

Problème du transport à la demande ou Dial A Ride Problem (DARP)

TRIOULEYRE-ROBERJOT Louis

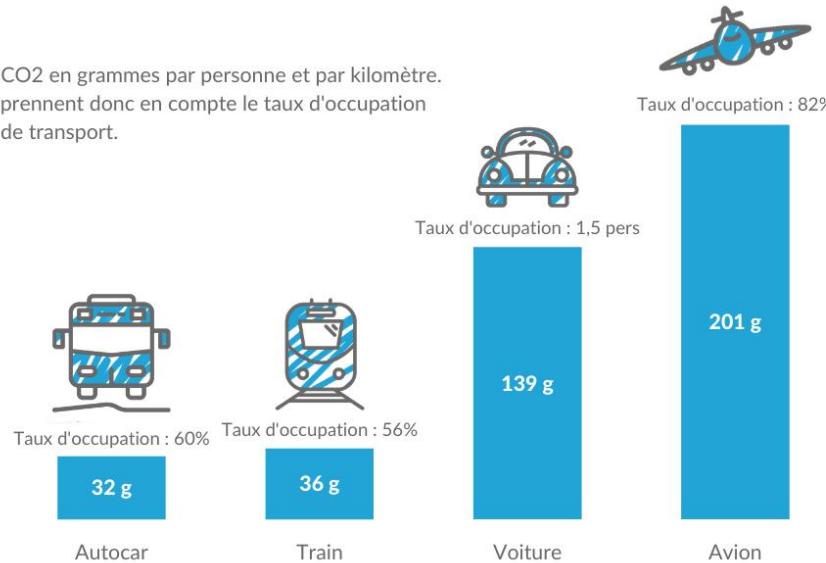
n° de candidat : 36417

Les enjeux

ÉMISSIONS DE CO2 ET MOYENS DE TRANSPORT



Émission de CO2 en grammes par personne et par kilomètre.
Ces chiffres prennent donc en compte le taux d'occupation
des moyens de transport.



Source : Agence fédérale allemande pour l'environnement (UBA)

Réduire les coûts et l'empreinte carbone en :

- incitant les gens à prendre davantage l'autocars
 - diminuer le prix
 - étendre le réseau
- parcourant moins de distance tout en conservant son objectif
 - le rendre plus efficace

Les objectifs du DARP

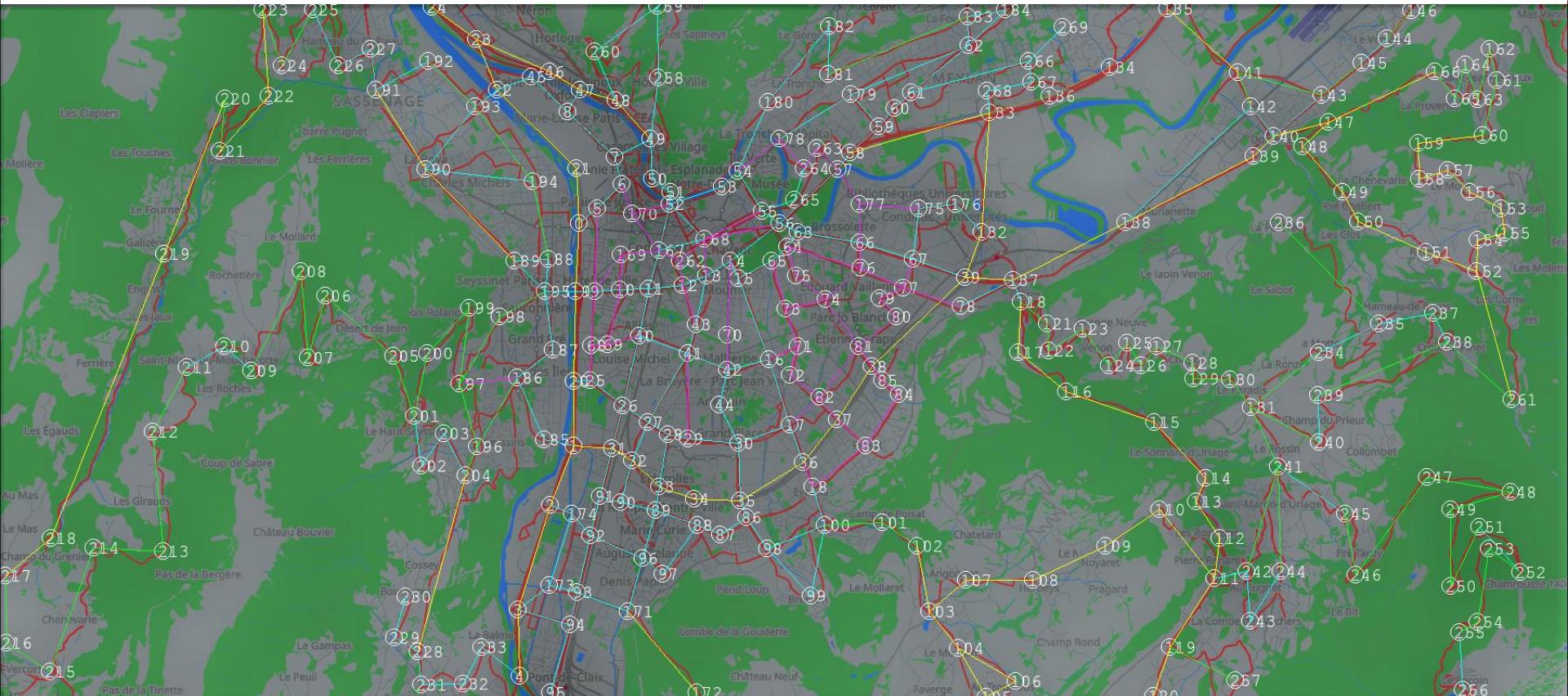
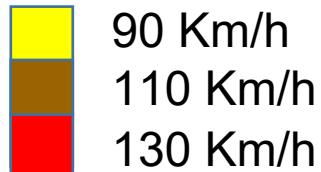
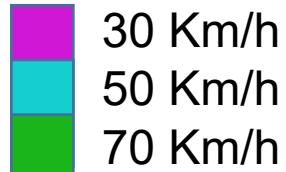
Faire en sorte que les bus :

- Circulent pleins en parcourant le moins de distance possible
- Satisfassent les demandes des **n** passagers le plus efficacement possible

Sommaire

1. Modélisation
2. DARP avec 1 bus
3. DARP avec **s** bus et algorithme de recuit simulé

1- Modélisation

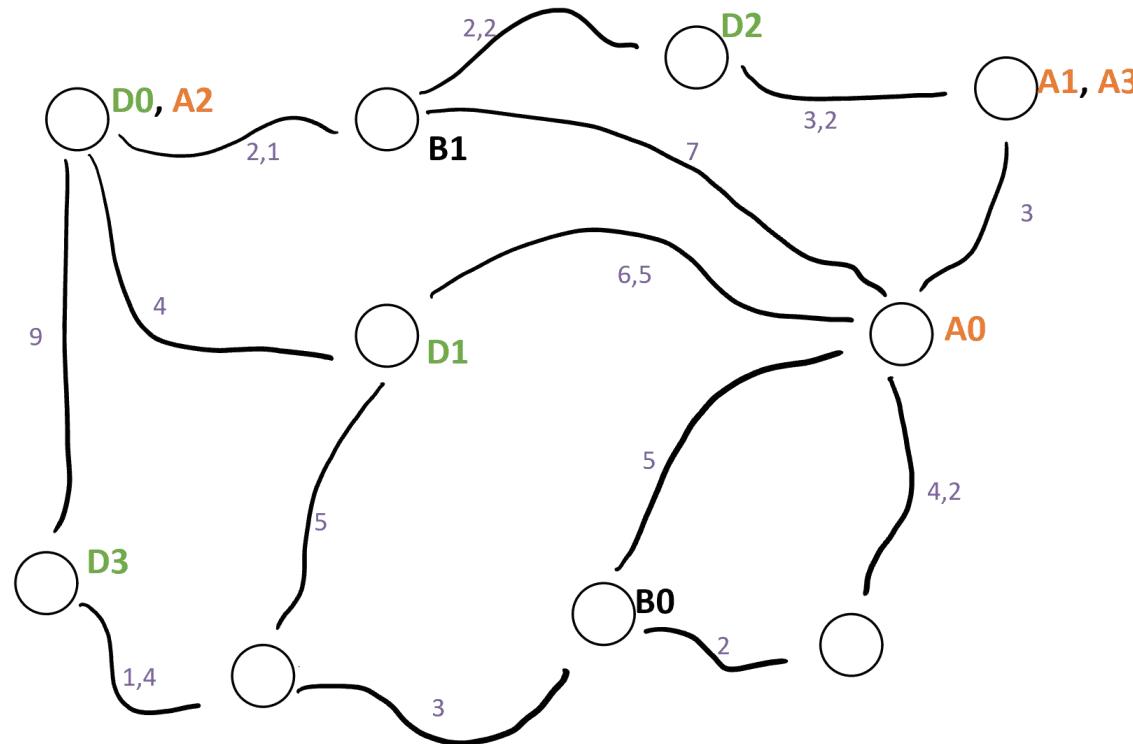


Réseau routier de la ville de Grenoble

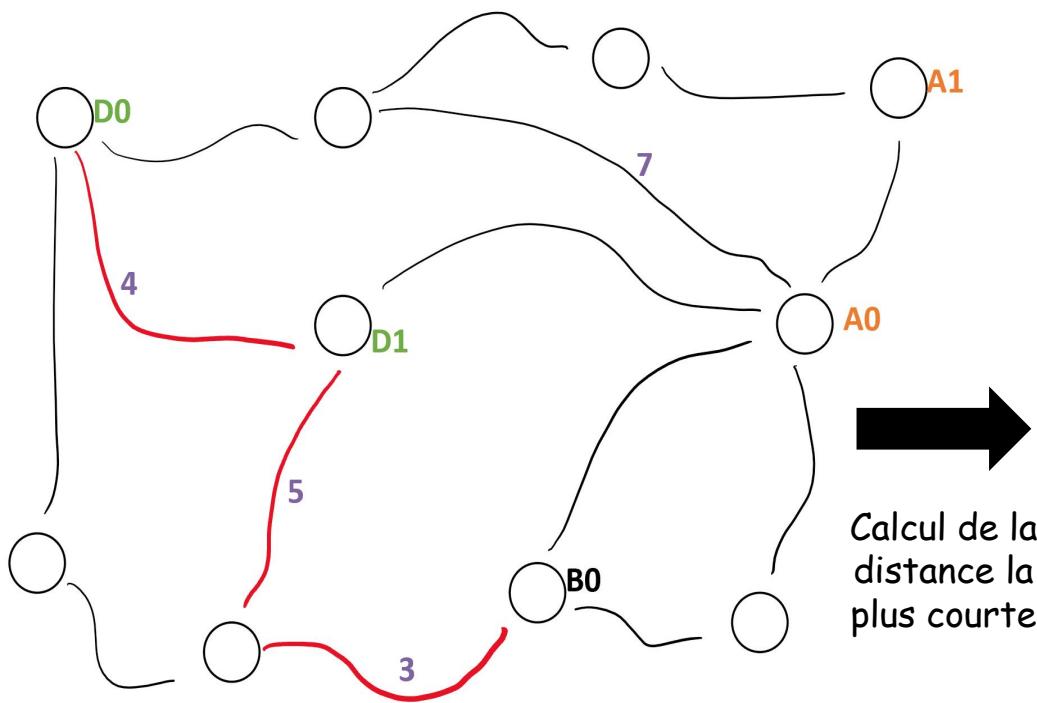
DEF : Temps entre deux sommets A et B : Temps pour aller de A à B à la vitesse limite

On modélise notre problème par :

- Un graphe connexe, non orienté et pondéré par le temps
- Un ensemble de passagers modélisé par leur sommet de départ et d'arrivée
- Un ensemble de bus modélisé par leur sommet d'origine

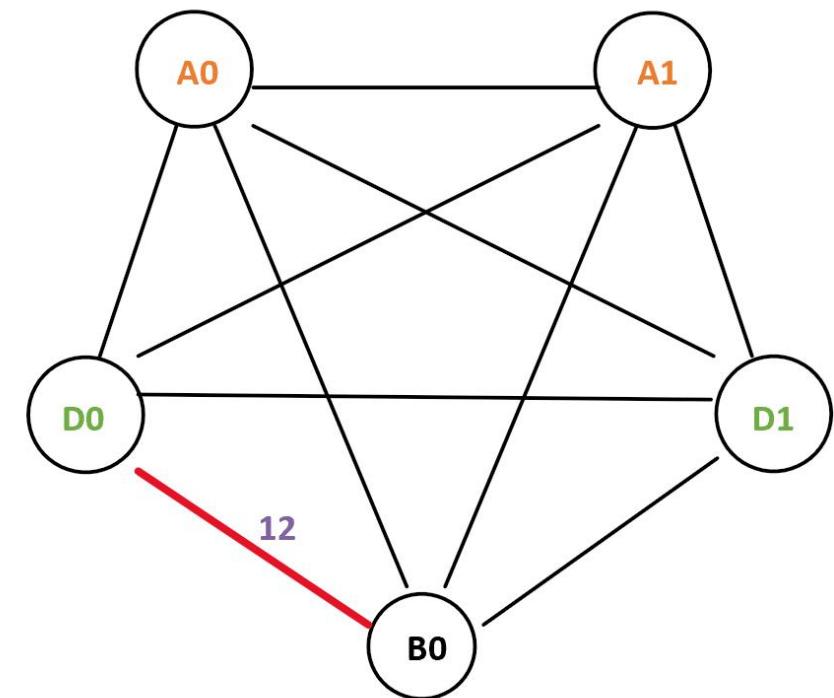


Modélisation d'une ligne de bus



Graphe initial

Calcul de la
distance la
plus courte



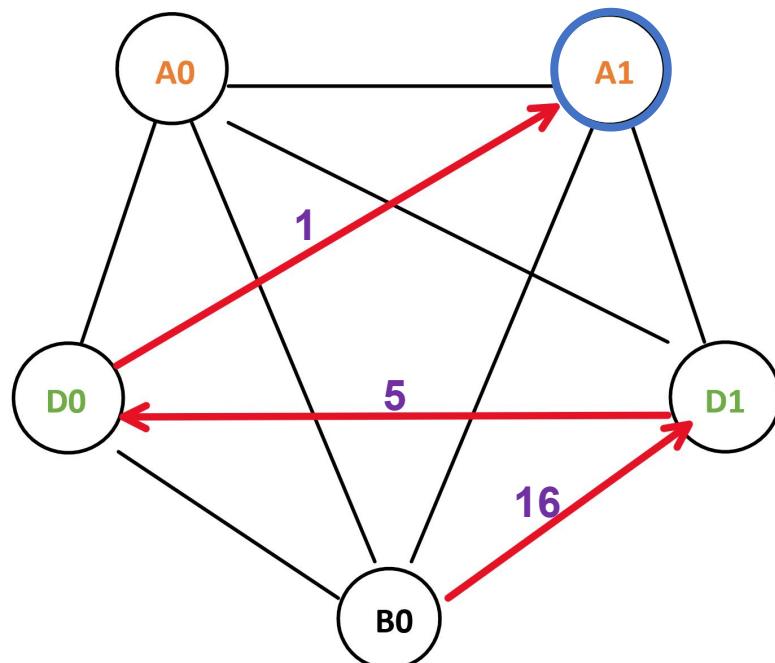
Graphe complet des distances
minimales entre points d'intérêt

Modélisation d'une ligne de bus

Tournée d'un bus : tout chemin dans le **graphe des distances**, d'origine le sommet «initial» du **bus**, se terminant en un point d'intérêt **y**, tel que pour tout passager x , Dx avant Ax

Longueur d'une tournée : le nombre d'arêtes de la tournée

Poids d'une tournée : la somme des **poids** des arêtes

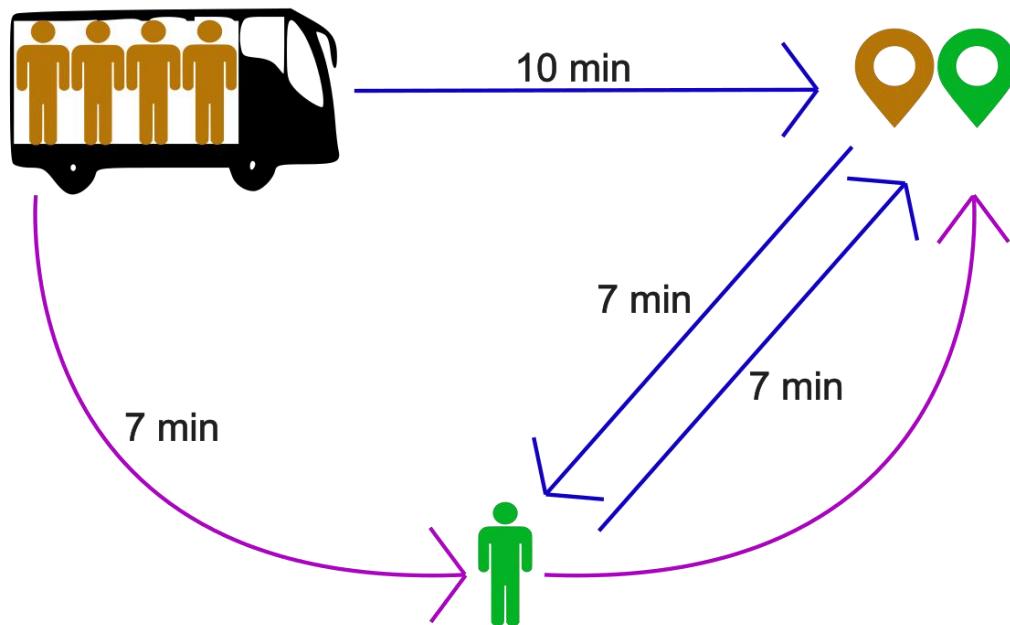


tournée $B0 \rightarrow A1$: **longueur** : 3

graphe des distances

Critère d'optimisation individuel :

$$\text{coût}_{\text{client}} = \sum_{\text{client}} \text{temps d'attente} + \text{temps de trajet du client}$$

