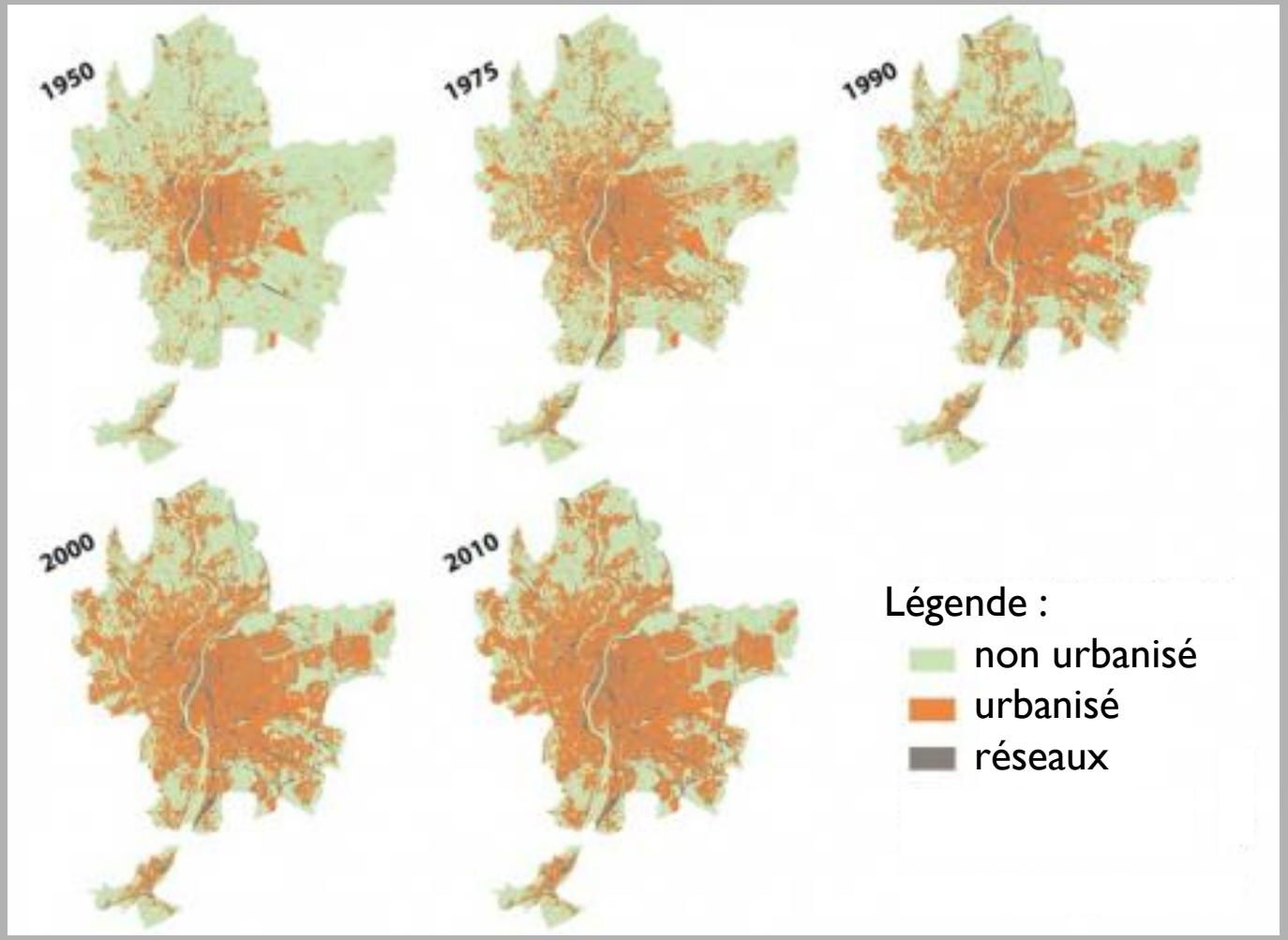
GESTION DU TRAFIC ROUTIER À UN CARREFOUR

INTRODUCTION

Etalement urbain de la ville de Lyon sur 60 ans



PROBLÉMATIQUE

Comment utiliser l'outil numérique afin d'optimiser le trafic routier à un carrefour et y fluidifier la circulation ?

