
Evaluation et Optimisation des Systèmes

# Problème de routage incertain multiobjectif

Benjamin FLOUS F3
François BARBIER 12/01/2025

# Table des matières

# I - Introduction

L'objectif de ce tp est d'explorer le problème de routage dans les réseaux de télécommunications, où un opérateur doit acheminer des demandes de trafic entre des ensembles de nœuds à travers un graphe orienté représentant son réseau. Chaque arc du graphe est caractérisé par une capacité installée et un coût de routage par unité de trafic. L'objectif est de répondre aux demandes de trafic tout en optimisant plusieurs critères, en utilisant donc une approche multiobjectif. On détaillera les objectifs plus précisément dans la partie qui suit.

# II - Problèmes de routage

On considère des fonctions objectif antagonistes, d'une part celles liées à l'ingénierie de trafic (allocation des ressources) et d'autre part celles liées à la qualité de son service de réseau privé virtuel. Pour cela, on considère trois fonctions objectif :

1. le coût de routage
2. l'utilisation maximum d'un arc
3. l'utilisation moyenne d'un arc

L'objectif de ce projet est de déterminer le chemin de routage pour toutes les demandes de trafic spécifiées qui correspondent à des solutions optimales ou efficaces pour les trois fonctions objectif mentionnées ci-dessus.

## III - Modèles linéaires

Pour chacune des trois fonctions objectif, on donne le programme linéaire permettant de trouver les chemins de routage optimaux. Tout d'abord, on définit les variables néessaires pour définir les problèmes :

$G = (V, A)$ : *Graphe orienté où $V$ est l'ensemble des nœuds et $A$ l'ensemble des arcs*

$c_a$ : *Capacité installée sur l'arc $a \in A$*

$w_a$ : *Coût de routage unitaire pour l'arc $a \in A$*

$t_{ij}$ : *Volume de trafic à acheminer du nœud $i \in S$ au nœud $j \in S \setminus \{i\}$*

$x^{ij}{}_a$ : *Variable représentant la proportion du trafic $t_{ij}$ passant par l'arc $a \in A$*

$f_a$ : *Flux total traversant l'arc $a$, défini par $f_a = \sum t_{ij} x^{ij}{}_a$ pour $i, j \in S$ et $i != j$*

### Minimisation du coût de routage

*Objectif* : Minimiser le coût total de routage pour toutes les demandes de trafic.

*Programme linéaire* : min $\sum_{a \in A} w_a \cdot f_a$

*Contraintes*

1. *Conservation du flux*

$$\sum_{\text{tous les arcs } a \text{ sortant de } v} x_a^{ij} - \sum_{\text{tous les arcs } a \text{ entrant dans } v} x_a^{ij} = \begin{cases} 1 & \text{si } v = i, \\ -1 & \text{si } v = j, \\ 0 & \text{sinon.} \end{cases}$$

2. *Capacité des arcs*

$$f_a \leq c_a, \quad \forall a \in A$$

3. *Non négativité sur la proportion de trafic*

$$0 \leq x_a^{ij} \leq 1, \quad \forall a \in A, \forall i, j \in S, i \neq j$$

## Minimisation de l'utilisation maximale d'un arc

*Objectif* : Minimiser l'utilisation maximale sur tous les arcs du réseau. Pour cela, on introduit une variable auxiliaire *U* représentant l'utilisation maximale des arcs.

*Programme linéaire* : min *U*

*Contraintes*

1. *Utilisation maximale des arcs*

$$f_a \leq U \,.\, c_a \,,\ \forall\ a \in A$$

2. *Mêmes contraines que précédemment*

## Minimisation de l'utilisation moyenne des arcs

*Objectif* : Minimiser l'utilisation moyenne des arcs

*Programme linéaire* : min $\frac{1}{|A|} \sum_{a \in A} \frac{f_a}{c_a}$

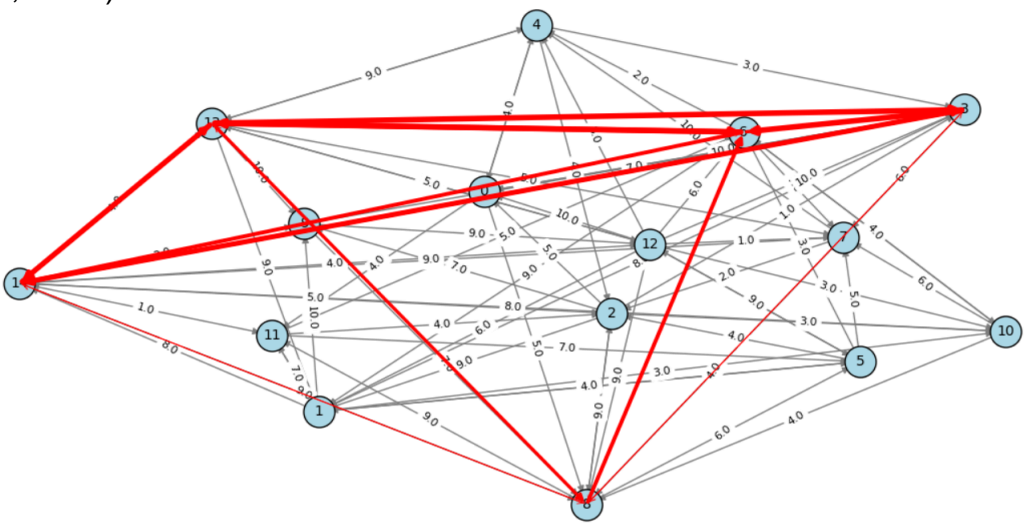
*Contraintes identiques à celle du premier programme linéaire*