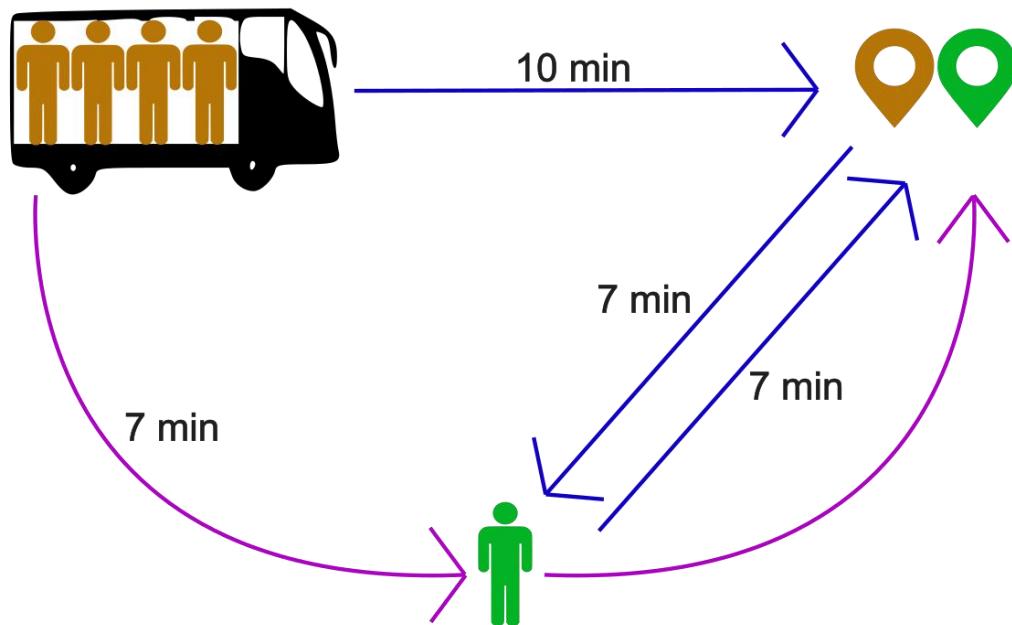


$$\text{coût}_{\text{client}} = 10 \times 4 + 24 = 64$$

$$\text{coût}_{\text{client}} = 14 \times 4 + 14 = 70$$

Critère d'optimisation collectif :

$\text{coût}_{\text{tournée}} = \text{poids de la tournée}$



$$\text{coût}_{\text{tournée}} = 10 + 14 = 24$$

$$\text{coût}_{\text{tournée}} = 14$$

Meilleure tournée pour le critère coût_{tournée} de façon naïve

Le nombre de tournées à comparer serait de :

$$\underline{2^n(n!)^2}$$

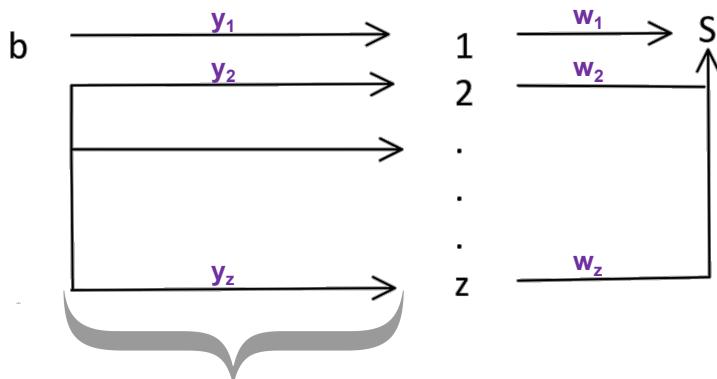
Meilleure tournée pour le critère coût_{tournée} de façon optimisée

On découpe le problème par la longueur de la tournée.

On note : **b** le sommet de départ du bus

Objectif : Pour tout sommet **s** déterminer une **tournée** de poids minimal de longueur **k** jusqu'au sommet **s**

Pour **k=1** : Toute **tournée** de longueur 1 est de poids minimale



Donc la **tournée** de poids minimal de longueur **k+1** jusqu'à **s** est :

$$\min_{1 \leq i \leq z} (y_i + w_i)$$

Plus courte tournée de longueur **k**

Pour $k=1$:

- $B_0 \rightarrow D_0 : 15$
- $B_0 \rightarrow D_1 : 45$

Pour $k=2$:

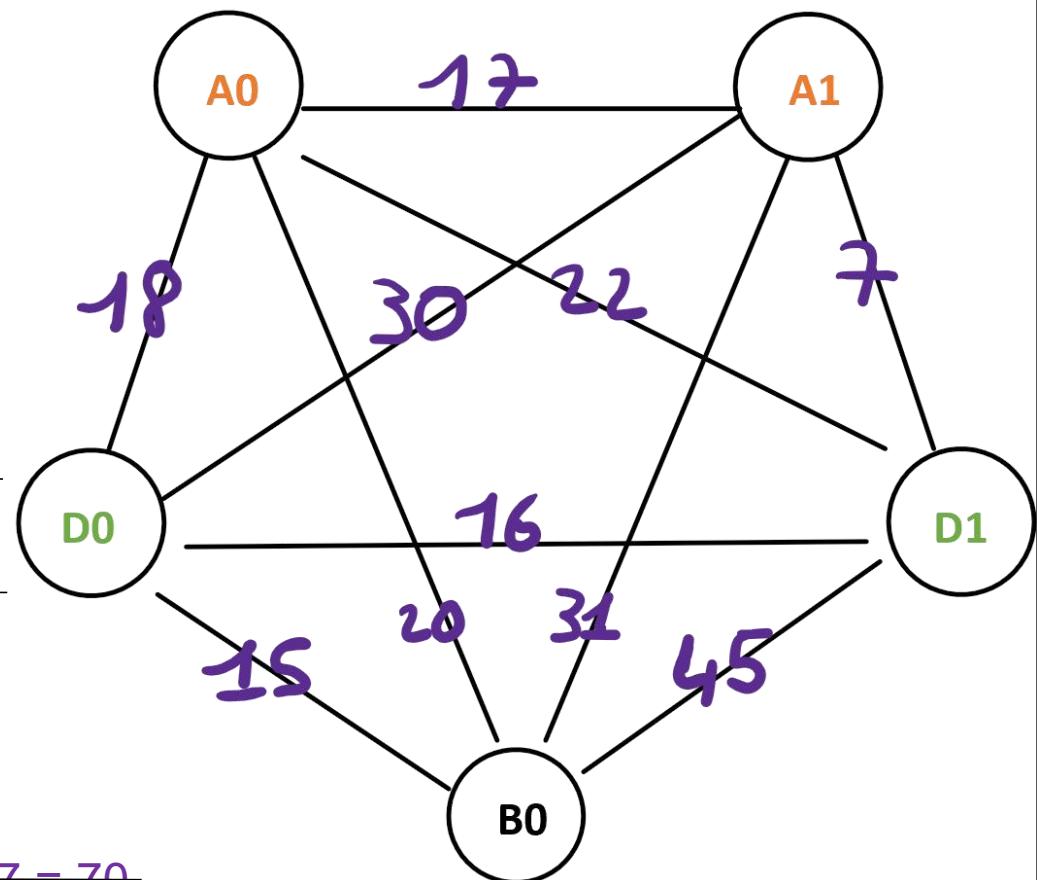
- $B_0 \rightarrow D_0 \rightarrow D_1 : 15 + 16 = 31$
- $B_0 \rightarrow D_0 \rightarrow A_0 : 15 + 18 = 33$
- $B_0 \rightarrow D_1 \rightarrow D_0 : 45 + 16 = 61$
- $B_0 \rightarrow D_1 \rightarrow A_1 : 45 + 7 = 52$

Pour $k=3$:

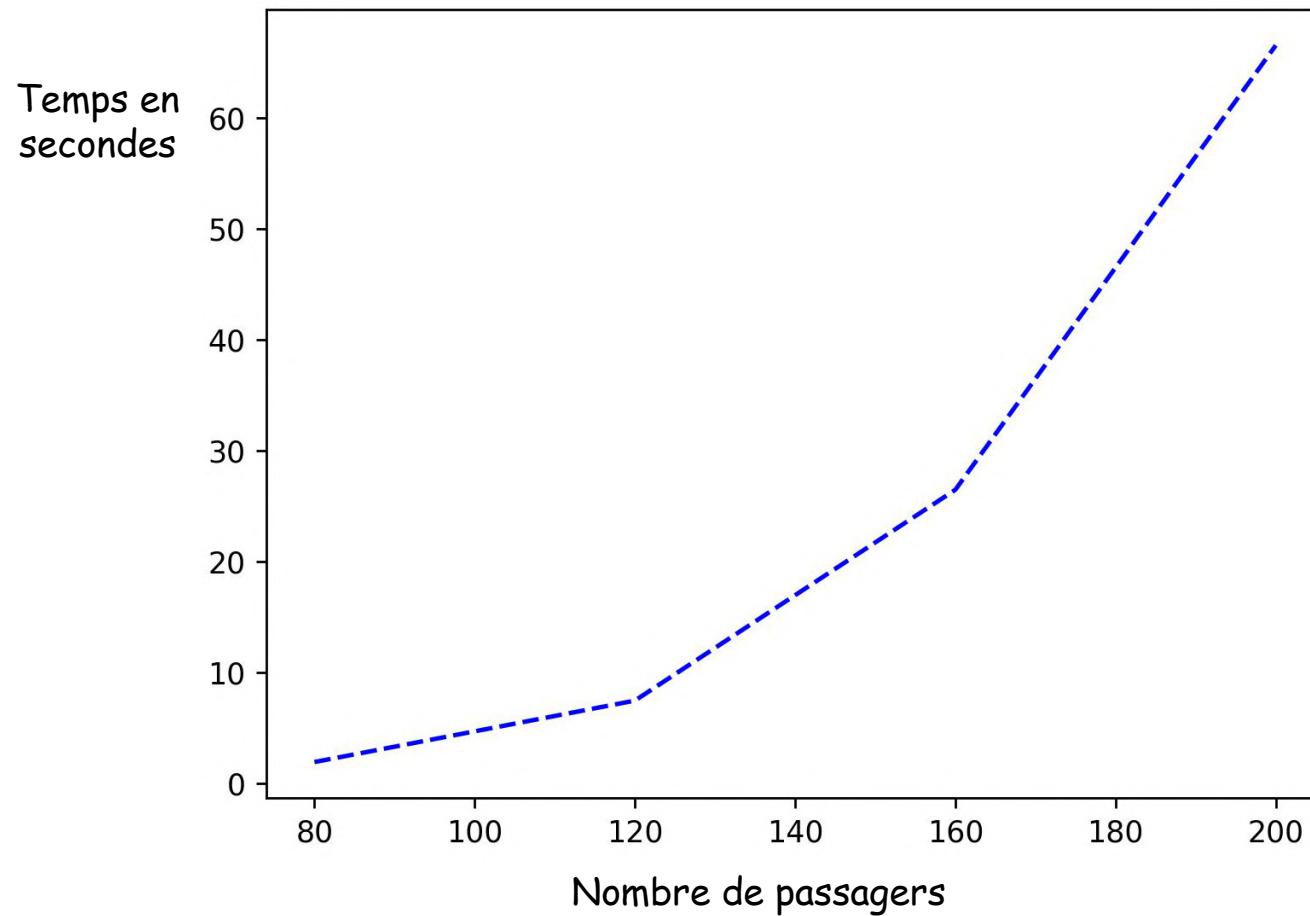
- $B_0 \rightarrow D_0 \rightarrow D_1 \rightarrow A_0 : 31 + 22 = 53$
 $\searrow D_1 \rightarrow D_0 \nearrow : 61 + 18 = 79$
- $B_0 \rightarrow D_0 \rightarrow D_1 \rightarrow A_1 : 31 + 7 = 38$
 $\searrow D_1 \rightarrow D_0 \nearrow : 61 + 30 = 91$
- $B_0 \rightarrow D_1 \rightarrow A_1 \rightarrow D_0 : 52 + 30 = 82$
- $B_0 \rightarrow D_0 \rightarrow A_0 \rightarrow D_1 : 33 + 22 = 55$

Pour $k=4$:

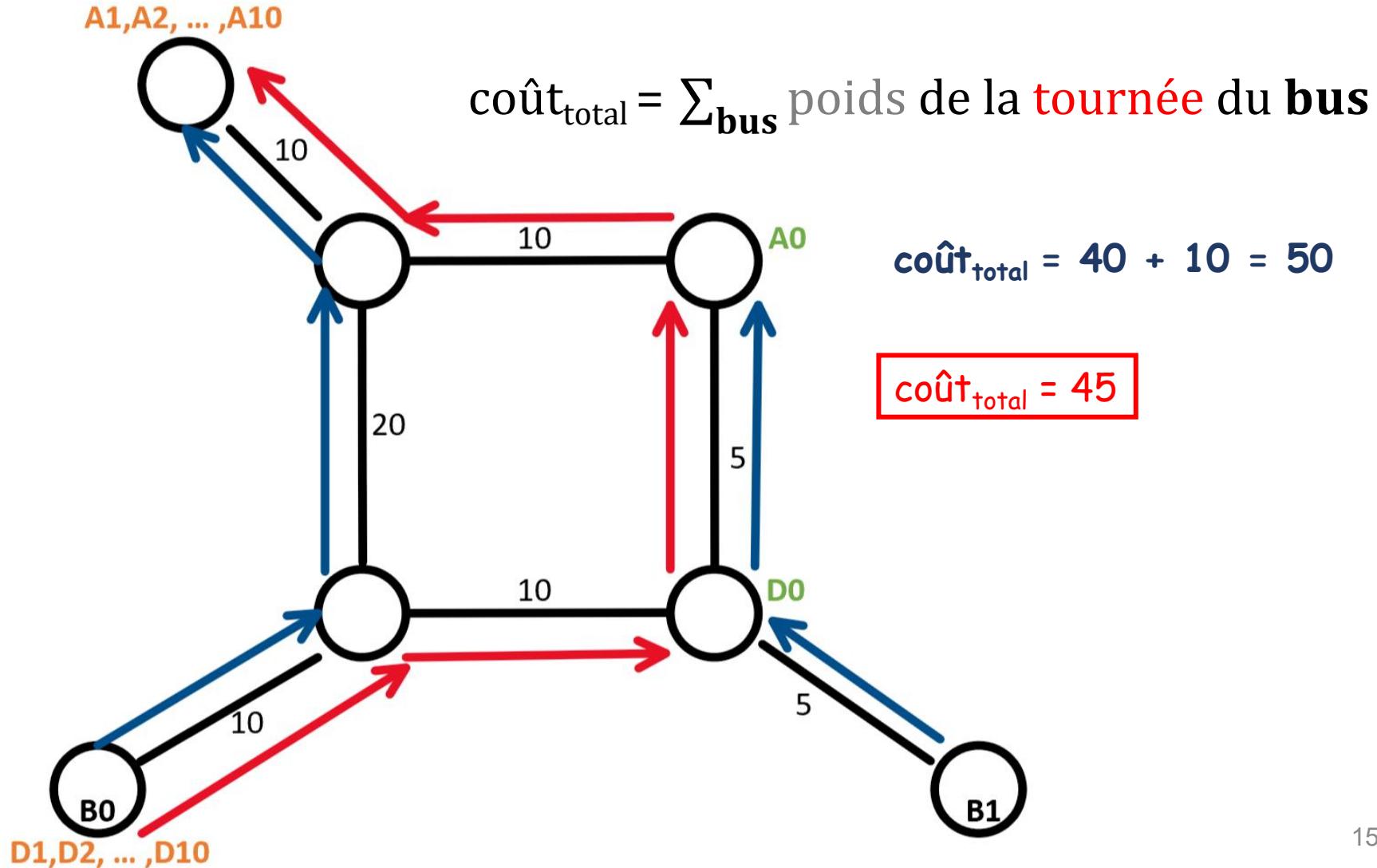
- $B_0 \rightarrow D_0 \rightarrow D_1 \rightarrow A_0 \rightarrow A_1 : 53 + 17 = 70$
 $\searrow D_0 \rightarrow A_0 \rightarrow D_1 \nearrow : 55 + 7 = 62$
- $B_0 \rightarrow D_0 \rightarrow D_1 \rightarrow A_1 \rightarrow A_0 : 38 + 17 = 55$
 $\searrow D_0 \rightarrow A_0 \rightarrow D_1 \nearrow : 55 + 22 = 77$



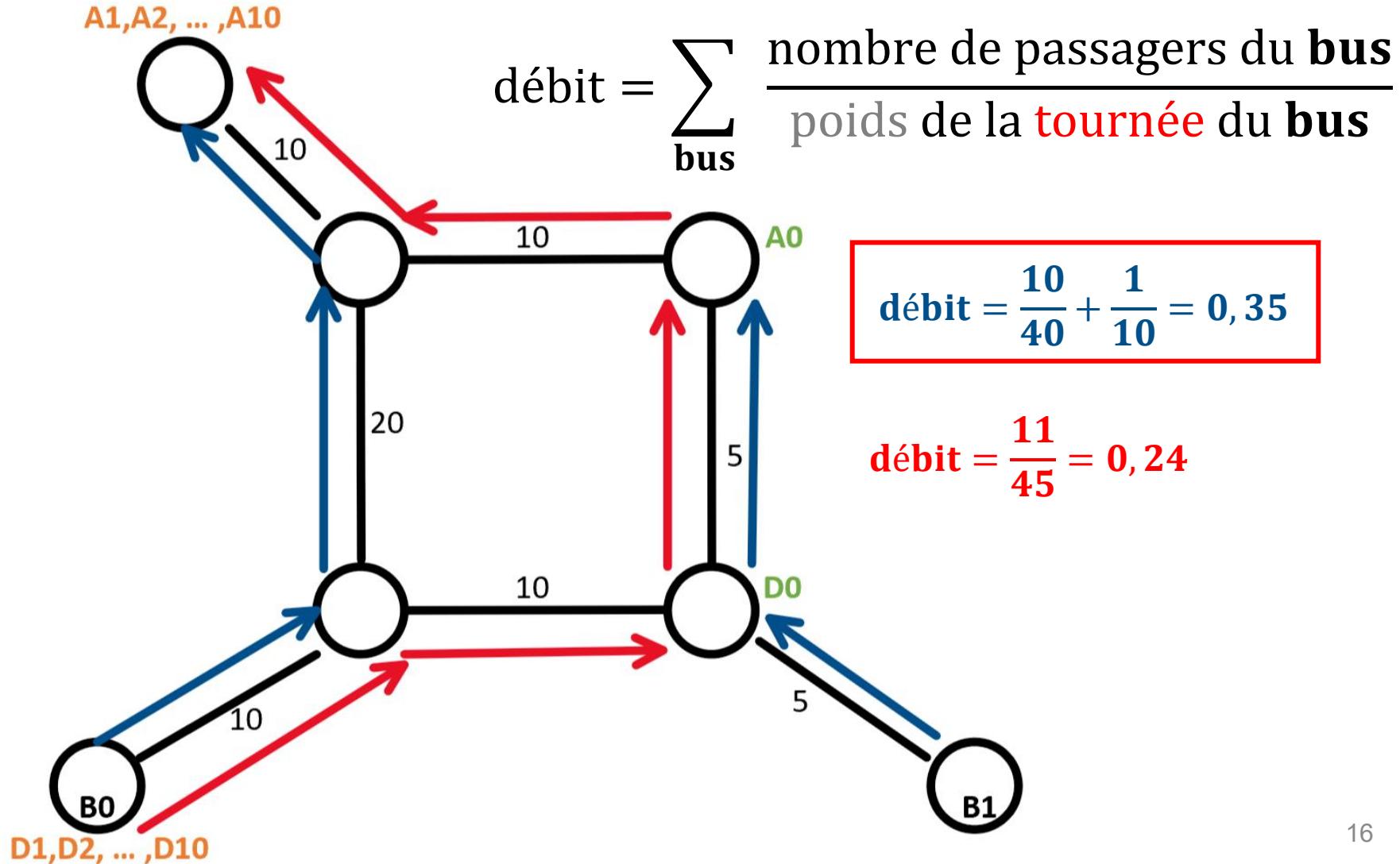
Test de performance



Recherche d'un critère pour le DARP avec **s** bus



Recherche d'un critère pour le DARP avec **s** bus



Résolution naïve du DARP avec **s** bus

Naivement, pour **s** bus il faut tester toutes les associations passagers/bus.

