MÉTRO ET GRAPHES

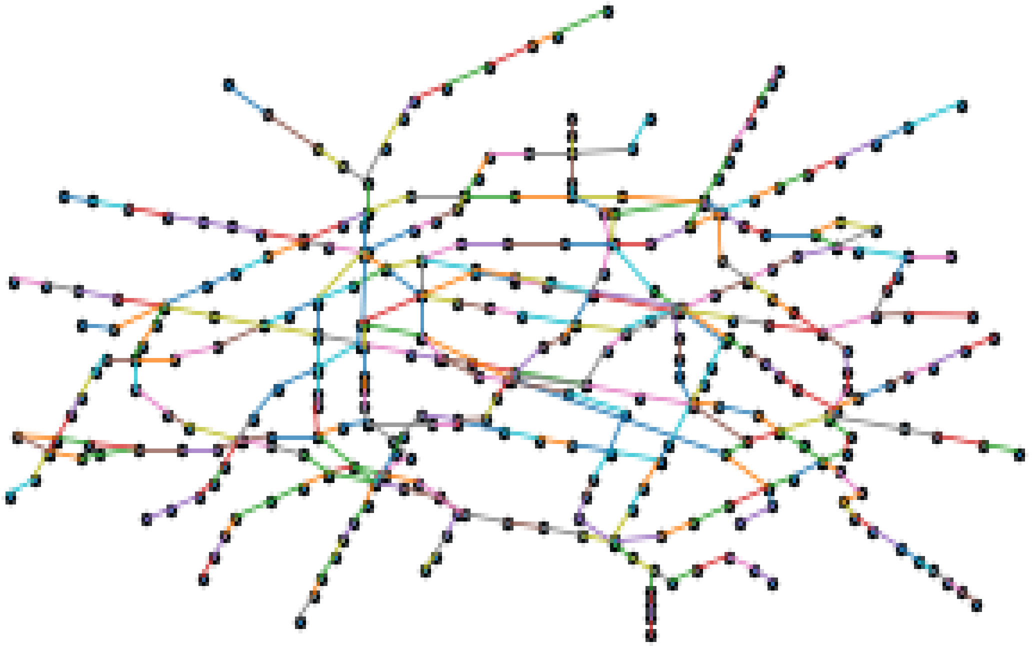
Maximilien HANTONNE
Candidat n°46419

Option Informatique

SOMMAIRE

- Présentation de l’étude
- Génération de graphes
- Analyse et comparaison des graphes
- Conclusion

