Tipe 2018-2019
Thème: transport
Numéro d’isncription:39888

Filière MP
option: SI

Optimisation du transport de l'eau

Objectif: Améliorer le fonctionnement du système de distribution d’eau

Comment optimiser le fonctionnement du réseau de distribution d’eau ?

- Etude physique permettant de déterminer les différents paramètres du réseau.
- Modélisation d’un réseau de transport de l’eau potable par la théorie des graphes.
- Maximisation du flot à l’aide de l’algorithme de Ford Fulkerson et implémentation avec un code python.
- Modélisation du même réseau à l’aide de la programmation linéaire.
- Etude mathématique de l’optimisation avec l’algèbre linéaire et la topologie.
- Application de ces approches sur des vraies données.

Contributions

Septembre 2018 :visite du barrage l'Aroussia
Prise de contact avec le professeur de physique Mr. Edward Tantart du lycée Saint-Louis.
Prise de contact avec le professeur de mathématiques Mr. Claude DesChamps
Implémentation de l'algorithme Dijikstra et Ford Fulkerson
Modélisation avec la programmation linéaire

PLAN

I. Introduction

II. Modélisation physique:

1-Equation de Bernoulli .

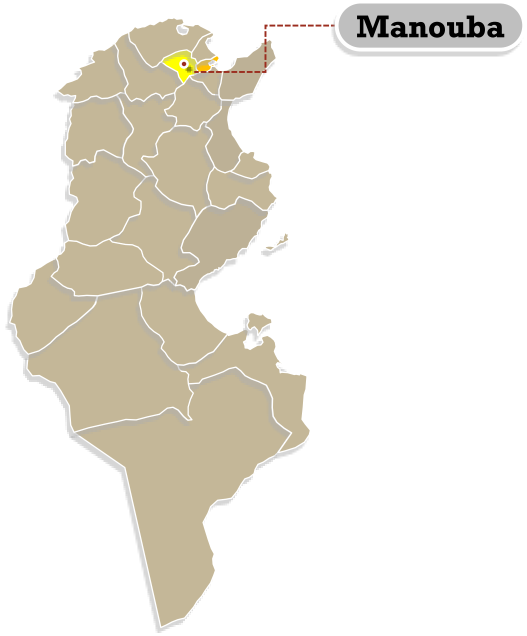
2-Résolution de l’équation du perte de charge

III - Modélisation mathématique

1. Modélisation du réseau des conduites par un graphe pondéré.
2. Explication de quelques algorithmes et les exécutés

MOTIVATIONS

- La Tunisie est un pays aride qui vit un stress hydrique à cause du manque de précipitations

