TP note

December 10, 2022

EL AMRANI Wadie
BAHOU Yassine
MOUKADDIME Nouhaila

1 Introduction

La borne de FORD-FULKERSON peut être améliorée si l'on implémente le calcul du chemin améliorant p via une recherche en largeur, c'est-à-dire si le chemin améliorant est un plus court chemin de s vers t dans le réseau résiduel, où chaque arc possède une distance (pondération) unitaire. Cette implémentation particulière de la méthode de Ford-Fulkerson a pour nom algorithme d'Edmonds-Karp. L'objectif de ce Tp est l'implementation de cet algorithme en langage python

2 Choix de la structure du graphe

Nous avons utilise une structure de dictionnaire, c'est l'une des facons les plus simples a gerer. voici un exemple de modelisation du graphe G donne en figure 1. Par le dictionnaire D , en figure 2 .Les sommets sont les clefs du dictionnaire, la valeur de la clef est la liste des voisins du sommet.

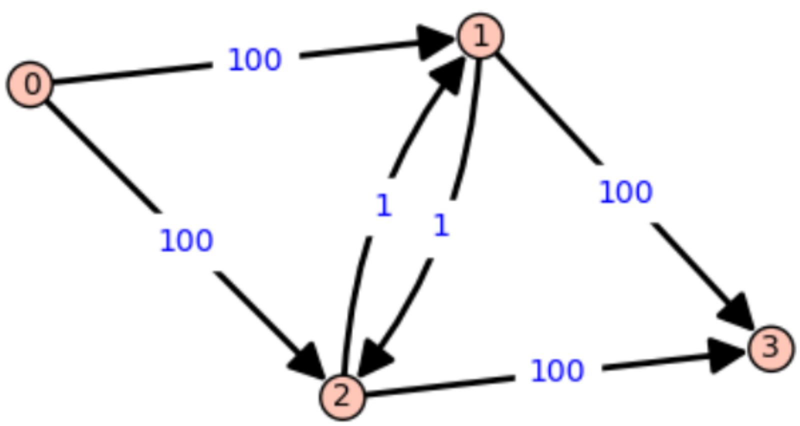


Figure 1: Graphe G

In [32]:
```
G={  0: [[1,100],[2,100]]              ,
     u: [[v1,c(u,v1)],[v2,c(u,v2)]]    ,
     2: [[1,1],[3,100]]                ,
     3: []
    }
```

Figure 2: Dictionnaire D

2.1 parcours des sommets

In [32]:
```
G={  0: [[1,100],[2,100]] ,
     1: [[3,100],[2,1]]   ,
     2: [[1,1],[3,100]]   ,
     3: []
    }
```

In [34]:
```
L=[]
for i in G:
    L.append(i)
L
```

Out[34]:
```
[0, 1, 2, 3]
```

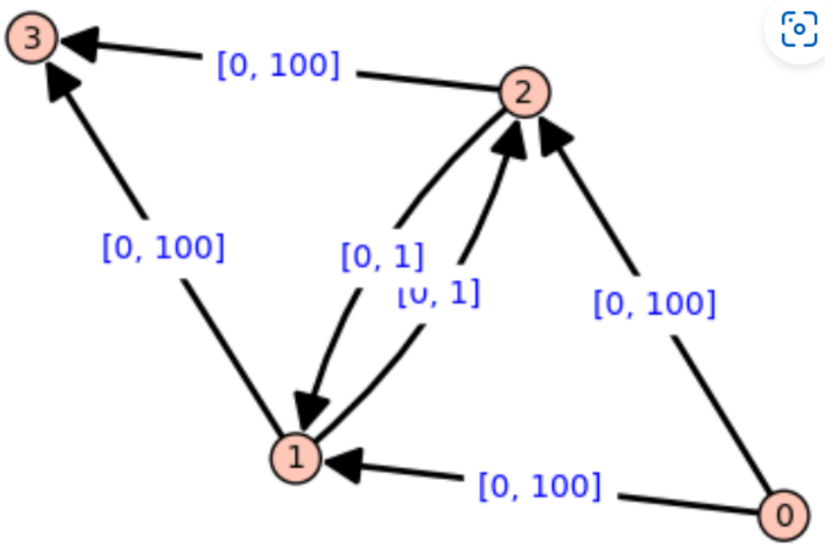
2.2 parcours des successeurs d'un sommet s

In [32]:
```
G={  0: [[1,100],[2,100]] ,
     1: [[3,100],[2,1]]   ,
     2: [[1,1],[3,100]]   ,
     3: []
    }
```