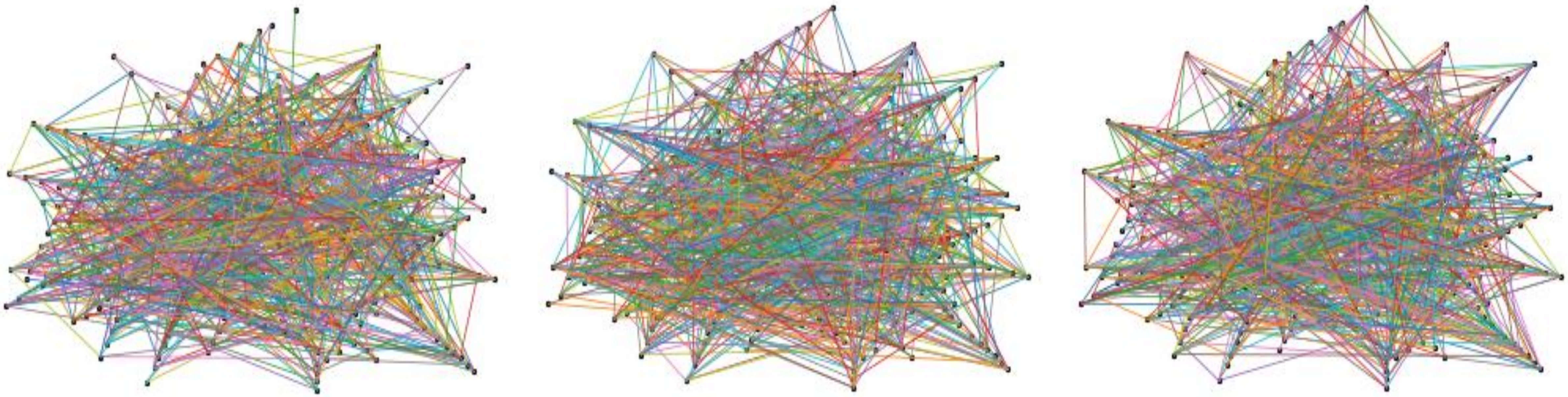
PRÉSENTATION DE L'ÉTUDE



GÉNÉRATION DE GRAPHES

MODÈLES ALÉATOIRES

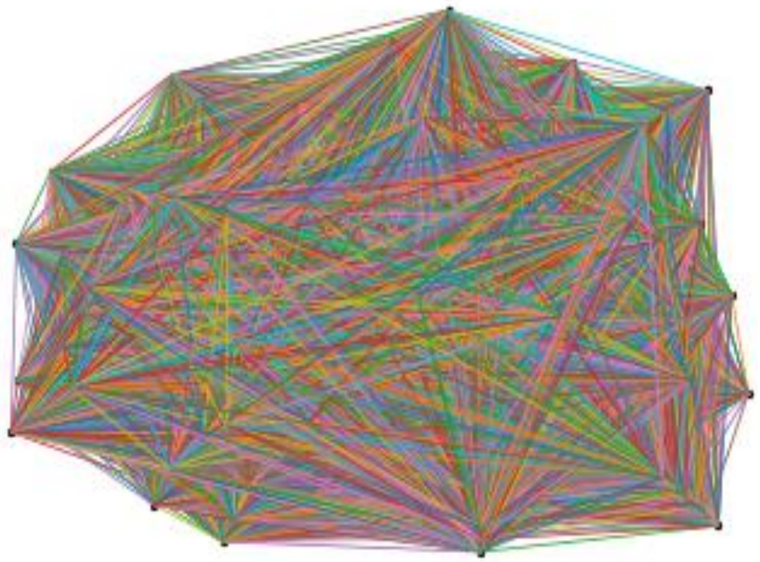
- *Modèle purement aléatoire*



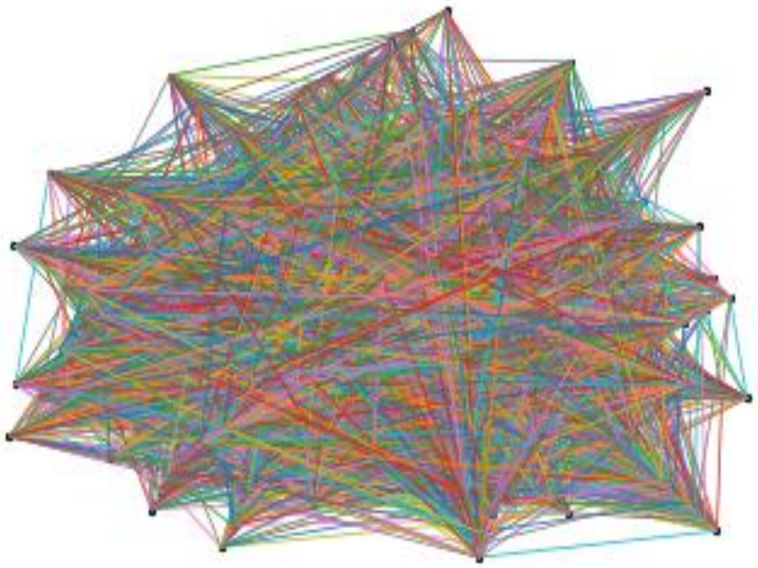
GÉNÉRATION DE GRAPHES

MODÈLES ALÉATOIRES

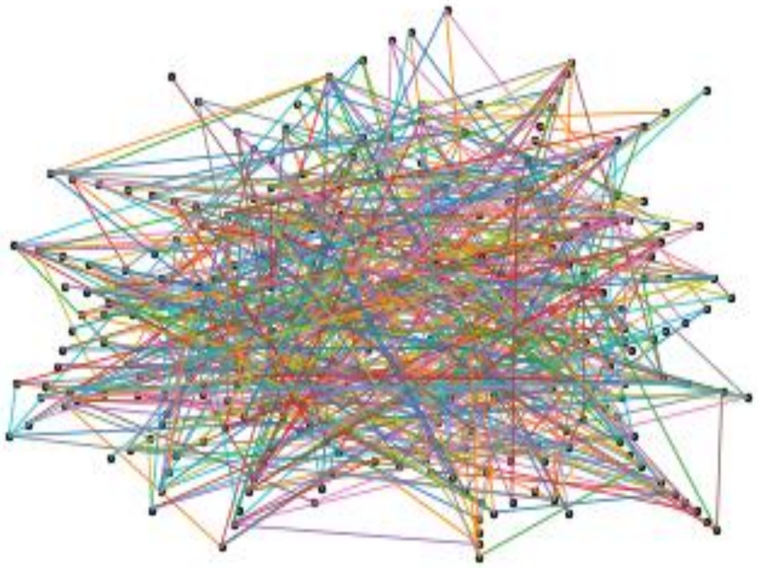
- *Modèle binomial ou modèle d'Erdös-Renyi*