## IV - Génération d'instances

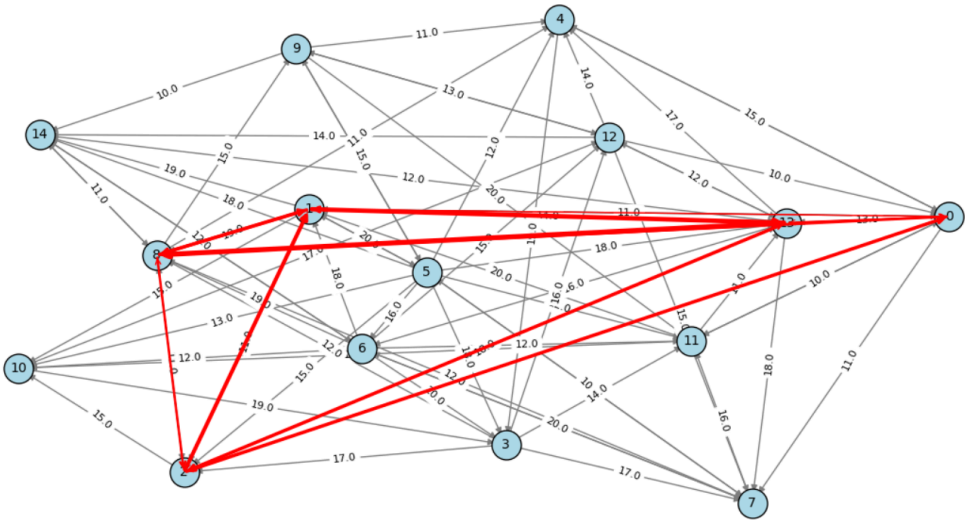
L'idée de cette partie est d'écrire un programme permettant de générer plusieurs types de graphes et donner plusieurs types d'information :

- des graphes appartenant à des classes données (complet, cycle, aléatoire)
- les capacités associées aux arcs des graphes
- les coûts de routage unitaire associés aux arcs des graphes
- des sous-ensembles *S* de nœuds et les vecteurs de demandes de trafic associés

Le résultat est contenu dans un fichier csv. Voici un exemple de graphe aléatoire (15 noeuds, S = 5) :



On peut voir en rouge les demandes de trafic entre les nœuds du sous-ensemble *S*, avec leur épaisseur proportionnelle au volume de trafic demandé.



On voit ci-dessus un autre exemple de graphe avec les mêmes paramètres (nombre de sommets et sous-ensemble *S*).

## V - Résolution des programmes linéaires

L'idée de cette partie est de construire un programme qui résout un problème linéaire(PL) mono-objectif. Il prendra en argument une instance de graphe, la demande de trafic et la fonction objectif choisie.

On définit pour cela les variables de décisions, puis les différentes contraintes. Ensuite, on liste les différentes fonctions objectif possibles. Il y aussi une partie débogage en cas d'infaisabilité du PL.

## VI - Expérimentations

Pour tester ce programme, nous avons créé une fonction *test_program* permettant de tester ce programme pour différentes fonctions objectif et aussi différents types de graphes. Il prend en entrée plusieurs paramètres :

- graph_type (string): type de graphe
- n_nodes (int): nombre de node dans *V*
- S_size (int): nombre de node dans *S*
- demand_range (tuple): tuple de 2 int pour le range des valeurs de demande
- capacity_range (tuple): tuple de 2 int pour le range des valeurs de capacité
- cost_range (tuple): tuple de 2 int pour le range des valeurs de capacité
- objective (string): nom de la fonction objectif choisie (cost, max_utlilization, avg_utlilization)