In [35]:
```
L=[]
s=0
for l in G[s]:
    L.append(l)
L
```

Out[35]:
```
[[1, 100], [2, 100]]
```

3 Structure de manipulation du graphe residuel




```
Out[38]: {0: [[1, [0, 100]], [2, [0, 100]]],
          1: [[3, [0, 100]], [2, [0, 1]]],
          2: [[1, [0, 1]], [3, [0, 100]]],
          3: []}
```

Figure 3: Graphe du reseau de flot nul Gf

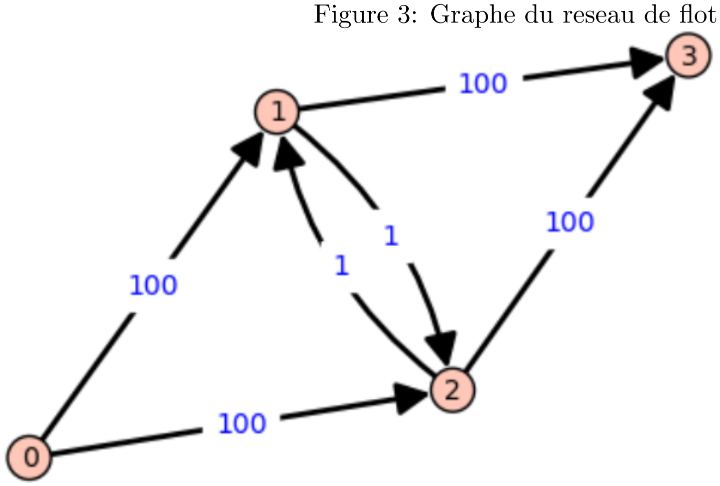


Figure 4: Graphe residuel associe a Gf

Intuitivement, étant donné un réseau de transport et un flot, le réseau résiduel est constitué des arcs qui peuvent supporter un flot plus important. Plus formellement, supposons qu'on ait un réseau de flux $G = (S, A)$ de source s et de puits t. Soit f un flot de$G$ et considérons un couple de sommets $u, v \in S$. La quantité de flux supplémentaire qu'il est possible d'ajouter entre u et v sans dépasser la capacité$c(u, v)$ est la capacité résiduelle de (u, v), donnée par $c_f(u, v) = c(u, v) - f(u, v)$.

3.1 Fonction initialiser flot

Cette fonction change la structure du graphe du depart en mettant a la place de sa capacite c, la liste [f,c] avec f est le flot sur l'arrete.

```
def initialiser_flot(D):
    D1= copy.deepcopy(D)
    for x in D1:
        for l in D1[x]:
            l[1]=[0,l[1]]
    return D1
```

3.2 Fonction graphe residuel

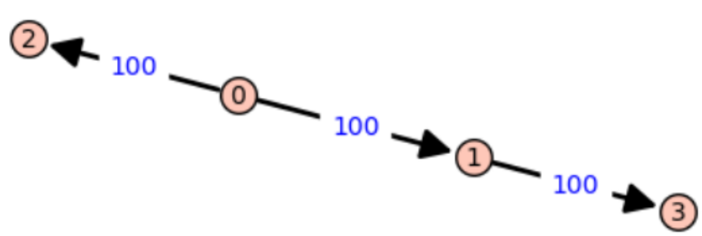
```
def graphe_residuel(D_flot):
    D_graphe_residuel={}
    flot=True
    #verification de contrainte de capacite
    for x in D_flot:
        for l in D_flot[x]:
            if l[1][0]>l[1][1]:
                flot=False
                print(f'le flot sur larrete {(x,l[0])} est plus grand que sa capacite')
    if flot:
        for x in D_flot:
            D_graphe_residuel[x]=[]
        for x in D_flot:
            for l in D_flot[x]:
                reste=l[1][1]-l[1][0]
                if l[1][0]!=0:
                    D_graphe_residuel[l[0]].append([x,l[1][0]])
                if reste >0:
                    D_graphe_residuel[x].append([l[0],reste])

    return D_graphe_residuel
```

3.3 Parcours en largeur du graphe residuel

Le parcours en largeur se fait a partir du sommet s,s'il existe un chemin vers t, le flot n'est pas maximal,et donc, on memorise ce chemin et on lui rajoute la valeur du flot minimum.

```
def BFS(D,s):
    l=[]
    p=[]
    c=[]
    for v in range(len(D)):
        l.append(9999)
        p.append(None)
        c.append(0)
    l[s]=0
    L=[]
    L.append(s)
    while (len(L)>0):
        v=L[0]
        L.remove(v)
        for k in D[v]:
                j=k[0]
                if c[j]==0:
                    L.append(j)
                    c[j]=1
                    p[j]=v
                    l[j]=l[v]+1
        c[v]=2
    D_BFS={}
    for i in range(len(p)):
        D_BFS[i]=[]
    for i in range(len(p)):
        if p[i]!=None:
            for v in D[p[i]]:
                if v[0]==i:
                    D_BFS[p[i]].append([i,D[p[i]][D[p[i]].index(v)][1]])
    return (p,D_BFS)
```

```
BFS(exemple,0)
g_bfs=construire_graphe(BFS(exemple,0)[1])
g_bfs.show(edge_labels=True)
```



