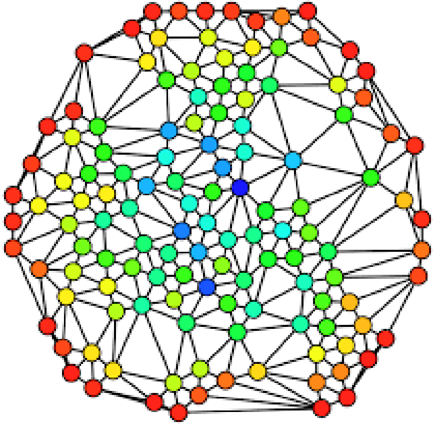
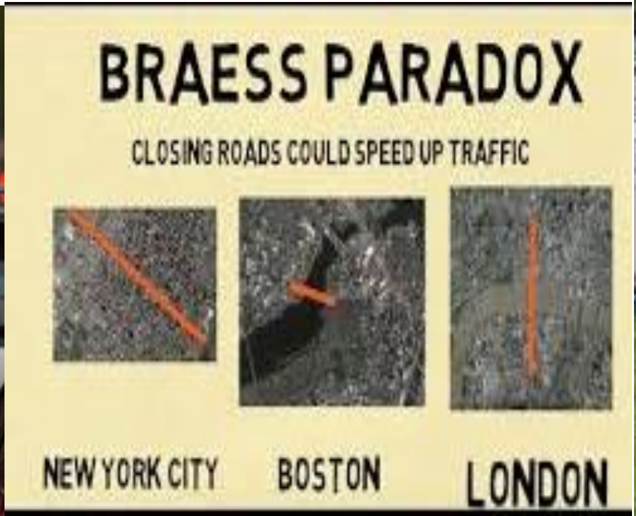
TIPE 2018/2019:

Numéro scei:36140 /MPSI



Transport routier: ralentir pour arriver plus vite !



Plan

I. *Modéliser le réseau routier à l'aide des mathématiques discrètes(la théorie des graphes).*

II. *Appliquer des algorithmes de recherche de plus court chemin (dijkstra, A*(star)) et comparaison de complexité.*

III. *Recherche d'une stratégie optimale: Paradoxe de Braess.*

IV. *Modéliser le trafic routier à l'aide d'une équation aux dérivées partielles.*

V. *Validation des différentes approches.*

Contributions:

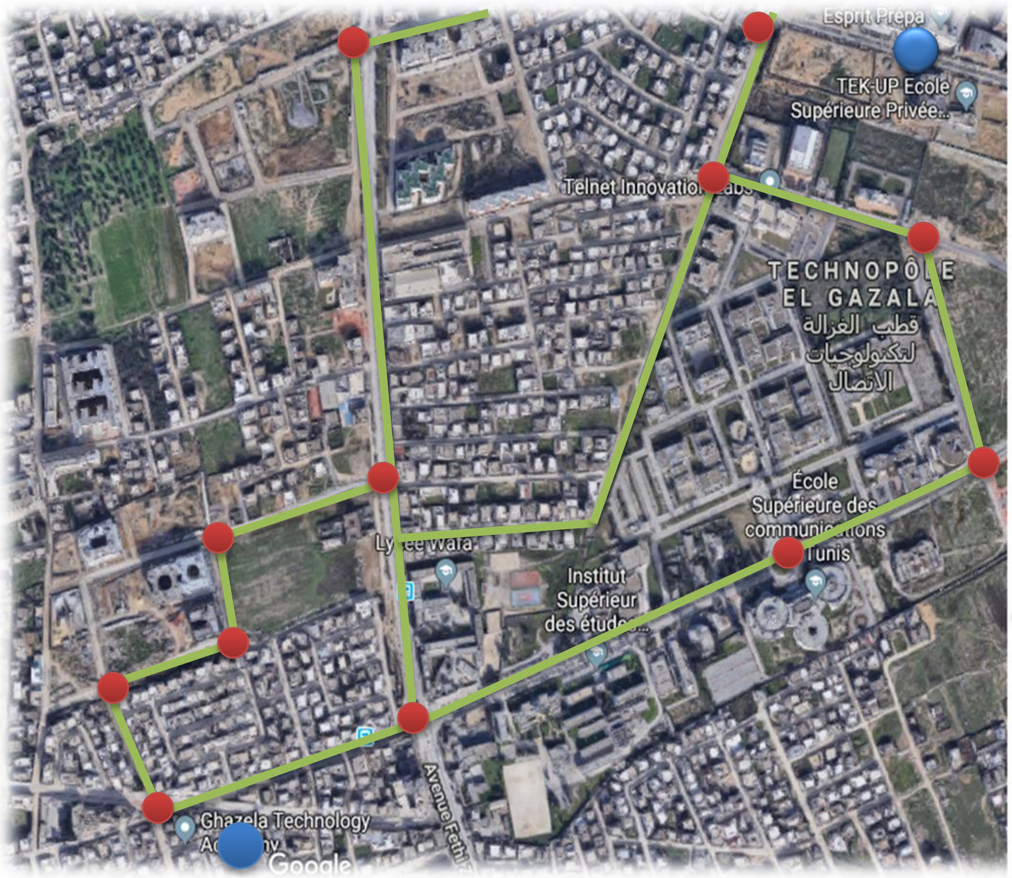
I — *Compréhension de la théorie des graphes et la modélisation de la carte de mon quartier.*

II — *Compréhension et implémentation des l'algorithmes Dijekstra et A*.*

III — *Etablir le lien entre l'emboutillage et les intéractions sociales et introduire le paradoxe de Braess.*

IV — *Etablir le lien entre l'emboutillage et la physique au moyen d'une modélisation et des transformations mathématiques(Transformé de Fourier).*

Modélisation: carte



-Utilisation d'une application (google maps).
-Mesure de la latitude et la longitude.
-Recherche des distances et l'implémentation dans un tableau(calcul à la main et utilisation de google maps).
-Calcul des durées au moyen d'une expérience:
mesure réelle à l'aide d'un chronomètre pendant toute l'année .

