Optimisation du temps de trajet lors d'un déplacement en milieu urbain

PLUMAIL Jean - n° 18922

Comment optimiser le temps de trajet d'un véhicule en milieu urbain ?

Problème du plus court chemin dans un graphe à pondération variable

I) Présentation des modèles

II) Solutions au problème

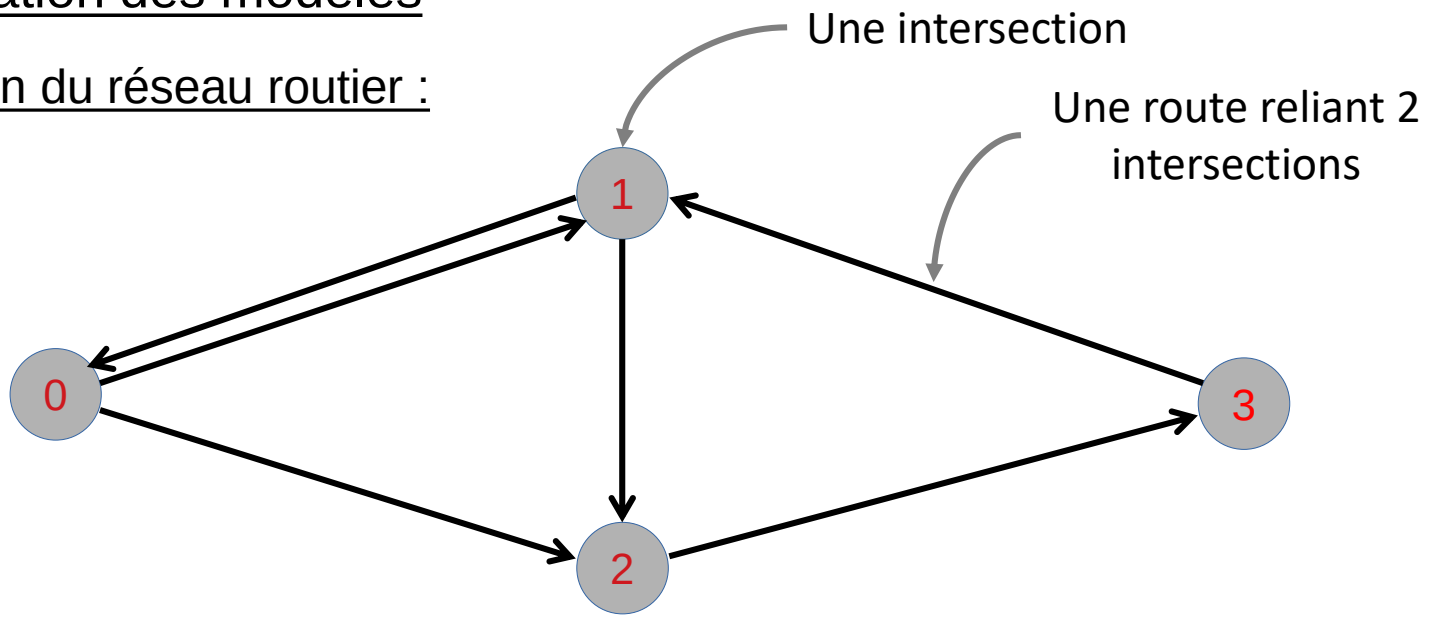
III) Étude d'un cas réel

IV) Influence sur le trafic

I) Présentation des modèles

Modélisation du réseau routier :

G = (S, A)

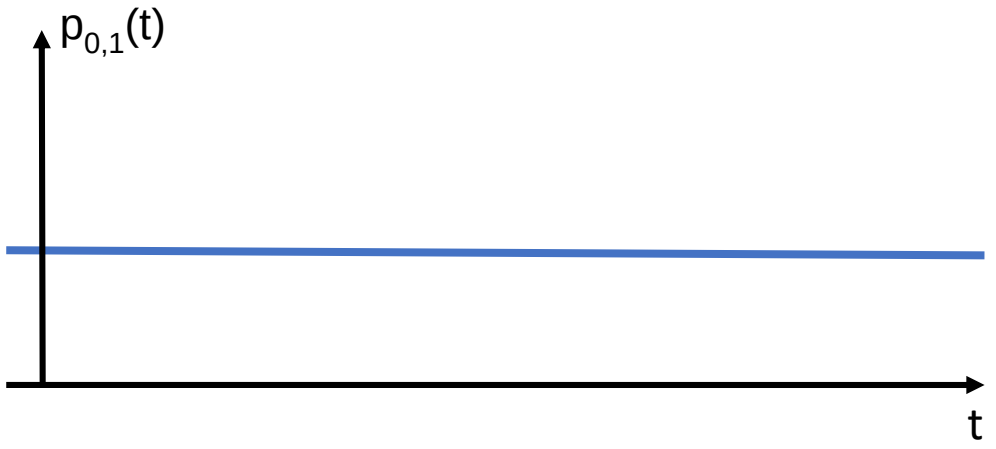


S = [0 ; 1 ; 2 ; 3]

A = [(0,1) ; (1,0) ; (0,2) ; (1,2) ; (3,1) ; (2,3)]

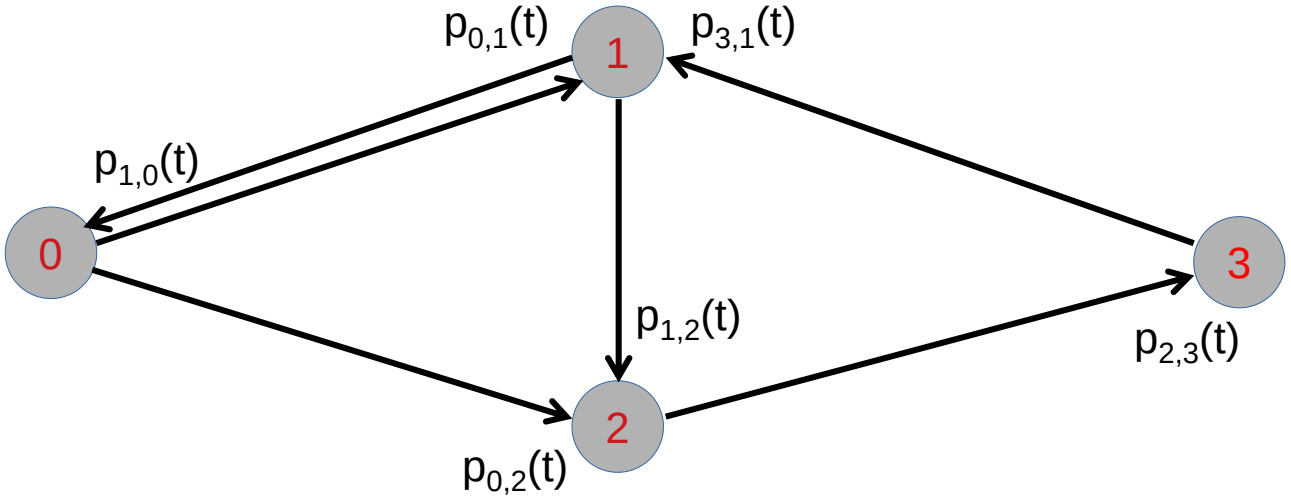
Une fonction poids :