p = 0,5



p = 0,05



p = 0,005

GÉNÉRATION DE GRAPHES

MODÈLES SEMI-ALÉATOIRES

- *M*odèle aléatoire prenant en compte les distances

$$p_{i,j} = \left(\frac{m}{D_{i,j}}\right)^k$$

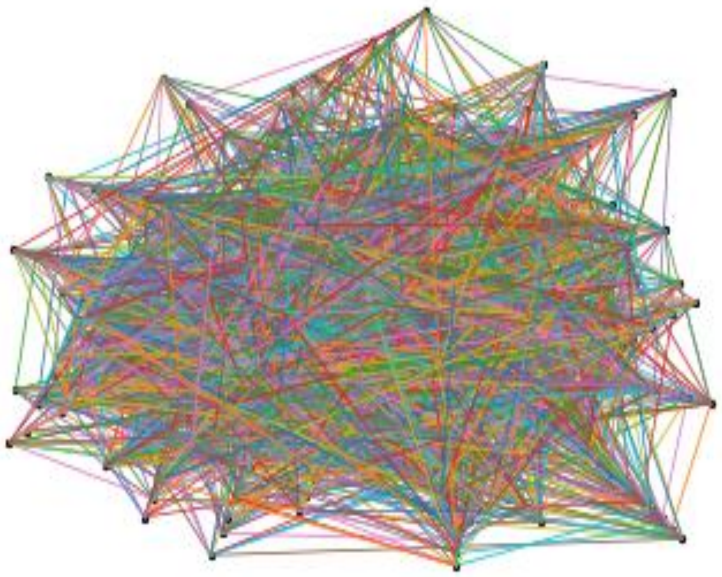
Où $D_{i,j}$ est la distance réelle entre les gares i et j