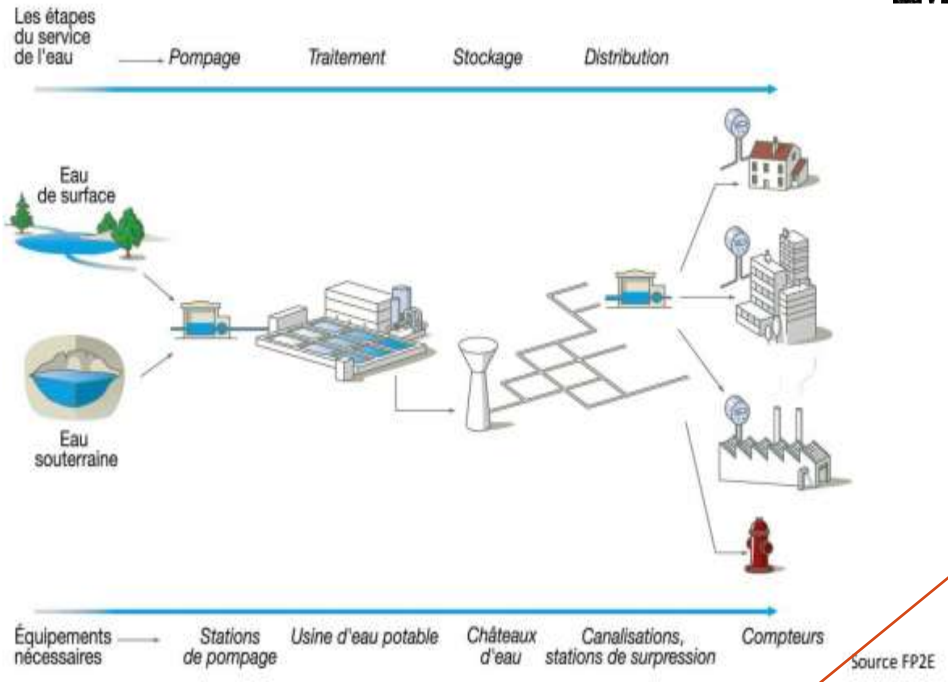


Tunisie

Solution

- Transport de l’eau à des grandes distances

MODÉLISATION



Latitude1

Longitude1

Source

Ville

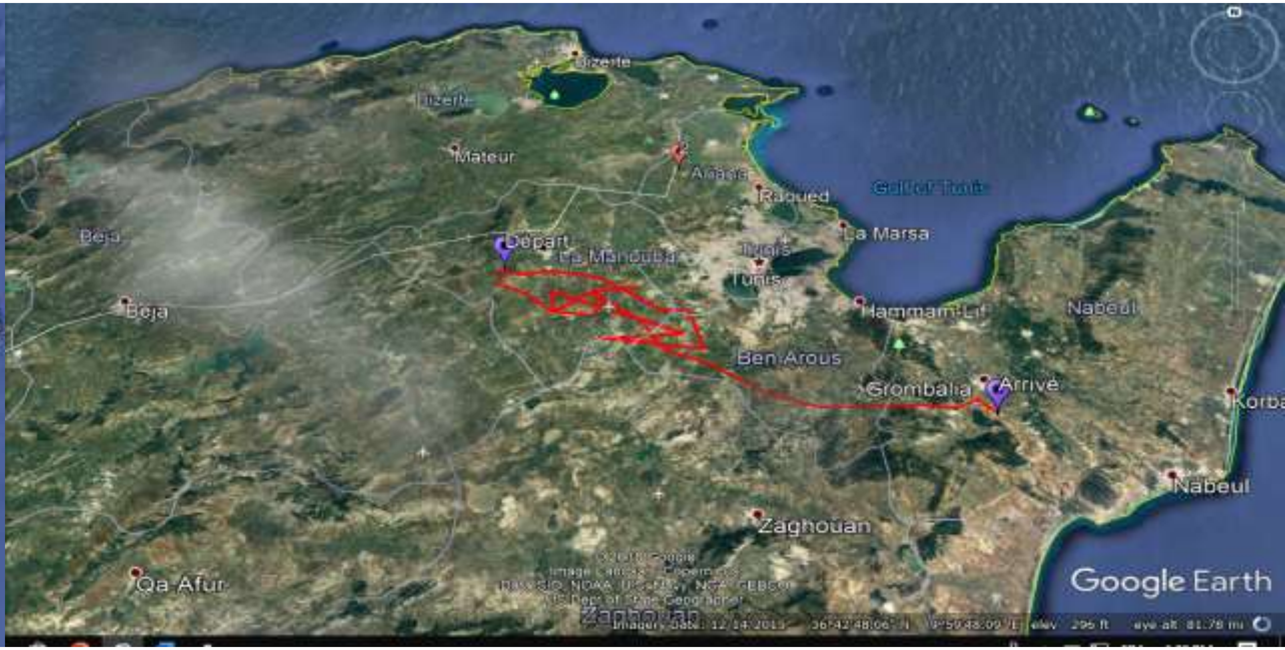
Latitude2

Longitude2

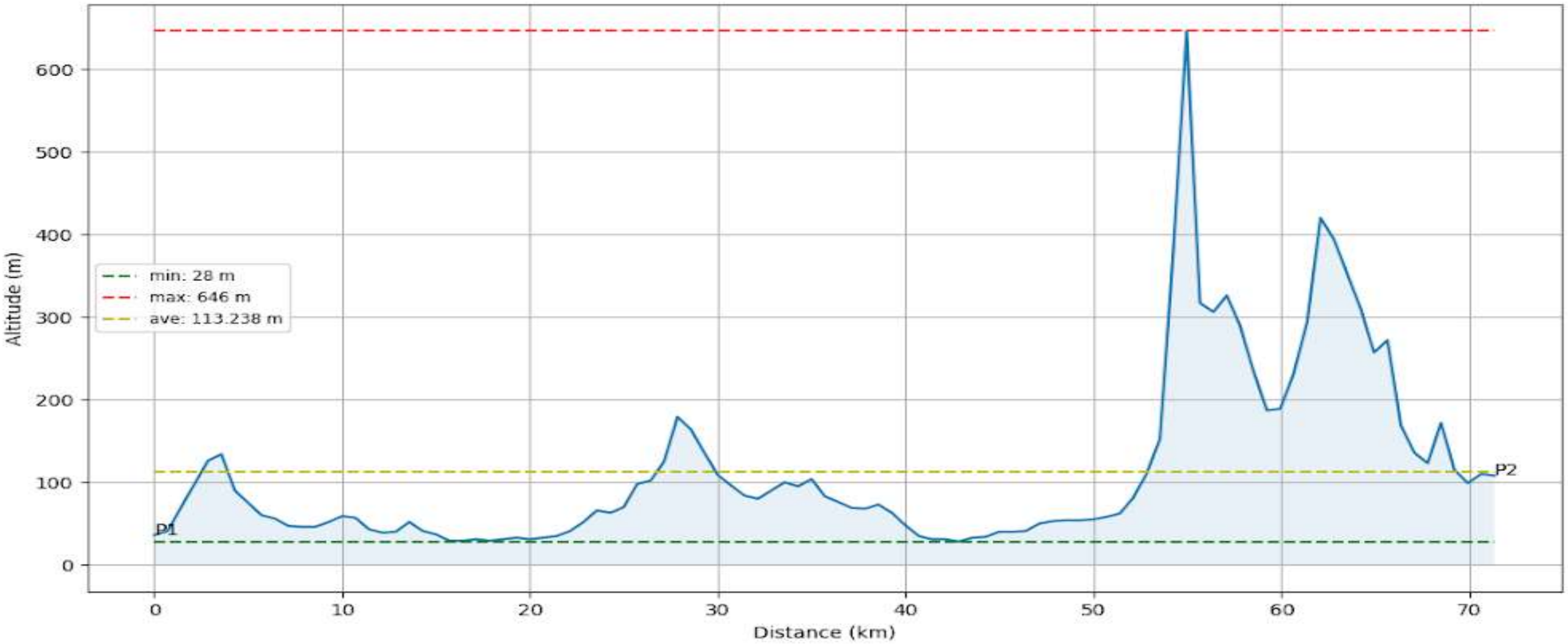
Distance

Comment peut on étudier le comportement du conduite ?