PLAN

1. Présentation du modèle
2. Numérisation du problème
3. Conclusion

HYPOTHÈSES

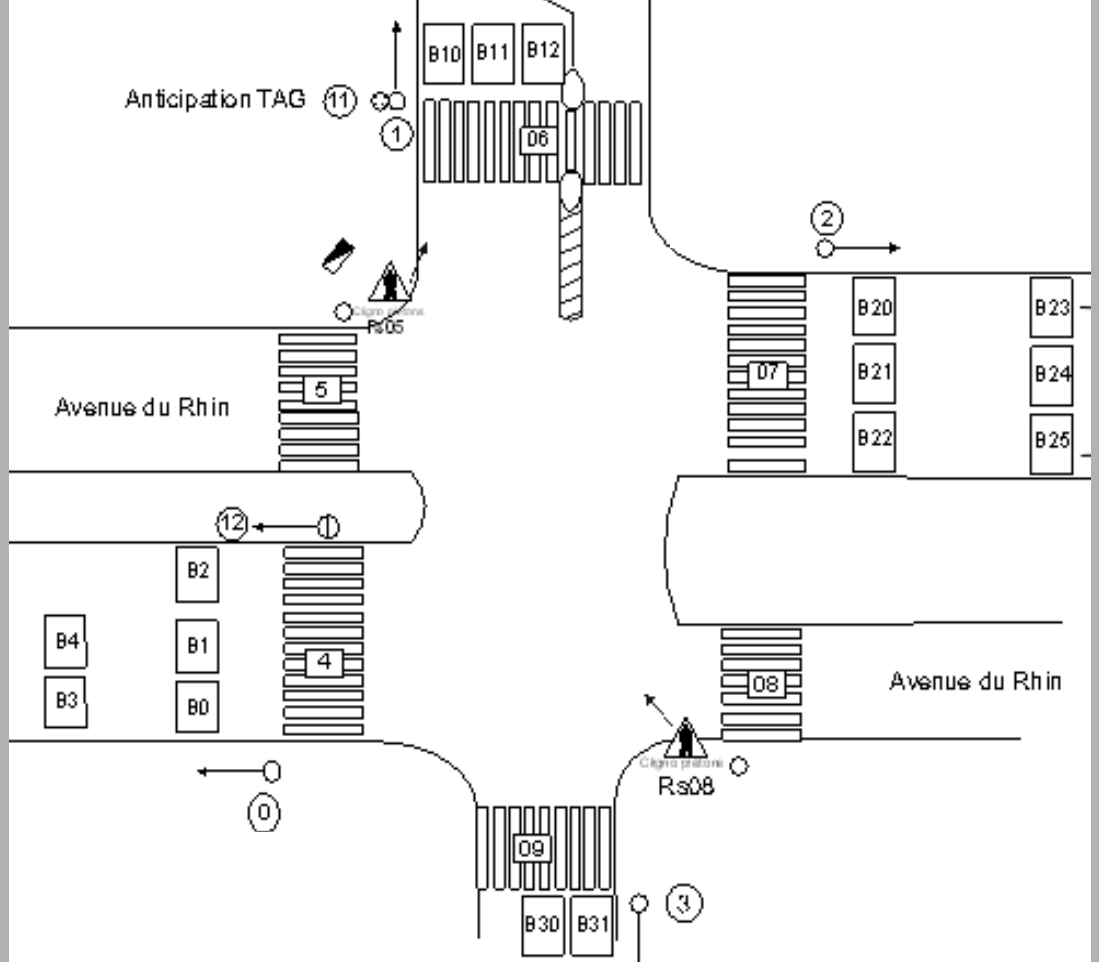
- Routes infiniment longues arrivant vers le carrefour
- Conditions météorologiques constantes le temps de l'étude
- Non prise en compte des piétons et des cyclistes
- Véhicules identiques
- Conducteur possédant un temps de réaction nul et respectant parfaitement le code de la route

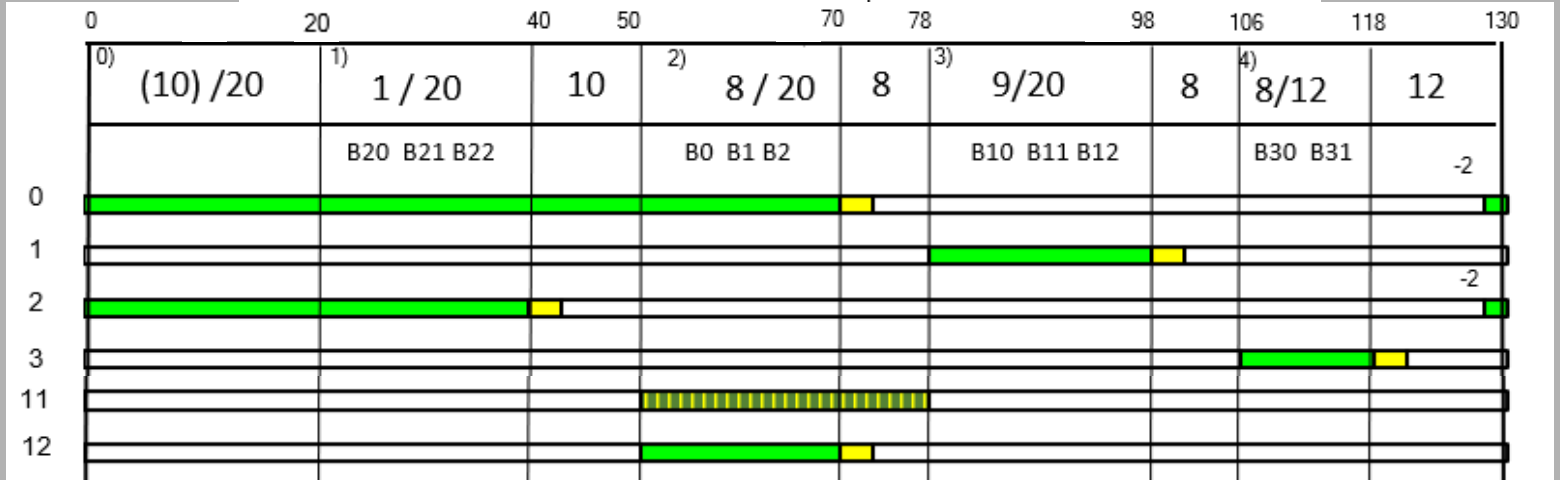
RÉALITÉ

SIRAC

Service de l'Information et de la Régulation Automatique de la Circulation

Photo 3D provenant de Google Earth


Anticipation TAG
11
1
B10
B11
B12
06
2
B20
B21
B22
B23
B24
B25
07
Rs05
Avenue du Rhin
5
12
B2
B4
B1
B3
B0
4
08
Avenue du Rhin
Rs08
0
09
B30
B31
3