Pour un graphe de type random avec pour paramètres en choissisant l'objectif de minimisation du coût (noeuds = 15, capacity_range = (10, 20), cost_range = (1.0, 5.0), S_size = 5, demand_range = (1, 20)), on obtient avec Gurobi :

```
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (win64 - Windows 11.0 (22631.2))

CPU model: AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx, instruction set [SSE2|AVX|AVX2]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 373 rows, 1460 columns and 4380 nonzeros
Model fingerprint: 0xbcbcf2da
Coefficient statistics:
  Matrix range     [1e+00, 2e+01]
  Objective range  [3e+00, 9e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+01]
Presolve removed 20 rows and 0 columns
Presolve time: 0.02s
Presolved: 353 rows, 1460 columns, 4161 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   5.500000e+01   0.000000e+00      0s
     234    9.8133277e+02   0.000000e+00   0.000000e+00      0s

Solved in 234 iterations and 0.03 seconds (0.00 work units)
Optimal objective  9.813327696e+02
Solution optimale pour cost: {(13, 2, 3, 2): 1.0, (13, 2, 4, 3): 1.0, (13, 2, 13, 4): 1.0, (13, 0, 12, 0): 1.0, (13, 0, 13, 1
2): 1.0, (13, 8, 6, 8): 0.8421052631578947, (13, 8, 12, 14): 0.1578947368421053, (13, 8, 13, 6): 0.8421052631578947, (13, 8, 13
, 12): 0.1578947368421053, (13, 8, 14, 8): 0.1578947368421053, (13, 1, 5, 1): 1.0, (13, 1, 7, 5): 1.0, (13, 1, 13, 7): 1.0, (2,
13, 2, 13): 1.0, (2, 0, 2, 8): 0.5384615384615384, (2, 0, 2, 13): 0.46153846153846156, (2, 0, 4, 0): 0.7692307692307693, (2, 0,
8, 4): 0.46153846153846156, (2, 0, 8, 9): 0.07692307692307687, (2, 0, 9, 11): 0.07692307692307687, (2, 0, 11, 0): 0.07692307692
307687, (2, 0, 12, 0): 0.15384615384615385, (2, 0, 13, 4): 0.3076923076923077, (2, 0, 13, 12): 0.15384615384615385, (2, 8, 2,
8): 1.0, (2, 1, 2, 8): 0.4, (2, 1, 2, 10): 0.5333333333333333, (2, 1, 2, 13): 0.06666666666666665, (2, 1, 5, 1): 0.066666666666
66667, (2, 1, 6, 1): 0.7999999999999999, (2, 1, 7, 5): 0.06666666666666667, (2, 1, 8, 9): 0.1333333333333334, (2, 1, 8, 14): 0.
2666666666666666, (2, 1, 9, 11): 0.1333333333333334, (2, 1, 10, 6): 0.5333333333333333, (2, 1, 11, 1): 0.1333333333333334, (2,
1, 13, 7): 0.06666666666666667, (2, 1, 14, 6): 0.2666666666666666, (0, 13, 0, 13): 1.0, (0, 2, 0, 1): 1.0, (0, 2, 1, 2): 1.0,
(0, 8, 0, 11): 1.0, (0, 8, 1, 8): 0.6, (0, 8, 6, 8): 0.3, (0, 8, 6, 14): 0.09999999999999987, (0, 8, 10, 6): 0.3999999999999999
, (0, 8, 11, 1): 0.6000000000000001, (0, 8, 11, 10): 0.3999999999999999, (0, 8, 14, 8): 0.09999999999999987, (0, 1, 0, 1): 1.0,
(8, 13, 8, 14): 1.0, (8, 13, 14, 13): 1.0, (8, 2, 8, 2): 1.0, (8, 0, 4, 0): 1.0, (8, 0, 8, 4): 1.0, (8, 1, 8, 9): 1.0, (8, 1, 9
, 11): 1.0, (8, 1, 11, 1): 1.0, (1, 13, 1, 2): 0.11764705882352941, (1, 13, 1, 13): 0.8235294117647058, (1, 13, 1, 14): 0.05882
352941176475, (1, 13, 2, 13): 0.11764705882352941, (1, 13, 14, 13): 0.05882352941176475, (1, 2, 1, 2): 1.0, (1, 0, 1, 0): 1.0,
(1, 8, 1, 8): 1.0}
Valeur optimale pour cost: 981.3327695886198
```