- Réalisation d’une étude longitudinale tout au long la conduite



Coupe topographique



Méthodologie

- La modélisation du transport de l'eau est complexe (topologie non plane: altitude variable, grandes distances, réseau hétérogène …)

- Nécessite d'une approche multidisciplinaire:

- - Utilisation de la mécanique des fluides pour comprendre le comportement de l'eau pendant son transport (débit, perte de charge, besoin…)

- -Utilisation des Mathématiques pour modéliser le réseau des conduites (mathématiques discrètes , la théorie des graphes) et optimiser le flot (programmation linéaire)

THÉORÈME DE BERNOULLI

Approximations:

- Un fluide incompressible
- Un fluide parfait
- En régime stationnaire
- On néglige les transferts d'énergie sous forme de chaleur

$$P_1 + \rho g z_1 + \rho \frac{v_1^2}{2} = p_2 + \rho g z_2 + \rho \frac{v_2^2}{2} + \Delta p_f$$

$$h_1 + z_1 + \frac{v_1^2}{2g} = h_2 + z_2 + \frac{v_2^2}{2g} + \Delta h_f$$

PERTE DE CHARGE :

$$\Delta p = \lambda \frac{L}{D}\frac{\rho V^2}{2}$$

On a

$$P_1 + \rho g z_1 + \rho \frac{V_1^2}{2} = p_2 + \rho g z_2 + \rho \frac{V_2^2}{2} + \Delta p_f$$

