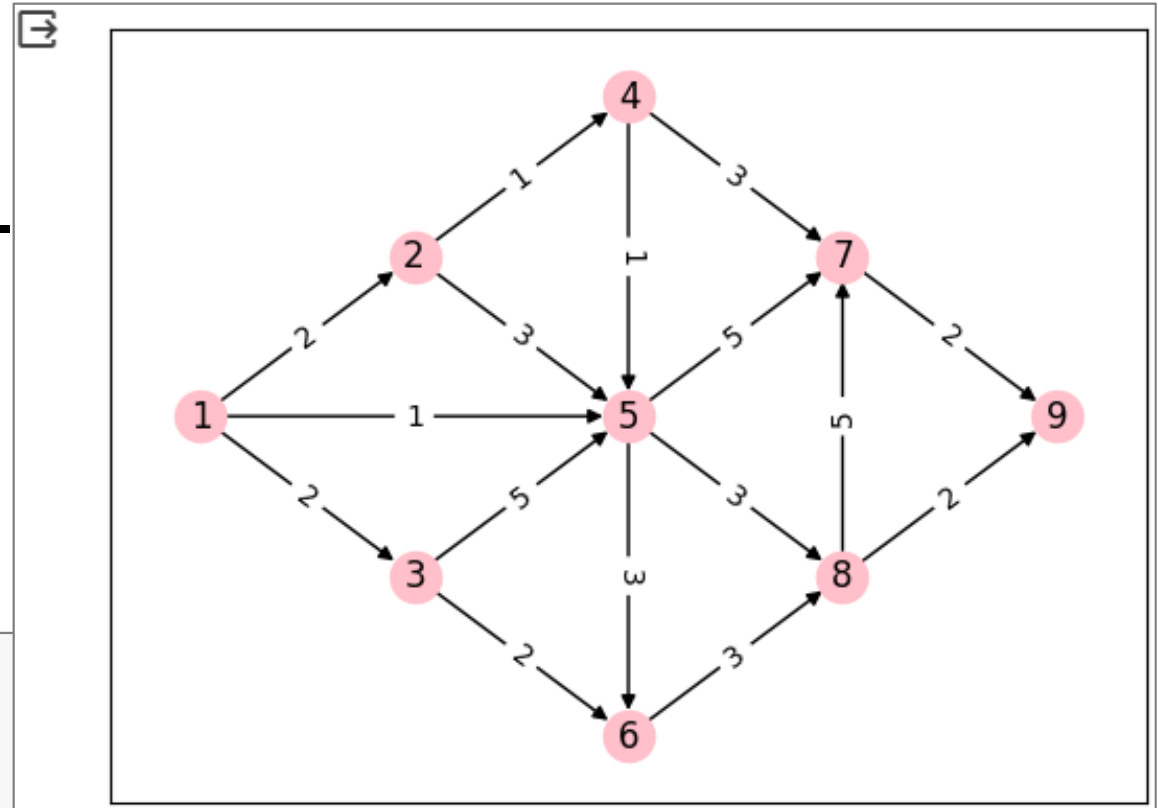
- グラフ定義&係数設定

```
%matplotlib inline
import matplotlib.pyplot as plt
import networkx as nx

G = nx.DiGraph()
G.add_nodes_from([1,2,3,4,5,6,7,8,9])
G.add_weighted_edges_from([(1,2,2),(1,3,2),(1,5,1),(2,4,1),(2,5,3),(3,5,5),(3,6,2),
(4,5,1),(4,7,3),(5,6,3),(5,7,5),(5,8,3),(6,8,3),(7,9,2),(8,7,5),(8,9,2)])

pos = {1:(0,3), 2:(1,4), 3:(1,2), 4:(2,5), 5:(2,3), 6:(2,1), 7:(3,4), 8:(3,2), 9:(4,3)} # 各点の位置座標設定
edge_labels = nx.get_edge_attributes(G, 'weight')

nx.draw_networkx_nodes(G, pos, node_color='pink') # 描画: 点, 点の位置, 点の色
nx.draw_networkx_labels(G, pos) # 描画: 点のラベル
nx.draw_networkx_edges(G, pos) # 描画: 枝
nx.draw_networkx_edge_labels(G, pos, edge_labels) # 描画: 枝の重み(容量)
plt.show()
```



最小カット問題の求解

※注: Python-MIPを利用して0-1整数線形最適化に定式化して解いているが, NetworkXの `minimum_cut(G,...)` 等で解いても良い

– MCの記述2(定式化)

定式化

```
from mip.model import *

I,J = [],[]
for (i,j) in G.edges():
    I.append(i), J.append(j)

m = Model("mcex1") # モデルの設定 (最小カット問題)

z = [[m.add_var(var_type="B", lb=0, ub=1) for j in J] for i in I] # 変数宣言: 最小力
y = [m.add_var(var_type="B", lb=0, ub=1) for v in G.nodes()] # 変数宣言: 最小力
m.objective = minimize(xsum(G.adj[i][j]['weight'] * z[i][j] for (i,j) in G.edges()))
for (i,j) in G.edges():
    m += y[i-1] - y[j-1] <= z[i][j] # 制約1: STカット (※y[]の添え字注意)
    m += y[1-1] == 1 # 制約2: source=1
    m += y[9-1] == 0 # 制約3: sink=9

m.optimize() # 最適化 (求解) の実行
```