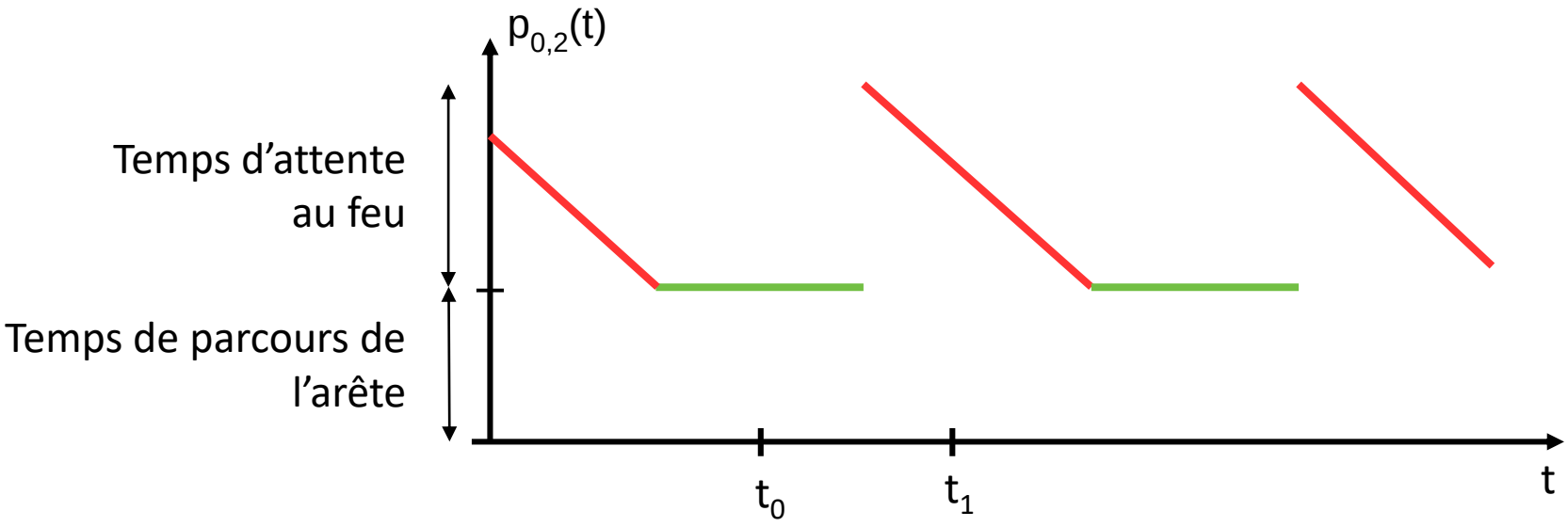
$$P : S \times S \times \mathbb{R} \to \mathbb{R}^{+}$$
$$(u, v, t) \mapsto p_{u,v}(t)$$



I) Présentation des modèles

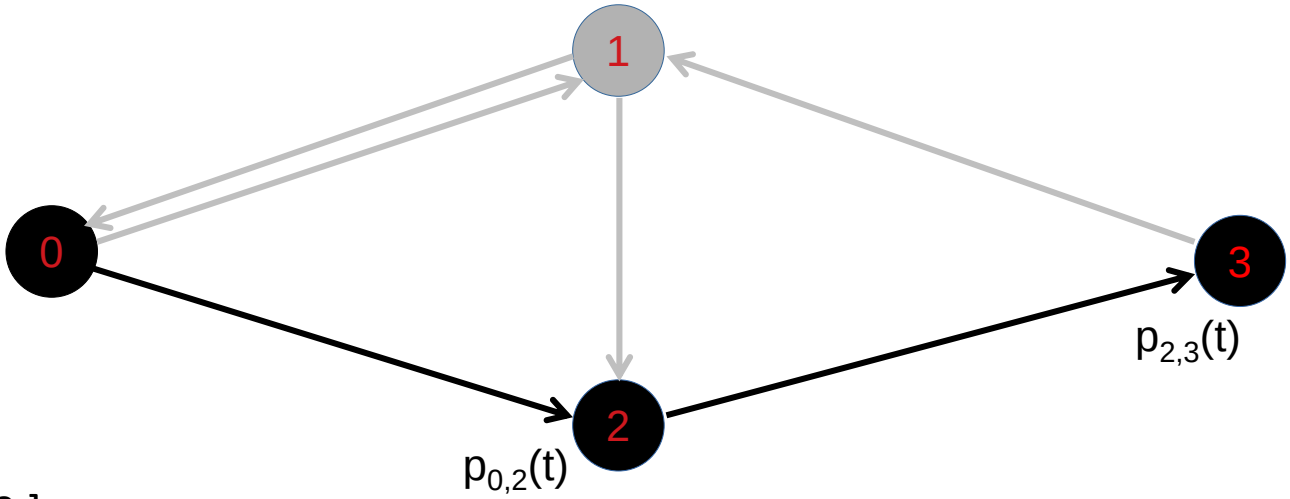
Modélisation d'un feu :





I) Présentation des modèles

Chemin : liste de sommets $c = [s_0, s_1, \dots, s_k]\ tel\ que : \forall i \in \{0,1,\dots,k-1\},\ (s_i, s_{i+1}) \in A$



c = [0 ; 2 ; 3]

Coût d'un chemin :

$$Cout : \left((s_0, s_1, \dots, s_k),\ t\right) \mapsto \begin{cases} Cout((s_0, s_1, \dots, s_{k-1}),\ t) + p_{s_{k-1}, s_k}(t + Cout((s_0, s_1, \dots, s_{k-1}),\ t)) \text{ si k > 0} \\ 0 \text{ si } k=0 \end{cases}$$

Plus court chemin entre 2 sommets a et b à un instant t :

$$PCC(a, b, t) = \min_{\substack{c=(a,s_1,\dots,s_k,b) \\ k\in\mathbb{N}}} Cout(c, t)$$

II) Solutions au problème

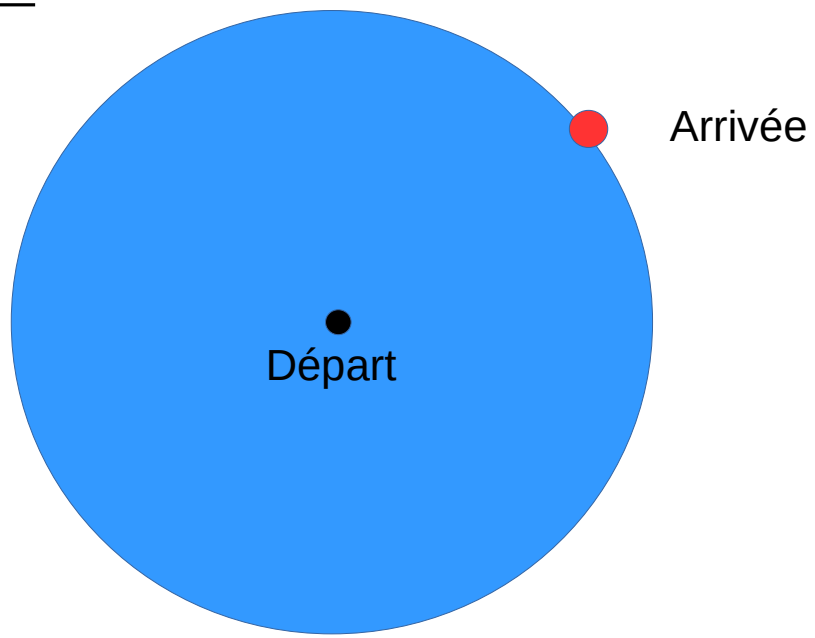
Méthode naïve pour trouver le PCC :

- Trouver tous les chemins du départ à l'arrivée
- Appliquer la fonction *Cout* à chacun d'entre eux
- Renvoyer le chemin ayant le coût minimal

L'algorithme a une complexité exponentielle par rapport au nombre de sommets du graphe.

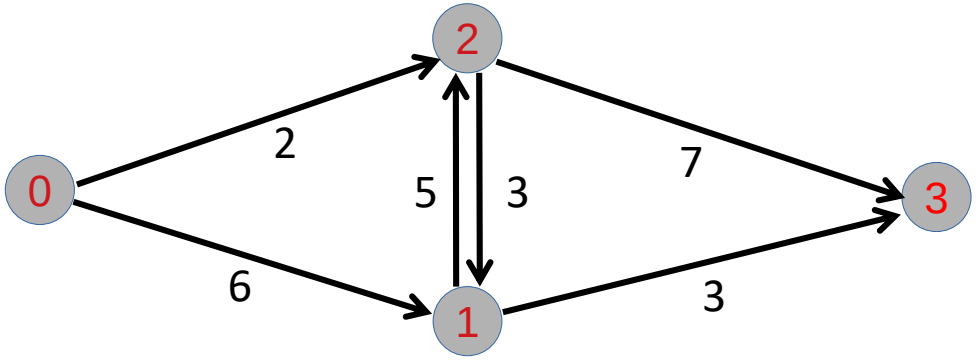
Un moyen plus adapté : Dijkstra

Une progression circulaire :



II) Solutions au problème

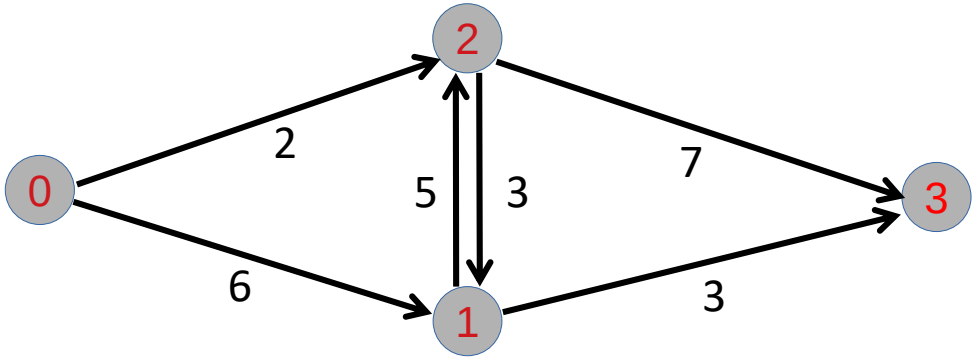
Un exemple :