$$\begin{cases} Z_1 = Z_2 \\ V_1 = V_2 \end{cases}$$

$$P_1 = P_2 + \Delta P_f$$

$$P_1 = P_2 + \Delta P$$

Avec $\Delta P = \Delta P_f$

$$L = \frac{2(P_1 - P_2)D}{\lambda \rho V^2}$$

Détermination des positions des stations de pompage

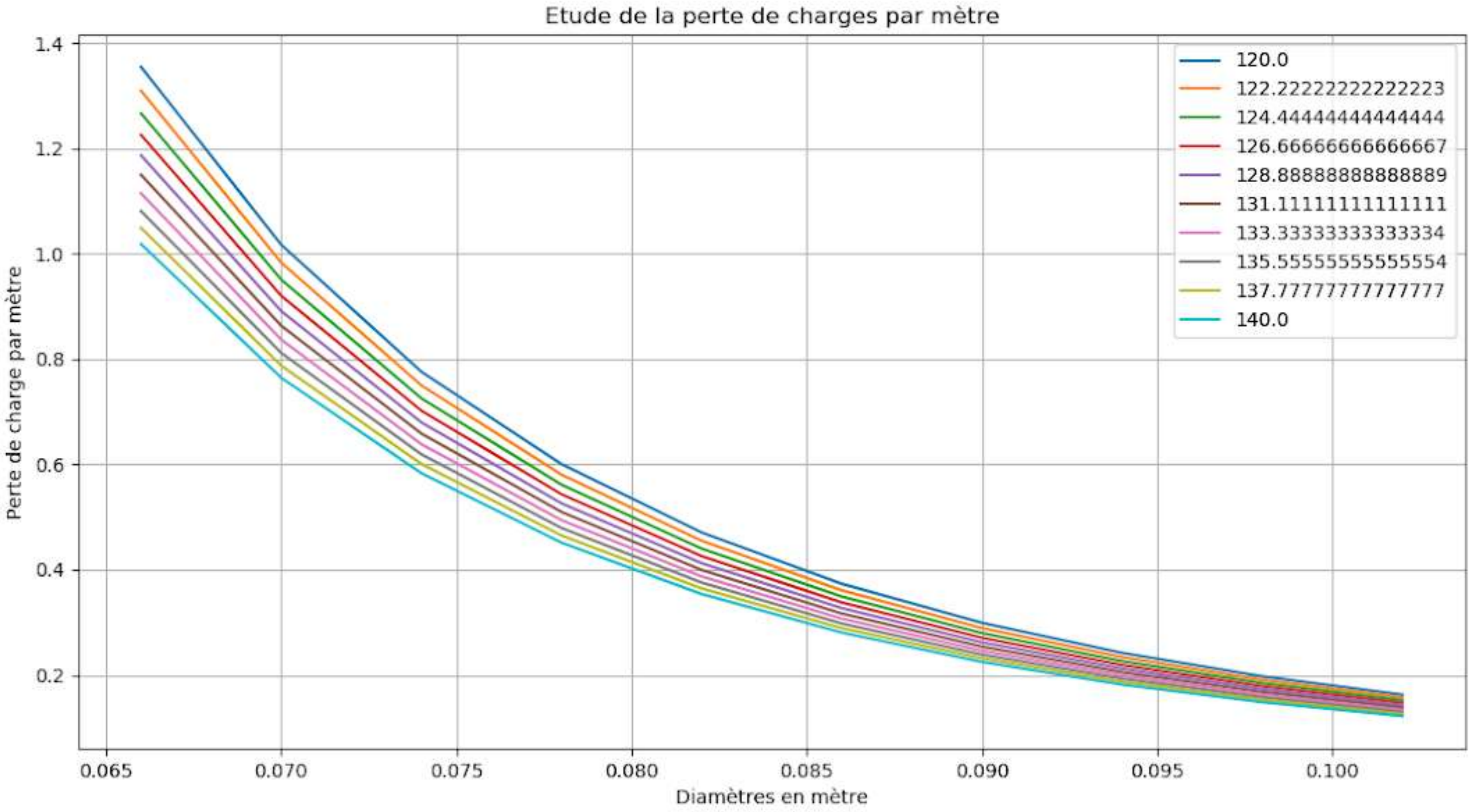
Une méthode plus simple

- Afin de calculer la perte de charge il est plus simple d'utiliser les loi empiriques en effet les ingénieurs préfèrent utiliser la formule de Williams-Hazen

$$j = 10{,}675 * \left(\frac{Q}{C}\right)^{1{,}852} * \frac{1}{D^{4{,}872}}$$

Nature du tuyau	C
PVC	150
PE	145
Acier revêtu	130-150
Fonte revêtue	135-150
Aluminium	120
Fonte encrassée	80-120

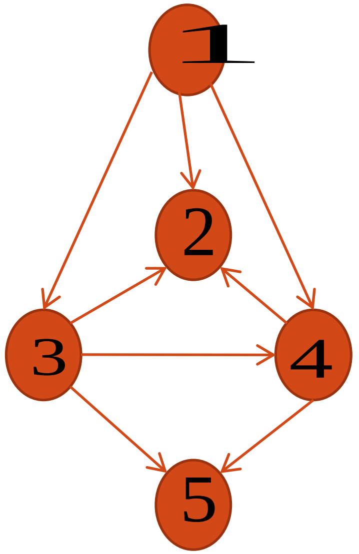
j: Perte de charge linéaire par unité de longueur en m/m
Q: Débit d'écoulement en m3/h ;
C: Coefficient de rugosité dépendant de la nature de la conduite
D: Diamètre intérieur de la conduite en mm ;


Etude de la perte de charges par mètre
Perte de charge par mètre
Diamètres en mètre
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.065
0.070
0.075
0.080
0.085
0.090
0.095
0.100
120.0
122.2222222222223
124.4444444444444
126.6666666666667
128.8888888888889
131.1111111111111
133.3333333333334
135.5555555555554
137.7777777777777
140.0

Comment peut-on modéliser ce problème ?

- Solution: on modélise ce problème par la théorie des graphes
- Les graphes font partie des mathématiques discrètes il est constitué de deux ensembles (ensemble de sommets, ensemble d'arêtes) avec une pondération .

on peut modéliser une carte géographique par une matrice d'adjacence.



$$
\begin{array}{c c}
 & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{pmatrix}
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$

On peut trouver le plus court chemin à travers l'algorithme de Dijkstra réalisé par le mathématicien et l'informaticien néerlandais Edsger Dijkstra.

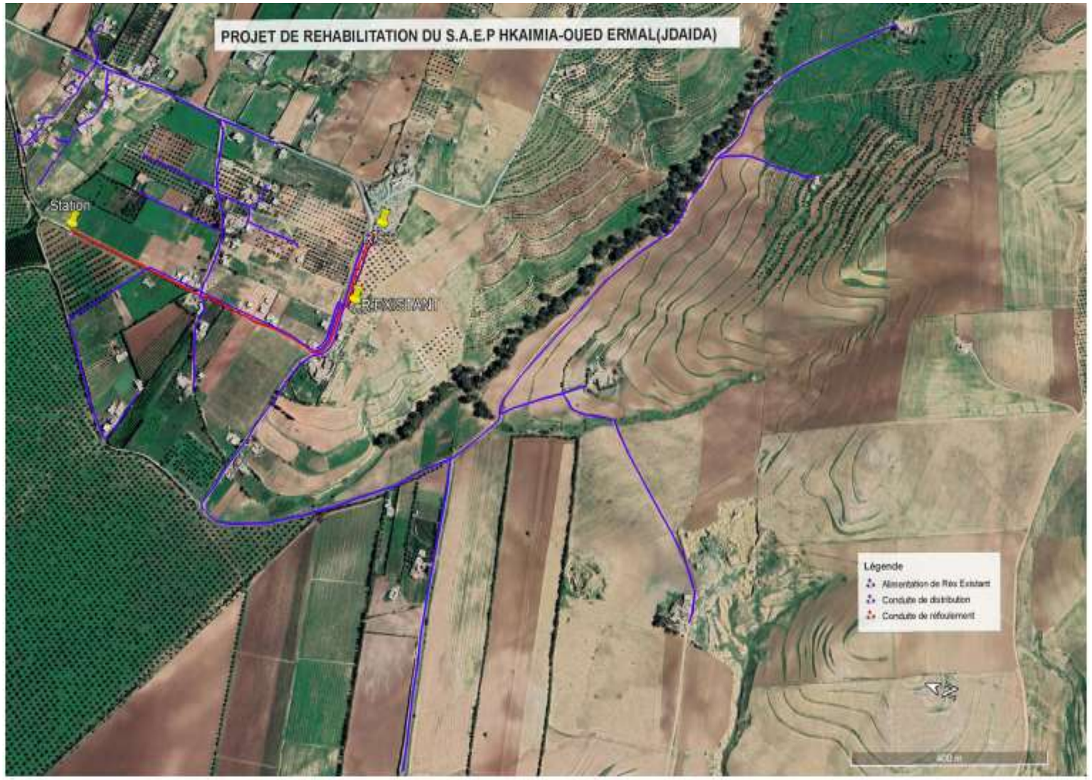
- G=(S,A) un graphe avec une pondération positive poids des arcs, s un sommet de S.
- P:=[]
- d[a]:=infinity pour chaque sommet a
- d[s]=0