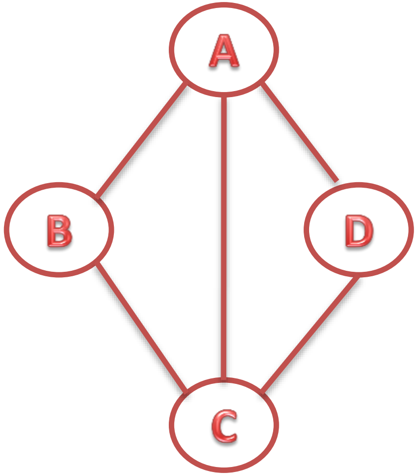
Fichier Excel

sommet départ	sommet arrivé	distance
A	B	90
B	C	400
C	D	180
B	D	400
D	E	650

Théorie des graphes

Problème: Comment modéliser une carte?

Solution: Les mathématiques discrètes: théorie des graphes.



Qu'est ce qu'un graphe?

-Un graphe est un modèle qui étudie un ensemble d'objets liés par des relations: un ensemble de sommets et d'arrêtes.

Où sont les graphes?

-Réseau: routier , aériens..

-Mathématique.

-Informatique.

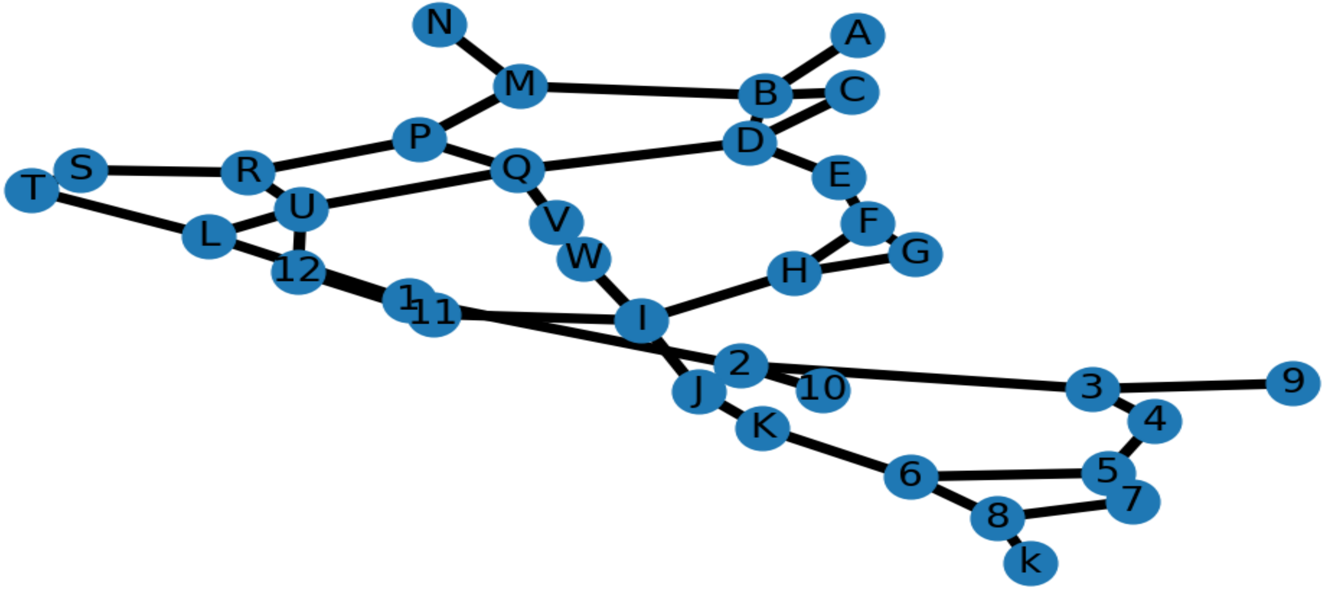
A quoi sert un graphe?

-Représenter des relations entre éléments.

Matrice d'adjacence:

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Modélisation: graphe



Comment déterminer le plus court chemin entre le point A et K ?

Application des algorithmes:

-Dijkstra:

Entrées: G=(S,A) un graphe avec une pondération positive poids des arcs, S_{deb} un sommet de S.

P:=0

d[a]:= $+\infty$

d[S_{deb}]=0

Initialisation

Tant que la liste P ne contient pas tout les sommets.