0
20
40
50
70
78
98
106
118
130
0) (10) /20
1) 1 / 20
10
2) 8 / 20
8
3) 9/20
8
4) 8/12
12
B20 B21 B22
B0 B1 B2
B10 B11 B12
B30 B31
-2
-2
0
1
2
3
11
12

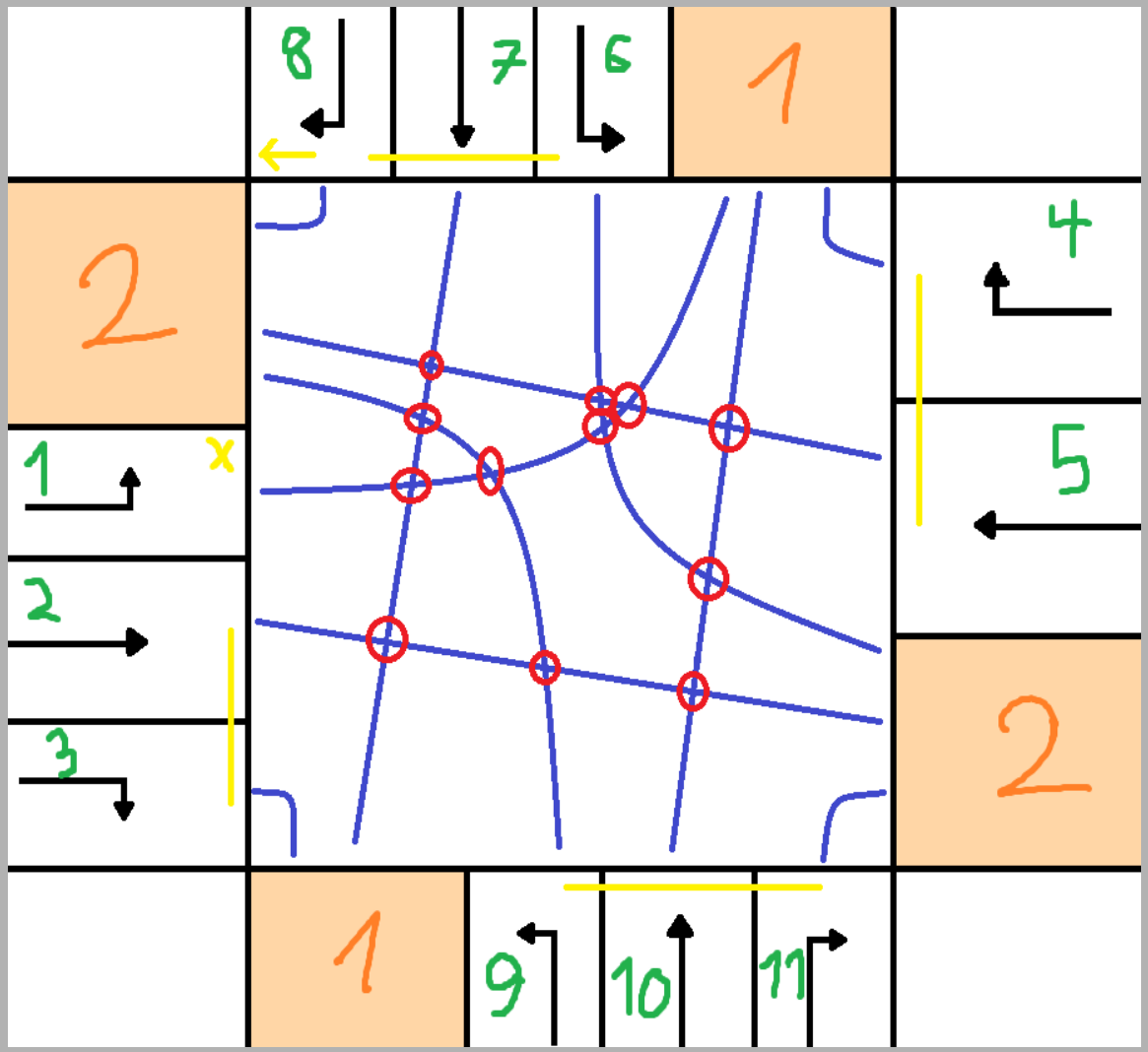
CARACTÉRISTIQUES DU MODÈLE

Voies :

- prioritaires : 1,2, 3, et 8
- secondaires : 4, 5, 6, 7, 8, 10 et 11

GFC :

