

Répartition des automobilistes sur un réseau routier et paradoxe de Braess.

Sommaire :

- I – Introduction
- II – Cas de Toulouse
- III – Deux pistes pour améliorer l'approche
- IV – Approche dynamique
- V – Utilisations possible

I – Introduction : Principe & données

- Répartition des véhicules sur les trajets de A vers D
- Fluidité du réseau
- Ajout d'un pont
- Évolution du comportement des véhicules

Temps sur une route en fonction du nombre de voiture.

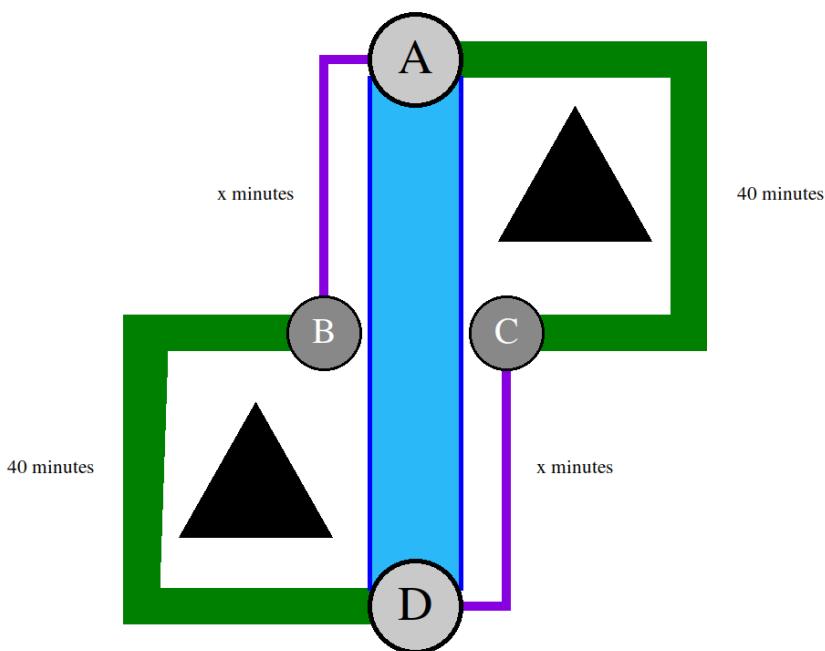
$$f_{ab}(x) = ax+b$$

a et b des coefficients caractéristiques.

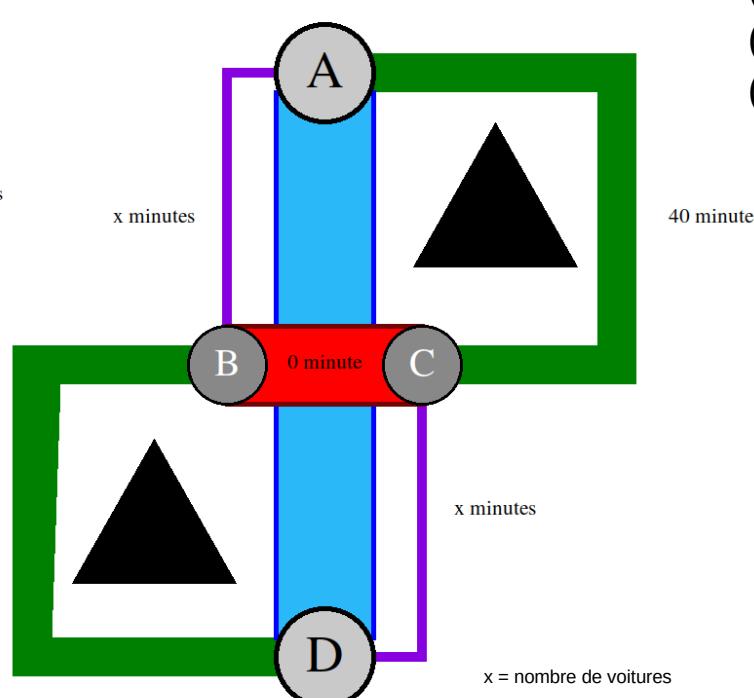
Liste des routes du réseau :

A vers B : (A,B,1,0)
A vers C : (A,C,0,40)
B vers C : (B,C,0,0)
B vers D : (B,D,0,40)
C vers D : (C,D,1,0)

Modélisation d'un réseau routier.



Modélisation après la construction du pont.



Liste des trajets possibles :

(A,B,D)
(A,C,D)
(A,B,C,D)
(A,C,B,D)

Matrice d'adjacence contenant les fonctions f_{ab}

	A	B	C	D
A		x	40	
B	x		0	40
C	40	0		x
D		40	x	

Algorithmes

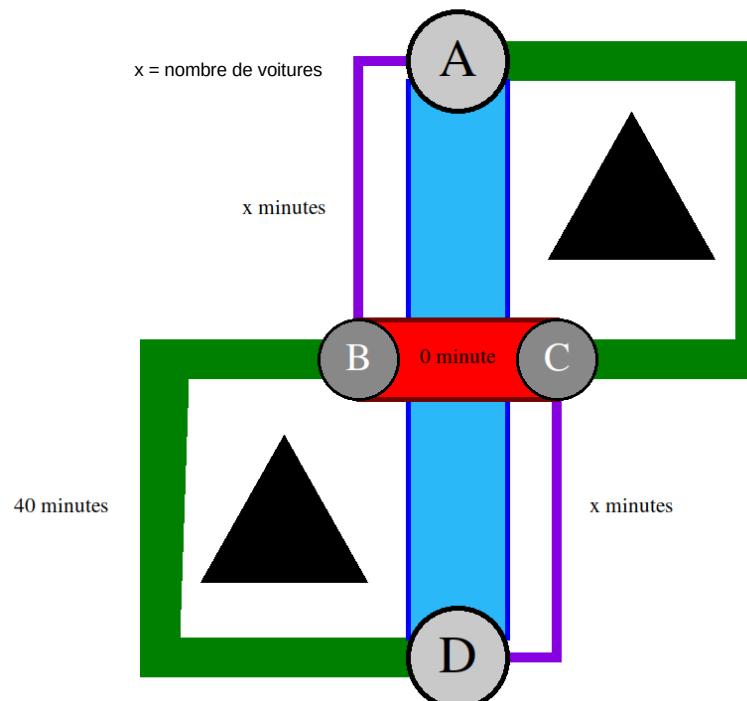
Modélisation après la construction du pont.

Répartition égoïste

- Principe
- Moyen de calcul
- Résultat

Pour 30 voitures :

Trajets	n° 1 : ABD	n° 2 : ACD	n° 3 : ABCD	n° 4 : ACBD
Nombre voitures	0	0	30	0



Temps moyen : 60 minutes

Répartition sociale

- Principe
- Moyen de calcul
- Résultat

Pour 30 voitures :

Trajets	n° 1 : ABD	n° 2 : ACD	n° 3 : ABCD	n° 4 : ACBD
Nombre voitures	0	0	20	10

Fonction temps :

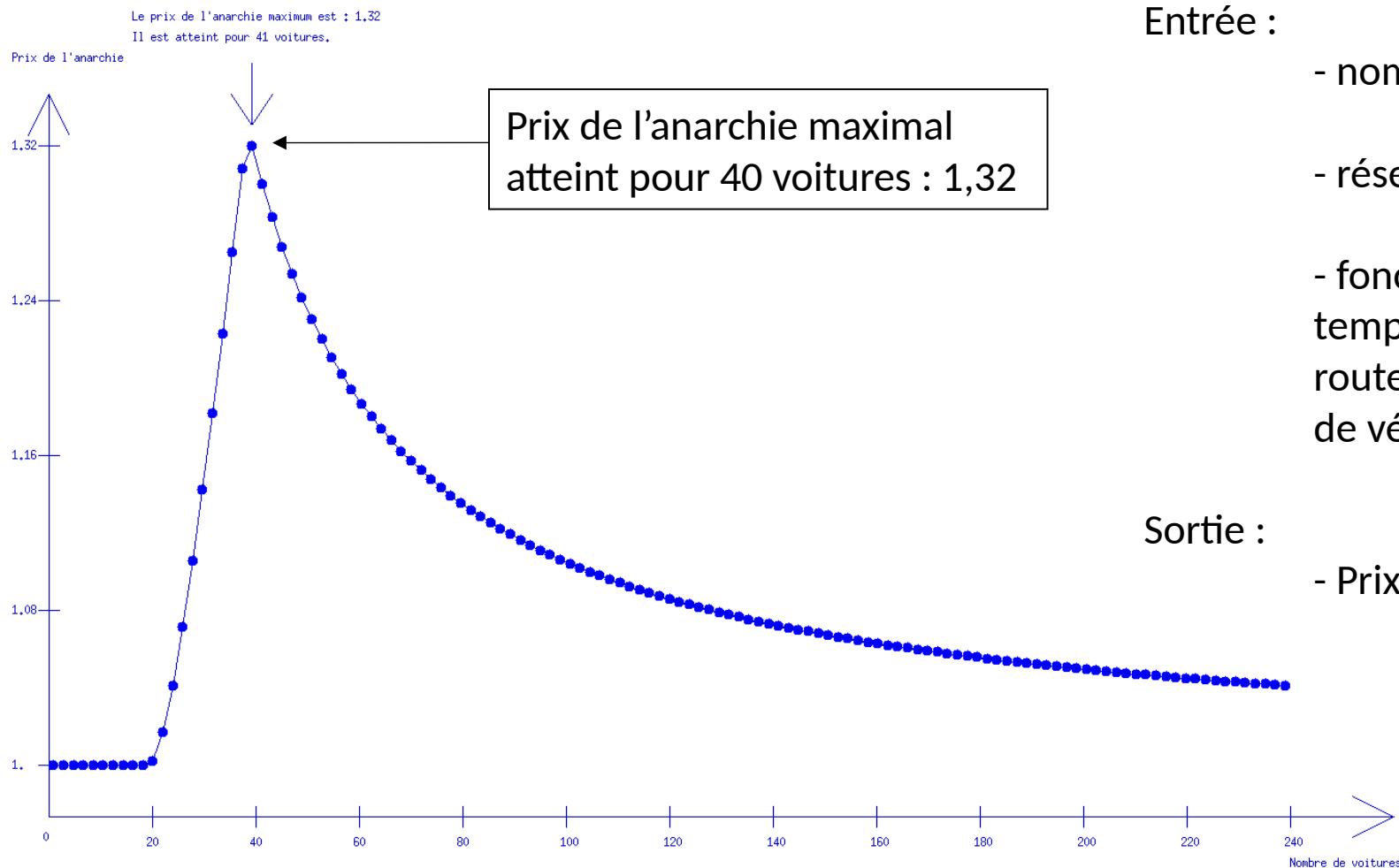
Donne, à partir d'une répartition, le temps moyen de parcours du réseau.

Temps moyen : 53,3 minutes

Résultats

Prix de l'anarchie après l'ajout d'un pont

Problème d'optimisation :

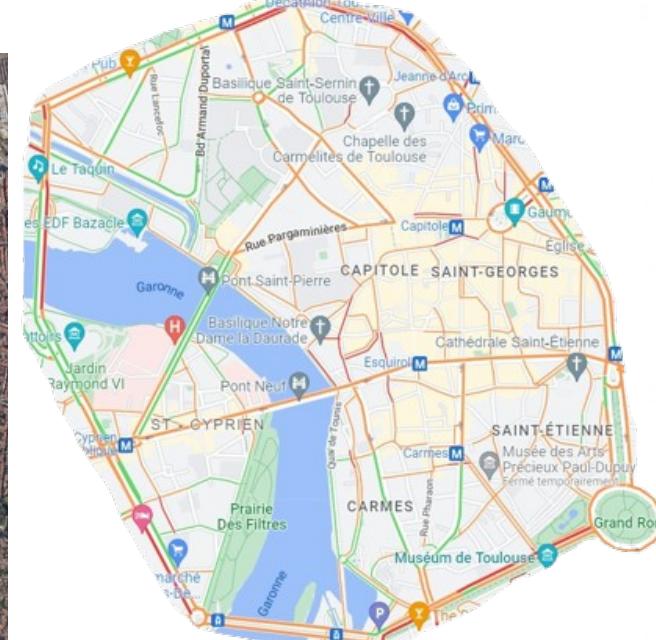


II - Cas de Toulouse : Modélisation



49 points du centre de Toulouse.