最適解と最適値の表示

```
if m.status.value==0: # もし, 最適解が求まったなら
    mcut = [] # 最小カット(最適解)のリスト初期化
    print("最適解:") # 最適解を表示
    for (i,j) in G.edges():
        if z[i][j].x == 1:
            print(" z[" , i , " , " , j , " ] = " , z[i][j].x)
            mcut.append((i,j)) # 最小カットのリスト取得
    for v in G.nodes():
        if y[v-1].x == 0:
            print(" y[" , v , " ] = " , y[v-1].x)
    print("最適値:" , m.objective_value , "=" , m.objective) # 目的関数値を表示
else: # もし, 最適解が求まらなかったなら
    print(" error: 最適解は求まりませんでした") # エラーメッセージを表示
```



最適解:

```
z[ 7 , 9 ] = 1.0
z[ 8 , 9 ] = 1.0
y[ 9 ] = 0.0
```

最適値: 4.0 = + 2.0var(18) + 2.0var(19) + var(21) + var(36) + 3.0var(37) + 5.0var(53) + 2.0var(54) + var(69) + 3.0var(71) + 3.0var(8

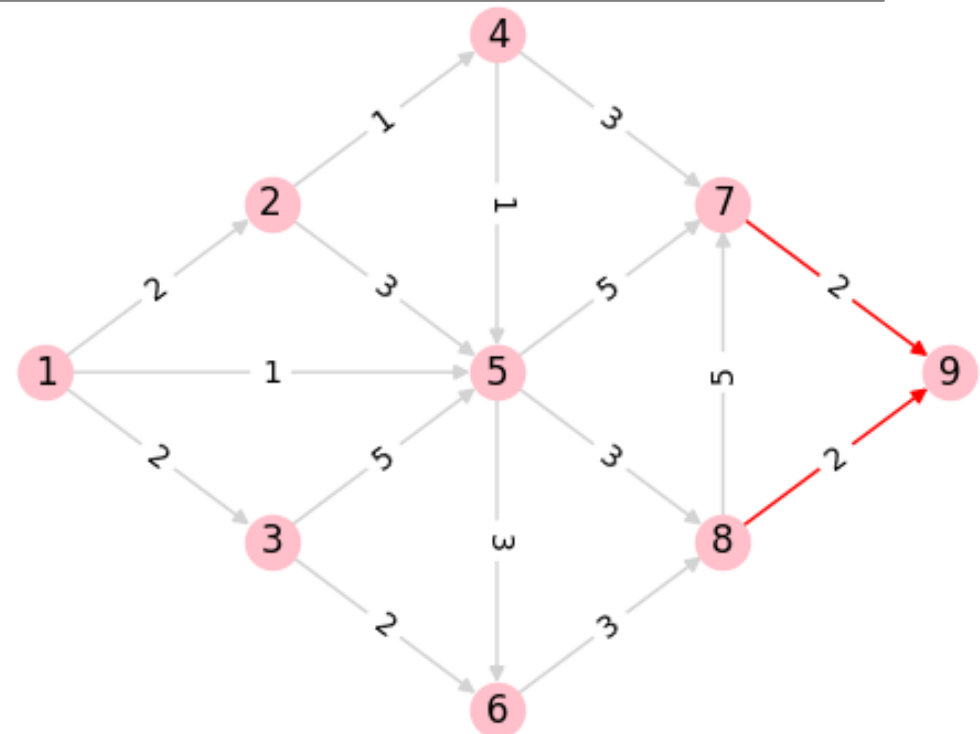
最小カット問題の求解

– MCの記述3 (結果の図示)

```
GR = G.copy()

GR.add_edges_from(mcut, color='red') # 最小カットの枝を赤色に
ecol_dict = nx.get_edge_attributes(GR, "color") # 枝の色属性を取得
ecol = [ecol_dict[p] if p in ecol_dict else 'lightgray' for p in G.edges()] # 最小カット以外の枝を薄灰色に

nx.draw_networkx_nodes(GR, pos, node_color='pink') # 描画: 点, 点の位置, 点の色
nx.draw_networkx_labels(GR, pos) # 描画: 点のラベル
nx.draw_networkx_edges(GR, pos, edge_color=ecol) # 描画: 枝, 枝の色
nx.draw_networkx_edge_labels(GR, pos, edge_labels) # 描画: 枝の重み
plt.show()
```



Objective Value:
2 + 2 = 4

【補足】最大流と最小カットの関係

- 最大フロー・最小カット定理 (max-flow min-cut theorem)

- th) 最大フローが存在するとき,

$$\text{最大流量} = \text{最小カット容量}$$

(資料の例題では, [最大流量 4] = [最小カット容量 4] で一致)