4 Algorithme d'Edmond-karp

```python
def Edmond_karp1(D,s,t):
# Initialisation
    D_flot=initialiser_flot(D)
    D_gf=graphe_residuel(D_flot)
    p=BFS(D_gf,s)[0]
    D_gf_bfs=BFS(D_gf,s)[1]
    L_chemin=[]
    chemin(D_gf_bfs,p,s,t,L_chemin)
    while(len(L_chemin)!=0):
        L=[]
# Calcul du flot minimum sur le chemin stocke dans la liste L_chemin
        for x in L_chemin:
            for v in D_gf_bfs[x[0]]:
                if v[0]==x[1]:
                    L.append(v[1])
                if(len(L)!=0):
                    min_capacite=min(L)
# Amelioration de la valeur du flot
        for x in L_chemin:
            for v in D_flot[x[0]]:
                if v[0]==x[1]:
                     v[1][0]=v[1][0]+min_capacite
# Recherche d'un chemin dans le graphe residuel du reseau de flot ameliore
        D_gf=graphe_residuel(D_flot)
        p=BFS(D_gf,s)[0]
        D_gf_bfs=BFS(D_gf,s)[1]
        L_chemin=[]
        chemin(D_gf_bfs,p,s,t,L_chemin)
# Valeur du flot max
    flot_max=0
    for x in D_flot[s]:
        flot_max=flot_max+x[1][0]
    print(f'la valeur du flot maximum est {flot_max}')
    return (L,D_flot,flot_max)
```

5 Applications

5.1 Exemple donne en TP

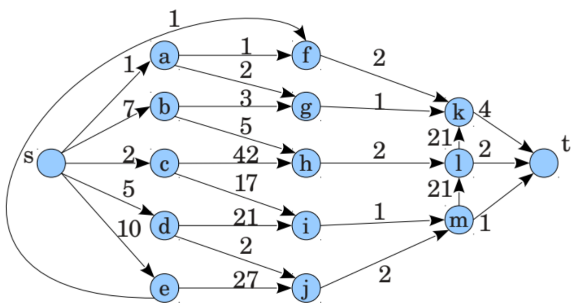


FIGURE 1 – Exemple de flot

```
D1={0:[[1,1],[2,7],[3,2],[4,5],[5,10]],
    1:[[6,1],[7,2]],
    2:[[7,3],[8,5]],
    3:[[8,42],[9,17]],
    4:[[9,21],[10,2]],
    5:[[10,27]],
    6:[[11,2]],
    7:[[11,1]],
    8:[[12,2]],
    9:[[12,1]],
    10:[[13,2]],
    11:[[14,4]],
    12:[[14,2],[11,21]],
    13:[[12,21],[14,1]],
    14:[]
    }
```