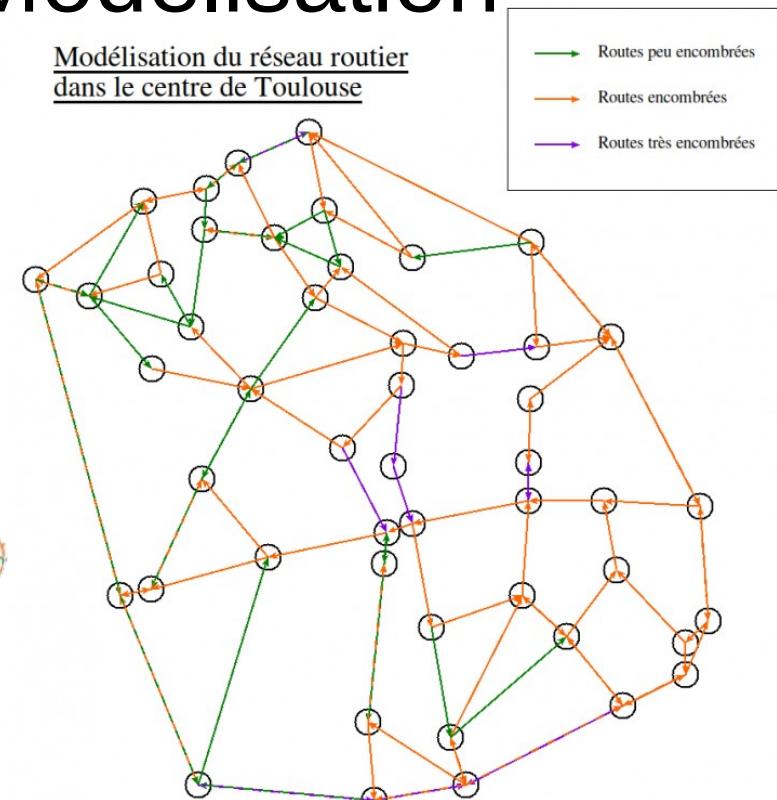
Fonction donnant le temps de traversé d'une route en fonction du nombre de voitures présentent sur la route.



109 routes les reliant.

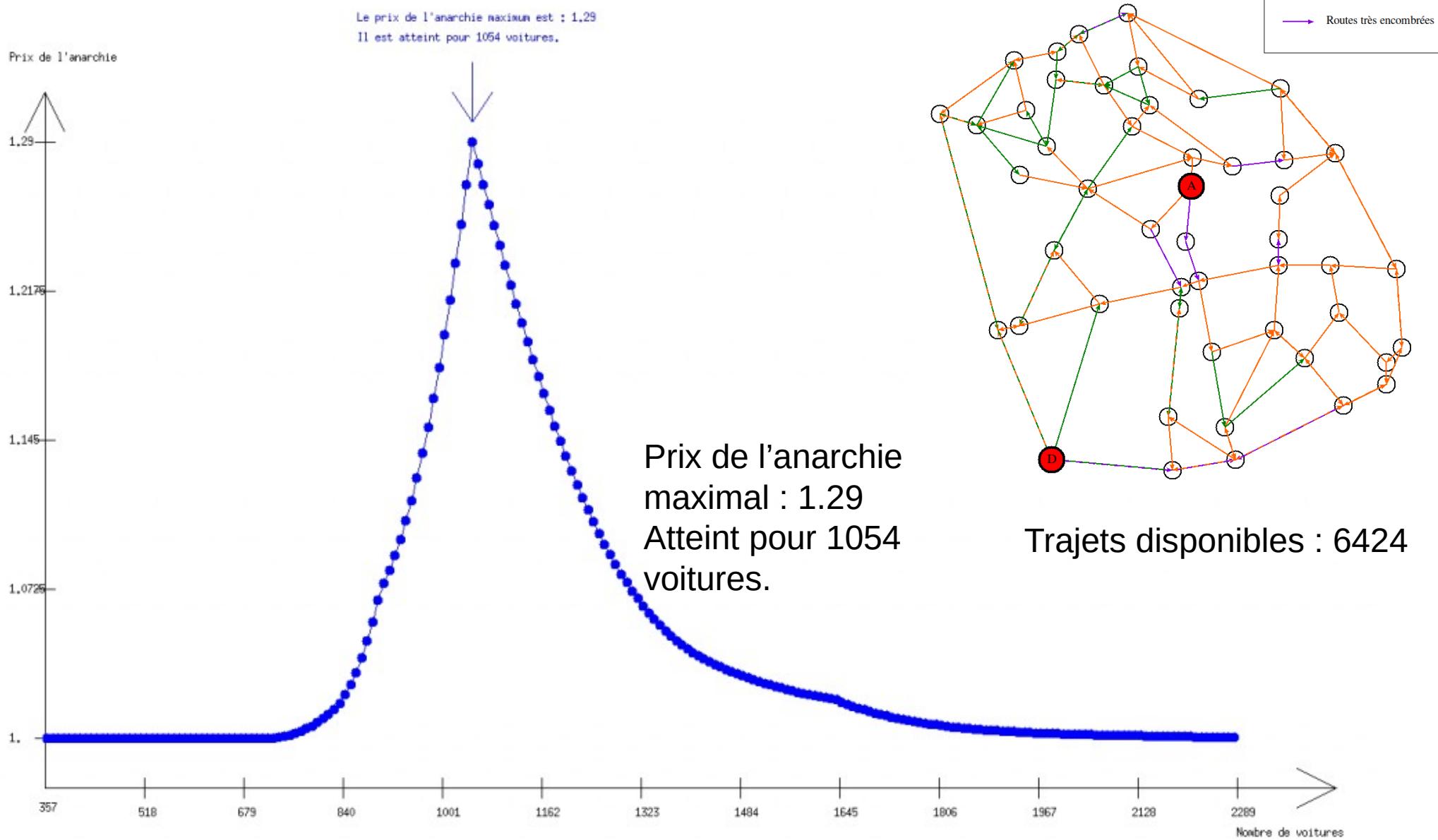
$$f_r(x) = a \times \left(1 + 0.2 \times \left(\frac{x}{b}\right)^{10}\right)$$

Modélisation du réseau routier dans le centre de Toulouse



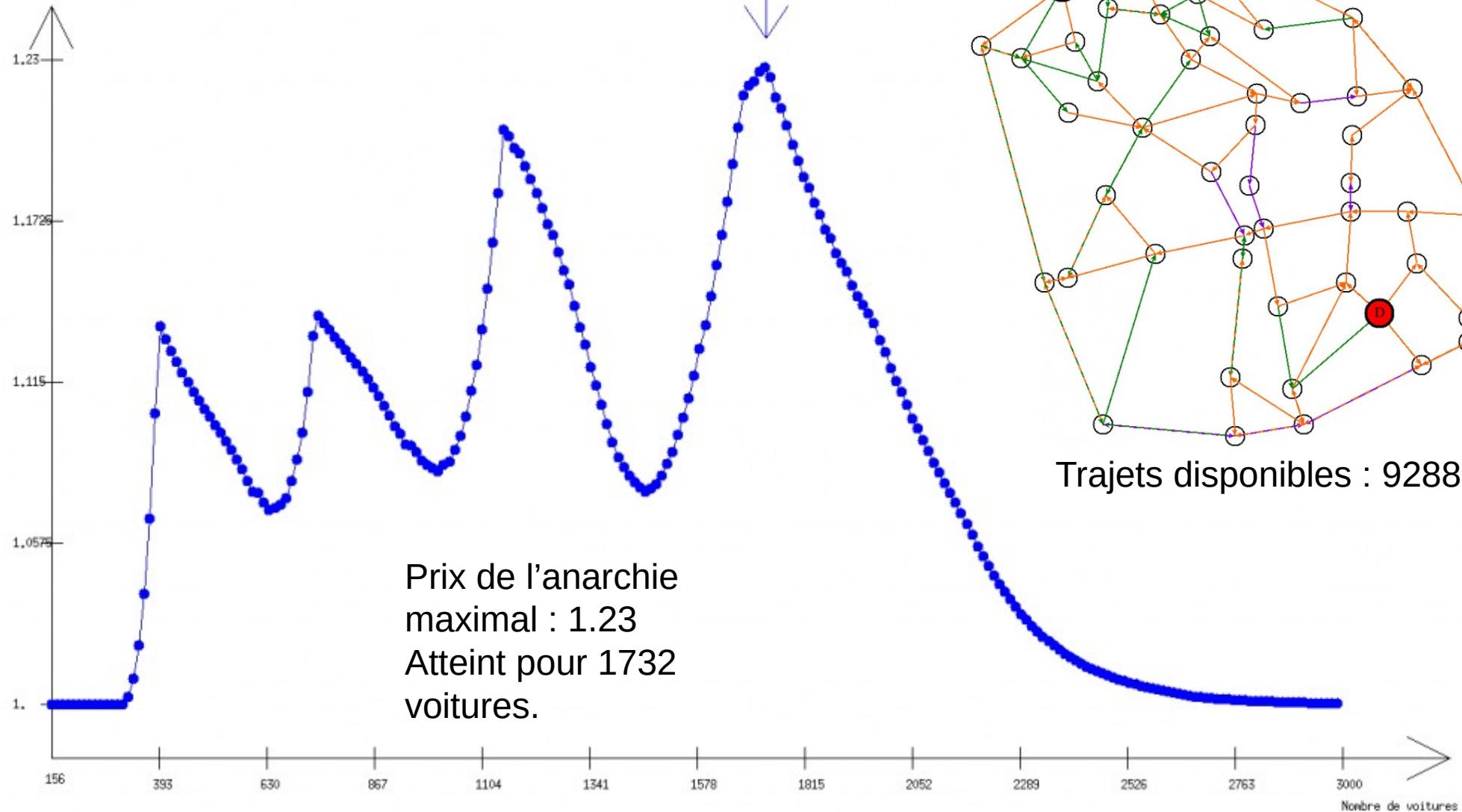
a = rapport entre taille de la route et vitesse maximale
b = capacité de la route