0	1	2	3
0	∞	∞	∞

II) Solutions au problème

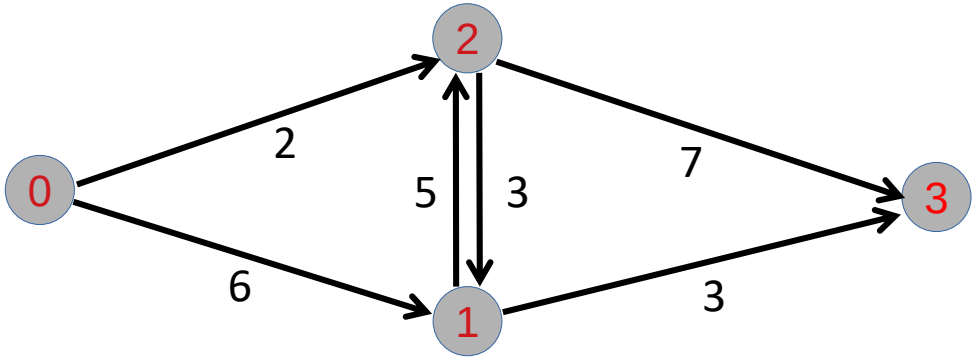
Un exemple :



0	1	2	3
0	∞	∞	∞
0	6	2	∞

II) Solutions au problème

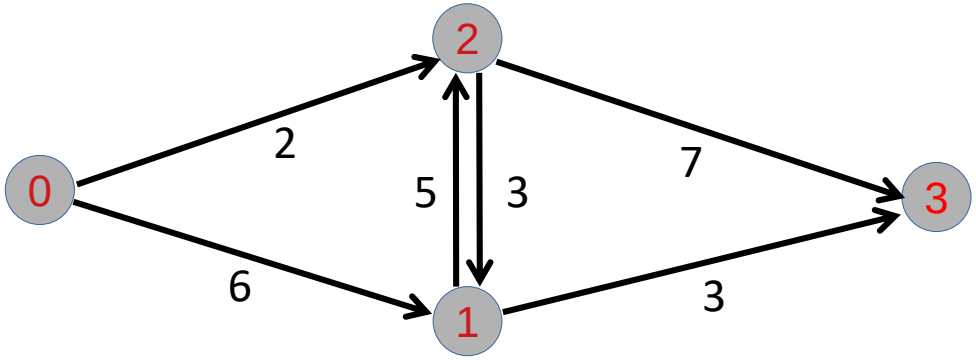
Un exemple :



0	1	2	3
0	∞	∞	∞
0	6	2	∞
	5	2	9

II) Solutions au problème

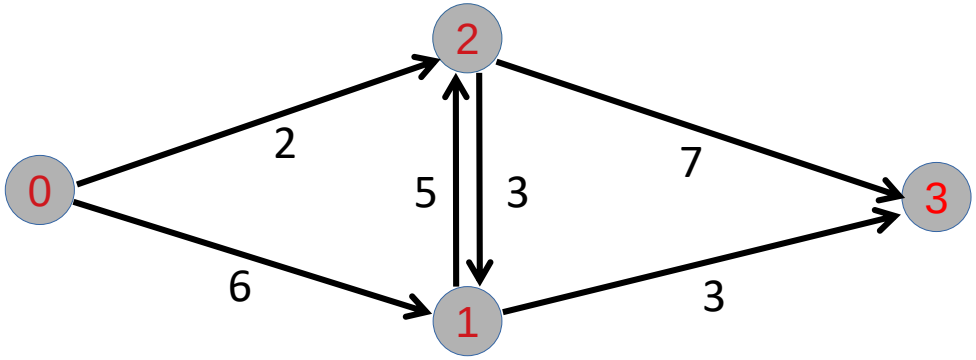
Un exemple :



0	1	2	3
0	∞	∞	∞
0	6	2	∞
	5	2	9
	5		8

II) Solutions au problème

Un exemple :



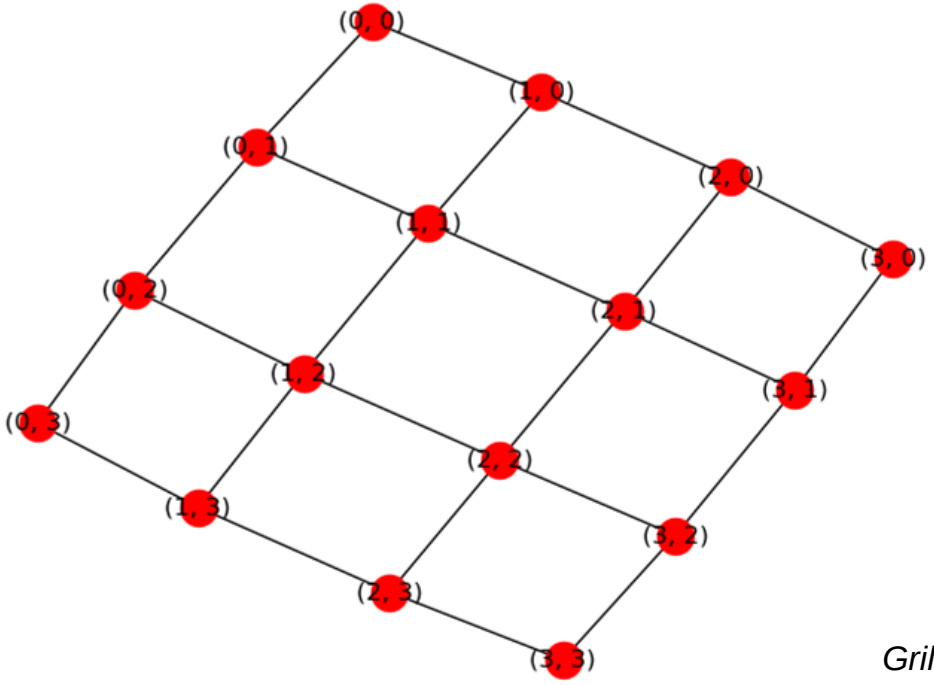
0	1	2	3
0	∞	∞	∞
0	6	2	∞
	5	2	9
	5		8
			8

Cet algorithme fonctionne avec des poids dépendants du temps tant que le graphe possède la propriété First-In-First-Out. [5]

II) Solutions au problème

- Cet algorithme est-il efficace ?
- Combien de temps gagne-t-on avec un assistant de navigation qui prend en compte les feux ?

Cas de la grille :



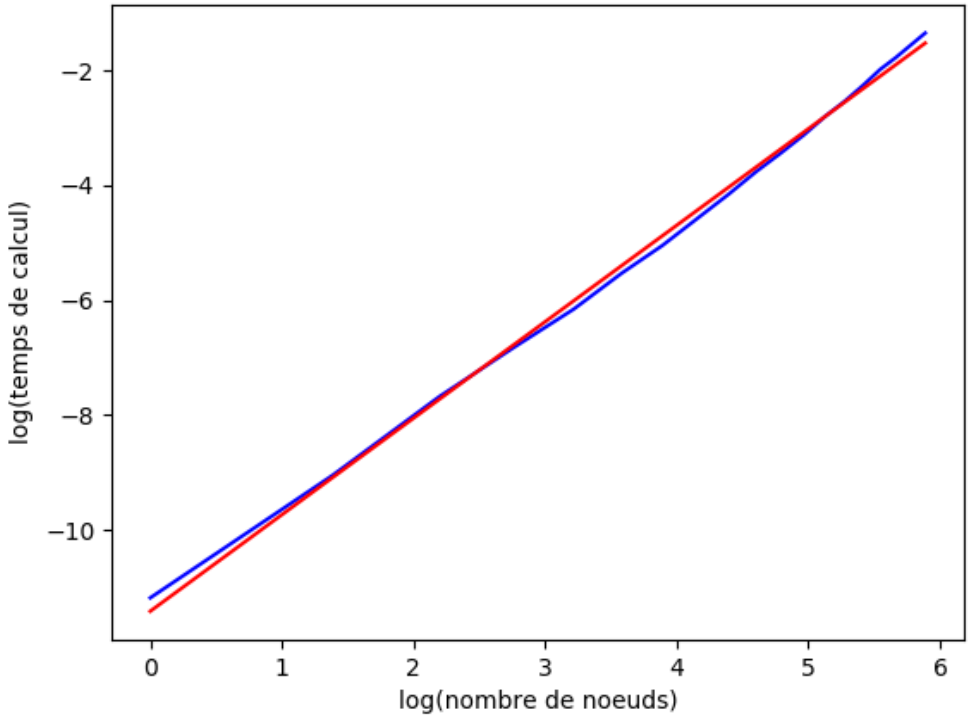
Grille 4x4

Des feux choisis de manière « réaliste ».