Figure 5: Structure dictionnaire

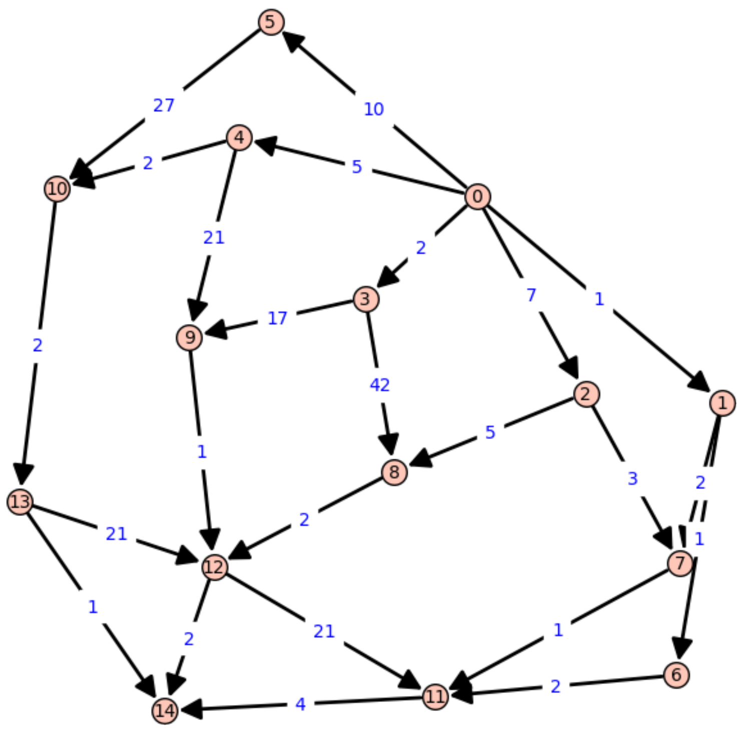


Figure 6: presentation par sagemath

5.2 Trace de l'execution de l'algorithme

Les figures suivantes representent les differentes etapes de l'algorithme.