et m = $min_{i,j \in V}\ D_{i,j}$

GÉNÉRATION DE GRAPHES

MODÈLES SEMI-ALÉATOIRES

- *Modèle* aléatoire prenant en compte les distances



k = 1

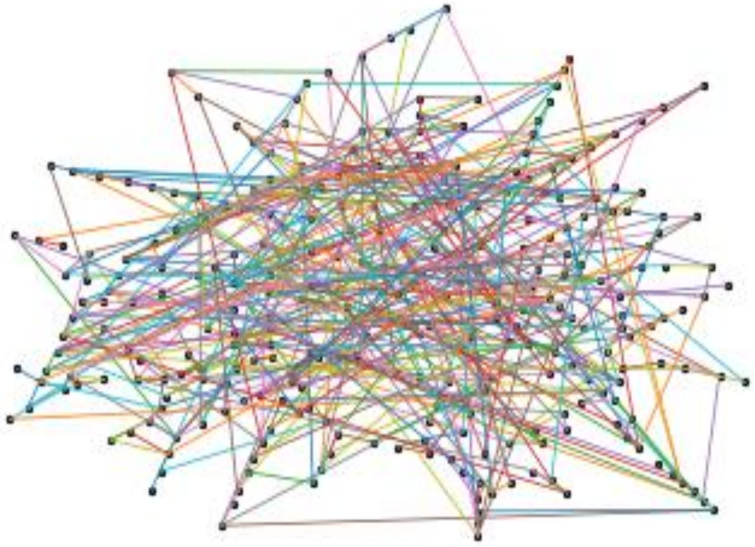
k = 2

k = 4

GÉNÉRATION DE GRAPHES

MODÈLES SEMI-ALÉATOIRES

- *Modèle petit monde (ou modèle de Watts-Strogatz)*



k = 3 et p = 0,5

k = 3 et p = 0,1

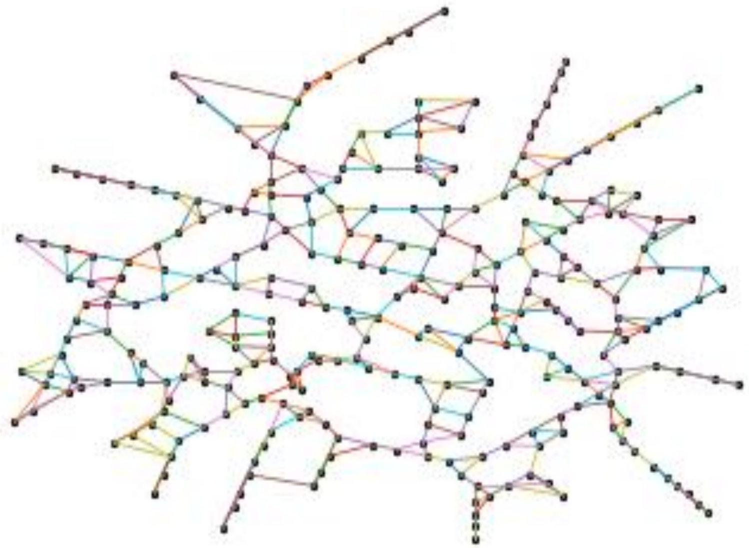
k = 3 et p = 0,05

GÉNÉRATION DE GRAPHES

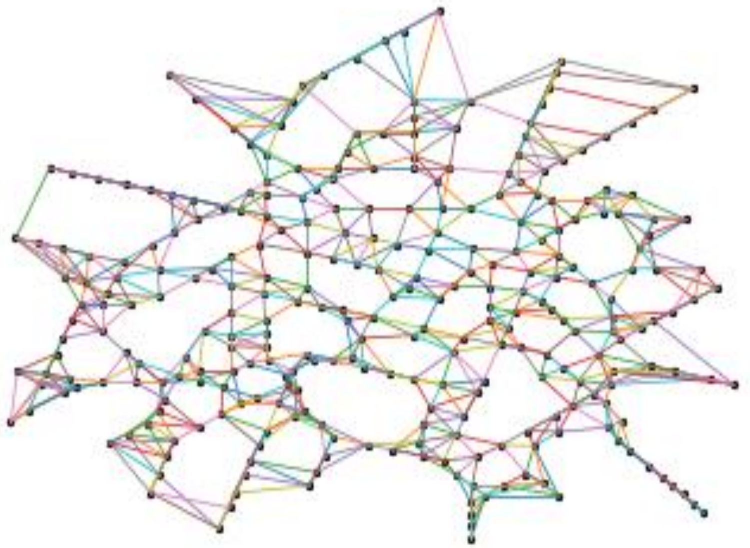
MODÈLES DÉTERMINISTES

- *Modèle petit monde ou modèle de Watts-Strogatz*

k = 2 et p = 0



k = 3 et p = 0



k = 5 et p = 0

GÉNÉRATION DE GRAPHES

MODÈLES DÉTERMINISTES

- *Modèle géométrique:*

$$R = K \times R_{min}$$

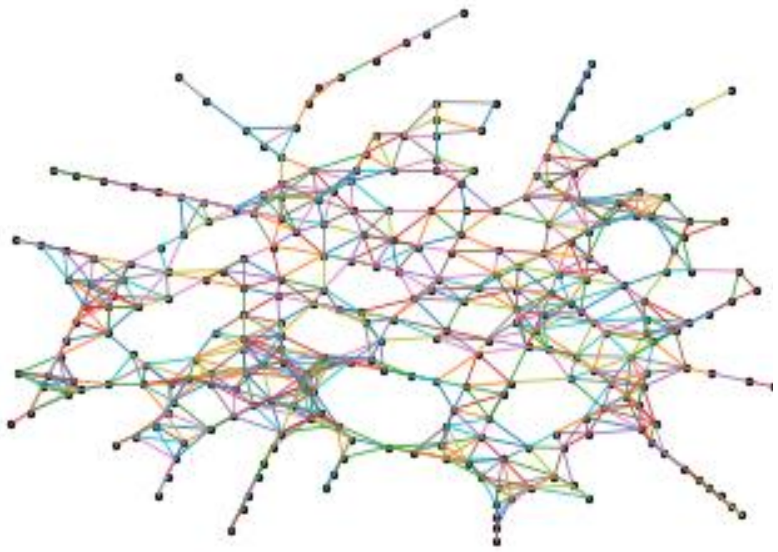
Où K est un réel supérieur à 1