II) Solutions au problème

- L'algorithme est-il efficace ?

Étude de la complexité de Dijkstra :



Simulation faite sur des grilles 2x2 à 20x20 (k=50)
Régression linéaire en rouge (pente de 1,7)

Expérimentalement, l'algorithme a une complexité en $O(n^{1,7})$ où n est le nombre de nœuds dans le graphe.

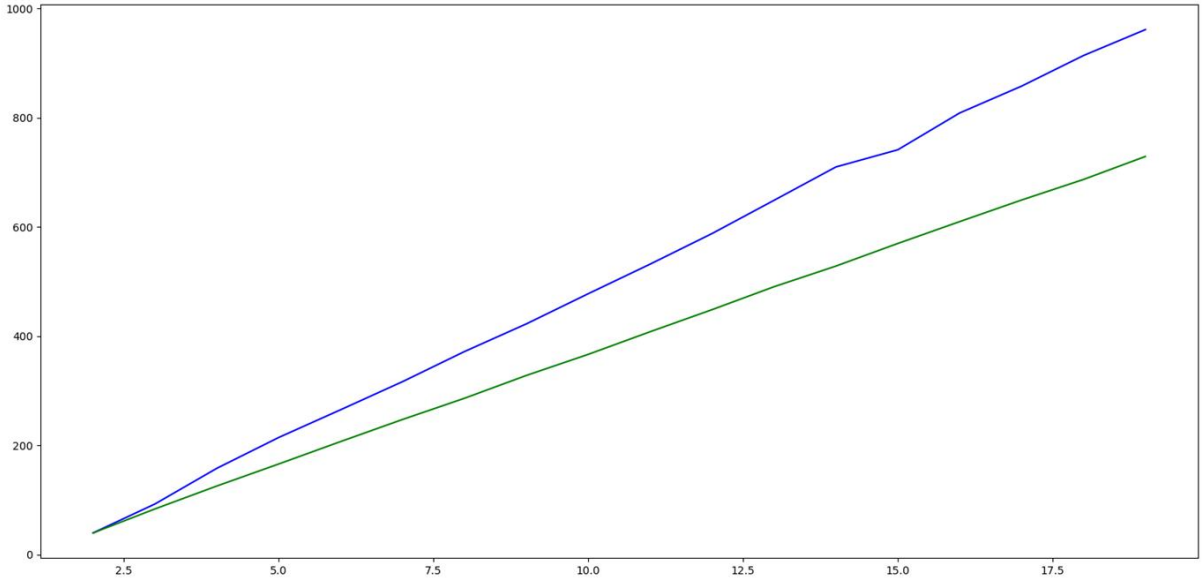
II) Solutions au problème

- Combien de temps gagne-t-on avec un assistant de navigation qui prend en compte les feux ?

On calcule un chemin C_1 avec la fonction poids P_1 (sans les feux)
On calcule un chemin C_2 avec la fonction poids P_2 (avec les feux)
On évalue ces chemins avec la fonction poids P_2

Comparaison des coûts moyens (selon P_2) de C_1 et de C_2 en fonction de la taille de la grille

Temps de trajet (s)



Taille de la grille

Simulation faite avec la période des feux fixée à 30s.

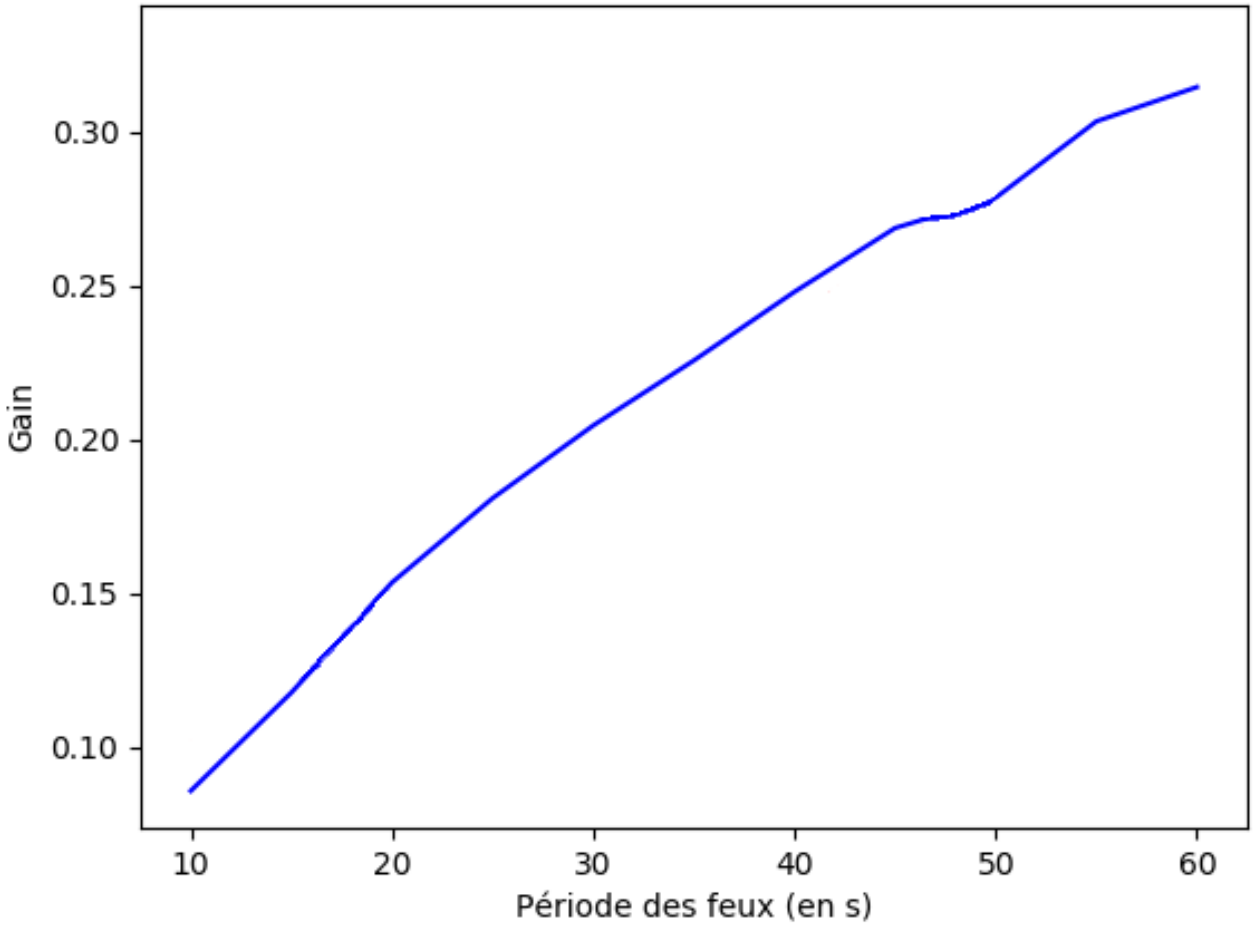
C_1 (assistant sans les feux)

C_2 (assistant avec les feux)

II) Solutions au problème

Influence de la période des feux :

Gain en fonction de la période des feux de la grille



Simulation faite sur une grille 5x5 (k=100)