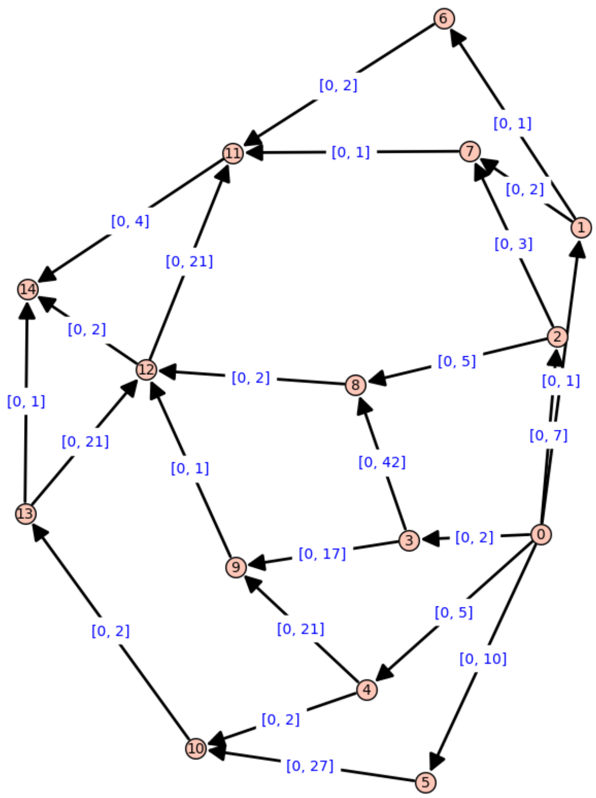


Figure 7: Initialisation du flot

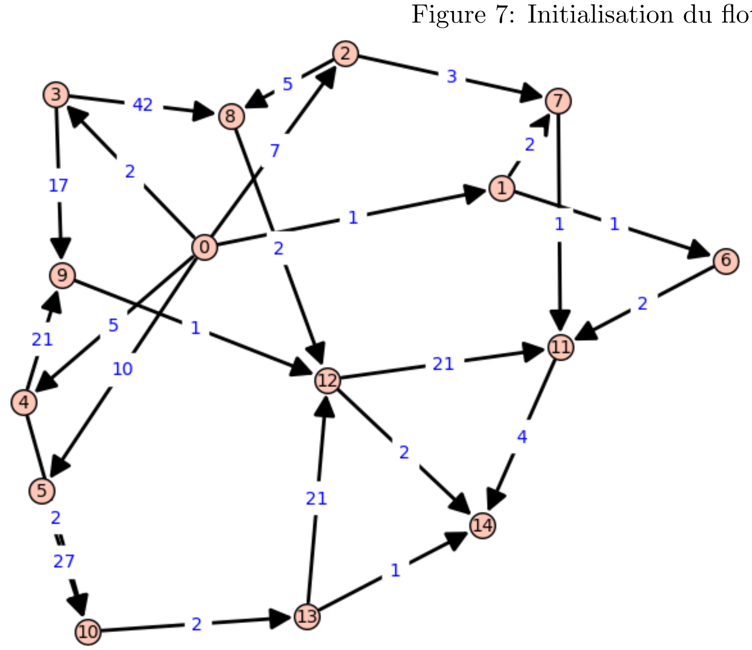


Figure 8: premier graphe residuel

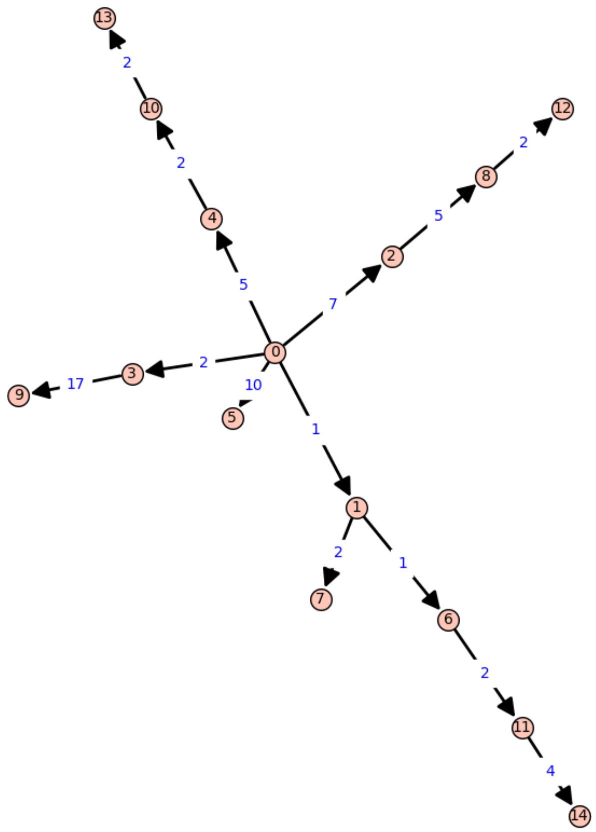


Figure 9: BFS

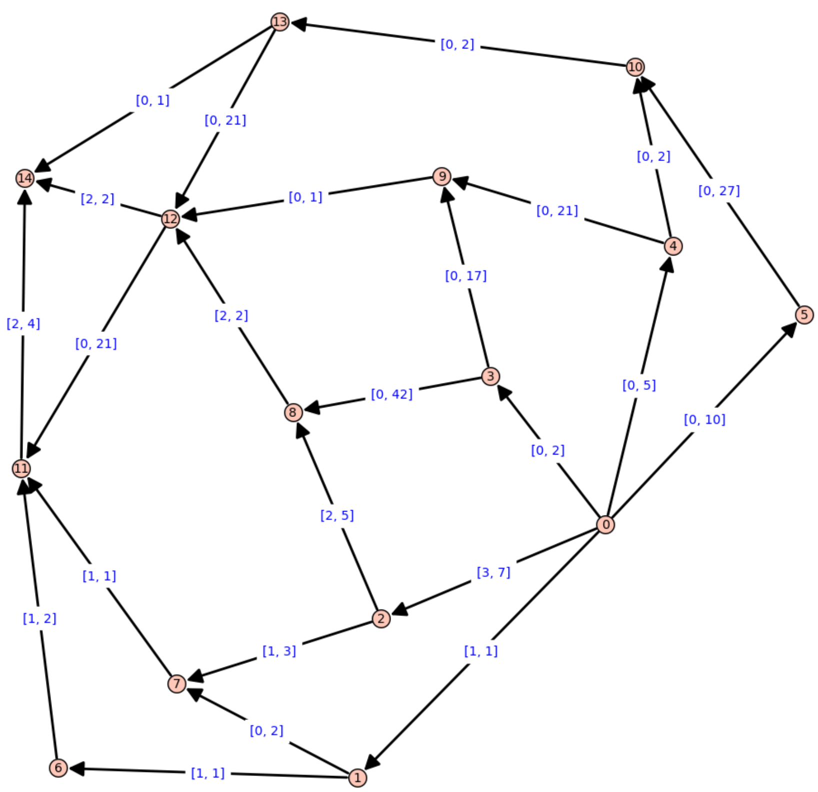


Figure 10: Troisieme iteration

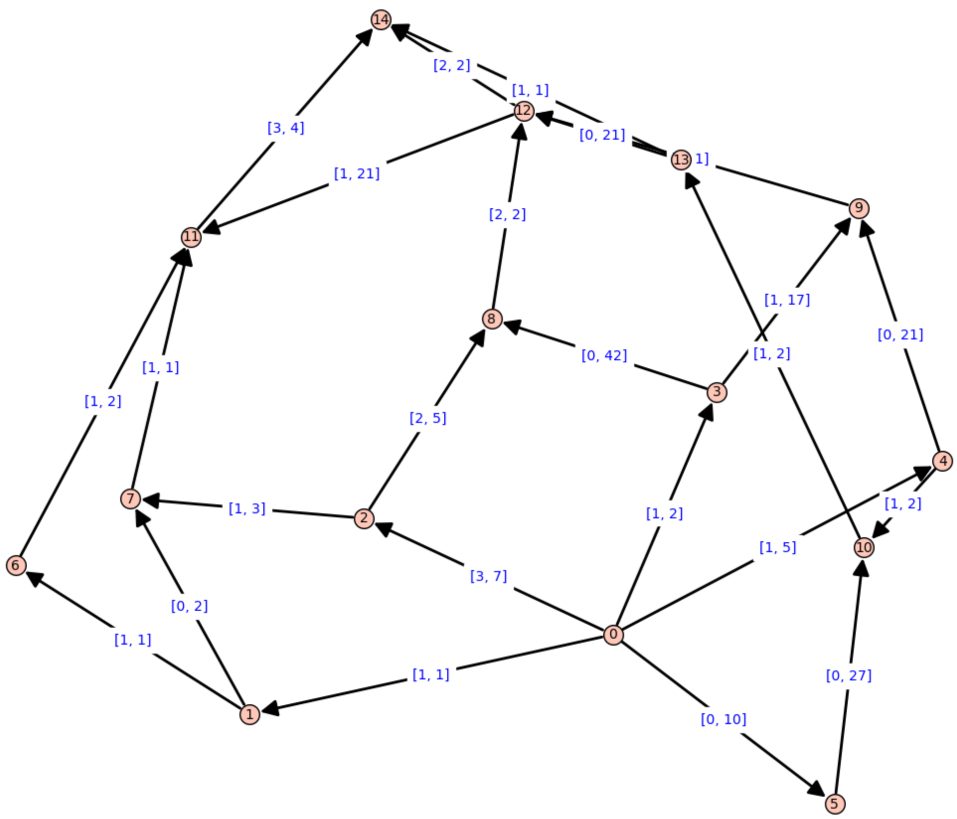


Figure 11: Avant derniere etape

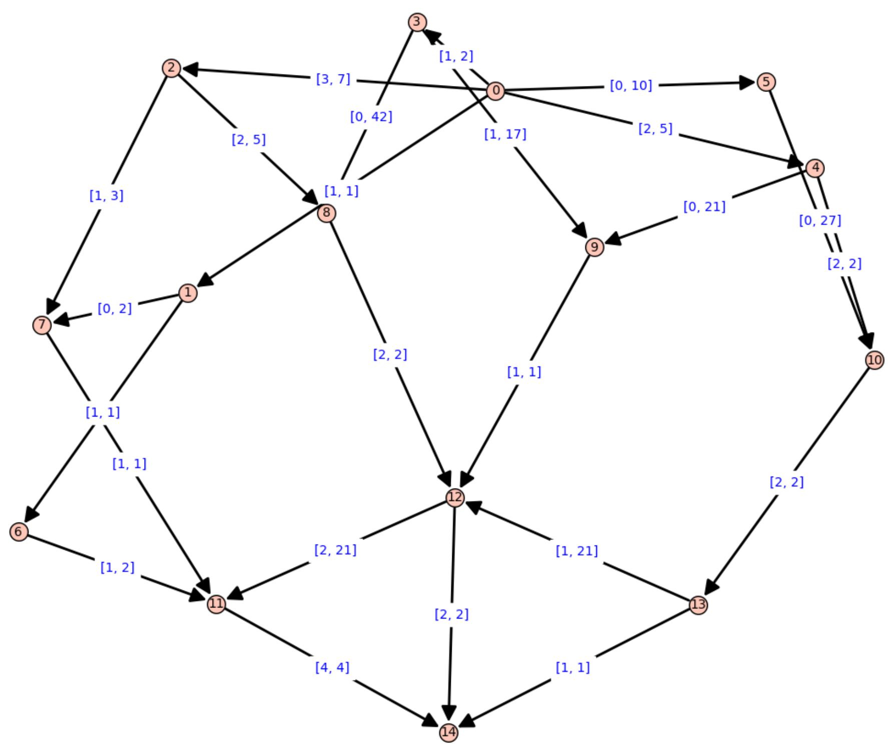


Figure 12: Derniere amelioration

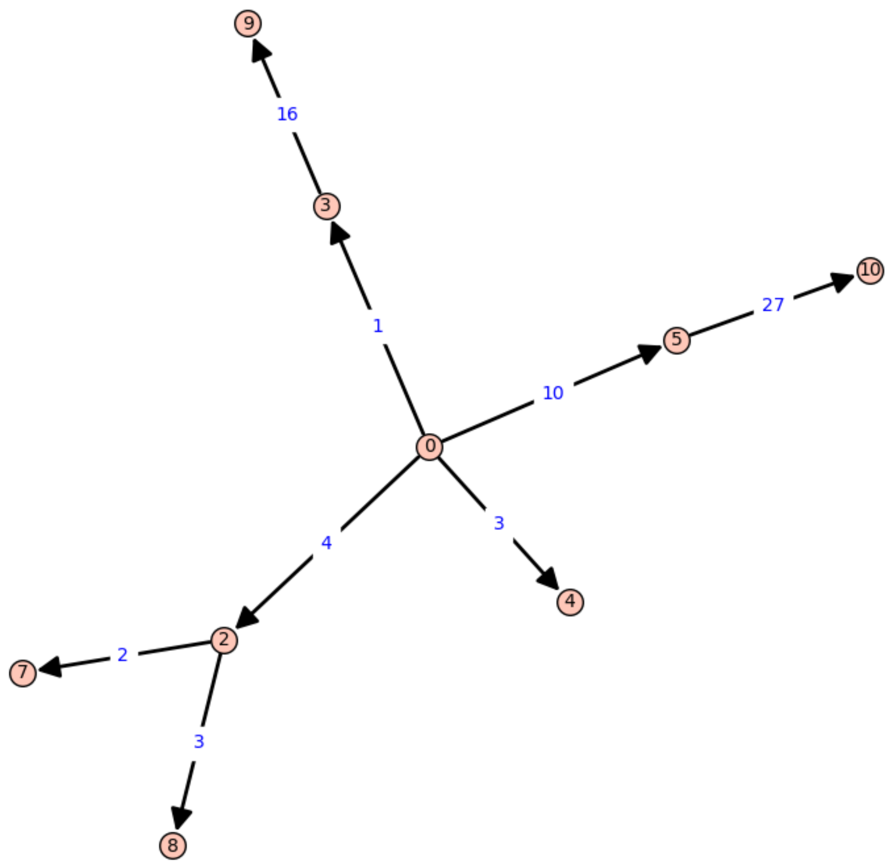


Figure 13: il n'existe aucun chemin de 0 a 14, l'algorithme s'arrete en 6 iteration

6 Analyse de complexite de l'algorithme

```python
def random_graph(number_vertices,number_edges,min_capacite,max_capacite):
    D={}
    # definition du dictionnaire
    for x in range(number_vertices):
        D[x]=[]
    arrete=0
    #creation d'arrete aleatoire entre sommets
    while(arrete<number_edges):
        sommet_depart=random.randint(0, number_vertices-1)