Et R_{min} le rayon minimal pour que le graphe soit connexe

GÉNÉRATION DE GRAPHES

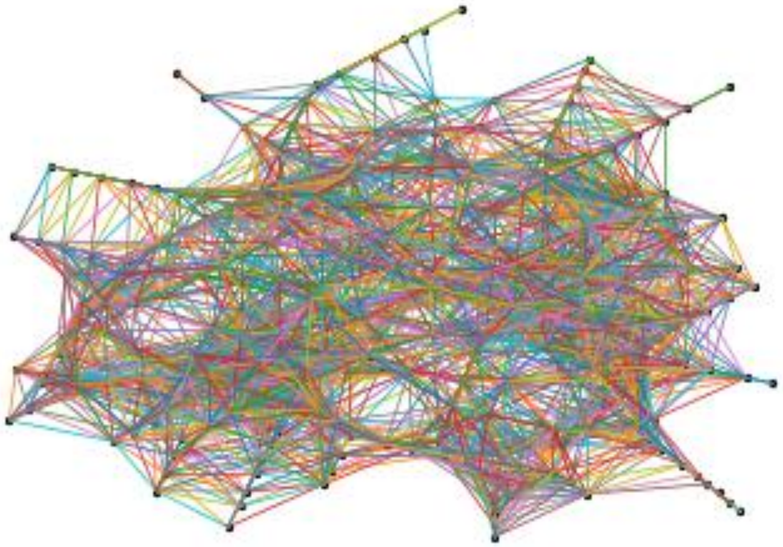
MODÈLES DÉTERMINISTES

- *Modèle* géométrique



K = 1

K = 1,25



K = 2

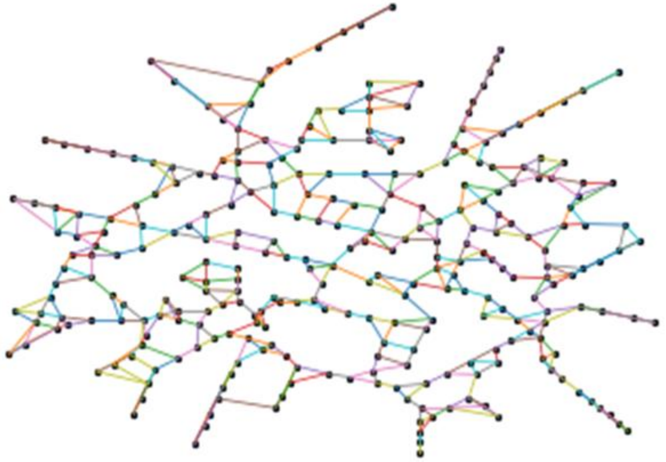
ANALYSE ET COMPARAISON DE GRAPHES

Modèle purement aléatoire

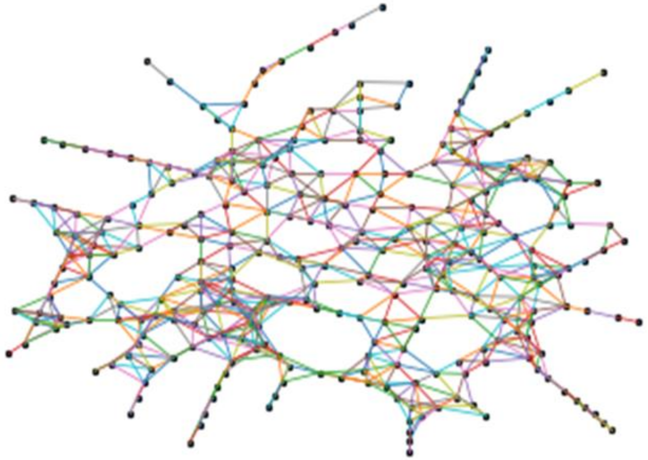
Modèle binomial (p=0,05)

Modèle semi-aléatoire (k=4)

Modèle petit monde (k=3 et p=0,1)



Modèle petit monde (k=3 et p=0)



Modèle géométrique (K=1)

ANALYSE ET COMPARAISON DES GRAPHES

ANALYSE DES GRAPHES

- Coût du graphe:

L x 150 millions d'euros

Où L est la longueur totale des arêtes (en km)

- Efficacité du graphe :

Efficacité: $\mathrm{E}(G) = \frac{1}{N(N-1)} \sum_{i \neq j \in V} \frac{1}{d_{i,j}}$