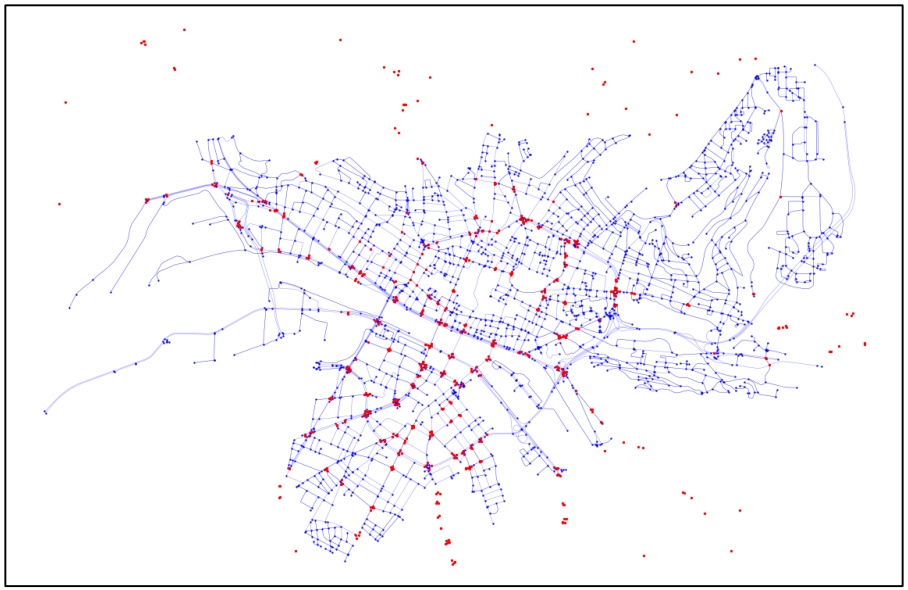
III) Étude d'un cas réel

Importation des données :

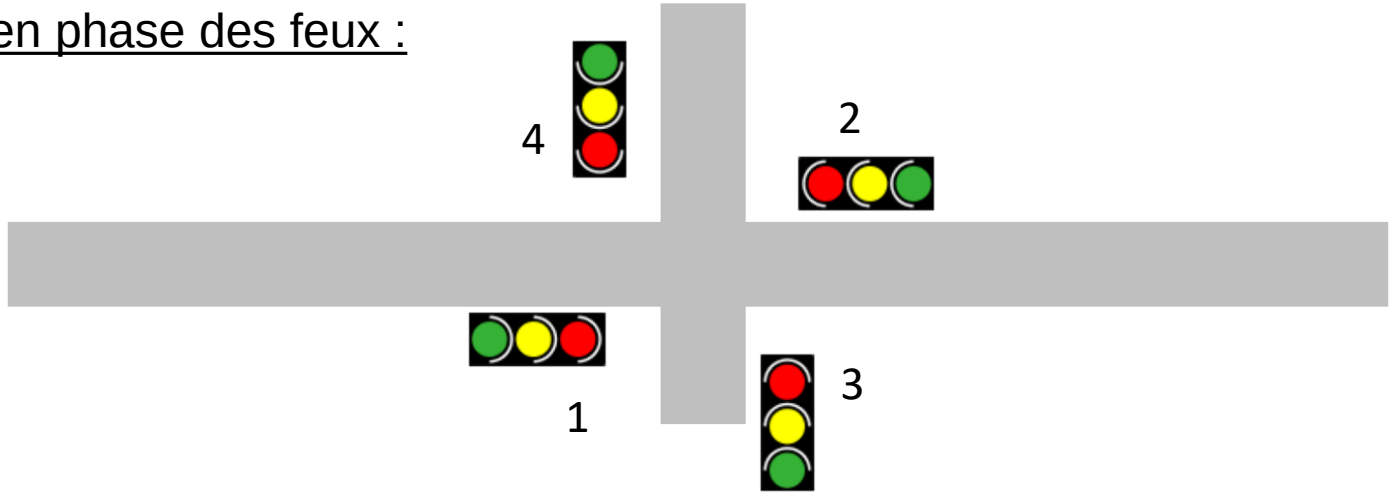
Le graphe de la carte routière d'une métropole obtenu grâce à OSMnx

III) Étude d'un cas réel

Importation des feux de la ville



Mise en phase des feux :



III) Étude d'un cas réel

Application des algorithmes au graphe de la ville :

Comparaison de 2 chemins

chemin calculé avec l'assistant sans les feux : 371 s

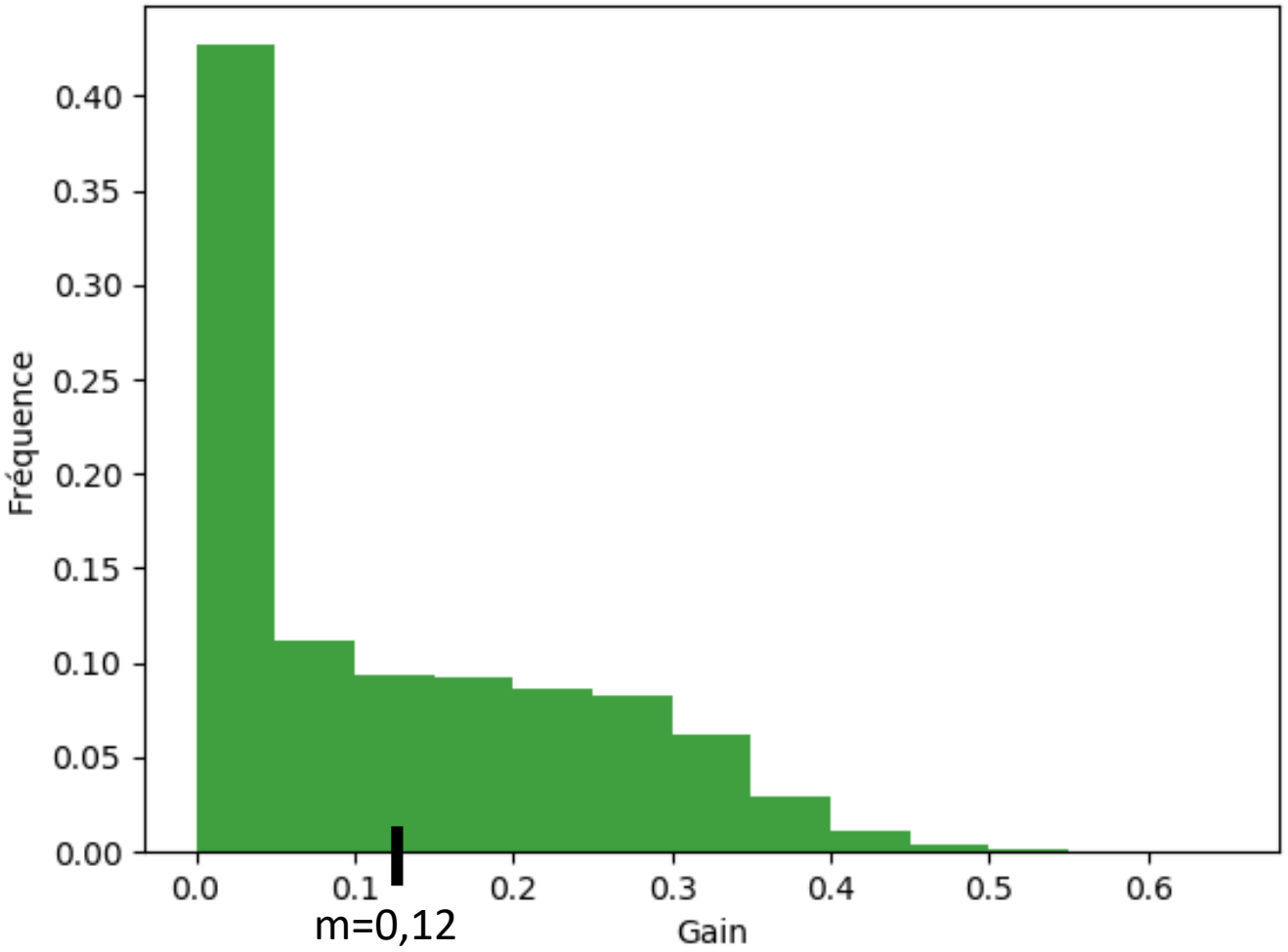
chemin calculé avec l'assistant avec les feux : 219 s

Gain de 41%

On fixe la période des feux à 30 s.