- 最大流問題を主問題(P)としたとき, 最小カット問題が双対問題(D)となる

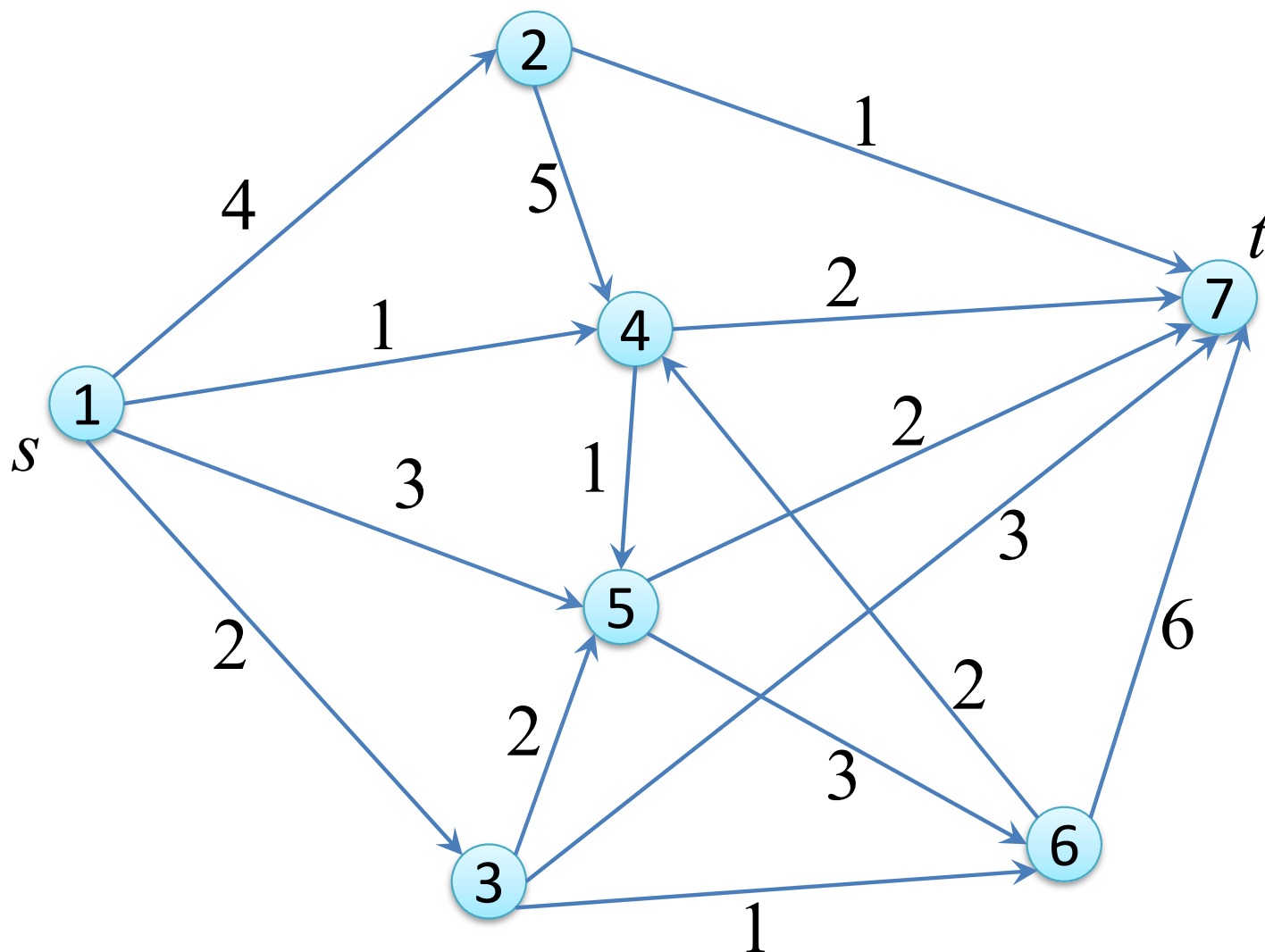
- 最大フロー・最小カット定理は, 双対定理の特殊ケース

- 双対定理 (Duality Theorem)