        sommet_arrive=random.randint(0, number_vertices-1)
        if sommet_depart!=sommet_arrive:
            if [sommet_arrive] not in D[sommet_depart]:
                D[sommet_depart].append([sommet_arrive])
                arrete=arrete+1
    # definition d'une source et puit aleatoire
    s=random.randint(0, number_vertices-1)
    t=random.randint(0, number_vertices-1)
    # aucune arrete n'est incidente a s, et aucune arrete n'est sortante de t
    for x in D:
        for arrete in D[x]:
            if arrete[0]==s:
                D[x].remove(arrete)
    for arrete in D[t]:
        D[t].remove(arrete)
    # poids aleatoires sur les arretes crees
    for x in D:
        for arrete in D[x]:
            poids=random.randint(min_capacite,max_capacite)
            arrete.append(poids)
    return (D,s,t)
```

```python
def nuage_temps_execution(nb_sommet,nb_min_arrete,nb_max_arrete,pas,min_capacite,max_capacite):
    liste_test=[]
    for j in range(nb_min_arrete,nb_max_arrete,pas):
            graphe=random_graph(nb_sommet,j,min_capacite,max_capacite)[0]
            s=random_graph(nb_sommet,j,min_capacite,max_capacite)[1]
            t=random_graph(nb_sommet,j,min_capacite,max_capacite)[2]
            start_time = time.perf_counter()
            Edmond_karp1(graphe,s,t)
            end_time = time.perf_counter()
            total_time = end_time - start_time
            liste_test.append([i,j,total_time])
    fig = plt.figure()
    ax1 = fig.add_subplot(111)
    sommets=[x[0] for x in liste_test]
    arrete=[x[1] for x in liste_test]
    temps_execution=[x[2] for x in liste_test]
    ax1.scatter(arrete,temps_execution, s=10, c='r')
    plt.show()
```

L’axe des abscisse represente le nombre d’arrete du graphe, l’axe des ordonnes represente le temps d’execution en seconde