III) Étude d'un cas réel

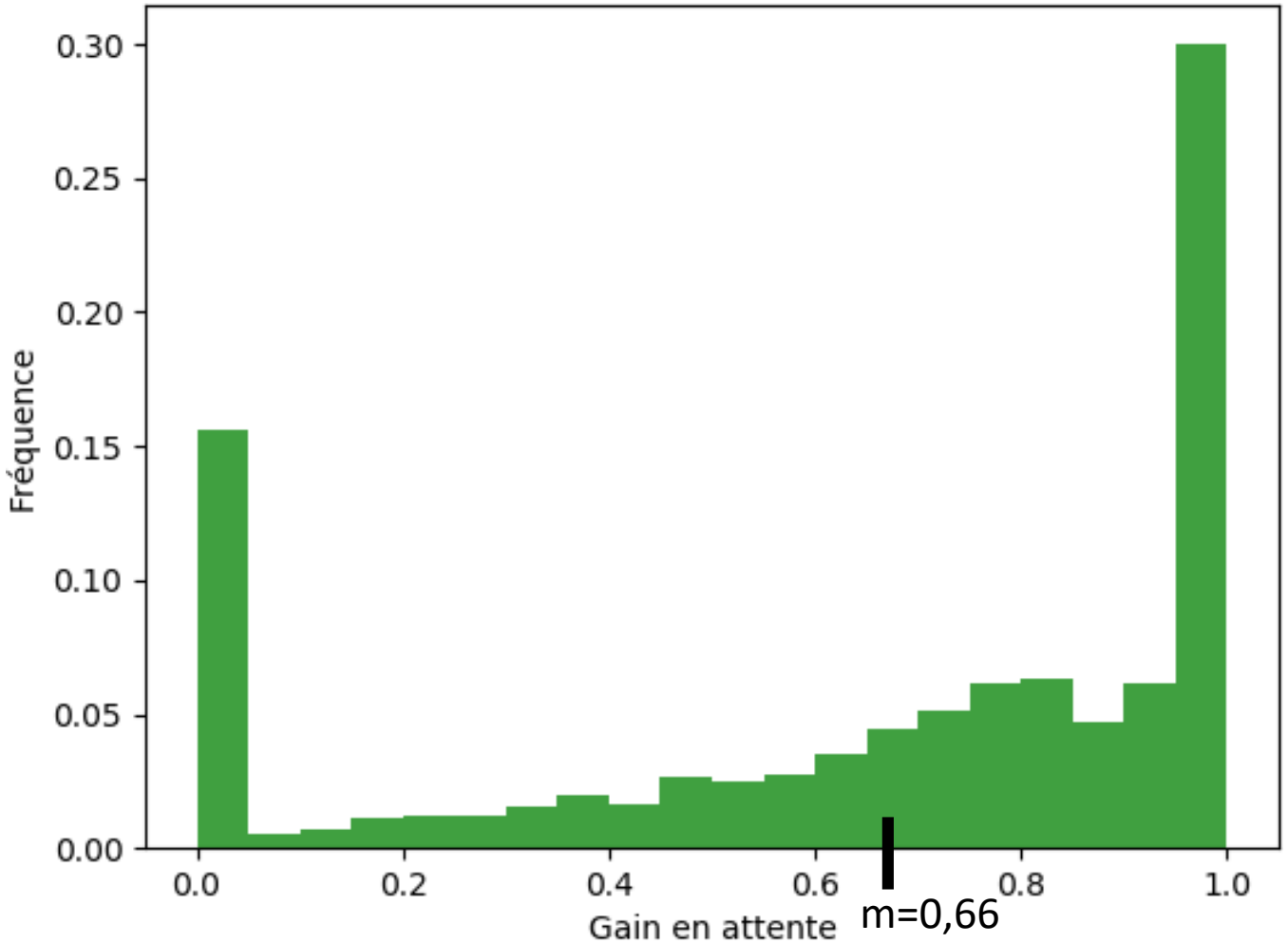
Répartition des gains pour 110 500 trajets différents



Simulation faite à partir de 50 nœuds vers tous les autres nœuds du graphe. (n=2211)

III) Étude d'un cas réel

Répartition des gains en attente pour 98 196 trajets différents



Simulation faite à partir de 50 nœuds vers tous les autres nœuds du graphe. (n=2211)

12 304 trajets sans assistant avaient déjà une attente nulle (11% des trajets).

III) Étude d'un cas réel

Ces simulations sont-elles pertinentes ?
Dans la pratique, on ne connaît pas les états initiaux et les phases exacts des feux, les trajets proposés par l'assistant restent-ils plus rapides ?

Expérience pratique :

- Réalisée dans une commune plus petite. (trajets plus rapides)
- Choix de 5 trajets dont l'algorithme prédit plus de 20 s d'économie.
- On compare les gains théoriques et expérimentaux.

III) Étude d'un cas réel

1

2

5

3

4

En rouge, les trajets calculés par l'assistant de navigation qui prend en compte les feux.

III) Étude d'un cas réel

Résultats :

Numéro	Économie théorique	Économie expérimentale	Écart
1	32 s	5 s	27 s
2	30 s	25 s	5 s
3	26 s	-15 s	41 s
4	20 s	30 s	10 s
5	43 s	10 s	33 s
Moyenne			23,2 s

Pour des trajets de courtes durées (environ 3min), une économie annoncée de plus de 20 s suffit pour gagner du temps dans la pratique.

IV) Influence sur le trafic

L'utilisation de l'algorithme à grande échelle peut-elle provoquée des embouteillages supplémentaires ?

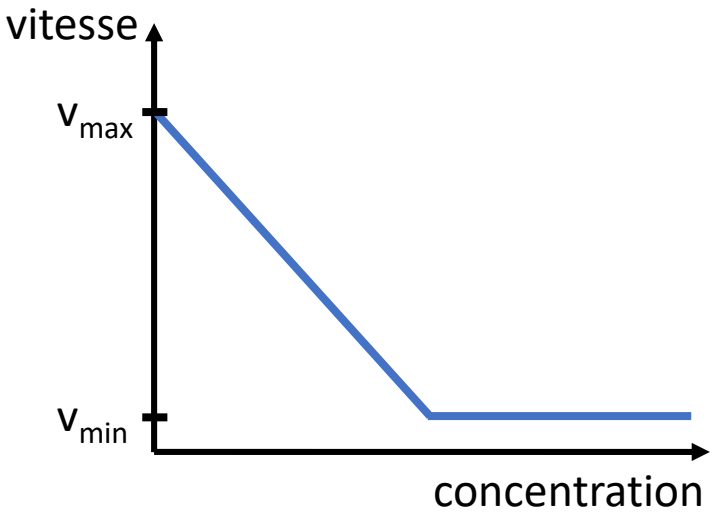
Zone industrielle

Zone résidentielle

IV) Influence sur le trafic

Présentation du modèle :

La vitesse du véhicule dépend de la densité du trafic sur l'arête :