- 2,3,4,5
- 1,2,3,8
- 6,7,8
- 9,10,11



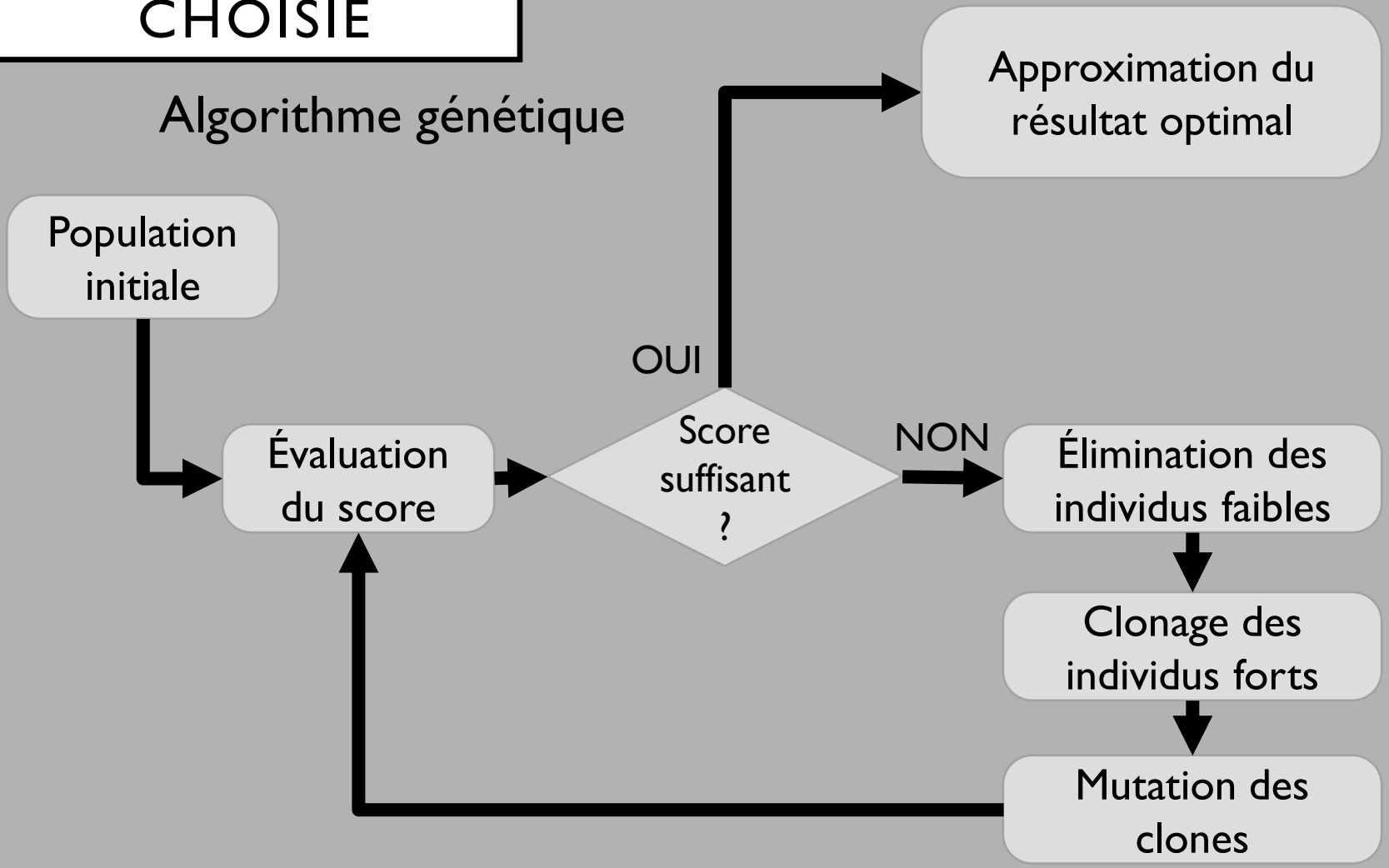
PREMIÈRE APPROCHE

Intelligence Artificielle :

- Grande flexibilité d'étude
  - Spatiale
  - Hypothèses
- Complexe à entrainer
  - Entrées compliquées
  - Temps de calcul long pour une réponse optimale