Pour un graphe de type complet en choisissant l'objectif de minimisation du coût (noeuds = 8, capacity_range = (10, 20), cost_range = (1.0, 5.0), S_size = 3, demand_range = (1, 20)), on obtient avec Gurobi :

```
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (win64 - Windows 11.0 (22631.2))

CPU model: AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx, instruction set [SSE2|AVX|AVX2]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 104 rows, 336 columns and 1008 nonzeros
Model fingerprint: 0xfa145377
Coefficient statistics:
  Matrix range     [1e+00, 2e+01]
  Objective range  [2e+00, 9e+01]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+01]
Presolve removed 6 rows and 0 columns
Presolve time: 0.02s
Presolved: 98 rows, 336 columns, 924 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    0.0000000e+00   1.600000e+01   0.000000e+00      0s
      38    2.1551595e+02   0.000000e+00   0.000000e+00      0s

Solved in 38 iterations and 0.03 seconds (0.00 work units)
Optimal objective  2.155159547e+02
Solution optimale pour cost: {(7, 3, 7, 3): 1.0, (7, 0, 7, 0): 1.0, (3, 7, 3, 7): 1.0, (3, 0, 3, 0): 1.0, (0, 7, 0, 1): 0.11111
111111111116, (0, 7, 0, 7): 0.8888888888888888, (0, 7, 1, 7): 0.11111111111111116, (0, 3, 0, 2): 0.0625, (0, 3, 0, 3): 0.9375,
(0, 3, 2, 3): 0.0625}
Valeur optimale pour cost: 215.5159546866027
```

Pour un graphe de type cycle en choissisant l'objectif de minimisation du coût (noeuds = 6, capacity_range = (40, 60), cost_range = (1.0, 5.0), S_size = 3, demand_range = (1, 20)), on obtient avec Gurobi :

```
Gurobi Optimizer version 12.0.0 build v12.0.0rc1 (win64 - Windows 11.0 (22631.2))

CPU model: AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx, instruction set [SSE2|AVX|AVX2]
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 42 rows, 36 columns and 108 nonzeros
Model fingerprint: 0x2a2bbebd
Coefficient statistics:
  Matrix range     [1e+00, 2e+01]
  Objective range  [3e+00, 1e+02]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 6e+01]
Presolve removed 42 rows and 36 columns
Presolve time: 0.00s
Presolve: All rows and columns removed
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    5.1660903e+02   0.000000e+00   0.000000e+00      0s

Solved in 0 iterations and 0.02 seconds (0.00 work units)
Optimal objective  5.166090321e+02
Solution optimale pour cost: {(1, 0, 1, 2): 1.0, (1, 0, 2, 3): 1.0, (1, 0, 3, 4): 1.0, (1, 0, 4, 5): 1.0, (1, 0, 5, 0): 1.0,
(1, 2, 1, 2): 1.0, (0, 1, 0, 1): 1.0, (0, 2, 0, 1): 1.0, (0, 2, 1, 2): 1.0, (2, 1, 0, 1): 1.0, (2, 1, 2, 3): 1.0, (2, 1, 3, 4):
1.0, (2, 1, 4, 5): 1.0, (2, 1, 5, 0): 1.0, (2, 0, 2, 3): 1.0, (2, 0, 3, 4): 1.0, (2, 0, 4, 5): 1.0, (2, 0, 5, 0): 1.0}
Valeur optimale pour cost: 516.6090321321374
```

On peut voir que les résultats diffèrent d'un type de graphe à un autre, cela est aussi lié au fait que les paramètres ne sont pas les mêmes.

# VII - Obtention de solutions efficaces

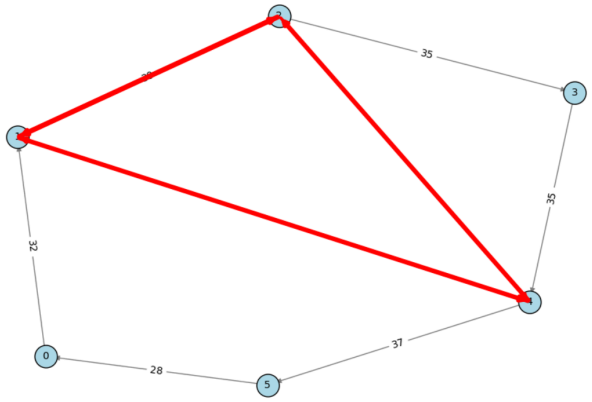
Dans cette partie, nous allons implémenter les méthodes :

- d'optimisation léxicographique,
- d'$\epsilon$-contraintes,
- de sommes ponderées

Le but est d'obtenir des solutions efficaces pour le probléme de routage avec les trois fonctions objectif définies précédemment.