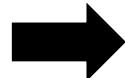
1^{er} passager : s choix de bus

.

.

.

n^{ème} passager : s choix de bus



sⁿ combinaisons

Pour s=3 et n=8, Exemple de **combinaison** : [{2,4} , {}, {1,3,5,6,7,8}]

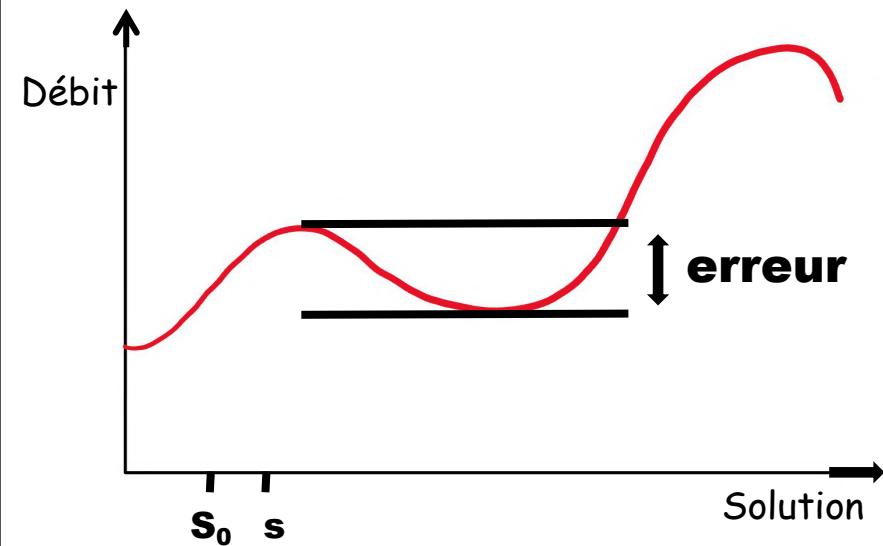
Pour toute **combinaison**

| Pour tout **bus** dans cette **combinaison**

| | je calcule le poids minimal de la **tournée** de ce **bus**

| | je calcule le débit de cette **combinaison**

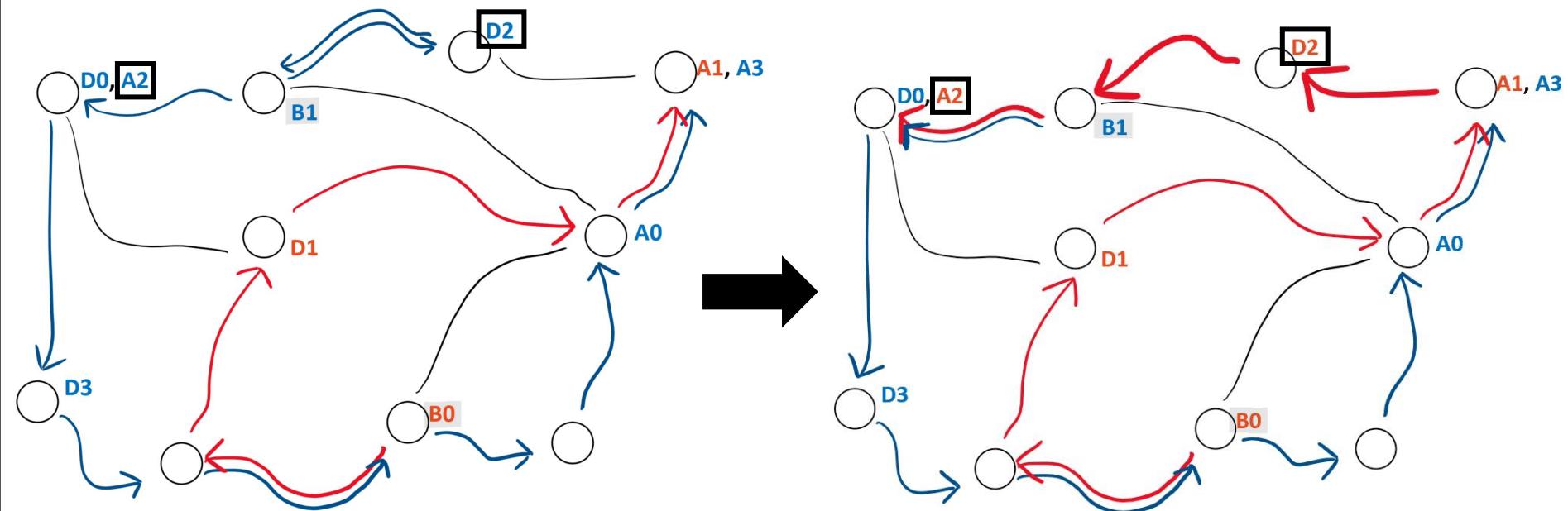
Méthode optimisée du recuit simulé



je génère une solution voisine
Si la nouvelle solution est meilleure
ou **erreur < Marge_erreur**
alors
 je conserve la solution
Sinon
 je la rejette

- On commence avec une **marge_erreur** importante
- Puis on fait décroître la **marge_erreur**

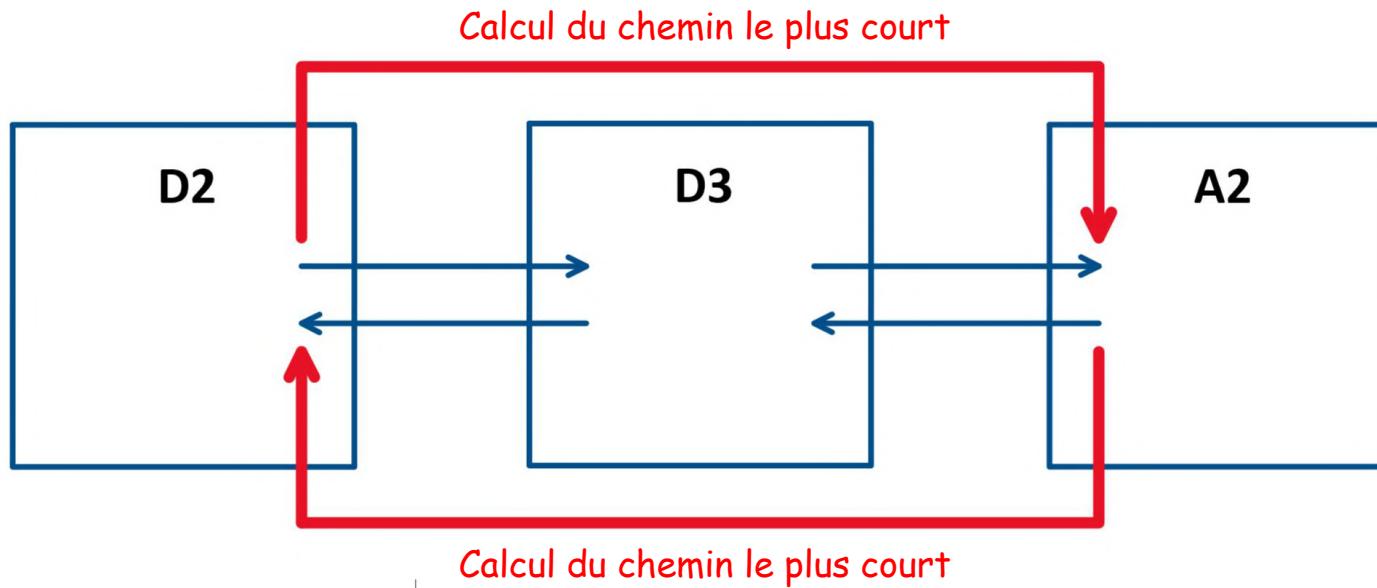
Exemple de solutions voisines



La personne **2** va du bus **1** au bus **0**

Structure de données pour la fonction «voisins»

Opération de suppression d'un sommet d'une tournée :

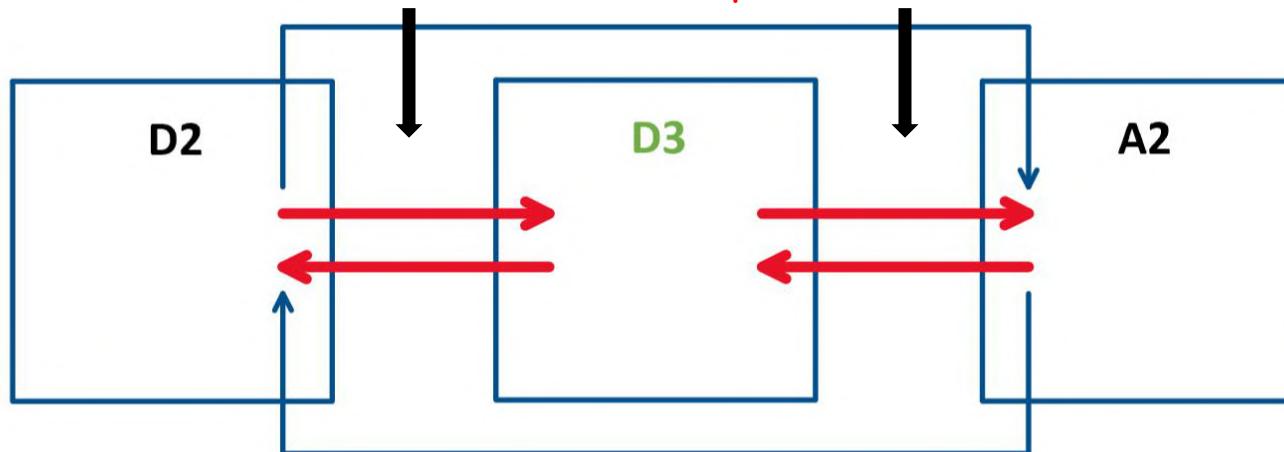


Structure de données pour la fonction «voisins»

Opération d'ajout d'une personne dans une tournée :

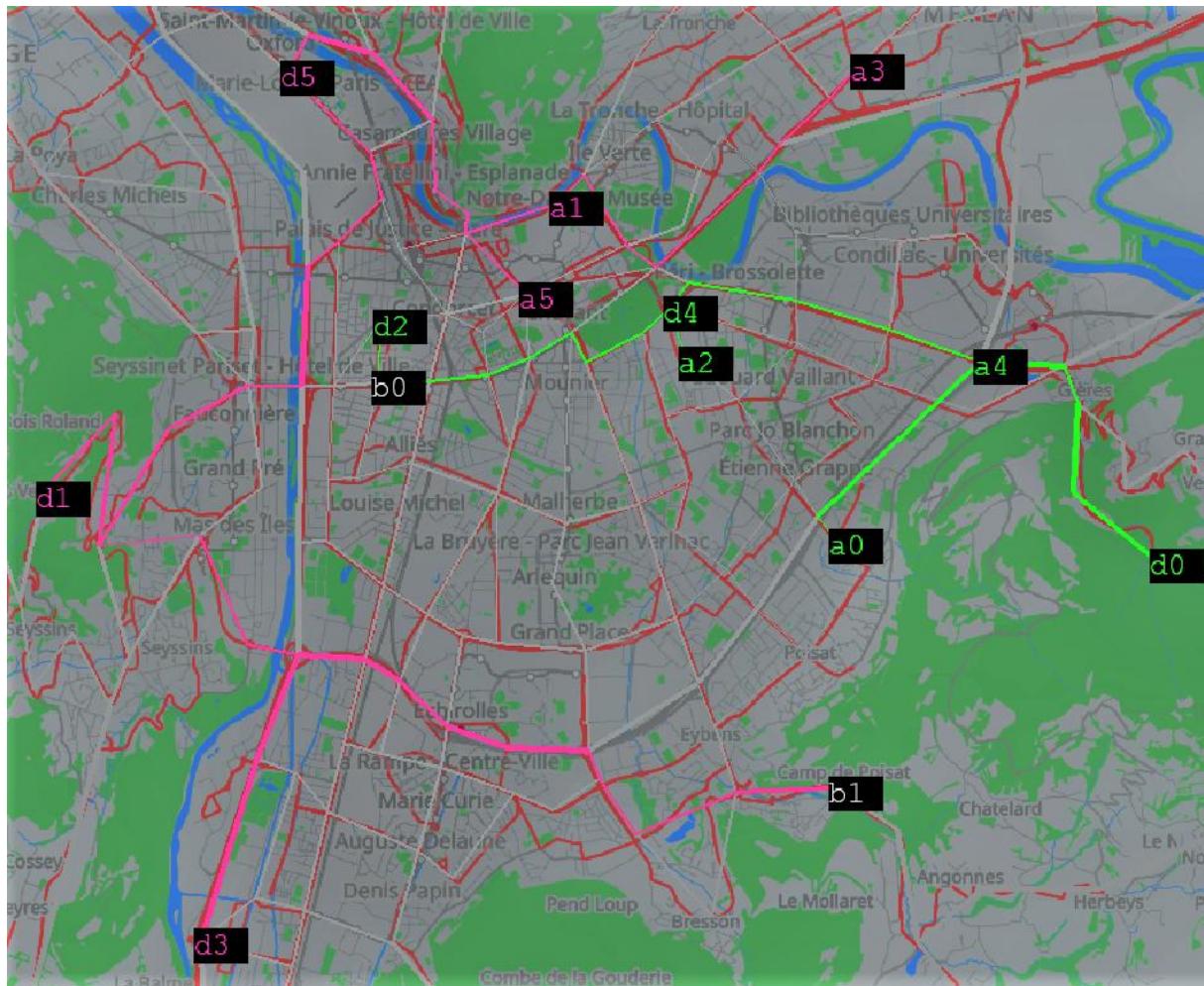


Calcul du chemin le plus court



Meilleure solution

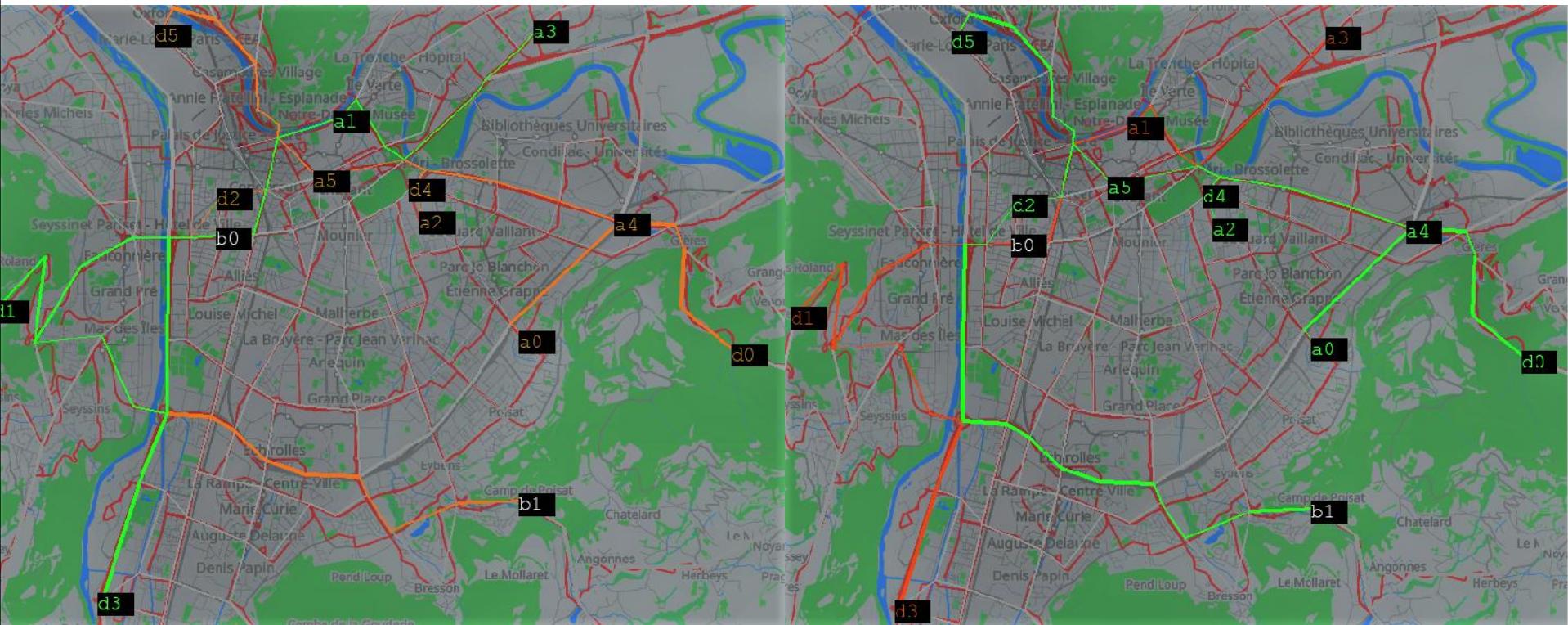
débit = 0.2171



Résultat du recuit simulé

débit : 0.1924

diminution de 11,36%



Test de performance du recuit simulé

Pour **300 personnes** et **20 bus**,

le recuit simulé s'exécute

pour seulement **9 itérations** de la fonction «voisins»