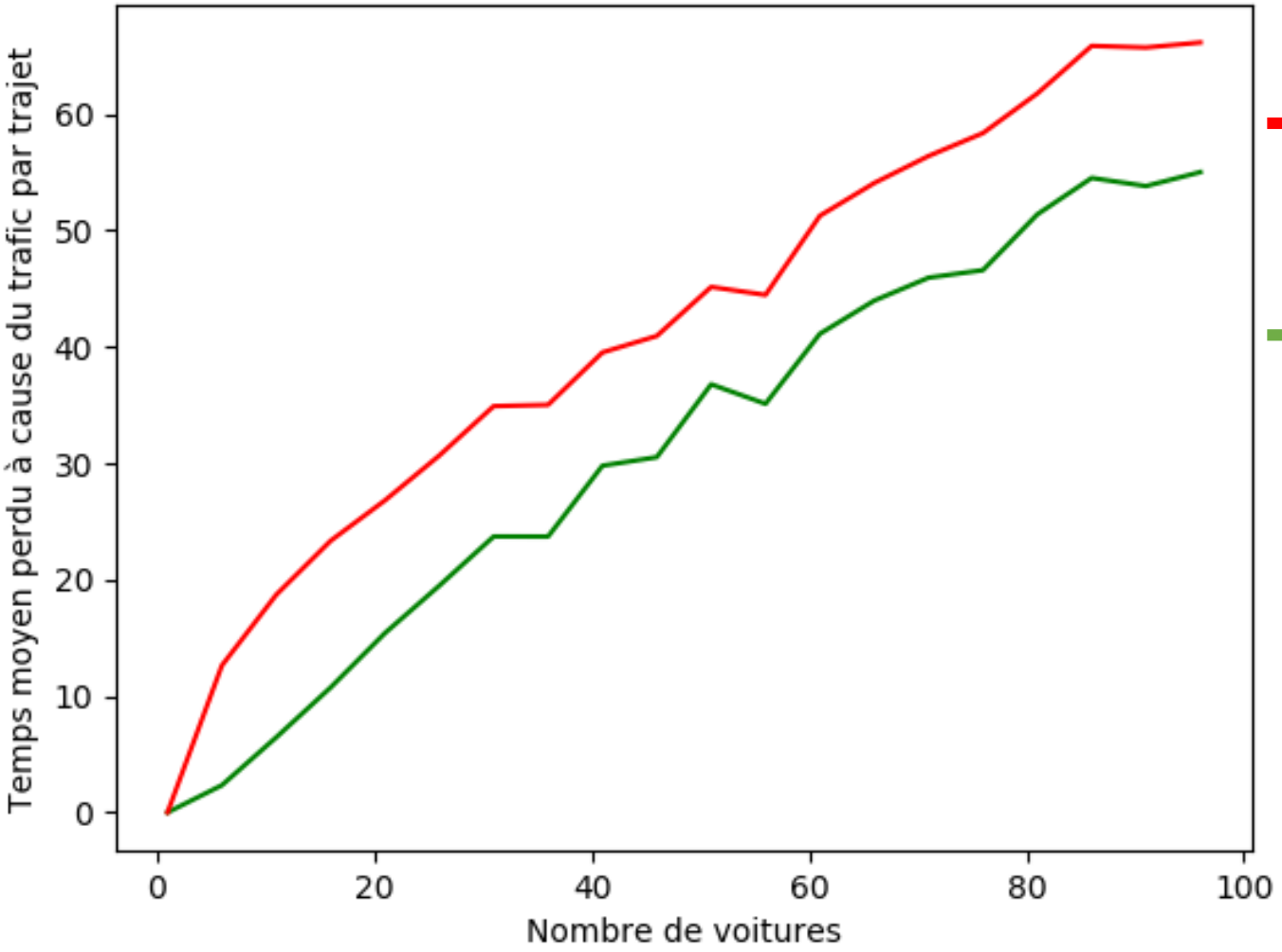


On définit également un débit μ = { 0,5 voitures par secondes quand le feu est vert ; 0 quand le feu est rouge

Tant qu'il reste des véhicules devant, la voiture reste sur l'arête.