INITIALISATIONS

- Tant que tout les sommets ne sont pas dans P
-     choisir un sommet a n'est pas dans P de plus petite distance d[a] ,mettre a dans P
-     pour chaque sommet b hors de P voisin de a
-         d[a]=min(d[b],d[a] + poids (a,b))
-     fin pour
- Fin tant que

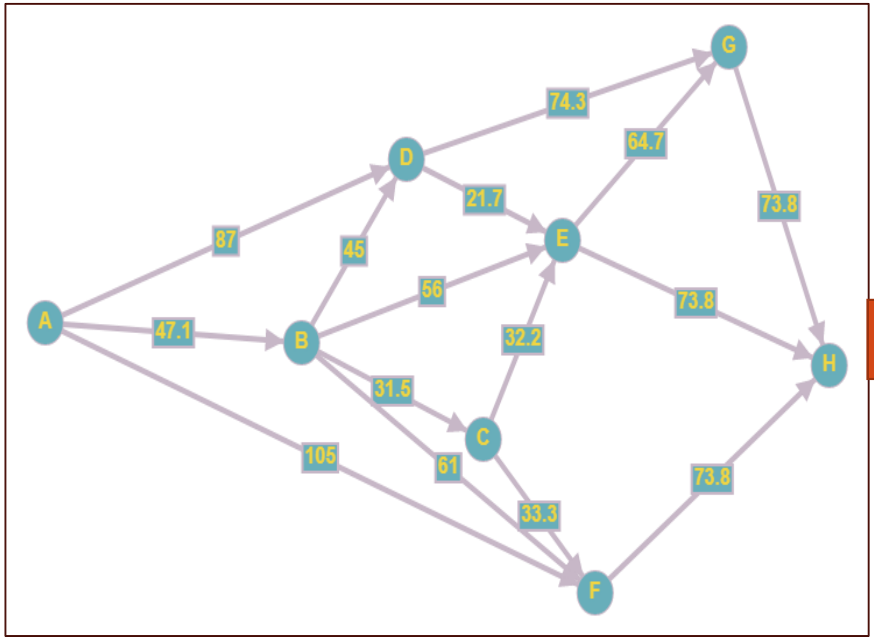
Boucle de traitements

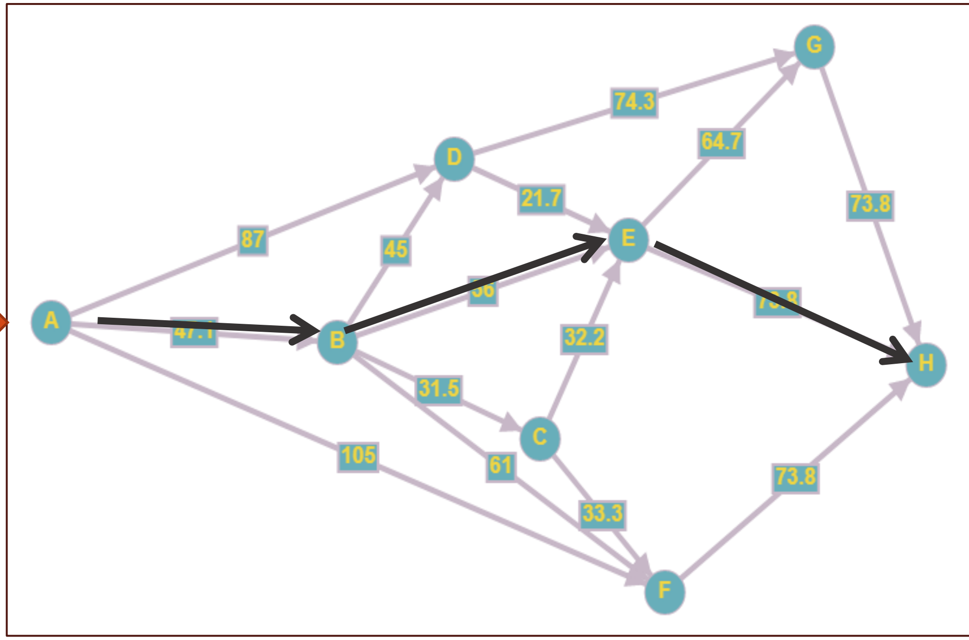
comment ça fonctionne l'algorithme de Dijkstra ?