Du rond point de Catalogne à Pierre de Fermat

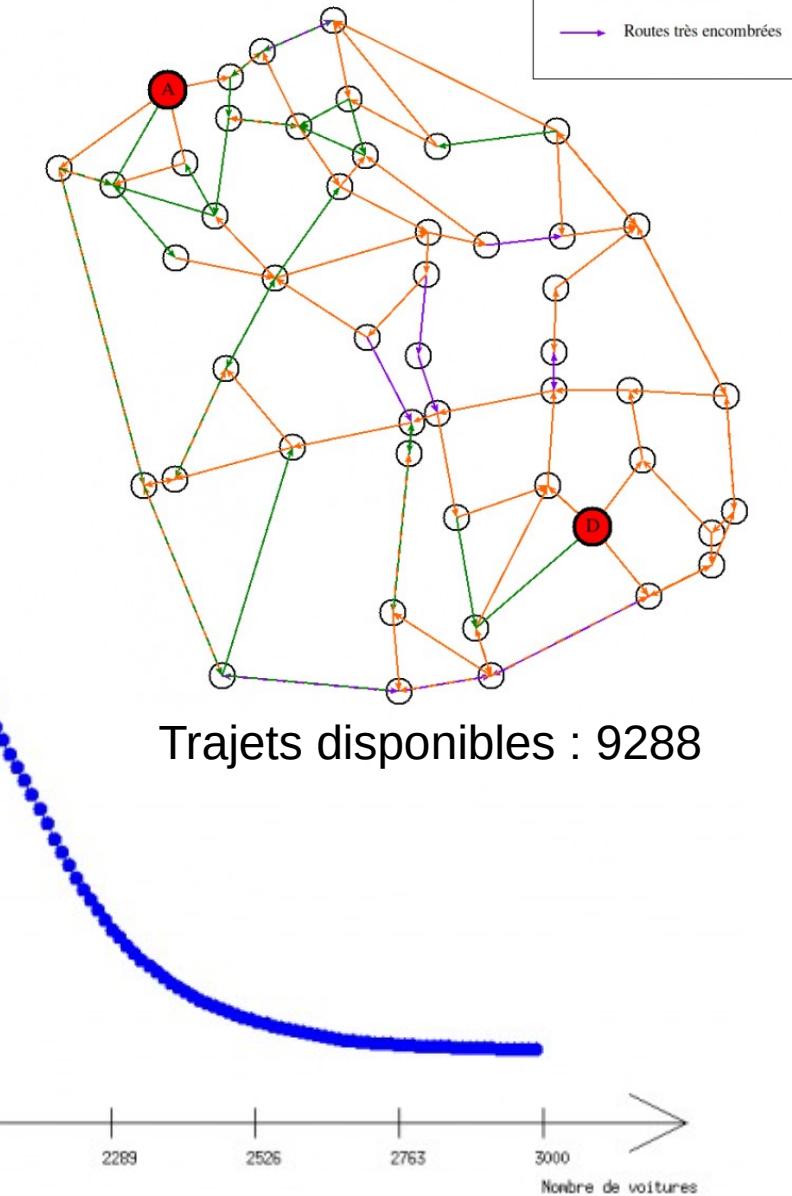


Du The Botanist Pub à la pharmacie du Jardin Royal

Prix de l'anarchie

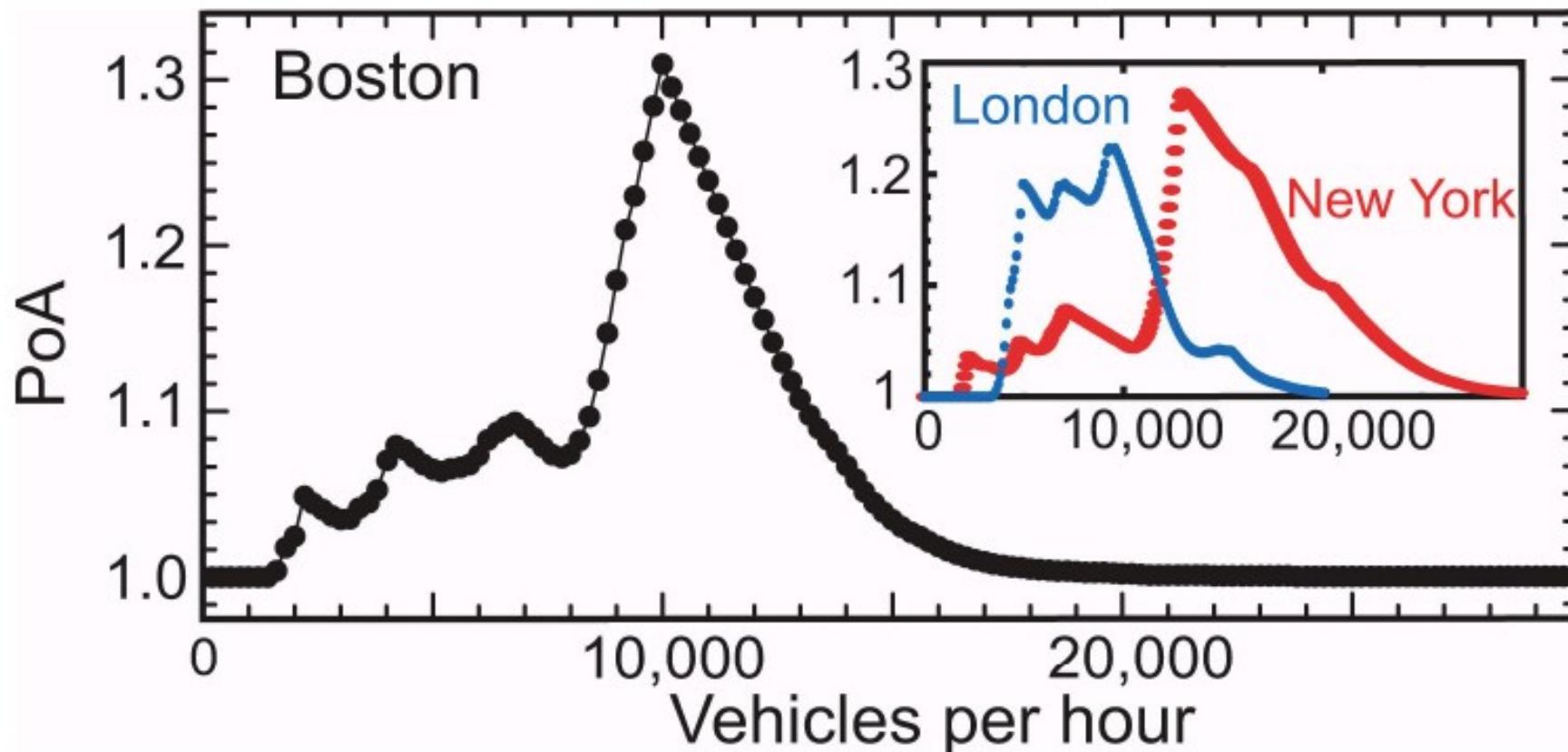


Modélisation du réseau routier dans le centre de Toulouse



Comparaison à des travaux existants

Young
Jeong
Gastner



Modèles choisis : Londres, New York et Boston

Légères différences sur la forme par rapport à nos résultats

Limites de l'approche

- Les voitures considérées collectivement.



- Fonction de calcul du temps : statique & lente → à améliorer.

- L'impact de chaque voiture est défini à l'avance uniquement suivant le trajet parcouru.

- Complexité des fonctions très moyennes.



Limites de l'approche

Approche qui est statique.
Différence avec la réalité.

Répartition égoïste

Trajets	n° 1 : ABD	n° 2 : ACD	n° 3 : ABCD	n° 4 : ACBD
Nombre voitures	0	0	21	0

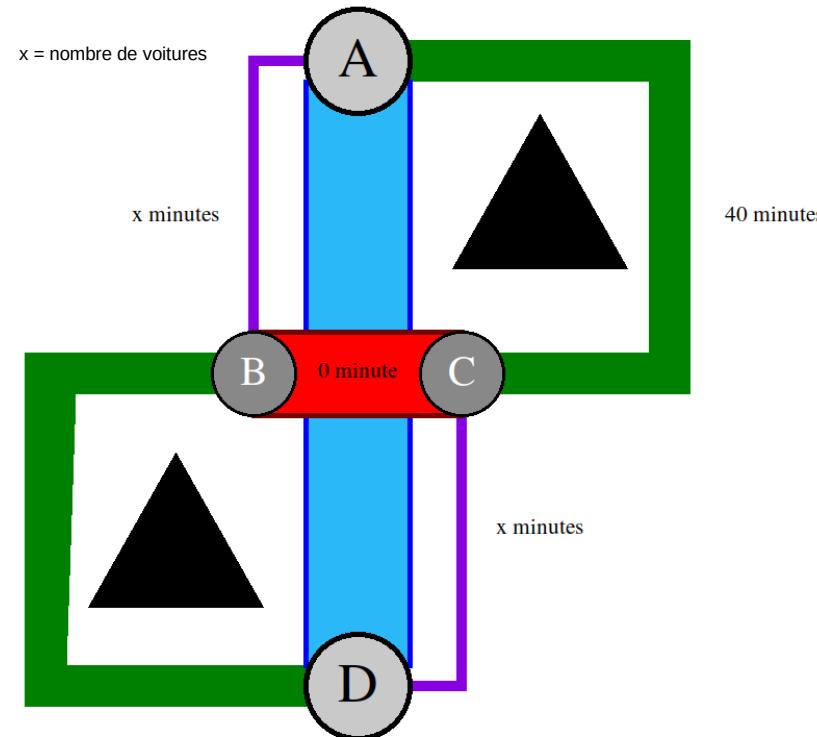
Temps moyen : 42 minutes

Répartition sociale

Trajets	n° 1 : ABD	n° 2 : ACD	n° 3 : ABCD	n° 4 : ACBD
Nombre voitures	0	0	20	1

Temps moyen : 41,9 minutes

Modélisation après la construction du pont.



En réalité

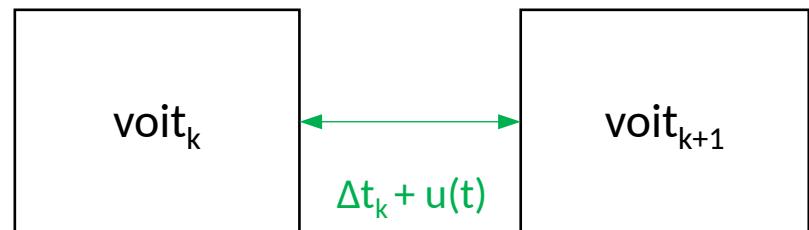
Trajets	n° 1 : ABD	n° 2 : ACD	n° 3 : ABCD	n° 4 : ACBD
Nombre voitures	0	1	20	0

Temps moyen : 40,05 minutes

III – Améliorer l'approche : matrice de priorité

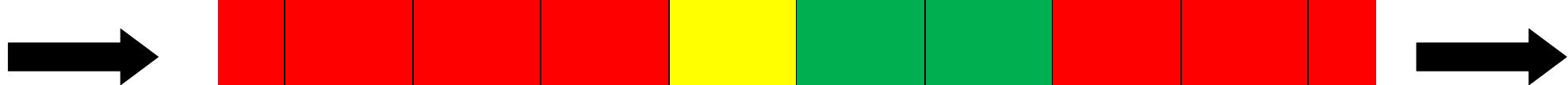
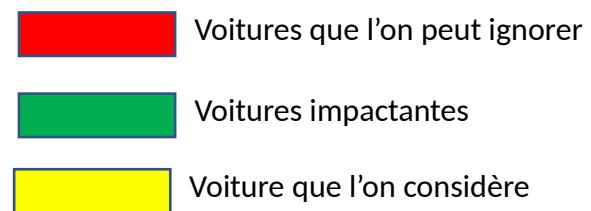
Considérer les voitures individuellement :

- Écart entre les voitures d'un même trajet



- Les voitures ne partent pas toutes en même temps

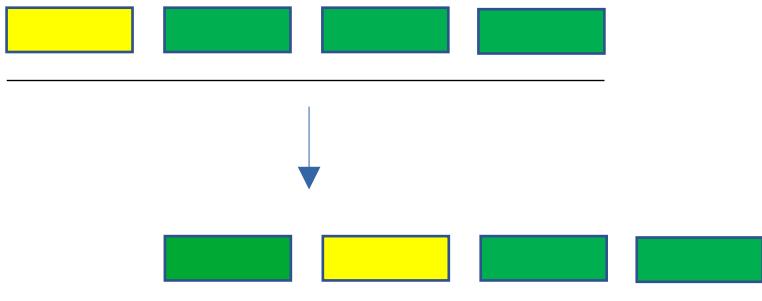
- On restreint le nombre de véhicule impactant le temps d'une voiture donnée



Limites et fonction répare

Ordre des voitures

Nombre de voiture sur une route



REPARÉ :

Entrée :

- M : Une matrice de priorité à réorganiser

Sortie :

- M après avoir été réorganisée

Tant que la fonction change la position d'au moins un véhicule :