- th) LPの主問題(P)と双対問題(D)がどちらも実行可能なら, いずれも最適解を持ち最適値が一致する

最小カット問題 minimum cut problem

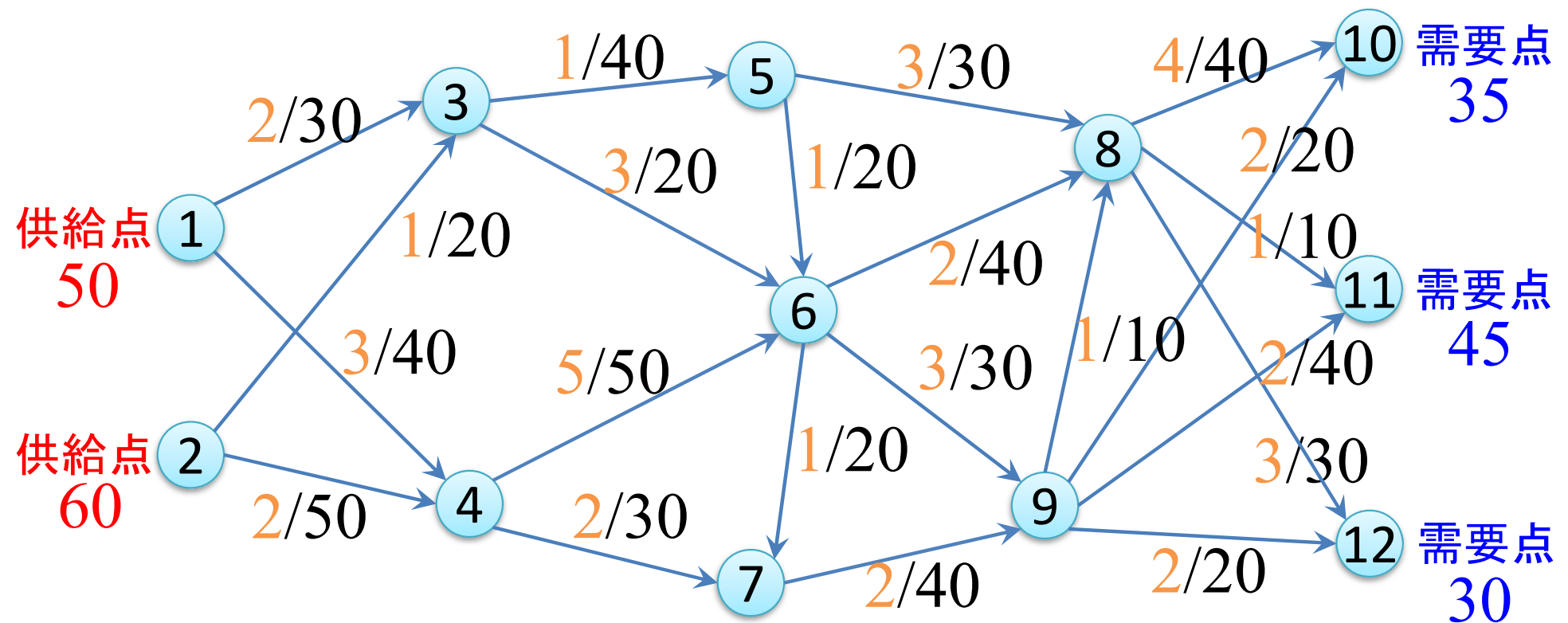
- 演習) グラフ $G=(V,E)$ について, $s=1, t=7$ の最小カットを求めよ



最小費用流問題 minimum cost flow problem

• 例題

グラフ $G=(V,E)$ と枝 $(i,j) \in E$ 上のコスト(cost) c_{ij} と容量(capacity) u_{ij} が与えられている
与えられた需要点の需要と供給点の供給量を満たすフローを考える
実行可能なフローflowのうちで費用最小となるものを求めよ



【演習】

LPに定式化して Excel Solver で求解せよ
(LPファイルで定式化を書くより, Excel の方が定式化が楽)

最小費用流問題の定式化と求解

• 例題：定式化例

実数変数 x_{ij} は、枝 $(i,j) \in E$ に流れる流量

$$\min. \quad \sum_{(i,j) \in E} c_{ij} x_{ij}$$

$$s.t. \quad \sum_{j \in V} x_{ij} - \sum_{j \in V} x_{ji} = b_i \quad (\forall i \in V) \quad \dots \textcircled{1}$$

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

点 i からの流出量の和

点 i への流入量の和

流量保存制約①の右辺定数 b_i の値は以下の通り

- ✓ 供給点 $i \in V$ について $b_i =$ その点の供給量
- ✓ 需要点 $i \in V$ について $b_i = -$ その点の需要量
- ✓ それ以外の点 $i \in V$ について $b_i = 0$ (流量保存)

最小費用流問題の定式化

- Excelへの記述

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	最小費用流問題				最小費用					点集合						
2		枝集合 E		min.						V	流出和	流入和		流出和-流入和		
3		i	j	c_{ij}	x_{ij}		capacity			i	$\sum_j x_{ij}$	$\sum_j x_{ji}$		$\sum_j x_{ij} - \sum_j x_{ji}$		需要供給
4		1	3	2		$\leq$	30		供給	1					=	50
5		1	4	3		$\leq$	40		供給	2					=	60
6		2	3	1		$\leq$	20			3					=	0
7		2	4	2		$\leq$	50			4					=	0
8		3	5	1		$\leq$	40			5					=	0
9		3	6	3		$\leq$	20			6					=	0
10		4	6	5		$\leq$	50			7					=	0
11		4	7	2		$\leq$	30			8					=	0
12		5	6	1		$\leq$	20			9					=	0
13		5	8	3		$\leq$	30		需要	10					=	-35
14		6	7	1		$\leq$	20		需要	11					=	-45
15		6	8	2		$\leq$	40		需要	12					=	-30
16		6	9	3		$\leq$	30									
17		7	9	2		$\leq$	40									
18		8	10	4		$\leq$	40									
19		8	11	1		$\leq$	10									
20		8	12	3		$\leq$	30									
21		9	8	1		$\leq$	10									
22		9	10	2		$\leq$	20									
23		9	11	2		$\leq$	40									
24		9	12	2		$\leq$	20									
25																