et en **37 secondes**

alors qu'il faudrait faire des **milliers d'itérations**

Conclusion

Pb : La fonction «**Voisins**» du recuit simulé a une complexité trop importante

Solution : réduire la complexité de l'opération d'ajout dans une tournée

Pb : Le recuit simulé ne donne pas de solution satisfaisante

Solution : Trouver une meilleure fonction **Marge_erreur**

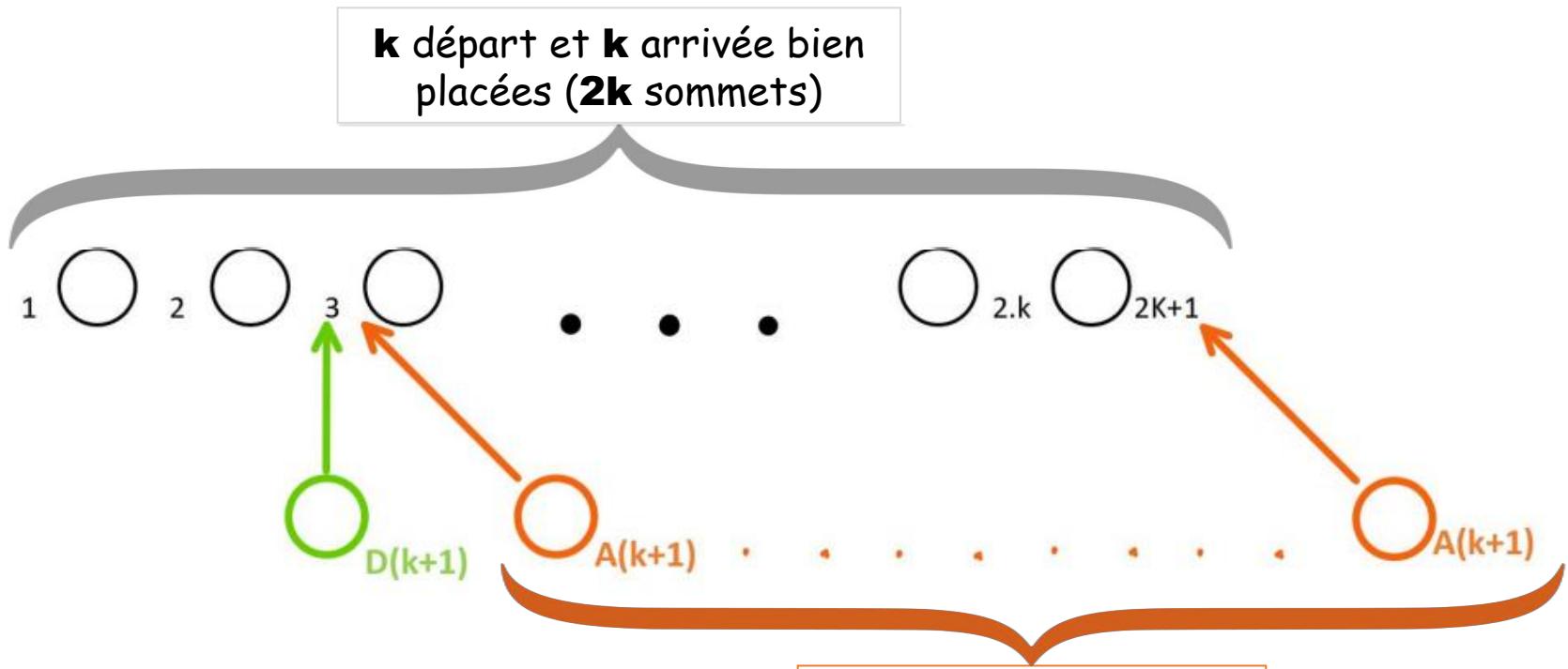
Pb : On ne considère pas le temps, ainsi que beaucoup d'autres contraintes !

Solution : Faire de l'apprentissage par renforcement

Annexe

- | | |
|---|-----|
| 1. Complexité de la solution naïve du DARP avec 1 bus | p27 |
| 2. Recuit simulé pour le problème du voyageur de commerce (TSP) | p30 |
| 3. Détails de la fonction marge_erreur | p33 |
| 4. Choix de l'algorithme A* pour le plus court chemin | p35 |
| 5. Code du DARP | |
| 1. Class DARP | p36 |
| 2. Plus court chemin et création du graphe simplifié | p37 |
| 3. Résolution naïve du DARP avec 1 bus | p42 |
| 4. Résolution optimisée du DARP avec 1 bus | p43 |
| 5. Résolution naïve du DARP avec s bus | p45 |
| 6. Affichage dynamique des solutions | p49 |
| 7. Recuit simulé | |
| a. Génération d'une solution aléatoire | p47 |
| b. Boucle principale du recuit simulé | p53 |
| c. Fonction «voisins» | p56 |
| d. Condition d'acceptation de la nouvelle solution | p55 |
| e. Fonctions de recherche des constantes | p63 |
| 8. Exécution finale du fichier | p64 |
| 6. Code du logiciel de création de graphe | |
| a. Création du fichier .txt | p68 |
| b. import du fichier .txt | p65 |
| 7. Code du recuit simulé pour le problème du voyageur de commerce | p73 |

Complexité de la solution naïve du problème à 1 bus



On note : **z** la position de **D(k+1)**

Complexité du rajout de la **k+1** personne :

$$\sum_{z=1}^{2k+1} (2k+1) - z + 1 = \sum_{z=1}^{2k+1} z = 2k^2 + 3k + 1$$

Complexité de la solution naïve du problème à 1 bus

Complexité du rajout de la **k+1** personne :

$$\sum_{z=1}^{2k+1} (2k + 1) - z + 1 = \sum_{z=1}^{2k+1} z = 2k^2 + 3k + 1$$

Finalement le nombre de tournée à comparer est :

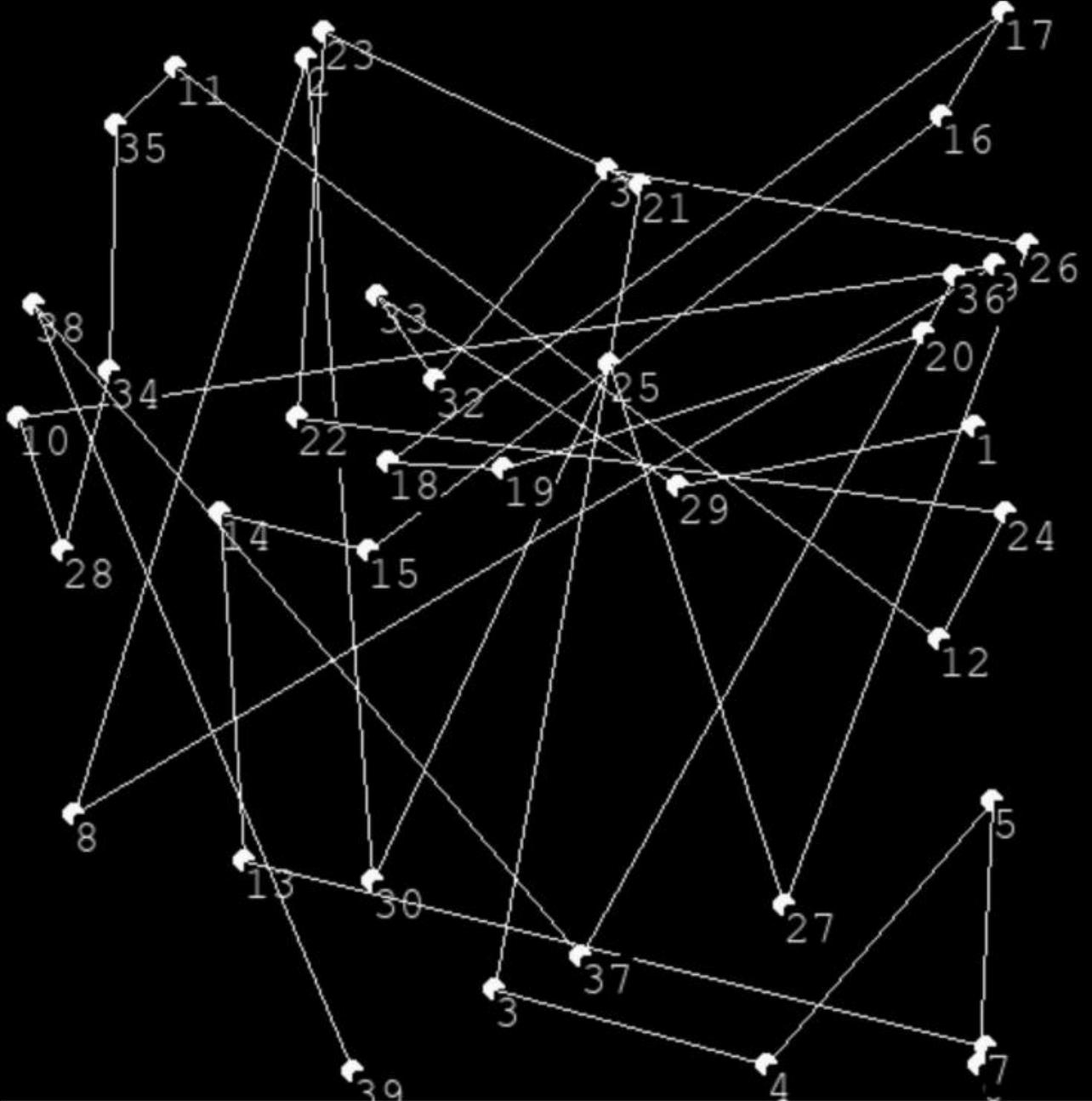
$$\prod_{k=0}^{n-1} 2k^2 + 3k + 1 \sim \underline{2^n(n!)^2}$$

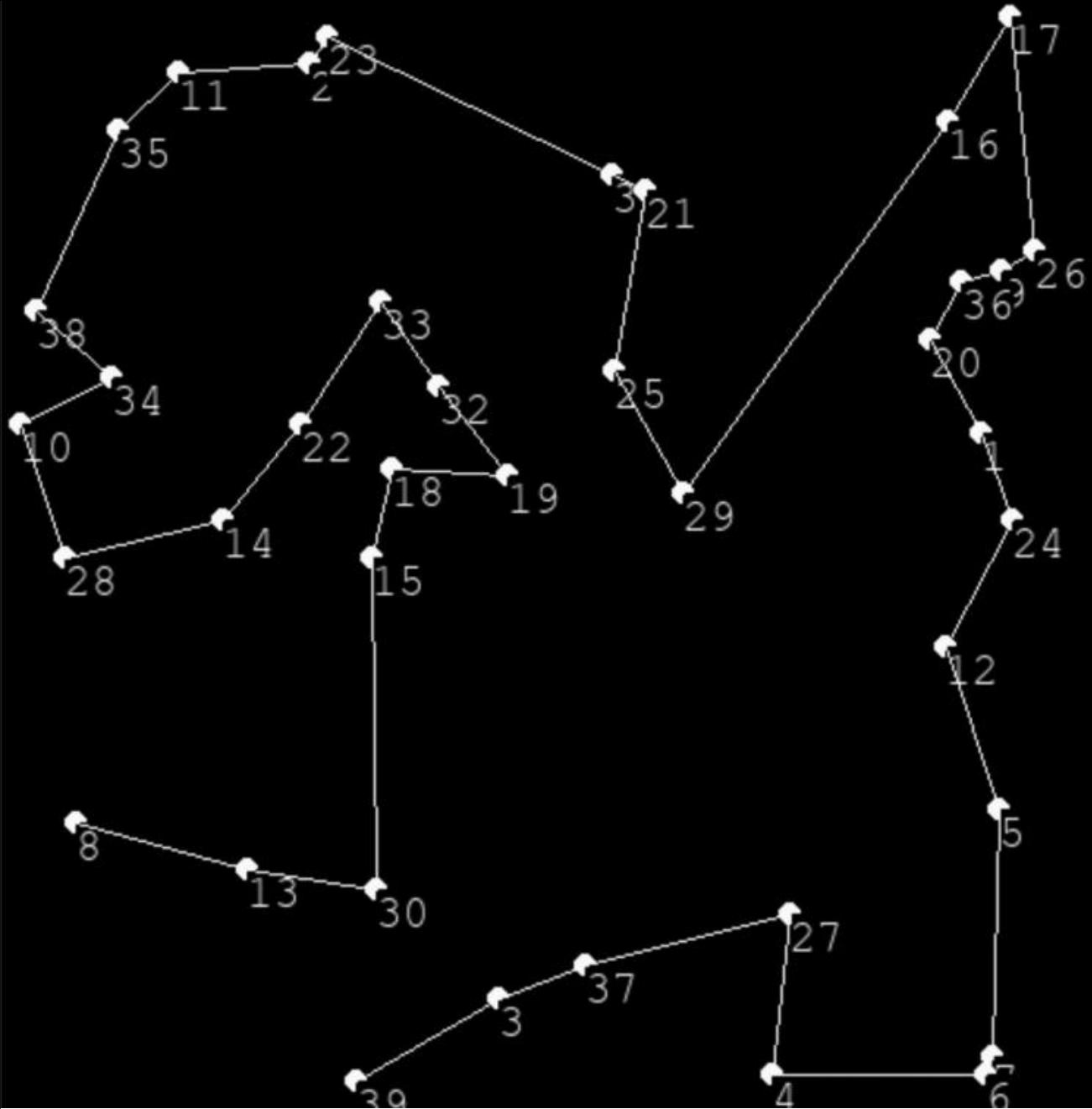
Complexité de la solution naïve du problème à 1 bus

BIS

$$\binom{2n}{2} \binom{2n-2}{2} \dots \binom{2}{2} = \frac{(2n)! \dots 2!}{2^n(2n-2)! \dots 1} = \frac{(2n)!}{2^n}$$

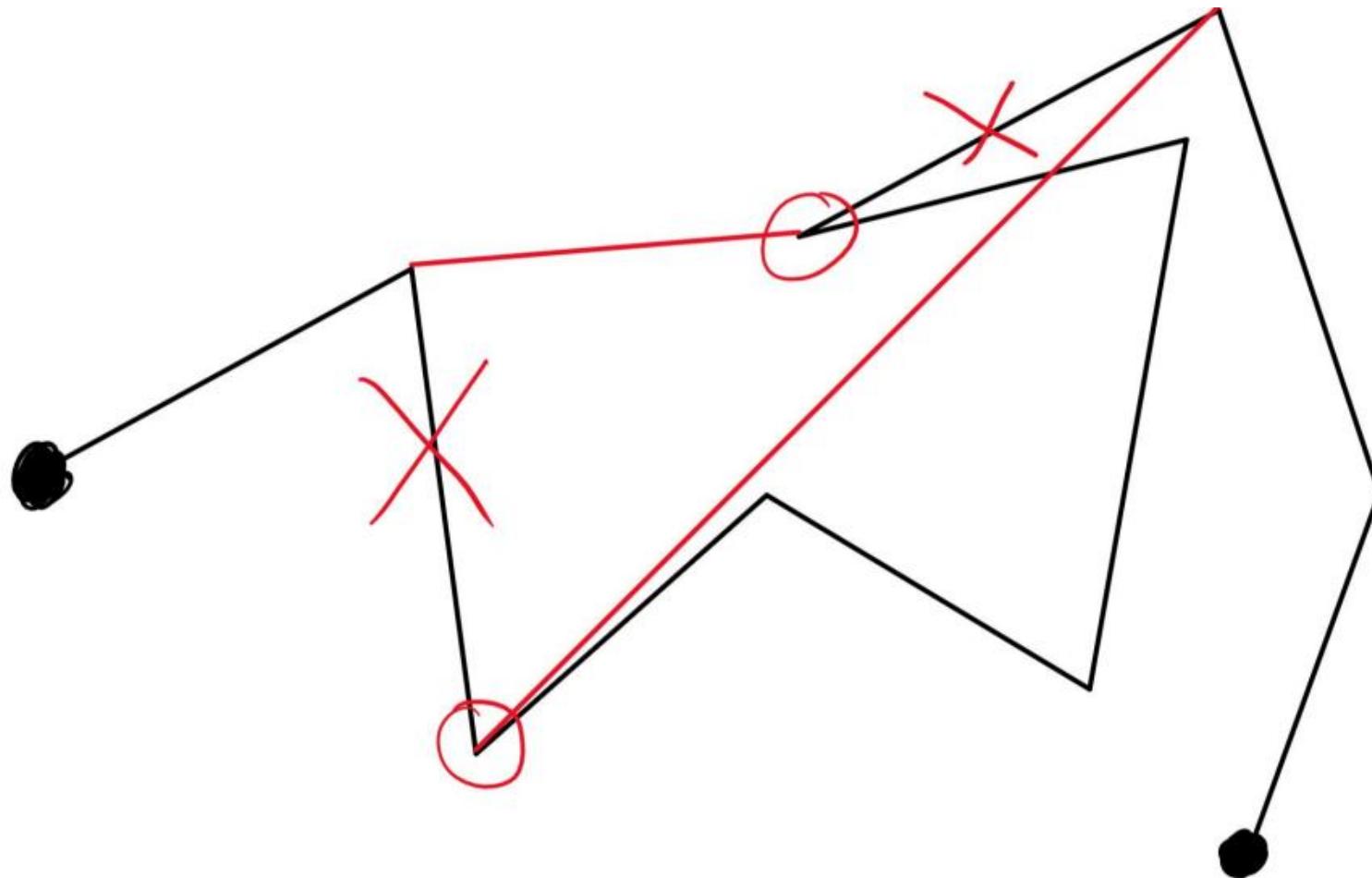
Solution initiale





40 points
17918 itérations
 $t_0 = 6^{1/2}$
 $t_f = 1$
alpha = 0.99995

fonction «voisins» pour le problème du TSP



Explication de la fonction marge_erreur

```
1 # s0 : solution initiale
2 # t0 : valeur de T initiale
3 # tf : valeur de T finale
4 # alpha : vitesse de refroidissement
5 S = s0
6 T = t0
7 while T >= tf :
8     S' = Voisins( S )
9     if débit( S' ) > débit( S ) :
10        S = S'
11    else :
12        erreur = débit( S ) - débit( S' )
13        if erreur <= Marge_erreur(T) :
14            S = S'
15    T *= alpha # T est décroissant
16 return S
```



if $\text{random}(0,1) < e^{-\text{erreur}/T}$:
 $S = S'$

erreur augmente => moins de chance de prendre cette solution
T diminue => moins de chance de prendre cette solution