Sur chaque route :

Calculer de nouveau les temps de passage

Trier les voitures par ordre de passage

Retourner M

Yellow rectangle
Voiture référence

Green rectangle
Voiture influençant le temps de parcours

—
Route

IV – Approche dynamique : Principe

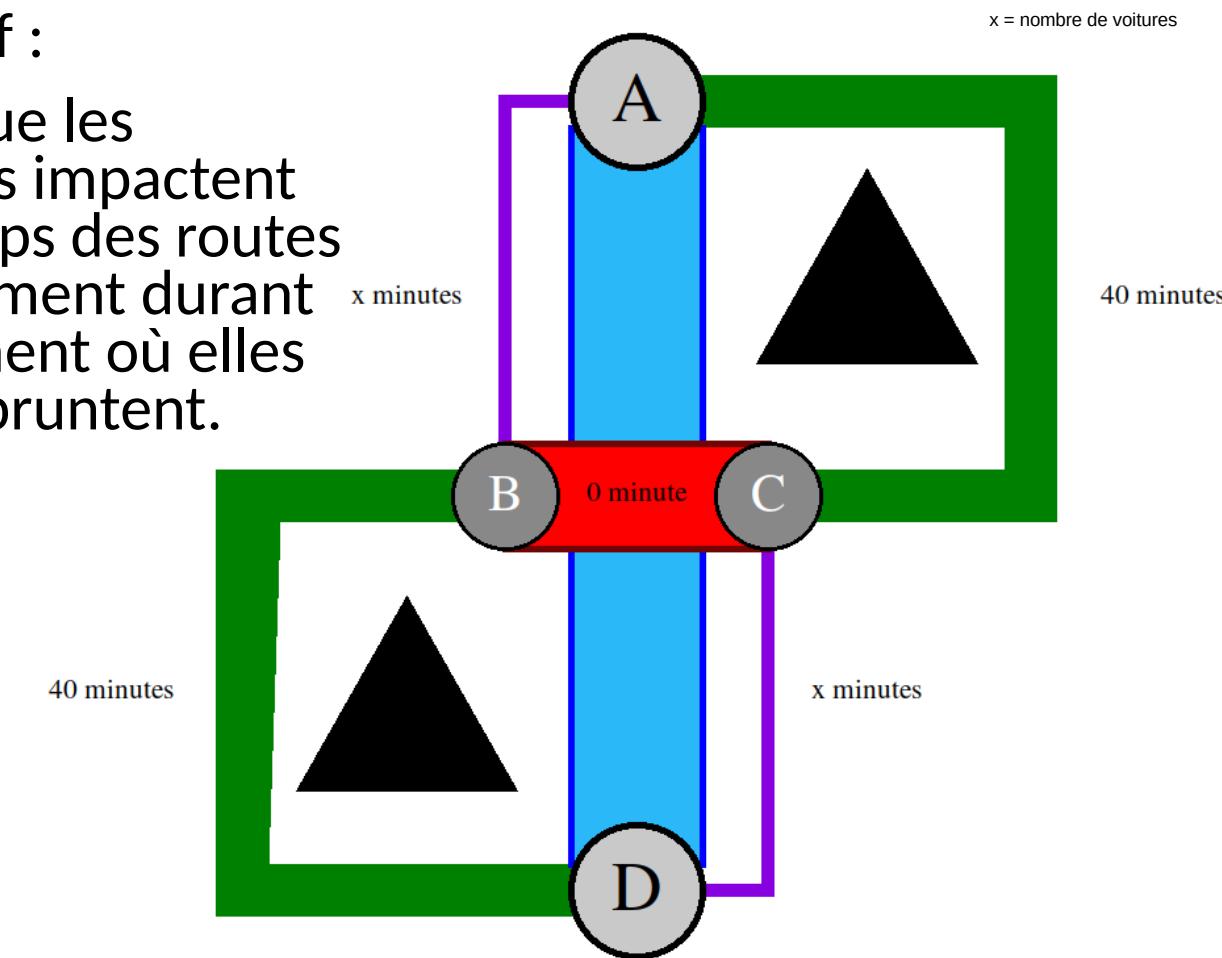
- Toutes les voitures partent exactement au même moment.

- Impact dynamique des autres voitures.

Objectif :

Faire que les voitures impactent les temps des routes uniquement durant le moment où elles les empruntent.

Modélisation après la construction du pont.



Tas de Fibonacci

Objectif : Gain en complexité

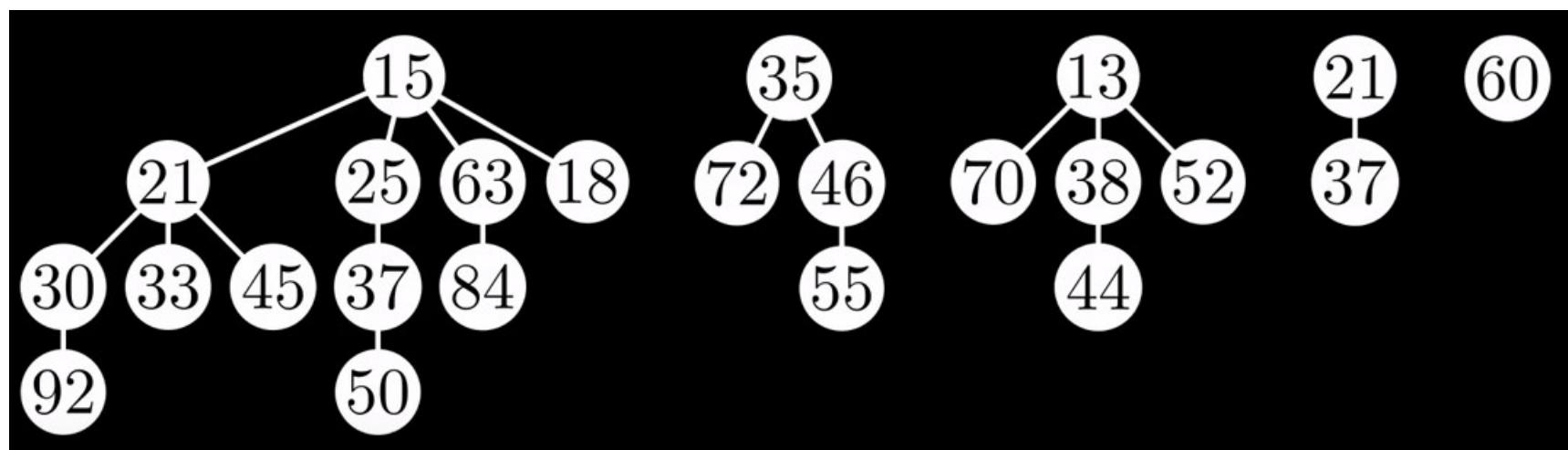
Obtenir minimum – O(1)
Union – O(1)
Ajouter un élément – O(1)
Décrémenter priorité – O(1)*
(marque)

Supprimer le minimum – O(log n)*
Supprimer un élément – O(log n)*
Augmenter priorité – O(log n)*

Arbres → Structure de tas

Table de hachage

Listes doublements chaînées



* : complexité amortie

Données utilisées

```
type 'a liste_doublement_chaine_circulaire = {
  t : 'a array;
  max : int; (* taille de t *)
  mutable premier : int;
  mutable nombre : int}
```

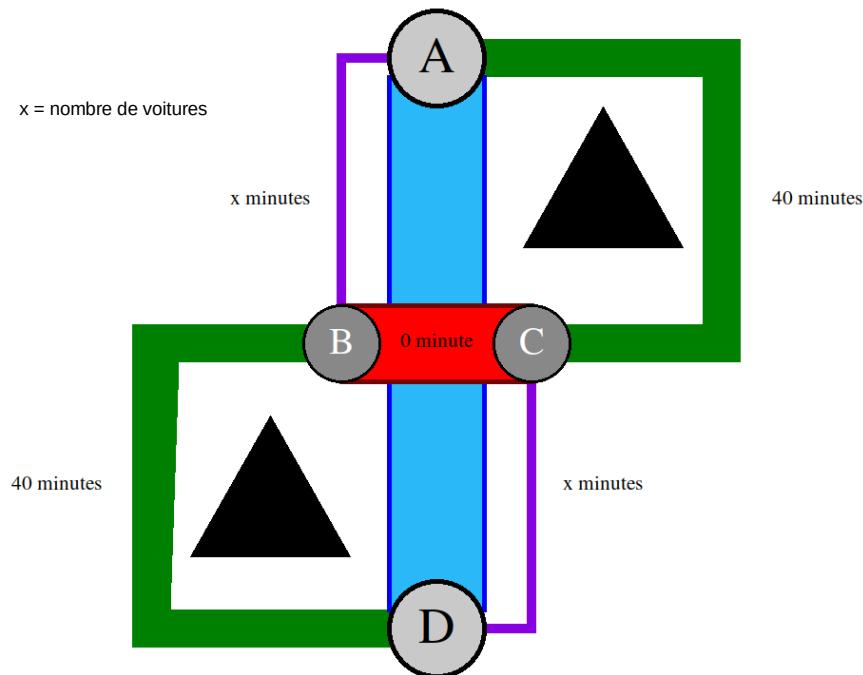
```
type 'a arbre =
| Vide
| Noeud of {mutable elem:'a ; mutable prio:float ; mutable fils:('a arbre)
liste_doublement_chaine_circulaire ; mutable pere:'a ; mutable marque:bool}
```

```
type 'a fibo = {
  table : ('a,'a arbre) Hashtbl.t;
  mutable liste_arbres : ('a arbre) liste_doublement_chaine_circulaire;
  mutable minimum : 'a arbre}
```

```
type repartition_route_direct = (num_route*int*int Queue.t*int fibo)fibo
```

Calcul du temps

Modélisation après la construction du pont.



Trajets	n° 1 : ABD	n° 2 : ACD	n° 3 : ABCD	n° 4 : ACBD
Nombre voitures	10	20	20	40

Étape initiale :

	n° 1 30'
	n° 3 30'
	AB 30' 30

	n° 4 40'
	n° 2 40'
	AC 40' 60

t = 0'

t = 30' t < minimum de la file

Chaque trajet ayant fini sa route est déplacé dans la route suivante

n° 3	
BC -∞ 0	

n° 1	
BD -∞ 0	

	n° 4 40'
	n° 2 40'
	AC 40' 60

Chaque trajet en attente est placé sur la route (!\ nombre de voitures)

N° 3 30'	
BC 30' 20	

N° 1 70'	
BD 70' 10	

	n° 4 40'
	n° 2 40'
	AC 40' 60

Calcul des répartitions égoïstes et sociales

Répartition égoïste

Pour toutes les voitures :

Pour chaque trajet où c'est nécessaire :

On modifie la répartition

On calcule le temps du trajet sur la répartition

On garde en mémoire la répartition associée au temps minimum, répartition qui devient définitive.