【入力する数式】

<1> [K4] = SUMIF(B\$4:B\$24, \$J4, \$E\$4:\$E\$24)
 →[K4]をコピーし, [K4:L15]へ貼り付け

<2> [N4] = K4 - L4
 →[N4]をコピーし, [N5:N15]へ貼り付け

目的関数 [E2] = SUMPRODUCT(D4:D24, E4:E24)

最小費用流問題の定式化

- ソルバー設定

ソルバーのパラメーター

目的セルの設定:(I) ↑

目標値: ☐ 最大値(M) ☒ 最小値(N) ☐ 指定値:(V)

変数セルの変更:(B) ↑

制約条件の対象:(U)

追加(A)
変更(C)
削除(D)
すべてリセット(R)
読み込み/保存(L)

☐ 制約のない変数を非負数にする(K)

解決方法の選択:(E) ↓ オプション(P)

解決方法
滑らかな非線形を示すソルバー問題には GRG 非線形エンジン、線形を示すソルバー問題には LP シンプレックス エンジン、滑らかではない非線形を示すソルバー問題にはエボリュショナリー エンジンを選択してください。

ヘルプ(H) 解決(S) 閉じる(Q)

- Excelソルバーで解いた結果

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	最小費用流問題				最小費用			点集合								
2	枝集合E			min.	1055				V	流出和	流入和	流出和-流入和				
3		i	j	c _{ij}	x _{ij}	capacity			i	Σ _j x _{ij}	Σ _j x _{ji}	Σ _j x _{ij} - Σ _j x _{ji} 需要供給				
4		1	3	2	30	≤	30	供給 供給	1	50	0	50 = 50				
5		1	4	3	20	≤	40		2	60	0	60 = 60				
6		2	3	1	20	≤	20		3	50	50	0 = 0				
7		2	4	2	40	≤	50		4	60	60	0 = 0				
8		3	5	1	40	≤	40		5	40	40	0 = 0				
9		3	6	3	10	≤	20		6	60	60	0 = 0				
10		4	6	5	30	≤	50		7	40	40	0 = 0				
11		4	7	2	30	≤	30		8	40	40	0 = 0				
12		5	6	1	20	≤	20		9	70	70	0 = 0				
13		5	8	3	20	≤	30	需要 需要 需要	10	0	35	-35 = -35				
14		6	7	1	10	≤	20		11	0	45	-45 = -45				
15		6	8	2	20	≤	40		12	0	30	-30 = -30				
16		6	9	3	30	≤	30									
17		7	9	2	40	≤	40									
18		8	10	4	15	≤	40									
19		8	11	1	10	≤	10									
20		8	12	3	15	≤	30									
21		9	8	1	0	≤	10									
22		9	10	2	20	≤	20									
23		9	11	2	35	≤	40									
24		9	12	2	15	≤	20									
【入力する数式】																
<1> [K4] = SUMIF(B\$4:B\$24, \$J4, \$E\$4:\$E\$24) →[K4]をコピーし, [K4:L15]へ貼り付け																
<2> [N4] = K4 - L4 →[N4]をコピーし, [N5:N15]へ貼り付け																
目的関数 [E2] = SUMPRODUCT(D4:D24, E4:E24)																

最小費用流問題の求解

- gurobi & cplex で解く準備
 - lpファイル [[mcf_ex1.lp](#)]

```
minimize
  2 x13 + 3 x14 + x23 + 2 x24 + x35 + 3 x36 + 5 x46 + 2 x47
  + x56 + 3 x58 + x67 + 2 x68 + 3 x69 + 2 x79
  + 4 x810 + x811 + 3 x812 + x98 + 2 x910 + 2 x911 + 2 x912

subject to
  x13 + x14 = 50
  x23 + x24 = 60
  x35 + x36 - x13 - x23 = 0
  x46 + x47 - x14 - x24 = 0
  x56 + x58 - x35 = 0
  x67 + x68 + x69 - x36 - x46 - x56 = 0
  x79 - x47 - x67 = 0
  x810 + x811 + x812 - x58 - x68 - x98 = 0
  x910 + x911 + x912 - x69 - x79 = 0
  -x810 - x910 = -35
  -x811 - x911 = -45
  -x812 - x912 = -30
```

```
bound
  x13 <= 30
  x14 <= 40
  x23 <= 20
  x24 <= 50
  x35 <= 40
  x36 <= 20
  x46 <= 50
  x47 <= 30
  x56 <= 20
  x58 <= 30
  x67 <= 20
  x68 <= 40
  x69 <= 30
  x79 <= 40
  x810 <= 40
  x811 <= 10
  x812 <= 30
  x98 <= 10
  x910 <= 20
  x911 <= 40
  x912 <= 20

end
```