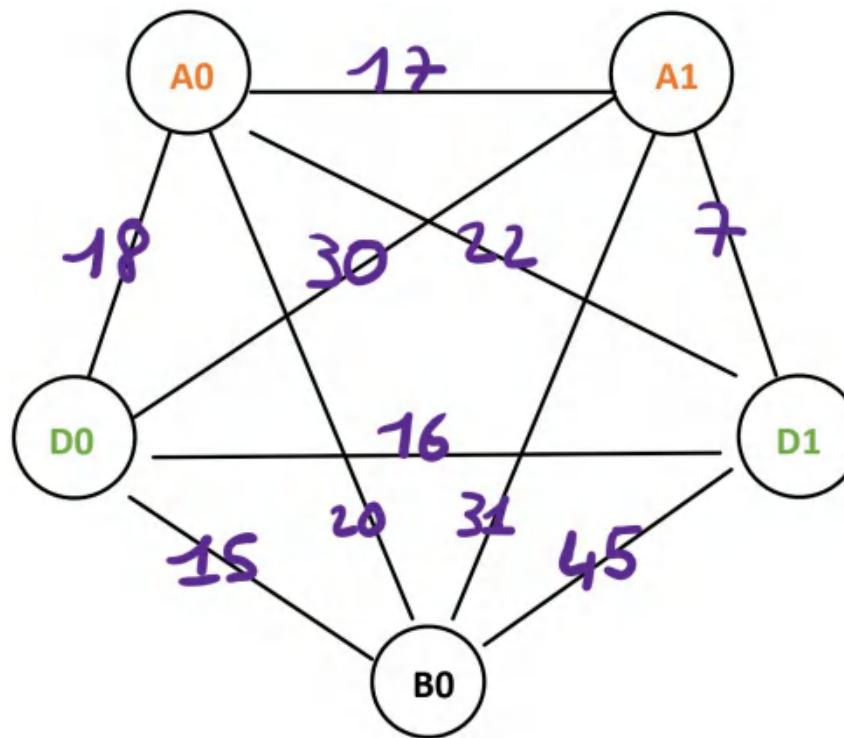
Le recuit simulé inspiré de la statistique de Boltzmann

$$N_i = \frac{1}{Z} \exp\left(-\frac{E_i}{K_b T}\right)$$

$$\frac{N_2}{N_1} = \exp\left(-\frac{(E_2 - E_1)}{K_b T}\right)$$

Plus court chemin pour le graphe simplifié

- Floyd Warshal **O(m^3)** où m est le nombre d'intersection
- $(2n + 1)$ dijtra **O($(2n+1) m \log(m)$)**
- $\sum_{k=1}^{2n} k = \frac{2n(2n+1)}{2}$ A^* avec ditionnaire



```

100 class DARP:
101
102     def __init__(self,M,V,C,H,m,s,S,n,D,A,w,t0,tf,alpha):
103         # M : matrice d'adjacence de la ville pondéré par la durée
104         # V : [ (x,y) , ...] position des sommets sur l'écran
105         # C : matrice d'adjacence contenant la couleur/vitesse des arêtes
106         # L : liste d'adjacence de la ville pondéré par la durée
107         # H : matrice heuristique de la ville ( distance à vol d'oiseau entre chaque sommet)
108         # m : taille de M ou nombre de sommet de la ville
109         # s : nombre de bus
110         # S : liste des sommets des bus
111         # n : nombre de personne
112         # D : liste des sommets de départ des passagers indiqué par leur numéros
113         # A : liste des sommets d'arrivé des passagers indiqué par leur numéros
114         # w : poids heuristique dans A*
115         # t0 : température initial
116         # tf : température final
117         # alpha : vitesse de refroidissement
118         # P : [ [nombre de passager, [liste des numero des passager] , [parcours du bus décomposé en liste] ,
119             # [ordre de récupération des passagers], poids du parcours, [liste des poids des trajet]] ... ] liste des bus
120         # G : dico G[(sommet i,sommet j)] = (poids du trajet i -> j, liste contenant le parcours de ce trajet) \ on a i<=j !
121
122         self.contraste_bus = 0.3
123         self.color_bus = None
124         self.aff_sommet = False
125         self.M = M
126         self.V = V
127         self.C = C
128         self.L = matrice_to_liste(M,m)
129         self.H = H
130         self.m = m
131         self.s = s
132         self.S = S
133         self.n = n
134         self.D = D
135         self.A = A
136         self.w = w
137         self.t0 = t0
138         self.tf = tf
139         self.alpha = alpha
140         self.P = [[0,[],[],[],0,[]] for i in range(s)]
141         self.G = {}

```

```

145
146     def A_star(self,d,a):
147         # d : point de départ
148         # a : point d'arrivé
149         if (d,a) in self.G:
150             return self.G[(d,a)]
151         else:
152             not_visited = [i for i in range(self.m) if i!=d]
153             D = [float('inf')] if i != d else 0 for i in range(self.m)]
154             DH = [float('inf')] if i != d else (self.w*self.H[a][d]) for i in range(self.m)] # temps (du depart a i) +
155                                         # distance (de i a arrive) = dh
156             #
157             P = [ 0 if i != d else 0 for i in range(self.m) ] # on note P[i] = j pour aller en i, il faut passer par j
158             pos = d
159             poids = 0 #ceci est le poids en temps !!
160
161             while ( not_visited != [] ) and (pos != a): # on arrete quand on est arrive
162
163                 for adj in self.L[pos]:
164                     if adj[0] in not_visited: # adj[0] -> sommet adjacent , adj[1] -> poids de l'arc
165                         # H[a][adj[0]] distance a vol d'oiseau du sommet adjacent pos
166                         if ( poids + adj[1] ) + (self.w*self.H[a][adj[0]]) < DH[adj[0]]:      # different de dijtra
167                             D[adj[0]] = poids + adj[1]
168                             DH[adj[0]] = ( poids + adj[1] ) + (self.w*self.H[a][adj[0]])          # |
169                             P[adj[0]] = pos                                         # |
170
171                 # on selectionne le min de DH
172                 minn_DH = DH[not_visited[0]]
173                 minn_D = D[not_visited[0]]
174                 ind_min = not_visited[0]
175                 for ind in not_visited[1:]:
176                     if DH[ind] < minn_DH:
177                         minn_DH = DH[ind]
178                         minn_D = D[ind]
179                         ind_min = ind
180
181                 poids = D[ind_min] # le poids en temps
182                 pos = ind_min
183                 not_visited.remove(pos) # on le marque comme visite
184
185                 G = [a] # parcours du bus final ( trajectoire du bus)
186                 cursor = a
187                 while cursor != d: # temps que je ne suis pas a l'arrive
188                     cursor = P[cursor]
189                     G.append(cursor)
190                 G.reverse()
191

```

```
192  
193  
194  
195  
196     self.G[(d,a)] = (poids,G.copy())  
     tt = G.copy()  
     tt.reverse()  
     self.G[(a,d)] = (poids,tt)  
return (poids,G)
```

```

198 def generer_graph_a_star_avec_chemin(self,n,D,A,b):
199     # n : nombre de passager du bus
200     # D : local
201     # A : local
202     # b : sommet du bus
203     F = np.zeros((2*n+1,2*n+1)) # cout du trajet
204     C = [[[] for _ in range(2*n+1)] for _ in range(2*n+1)] # chemin du trajet
205
206
207     # les arcs : bus => passager / passager => bus
208     for i in range(n):
209         if b == D[i]:
210             F[0,2*i +1] = 0
211             F[2*i +1,0] = 0
212             C[0][2*i +1] = []
213             C[2*i +1][0] = []
214         else:
215             poids,Trajet = DARP.A_star(self,b,D[i])
216             F[0,2*i +1] = poids
217             F[2*i +1,0] = poids
218             C[0][2*i +1] = Trajet.copy()
219             Trajet.reverse()
220             C[2*i +1][0] = Trajet.copy() # le chemin a l'invers
221
222
223     # les arcs : bus => destination des passagers / destination des passagers => bus
224     for i in range(n):
225         if b == A[i]:
226             F[0,2*i +2] = 0
227             F[2*i +2,0] = 0
228             C[0][2*i+2] = []
229             C[2*i+2][0] = []
230         else:
231             poids,Trajet = DARP.A_star(self,b,A[i])
232
233             F[0,2*i +2] = poids
234             F[2*i +2,0] = poids
235             C[0][2*i +2] = Trajet.copy()
236             Trajet.reverse()
237             C[2*i +2][0] = Trajet.copy() # le chemin a l'invers
238
239
240     # les arcs : passager i => passager j / passager j => passager i
241     for i in range(n):
242         for j in range(i+1,n): # on evite ainsi de recalculer la meme chose
243             if D[i] == D[j]:
244                 F[2*i +1,2*j +1] = 0

```

```

245
246     F[2*j +1,2*i +1] = 0
247     C[2*i +1][2*j +1] = []
248     C[2*j +1][2*i +1] = []
249 else:
250     poids,Trajet = DARP.A_star(self,D[i],D[j])
251
252     F[2*i +1,2*j +1] = poids
253     F[2*j +1,2*i +1] = poids
254     C[2*i +1][2*j +1] = Trajet.copy()
255     Trajet.reverse()
256     C[2*j +1][2*i +1] = Trajet.copy()
257
258 # les arcs : destination i => destination j / destination j => destination i
259 for i in range(n):
260     for j in range(i+1,n): # on evite ainsi de recalculer la meme chose
261         if A[i] == A[j]:
262             F[2*i +2,2*j +2] = 0
263             F[2*j +2,2*i +2] = 0
264             C[2*i +2][2*j +2] = []
265             C[2*j +2][2*i +2] = []
266         else:
267
268             poids,Trajet = DARP.A_star(self,A[i],A[j])
269
270             F[2*i +2,2*j +2] = poids
271             F[2*j +2,2*i +2] = poids
272             C[2*i +2][2*j +2] = Trajet.copy()
273             Trajet.reverse()
274             C[2*j +2][2*i +2] = Trajet.copy()
275
276 # les arcs : passager i => destination j / destination j => passager i
277 for i in range(n):
278     for j in range(n): # on evite ainsi de recalculer la meme chose
279         if D[i] == A[j]:
280             F[2*i +1,2*j +2] = 0
281             F[2*j +2,2*i +1] = 0
282             C[2*i +1][2*j +2] = []
283             C[2*j +2][2*i +1] = []
284         else:
285
286             poids,Trajet = DARP.A_star(self,D[i],A[j])
287
288             F[2*i +1,2*j +2] = poids
289             F[2*j +2,2*i +1] = poids
290             C[2*i +1][2*j +2] = Trajet.copy()
291             Trajet.reverse()

```

```
292 ..... C[2*j +2][2*i +1] = Trajet.copy()  
293  
294     return (F,C)
```

```

11 def bus1_chemin_naive(M,n):
12     # [ [ [chemin] , poids du chemin] , ... ]
13     def aux(M,n,pos,Choix,t_choix,Chemin,t_chemin,poids):
14         # M numpy : matrice tsp
15         # n : nombre de passager
16         # pos : position actuel
17         # Choix : liste des possibilité pour le bus
18         # | structure : [ (1 si sommet j disponible) ou (0 si sommet j non disponible)] ou on note j l'indice :
19         # |       | on a j pairs sont les sommets des passagers et j impairs sont les sommets des destinations des passagers
20         #
21         # t_choix : taille de Choix
22         # Chemin : liste du chemin actuellement emprunté
23         # t_chemin : taille de Chemin
24         # poids : poids actuel du chemin
25         global finale
26         if t_chemin == 2*n +1:
27             finale.append([Chemin,poids])
28         for i in range(t_choix):
29             if Choix[i] == 1:
30                 if i%2 == 0: # c'est un passager
31                     Tmp_choix = Choix.copy()
32                     Tmp_chemin = Chemin.copy()
33                     Tmp_choix[i] = 0
34                     Tmp_choix[i+1] = 1
35                     Tmp_chemin.append(i+1)
36                     aux(M,n,i+1,Tmp_choix,t_choix,Tmp_chemin,t_chemin + 1, poids + M[pos,i+1])
37             else:
38                 Tmp_choix = Choix.copy()
39                 Tmp_chemin = Chemin.copy()
40                 Tmp_choix[i] = 0
41                 Tmp_chemin.append(i+1)
42                 aux(M,n,i+1,Tmp_choix,t_choix,Tmp_chemin,t_chemin + 1, poids + M[pos,i+1])
43         aux(M,n,0,[ (i+1)%2 for i in range(2*n)] , 2*n , [0] , 1, 0)
44     return min(finale,key=cmp1)

```

```

296 def bus1_chemin_optimise1(M,d,n):
297     # M (numpy) : matrice tsp
298     # d : sommet de départ du bus
299     # n : nombre de passager
300     a = []
301
302     L = []// liste des chemins à k-1
303     L1 = [] # liste temporaire
304     L2 = []
305
306     # structure des liste L1 et L2
307     # chaque élément sont des listes : [ 'sommet d'arrivée' , 'poids du chemin' , 'état du chemin' , 'chemin' (liste de sommet) ]
308
309
310     switchL = True
311     Gen = [ (m+1)%2 for m in range(2*n)]
312
313     for k in range(1,2*n+1):
314         # parcours de longueur k
315         # A chaque étape on switch de liste entre L1 et L2
316
317         if k == 1: # initialisation
318             for f in range(n):
319                 # on modifie l'état du chemin
320                 a = Gen.copy()
321                 a[2*f] = 0
322                 a[2*f +1] = 1
323                 L1.append([2*f+1, M[d,2*f+1] , a ,[0, 2*(f+1) -1] ].copy())
324                 #fin de l'étape, on change de liste
325                 switchL = True
326
327         else:
328             L=[]
329             # gestion des listes temporaires
330             if switchL:
331                 L = L1.copy()
332                 L2.clear()
333                 switchL = False
334             else:
335                 L = L2.copy()
336                 L1.clear()
337                 switchL = True
338
339             for p in range(2*n):
340                 # on regarde tous les sommets d'arrivées possibles.
341
342                 # On doit trouver le chemin minimum de taille k et qui arrive au sommet p

```

```

343
344         if j[2][p] == 1: # on regarde si les sommets d'arrivés sont potentiellement accessibles
345
346             if p % 2 == 0: # ce sommet est un client
347                 # on change l'état de notre chemin
348                 a = j[2].copy()
349                 a[p] = 0
350                 a[p+1] = 1
351                 J.append([ p+1, j[1] + M[j[0],p+1] ,a, j[3] + [p+1] ].copy())
352
353             else: # ce sommet est une destination
354                 # on change l'état de notre chemin
355                 a = j[2].copy()
356                 a[p] = 0
357                 J.append([ p+1, j[1] + M[j[0],p+1] ,a, j[3] + [p+1] ].copy())
358
359             if J!=[]:
360                 jf = min(J,key=cmp1)
361                 # je sélectionne la liste qui a un poids minimum
362                 if switchL :
363                     L1.append(jf)
364                 else:
365                     L2.append(jf)
366
367             if switchL :
368                 m = min(L1,key=cmp1)
369             else:
370                 m = min(L2,key=cmp1)
371             return (m[1],m[3]) # ( poids , chemin )
372             # a la fin, on prend le chemin qui a un poids minimum.
373

```

```

374     def multi_bus_attribution_naive(self):
375         global Per
376         Per = [] # [ [ num des passager] , ... (pour les s bus) ] ... (combinaison) ]
377
378     def generer_permu(n,s,i,permut):
379
380         if i == n:
381             Per.append(permut)
382             return
383         for b in range(s):
384             tmp = copy.deepcopy(permut)
385             tmp[b].append(i)
386             generer_permu(n,s,i+1,tmp)
387         generer_permu(self.n,self.s,0,[[] for _ in range(self.s)])
388
389         start = True
390         debit_max = 0
391
392
393
394         for perm in Per: # on test tout les permutations
395             som_deb = 0
396             tmp_P = [[0,[],[],[],0,[]] for i in range(s)]
397             for k in range(s): # pour les k bus
398                 pt = [] # parcours temporaire
399                 pt_m = []
400
401                 d = []
402                 a = []
403                 nbs_pa = 0
404                 ordre = []
405                 for pa in perm[k]: # on itere les passager dans le bus k
406                     d.append(D[pa]) # on recupere les sommets de départ / arrive de c'est passager
407                     a.append(A[pa])
408                     ordre.append(pa)
409                     nbs_pa += 1
410
411                 if nbs_pa == 0:
412                     tmp_P[k][0] = nbs_pa
413                     tmp_P[k][1] = []
414                     tmp_P[k][2] = [self.S[k]]
415                     tmp_P[k][3] = []
416                     tmp_P[k][4] = 0
417                     tmp_P[k][5] = []
418                 else:
419                     # OPTI
420                     R,T = DARP.generer_graph_a_star_avec_chemin(self,nbs_pa,d,a,self.S[k]) # on genere les graphs associés

```

```

421 (poids , pt ) = DARP.bus1_chemin_optimise1(R,0,nbs_pa)
422 parcours = []
423
424 ok = [False for _ in range(self.n)]
425
426 for g in range(1,len(pt)):
427     xx = ordre[(pt[g]-1)//2]
428     if ok[xx]:
429         parcours.append((xx*2)+1)
430     else:
431         parcours.append(xx*2)
432         ok[xx] = True
433
434 som_deb += nbs_pa / poids
435 pt_s_d =[pt[0]]
436 taille_pt = len(pt)
437 prec = pt[0]
438 for o in range(1,taille_pt):
439     if prec != pt[o]:
440         pt_s_d.append(pt[o])
441         prec = pt[o]
442
443 pt_m.clear()
444 pt_m = [T[pt_s_d[0]][pt_s_d[1]]]
445 for o in range(1,taille_pt -1):
446     if pt_m == []:
447         pt_m.append(T[pt_s_d[o]][pt_s_d[o+1]])
448     else:
449         pt_m.append(T[pt_s_d[o]][pt_s_d[o+1]][1:])
450 Poids1 = []
451 for gg in range(len(pt)-1):
452     Poids1.append(R[pt[gg]][pt[gg+1]])
453
454 tmp_P[k][0] = nbs_pa
455 tmp_P[k][1] = perm[k].copy()
456 tmp_P[k][2] = pt_m.copy()
457 tmp_P[k][3] = parcours.copy()
458 tmp_P[k][4] = poids
459 tmp_P[k][5] = Poids1.copy()
460 if start or (som_deb > debit_max):
461     start = False
462     debit_max = som_deb
463     self.P = copy.deepcopy(tmp_P)
464 print(debit_max)