Répartition sociale

Pour toutes les voitures :

Pour chaque trajet où c'est nécessaire :

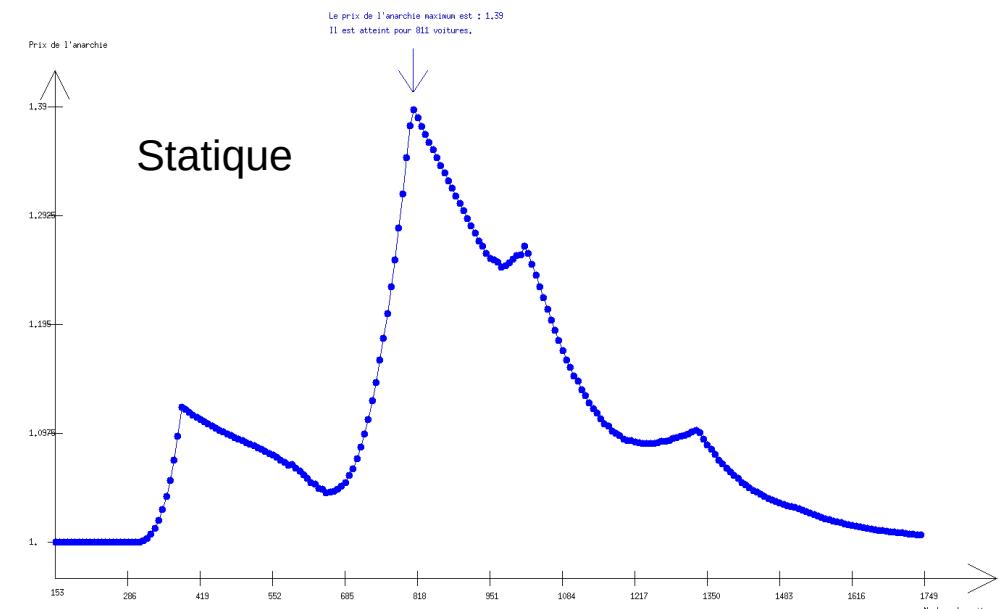
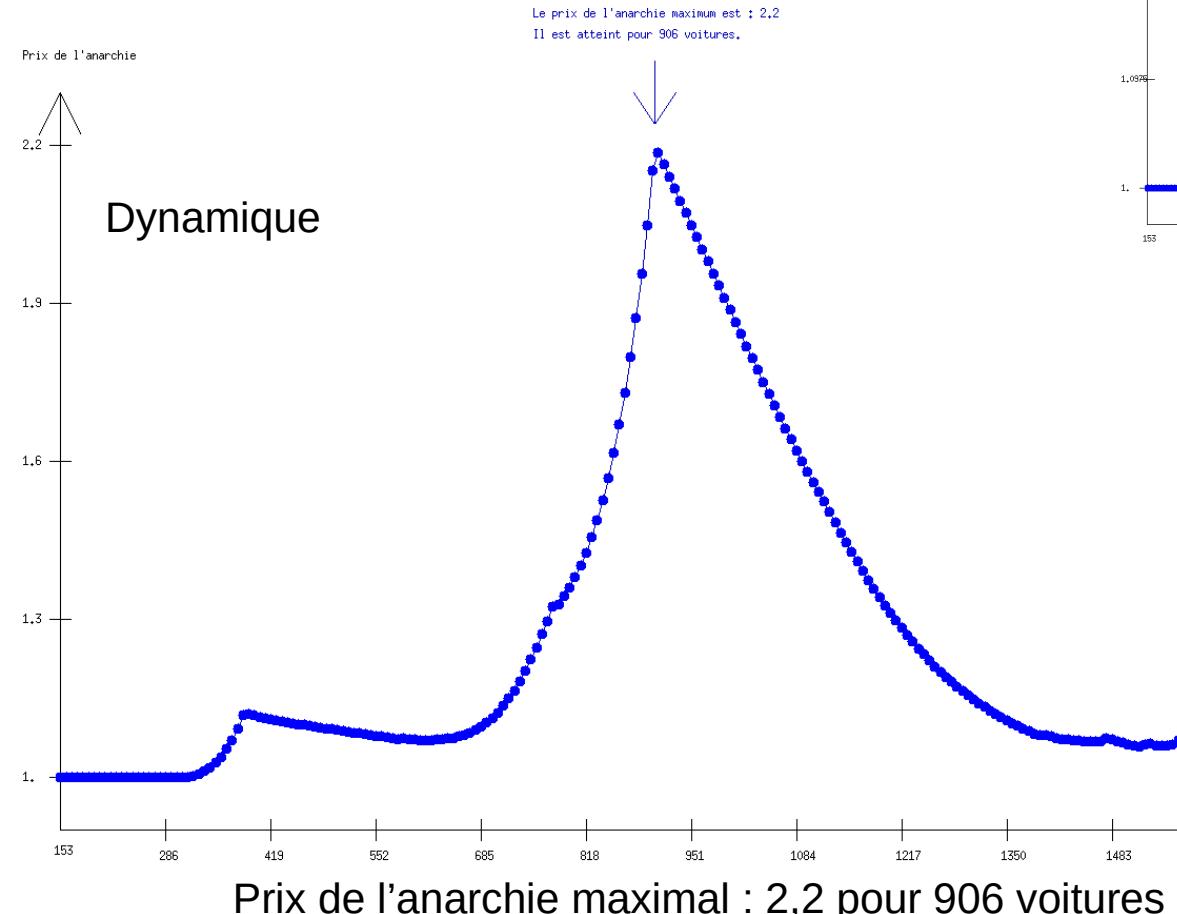
On modifie la répartition

On calcule le temps de la répartition

On garde en mémoire la répartition associée au temps minimum, répartition qui devient définitive.

V – Utilisations

- Même forme générale
- Minimisation des effets par le modèle statique

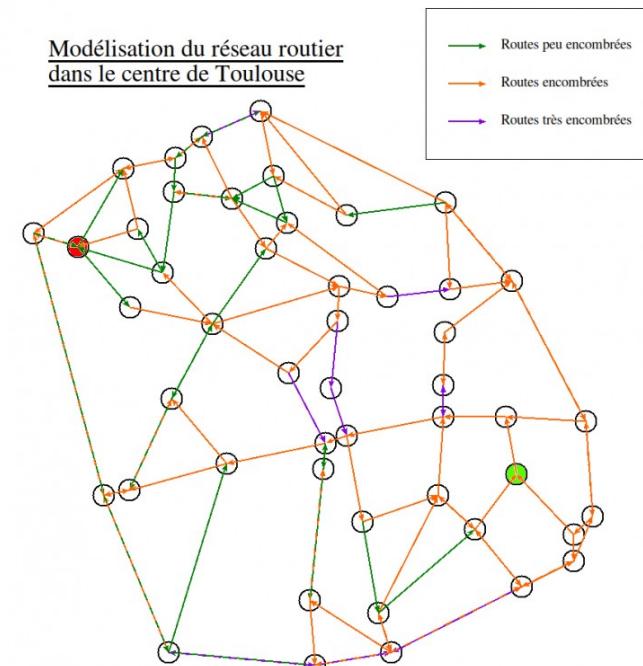


Dynamique :
Temps égoïste : 129,8
Temps social : 58,9

Statique :
Temps égoïste : 82,7
Temps social : 63,7

11005 trajets

Modélisation du réseau routier dans le centre de Toulouse



V – Utilisations

Répartition égoïste

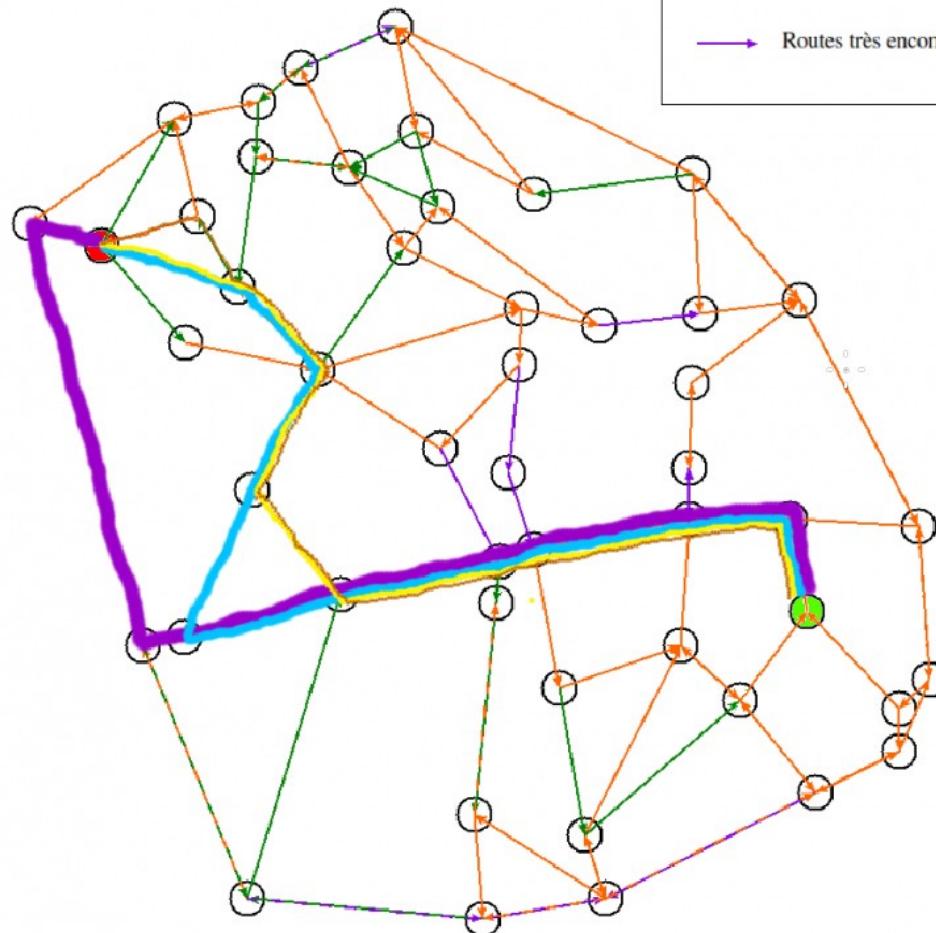
10762 : 424
10548 : 343
10071 : 135
10051 : 4