最小費用流問題の求解

- gurobiで解く

- 解いた結果

```
gurobi> m.printAttr('X')
```

Variable	X
x13	30
x14	20
x23	20
x24	40
x35	40
x36	10
x46	30
x47	30
x56	10
x58	30
x67	10
x68	10
x69	30
x79	40
x810	15
x811	10
x812	15
x910	20
x911	35
x912	15

```
gurobi> m.ObjVal  
1055.0
```

```
gurobi> m = read('mcf_ex1.lp')  
Read LP format model from file mcf_ex1.lp  
Reading time = 0.00 seconds  
: 12 rows, 21 columns, 41 nonzeros  
gurobi> m.optimize()  
Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)  
Thread count: 10 physical cores, 20 logical processors, using up to 20 threads  
Optimize a model with 12 rows, 21 columns and 41 nonzeros  
Model fingerprint: 0xc556d62e  
Coefficient statistics:  
Matrix range [1e+00, 1e+00]  
Objective range [1e+00, 5e+00]  
Bounds range [1e+01, 5e+01]  
RHS range [3e+01, 6e+01]  
Presolve removed 5 rows and 6 columns  
Presolve time: 0.00s  
Presolved: 7 rows, 15 columns, 29 nonzeros  
  
Iteration    Objective      Primal Inf.    Dual Inf.      Time  
      0      7.0980100e+02  6.000275e+01  0.000000e+00      0s  
      9      1.0550000e+03  0.000000e+00  0.000000e+00      0s  
  
Solved in 9 iterations and 0.01 seconds (0.00 work units)  
Optimal objective 1.055000000e+03
```

最小費用流問題の求解

- **cplex**で解く
- 解いた結果

```
CPLEX> read mcf_ex1.lp
Problem 'mcf_ex1.lp' read.
Read time = 0.00 sec. (0.00 ticks)
CPLEX> opt
Version identifier: 12.10.0.0 | 2019-11-26 | 843d4de2ae
Tried aggregator 1 time.
LP Presolve eliminated 0 rows and 1 columns.
Aggregator did 5 substitutions.
Reduced LP has 7 rows, 15 columns, and 29 nonzeros.
Presolve time = 0.02 sec. (0.01 ticks)
Initializing dual steep norms . . .

Iteration log . . .
Iteration:      1      Dual objective      =      650.000000

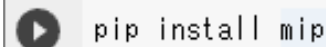
Dual simplex - Optimal: Objective = 1.0550000000e+03
Solution time =    0.02 sec. Iterations = 8 (0)
Deterministic time = 0.03 ticks (1.71 ticks/sec)

CPLEX> d so v -
Variable Name      Solution Value
x13                30.000000
x14                20.000000
x23                20.000000
x24                40.000000
x35                40.000000
x36                10.000000
x46                30.000000
x47                30.000000
x56                20.000000
x58                20.000000
x68                35.000000
x69                25.000000
x79                30.000000
x810               15.000000
x811               10.000000
x812               30.000000
x910               20.000000
x911               35.000000
All other variables in the range 1-21 are 0.
CPLEX>
```

最小費用流問題の求解

- Python-MIP で解く

— Python-mip インストール & MCFの記述1(グラフ定義/係数設定)



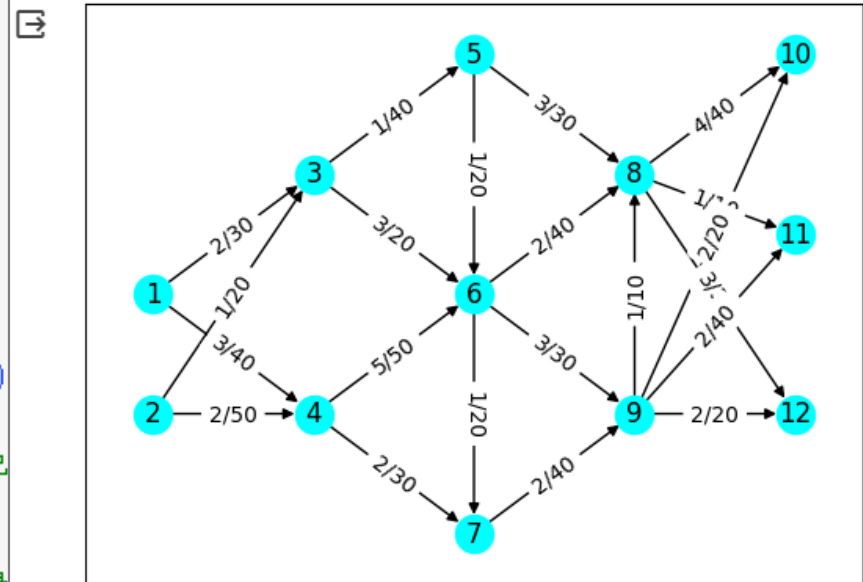
```
%matplotlib inline
import matplotlib.pyplot as plt
import networkx as nx

G = nx.DiGraph()
nodes = [(1, {'ds':50}), (2, {'ds':60}),
         (3, {'ds':0}), (4, {'ds':0}), (5, {'ds':0}), (6, {'ds':0}), (7, {'ds':0}), (8, {'ds':0}), (9, {'ds':0}),
         (10, {'ds':-35}), (11, {'ds':-45}), (12, {'ds':-30})]
edges = [(1,3,{'cost':2,'capacity':30}), (1,4,{'cost':3,'capacity':40}),
        (2,3,{'cost':1,'capacity':20}), (2,4,{'cost':2,'capacity':50}), (3,5,{'cost':1,'capacity':40}),
        (3,6,{'cost':3,'capacity':20}), (4,6,{'cost':5,'capacity':50}), (4,7,{'cost':2,'capacity':30}),
        (5,6,{'cost':1,'capacity':20}), (5,8,{'cost':3,'capacity':30}), (6,7,{'cost':1,'capacity':20}),
        (6,8,{'cost':2,'capacity':40}), (6,9,{'cost':3,'capacity':30}), (7,9,{'cost':2,'capacity':40}),
        (8,10,{'cost':4,'capacity':40}), (8,11,{'cost':1,'capacity':10}), (8,12,{'cost':3,'capacity':30}),
        (9,8,{'cost':1,'capacity':10}), (9,10,{'cost':2,'capacity':20}), (9,11,{'cost':2,'capacity':40}),
        (9,12,{'cost':2,'capacity':20})]
G.add_nodes_from(nodes)
G.add_edges_from(edges)

pos = {1:(0,2), 2:(0,1), 3:(1,3), 4:(1,1), 5:(2,4), 6:(2,2), 7:(2,0),
       8:(3,3), 9:(3,1), 10:(4,4), 11:(4,2.5), 12:(4,1)} # 各点の位置座標設定

edge_labels = nx.get_edge_attributes(G, 'cost')
for (i,j) in G.edges():
    edge_labels[i,j] = str(G.adj[i][j]['cost'])+'/'+str(G.adj[i][j]['capacity'])

nx.draw_networkx_nodes(G, pos, node_color='cyan') # 描画: 点, 点の位置, 点の色
nx.draw_networkx_labels(G, pos)                  # 描画: 点のラベル
nx.draw_networkx_edges(G, pos)                   # 描画: 枝
nx.draw_networkx_edge_labels(G, pos, edge_labels) # 描画: 枝の重み(コスト/容量)
plt.show()
```