```

```

465 def multi_bus_aleatoire(self):
466     global Per
467     Per = [[[] for _ in range(self.s)]] # [ [ [num des passager] , ... (pour les s bus) ] ... (combinaison) ]
468     lia = [i for i in range(self.n)]
469     random.shuffle(lia) # liste de personne aléatoire
470     for bu in range(self.s): # une personne parr bus
471         Per[0][bu].append(lia[bu])
472     for pe in range(self.s,self.n): # on place le reste
473         bdp = random.randint(0,self.s -1)
474         Per[0][bdp].append(lia[pe])
475     start = True
476     debit_max = 0
477     for perm in Per: #une permutation est selectionner !
478         som_deb = 0
479         tmp_P = [[0,[],[],[],0,[]] for i in range(s)]
480         for k in range(s): # pour les k bus
481             pt = [] # parcours temporaire
482             pt_m = []
483             d = []
484             a = []
485             nbs_pa = 0
486             ordre = []
487             for pa in perm[k]: # on itere les passager dans le bus k
488                 d.append(D[pa]) # on recupere les sommets de départ / arrive de c'est passager
489                 a.append(A[pa])
490                 ordre.append(pa)
491                 nbs_pa +=
492             if nbs_pa == 0:
493                 tmp_P[k][0] = nbs_pa
494                 tmp_P[k][1] = []
495                 tmp_P[k][2] = [self.S[k]]
496                 tmp_P[k][3] = []
497                 tmp_P[k][4] = 0
498                 tmp_P[k][5] = []
499             else:
500                 R,T = DARP.generer_graph_a_star_avec_chemin(self,nbs_pa,d,a,self.S[k]) # on genere les graphs associés
501                 (poids , pt) = DARP.bus1_chemin_optimise1(R,0,nbs_pa)
502                 parcours = []
503                 ok = [False for _ in range(self.n)]
504                 for g in range(1,len(pt)):
505                     xx = ordre[(pt[g]-1)//2]
506                     if ok[xx]:
507                         parcours.append((xx*2)+1)
508                     else:
509                         parcours.append(xx*2)
510                         ok[xx] = True
511                 som_deb += nbs_pa / poids

```

```

512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
pt_s_d =[pt[0]]
taille_pt = len(pt)
prec = pt[0]
for o in range(1,taille_pt):
    if prec != pt[o]:
        pt_s_d.append(pt[o])
    prec = pt[o]
pt_m.clear()
pt_m = [T[pt_s_d[0]][pt_s_d[1]]]
for o in range(1,taille_pt -1):
    if pt_m == []:
        pt_m.append(T[pt_s_d[o]][pt_s_d[o+1]])
    else:
        pt_m.append(T[pt_s_d[o]][pt_s_d[o+1]][1:])
PoidsL = []
for gg in range(len(pt)-1):
    PoidsL.append(R[pt[gg]][pt[gg+1]])
tmp_P[k][0] = nbs_pa
tmp_P[k][1] = perm[k].copy()
tmp_P[k][2] = pt_m.copy()
tmp_P[k][3] = parcours.copy()
tmp_P[k][4] = poids
tmp_P[k][5] = PoidsL.copy()
if start or (som_deb > debit_max):
    start = False
    debit_max = som_deb
self.P = copy.deepcopy(tmp_P)

```

```

540
541     def affiche(self):
542         global P2
543         P2 = [[] for _ in range(self.s)]
544         for b in range(self.s):
545             for el in self.P[b][2]:
546                 for ell in el:
547                     P2[b].append(ell)
548     def rendu(screen, bg, Color_bus, Color_sommet, prio, sel_arc):
549         police = pygame.font.SysFont("monospace", 20)
550         screen.blit(bg, (0,0))
551
552         image_text = police.render(str(prio), 1, Color_bus[prio], (0,0,0))
553         screen.blit(image_text, (10,10))
554         for i in range(1, self.m):
555             for j in range(i):
556                 if self.C[i][j] != 0:
557                     pygame.draw.line(screen, (100,100,100), self.V[i], self.V[j], width=self.C[i][j])
558         Color_bus_bis = Color_bus.copy()
559         for ll in range(self.s):
560             if ll != prio:
561                 Color_bus_bis[ll] = eclaircisement(Color_bus[ll], self.contraste_bus)
562         for b in range(self.s): # on itere les bus
563             if b != prio:
564                 for i in range(len(P2[b])-1):
565                     sd = P2[b][i] # sommet de depart
566                     sa = P2[b][i+1] # sommet d'arrive
567                     pygame.draw.line(screen, Color_bus_bis[b], self.V[sd], self.V[sa], width=self.C[sd][sa])
568         for i in range(len(P2[prio])-1):
569             if i != sel_arc:
570                 sd = P2[prio][i] # sommet de depart
571                 sa = P2[prio][i+1] # sommet d'arrive
572                 pygame.draw.line(screen, Color_bus_bis[prio], self.V[sd], self.V[sa], width=self.C[sd][sa])
573         if sel_arc != -1:
574             sd = P2[prio][sel_arc] # sommet de depart
575             sa = P2[prio][sel_arc+1] # sommet d'arrive
576             pygame.draw.line(screen, (0,0,0), self.V[sd], self.V[sa], width=self.C[sd][sa])
577         cpt_sommet = 0
578         for som in self.V:
579             if self.aff_sommet:
580                 image_text = police.render(str(cpt_sommet), 1, (255,255,255))
581                 screen.blit(image_text, (som[0]-6, som[1]-10))
582                 image_text = police.render(Aff_sommet[cpt_sommet], 0.5, Color_sommet[cpt_sommet], (0,0,0))
583                 screen.blit(image_text, (som[0]-4, som[1]-7))
584                 cpt_sommet += 1
585         pygame.display.update()

```

```

586 def eclairsisement(color,x): # assombrit le vert
587     c = [0,0,0]
588     if color[0] > color[2]:
589         maxx = 0
590     else:
591         maxx = 2
592     c[maxx] = (230 - color[maxx])
593     for j in range(3):
594         if j!=maxx:
595             c[j] = -(color[j])
596     return (color[0]+int(x*c[0]),color[1]+int(x*c[1]),color[2]+int(x*c[2]))
597
598 global SIZE,RAYON,COLOR
599 pygame.init()
600 pygame.display.set_caption("TIPE")
601
602 screen = pygame.display.set_mode((SIZEx,SIZEy))
603 bg = pygame.image.load("map_gre2.png")
604
605 running = True
606 if self.color_bus == None:
607     Color_bus = [(random.randint(0,230),random.randint(0,230),random.randint(0,230)) for _ in range(s)]
608 else:
609     Color_bus = self.color_bus
610 Color_sommet = [(0,0,0) for _ in range(m)] # couleur de chaque sommet
611
612 Aff_sommet = [" " for _ in range(m)]
613 for i in range(self.s):
614     Aff_sommet[self.S[i]] += ("b" + str(i) + " ")
615     Color_sommet[self.S[i]] = (200,200,200) # <-- couleur des bus !
616
617 for i in range(self.n):
618     Aff_sommet[self.D[i]] += ("d" + str(i) + " ")
619     Aff_sommet[self.A[i]] += ("a" + str(i) + " ")
620     for kk in range(self.s):
621         if self.D[i] in P2[kk]: # ce sommet est desservie par le bus kk
622             Color_sommet[self.D[i]] = Color_bus[kk]
623             Color_sommet[self.A[i]] = Color_bus[kk]
624
625 rendu(screen,bg,Color_bus,Color_sommet,0,-1)
626 select = 0
627 sel_arc = 0
628 before_color_bus = Color_bus[0]
629 taille_coresp = []
630 for hh in range(self.s):
631     taille_coresp.append(len(P2[hh]))

```

```

634
635     while running:
636         for event in pygame.event.get():
637
638             if event.type == pygame.QUIT:
639                 running = False
640             if event.type == pygame.KEYDOWN:
641
642                 if event.key == 1073741903: # fleche de droite
643                     sel_arc = 0
644                     if select < (self.s-1):
645                         for i in range(self.n):
646                             if i in self.P[select][1]:
647                                 Color_sommet[self.D[i]] = before_color_bus
648                             if i in self.P[select][1]:
649                                 Color_sommet[self.A[i]] = before_color_bus
650                     Color_bus[select] = before_color_bus # on remet la bonne couleur
651                     select += 1
652                     before_color_bus = Color_bus[select] # on change la nouvelle
653                     Color_bus[select] = (0,255,0)
654                     for i in range(self.n):
655                         if i in self.P[select][1]:
656                             Color_sommet[self.D[i]] = (0,255,0)
657                         if i in self.P[select][1]:
658                             Color_sommet[self.A[i]] = (0,255,0)
659                     rendu(screen, bg, Color_bus, Color_sommet, select, -1)
660             if event.key == 1073741904: # fleche de gauche
661                 sel_arc = 0
662                 if select > 0:
663                     for i in range(self.n):
664                         if i in self.P[select][1]:
665                             Color_sommet[self.D[i]] = before_color_bus
666                         if i in self.P[select][1]:
667                             Color_sommet[self.A[i]] = before_color_bus
668                     Color_bus[select] = before_color_bus # on remet la bonne couleur
669                     select -= 1
670                     before_color_bus = Color_bus[select] # on change la nouvelle
671                     Color_bus[select] = (0,255,0)
672                     for i in range(self.n):
673                         if i in self.P[select][1]:
674                             Color_sommet[self.D[i]] = (0,255,0)
675                         if i in self.P[select][1]:
676                             Color_sommet[self.A[i]] = (0,255,0)
677                     rendu(screen, bg, Color_bus, Color_sommet, select, -1)
678             if event.key == 1073741906: # fleche du haut
679                 if sel_arc < (taille_coresp[select]-2):
680                     rendu(screen, bg, Color_bus, Color_sommet, select, -1)
681                     sel_arc += 1

```

```
681  
682 ▼  
683 ▼  
684  
685  
686  
rendu(screen,bg,Color_bus,Color_sommet,select,sel_arc)  
if event.key == 1073741905: # fleche du bas  
    if sel_arc > 0:  
        rendu(screen,bg,Color_bus,Color_sommet,select,-1)  
        sel_arc -= 1  
    rendu(screen,bg,Color_bus,Color_sommet,select,sel_arc)
```

```

888
889     def multi_bus_attribution_recu_i_simule(self):
890         global Ordre,Bu,First,Ppb,Tdb
891
892         # =====
893         DARP.multi_bus_aleatoire(self) # solution initial
894         Ppb = [0 for _ in range(self.s)] # Le nombre de personne par bus
895         for b in range(self.s):
896             Ppb[b] = self.P[b][0]
897         Tdb = [0 for _ in range(self.s)] # c est le temps de parcours de chaque bus
898         for b in range(self.s):
899             Tdb[b] = self.P[b][4]
900         Ordre = [{}) for _ in range(self.s)]
901         # Ordre[num_bus][depart paire/arrive impaire personne] = [(depart/arrive,[chemin],poids), (depart/arrive,[chemin],poids)]
902         Bu = [0 for _ in range(self.n)] # donne le bus des personnes
903         for b in range(self.s):
904             for pa in self.P[b][1]:
905                 Bu[pa] = b
906
907         # P : [  [nombre de passager, [liste des numero des passager] , [parcours du bus] , [ordre de recuperation des passagers],
908         #           poids du parcours] ... ] liste des bus
909         First = [-1 for _ in range(self.s)] #First[bus] = premier sommet a prendre
910
911         #creation de l'ensemble
912         for b in range(self.s):
913             # connection du 1er a gauche au bus et a droite au reste
914             if self.P[b][0] != 0:
915                 First[b] = self.P[b][3][0]
916
917                 Ordre[b][self.P[b][3][0]] = [(None,self.P[b][2][0],self.P[b][5][0]),(self.P[b][3][1],self.P[b][2][1],self.P[b][5][1])]
918                 for i in range(1,self.P[b][0]*2 - 1):
919                     Ordre[b][self.P[b][3][i]] = [(self.P[b][3][i-1],self.P[b][2][i],self.P[b][5][i]),(self.P[b][3][i+1],
920                         self.P[b][2][i+1],self.P[b][5][i+1])]
921
922                     Ordre[b][self.P[b][3][-1]] = [(self.P[b][3][-2],self.P[b][2][-1],self.P[b][5][-1]),(None,None,None)]
923         # =====
924         anc_deb = 0
925         for b in range(self.s):
926             if Tdb[b] != 0:
927                 anc_deb+= Ppb[b]/Tdb[b]
928         temperature = self.t0
929         ct = 0
930         cta = 0
931         while temperature>self.tf:
932             ct+=1
933             cta += 1
934             if cta > 10000:
935                 cta = 0
936             voisinage(temperature)
937             temperature *= self.alpha
938         print("nbs d'iteration",ct)

```

```
1107  
1108     deb = 0  
1109     for b in range(self.s):  
1110         if Tdb[b] != 0:  
1111             deb+= Ppb[b]/Tdb[b]  
1112     self.P = copy.deepcopy(trfm_ordre_to_p())  
1113     print(deb)  
1114     return (deb/anc_deb)  
-----
```

```

667 def affiche_ordre(Ordre,First):
668     for b in range(self.s):
669         print()
670         print("bus :" +str(b))
671         state = First[b]
672         while state != None:
673             L = Ordre[b][state]
674             print(state,L)
675             state = L[1][0]
676             print()
677 def condition(bpp,bd,tps_enl,tps_raj,temperature):
678     # bpp : bus de la personne a prendre ( qu'on retire du bus )
679     # bd : nouveau bus de la personne a prendre
680     # tps_enl : temps du bus ou l'on va enlever un passager
681     # tps_raj : temps du bus ou l'on va rajouter un passager
682     # delta_deb = nv_deb - ancien_deb
683     delta_deb = ((Ppb[bpp]-1)/tps_enl) + ((Ppb[bd]+1)/tps_raj) - (Ppb[bd]/Tdb[bd]) - (Ppb[bpp]/Tdb[bpp])
684     if delta_deb > 0: # la nv solution est meilleure !
685         return True
686     else:
687         if random.random() < math.exp(-abs(delta_deb)/temperature):
688             return True
689         else:
690             return False
691 def condition2(bpp,bd,tps_enl,tps_raj,temperature):
692     return True
693 def trfm_ordre_to_p():
694     P2 = [[0,[],[],[],0,[]] for i in range(s)]
695     for p in range(self.n): # je rajoute la liste des personnes pris par les bus
696         P2[Bu[p]][1].append(p)
697     for b in range(self.s):
698         P2[b][4] = Tdb[b]
699     for b in range(self.s):
700         P2[b][0] = Ppb[b]
701         state = First[b]
702         while state != None:
703             L = Ordre[b][state]
704             P2[b][2].append(L[0][1])
705             P2[b][3].append(state)
706             P2[b][5].append(L[0][2])
707             state = L[1][0]
708     return P2
709 def trfm_etat_to_sommet(etat):
710     if etat % 2 == 0:# depart
711         return self.D[etat//2]
712     else: # arrive
713         return self.A[etat//2]

```

```

715 def voisinage(temperature):
716     #on selectionne la personne a prendre
717     Bnv = set() # bus avec un nbs de passager sup a 2
718     for b in range(self.s):
719         if Ppb[b] >= 2:
720             Bnv.add(b)
721     P_enl = []
722     taille_P_enl = 0
723     for p in range(self.n):
724         if Bu[p] in Bnv:
725             P_enl.append(p)
726             taille_P_enl += 1
727     pp = P_enl[random.randint(0,taille_P_enl-1)]
728     #pp = 2
729     bpp = Bu[pp] # bus de la personne a prendre ( qu'on retire du bus )
730     # pp * 2 point de depart de la pp
731     bd = random.randint(0,self.s-2) # choix du nouveau bus de la personne a prendre
732     if bd >= bpp: # histoire que ça serve a quel que chose
733         bd += 1
734     tps_nv_b_enl = 0
735     tps_nv_b_raj = 0
736     # =====
737     # on enleve la personne dans le bus selectionné
738     L = Ordre[bpp][pp*2]
739     avant_d_enl,ch_d_avant_enl,poids_d_av_enl = L[0]
740     apres_d_enl,ch_d_apres_enl,poids_d_ap_enl = L[1]
741
742     L = Ordre[bpp][pp*2+1]
743     avant_a_enl,ch_a_avant_enl,poids_a_av_enl = L[0]
744     apres_a_enl,ch_a_apres_enl,poids_a_ap_enl = L[1]
745
746     if apres_d_enl == 2*pp +1 or avant_a_enl == 2*pp:
747         cote_a_cote_enl = True
748     else:
749         cote_a_cote_enl = False
750
751     if cote_a_cote_enl:
752         if avant_d_enl == None and apres_a_enl == None:
753             tps_nv_b_enl = 0
754             # il y a rien
755         elif avant_d_enl == None: # avant il y a le bus
756             # rien
757             poids_a_enl,Trajet_a_enl = DARP.A_star(self,self.S[bpp],trfm_etat_to_sommet(apres_a_enl))
758             tps_nv_b_enl = Tdb[bpp] - (Ordre[bpp][pp*2][0][2] + Ordre[bpp][pp*2][1][2] + Ordre[bpp][pp*2+1][1][2]) +
759                                         poids_a_enl
760         elif apres_a_enl == None: #apres il y a rien
761             poids_d_enl,Trajet_d_enl = (0,[])