10762 : 313
10548 : 230
10071 : 36
10285 : 103
10030 : 43
2041 : 149
1318 : 32

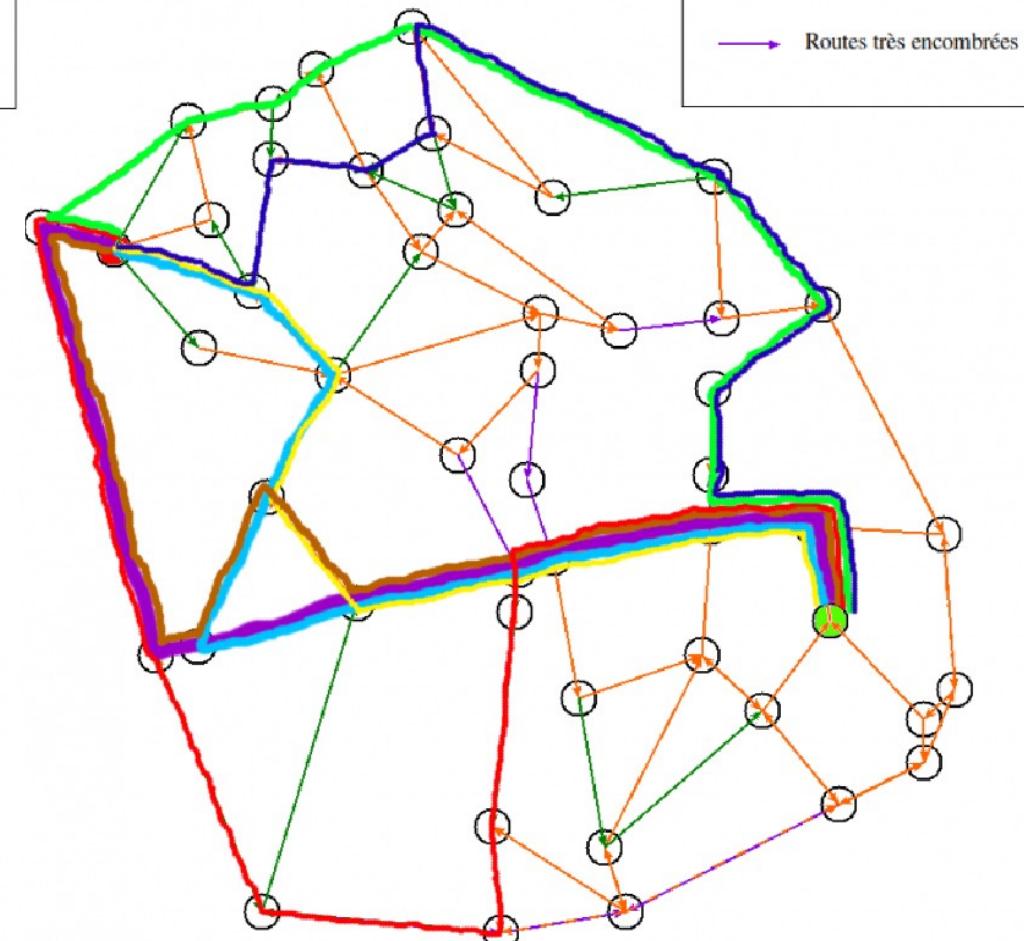
Répartition sociale

11005 trajets

Modélisation du réseau routier dans le centre de Toulouse



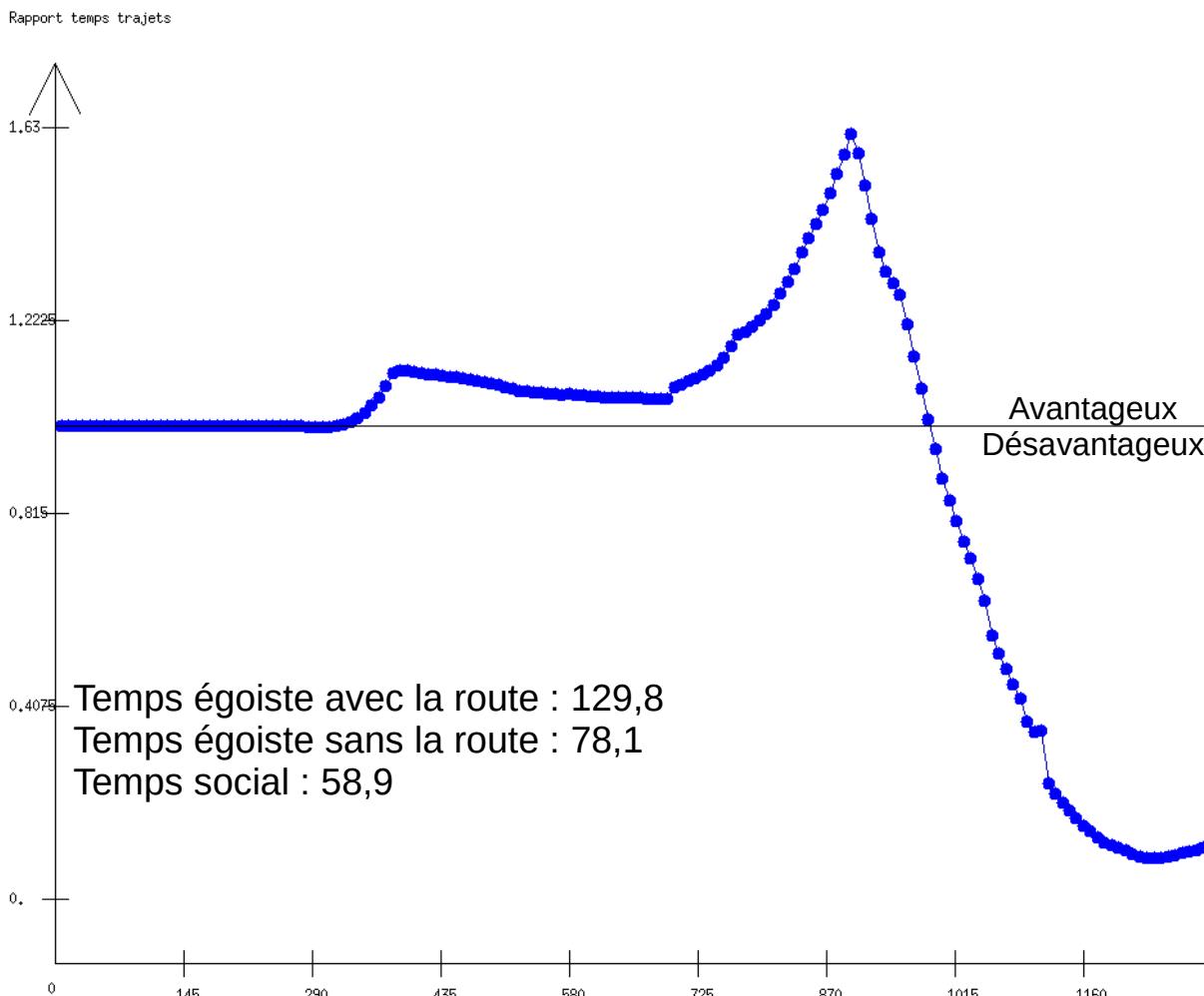
Modélisation du réseau routier dans le centre de Toulouse



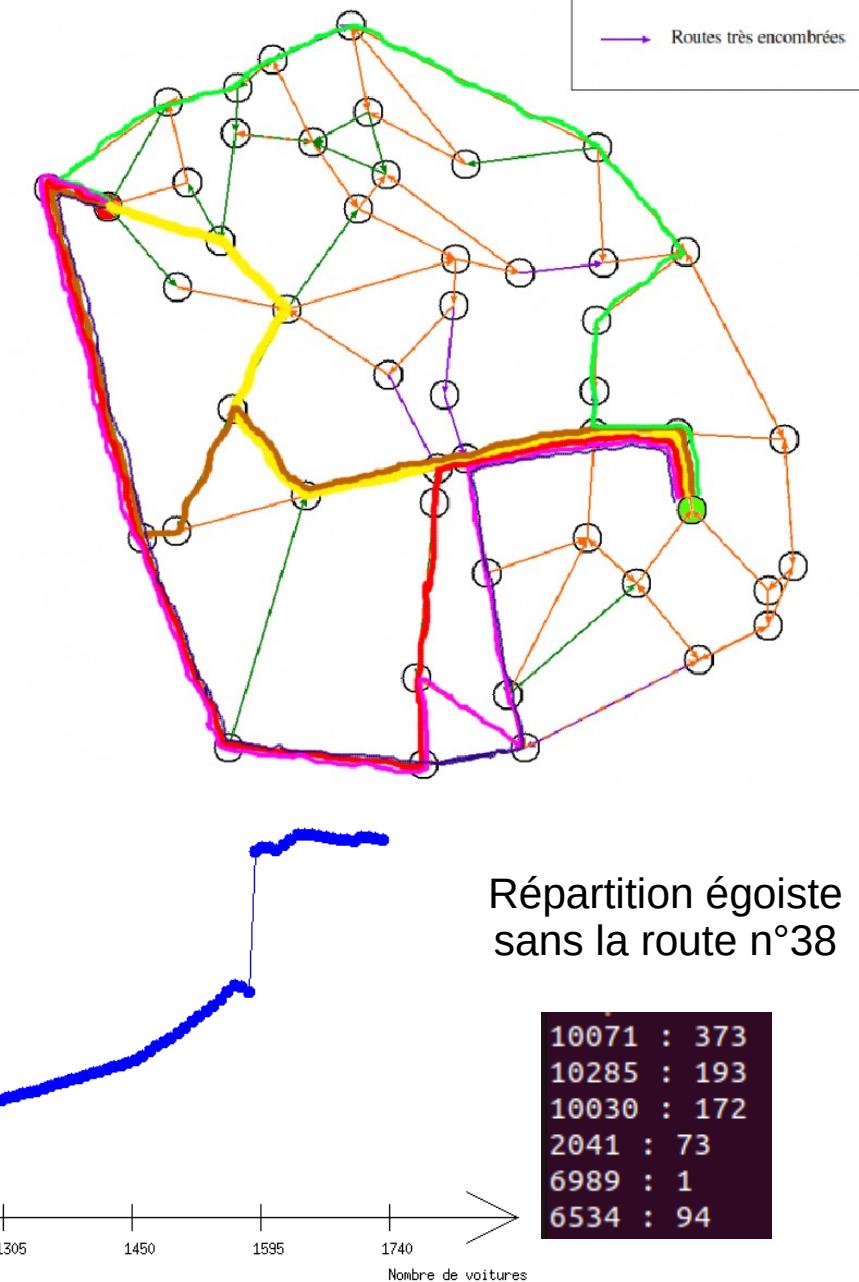
Routes inutiles ?

Suppression de la route n°38.

Pour 906 voitures on économise en moyenne 40 % du temps du trajet.



Modélisation du réseau routier dans le centre de Toulouse



Conclusion

- Se rapprocher encore plus de la réalité
- Ouverture : exploitation du problème

```

1 #require "graphics"
2 #load "graphics.cma"
3
4 (*Tas de Fibonacci*)
5
6
7 type lieu = int          (* numéro du lieu *)
8 type num_route = int      (* numéro de route *)
9 type route = (lieu*lieu*float*int)    (* (dep,arr,tps de + par
voit,tps init) *)
10 type trajet = num_route list
11 type num_trajet = int
12 type tab_route = route array      (* tableau des routes à
considéré *)
13 type tab_trajet = trajet array
14 type mat_adj_indice = int option array array (* matrice
d'adjacence representant l'ensemble des routes *)
15 type mat_adj_fonction_temps = (int->float) option array array
16 type repartition_trajet = int array
17
18 type 'a liste_doublement_chaine_circulaire = {
19   t : 'a array;
20   max : int; (* taille de t *)
21   mutable premier : int;
22   mutable nombre : int}
23
24 type 'a arbre =
25   |Vide
26   |Noeud of {mutable elem:'a ; mutable prio:float ; mutable fils
('a arbre) liste_doublement_chaine_circulaire ; mutable
pere:'a ; mutable marque:bool}
27
28 type 'a fibo = {
29   table : ('a,'a arbre) Hashtbl.t;
30   mutable liste_arbres : ('a arbre)
liste_doublement_chaine_circulaire;
31   mutable minimum : 'a arbre}
32
33 type repartition_route_direct = (num_route*int*int Queue.t*int
fibo)fibo
34
35 let cree_ldcc_vide (m:int) (e:'a) : 'a
liste_doublement_chaine_circulaire =
36   {t = Array.make m e ; max = m ; premier = 0 ; nombre = 0}
37
38 let ajoute_ldcc (e:'a) (l:'a
liste_doublement_chaine_circulaire) : unit =
39   if l.nombre = l.max then failwith "pas assez de place"
40   else begin
41     l.nombre <- l.nombre + 1;
42     if l.premier + l.nombre > l.max then
43       l.t.(l.premier + l.nombre - l.max - 1) <- e
44     l.t.(l.premier + l.nombre - 1) <- e
45   end
46
47
48 exception Probleme
49
50 let supprime_ldcc (e:'a) (l:'a
liste_doublement_chaine_circulaire) : unit =
51   let i = ref l.premier in
52   let j = ref 0 in
53   while l.t.(!i) <> e && !j <= l.nombre do
54     j:=!j+1;
55     if !i+1 < l.max then i := !i + 1
56     else i := 0
57   done;
58   if !j>l.nombre then raise (Probleme)
59   else
60     l.t.(!i) <- l.t.(l.premier);
61     l.nombre <- l.nombre - 1;
62     if l.premier + 1 < l.max then l.premier <- l.premier + 1
63     else l.premier <- 0
64
65
66
67 let cree_fibo_vide (m:int) : 'a fibo =
68   {table = Hashtbl.create m ; liste_arbres = cree_ldcc_vide m
Vide ; minimum=Vide}
69
70 (*
71 let optimise_fibo (f:'a fibo) : unit =
72   let p = f.liste_arbres.premier in
73   let d = ref 0 in
74   begin
75     if p + f.liste_arbres.nombre >= f.liste_arbres.max then
76       d := p + f.liste_arbres.nombre - f.liste_arbres.max
77     else d := p + f.liste_arbres.nombre
78   end;
79   if !d < p then
80     for i = !d + 2 to p-2 do
81       f.liste_arbres.t.(i) <- Vide
82     done
83   else
84     for i = 1 to p-2 do
85       f.liste_arbres.t.(i) <- Vide
86     done;
87     for i = !d + 2 to f.liste_arbres.max - 1 do
88       f.liste_arbres.t.(i) <- Vide
89     done
90   *)
91
92 let union_fibo (f1:'a fibo) (f2:'a fibo) : unit =