MÉTHODE CHOISIE

Algorithme génétique



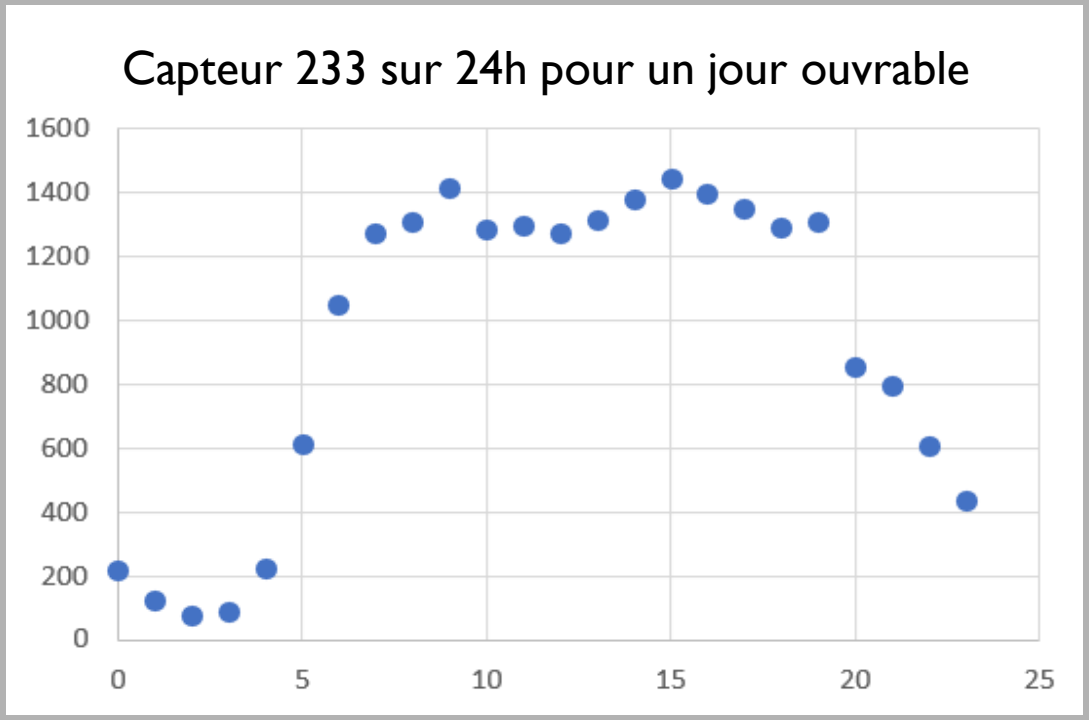
MÉTHODE CHOISIE

Vocabulaire

- Individus
- Population
- Génération
- Score
- Croisement/ Clonage
- Mutation

```
init = [130,
        ['0',[1,2],1,[128,72]],
        ['1',[3],1,[78,20]],
        ['2',[1],1,[128,42]],
        ['3',[4],1,[106,12]],
        ['4',[2,3],1,[50,28]],
        ['5',[2],1,[50,20]]]
```

TESTS ET RÉSULTATS



TESTS ET RÉSULTATS

Avec 1h d'enregistrement, de16h30 à 17h30, j'ai pu obtenir les valeurs moyennes suivantes :

- Est ➔ Nord : 4 (4)
- Est ➔ Ouest : 20 (5,6)

- Ouest ➔ Nord : 11 (1)
- Ouest ➔ Est : 28 (2,3)
- Ouest ➔ Sud : 2 (3)

- Nord ➔ Sud : 4 (8)
- Nord ➔ Est : 6 (7,8)
- Nord ➔ Ouest : 8 (9)

- Sud ➔ Nord : 4 (11)
- Sud ➔ Ouest : 4 (10)
- Sud ➔ Est : 1 (11)

TESTS ET RÉSULTATS

Merci pour votre attention !

- Fei Yan. Contribution à la modélisation et à la régulation du trafic aux intersections : intégration des communications Vehicule-Infrastructure. Ordinateur et société [cs.CY]. Université de Technologie de Belfort-Montbeliard, 2012. Français. NNT : 2012BELF0175. tel-00720641 (Thèse)
- http://dspace.univ-guelma.dz/jspui/handle/123456789/13285 (Mémoire)
- Guillaume Costeseque. Modélisation et simulation dans le contexte du trafic routier. Varenne F. and Silberstein M. Modéliser et simuler. Epistémologies et pratiques de la modélisation et de la simulation, Editions Matériologiques, 2013. hal-00965010 (Mémoire)
- Sébastien Faye, Claude Chaudet, Isabelle Demeure. Contrôle du trafic routier urbain par un réseau fixe de capteurs sans fil. 2012. hal-00781140 (Rapport technique)
- https://www.youtube.com/watch?v=CQu4wFLC79U (vidéo)
- Concours commun Mines Ponts ; épreuve d'informatique commune ; 2017 (sujet concours)

```python
from random import randint
import numpy as np

#####################
#VARIABLE NECESSAIRE#
#####################

init = [130,
        ['0',[1,2],1,[128,72]],
        ['1',[3],1,[78,20]],
        ['2',[1],1,[128,42]],
        ['3',[4],1,[106,12]],
        ['4',[2,3],1,[50,28]],
        ['5',[2],1,[50,20]]]

tableau = [['4',['233,0',161,89,57,66,164,454,776,941,969,1044,950,957,941,971,1019,1066,1032,997,956,965,633,587,448,323],
                ['233 1',56,31,20,23,57,159,273,330,340,367,334,336,331,341,358,374,362,350,336,339,223,206,158,114],
                ['234',209,144,92,90,82,146,353,566,523,643,803,837,698,856,868,882,741,617,725,784,590,538,454,352],
                ['240 0',30,15,10,10,17,38,75,220,328,179,183,229,228,231,235,265,269,277,266,222,127,112,106,73],
                ['240 1',27,13,9,9,16,35,69,204,303,166,169,211,211,213,216,245,248,256,246,204,118,103,97,68]],
           ['6',['233 0',186,94,54,79,161,423,738,1070,929,1043,1100,997,1137,1138,1099,1072,1146,1155,1092,1075,762,648,568,489],
                ['233 1',66,33,19,28,56,149,259,376,327,366,387,350,399,400,386,377,403,406,384,378,268,228,199,172],
                ['234',273,165,110,115,77,147,370,603,660,720,891,802,773,947,716,648,692,608,622,846,697,636,615,531],
                ['240 0',46,14,9,11,21,41,87,212,308,209,201,223,252,246,275,284,339,329,298,235,138,124,91,97],
                ['240 1',43,12,8,10,20,37,80,195,284,192,185,205,232,228,254,263,313,303,276,217,127,115,84,90]]]

capteurs = ['233','234','240']