## Optimisation léxicographique

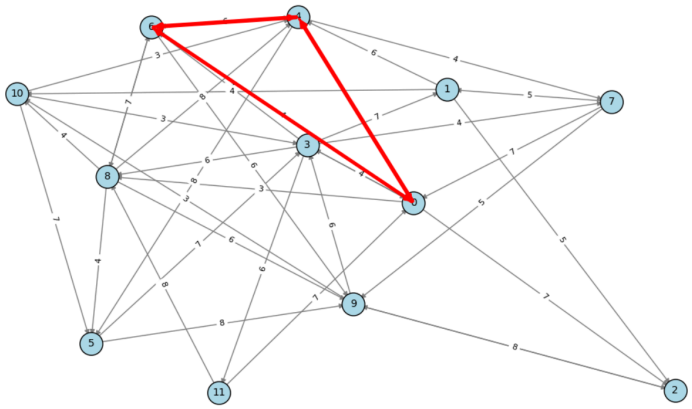
1. *Graphe de type cycle*




On obtient avec Gurobi les résultats suivants :

```
Solved in 0 iterations and 0.02 seconds (0.00 work units)
Optimal objective  7.607755420e-01
Solution optimale : {(4, 1, 0, 1): 1.0, (4, 1, 4, 5): 1.0, (4, 1, 5, 0): 1.0, (4, 2, 0, 1): 1.0, (4, 2, 1, 2): 1.0, (4, 2, 4,
5): 1.0, (4, 2, 5, 0): 1.0, (1, 4, 1, 2): 1.0, (1, 4, 2, 3): 1.0, (1, 4, 3, 4): 1.0, (1, 2, 1, 2): 1.0, (2, 4, 2, 3): 1.0, (2,
4, 3, 4): 1.0, (2, 1, 0, 1): 1.0, (2, 1, 2, 3): 1.0, (2, 1, 3, 4): 1.0, (2, 1, 4, 5): 1.0, (2, 1, 5, 0): 1.0}
Résultats lexicographiques : {'cost': 536.0736391667267, 'max_utilization': 0.9285714285714285, 'avg_utilization': 0.7607755420
25542}
```

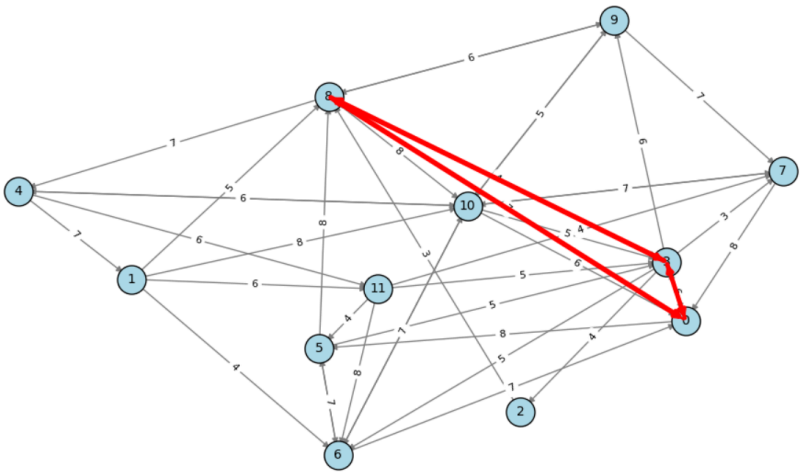
2. *Graphe de type complet*



On obtient avec Gurobi les résultats suivants :

```
Solved in 42 iterations and 0.02 seconds (0.00 work units)
Optimal objective  2.019761797e+02
Solution optimale : {(3, 2, 1, 2): 0.16666666666666674, (3, 2, 3, 1): 0.16666666666666674, (3, 2, 3, 2): 0.5, (3, 2, 3, 4): 0.3
333333333333333, (3, 2, 4, 2): 0.3333333333333333, (3, 0, 1, 0): 0.22222222222222227, (3, 0, 3, 0): 0.6666666666666666, (3, 0,
3, 1): 0.22222222222222227, (3, 0, 3, 5): 0.1111111111111111, (3, 0, 5, 0): 0.1111111111111111, (2, 3, 1, 3): 0.384615384615384
6, (2, 3, 2, 1): 0.3846153846153847, (2, 3, 2, 3): 0.6153846153846154, (2, 0, 1, 0): 0.0833333333333333, (2, 0, 2, 0): 0.833333
3333333334, (2, 0, 2, 1): 0.08333333333333329, (2, 0, 2, 4): 0.08333333333333345, (2, 0, 4, 0): 0.08333333333333345, (0, 3, 0,
3): 1.0, (0, 2, 0, 1): 0.5, (0, 2, 0, 2): 0.5, (0, 2, 1, 2): 0.5}
Coût total : 201.97617969572576
Utilisation maximale : 2.5
Utilisation moyenne : 0.41687830687830696
```