```

```

91
92 let union_fibo (f:'a fibo) (g:'a fibo) : unit =
93 (* f ou g n'est pas vide *)
94 begin
95 match f.minimum,g.minimum with
96 |Noeud n1, Noeud n2 when n1.prio > n2.prio -> f.minimum <-
97 g.minimum
98 |Noeud n1, _-> ()
99 |_|_ -> f.minimum <- g.minimum
100 end;
101 if f.liste_arbres.nombre + g.liste_arbres.nombre >
102 f.liste_arbres.max then failwith "pas assez de place dans fibo"
103 else
104 for i = 0 to g.liste_arbres.nombre - 1 do
105 ajoute_ldcc g.liste_arbres.t.(i) f.liste_arbres
106 done;
107 Hashtbl.iter (fun i a -> Hashtbl.add f.table i a) g.table
108
109
110 let ajoute_fibo (e:'a) (p:float) (f:'a fibo) : unit =
111 let f_bis = cree_fibo_vide 1 in
112 let arbre = Noeud {elem=e ; prio=p ; fils = cree_ldcc_vide 10
113 Vide ; pere = e ; marque = false} in
114 f_bis.minimum <- arbre;
115 f_bis.liste_arbres.nombre <- 1;
116 f_bis.liste_arbres.t.(0) <- arbre;
117 Hashtbl.add f.table e arbre;
118 union_fibo f f_bis
119
120 let obtenir_min_fibo_elem (f:'a fibo) : 'a =
121 match f.minimum with
122 |Vide -> failwith "elem : le tas est vide donc il n'y a pas de
123 minimum"
124 |Noeud{elem=e} -> e
125
126 let obtenir_min_fibo_prio (f:'a fibo) : float =
127 match f.minimum with
128 |Vide -> Float.infinity
129 |Noeud{prio=p} -> p
130
131 let degré_arbre (a:'a arbre) =
132 match a with
133 |Vide -> 0
134 |Noeud{elem=e ; prio=p ; fils = l} -> l.nombre
135
136 let extraire_min_fibo (f:'a fibo) : 'a =
137 supprime_ldcc f.minimum f.liste_arbres;
138 match f.minimum with
139 |Vide -> failwith "erreur15"
140 |Noeud n ->
141 let mini = n.elem in
142 begin
143 let j = ref n.fils.premier in
144 for i = 0 to degré_arbre f.minimum - 1 do
145 match n.fils.t.(!j) with
146 |Vide -> failwith "erreur16"
147 |Noeud n_fils ->
148 n_fils.pere <- n_fils.elem;
149 ajoute_ldcc (Noeud n_fils) f.liste_arbres;
150 Hashtbl.remove f.table n_fils.elem;
151 Hashtbl.add f.table n_fils.elem (Noeud n_fils);
152 if !j + 1 < n.fils.max then j := !j + 1
153 else j := 0
154 done;
155 end;
156 let tab = Array.make f.liste_arbres.max Vide in
157 let j = ref f.liste_arbres.premier in
158 for i = 0 to f.liste_arbres.nombre - 1 do
159 let rec aux (arb:'a arbre) : unit =
160 match tab.(degré_arbre arb) with
161 |Vide -> tab.(degré_arbre arb) <- arb
162 |Noeud m ->
163 supprime_ldcc (Noeud m) f.liste_arbres;
164 supprime_ldcc arb f.liste_arbres;
165 match arb with
166 |Vide -> failwith "erreur17"
167 |Noeud a when a.prio < m.prio ->
168 m.pere <- a.elem;
169 a.pere <- a.elem;
170 ajoute_ldcc (Noeud m) a.fils;
171 ajoute_ldcc (Noeud a) f.liste_arbres;
172 tab.(degré_arbre (Noeud a) - 1) <- Vide;
173 aux (Noeud a)
174 |Noeud a ->
175 a.pere <- m.elem;
176 m.pere <- m.elem;
177 ajoute_ldcc (Noeud a) m.fils;
178 ajoute_ldcc (Noeud m) f.liste_arbres;
179 tab.(degré_arbre (Noeud m) - 1) <- Vide;
180 aux (Noeud m)
181 in aux f.liste_arbres.t.(!j);
182 if !j + 1 < f.liste_arbres.max then j := !j + 1
183 else j := 0
184 done;
185 let min_j = ref mini in
186 let min_p = ref Float.infinity in
187 let j = ref f.liste_arbres.premier in
188 for i = 0 to f.liste_arbres.nombre - 1 do

```

```

188 for i = 0 to f.liste_arbres.nombre - 1 do
189   match f.liste_arbres.t.(!j) with
190   | Vide -> ()
191   | Noeud n ->
192     Hashtbl.remove f.table n.elem;
193     Hashtbl.add f.table n.elem (Noeud n);
194     if !min_p >= n.prio then begin
195       min_p := n.prio;
196       min_j := n.elem
197     end;
198     if !j + 1 < f.liste_arbres.max then j := !j + 1
199     else j := 0
200   done;
201   if !min_j = mini then f.minimum <- Vide
202   else f.minimum <- Hashtbl.find f.table !min_j;
203   mini
204
205 let diminuer_prio_fibo (e:'a) (p:float) (f:'a fibo) : unit =
206   match Hashtbl.find f.table e with
207   | Vide -> failwith "erreur18"
208   | Noeud n ->
209     begin
210       match f.minimum with
211       | Vide -> failwith "erreur19"
212       | Noeud mi when mi.prio < p -> ()
213       |_ -> f.minimum <- Noeud n
214     end;
215   n.prio <- p;
216   let continu = ref true in
217   let n_pere = ref n.pere in
218   let fils = ref (Noeud n) in
219   while !continu do
220     match Hashtbl.find f.table !n_pere with
221     | Vide -> failwith "erreur20"
222     | Noeud nn when nn.prio <= n.prio -> continu := false
223     | Noeud nn ->
224       supprime_ldcc (!fils) nn.fils;
225       match !fils with
226       | Vide -> failwith "erreur20.5"
227       | Noeud m ->
228         m.pere <- m.elem;
229         ajoute_ldcc (Noeud m) f.liste_arbres;
230         Hashtbl.remove f.table m.elem;
231         Hashtbl.add f.table m.elem (Noeud m);
232         if nn.elem <> nn.pere then
233           if nn.marque then begin
234             fils := Noeud nn;
235             n_pere := nn.pere
236           end
237           else begin
238             nn.marque <- true;
239
240           else begin
241             nn.marque <- true;
242             continu := false
243           end
244           else continu := false
245         done;
246         match f.minimum with
247         | Vide -> failwith "erreur21"
248         | Noeud mi -> f.minimum <- Hashtbl.find f.table mi.elem
249
250 let supprime_element_fibo (e:'a) (f:'a fibo) : unit =
251   diminuer_prio_fibo e Float.neg_infinity f;
252   let _ = extraire_min_fibo f in
253   ()
254
255 let augmenter_prio_fibo (e:'a) (p:float) (f:'a fibo) : unit =
256   supprime_element_fibo e f;
257   ajoute_fibo e p f
258
259 let f =
260   let g = cree_fibo_vide 15 in
261   ajoute_fibo 1 1. g;
262   ajoute_fibo 2 2. g;
263   ajoute_fibo 3 3. g;
264   ajoute_fibo 4 4. g;
265   ajoute_fibo 5 5. g;
266   g
267
268
269 let est_vide_fibo (f:'a fibo) : bool =
270   f.liste_arbres.nombre = 0
271
272
273
274 (*
275 let tab_routes : tab_route = [(0,1,0.,40);(2,3,0.,40);
276   (0,2,1.,0);(1,3,1.,0);(2,1,0.,0);(1,2,0.,0)]]
277 let nb_lieu = 4
278 let dep = 0
279 let arr = 3
280 let voit_tot = 40
281 let fonction_temps_route (t:tab_route)(i:num_route)(x:int) =
282   (* calcule le temps d'emprunt de la route i de  *)
283   let (_,_,a,b) = tab_routes.(i) in
284     (float_of_int(b)+.a*.float_of_int(x))
285   *)
286 let vp = 450.          (* petite route non encombrée - tps de +
287   par voit *)

```

```

285 *)
286 let vp = 450. (* petite route non encombrée - tps de +
  par voit *)
287 let vm = 300. (* petite route moyennement encombrée -
  tps de + par voit *)
288 let vg = 150. (* petite route très encombrée - tps de +
  par voit *)
289 let op = 1000. (* petite route non encombrée - tps de +
  + par voit *)
290 let om = 850. (* petite route moyennement encombrée -
  tps de + par voit *)
291 let og = 700. (* petite route très encombrée - tps de +
  par voit *)
292 let rp = 1550. (* petite route non encombrée - tps de +
  + par voit *)
293 let rm = 1400. (* petite route moyennement encombrée -
  tps de + par voit *)
294 let rg = 1250. (* petite route très encombrée - tps de +
  + par voit *)
295
296 let voit_tot = 1
297 let nb_lieu = 49
298 (*
299 let dep = 0
300 let arr = 26 (*25 cool*) (*39 montagne*)
301 *)
302
303 let tab_routes : tab_route = [
304 (1,0,og,200);(13,0,vg,400);(14,0,og,43);
305 (0,1,vg,212);(21,1,vg,166);
306 (3,2,rg,133);(20,2,og,43);(21,2,rg,100);
307 (2,3,og,133);(4,3,og,66);(35,3,om,83);
308 (3,4,og,63);(5,4,og,33);
309 (6,5,og,40);
310 (4,6,og,166);(7,6,og,133);
311 (6,7,og,133);(8,7,og,217);
312 (7,8,og,217);(9,8,og,150);(33,8,op,450);(42,8,om,175);
313 (8,9,og,133);(10,9,og,217);
314 (9,10,og,233);(11,10,rg,80);(43,10,op,550);
315 (10,11,vg,77);(30,11,om,150);(36,11,og,37);
316 (13,12,vg,166);(31,12,vg,133);(36,12,og,66);(45,12,om,150);
317 (0,13,og,366);(12,13,og,266);(31,13,om,105);
318 (0,14,og,33);(15,14,og,117);(16,14,vg,133);
319 (1,15,vp,700);(23,15,og,117);
320 (14,16,og,133);(15,16,op,300);(25,16,vg,100);
321 (18,17,og,117);(40,17,rp,170);
322 (19,18,og,93);(37,18,og,66);(41,18,rm,45);
323 (20,19,op,450);(35,19,og,60);(48,19,op,350);
324 (2,20,om,70);(48,20,vp,350);
325 (1,21,rg,183);(2,21,og,100);(22,21,om,125);
326 (2,22,og,117);(24,22,vm,250);
327 (17,23,og,17);(46,23,rp,350);(24,23,vp,60);
328 (22,24,om,300);(23,24,vp,60);
329 (16,25,vm,150);(46,25,op,350);(47,25,op,500);
330 (25,26,om,225);(27,26,om,135);
331 (25,27,vm,175);(30,27,om,140);
332 (27,28,om,70);(29,28,om,175);(38,28,vp,270);
333 (26,29,om,55);
334 (28,30,vm,140);(38,30,vm,100);(44,30,vm,95);
335 (13,31,vm,90);(32,31,vm,175);(45,31,op,250);
336 (25,32,op,270);(44,32,vm,175);
337 (41,33,om,115);
338 (5,34,op,390);(35,34,op,270);
339 (3,35,om,125);(19,35,og,60);(20,35,vm,175);
340 (11,36,vg,40);(12,36,og,66);
341 (7,37,og,100);(34,37,op,300);
342 (10,38,op,240);(43,38,op,400);
343 (26,39,op,280);
344 (39,40,rp,270);
345 (18,41,rp,100);(33,41,om,90);
346 (9,42,op,550);(29,42,rp,170);
347 (9,43,vp,300);
348 (30,44,om,90);(36,44,vm,95);
349 (32,45,vp,250);
350 (39,46,op,300);
351 (31,47,vm,150);
352 (17,48,om,150)|]
353
354
355 let fonction_temps_route (t:tab_route)(i:num_route)(x:int) =
  (* calcule le temps d'emprunt de la route i par x voitures *)
356 let (_,_,a,b) = tab_routes.(i) in
357 (float_of_int(b)/.30.)*(1+.0.2*(float_of_int(x)/.a)**10.)
358
359
360 let graphe_trajet (t:tab_route) : mat_adj_indice = (* cree la
  matrice d'adjacence du graphe, matrice indicé par le numéro des
  routes *)
361 let tab = Array.make_matrix nb_lieu nb_lieu None in
362 for i=0 to Array.length t - 1 do
363   let (dep,arr,a,b) = t.(i) in
364     tab.(dep).(arr) <- Some(i)
365   done;
366 tab
367
368
369 let graphe_trajet_sans_i (t:tab_route) (i:num_route) :
  mat_adj_indice = (* cree la matrice d'adjacence du graphe
  privé de la route i, matrice indicé par le numéro des routes *)
370 let tab = Array.make_matrix nb_lieu nb_lieu None in
371 for j=0 to nb_lieu-1 do
372   if j < i then begin
373     let (dep,arr,a,b) = t.(j) in