Efficacité globale: $\mathrm{E}_{glob}(G) = \frac{E(G)}{E(G^{complet})}$

Où $d_{i,j}$ est la distance dans le graphe entre les gares i et j

ANALYSE ET COMPARAISON DES GRAPHES

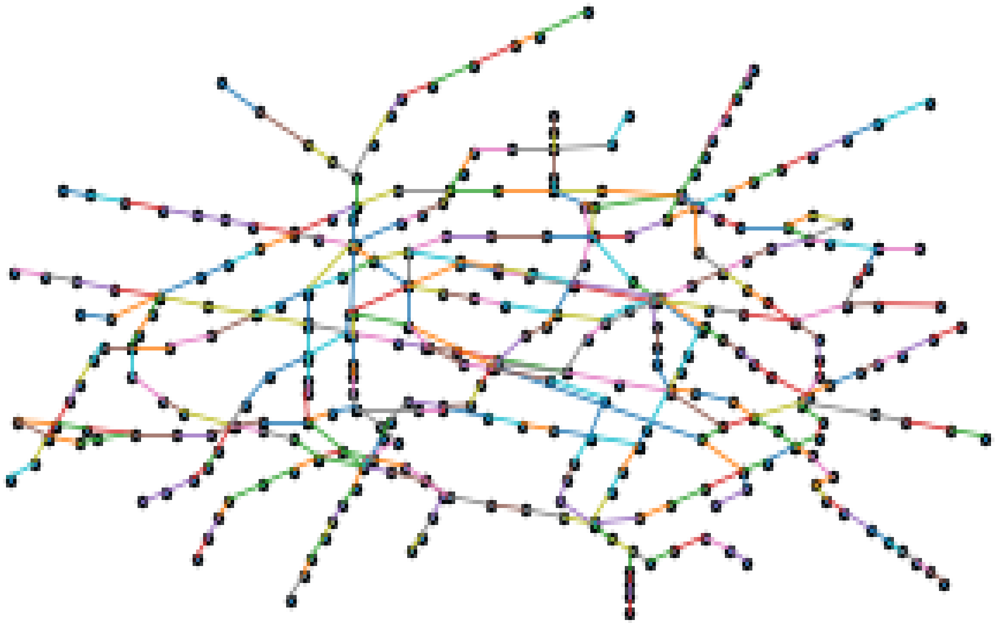
ANALYSE DES GRAPHES

- Robustesse du graphe: *(par stratégie d'attaque aléatoire)*

$$R = \frac{E_{glob}(G_{après\ attaque})}{E_{glob}(G_{avant\ attaque})}$$

ANALYSE ET COMPARAISON DES GRAPHES

COMPARAISON DES GRAPHES



Coût : 23 milliards d’euros

Efficacité globale : 0,81

Robustesse : 0,89

Modèle purement aléatoire

Modèle binomial

Modèle semi-aléatoire

Coût : 350 milliards d'euros

Efficacité globale: 0,25

Robustesse : 0,978