```

```

762         # rien
763         tps_nv_b_enl = Tdb[bpp] - (Ordre[bpp][pp*2][1][2] + Ordre[bpp][pp*2][0][2])
764     else:
765         poids_d_enl,Trajet_d_enl = DARP.A_star(self,trfm_etat_to_sommet(avant_d_enl),trfm_etat_to_sommet(apres_a_enl))
766         tps_nv_b_enl = Tdb[bpp] - (Ordre[bpp][pp*2][0][2] + Ordre[bpp][pp*2][1][2] + Ordre[bpp][pp*2+1][1][2]) +
767                                     poids_d_enl
768     else:
769         if avant_d_enl == None and apres_a_enl == None:
770             poids_d_enl,Trajet_d_enl = DARP.A_star(self,self.S[bpp],trfm_etat_to_sommet(apres_d_enl))
771             poids_a_enl,Trajet_a_enl = (0,[])
772             tps_nv_b_enl = Tdb[bpp] - (Ordre[bpp][pp*2][0][2] + Ordre[bpp][pp*2][1][2] + Ordre[bpp][pp*2+1][0][2]) +
773                                     poids_d_enl
774         elif avant_d_enl == None: # avant il y a le bus
775             poids_d_enl,Trajet_d_enl = DARP.A_star(self,self.S[bpp],trfm_etat_to_sommet(apres_d_enl))
776             poids_a_enl,Trajet_a_enl = DARP.A_star(self,trfm_etat_to_sommet(avant_a_enl),trfm_etat_to_sommet(apres_a_enl))
777             tps_nv_b_enl = Tdb[bpp] - (Ordre[bpp][pp*2][0][2] + Ordre[bpp][pp*2][1][2] + Ordre[bpp][pp*2+1][0][2] +
778                                         Ordre[bpp][pp*2+1][1][2]) + poids_d_enl + poids_a_enl
779         elif apres_a_enl == None: #apres il y a rien
780             poids_d_enl,Trajet_d_enl = DARP.A_star(self,trfm_etat_to_sommet(avant_d_enl),trfm_etat_to_sommet(apres_d_enl))
781             poids_a_enl,Trajet_a_enl = (0,[])
782             tps_nv_b_enl = Tdb[bpp] - (Ordre[bpp][pp*2][0][2] + Ordre[bpp][pp*2][1][2] + Ordre[bpp][pp*2+1][0][2]) +
783                                     poids_d_enl
784     else:
785         poids_d_enl,Trajet_d_enl = DARP.A_star(self,trfm_etat_to_sommet(avant_d_enl),trfm_etat_to_sommet(apres_d_enl))
786         poids_a_enl,Trajet_a_enl = DARP.A_star(self,trfm_etat_to_sommet(avant_a_enl),trfm_etat_to_sommet(apres_a_enl))
787         tps_nv_b_enl = Tdb[bpp] - (Ordre[bpp][pp*2][0][2] + Ordre[bpp][pp*2][1][2] + Ordre[bpp][pp*2+1][0][2] +
788                                         Ordre[bpp][pp*2+1][1][2]) + poids_d_enl + poids_a_enl
789
790     # =====
791     # poids et trajet du rajout de la personne (dans l'autre bus)
792
793     # variable de stockage du min
794     # ce sont des etats !! minn !
795     minn = float('inf')
796     poids_m_d_ava = 0
797     poids_m_d_pro = 0
798     poids_m_a_ava = 0
799     poids_m_a_pro = 0
800     avant_d_raj = 0
801     avant_a_raj = 0
802     apres_d_raj = 0
803     apres_a_raj = 0
804     ch_avant_d_raj = []
805     ch_avant_a_raj = []
806     ch_apres_d_raj = []
807     ch_apres_a_raj = []
808     min_cote_a_cote = True

```

```

809 #init !!!!!
810 etat_d_ava = None
811 etat_d_pro = pp*2 +1 # arrie, init cote a cote
812 etat_a_ava = pp*2 #init cote a cote
813 etat_a_pro = First[bd]
814 while True != None: # on insere la ou c'est le mieux
815     cote_a_cote = True
816     while True:
817         if cote_a_cote:
818             if etat_d_ava == None and etat_a_pro == None:
819                 poids1,Trajet_raj_d1 = DARP.A_star(self,self.S[bd],self.D[pp])
820                 poids2,Trajet_raj_d2 = DARP.A_star(self,self.D[pp],self.A[pp])
821                 poids3,Trajet_raj_a1 = (0,[])
822                 tps_nv_b_raj = (poids1 + poids2)
823             if etat_d_ava == None: #avant etat_d_ava il y a le bus
824                 poids1,Trajet_raj_d1 = DARP.A_star(self,self.S[bd],self.D[pp])
825                 poids2,Trajet_raj_d2 = DARP.A_star(self,self.D[pp],self.A[pp])
826                 poids3,Trajet_raj_a1 = DARP.A_star(self,self.A[pp],trfm_etat_to_sommet(etat_a_pro))
827                 tps_nv_b_raj = Tdb[bd] - (Ordre[bd][etat_a_pro][0][2]) + poids1 + poids2 + poids3
828             elif etat_a_pro == None: # tout après il y a rien
829                 poids1,Trajet_raj_d1 = DARP.A_star(self,trfm_etat_to_sommet(etat_d_ava),self.D[pp])
830                 poids2,Trajet_raj_d2 = DARP.A_star(self,self.D[pp],self.A[pp])
831                 poids3,Trajet_raj_a1 = (0,[])
832                 tps_nv_b_raj = Tdb[bd] + poids1 + poids2
833             else:
834                 poids1,Trajet_raj_d1 = DARP.A_star(self,trfm_etat_to_sommet(etat_d_ava),self.D[pp])
835                 poids2,Trajet_raj_d2 = DARP.A_star(self,self.D[pp],self.A[pp])
836                 poids3,Trajet_raj_a1 = DARP.A_star(self,self.A[pp],trfm_etat_to_sommet(etat_a_pro))
837                 tps_nv_b_raj = Tdb[bd] - (Ordre[bd][etat_d_ava][1][2]) + poids1 + poids2 + poids3
838             if tps_nv_b_raj < minn: # on selectionne le best
839                 minn = tps_nv_b_raj
840                 avant_d_raj = etat_d_ava
841                 apres_d_raj = pp*2 +1
842                 avant_a_raj = pp*2
843                 apres_a_raj = etat_a_pro
844                 poids_m_d_ava = poids1
845                 poids_m_d_pro = poids2
846                 poids_m_a_ava = poids2
847                 poids_m_a_pro = poids3
848                 ch_avant_d_raj = Trajet_raj_d1
849                 ch_apres_d_raj = Trajet_raj_d2
850                 ch_avant_a_raj = Trajet_raj_d2
851                 ch_apres_a_raj = Trajet_raj_a1
852                 min_cote_a_cote = True
853             if etat_a_pro == None: # si on est cote a cote est que le prochaine est None : break
854                 break
855             etat_a_ava = etat_a_pro

```

```

856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
etat_a_pro = Ordre[bd][etat_a_pro][1][0]
etat_d_pro = etat_a_ava
# attention a bien modifier le depart

cote_a_cote = False
else:
    if etat_a_pro == None and etat_d_ava == None: # tout après il y a rien et tout avant il y a le bus
        poids_raj_d1,Trajet_raj_d1 = DARP.A_star(self,self.S[bd],self.D[pp])
        poids_raj_d2,Trajet_raj_d2 = DARP.A_star(self,self.D[pp],trfm_etat_to_sommet(etat_d_pro))
        poids_raj_a1,Trajet_raj_a1 = DARP.A_star(self,trfm_etat_to_sommet(etat_a_ava),self.A[pp])
        poids_raj_a2,Trajet_raj_a2 = (0,[])
        tps_nv_b_raj = Tdb[bd] - Ordre[bd][etat_d_pro][0][2] + poids_raj_d1 + poids_raj_d2 + poids_raj_a1

    elif etat_d_ava == None: # avant il y a le bus
        poids_raj_d1,Trajet_raj_d1 = DARP.A_star(self,self.S[bd],self.D[pp])
        poids_raj_d2,Trajet_raj_d2 = DARP.A_star(self,self.D[pp],trfm_etat_to_sommet(etat_d_pro))
        poids_raj_a1,Trajet_raj_a1 = DARP.A_star(self,trfm_etat_to_sommet(etat_a_ava),self.A[pp])
        poids_raj_a2,Trajet_raj_a2 = DARP.A_star(self,self.A[pp],trfm_etat_to_sommet(etat_a_pro))
        tps_nv_b_raj = Tdb[bd] - ( Ordre[bd][etat_d_pro][0][2] + Ordre[bd][etat_a_pro][0][2] ) + poids_raj_d1 +
                           poids_raj_d2 + poids_raj_a1 + poids_raj_a2

    elif etat_a_pro == None: # tout après il y a rien
        poids_raj_d1,Trajet_raj_d1 = DARP.A_star(self,trfm_etat_to_sommet(etat_d_ava),self.D[pp])
        poids_raj_d2,Trajet_raj_d2 = DARP.A_star(self,self.D[pp],trfm_etat_to_sommet(etat_d_pro))
        poids_raj_a1,Trajet_raj_a1 = DARP.A_star(self,trfm_etat_to_sommet(etat_a_ava),self.A[pp])
        poids_raj_a2,Trajet_raj_a2 = (0,[])
        tps_nv_b_raj = Tdb[bd] - ( Ordre[bd][etat_d_ava][1][2] ) + poids_raj_d1 + poids_raj_d2 + poids_raj_a1

    else:
        poids_raj_d1,Trajet_raj_d1 = DARP.A_star(self,trfm_etat_to_sommet(etat_d_ava),self.D[pp])
        poids_raj_d2,Trajet_raj_d2 = DARP.A_star(self,self.D[pp],trfm_etat_to_sommet(etat_d_pro))
        poids_raj_a1,Trajet_raj_a1 = DARP.A_star(self,trfm_etat_to_sommet(etat_a_ava),self.A[pp])
        poids_raj_a2,Trajet_raj_a2 = DARP.A_star(self,self.A[pp],trfm_etat_to_sommet(etat_a_pro))
        tps_nv_b_raj = Tdb[bd] - ( Ordre[bd][etat_d_ava][1][2] + Ordre[bd][etat_a_pro][0][2] ) + poids_raj_d1 +
                           poids_raj_d2 + poids_raj_a1 + poids_raj_a2

if tps_nv_b_raj < minn:
    minn = tps_nv_b_raj
    avant_d_raj = etat_d_ava
    apres_d_raj = etat_d_pro
    avant_a_raj = etat_a_ava
    apres_a_raj = etat_a_pro
    poids_m_d_ava = poids_raj_d1
    poids_m_d_pro = poids_raj_d2
    poids_m_a_ava = poids_raj_a1
    poids_m_a_pro = poids_raj_a2
    ch_avant_d_raj = Trajet_raj_d1
    ch_apres_d_raj = Trajet_raj_d2

```

```

903         ch_avant_a_raj = Trajet_raj_a1
904         ch_apres_a_raj = Trajet_raj_a2
905         min_cote_a_cote = False
906         if etat_a_pro == None: # on fait un tour de plus
907             break
908         etat_a_ava = etat_a_pro
909         etat_a_pro = Ordre[bd][etat_a_pro][1][0]
910
911
912         if cote_a_cote and etat_a_pro == None: # le prochain d est l'arrive
913             break
914         etat_d_ava = etat_d_pro
915         etat_d_pro = pp*2 +1 #ok
916         etat_a_ava = pp*2
917         etat_a_pro = Ordre[bd][etat_d_ava][1][0]
918
919
920
921     if condition(bpp,bd,tps_nv_b_enl,tps_nv_b_raj,temperature):
922         # =====
923         # on enleve la personne de bpp
924
925         # on enregistre les nouveaux temps et nbs de personne
926         Tdb[bpp] = tps_nv_b_enl
927         Tdb[bd] = tps_nv_b_raj
928         Ppb[bpp] += -1
929         Ppb[bd] += 1
930         Bu[pp] = bd
931
932         if cote_a_cote_enl:
933             if avant_d_enl == None and apres_a_enl == None:
934                 First[bpp] = -1
935                 Ordre[bpp].clear() # plus rien
936
937             elif avant_d_enl == None: # avant il y a le bus
938                 First[bpp] = apres_a_enl
939                 Ordre[bpp][apres_a_enl] = [(None,Trajet_a_enl,poids_a_enl),Ordre[bpp][apres_a_enl][1]]
940
941             elif apres_a_enl == None: # apres il y a rie,
942                 Ordre[bpp][avant_d_enl] = [Ordre[bpp][avant_d_enl][0],(None,None,None)]
943
944             else:
945                 Ordre[bpp][avant_d_enl] = [Ordre[bpp][avant_d_enl][0],(apres_a_enl,Trajet_d_enl,poids_d_enl)]
946                 Ordre[bpp][apres_a_enl] = [(avant_d_enl,Trajet_d_enl,poids_d_enl),Ordre[bpp][apres_a_enl][1]]
947
948         else:
949             if avant_d_enl == None and apres_a_enl == None:

```

```

950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
# depart
First[bpp] = apres_d_enl
Ordre[bpp][apres_d_enl] = [(None,Trajet_d_enl,poids_d_enl),Ordre[bpp][apres_d_enl][1]]
# arrive
Ordre[bpp][avant_a_enl] = [Ordre[bpp][avant_a_enl][0],(None,None,None)]

elif avant_d_enl == None: # avant il y a le bus
# depart
First[bpp] = apres_d_enl
Ordre[bpp][apres_d_enl] = [(None,Trajet_d_enl,poids_d_enl),Ordre[bpp][apres_d_enl][1]]
# arrive
Ordre[bpp][avant_a_enl] = [Ordre[bpp][avant_a_enl][0],(apres_a_enl,Trajet_a_enl,poids_a_enl)]
Ordre[bpp][apres_a_enl] = [(avant_a_enl,Trajet_a_enl,poids_a_enl),Ordre[bpp][apres_a_enl][1]]
elif apres_a_enl == None: # apres il y a rien
# depart
Ordre[bpp][avant_d_enl] = [Ordre[bpp][avant_d_enl][0],(apres_d_enl,Trajet_d_enl,poids_d_enl)]
Ordre[bpp][apres_d_enl] = [(avant_d_enl,Trajet_d_enl,poids_d_enl),Ordre[bpp][apres_d_enl][1]]
# arrive
Ordre[bpp][avant_a_enl] = [Ordre[bpp][avant_a_enl][0],(None,None,None)]
else:
# depart
Ordre[bpp][avant_d_enl] = [Ordre[bpp][avant_d_enl][0],(apres_d_enl,Trajet_d_enl,poids_d_enl)]
Ordre[bpp][apres_d_enl] = [(avant_d_enl,Trajet_d_enl,poids_d_enl),Ordre[bpp][apres_d_enl][1]]
# arrive
Ordre[bpp][avant_a_enl] = [Ordre[bpp][avant_a_enl][0],(apres_a_enl,Trajet_a_enl,poids_a_enl)]
Ordre[bpp][apres_a_enl] = [(avant_a_enl,Trajet_a_enl,poids_a_enl),Ordre[bpp][apres_a_enl][1]]
# =====
# on rajoute la personne dans le bus bd

if min_cote_a_cote:
    if apres_a_raj == None and avant_d_raj == None: # tout après il y a rien et tout avant il y a le bus
        First[bd] = pp*2 # le 1er c'est le départ
        # rien
        Ordre[bd][pp*2] = [(None,ch_avant_d_raj,poids_m_d_ava),(apres_d_raj,ch_apres_d_raj,poids_m_d_pro)]
        Ordre[bd][pp*2 +1] = [(avant_a_raj,ch_avant_a_raj,poids_m_a_ava),(None,None,None)]
        #rien

    elif avant_d_raj == None: # avant il y a le bus
        First[bd] = pp*2 # le 1er c'est le départ
        # rien
        Ordre[bd][pp*2] = [(None,ch_avant_d_raj,poids_m_d_ava),(apres_d_raj,ch_apres_d_raj,poids_m_d_pro)]
        Ordre[bd][pp*2 +1] = [(avant_a_raj,ch_avant_a_raj,poids_m_a_ava),(apres_a_raj,ch_apres_a_raj,poids_m_a_pro)]
        Ordre[bd][apres_a_raj] = [(2*pp+1,ch_apres_a_raj,poids_m_a_pro),Ordre[bd][apres_a_raj][1]]

    elif apres_a_raj == None: # tout après il y a rien

```

```

997 Ordre[bd][avant_d_raj] = [Ordre[bd][avant_d_raj][0],(pp*2,ch_avant_d_raj,poids_m_d_ava)]
998 Ordre[bd][pp*2] = [(avant_d_raj,ch_avant_d_raj,poids_m_d_ava),(apres_d_raj,ch_apres_d_raj,poids_m_d_pro)]
999
1000
1001
1002
1003 Ordre[bd][pp*2 +1] = [(avant_a_raj,ch_avant_a_raj,poids_m_a_ava),(None,None,None)]
1004 #rien
1005
1006 else:
1007     Ordre[bd][avant_d_raj] = [Ordre[bd][avant_d_raj][0],(pp*2,ch_avant_d_raj,poids_m_d_ava)]
1008     Ordre[bd][pp*2] = [(avant_d_raj,ch_avant_d_raj,poids_m_d_ava),(apres_d_raj,ch_apres_d_raj,poids_m_d_pro)]
1009
1010     Ordre[bd][pp*2 +1] = [(avant_a_raj,ch_avant_a_raj,poids_m_a_ava),(apres_a_raj,ch_apres_a_raj,poids_m_a_pro)]
1011     Ordre[bd][apres_a_raj] = [(2*pp+1,ch_apres_a_raj,poids_m_a_pro),Ordre[bd][apres_a_raj][1]]
1012
1013 else:
1014     if apres_a_raj == None and avant_d_raj == None: # tout après il y a rien et tout avant il y a le bus
1015         First[bd] = pp*2 # le 1er c'est le départ
1016         # rien
1017         Ordre[bd][pp*2] = [(None,ch_avant_d_raj,poids_m_d_ava),(apres_d_raj,ch_apres_d_raj,poids_m_d_pro)]
1018         Ordre[bd][apres_d_raj] = [(2*pp,ch_apres_d_raj,poids_m_d_pro),Ordre[bd][apres_d_raj][1]]
1019         # l arrive
1020         Ordre[bd][avant_a_raj] = [Ordre[bd][avant_a_raj][0],(2*pp+1,ch_avant_a_raj,poids_m_a_ava)]
1021         Ordre[bd][pp*2 +1] = [(avant_a_raj,ch_avant_a_raj,poids_m_a_ava),(None,None,None)]
1022         #rien
1023     elif avant_d_raj == None: # avant il y a le bus
1024         First[bd] = pp*2 # le 1er c'est le départ
1025         # rien
1026         Ordre[bd][pp*2] = [(None,ch_avant_d_raj,poids_m_d_ava),(apres_d_raj,ch_apres_d_raj,poids_m_d_pro)]
1027         Ordre[bd][apres_d_raj] = [(2*pp,ch_apres_d_raj,poids_m_d_pro),Ordre[bd][apres_d_raj][1]]
1028         # l arrive
1029         Ordre[bd][avant_a_raj] = [Ordre[bd][avant_a_raj][0],(2*pp+1,ch_avant_a_raj,poids_m_a_ava)]
1030         Ordre[bd][pp*2 +1] = [(avant_a_raj,ch_avant_a_raj,poids_m_a_ava),(apres_a_raj,ch_apres_a_raj,poids_m_a_pro)]
1031         Ordre[bd][apres_a_raj] = [(2*pp+1,ch_apres_a_raj,poids_m_a_pro),Ordre[bd][apres_a_raj][1]]
1032     elif apres_a_raj == None: # tout après il y a rien
1033         Ordre[bd][avant_d_raj] = [Ordre[bd][avant_d_raj][0],(pp*2,ch_avant_d_raj,poids_m_d_ava)]
1034         Ordre[bd][pp*2] = [(avant_d_raj,ch_avant_d_raj,poids_m_d_ava),(apres_d_raj,ch_apres_d_raj,poids_m_d_pro)]
1035         Ordre[bd][apres_d_raj] = [(2*pp,ch_apres_d_raj,poids_m_d_pro),Ordre[bd][apres_d_raj][1]]
1036         # l arrive
1037         Ordre[bd][avant_a_raj] = [Ordre[bd][avant_a_raj][0],(2*pp+1,ch_avant_a_raj,poids_m_a_ava)]
1038         Ordre[bd][pp*2 +1] = [(avant_a_raj,ch_avant_a_raj,poids_m_a_ava),(None,None,None)]
1039         #rien
1040     else:
1041         Ordre[bd][avant_d_raj] = [Ordre[bd][avant_d_raj][0],(pp*2,ch_avant_d_raj,poids_m_d_ava)]
1042         Ordre[bd][pp*2] = [(avant_d_raj,ch_avant_d_raj,poids_m_d_ava),(apres_d_raj,ch_apres_d_raj,poids_m_d_pro)]
1043         Ordre[bd][apres_d_raj] = [(2*pp,ch_apres_d_raj,poids_m_d_pro),Ordre[bd][apres_d_raj][1]]
1044
1045 return 0