Choisir un sommet s hors de P de plus petite distance d[s]

Mettre s dans P

Pour chaque sommet a hors de P adjacent de s

d[a]=min(d[a],d[s]+poids(s, a))

Fin pour

Fin Tant Que

Traitements

Algorithme de Dijkstra:

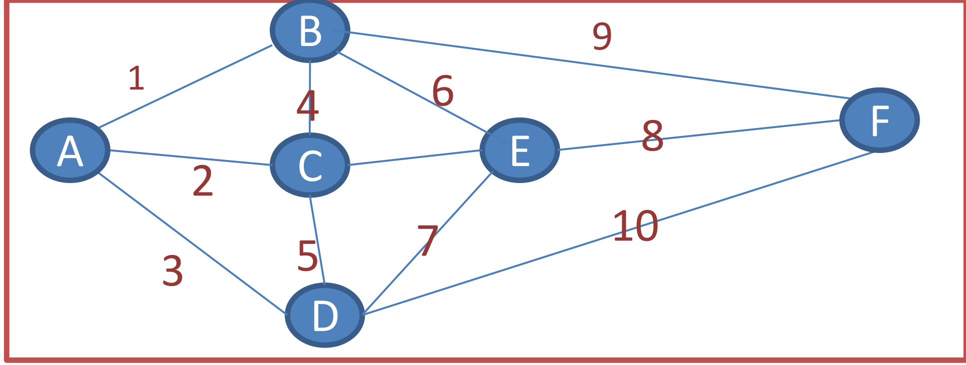
Entrées: G=(S,A) un graphe avec une pondération positive poids des arcs, S_{deb} un sommet de S.

P:=0
d[a]:=+ ∞
d[S_{deb}]=0

Initialisation

Tant que la liste P ne contient pas tout les sommets.
    Choisir un sommet s hors de P de plus petite distance d[s]
    Mettre s dans P
    Pour chaque sommet a hors de P adjacent de s
        d[a]=min(d[a],d[s]+poids(s, a))
    Fin pour
Fin Tant Que

Traitements

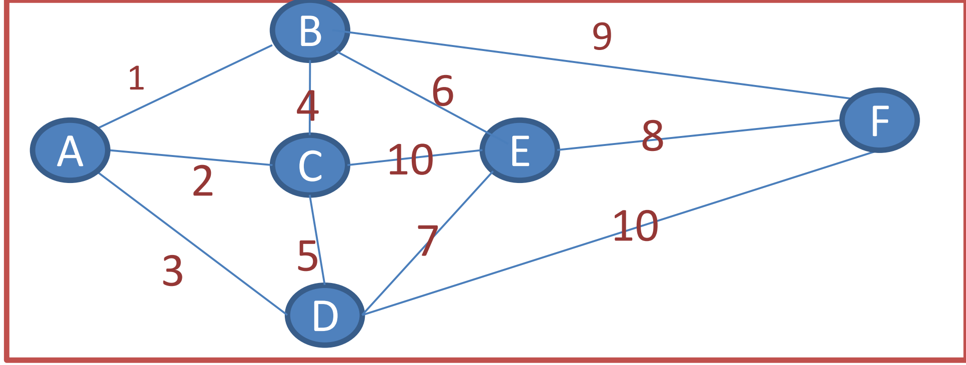


P={A}

Sommet	A	B	C	D	E	F
A	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

Algorithme de Dijkstra:

Entrées: G=(S,A) un graphe avec une pondération positive poids des arcs, S_{deb} un sommet de S.
P:=0
d[a]:=+ ∞
d[S_{deb}]=0
Initialisation
Tant que la liste P ne contient pas tout les sommets.
    Choisir un sommet s hors de P de plus petite distance d[s]
    Mettre s dans P
    Pour chaque sommet a hors de P adjacent de s
        d[a]=min(d[a],d[s]+poids(s, a))
    Fin pour
Fin Tant Que
Traitements



P={A,B}

Sommet	A	B	C	D	E	F
A	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞
B		1(A)	2(A)	3(A)	+ ∞	+ ∞

Algorithme de Dijkstra:

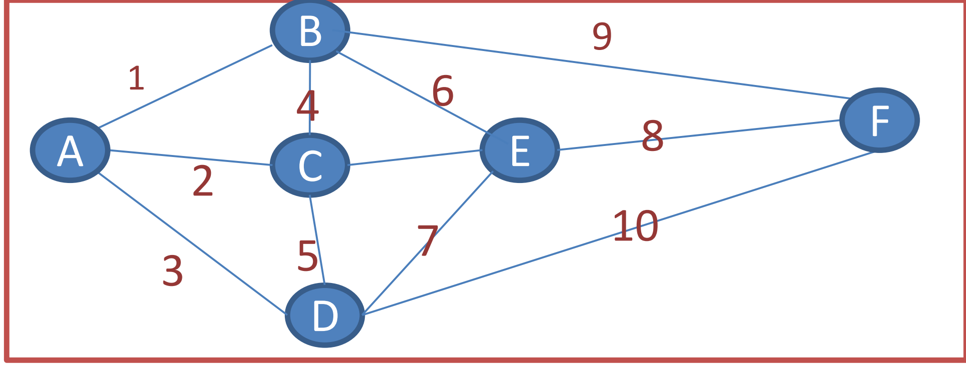
Entrées: G=(S,A) un graphe avec une pondération positive poids des arcs, S_{deb} un sommet de S.