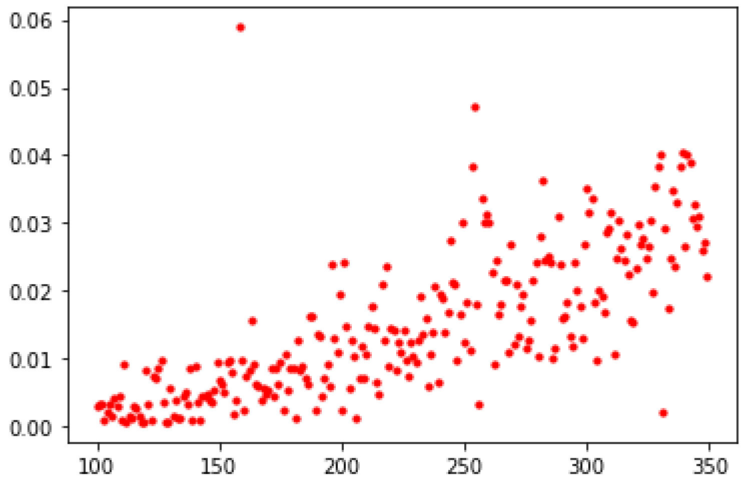


Figure 14: Graphe de 30 sommets

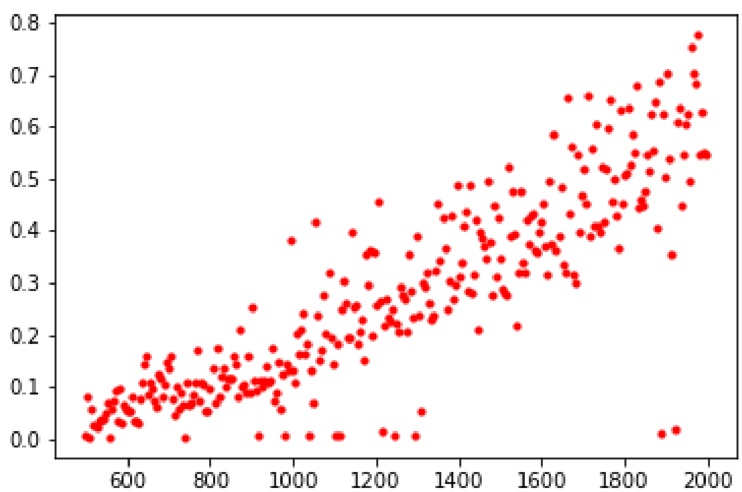


Figure 15: Graphe de 50 sommets

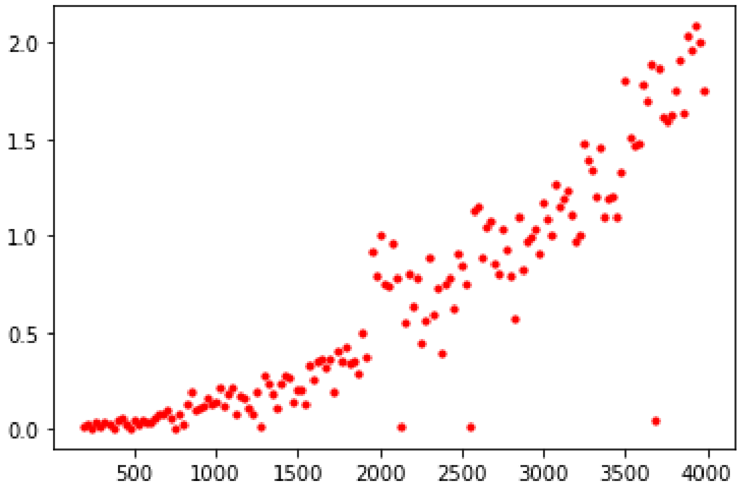


Figure 16: Graphe de 75 sommets

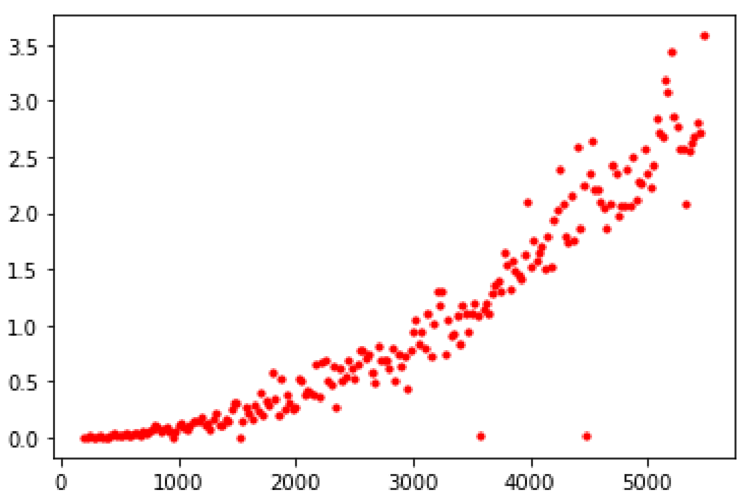


Figure 17: Graphe de 80 sommets