```

```
################
#CODE PREALABLE#
################

def valeur_tab(tab,date,cap,heure,direction):
    k=0
    if date =='4':
        i=0
    elif date=='6':
        i=1
    if cap=='233':
        j=1
        if direction==0 :
            k=0
        else:
            k=1
    elif cap=='234':
        j=3
        k=0
    elif cap=='240':
        j=4
        if direction==0 :
            k=0
        else:
            k=1
    elif cap=='sud' :
        j=6
    return tab[i][j+k][heure+1]
```

```
def sort_1(L,i,p):
    rg = 0
    while i != len(L):
        mini=np.inf
        cran = []
        for j in range(i,len(L)):
            if L[j][p]<mini:
                mini=L[j][p]
                rg=j
        cran = L[rg]
        L.pop(rg)
        L.insert(i,cran)
        i+=1
    return L

def tri_1(L):
    n=len(L)
    trie = []
    r = []
    rg = 0
    backup=[]
    backup.extend(L)
    while len(trie)<n:
        maxi = -np.inf
        for i in range(len(L)):
            a = L[i][0]
            b = L[i][1]
            if (a+b)/2>maxi:
                maxi = (a+b)/2
                r=L[i]
                rg = i
        trie.append(r)
        L.pop(rg)
    L.extend(backup)
    return trie
```

```
def tri_unique(L):
    temps = 0
    while temps<len(L):
        a = L[temps]
        for i in range(len(L)-1,temps,-1):
            if L[i]==a:
                L.pop(i)
        temps+=1
    return L


def en_commun(a,b):
    commun = False
    for x in range(len(a)):
        for y in range(len(b)):
            if a[x]==b[y]:
                commun = True
                return commun
    return commun


def p_233():
    direction=0
    pourcentage = randint(1,100)
    if pourcentage<=26:
        direction = 1
    return direction

def p_240():
    direction=0
    pourcentage = randint(1,100)
    if pourcentage<=48:
        direction = 1
    return direction
```

```python
def creation_pop(n):
    population=[]                                   #initialisation de la population
    #creation d'un nombre n d'individu
    for _ in range(n):
        ind=[]                                      #initialisation de l'individu
        taille = randint(12,300)                    #temps totale de la boucle
        ind.append(taille)                          #ajout du temps total de la boucle à l'individu
        for i in range(6):                          #caracterisation des 6 feux
            feux = [str(i)]                         #nom des feux
            if i==0:                                #affectation de chaque feux à son GFC correspondant
                feux.append([1,2])
            elif i==1:
                feux.append([3])
            elif i==2:
                feux.append([1])
            elif i==3:
                feux.append([4])
            elif i==4:
                feux.append([2,3])
            elif i==5:
                feux.append([2])
            dt=0
            passage=0
            while dt<taille:                        #caracterisation de chaque passage
                debut = randint(dt,taille)
                temps = randint(0,taille-dt)#temps du passage actuel ne pouvant pas depasser une certaine valeur
                passa = [debut,temps]               #caracteriqtique de chaque passage
                feux.append(passa)
                dt=debut+temps+2
                passage+=1
            feux.insert(2,passage)
            ind.append(feux)                        #ajout de chaque feux à l'individu
        population.append(ind)                      #ajout de chaque individu à la population
    return population
```

```
"""
Conditions de viabilité :
    0 le temps de passage au vert doit être inférieur au temps total de la boucle
    1 le temps de passage au vert doit être supérieur à 1 seconde
    2 deux passage au vert ne peuvent pas se lancer simultanément
    3 tous les feux doivent s'allumer au moins une fois à part 4 qui peut déroger à cette règle
    4 deux passage ne peuvent pas se superposer sur le retour de la boucle
    5 si '4' et '1' sont allumés en même temps, l'allumage de '4' est inutile

    deux feux de GFC différents ne peuvent être allumés simultanément
    les deux secondes de sécurité obligatoire sont respecté pour tous les feux
"""
```

```python
def verif_pop(population):
    #                 verification de la viabilite de l'indivu
    #condition zero
    for indi in range(len(population)-1,-1,-1):             #pour tous les individus, on verifie que
        boucle = population[indi][0]
        for i in range(1,7):                            #tous les feux soient bien valides
            temps = 0
            for j in range(population[indi][i][2]):         #temps total pour tous les passages
                temps += population[indi][i][j+3][1]
            if temps > boucle :
                population.pop(indi)
                break

    #première condition
    for indi in range(len(population)):                      #pour tous les individus de la population
        for i in range(1,7):                            #pour tous les feux de l'individu
            k=0
            for j in range(population[indi][i][2]-1,-1,-1):  #pour tous les passages au vert du feux
                if population[indi][i][j+3][1]==0:           #On verifie la pertinence du passage au vert (temps de vert nul)
                    population[indi][i].pop(j+3)
                    k+=1
            population[indi][i][2]-=k

    #deuxième condition
    for indi in range(len(population)):                      #pour tous les individus de la population
        for i in range(1,7):                            #pour tous les feux de l'individu
            k=0
            for j in range(population[indi][i][2]-1,0,-1):  #pour tous les passages au vert du feux
                if population[indi][i][j+3][0]==population[indi][i][j+2][0]: #le moment de commencement de deux passage consecutifs
                    population[indi][i].pop(j+3)
                    k+=1
            population[indi][i][2]-=k

    #troisième condition
    for indi in range(len(population)-1,-1,-1):              #pour tous les individus de la population
        for i in range(1,7):                            #pour tous les feux de l'individu
            if population[indi][i][2]==0:
                population.pop(indi)
                break
```