P:=0
d[a]:=+ ∞
d[S_{deb}]=0

Initialisation

Tant que la liste P ne contient pas tout les sommets.
    Choisir un sommet s hors de P de plus petite distance d[s]
    Mettre s dans P
    Pour chaque sommet a hors de P adjacent de s
        d[a]=min(d[a],d[s]+poids(s, a))
    Fin pour
Fin Tant Que

Traitements



P={A,B,C}

Sommet	A	B	C	D	E	F
A	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞
B		1(A)	2(A)	3(A)	+ ∞	+ ∞
C			5(B)	3(A)	7(B)	10(B)

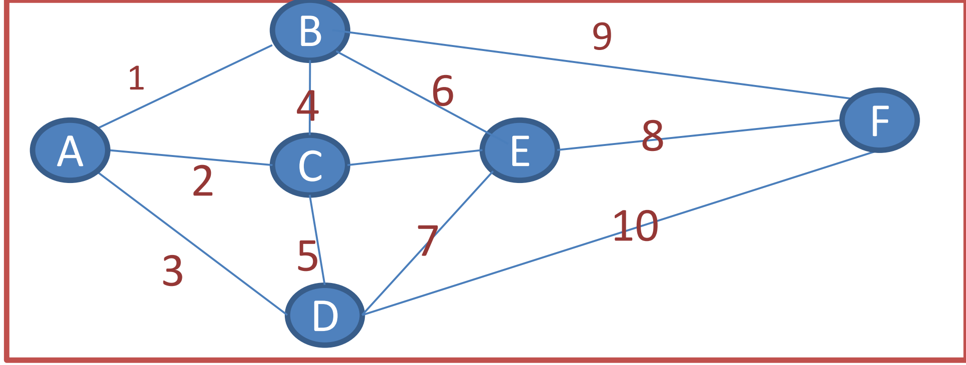
Algorithme de Dijkstra:

Entrées: G=(S,A) un graphe avec une pondération positive poids des arcs, S_{deb} un sommet de S.
P:=0
d[a]:=+ ∞
d[S_{deb}]=0

$\Big\}$ Initialisation

Tant que la liste P ne contient pas tout les sommets.
    Choisir un sommet s hors de P de plus petite distance d[s]
    Mettre s dans P
    Pour chaque sommet a hors de P adjacent de s
        d[a]=min(d[a],d[s]+poids(s, a))
    Fin pour
Fin Tant Que

$\Big\}$ Traitements



P={A,B,C,D}

Sommet	A	B	C	D	E	F
A	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞
B		1(A)	2(A)	3(A)	+ ∞	+ ∞
C			5(B)	3(A)	7(B)	10(B)
D				7(C)	12(C)	10(B)

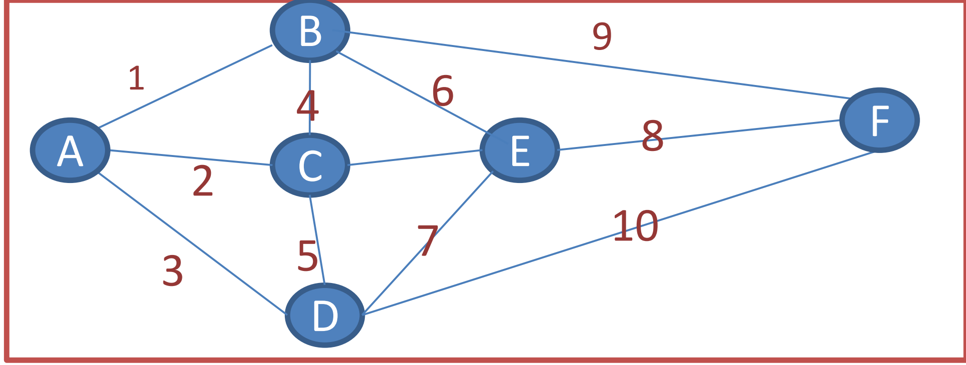
Algorithme de Dijkstra:

Entrées: G=(S,A) un graphe avec une pondération positive poids des arcs, S_{deb} un sommet de S.
P:=0
d[a]:=+ ∞
d[S_{deb}]=0

Initialisation

Tant que la liste P ne contient pas tout les sommets.
    Choisir un sommet s hors de P de plus petite distance d[s]
    Mettre s dans P
    Pour chaque sommet a hors de P adjacent de s
        d[a]=min(d[a],d[s]+poids(s, a))
    Fin pour
Fin Tant Que

Traitements



P={A,B,C,D,E}

Sommet	A	B	C	D	E	F
A	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞
B		1(A)	2(A)	3(A)	+ ∞	+ ∞
C			5(B)	3(A)	7(B)	10(B)
D				7(C)	12(C)	10(B)
E					10(D)	13(D)

Algorithme de Dijkstra:

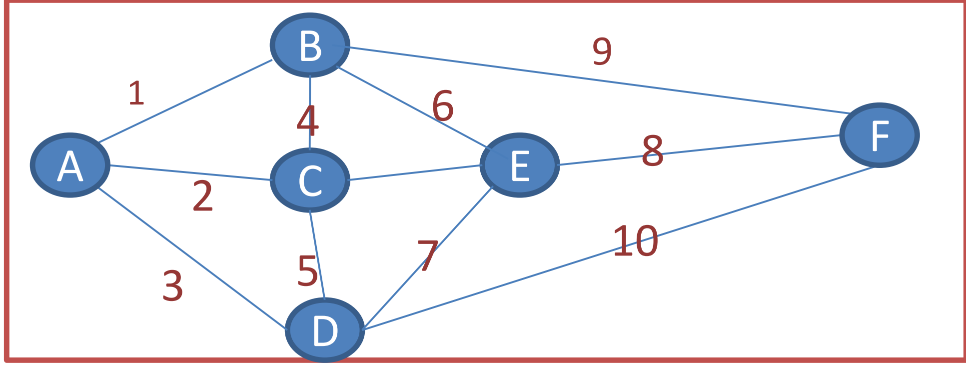
Entrées: G=(S,A) un graphe avec une pondération positive poids des arcs, S_{deb} un sommet de S.