L’algorithme de Dijikstra permet de trouver le plus court chemin entre 2 points mais dans notre cas on veut déterminer qui maximise le flot de l’eau transporté

Application d'algorithme de Dijikstra





CALCUL DE COMPLEXITÉ

- La complexité, ou le coût, d'un algorithme ou d'une fonction Python est le nombre d'opérations élémentaires nécessaires à son exécution dans le pire cas.

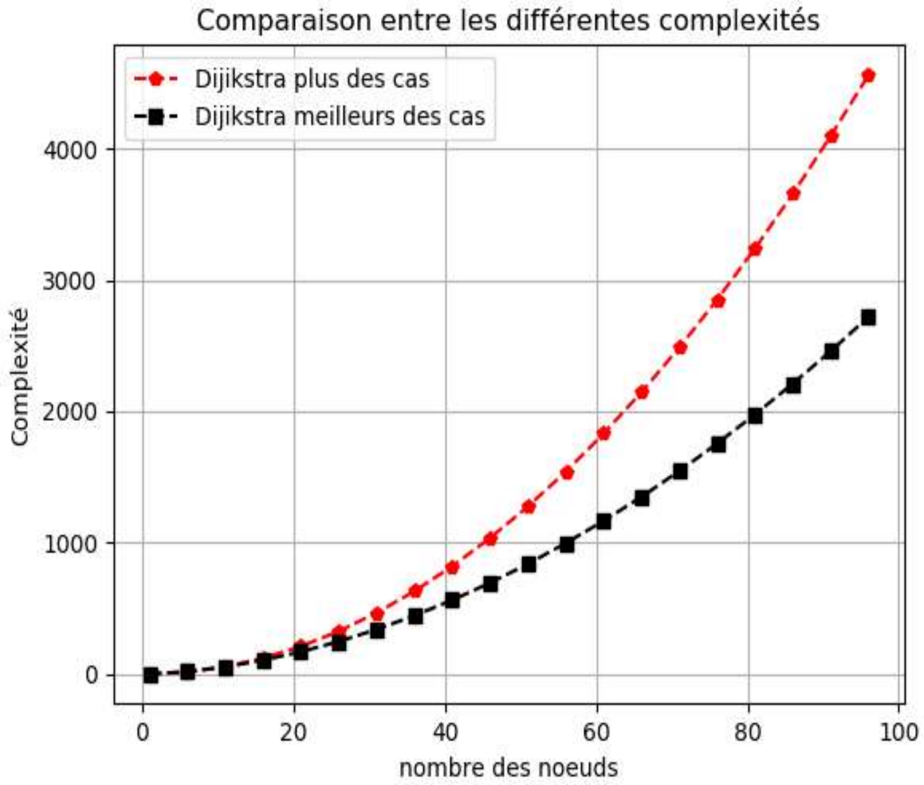
complexité temporelle

une fonction de n qui
mesure le temps de calcul
pour une donnée de taille n

O(m + n log(n))

n le nombre de nœuds

m le nombre d'arcs

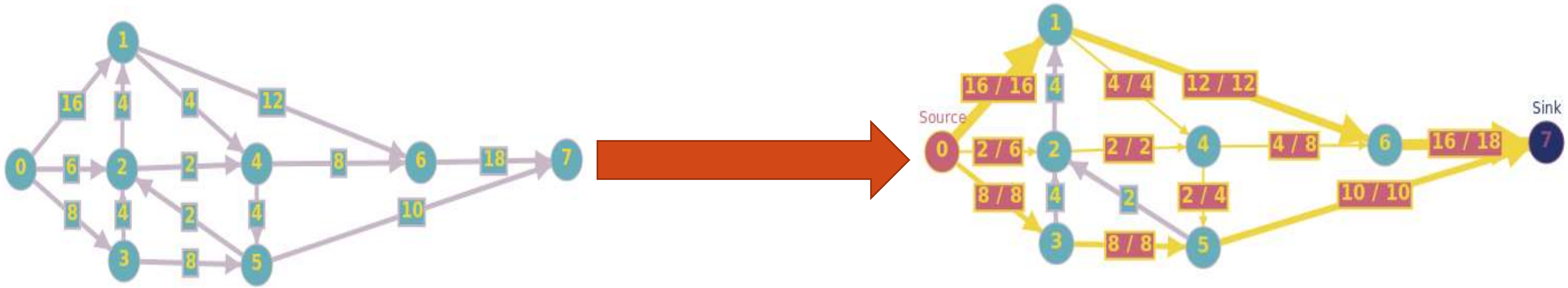


Mais , est ce que trouver le plus court chemin maximise le flot ce qui résulte une production maximale ?