Suite de la fonction à suivre ➔

```
#quatrième condition
for indi in range(len(population)-1,-1,-1):              #pour tous les individus de la population
    for i in range(1,7):                            #pour tous les feux de l'individu
        if population[indi][i][2]==0:
            population.pop(indi)
            break
        retour = int(population[indi][i][-1][1])+int(population[indi][i][-1][0])-int(population[indi][0])
        if retour>=population[indi][i][3][0]:
            population[indi][i].pop(-1)
            population[indi][i][2]-=1
            if population[indi][i][2]==0:
                population.pop(indi)
                break
#cinquième condition
for indi in range(len(population)-1,-1,-1):              #pour tous les individus de la population
    #print(' ',population[indi],i)
    for i in range(len(population[indi][5])-3-1,-1,-1):
        for j in range(population[indi][2][2]-1,-1,-1):
            debut_4 = population[indi][5][3+i][0]
            fin_4 = debut_4+population[indi][5][3+i][1]
            debut_1 = population[indi][2][3+j][0]
            fin_1 = debut_1+population[indi][2][3+j][1]
            if debut_4<debut_1 and debut_1<fin_4:          #mets les feux 4 et 1 à la suite dans cet ordre
                population[indi][5][3+i][1]=debut_1-debut_4
            elif debut_4<debut_1 and fin_1<fin_4 :         #enlève la periode d'activation de 4 lorque 1 l'est aussi
                fin_n = debut_1-debut_4                    #et commence après 4
                debut_n = fin_1
                if debut_n > population[indi][0]:
                    debut_n -= population[indi][0]
                population[indi][5].pop(3+i)
                population.insert(3+i,[debut_4,fin_n])
                population.insert(3+i+1,[debut_n,fin_4-debut_n])
                population[indi][5][2]+=1
            elif debut_1<debut_4 and debut_4<fin_1:        #mets les feux 1 et 4 à la suite dans cet ordre
                if fin_1 > population[indi][0]:
                    fin_1 -= population[indi][0]
                population[indi][5][3+i][0]=fin_1
            elif debut_1<=debut_4 :                        #enlève la periode d'activation de 4 lorque 1 l'est aussi
                if fin_4<=fin_1:                           #et commence avant 4
                    population[indi][5].pop(3+i)
                    population[indi][5][2]-=1
                    break
                else :
                    if fin_1 > population[indi][0]:
                        fin_1 -= population[indi][0]
                    population[indi][5][3+i][0]=fin_1
```

Suite de la fonction à suivre ➔

```
        #remaniement de l'individu pour le faire rentrer dans les conditions donnees prealablement
        for i in range(1,7):
            k=0
            for j in range(len(population[indi][i])-3-1,0,-1):  #pour tous les passages au vert du feux
                if population[indi][i][j+3][0]==population[indi][i][j+2][0]: #le moment de commencement de deux passage consecutifs
                    population[indi][i].pop(j+3)
                    k+=1
            population[indi][i][2]-=k
        for i in range(1,7):
            sort_l(population[indi][i],3,0)
        for i in range(1,7):                            #pour tous les feux de l'individu
            if population[indi][i][2]==0:
                population.pop(indi)
                break

    #condition externe
    for indi in range(len(population)-1,-1,-1):
        if population[indi][0]<12:
            population.pop(indi)
        for i in range(1,7):
            if indi>=len(population):
                break
            for j in range(population[indi][i][2]):
                if population[indi][i][3+j][0]<0 or population[indi][i][3+j][1]<0:
                    population.pop(indi)
                    break
    return population
```