P:=0
d[a]:=+ ∞
d[S_{deb}]=0

Initialisation

Tant que la liste P ne contient pas tout les sommets.
    Choisir un sommet s hors de P de plus petite distance d[s]
    Mettre s dans P
    Pour chaque sommet a hors de P adjacent de s
        d[a]=min(d[a],d[s]+poids(s, a))
    Fin pour
Fin Tant Que

Traitements



P={A,B,C,D,E,F}

Sommet	A	B	C	D	E	F
A	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞	+ ∞
B		1(A)	2(A)	3(A)	+ ∞	+ ∞
C			5(B)	3(A)	7(B)	10(B)
D				7(C)	12(C)	10(B)
E					10(D)	13(D)
F						15(F)

Complexité

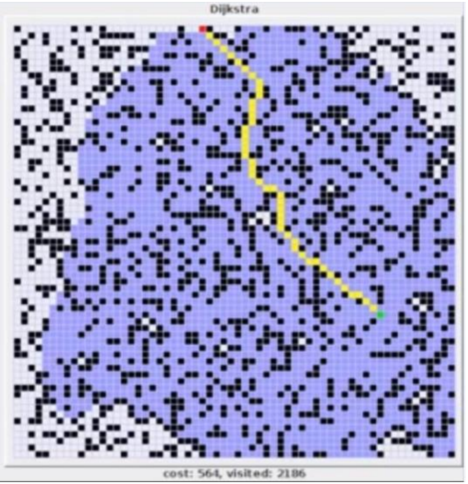
La complexité, ou le coût, d’un algorithme ou d’une fonction Python est le nombre d’opérations élémentaires nécessaires à son exécution dans le pire cas.