Problème du flot maximal sur un réseau:
Soit un réseau (un graphe pondéré et orienté dont les poids sont des capacités), le problème du flot maximal consiste à déterminer un flot sur ce réseau, compatible avec les capacités, tel que la valeur du flot circulant entre la source et le puits soit maximale.

Principe de l'algorithme de Ford-Fulkerson
-On part d'un flot f compatible avec les capacités.
- On cherche une chaîne reliant s et p pour laquelle le flot f peut être amélioré (ou augmenté).
- Si une telle chaîne n'existe pas, le flot f est maximal et le problème est résolu.
- Si une telle chaîne existe, on augmente au maximum le flot f sur cette chaîne améliorante.
- On réitère la recherche d'une chaîne améliorante autant de fois que possible.
- Lorsqu'il n'existe plus de chaîne améliorante, le dernier flot calculé est maximal.

Application de l'algorithme Ford Fulkerson



L'algorithme de Ford Fulkerson

Données : un réseau G=(X,A,C)

Résultat : φ

= 0 — Initialisation

Répéter

chercher une chaine améliorante de s à t

si existe alors

calculer

augmenter φ sur les arcs de μ+ de δ

diminuer φ sur les arcs de μ- de δ

Jusqu'à il n'existe plus de chaine améliorante

Retourner φ

Boucle de traitements

Conclusion

- Le problème de minimisation des coûts du transport de l'eau a été étudié.
- L'optimisation du transport de l'eau est complexe nécessite une approche fondée sur : la mécanique des fluides, l'algorithmique, simulation .
- L'algorithme de Dijkstra donne le plus court chemin plus nécessairement optimal d'où utilisation de l'algorithme de Ford Fulkerson.

ANNEXE

1. Calcul distance

```
from math import sin, cos, sqrt, atan2, radians


def distance(q,s,d,f):
    R = 6373.0
    lat1 = radians(q)
    lon1 = radians(s)
    lat2 = radians(d)
    lon2 = radians(f)

    dlon = lon2 - lon1
    dlat = lat2 - lat1

    a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
    c = 2 * atan2(sqrt(a), sqrt(1 - a))

    distance = R * c

    return(distance)
```

2. Réalisation d'une coupe longitudinale

```python
import urllib.request
import json
import math
import matplotlib.pyplot as plt



#START-END POINT
def conversion_degre(d,min,sec):

    return d + (min/60) + (sec/3600)

latitude1 = 36.799616666666665
longitude1 = 9.773213888888889
latitude2 = 36.550769444444444
longitude2 = 10.510544444444445

P1=[latitude1,longitude1]
P2=[latitude2,longitude2]

#NUMBER OF POINTS
s=100
interval_lat=(P2[0]-P1[0])/s #interval for latitude
interval_lon=(P2[1]-P1[1])/s #interval for longitude

#SET A NEW VARIABLE FOR START POINT
lat0=P1[0]
lon0=P1[1]

#LATITUDE AND LONGITUDE LIST
lat_list=[lat0]
lon_list=[lon0]
```

```python
#LATITUDE AND LONGITUDE LIST
lat_list=[lat0]
lon_list=[lon0]

#GENERATING POINTS
for i in range(s):
    lat_step=lat0+interval_lat
    lon_step=lon0+interval_lon
    lon0=lon_step
    lat0=lat_step
    lat_list.append(lat_step)
    lon_list.append(lon_step)

#HAVERSINE FUNCTION
def haversine(lat1,lon1,lat2,lon2):
    lat1_rad=math.radians(lat1)
    lat2_rad=math.radians(lat2)
    lon1_rad=math.radians(lon1)
    lon2_rad=math.radians(lon2)
    delta_lat=lat2_rad-lat1_rad
    delta_lon=lon2_rad-lon1_rad
    a=math.sqrt((math.sin(delta_lat/2))**2+math.cos(lat1_rad)*math.cos(lat2_rad)
    d=2*6371000*math.asin(a)
    return d

#DISTANCE CALCULATION
d_list=[]
for j in range(len(lat_list)):
    lat_p=lat_list[j]
    lon_p=lon_list[j]
    dp=haversine(lat0,lon0,lat_p,lon_p)/1000 #km
    d_list.append(dp)
d_list_rev=d_list[::-1] #reverse list
```

```python
#CONSTRUCT JSON
d_ar=[{}]*len(lat_list)
for i in range(len(lat_list)):
    d_ar[i]={"latitude":lat_list[i],"longitude":lon_list[i]}
location={"locations":d_ar}
json_data=json.dumps(location,skipkeys=int).encode('utf8')

#SEND REQUEST
url="https://api.open-elevation.com/api/v1/lookup"
response = urllib.request.Request(url,json_data,headers={'Content-Type': 'applic
fp=urllib.request.urlopen(response)

#RESPONSE PROCESSING
res_byte=fp.read()
res_str=res_byte.decode("utf8")
js_str=json.loads(res_str)
#print(js_str)
fp.close()

#GETTING ELEVATION
response_len=len(js_str['results'])
elev_list=[]
for j in range(response_len):
    elev_list.append(js_str['results'][j]['elevation'])

#BASIC STAT INFORMATION
mean_elev=round((sum(elev_list)/len(elev_list)),3)
min_elev=min(elev_list)
max_elev=max(elev_list)
distance=d_list_rev[-1]
```

```python
#PLOT ELEVATION PROFILE
base_reg=0
plt.figure(figsize=(10,4))
plt.plot(d_list_rev,elev_list)
plt.plot([0,distance],[min_elev,min_elev],'--g',label='min: '+str(min_elev)+' m'
plt.plot([0,distance],[max_elev,max_elev],'--r',label='max: '+str(max_elev)+' m'
plt.plot([0,distance],[mean_elev,mean_elev],'--y',label='ave: '+str(mean_elev)+'
plt.fill_between(d_list_rev,elev_list,base_reg,alpha=0.1)
plt.text(d_list_rev[0],elev_list[0],"P1")
plt.text(d_list_rev[-1],elev_list[-1],"P2")
plt.xlabel("Distance (km)")
plt.ylabel("Altitude (m)")
plt.grid()
plt.legend(fontsize='small')
plt.show()
```