Coût : 400 milliards d'euros

Efficacité globale: 0,30

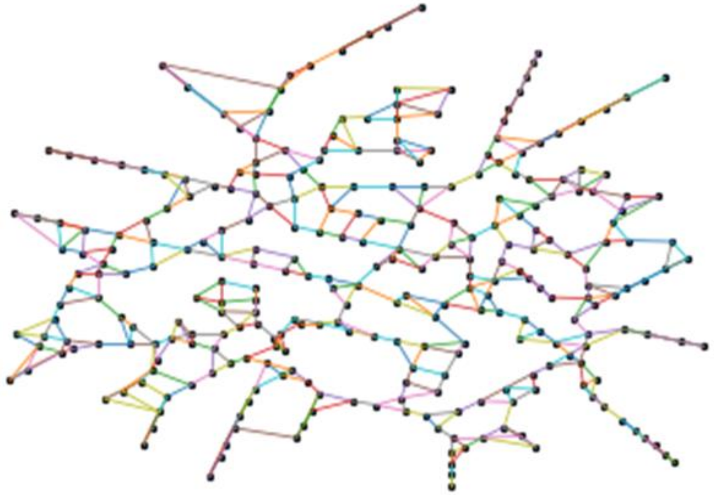
Robustesse : 0,982

Coût : 120 milliards d'euros

Efficacité globale: 0,91

Robustesse : 0,992

Modèle petit monde aléatoire



Modèle petit monde déterministe

Modèle géométrique

Coût : 48 milliards d'euros

Efficacité globale: 0,76

Robustesse : 0,962

Coût : 27 milliards d'euros

Efficacité globale: 0,79

Robustesse : 0,949

Coût : 58 milliards d'euros

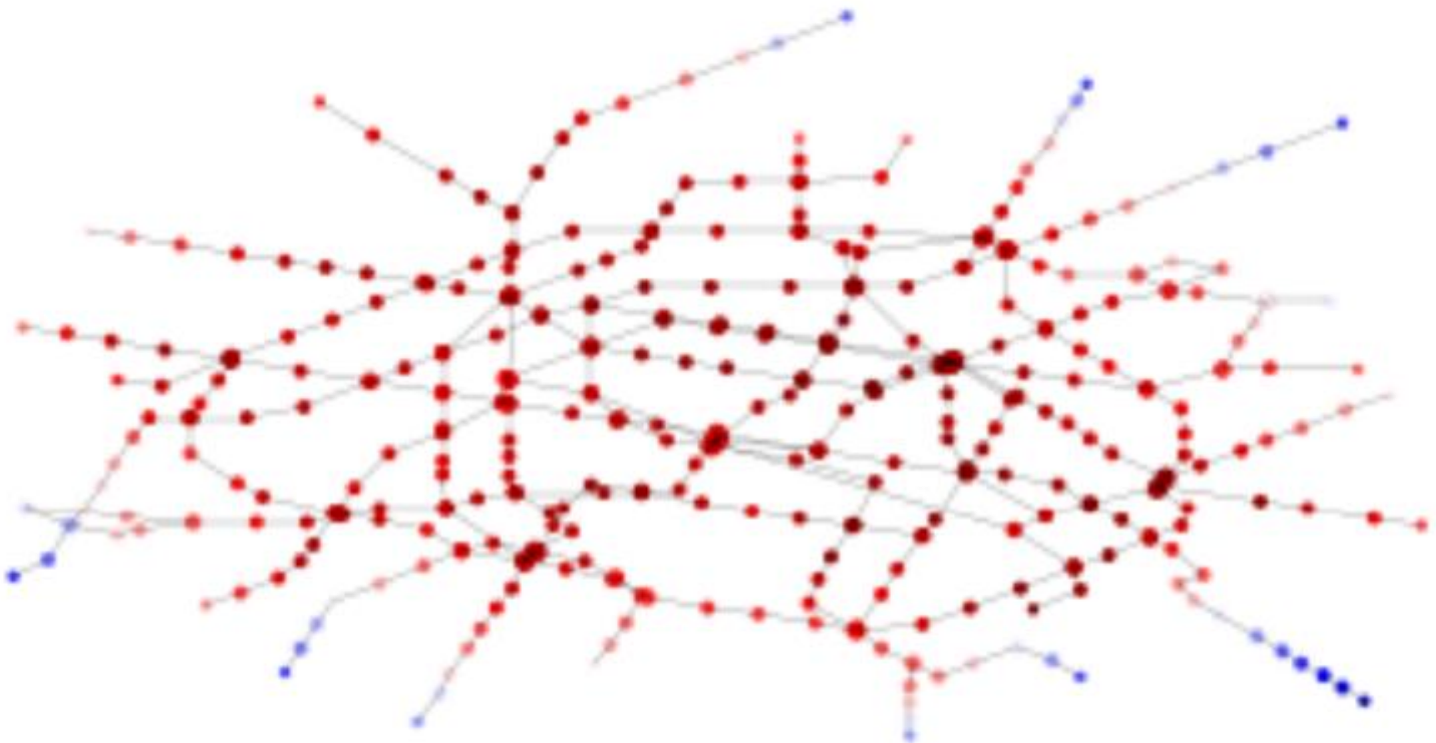
Efficacité globale: 0,92

Robustesse : 0,965

CONCLUSION

- Avantages et inconvénients de chaque modèle
- Limites du développement du métro avec ce modèle