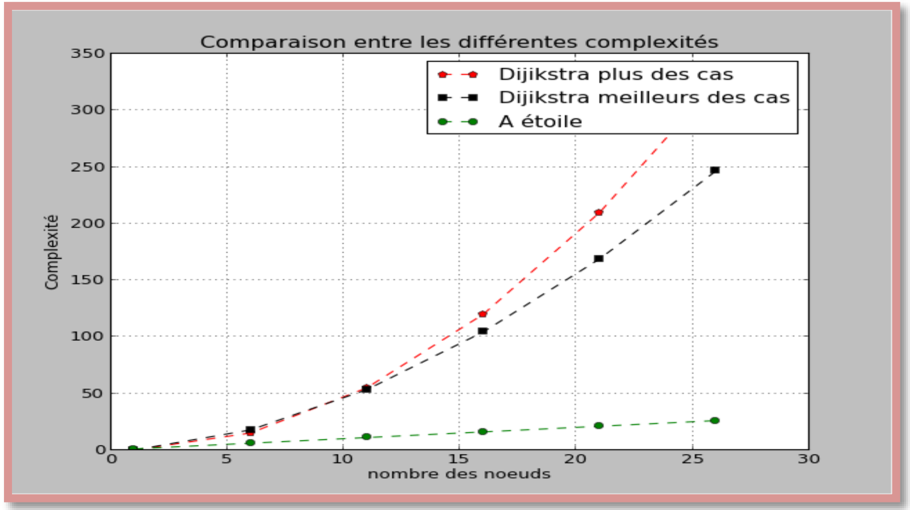
Calcul de complexité

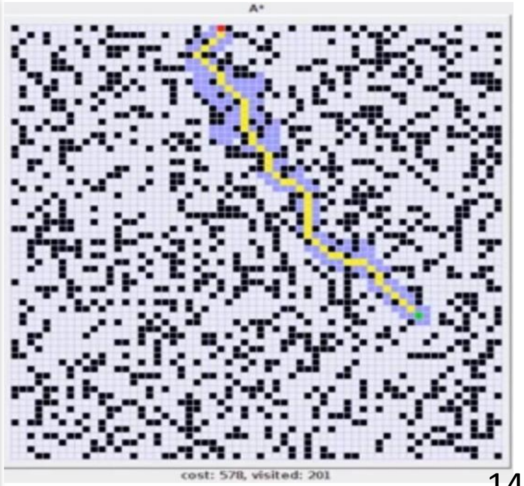
Dijkstra: $O(n^2)$

n nœuds et a arcs

A* (Star): $O(a)$



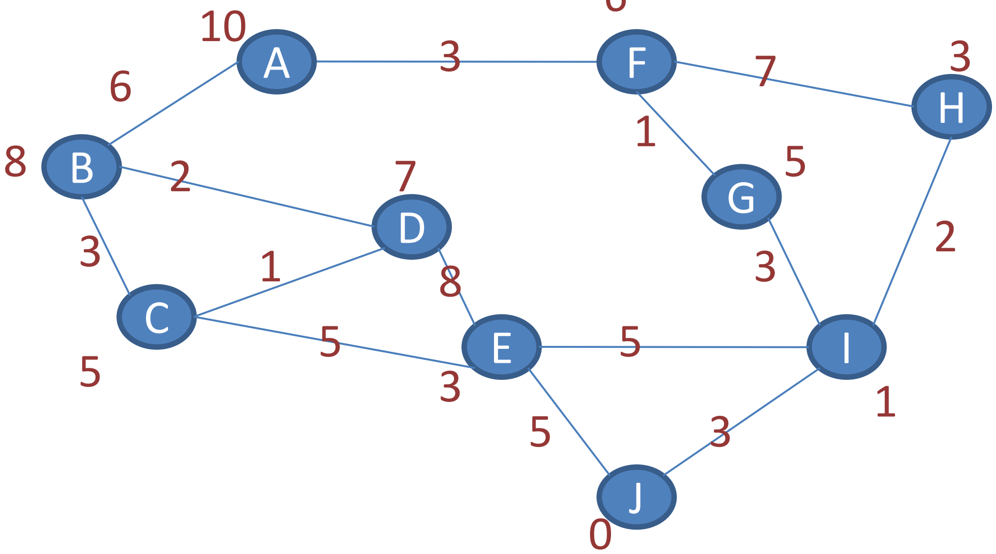


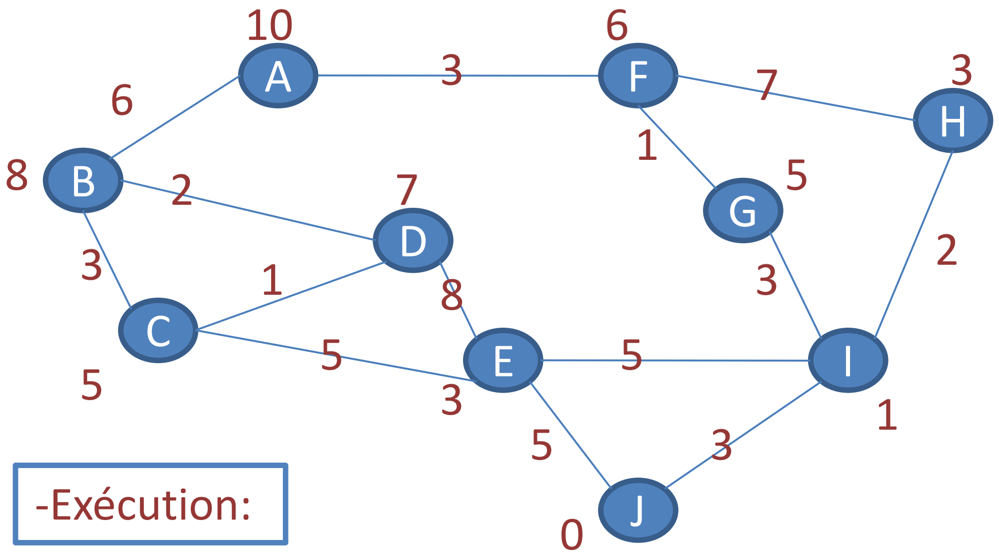


L'algorithme A*(star):

-Principe: éviter de parcourir des chemins estimés trop long par rapport à ceux connus.
-f(n): une fonction d'évaluation de son coût total pour un sommet n.
-h(n): une heuristique de coût estimé (minimum) pour aller d'un sommet n vers un autre n+1.
-g(n): le coût réel pour atteindre le sommet **n**.
-La fonction f(n) est le coût total estimé :

$$f(n) = h(n)+g(n)$$





-Exécution:

F(B)=6+8=14
F(F)=3+6=9
F(F) <F(B)

Chemin: A-F

F(G)=3+1+5=9
F(H)=7+3+3=13
F(G) <F(H)

Chemin: A-F-G

F(I)=3+1+3+1=8

Chemin: A-F-G-I

F(E)=3+1+3+5+3=15
F(H)=3+1+3+2+3=12
F(J)=3+1+3+3+0=10

Chemin: A-F-G-I-J

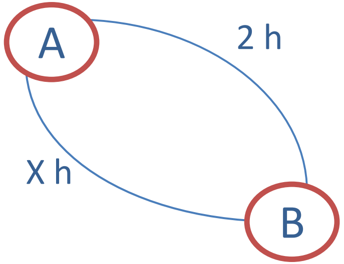
Le chemin le plus court pour aller du point A au point J est selon l'Algorithme A*:

A-F-G-I-J

Problème: Difficulté des mathématiques discrètes à inclure des contraintes comme: « le comportement collectif des conducteurs »

Solution: Théorie des jeux: Le Paradoxe de Braess

Le phénomène:



Conducteur au point A

Deux scénarios

Le comportement égoïste: chaque conducteur est libre de choisir sa route.

Le comportement social: le conducteur est forcé à prendre une certaine route.

Quel est le meilleur choix?

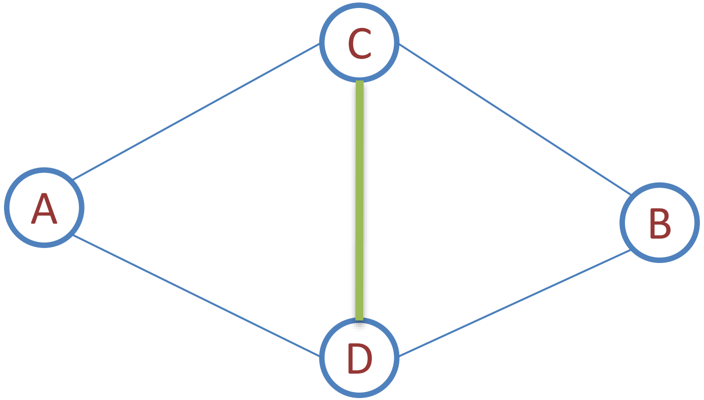
Optimum social: un chemin pour lequel le temps total du trajet est aussi petit:
On le note: T-social

Optimum égoïste: chaque conducteur trouve son compte avec un chemin qui est caractérisé par un conducteur qui change de route sans que le autres le suivent. On note le temps du trajet égoïste: T- égoïste

Modélisation:

Problème: Embouteillage

Solution: Construction d'une voie express A-C avec T=T-exp



Chaque conducteur choisit son chemin pour minimiser égoïstement son propre temps?