最小費用流問題の求解

※注: Python-MIPを利用して0-1整数線形最適化に定式化して解いているが, NetworkXの `min_cost_flow(G,...)` 等で解いても良い

– MCFの記述2(定式化)

定式化

最適解
と
最適値
の表示

```
from mip.model import *

I,J = [],[]
for (i,j) in G.edges():
    I.append(i), J.append(j)
b = nx.get_node_attributes(G,'ds')

m = Model("mcfex1") # モデルの設定 (最小費用流問題)

x = [[m.add_var(var_type="C", lb=0) for j in J] for i in I] # 変数宣言: 最小費用流(非負)
m.objective = minimize(xsum(G.adj[i][j]['cost'] * x[i][j] for (i,j) in G.edges())) # 目的関数: 最小費用流
for k in G.nodes():
    m += xsum(x[i][j] for (i,j) in G.edges() if i==k) - xsum(x[i][j] for (i,j) in G.edges() if j==k) == b[k]
for (i,j) in G.edges():
    m += x[i][j] <= G.adj[i][j]['capacity'] # 制約2: 各枝 流量<=容量

m.optimize() # 最適化 (求解) の実行

if m.status.value==0: # もし, 最適解が求まったなら
    mcf = [] # 最小費用流(最適解)のリスト初期化
    print("最適解:") # 最適解を表示
    for (i,j) in G.edges():
        if x[i][j].x > 0:
            print(" x[" + i + "," + j + "] = ", x[i][j].x)
            mcf.append((i,j)) # 最小費用流のリスト取得
    print("最適値:", m.objective_value, "=", m.objective) # 目的関数値を表示
else: # もし, 最適解が求まらなかったなら
    print(" error: 最適解は求まりませんでした") # エラーメッセージを表示
```

⇒ 最適解:

<結果>

```
x[ 1 , 3 ] = 30.0
x[ 1 , 4 ] = 20.0
x[ 2 , 3 ] = 20.0
x[ 2 , 4 ] = 40.0
x[ 3 , 5 ] = 40.0
x[ 3 , 6 ] = 10.0
x[ 4 , 6 ] = 30.0
x[ 4 , 7 ] = 30.0
x[ 5 , 6 ] = 10.0
x[ 5 , 8 ] = 30.0
x[ 6 , 8 ] = 20.0
x[ 6 , 9 ] = 30.0
x[ 7 , 9 ] = 30.0
x[ 8 , 10 ] = 15.0
x[ 8 , 11 ] = 10.0
x[ 8 , 12 ] = 25.0
x[ 9 , 10 ] = 20.0
x[ 9 , 11 ] = 35.0
x[ 9 , 12 ] = 5.0
```