ANNEXES



ANNEXES

```python
def métro_aléa(gares, dist):
    global nbg
    nbg = len(gares)
    adj = adj_vide(gares)
    while connexe(adj):
        a = rd.randint(0,nbg-1)
        b = rd.randint(0,nbg-1)
        while (b==a or adj[a][b]== 1):
            a = rd.randint(0,nbg-1)
            b = rd.randint(0,nbg-1)
        mod_arete(adj,a,b)
        mod_arete(adj,b,a)
    afficher(gares,adj)
    #amélio_s(gares,adj,dist,50000)
    #amélio_c(gares,adj,dist,150)
    stats(adj,dist)
```

```python
def métro_bino(gares, dist, p):
    global nbg
    nbg = len(gares)
    adj = adj_bino(gares, p)
    while connexe(adj):
        adj = adj_bino(gares,p)
    afficher(gares, adj)
    #afficher2(gares, adj)
    stats(adj,dist)
```

```python
def adj_bino(gares, p):
    nbg = len(gares)
    adjacence = np.zeros((nbg,nbg))
    for i in range (nbg):
        for j in range (nbg):
            if i != j :
                r = rd.random()
                if r < p:
                    adjacence [i,j] = 1
                    adjacence [j,i] = 1
    return adjacence
```

ANNEXES

```python
def métro_semi(gares, dist, k):
    global nbg
    nbg = len(gares)
    adj = adj_vide(gares)
    m = ppe(dist)
    while connexe(adj):
        for i in range (nbg):
            for j in range (nbg):
                if i != j :
                    d = (m**k/(dist[i][j]**k))
                    r = rd.random()
                    if r < d:
                        if adj[i][j]==0:
                            mod_arete(adj,i,j)
                            mod_arete(adj,j,i)
    afficher(gares, adj)
    #afficher2(gares, adj)
    stats(adj,dist)
```

```python
def métro_ptit(gares, dist, k, p):
    global nbg
    nbg = len(gares)
    adj = adj_vide(gares)
    for i in range(nbg):
        l = kmin(list(dist[i]), k)
        for j in range(k):
            if adj[i][l[j]] == 0:
                mod_arete(adj,i,l[j])
                mod_arete(adj,l[j],i)
    are = aretes(adj)
    for k in range(len(are)):
        a = are[k][0]
        b = are[k][1]
        c = rd.randint(0,nbg-1)
        r = rd.random()
        if r < p:
            while [a,c] in are:
                c = rd.randint(0,nbg-1)
            adj[a][b]=0
            adj[b][a]=0
            adj[a][c]=1
            adj[c][a]=1
    afficher(gares, adj)
    #afficher2(gares, adj)
    stats(adj,dist)
```

ANNEXES

```python
def métro_géo(gares, dist, r):
    global nbg
    nbg = len(gares)
    adj = adj_vide(gares)
    m = ppe(dist)
    for i in range (nbg):
        for j in range (nbg):
            if dist[i][j]>0 and dist[i][j]< m*r:
                    if adj[i][j]==0:
                        mod_arete(adj,i,j)
                        mod_arete(adj,j,i)
    afficher(gares, adj)
    #afficher2(gares, adj)
    stats(adj,dist)
```

ANNEXES

```python
def effi_glob(adj, dist):
    global nbg
    nbg = len(adj)
    adp = adj_comp(adj)
    f = effi_norm(adp, dist)
    e = effi_norm(adj, dist)
    return e/f

def effi_norm(adj, dist):
    traj = trajet(adj, dist)
    e = 0
    for i in range(nbg):
        for j in range(i+1,nbg):
            if traj[i][j] > 0:
                e = e + 1/traj[i][j]
    eff = e / (nbg*(nbg-1))
    return eff
```

```python
def dijkstra(adj, dist, s):
    P = [s]
    Q = []
    a = nbg-1
    for k in range(nbg):
        Q.append(a)
        a = a-1
    Q.remove(s)
    d = [np.inf]*nbg
    d[s] = 0
    for k in retire(voisins(s, adj), P):
        d[k] = dist[s][k]
    while Q != [] :
        i = mini(d,Q)
        Q.remove(i)
        P.append(i)
        for j in retire(voisins(i, adj),P):
            if d[j] > d[i] + dist[i][j]:
                d[j] = d[i] + dist[i][j]
    return d
```

```python
def trajet(adj, dist):
    global nbg
    nbg = len(adj)
    traj = []
    for k in range(nbg):
        traj.append(dijkstra(adj,dist,k))
    return traj
```

```python
def robust_aléa(adj, dist, tours, t, e):
    m = 0
    for k in range(tours):
        ade = enlevare(adj, dist, t)
        f = effi_norm(ade,dist)
        m=m+f/e
    m = m/tours
    return m
```