3. *Graphe de type aléatoire*

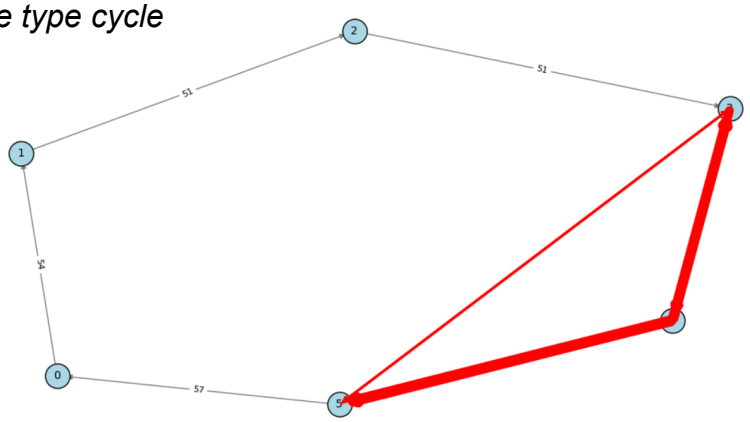



On obtient avec Gurobi les résultats suivants :

```
Solved in 37 iterations and 0.04 seconds (0.00 work units)
Optimal objective  1.687058639e+02
Solution optimale : {(3, 8, 2, 8): 0.375, (3, 8, 3, 2): 0.375, (3, 8, 3, 6): 0.625, (3, 8, 5, 8): 0.625, (3, 8, 6, 5): 0.625,
(3, 0, 3, 0): 0.8571428571428571, (3, 0, 3, 7): 0.1428571428571429, (3, 0, 7, 0): 0.1428571428571429, (8, 3, 4, 11): 0.33333333
333333337, (8, 3, 8, 3): 0.6666666666666666, (8, 3, 8, 4): 0.33333333333333337, (8, 3, 11, 3): 0.33333333333333337, (8, 0, 6,
0): 0.25, (8, 0, 8, 10): 1.0, (8, 0, 10, 0): 0.75, (8, 0, 10, 6): 0.25, (0, 3, 0, 3): 1.0, (0, 8, 0, 8): 1.0}
Coût total : 168.70586388987553
Utilisation maximale : 2.5
Utilisation moyenne : 0.28559814169570263
```

## ***ϵ*-contraintes**

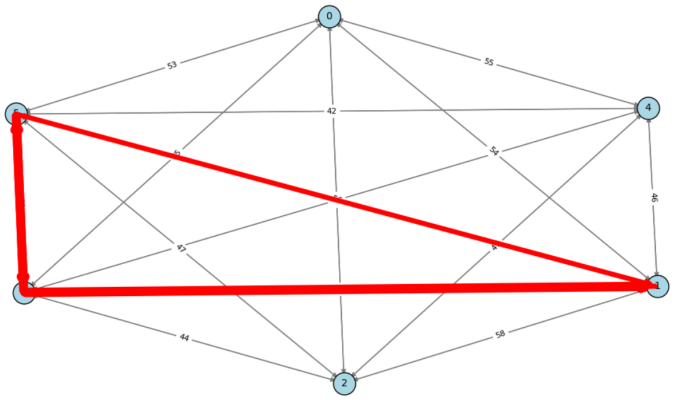
1. *Graphe de type cycle*



On obtient avec Gurobi les résultats suivants :

```
Solved in 0 iterations and 0.02 seconds (0.00 work units)
Optimal objective  5.714455201e+02
Solution optimale : {(3, 5, 3, 4): 1.0, (3, 5, 4, 5): 1.0, (3, 4, 3, 4): 1.0, (5, 3, 0, 1): 1.0, (5, 3, 1, 2): 1.0, (5, 3, 2,
3): 1.0, (5, 3, 5, 0): 1.0, (5, 4, 0, 1): 1.0, (5, 4, 1, 2): 1.0, (5, 4, 2, 3): 1.0, (5, 4, 3, 4): 1.0, (5, 4, 5, 0): 1.0, (4,
3, 0, 1): 1.0, (4, 3, 1, 2): 1.0, (4, 3, 2, 3): 1.0, (4, 3, 4, 5): 1.0, (4, 3, 5, 0): 1.0, (4, 5, 4, 5): 1.0}
Coût total : 571.4455200844237
Utilisation maximale : 2.5
Utilisation moyenne : 0.5226168170391212
```