Fin de la fonction verif_pop()

```
###############
#CODE MUTATION#
###############

def mutation(population):
    for indi in range(len(population)):              #pour tous les individus de population
        mutation_b = randint(-10,10)                #mutations pour le temps de la boucle
        population[indi][0]+=mutation_b                   #mutation du temps de la boucle
        for i in range(1,7):                        #pour tous les feux de l'individu
            for j in range(population[indi][i][2]):       #pour tous les passages du feux
                for t in range(2):
                    mutation_f = randint(-5,5)
                    population[indi][i][3+j][t]+=mutation_f
    verif_pop(population)                                      #verification de la viabilite des passages
    return population


def clonage_mutation(population,n):
    print('clonage et mutation')
    population_clone = []
    population_clone.extend(population)
    clones = []
    for _ in range(3):
        for i in range(len(population)):
            clones.append(population[i])
        mutation(clones)
        population_clone.extend(clones)
    return population_clone

##############
```

```
#############
#CODES SCORE#
#############


def direct(cap):
    if cap=='233':
        direction = p_233()
    elif cap=='240':
        direction = p_240()
    return direction


def nmb_boucle(individu):
    temps = individu[0]
    nombre = int(3600//temps)
    return nombre


def nmb_vehicule_b(individu,date,cap,heure,direction):
    n = nmb_boucle(individu)
    valeur = valeur_tab(tableau,date,cap,heure,direction)
    return int(valeur//n)


def nmb_vehicule_s(individu,date,cap,heure,direction):
    nmb = nmb_vehicule_b(individu,date,cap,heure,direction)
    temps_boucle = individu[0]
    return round((nmb/temps_boucle),3)
```

```python
def score1(population,date,heure):
    score = []
    for indi in range(len(population)-1,-1,-1):           #pour tous les individus de la population
        boucle = population[indi][0]
        score_indi=0
        for cap in capteurs:                        #pour toutes les voies du carrefour
            score_f=0
            if cap == '233' :
                direction = p_233()
                if direction == 0 :
                    j=1
                else :
                    j=6
            elif cap=='240' :
                direction = p_240()
                if direction == 0 :
                    j=2
                else :
                    j=5
            elif cap=='234':
                j=3
            else :
                j=4
            vehicule_s = nmb_vehicule_s(population[indi], date, cap, heure, direction)
            for k in range(population[indi][j][2]):  #pour tous les passages de chaque feu
                temps = boucle-population[indi][j][3+k][1]
                vehicules = vehicule_s
                sec = 0
                score_p = 0
                while sec<=temps:                   #tant que le nombre de secondes attendu est inferieur au temps de feu rouge
                    score_p+=vehicules
                    vehicules+=vehicule_s
                    sec+=1
                score_f+=score_p
            if population[indi][j][2]==0:
                population.pop(indi)
                break
            score_f/=population[indi][j][2]
            score_indi+=score_f
        score_indi/=7
        if score_indi !=0 :
            score.append(1/score_indi)
        else :
            score.append(0)
    return score
```

```python
def score2(population):
    score=[]
    for indi in range(len(population)):              #pour tous les individus de la population
        score_indi=0
        for i in range(1,7):                          #pour tous les feux de l'individu
            for j in range(population[indi][i][2]):
                test = population[indi][i][3+j][0]+population[indi][i][3+j][1] #temps de passage au vert du feux test
                if test<=population[indi][0]:
                    for k in range(1,7):              #indice du feux a  tester
                        if en_commun(population[indi][i][1],population[indi][k][1])==False:
                            for t in range(population[indi][k][2]): #indice du passage du feu a  tester
                                if population[indi][i][3+j][0]<population[indi][k][3+t][0] and population[indi][k][3+t][0]<=test:
                                    score_indi+=1
                elif test>population[indi][0]:
                    test-=population[indi][0]
                    for k in range(1,7):              #indice du feux a  tester
                        if en_commun(population[indi][i][1],population[indi][k][1])==False:
                            for t in range(population[indi][k][2]): #indice du passage du feu a  tester
                                if population[indi][k][3+t][0]>0 and population[indi][k][3+t][0]<=test or population[indi][k][3+t][0]>population[indi][i][3+j][0]:
                                    score_indi+=1
        if score_indi !=0 :
            score.append(1/score_indi)
        else :
            score.append(100)
    return score
```

```
################
#CODE SELECTION#
################

def selection(population,score,nombre,date,heure):
    print('selection')
    score_trie = tri_l(score)
    print(score_trie,nombre)
    selection_score = [score_trie[i] for i in range(nombre)]
    for j in range(1,nombre+1):
        pourcentage = randint(1,100)
        if pourcentage<=5 :
            selection_score.append(score_trie[-j])
    nouvelle_population = []

    for i in range(len(selection_score)):
        indi = selection_score[i][2]
        nouvelle_population.append(population[indi])

    crash_nul = sort_l(score_trie,0,1)
    sco = [crash_nul[i] for i in range(3)]
    for i in range(3):
        nouvelle_population.append(population[sco[i][2]])

    return nouvelle_population
```

```
###############
##CODE DE FIN#
###############


def initial_r(n):
    population_total=[]
    #On boucle afin d'obtenir une population d'au moins n individus
    while len(population_total)<=n:
        population=creation_pop(n)
        verif_pop(population)
        population_total.extend(population)
    return population_total


def score_pop(population,date,heure):
    print('score')
    score_f = []
    s1 = score1(population,date,heure)
    s2 = score2(population)
    for indi in range(len(population)):
        score_f.append([s1[indi],s2[indi],indi])
    return score_f
```

```python
def final(n_indi,date,heure,score_mini,nombre_s):
    population = initial_r(n_indi)
    sco = score_pop(population,date,heure)
    sco_t = tri_l(sco)
    i=0
    T=False
    while sco_t[0][1]!=0 and (sco_t[0][0]+sco_t[0][1])/2 < score_mini and i<1000 :
        selec = selection(population, sco_t, nombre_s, date, heure)
        clo = clonage_mutation(selec, n_indi)
        clo_v = verif_pop(clo)
        if clo_v ==[] :
            T=True
            break
        population = clo_v
        sco = score_pop(population,date,heure)
        sco_t = tri_l(sco)
        i+=1

    if T==False:
        return population[sco_t[0][2]]
    else :
        print('Recommencer, il y a eu une erreur dans la verification de la population')
```