```

```

1102 def recherche_cts_t0(M,V,C,H,m,s,S,n,D,A,ca):
1103     #(t0, tf, alpha)
1104     a = 0.0005
1105     b = 0.00000001
1106     ite = 10
1107     nbs = 1
1108     pas = (b-a)/ite
1109
1110     for i in range(ite):
1111         sys = DARP(M,V,C,H,m,s,S,n,D,A,ca,0.001,a+pas*i,0.855)
1112         rend = 0
1113         for _ in range(nbs):
1114             rend += sys.multi_bus_attribution_recui_simule()
1115         rend = rend / nbs
1116         print( a+pas*i, " - ",rend)
1117
1118
1119 def recherche_cts_alpha(M,V,C,H,m,s,S,n,D,A,ca):
1120     #(t0, tf, alpha)
1121     a = 0.85
1122     b = 0.90
1123     ite = 20
1124     nbs = 5
1125     pas = (b-a)/ite
1126
1127     for i in range(ite):
1128         sys = DARP(M,V,C,H,m,s,S,n,D,A,ca,0.01,0.00120759,a+pas*i)
1129         rend = 0
1130         for _ in range(nbs):
1131             rend += sys.multi_bus_attribution_recui_simule()
1132         rend = rend / nbs
1133         print( a+pas*i, " - ",rend)

```

```

1135 m = 270
1136 M = inject_data_distance("data_distance.txt", 270)
1137 V = inject_data_liste("data_listesommet.txt",270)
1138 H = inject_data_heuristique("data_heuristique.txt",270)
1139 C = inject_data_color("data_color.txt",270)
1140
1141 s = 2
1142 S = [random.randint(0,m-1) for _ in range(s)]
1143
1144 n = 6
1145 D = [random.randint(0,m-1) for _ in range(n)]
1146 A = [random.randint(0,m-1) for _ in range(n)]
1147
1148
1149
1150 t1 = time.time()
1151 sys = DARP(M,V,C,H,m,s,S,n,D,A,0.01,0.001,0.00025,0.855)
1152 sys.aff_sommet = False
1153 sys.contraste_bus = 0.5
1154 rend = sys.multi_bus_attribution_naive()
1155 print(rend)
1156 print(time.time() - t1)
1157 sys.affiche()
1158
1159 t1 = time.time()
1160 sys = DARP(M,V,C,H,m,s,S,n,D,A,0.01,0.001,0.00025,0.855)
1161 sys.aff_sommet = False
1162 sys.contraste_bus = 0.5
1163 rend = sys.multi_bus_attribution_recui_simule()
1164 print(rend)
1165 print(time.time() - t1)
1166 sys.affiche()

```

```

# hypothèse : graph de la ville est connexe non orienté et pondéré
def inject_data_distance(file,size):
    # file : chemin d'accès au fichier correspondant
    # size : nombre de sommet de la ville.
    f = open(file,"r")
    M = [[] for _ in range(size)]
    for i in range(size):
        ligne = f.readline()
        tab_ligne = ligne.split(", ")
        tab_ligne[0] = tab_ligne[0][1:]
        tab_ligne[-1] = tab_ligne[-1].split("]")[0]
        for j in range(size):
            if tab_ligne[j] == "inf":
                M[i].append(float('inf'))
            else:
                M[i].append(float(tab_ligne[j]))
    return M
33

def inject_data_heuristique(file,size):
    # file : chemin d'accès au fichier correspondant
    # size : nombre de sommet de la ville.
    f = open(file,"r")
    M = [[] for _ in range(size)]
    for i in range(size):
        ligne = f.readline()
        tab_ligne = ligne.split(", ")
        tab_ligne[0] = tab_ligne[0][1:] # on enlève le 1 er char
        tab_ligne[-1] = tab_ligne[-1].split("]")[0] # et les derniers char
        for j in range(size):
            M[i].append(float(tab_ligne[j]))
    return M
47

def inject_data_liste(file,size):
    # file : chemin d'accès au fichier correspondant
    # size : nombre de sommet de la ville.
    f = open(file,"r")
    M = []
    ligne = f.readline()
    tab_ligne = ligne.split(", ")
    tab_ligne[0] = tab_ligne[0][2:] # on enlève les 2 premier char
    tab_ligne[-1] = tab_ligne[-1].split("]")[0] # et les derniers char
    for j in range(size):
        ctt = tab_ligne[j].split(", ")
        M.append( (int(ctt[0]), int(ctt[1])) )
    return M
60

```

```

62 def inject_data_color(file,size):
63     # file : chemin d'accès au fichier correspondant
64     # size : nombre de sommet de la ville.
65     f = open(file,"r")
66     M = [[] for _ in range(size)]
67     for i in range(size):
68         ligne = f.readline()
69         tab_ligne = ligne.split(", ")
70         tab_ligne[0] = tab_ligne[0][1:] # on enleve le 1 er char
71         tab_ligne[-1] = tab_ligne[-1].split("]")[0] # et les derniers char
72         for j in range(size):
73             M[i].append(int(tab_ligne[j]))
74     return M
75
76 def matrice_to_liste(M,m):
77     # M : matrice d'adjacence
78     N = [[] for _ in range(m)]
79     for i in range(m):
80         for j in range(m):
81             if M[i][j] != float('inf') and i!=j:
82                 N[i].append((j,M[i][j]))
83     return N
84 # liste d'adjacence:
85 #   N = [ [ (sommet suivant le sommet i note j, poids de l'arc i j) , ...] , ...]
86 #       N[i] les sommets adjacents a i
87
88 def cmp1(e):
89     return e[1]
90
91 def cmp2(e):
92     return e[3]

```

Logiciel de création de graphe

```

1 import pygame
2 SIZEx = 1800
3 SIZEy = 800
4 RAYON = 10
5 COLOR = [(255,0,255),(0,255,255),(0,255,0),(255,255,0),(155,100,0),(255,0,0)]
6
7 bg = pygame.image.load("map_gre2.png")
8
9 def distance(a,b):
10     return ((a[0] - b[0])**2 + (a[1] - b[1])**2)**(1/2)
11
12 def rendu(screen,event,nbs_sommet,police,L,C):
13     global SIZE,RAYON,COLOR,bg
14     screen.blit(bg,(0,0)) #reset all entitie
15
16     for i in range(nbs_sommet): # affiche les sommets
17         pygame.draw.circle(screen,(255,255,255),L[i],RAYON,width=1)
18
19         image_text = police.render(str(i),1,(255,255,255))
20         screen.blit(image_text,(L[i][0]-6,L[i][1]-10))
21
22     for i in range(nbs_sommet):
23         for j in range(i):
24             if C[i][j] != 0:
25                 pygame.draw.line(screen,COLOR[C[i][j]-1],L[i],L[j])
26
27
28     pygame.display.update()
29
30 def main():
31     global SIZE,RAYON,COLOR,bg
32     pygame.init()
33     pygame.display.set_caption("TIEPE")
34     screen = pygame.display.set_mode((SIZEx,SIZEy))
35
36     screen.blit(bg,(0,0))
37
38     M = [[[]]] # temps
39     H = [[[]]] # distance a vole d'oiseau
40     C = [[[]]] # affichage des routes, en couleur 1 , 2 , ... 6
41
42     running = True
43     L=[] # position des sommets
44     nbs_sommet = 0
45
46     ind_arrete = -1
47     police = pygame.font.SysFont("monospace",20)
48     pygame.display.update()

```

```

49 while running:
50     for event in pygame.event.get():
51         if event.type == pygame.QUIT:
52             running = False
53         if event.type == pygame.MOUSEBUTTONDOWN:
54             L.append(event.pos)
55             nbs_sommet += 1
56             if nbs_sommet == 2:
57                 M = [[0,float('inf')], [float('inf'),0]]
58                 H = [[0,distance(L[0],L[1])], [distance(L[0],L[1]),0]]
59                 C = [[0,0], [0,0]]
60             else:
61                 M.append([float('inf') if i != (nbs_sommet-1) else 0 for i in range(nbs_sommet)])
62                 H.append([float('inf') if i != (nbs_sommet-1) else 0 for i in range(nbs_sommet)])
63                 C.append([0 if i != (nbs_sommet-1) else 0 for i in range(nbs_sommet)])
64             for i in range(nbs_sommet-1):
65                 M[i].append(float('inf'))
66                 C[i].append(0)
67                 dis = distance(L[i],L[-1])
68                 H[i].append(dis)
69                 H[-1][i] = dis
70
71             print()
72             print(L)
73             for el in M:
74                 print(el)
75             print("====")
76             for el in H:
77                 print(el)
78             print("====")
79             for el in C:
80                 print(el)
81             pygame.draw.circle(screen,(255,255,255),event.pos,RAYON,width=1)
82             image_text = police.render( str(nbs_sommet-1),1,(255,255,255))
83             screen.blit(image_text,(event.pos[0]-6,event.pos[1]-10))
84             pygame.display.update()
85         if event.type == pygame.KEYDOWN:
86             cc = 0
87             print(event)
88             if event.key == 109: # touche m enleve le dernier sommet
89                 del L[-1]
90                 for i in range(nbs_sommet):
91                     del H[i][-1]
92                     del M[i][-1]
93                     del C[i][-1]
94                     del C[-1]
95                     del H[-1]
96                     del M[-1]

```

```

97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
nbs_sommet -= 1
rendu(screen,event,nbs_sommet,police,L,C)
if event.key == 108: # touche l , enleve l'arrete selectionner
    s_d = input("sommet du depart de l'arrete ->")
    s_a = input("sommet de l'arrive de l'arrete ->")
    try:
        d = int(s_d)
        a = int(s_a)
        C[a][d] = 0
        C[d][a] = 0
        M[a][d] = float('inf')
        M[d][a] = float('inf')
        rendu(screen,event,nbs_sommet,police,L,C)
    except:
        print("erreur syntaxe")
if event.key == 107: # touche k , enleve sommet selectionner
    s_s = input("sommet ->")
    try:
        s = int(s_s)
        del L[s]
        for i in range(nbs_sommet):
            del H[i][s]
            del M[i][s]
            del C[i][s]
        del C[s]
        del H[s]
        del M[s]
        nbs_sommet -= 1
        rendu(screen,event,nbs_sommet,police,L,C)
    except:
        print('erreur syntaxe')
if event.key == 97: #a
    cc = 1
    vitesse = 30
if event.key == 122: #z
    cc = 2
    vitesse = 50
if event.key == 101: # e
    cc = 3
    vitesse = 70
if event.key == 114: #r
    cc = 4
    vitesse = 90
if event.key == 116: #t
    cc = 5
    vitesse = 110
if event.key == 121: #y
    cc = 6

```

```

145         vitesse = 130
146         if cc != 0:
147             ppos = pygame.mouse.get_pos()
148             for i in range(len(L)):
149                 dis = distance(ppos,L[i])
150                 if dis <= RAYON:
151                     if ind_arrete != (-1):
152                         dd = distance([ind_arrete],L[i])
153                         tps = dd / vitesse
154                         M[i][ind_arrete] = tps
155                         M[ind_arrete][i] = tps
156                         C[i][ind_arrete] = cc
157                         C[ind_arrete][i] = cc
158                         pygame.draw.line(screen,COLOR[cc-1],L[ind_arrete],L[i])
159
160             print()
161             print(L)
162             for el in M:
163                 print(el)
164             print("====")
165             for el in H:
166                 print(el)
167             ind_arrete = -1
168             pygame.draw.circle(screen,(255,255,255),(10,10),RAYON,width=1)
169             pygame.display.update()
170
171         else:
172             ind_arrete = i
173             pygame.draw.circle(screen,(0,0,0),(10,10),RAYON,width=1)
174             pygame.display.update()
175
176 main()

```

TSP

```

1 import math,copy,sys,random,time
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import pygame
5 SIZEx = 800
6 SIZEy = 800
7 #random.seed(15268)
8 pygame.init()
9 pygame.display.set_caption("TSP")
10 screen = pygame.display.set_mode((SIZEx,SIZEy))
11
12 class Graphe:
13     def __init__(self,graphe,T = -1, alpha = -1, stop_t = -1):
14         self.graphe = graphe
15         self.start = math.sqrt(7) if T == -1 else T
16         self.alpha = 0.995 if alpha == -1 else alpha
17         self.stop = 0.00000001 if stop_t == -1 else stop_t
18     def arcExist(self,i,j):
19         if self.graphe[i][j]!=None:
20             return True
21     def longueur(self,seq):
22         return Len(seq)
23     def distance(self,seq):
24         if self.longueur(seq)>1:
25             cout=0
26             for i in range(self.longueur(seq)-1):
27                 if self.arcExist(seq[i],seq[i+1]):
28                     cout+=self.graphe[seq[i]][seq[i+1]]
29             cout+=self.graphe[seq[-1]][seq[0]]
30             return cout
31         else:
32             return None
33     def proba(self,dist_parcours,dist_p):
34         return math.exp(-abs(dist_p-dist_parcours)/self.start)
35     def permutation(self,ordre):
36         global M
37         d = longueur(x,y,ordre)
38         d0 = d+1
39         it = 1
40         while d < d0 :
41             it += 1
42             d0 = d
43             for i in range(0,Len(ordre)-1) :
44                 for j in range(i+2,Len(ordre)):
45                     r = ordre[i:j].copy()
46                     r.reverse()
47                     ordre2 = ordre[:i] + r + ordre[j:]
48                     t = longueur(x,y,ordre2)

```

```

49
50
51         if t < d :
52             d = t
53             ordre = ordre2
54
55     return ordre
56
57 def recuit(self,parcours):
58     bestSolution=copy.copy(parcours)
59     bestCout=self.distance(parcours)
60     bestSolution.append(bestSolution[0])
61     print("parcours initial")
62
63     print(bestSolution)
64     print(bestCout)
65     affiche(bestSolution)
66     ct = 0
67     while self.start>self.stop:
68         ct += 1
69         if ct >10000:
70             print(self.start)
71             affiche(p)
72             ct = 0
73         p=copy.copy(parcours)
74         j=random.randint(2,len(p)-1)
75         i=random.randint(0,j-1)
76         p[i:(j+1)]=reversed(p[i:(j+1)])
77
78         if self.distance(p)<bestCout:
79             bestSolution=copy.copy(p)
80             bestCout=self.distance(p)
81             parcours=copy.copy(p)
82             bestSolution.append(bestSolution[0])
83
84         else:
85             if random.random()<self.proba(self.distance(parcours),self.distance(p)):
86                 parcours=copy.copy(p)
87
88             self.start *= self.alpha
89             print(bestSolution)
90             print(bestCout)
91             print('Solution finale')
92             affiche(bestSolution)
93
94
95 def affiche(Ordre):
96     global M
97     x = []
98     y = []
99     screen.fill((0,0,0))
100    pygame.draw.circle(screen,(250,250,250),(M[Ordre[0]-1][0],M[Ordre[0]-1][1]),5)

```

```

97    for i in range(1,n-1):
98        pygame.draw.circle(screen,(250,250,250),(M[Ordre[i]-1][0],M[Ordre[i]-1][1]),5)
99        pygame.draw.line(screen,(250,250,250),(M[Ordre[i-1]-1][0],M[Ordre[i-1]-1][1]),(M[Ordre[i]-1][0],M[Ordre[i]-1][1]))
100    pygame.display.update()
101    #pygame.time.wait(5)
102
103 def genere_graph(n):
104     M = [(random.randint(0,SIZEx),random.randint(0,SIZEy)) for i in range(n)]
105
106     H = [[0 for _ in range(n)] for _ in range(n)]
107     for i in range(n):
108         for j in range(i+1,n):
109             H[i][j] = int(math.sqrt(abs(M[i][0] - M[j][0])**2 + abs(M[i][1] - M[j][1])))
110             H[j][i] = int(math.sqrt(abs(M[i][0] - M[j][0])**2 + abs(M[i][1] - M[j][1])))
111     H[0] = 0
112     return M,H
113
114 global M,H
115
116 n = 100
117 M,H = genere_graph(n)
118
119 g=Graphe(H,math.sqrt(7),0.9995,0,1)
120 g.recuit([i for i in range(1,n)])
121 running = True
122 while running:
123     for event in pygame.event.get():
124         if event.type == pygame.QUIT:
125             running = False
126

```