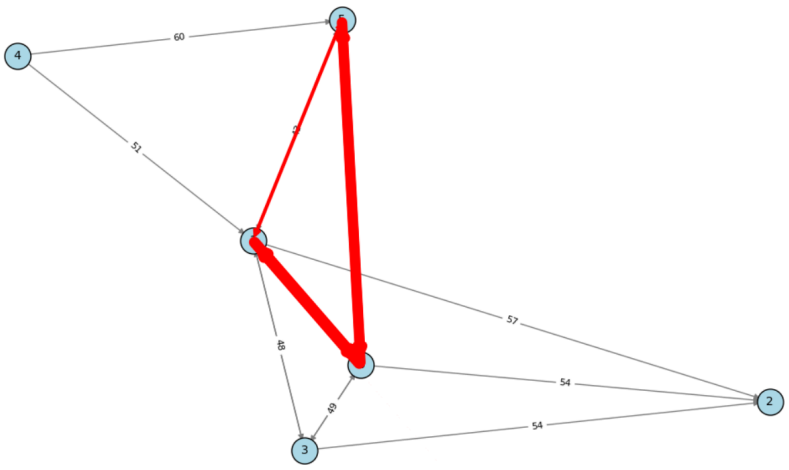
## 2. *Graphe de type complet*




On obtient avec Gurobi les résultats suivants :

```
Solved in 17 iterations and 0.01 seconds (0.00 work units)
Optimal objective  2.641899086e+02
Solution optimale : {(1, 3, 1, 3): 1.0, (1, 5, 1, 5): 1.0, (3, 1, 3, 1): 1.0, (3, 5, 3, 5): 1.0, (5, 1, 0, 1): 1.0, (5, 1, 5,
0): 1.0, (5, 3, 5, 3): 1.0}
Coût total : 264.18990859195384
Utilisation maximale : 2.5
Utilisation moyenne : 0.0551585869862173
```

## 3. *Graphe de type aléatoire*

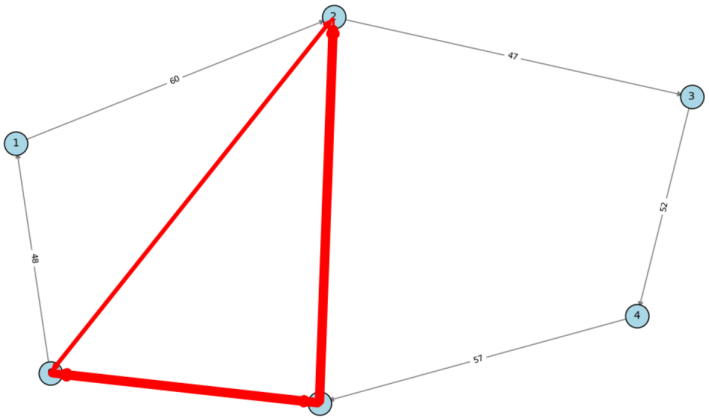


On obtient avec Gurobi les résultats suivants :

```
Solved in 0 iterations and 0.02 seconds (0.00 work units)
Optimal objective  5.277908501e+02
Solution optimale : {(1, 0, 1, 3): 1.0, (1, 0, 3, 0): 1.0, (1, 5, 1, 5): 1.0, (0, 1, 0, 3): 1.0, (0, 1, 3, 1): 1.0, (0, 5, 0,
3): 1.0, (0, 5, 1, 5): 1.0, (0, 5, 3, 1): 1.0, (5, 1, 5, 1): 1.0, (5, 0, 1, 3): 1.0, (5, 0, 3, 0): 1.0, (5, 0, 5, 1): 1.0}
Coût total : 527.790850053551
Utilisation maximale : 2.5
Utilisation moyenne : 0.3730891453256297
```

## Sommes ponderées

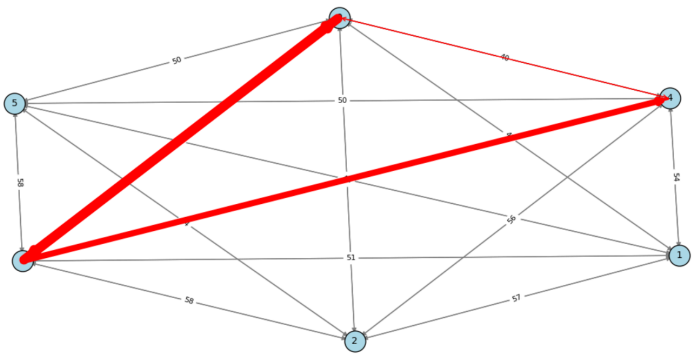
1. *Graphe de type cycle*




On obtient avec Gurobi les résultats suivants :

```
Solved in 0 iterations and 0.02 seconds (0.00 work units)
Optimal objective  3.015397463e+02
Solution optimale : {(2, 5, 2, 3): 1.0, (2, 5, 3, 4): 1.0, (2, 5, 4, 5): 1.0, (2, 0, 2, 3): 1.0, (2, 0, 3, 4): 1.0, (2, 0, 4,
5): 1.0, (2, 0, 5, 0): 1.0, (5, 2, 0, 1): 1.0, (5, 2, 1, 2): 1.0, (5, 2, 5, 0): 1.0, (5, 0, 5, 0): 1.0, (0, 2, 0, 1): 1.0, (0,
2, 1, 2): 1.0, (0, 5, 0, 1): 1.0, (0, 5, 1, 2): 1.0, (0, 5, 2, 3): 1.0, (0, 5, 3, 4): 1.0, (0, 5, 4, 5): 1.0}
Coût total : 602.2919260443043
Utilisation maximale : 0.875
Utilisation moyenne : 0.6564164178175935
```