IV) Influence sur le trafic

Durée moyenne perdue dans le trafic en fonction du nombre de voitures



Tous les véhicules utilisent l'assistant avec les feux

Tous les véhicules utilisent l'assistant sans les feux

Simulation réalisée pour des véhicules partant tous de la zone résidentielle dans un intervalle de temps de 30 s. (k=10)

```
'''matrice feu : matrice nxn (n nombre d'intersections)
                        c(i,j) -> état initial et période du feu se situant à l'intersection j
venant de i'''


def attente(F,i,j,t):
        '''arguments : matrice de feux, de i vers j, temps
        renvoie le temps d'attente à ce feu'''
        d=0
        tfeu=F[i][j][0]
        if tfeu == 0:
                return 0
        cycle = F[i][j][1]
        quotient, reste = t // cycle, t% cycle
        if quotient%2 == 0:
                if abs(tfeu)>reste:
                        if tfeu<=0:
                          d = tfeu + reste
                        else:
                          d = tfeu - reste
                else:
                        if tfeu>0:
                          d= -cycle + (-tfeu+reste)
                        else:
                          d=cycle - (tfeu+reste)
        else :
                if abs(tfeu)>reste:
                        if -tfeu<=0:
                          d=-tfeu+reste
                        else:
                          d=-tfeu-reste
                else:
                        if -tfeu>0:
                          d=-cycle+(tfeu+reste)
                        else:
                          d=cycle-(-tfeu+reste)
        return d
```

```
def cout_chemin(chemin,F,G,t0):
        ''' arguments : chemin (liste de feux), matrice des feux, matrice graphe
                renvoie le cout en temps du chemin'''
        if len(chemin) == 0:
                return float('inf')
        cout = t0
        for k in range(len(chemin)-1):
                cout += G[chemin[k],chemin[k+1]]
                a = max(0,attente(F, chemin[k],chemin[k+1], cout))
                cout += a
        return cout-t0

def attente_chemin(chemin,F,G,t0):
        ''' arguments : chemin (liste de feux), matrice des feux, matrice graphe
                renvoie le cout en temps du chemin'''
        if len(chemin) == 0:
                return float('inf')
        attente_tot = 0
        cout = t0
        for k in range(len(chemin)-1):
                cout += G[chemin[k],chemin[k+1]]
                a = max(0,attente(F, chemin[k],chemin[k+1], cout))
                cout += a
                attente_tot += a
        return attente_tot
```

```
def chemins_naif(A,P,position,objectif):
        '''arguments : matrice adjacente, noeuds déjà parcourus, position actuelle,
objectif
           renvoie la liste de tous les chemins possibles de la position à
l'objectif'''
        if position == objectif:
                return [[objectif]]
        else:
                P2 = list(P)
                P2[position] = True
                liste_chemins = []
                for i in range(len(A)):
                        if (A[position,i] != 0) and (P[i] == False):
                                liste_chemins2 = chemins_naif(A,P2,i,objectif)
                                for j in range(len(liste_chemins2)):
                                        liste_chemins2[j] = [position] +
liste_chemins2[j]
                                liste_chemins = liste_chemins + liste_chemins2
                return liste_chemins

def sol_naive(G,F,depart,objectif,t0):
        n = len(G)
        P = [False for i in range(n)]
        liste_chemins = chemins_naif(G,P,depart,objectif)
        meilleur_chemin = liste_chemins[0]
        meilleur_cout = cout_chemin(meilleur_chemin,F,G,t0)
        for i in range(1,len(liste_chemins)):
                if cout_chemin(liste_chemins[i],F,G,t0) < meilleur_cout:
                        meilleur_cout = cout_chemin(liste_chemins[i],F,G,t0)
                        meilleur_chemin = liste_chemins[i]
        return meilleur_chemin
```

```
def dijkstra2(G,depart):
    n = len(G)
    PCC = [[] for i in range(n)]
    T = [float('inf') for i in range(n)]

    PCC[depart] = [depart]
    T[depart] = 0
    Q = set([depart])
    F = set()
    while len(Q) != 0:
        cout_min = float('inf')
        for j in Q:
            if len(PCC[j]) != 0:
                cout = T[j]
                if cout < cout_min:
                    i = j
                    cout_min = cout
        Q = Q - set([i])
        F = F | set([i])
        for j in range(n):
                if G[i,j] != 0:
                    nouveau_cout_j = T[i] + G[i,j]
                    if nouveau_cout_j < T[j] or (len(PCC[j]) == 0):
                        PCC[j] = PCC[i] + [j]
                        T[j] = nouveau_cout_j
                        Q = Q | set([j])
    return PCC,T
```

```
def dijkstra_dynamique2(G,feux,depart,t0):
    n = len(G) #n=nb noeuds
    PCC = [[] for i in range(n)] #initialisation liste chemins les plus
courts
    T = [float('inf') for i in range(n)]
    PCC[depart] = [depart]
    T[depart] = t0
    Q = set([depart])
    F = set()
    while len(Q) != 0:
        cout_min = float('inf')
        for j in Q:
               if len(PCC[j]) != 0:
                      if T[j] < cout_min:
                             i = j
                             cout_min = T[j]
        Q = Q - set([i])
        F = F | set([i])
        for j in range(n):
            if G[i,j] != 0: #s'ils sont voisins
                date_i = T[i]
                nouvelle_date_j = date_i +
cout_chemin([i,j],feux,G,date_i)
                if nouvelle_date_j < T[j] or (len(PCC[j]) == 0):
                    PCC[j] = PCC[i] + [j]
                    T[j] = nouvelle_date_j
                    Q = Q | set([j])
    return PCC, T
```

```
def vitesse(L,N,vmax,n_voies):
        Kmax = (1/6)*n_voies
        K = N/L
        v = vmax*(1-K/Kmax)
        return max(1.4,v) # vitesse d'embouteillage à 5km/h
def parcours_arete(t0, L, N, vmax,n_voies, F, i, j):
        mus = 0.5*n_voies # débit sortant par seconde quand le feu est vert
        v = vitesse(L,N-1,vmax,n_voies) # en m.s-1

        n = N
        t = t0
        if F[i,j][0] == 0 and F[i,j][1] == 0: # il n'y a pas de feu
                t = t0 + (N-1)/mus
        else:
                while n-1>0:
                        att = attente(F,i,j,t)
                        if abs(att) < 10**(-3):
                                if att > 0: att = -F[i,j][1]
                                else: att = F[i,j][1]
                        if att < 0: # le feu est vert
                                n = n + mus*att
                                t = t - att
                        else:
                                t = t + att
        t = t+(n-1)/mus
        if v>0:
                if t < t0 + L/v:
                        t = t0 + L/v + max(0,attente(F,i,j,t0+L/v))
        return t
```

```
def evenements(F,voitures,aretes,index):
        '''voitures (int*liste) liste : liste des dates de départ*chemin des voitures
           renvoie E la matrice des évènements'''
        n = len(F)
        E = [[[] for i in range(n)] for j in range(n)]
        N = np.zeros((n,n))
        # la file se présente ainsi : num voiture*à l'instant t*arête précédente
        file_p = [(i,v[0],(-1,v[1][0])) for i,v in enumerate(voitures)]
        file_p = sorted(file_p, key=lambda j: j[1])
        while len(file_p) > 0:
                e = file_p.pop(0)
                num, t0, (u,v) = e
                chemin = voitures[num][1]
                # la voiture quitte le tronçon précédent
                if u > -1:
                        N[u,v] -= 1
                        E[u][v].append((num,t0,-1))
                # si le trajet n'est pas terminé
                if v != chemin[-1]:
                        w = chemin[chemin.index(v)+1]
                        # la voiture arrive sur un nouveau tronçon
                        N[v,w] += 1
                        E[v][w].append((num,t0,1))
                        # On récupère n le nombre de voitures sur le tronçon i-j, vmax et L la
longueur du tronçon
                        n = N[v,w]
                        voies, vmax, l = get_infos(aretes,index,v,w)
                        # on calcule la date de sortie du tronçon
                        t1 = parcours_arete(t0, l, n, vmax,voies, F, v, w)
                        # on met à jour la file de priorité
                        n_e = (num,t1,(v,w))
                        if len(file_p) > 0:
                                k = 0
                                while k<len(file_p) and t1>file_p[k][1]:
                                        k += 1
                                file_p = file_p[:k] + [n_e] + file_p[k:]
                        else:
                                file_p = [n_e]
        return E
```

```
def creer_dic(noeuds, aretes, feux):
	'''aretes : base de donnees des aretes (avec position
géographique précise/géométrie)
	feux : base de donees des feux (numéro associé à un
point(x,y))
	renvoie une base de données qui associe à chaque feu les
arêtes qu'il intersecte'''
	dic = {}
	to_delete = []
	aretes['feu'] = False
	for i, feu in feux.iterrows():
		compteur = 0
		dic[feu['osmid']] = []
		for j,arete in aretes.iterrows():
			if
feu['geometry'].intersects(arete['geometry']):
				compteur += 1
				dic[feu['osmid']].append(arete)
		if compteur == 0:
			to_delete.append(feu['osmid'])

	for osmid in to_delete:
		del dic[osmid]
	feux_aretes(dic,noeuds,aretes,feux)
	return dic
```

```
def feux_aretes(dic, noeuds, aretes, feux):
        '''dic : base de données feux/arêtes
        aretes et feux : base de données arêtes et feux
        modifie dic pour montrer quelles sont réellement les arêtes qui concernent le
feu'''
        for cle in dic.keys():
                for a in dic[cle]:
                        a['feu'] = False
        for cle in dic.keys():
                if len(dic[cle]) == 1:
                        dic[cle][0]['feu'] = True
                if len(dic[cle]) == 2:
                        u = dic[cle][0]['u']
                        v = dic[cle][0]['v']
                        posu = noeuds.loc[u,'geometry']
                        posv = noeuds.loc[v,'geometry']
                        posf = feux.loc[cle, 'geometry']
                        if posu.distance(posf) > posv.distance(posf):
                                dic[cle][0]['feu'] = True
                        else:
                                dic[cle][1]['feu'] = True
                if len(dic[cle]) > 2:
                        posf = feux.loc[cle,'geometry']
                        mini = 1000
                        for a in dic[cle]:
                                v = a['v']
                                posv = noeuds.loc[v,'geometry']
                                if posv.distance(posf) < mini:
                                        mini = posv.distance(posf)
                                        i = v
                        for a in dic[cle]:
                                if a['v'] == i:
                                        a['feu'] = True
```

```
def modif_aretes(aretes, dic):
        '''ajoute un champ "feu" dans la base de données arêtes avec un booléen qui
indique s'il y a un feu'''
        aretes['feu'] = 0

        for feu in dic.keys():
                for arete in dic[feu]:
                        if arete['feu']:
                                aretes.loc[arete.name, 'feu'] = feu

def creer_intersections(aretes,noeuds):
        '''creer une base intersections qui liste les intersections : une intersection
constituée d'arêtes'''
        aretes['groupe_feu'] = 0
        aretes_feux = aretes[aretes.feu != 0]

        noeuds['intersection_feu'] = False
        intersections = []
        c = {i:0 for i in range(10)}
        for i, noeud in noeuds.iterrows(): #On itère à travers les noeuds
                intersection = aretes_feux[aretes_feux.v==int(noeud.osmid)] #On réunit
toutes les arêtes ayant un feu qui arrivent sur le noeud

                if len(intersection) > 0:
                        intersections.append(intersection)
                        noeuds.loc[i,'intersection_feu'] = True
                        c[len(intersection)] += 1
        c = list(c.values())
        print(sum(c))
        c = [i/sum(c) for i in c]
        print(c)
        noeuds_feux = noeuds[noeuds.intersection_feu==True]

        return intersections, aretes_feux, noeuds_feux
```

```
def coupler_feux(intersections):
        '''modifie intersections pour lier les feux qui sont en phase
           si 2 feux sont en phase, ils ont le même numéro 'groupe feu'
           si les 2 feux sont en opposition de phase, ils ont leurs 'groupe feu' qui ont
le même quotient par rapport à 2'''
        i = 1
        for intersection in intersections:
                h = intersection['highway'].value_counts()
                if len(intersection) < 3:
                        intersection.loc[:,'groupe_feu'] = i
                elif len(h) >= 2 and h.max() == 2:
                        intersection.loc[intersection['highway']==h.idxmax(),
'groupe_feu'] = i
                        intersection.loc[intersection['highway']!=h.idxmax(),
'groupe_feu'] = i+1
                else: # on cherche 2 arêtes qui ont le même nom
                        nom = intersection.iloc[0]['name']
                        nb_noms_uniques = intersection['name'].value_counts()
                        print(intersection.columns)
                        if nb_noms_uniques.shape[0] == 2:

        intersection.loc[intersection['name']==nb_noms_uniques[:1].index.values.astype(st
r)[0],'groupe_feu'] = i

        intersection.loc[intersection['name']==nb_noms_uniques[1:].index.values.astype(st
r)[0],'groupe_feu'] = i+1
                        else:

        intersection.loc[intersection.index.values[:len(intersection)//2],'groupe_feu'] =
i

        intersection.loc[intersection.index.values[len(intersection)//2:],'groupe_feu'] =
i+1
                i += 2
```

```
def inter_vers_aretes(aretes,intersections):
        for intersection in intersections:
                for i,arete in intersection.iterrows():
                        aretes.loc[arete.name, 'groupe_feu'] = arete['groupe_feu']



def feux_vers_aretes(noeuds,aretes,feux):
        dic = creer_dic(noeuds,aretes,feux)
        modif_aretes(aretes,dic)
        intersections, _, _ = creer_intersections(aretes,noeuds)
        coupler_feux(intersections)
        inter_vers_aretes(aretes,intersections)
        return aretes
```

```
def data_vers_mat(noeuds, aretes, infos):
        periode = infos[0]
        n = len(noeuds)

        index_n = noeuds['osmid'].astype('int64')
        index_n = list(index_n)
        matadj = np.zeros((n,n))
        matfeu = np.zeros((n,n,2))
        c = 0
        for i, arete in aretes.iterrows():
                v = arete['maxspeed']
                if v is None or v == 0:
                        v = 50.
                else:
                        if type(v) == list:
                                v = [float(i) for i in v]
                                v = sum(v)/len(v)
                        else:
                                v = float(v)
                                if np.isnan(v): v = 50.
                        t = float(arete['length']) / (v/3.6)
                matadj[index_n.index(arete['u']),index_n.index(arete['v'])] = t
        i = 1
        p = aretes['groupe_feu'].max()
        for i in range(1,p+1,2):
                r = random()
                for j, arete in aretes[aretes['groupe_feu']==i].iterrows():
                        matfeu[index_n.index(arete['u']),index_n.index(arete['v'])] =
[-periode*r,periode]
                for j, arete in aretes[aretes['groupe_feu']==i+1].iterrows():
                        matfeu[index_n.index(arete['u']),index_n.index(arete['v'])] =
[periode*r,periode]
        return matadj,matfeu,index_n
```