A-D
C-B

Des routes principales : le temps de parcours est indépendant de x noté T

A-C
D-B

Des routes étroites avec de nombreux feux, stop: le temps de parcours dépendant de x. T(x)=x/100 où x:nombre de conducteur.

On pose: x=500 / T-in=10 min/ T=0=T(total)/ T-exp=4 min

Explication:

-Premier choix: A-C-B: où A-D-B: T(total)=15 minutes

-Second choix: A-D: T=10 où A-C: $x/100 \leq 5$ → le choix sera: A-C

C-B: : T=10 min
C-B-D: $T=4+x/100 \leq 4+10=14$
→ A-C-D-B: T=14+10=24min
A-C-B: T=15 min

→ Résultat contraire : T augmente avec la nouvelle route.

T-égoïste=24 min> T-social=15 min

- Hypothèses: Soient les hypothèses simplificatrices suivantes:

-Les voitures circulent sur une autoroute à voie unique.
-La taille de chaque voiture est négligeable.
-Chaque voiture a une vitesse inhérente v qui diffère l'un à l'autre.
-Interdiction de dépasser une voiture.
-Si il n y a pas de voiture devant: la voiture bouge avec v
 sinon: les voitures les plus lentes empêchent les voitures les plus rapides d'aller de l'avant.
-L'avancement est induit par les différences des vitesses.

$X_i(t)$:La position
$V_i(t)$:La vitesse

-Si la voiture i approche la voiture plus lente i+1 devant, la vitesse de i devient égale à celle de i+1.

(1)

(2)

(1) Quand une voiture plus rapide approche d'une voiture plus lente, la voiture la plus rapide assume la vitesse de la voiture la plus lente et deux voitures se déplacent ensemble avec la même vitesse.

(2) La voiture se déplace avec la même vitesse inchangée.

Equation de burgers:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

condition initiale:

$$u(x,0) = u_0(x)$$

-Cette équation peut être linéaire à l'aide de la transformation de **Hopf-cole**:

-On pose: $$u(x,t) = -\frac{2\nu}{\phi}\frac{\partial \phi}{\partial x}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

Un nouveau changement de variable donne:

$$\varphi = -2\ln(\phi)$$

Une intégration par partie donne:

$$\frac{\partial \varphi}{\partial t} + \frac{1}{2}\varphi_x^2 = \nu\frac{\partial^2 \varphi}{\partial x^2}$$

On trouve l' équation de la chaleur:

$$\frac{\partial \phi}{\partial t} = \nu\frac{\partial^2 \phi}{\partial x^2}$$

Equation de la chaleur:

$$\frac{\partial \phi}{\partial t} = \nu \frac{\partial^2 \phi}{\partial x^2}$$

La transformée de Fourier

$$\tilde{\phi}(k,t) = \sqrt{\sigma}\, e^{-\frac{K^2 \sigma}{2}} e^{-DK^2 t}$$

Transformée de Fourier inverse

-On peut appliquer comme condition initiale une impulsion de Dirac: T(x,0)=δ(x) mais elle ne représente pas vraiment un phénomène physique, on utilise alors une gaussienne

Conditions initiales et aux limites :