3. Ford.Fulkerson

```python
def ford_fulkerson(graph, source, sink, debug=None):
    flow, path = 0, True

    while path:
        # search for path with flow reserve
        path, reserve = depth_first_search(graph, source, sink)
        flow += reserve
        # increase flow along the path
        for v, u in zip(path, path[1:]):
            if graph.has_edge(v, u):
                graph[v][u]['flow'] += reserve
            else:
                graph[u][v]['flow'] -= reserve

        # show intermediate results
        if callable(debug):
            debug(graph, path, reserve, flow)

def depth_first_search(graph, source, sink):
    undirected = graph.to_undirected()
    explored = {source}
    stack = [(source, 0, undirected[source])]

    while stack:
        v, _, neighbours = stack[-1]
        if v == sink:
            break

        # search the next neighbour
        while neighbours:
            u, e = neighbours.popitem()
            if u not in explored:
                break
        else:
            stack.pop()
            continue

        # current flow and capacity
```

```python
        # current flow and capacity
        in_direction = graph.has_edge(v, u)
        capacity = e['capacity']
        flow = e['flow']
        # increase or redirect flow at the edge
        if in_direction and flow < capacity:
            stack.append((u, capacity - flow, undirected[u]))
            explored.add(u)
        elif not in_direction and flow:
            stack.append((u, flow, undirected[u]))
            explored.add(u)
    # (source, sink) path and its flow reserve
    reserve = min((f for _, f, _ in stack[1:]), default=0)
    path = [v for v, _, _ in stack]

    return path, reserve
```

```python
graph = nx.DiGraph()
graph.add_nodes_from('ABCDEFGH')
graph.add_edges_from([
    ('A', 'B', {'capacity': 4, 'flow': 0}),
    ('A', 'C', {'capacity': 5, 'flow': 0}),
    ('A', 'D', {'capacity': 7, 'flow': 0}),
    ('B', 'E', {'capacity': 7, 'flow': 0}),
    ('C', 'E', {'capacity': 6, 'flow': 0}),
    ('C', 'F', {'capacity': 4, 'flow': 0}),
    ('C', 'G', {'capacity': 1, 'flow': 0}),
    ('D', 'F', {'capacity': 8, 'flow': 0}),
    ('D', 'G', {'capacity': 1, 'flow': 0}),
    ('E', 'H', {'capacity': 7, 'flow': 0}),
    ('F', 'H', {'capacity': 6, 'flow': 0}),
    ('G', 'H', {'capacity': 4, 'flow': 0}),
```

4-Perte de charge

```
@author: Fares
"""
import numpy as np
import matplotlib.pyplot as plt
CWH = np.linspace(120,140,10)
vitesse = np.linspace(0.2,1.2,10)
annees = np.array([2019, 2024, 2029, 2034, 2039])
diametre = np.linspace(0.066,0.102,10)
besoin_fictif = 0.110

coef_pointe = 1.8
nbre_moyen_famille = 4.5
coef_augmentation = 1.2

def debit(besoin_fictif,coef_pointe,coef_augmentation,nbre_moyen_famille):
    # coefficient de pointe pour un débit
    # suffisant
    return (2.5 * besoin_fictif * coef_pointe * coef_augmentation * nbre_moyen_famille)/86.4
def perte_charge(q,c,d):
    formule = (10.674 * q**(1.852)) / (c**1.852 * d**4.871)
    return formule
# traçage perte en fonction de D

Q = debit(besoin_fictif,coef_pointe,coef_augmentation,nbre_moyen_famille)

for i in range(len(CWH)):
    perte = [perte_charge(Q,CWH[i],d) for d in diametre]
    plt.plot(diametre,perte,label=CWH[i])
    plt.xlabel("Diamètres en mètre")
    plt.ylabel("Perte de charge par mètre")
    plt.title("Etude de la perte de charges par mètre")


plt.legend()
plt.grid()
plt.show()

perte = [perte_charge(Q,CWH,diametre)]




#!/usr/bin/env python3
# -*- coding: utf
```