最適値: 1055.0 = + 2.0var(24) + 3.0var(25) +

最小費用流問題の求解

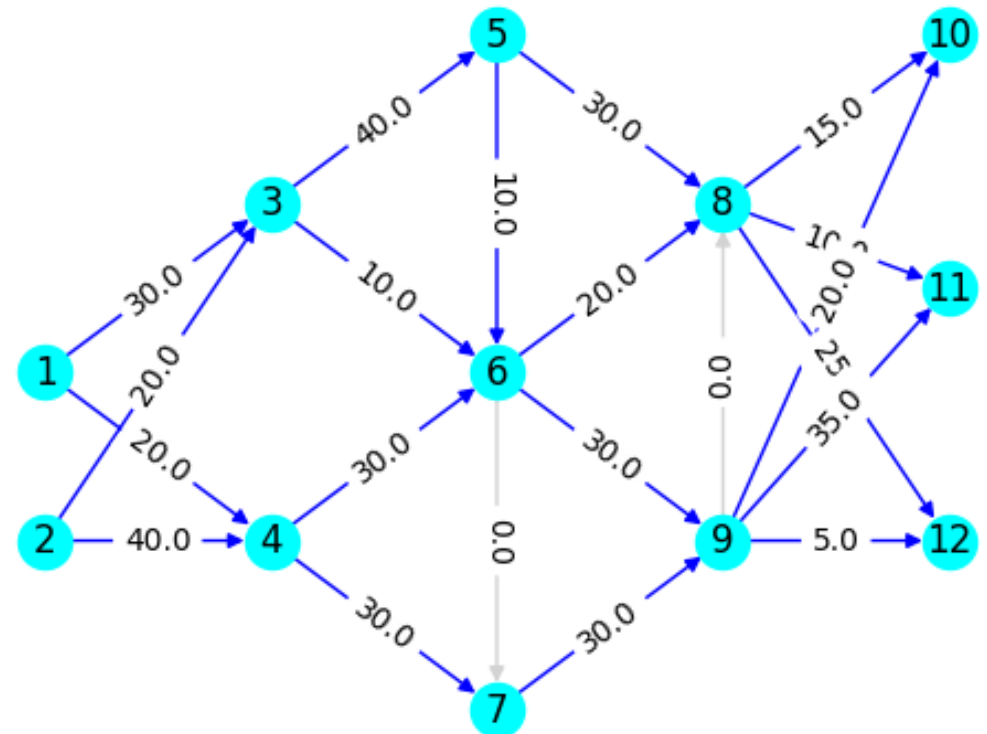
– MCFの記述3 (結果の図示)

```
GR = G.copy()

GR.add_edges_from(mcf, color='blue')          # 最小費用流の枝を青色に
ecol_dict = nx.get_edge_attributes(GR, 'color') # 枝の色属性を取得
ecol = [ecol_dict[p] if p in ecol_dict else 'lightgray' for p in G.edges()] # 最小費用流以外の枝を薄灰色に

for (i,j) in GR.edges():
    GR.adj[i][j]["capacity"] = x[i][j].x
edge_labels = nx.get_edge_attributes(GR, 'capacity')

nx.draw_networkx_nodes(GR, pos, node_color='cyan')
nx.draw_networkx_labels(GR, pos)
nx.draw_networkx_edges(GR, pos, edge_color=ecol)
nx.draw_networkx_edge_labels(GR, pos, edge_labels)
plt.show()
```



Objective Value:

$$2 \cdot 30 + 3 \cdot 20 + \dots = 1055$$

【補足】

- 最小費用流問題は、最短路問題と最大流問題を含む
 - 最小費用流問題の定式化において以下の設定をすれば良い
 - ✓ スタート点 $i=s$ について, $b_s=1$
 - ✓ ゴール点 $i=t$ について, $b_t=-1$
 - ✓ それ以外の点 i について, $b_i=0$
 - ✓ 全ての枝 (i,j) の容量 $u_{ij}=\infty$
 - 最小費用流問題の定式化において以下の設定をすれば良い
 - ✓ スタート点 $i=s$ について, $b_s=f$ ※ $f = \sum c_{sj}x_{sj} - \sum c_{js}x_{js}$
 - ✓ ゴール点 $i=t$ について, $b_t=-f$ ※ この流量制約冗長 (削除可)
 - ✓ それ以外の点 i について, $b_i=0$
 - ✓ スタート点からの枝 (s,j) のコスト $c_{sj}=1$
 - ✓ それ以外の枝 (i,j) のコスト $c_{ij}=0$

最短路問題

最大流問題

参考文献

1. 今野浩 「線形計画法」 日科技連 (1987)
2. 藤田・今野・田邊 「最適化法」 岩波書店 (1994)
3. 田村明久・村松正和 「最適化法」 共立出版 (2002)
4. 坂和正敏 「線形計画法の基礎と応用」 朝倉書店 (2012)
5. 小島・土谷・水野・矢部 「内点法」 朝倉書店 (2001)
6. *A. Schrijver: Theory of Linear and Integer Programming, John Wiley and Sons, 1986.*
7. *L.A. Wolsey: Integer Programming, John Wiley and Sons, 1998.*
8. *M. Conforti, G. Cornuejols and G.Zambelli: Integer Programming, Springer, 2014.*
9. 久保幹雄, J.P.ペドロソ, 村松正和, A.レイス：あたらしい数理最適化, 近代科学社, 2012.
10. 久保幹雄, 小林和博, 斉藤努, 並木誠, 橋本英樹：Python言語によるビジネスアナリティクス, 近代科学社, 2016.