-T(0,t)=0 ∀t

-T(∞,t)=0 ∀t

$$-T(x,0) = \frac{e^{-\frac{x^2}{2\sigma^2}}}{\sqrt{2\Pi}} \forall x$$

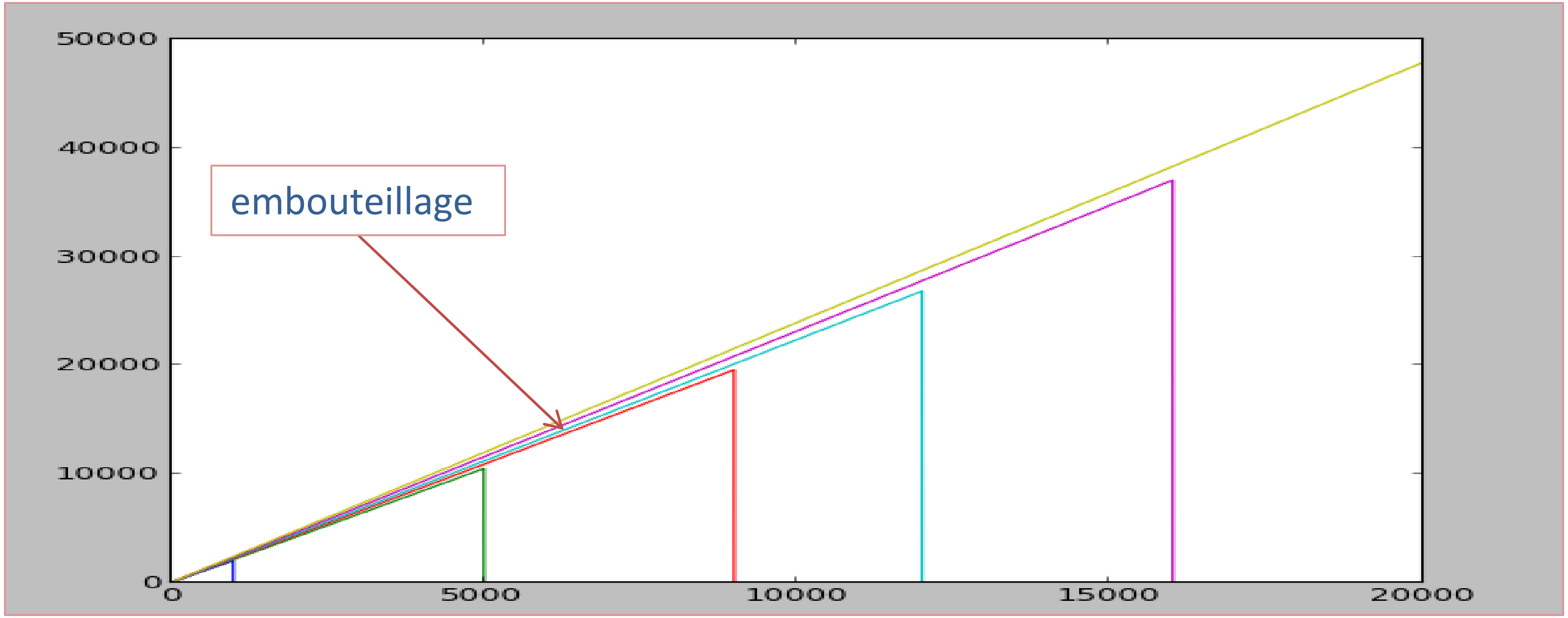
$$\phi(x,t) = \frac{\sqrt{2\sigma}}{\sqrt{4Dt + 2\sigma}} e^{-\frac{x^2}{4Dt+2\sigma}}$$

Hopf-Cole inverse

On retrouve La vitesse

$$U(x,t) = \frac{4\nu x}{4Dt + 2\sigma}$$

Simulation numérique:



Une augmentation excessive de vitesse provoque des embouteillages.

Conclusion: Pendant mon travail durant cette année:

I

-Algorithme de Dijekstra: Algorithme itératif (force brute): mauvaise complexité.

II

-Algorithme A*: Algorithme basé sur une heuristique: Amélioration de la complexité grâce à l'intelligence artificielle.

III

-Paradoxe de Braess: utilisation de la théorie de jeux pour modéliser les interactions sociales.

IV

-Modélisation physique avec les équations aux dérivées partielles.

V

-Problème difficile nécessite une approche multidisciplinaire.

Annexe:

Courbe Complexité:

```
import numpy as np
import matplotlib.pyplot as plt

N_total_noeuds = 44

liste_noeuds = range(1,N_total_noeuds,5)

def dijikstra_complexite_pire_cas(noeuds):

    return noeuds*(noeuds-1)*0.5

def dijikstra_complexite_meilleur_cas(noeuds,aretes):

    return aretes + noeuds*np.log(noeuds)

def a_etoile_complexite(noeuds):

    return  noeuds

def densite_graphe(noeuds,aretes):

    return 2*(aretes)/(noeuds*(noeuds - 1))
```

```
densite = 0.5
aretes = [int(densite * (n*(n-1))/2) for n in liste_noeuds]

complexite1 = [dijikstra_complexite_pire_cas(x) for x in liste_noeuds]

complexite2 = [dijikstra_complexite_meilleur_cas(x,y) for x,y in zip(liste_noeuds,aretes)]

complexite3 = [a_etoile_complexite(x) for x in liste_noeuds]


plt.plot(liste_noeuds,complexite1,"rp--",label="Dijikstra plus des cas")
plt.plot(liste_noeuds,complexite2,"ks--",label="Dijikstra meilleurs des cas")
plt.plot(liste_noeuds,complexite3,"go--",label="A étoile ")

plt.xlabel("nombre des noeuds")
plt.ylabel("Complexité")

plt.legend()
plt.title("Comparaison entre les différentes complexités")

plt.grid()


plt.yscale('log')
plt.show()
```

Courbe du graphe:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx

file = pd.read_excel("tableau distance.xlsx")

type(file)
liste_arete = file.iloc[:,0:3].values
type(liste_arete)



liste_arete.shape

dictionnaire = {}
Nl = liste_arete.shape[0]
Nc = liste_arete.shape[1]

for i in range(Nl):

    dictionnaire[liste_arete[i,0]] = {liste_arete[i,1]: {'weight':liste_arete[i,2]}}


import matplotlib.pyplot as plt
import networkx as nx

G = nx.Graph()
```

```python
for i in range(N1):
    G.add_edge(liste_arete[i,0], liste_arete[i,1], weight=liste_arete[i,2])




elarge = [(u, v) for (u, v, d) in G.edges(data=True) if d['weight'] > 0.5]
esmall = [(u, v) for (u, v, d) in G.edges(data=True) if d['weight'] <= 0.5]

pos = nx.spring_layout(G)  # positions for all nodes

# nodes
nx.draw_networkx_nodes(G, pos, node_size=700)

# edges
nx.draw_networkx_edges(G, pos, edgelist=elarge,
                       width=6)
nx.draw_networkx_edges(G, pos, edgelist=esmall,
                       width=6, alpha=0.5, edge_color='b', style='dashed')

# labels
nx.draw_networkx_labels(G, pos, font_size=20, font_family='sans-serif')

plt.axis('off')
plt.show()
```