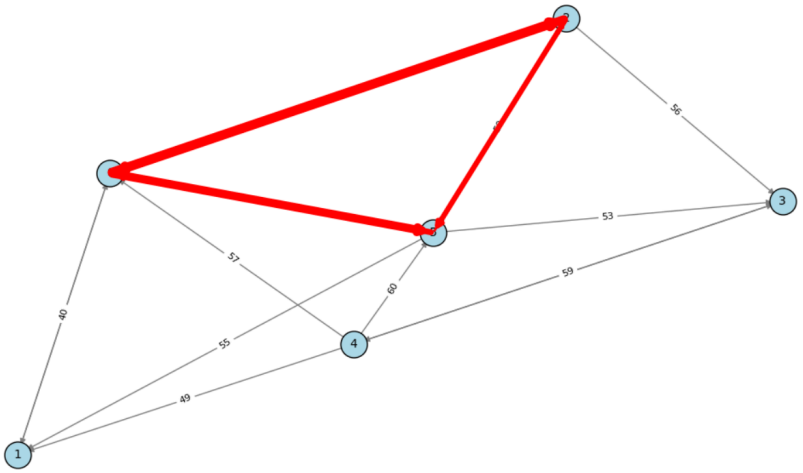
2. *Graphe de type complet*



On obtient avec Gurobi les résultats suivants :

```
Solved in 15 iterations and 0.02 seconds (0.00 work units)
Optimal objective  7.500975122e+01
Solution optimale : {(4, 0, 2, 0): 1.0, (4, 0, 4, 2): 1.0, (4, 3, 4, 3): 1.0, (0, 4, 0, 3): 1.0, (0, 4, 3, 4): 1.0, (0, 3, 0,
3): 1.0, (3, 4, 3, 4): 1.0, (3, 0, 2, 0): 1.0, (3, 0, 3, 2): 1.0}
Coût total : 149.68339119102671
Utilisation maximale : 0.5238095238095237
Utilisation moyenne : 0.054563814031359674
```

### 3. *Graphe de type aléatoire*




On obtient avec Gurobi les résultats suivants :

```
Solved in 10 iterations and 0.04 seconds (0.00 work units)
Optimal objective  2.292625660e+02
Solution optimale : {(2, 5, 2, 5): 1.0, (2, 0, 2, 3): 1.0, (2, 0, 3, 4): 1.0, (2, 0, 4, 0): 1.0, (5, 2, 5, 2): 1.0, (5, 0, 1,
0): 1.0, (5, 0, 5, 1): 1.0, (0, 2, 0, 2): 1.0, (0, 5, 0, 2): 1.0, (0, 5, 2, 5): 1.0}
Coût total : 458.08523168050806
Utilisation maximale : 0.6122448979591837
Utilisation moyenne : 0.18138363116176806
```

## VIII - Problème de routage incertain

On considère dans cette partie de nouveau le problème de routage introduit dans la première partie. Nous allons maintenant considérer la situation où les vecteurs de demandes de trafic et les vecteurs de coûts unitaires de routage ne sont pas explicitement connus.

On peut voir ci-dessous la contrepartie robuste du problème initial :

```python
# Fonction objectif robuste
if objective == 'cost':
    cost = sum(
        UW[(u, v)]['max'] * x[(i, j, u, v)] for u, v in G.edges for (i, j), bounds in UT.items()
    )
    model.setObjective(cost, GRB.MINIMIZE)
elif objective == 'max_utilization':
    max_utilization = model.addVar(lb=0, ub=GRB.INFINITY, name="max_utilization")
    for u, v in G.edges:
        model.addConstr(
            max_utilization >=
            sum(bounds['max'] * x[(i, j, u, v)] for (i, j), bounds in UT.items()) / G[u][v]['capacity'],
            name=f"utilization_{u}_{v}"
        )
    model.setObjective(max_utilization, GRB.MINIMIZE)
elif objective == 'avg_utilization':
    avg_utilization = sum(
        sum(bounds['max'] * x[(i, j, u, v)] for (i, j), bounds in UT.items()) / G[u][v]['capacity']
        for u, v in G.edges
    ) / G.number_of_edges()
    model.setObjective(avg_utilization, GRB.MINIMIZE)
else:
    raise ValueError("Objectif non reconnu")
```

## IX - Conclusion

Le problème de routage de trafic dans des réseaux virtuels privés a été abordé sous l'angle de l'optimisation, en prenant en compte plusieurs fonctions objectif antagonistes. L'objectif principal était de déterminer les meilleurs chemins de routage pour acheminer du trafic tout en équilibrant les objectifs liés au trafic et à la qualité du service.

Les résultats obtenus ont varié de manière assez forte en fonction de la méthode d'optimisation choisie, de la structure du graphe utilisé, ainsi que des paramètres comme les coûts de routage et les capacités des arcs. Les approches multi-objectifs permettent de mieux équilibrer les compromis entre coût, qualité de service et utilisation des ressources, tandis que les techniques robustes garantissent des solutions plus flexibles face à l'incertitude.