```

```

372     if j <> i then begin
373       let (dep,arr,a,b) = t.(j) in
374       tab.(dep).(arr) <- Some(j)
375     end;
376   done;
377   tab
378
379
380
381
382
383 let tab_trajet (m:mat_adj_indice) (dep:lieu) (arr:lieu)
384   (t:tab_route) : tab_trajet = (* renvoie le tableau
385   des trajets possibles entre dep et arr d'après m *)
386 let liste_traj = ref [] in
387 let lieu_visite = Array.make nb_lieu false in
388 lieu_visite.(dep) <- true;
389 let rec aux (l_actu:lieu) (traj_actu:trajet) (visite:bool
390   array) =
391   for i = 0 to nb_lieu-1 do
392     match m.(l_actu).(i),visite.(i) with
393     |_,true -> ();
394     |None,false -> ();
395     |Some(n_route),false ->
396       if i = arr then liste_traj := (List.rev
397         (n_route:::traj_actu))::!liste_traj
398       else
399         if (List.mem n_route traj_actu) then ()
400         else begin
401           let visite_bis = Array.copy visite in
402             visite_bis.(i) <- true;
403             aux i (n_route:::traj_actu) visite_bis
404           end;
405         done;
406   in aux dep [] lieu_visite;
407   Array.of_list (!liste_traj)
408
409
410 let mat_temps_route (m:mat_adj_indice)(t:tab_route) :
411   mat_adj_fonction_temps =
412   let tab = Array.make_matrix nb_lieu nb_lieu None in
413   for i = 0 to nb_lieu-1 do
414     for j = 0 to nb_lieu-1 do
415       match m.(i).(j) with
416       |None -> tab.(i).(j) <- None
417       |Some(n_route) -> tab.(i).(j)<-Some(fonction_temps_route t
418         n_route)
419     done;
420   tab
421
422
423
424
425
426 (*V1 : On découpe le temps en paquet et on compte le nombre de
427   voitures qui sont sur la route dans le paquet*)
428
429
430
431 let temps (r:repartition_trajet)(mf:mat_adj_fonction_temps)
432   (tab_traj:tab_trajet)(tab_routes:tab_route) (voit_tot:int)=
433   (*calcule le temps de trajet associé à la répartition mv*)
434   let nb_routes = Array.length tab_routes in
435   let nb_traj = Array.length tab_traj in
436   let tps = ref 0. in
437   let tps_tot = ref 0. in
438   let nb_voit = ref 0 in
439   Array.iter (fun x -> nb_voit := !nb_voit + x) r;
440   let init_route = Array.make nb_routes 0 in
441   let tab_t = Array.copy tab_traj in
442   let table = Array.make nb_routes None in
443   let utile = ref 0 in
444   let rep = Array.copy r in
445   for i = 0 to nb_traj - 1 do
446     if rep.(i) <> 0 then utile := !utile + 1;
447     match tab_t.(i) with
448     |[] -> failwith "erreur1"
449     |e:: suiv ->
450       init_route.(e) <- init_route.(e) + rep.(i);
451     done;
452   let r_routes : repartition_route_direct = cree_fibo_vide !utile
453   in
454   for i = 0 to nb_routes - 1 do
455     let (d,a,_,_) = tab_routes.(i) in
456     match init_route.(i),mf.(d).(a) with
457     |0,_ -> ()
458     |n,None -> failwith "erreur2"
459     |n,Some(f) ->
460       let v = (i,n,Queue.create(),cree_fibo_vide !utile) in
461         ajoute_fibo v (f n) r_routes;
462         table.(i) <- Some(v)
463     done;
464   for i = 0 to nb_traj - 1 do
465     match rep.(i),tab_t.(i) with
466     |0,_ -> tab_t.(i) <- []
467     |[],_ -> failwith "erreur3"

```

```

464 |_,[] -> failwith "erreur3"
465 |n,e::suiv ->
466   let (d,a,_,_) = tab_routes.(e) in
467   begin
468     match mf.(d).(a) with
469     |None -> failwith "erreur4"
470     |Some(f) ->
471       match table.(e) with
472       |Some(aa) ->
473         let (nr,nv,q,fib) = aa in
474         supprime_element_fibo (aa) r_routes;
475         ajoute_fibo i (f init_route.(e)) fib;
476         ajoute_fibo (nr,nv,q,fib) (obtenir_min_fibo_prio fib)
477         r_routes;
478         table.(e) <- Some(nr,nv,q,fib)
479       |_ -> failwith "erreur5"
480     end;
481     tab_t.(i) <- suiv
482   done;
483   let dans_q = Array.make nb_routes false in
484   let comp = ref 0 in
485   while not (est_vide_fibo r_routes) do
486     (*print_int !comp;
487     print_string " ";
488     print_float !tps_tot;
489     print_newline();
490     comp:=!comp+1;*)
491     let continu = ref true in
492     while !continu do
493       let (n_route,nb_voit,q,fib) = extraire_min_fibo r_routes in
494       if est_vide_fibo fib then ()
495       else begin
496         match fib.minimum with
497         |Vide -> failwith "erreur6"
498         |Noeud e ->
499           tps := e.prio;
500           let _ = extraire_min_fibo fib in
501           if est_vide_fibo fib && Queue.is_empty q then table.
502             (n_route) <- None;
503             match tab_t.(e.elem) with
504             |[] -> if table.(n_route)<>None then begin
505               ajoute_fibo (n_route,nb_voit-rep.(e.elem),q,fib)
506               (obtenir_min_fibo_prio fib) r_routes;
507               match table.(n_route) with
508               |None -> failwith "erreur7"
509               |Some (a,b,c,d) -> table.(n_route) <- Some (a,nb_voit
510               rep.(e.elem),c,d)
511             end;
512             (*print_int e.elem;
513             print_string "(";
514             print_int rep.(e.elem);
515             print_string "), tps : ";
516             (e.elem));
517             (*print_float !tps_tot;
518             print_newline();*)
519             |i::suiv ->
520               (*print_string "b";*)
521               tab_t.(e.elem) <- suiv;
522               let (a,b,_,_) = tab_routes.(i) in
523               match mf.(a).(b) with
524               |None -> failwith "erreur8"
525               |Some f ->
526                 match table.(i) with
527                 |None ->
528                   let qq = Queue.create() in
529                     Queue.add e.elem qq;
530                     let v = (i,0,qq,cree_fibo_vide !utile) in
531                     ajoute_fibo v Float.infinity r_routes;
532                     table.(i) <- Some(v);
533                     dans_q.(i) <- true;
534                     if (nb_voit <> rep.(e.elem)) && (not (est_vide_fibo
535                     fib)) then begin
536                       ajoute_fibo (n_route,nb_voit - rep.
537                         (e.elem),q,fib) (obtenir_min_fibo_prio fib) r_routes;
538                       match table.(n_route) with
539                         |None -> failwith "erreur9"
540                         |Some (a,b,c,d) -> table.(n_route) <- Some
541                           (a,nb_voit - rep.(e.elem),c,d)
542                         end
543                         else table.(n_route) <- None
544                         |Some(v) ->
545                           let (r,voit,qq,fi) = v in
546                             let k = Hashtbl.find r_routes.table v in
547                               Hashtbl.remove r_routes.table v;
548                               Queue.add e.elem qq;
549                               table.(i) <- Some(v);
550                               Hashtbl.add r_routes.table v k;
551                               dans_q.(i) <- true;
552                               if (nb_voit <> rep.(e.elem)) && (not (est_vide_fibo
553                               fib)) then begin
554                                 ajoute_fibo (n_route,nb_voit - rep.(e.elem),q,fib)
555                                 (obtenir_min_fibo_prio fib) r_routes;
556                                 match table.(n_route) with
557                                   |None -> failwith "erreur10"
558                                   |Some (a,b,c,d) -> table.(n_route) <- Some
559                                     (a,nb_voit - rep.(e.elem),c,d)
560                                   end
561             (*print_int rep.(e.elem);
562             print_string ", tps : ";
563             print_float !tps_tot;
564             print_newline();*)

```

```

554
555         end
556     else table.(n_route) <- None
557   end;
558   if est_vide_fibo r_routes then continu := false
559   else if obtenir_min_fibo_prio r_routes > !tps then
560     continu := false
561   done;
562   for i = 0 to nb_routes - 1 do
563     match dans_q.(i) with
564     |false -> ()
565     |true -> begin
566       dans_q.(i) <- false;
567       match table.(i) with
568       |None -> failwith "erreur11"
569       |Some e ->
570         match Hashtbl.find r_routes.table e with
571         |Vide -> failwith "erreur12"
572         |Noeud n ->
573           let (nr,nv,q,fib) = n.elem in
574           let traj = ref (Queue.pop q) in
575           let prem = !traj in
576           let somme = ref (nv + rep.(!traj)) in
577             Queue.add (!traj) q;
578             while Queue.top q <> prem do
579               traj := Queue.pop q;
580               Queue.add (!traj) q;
581               somme := !somme + rep.(!traj)
582             done;
583             supprime_element_fibo n.elem r_routes;
584             let (a,b,_,_) = tab_routes.(nr) in
585             match mf.(a).(b) with
586             |None -> failwith "erreur13"
587             |Some f ->
588               while not (Queue.is_empty q) do
589                 ajoute_fibo (Queue.pop q) (!tps +. f !somme) fib
590               done;
591               let ppp = obtenir_min_fibo_prio fib in
592               ajoute_fibo (nr,!somme,q,fib) ppp r_routes;
593               begin
594                 match table.(nr) with
595                 |None -> failwith "erreur14"
596                 |Some (a,b,c,d) -> table.(nr) <- Some (a,!
597                   somme,c,d)
598               end;
599               Hashtbl.remove r_routes.table e;
600               Hashtbl.add r_routes.table e (Noeud n);
601             end;
602             done;
603             !tps_tot /. (float_of_int voit_tot)
604
605
606
607 let aaa (tab:int array)(t:tab_trajet)(tbase:tab_trajet) =
608   for i = 0 to Array.length tab - 1 do
609     if tab.(i) <> 0 then
610       begin
611         let traj = t.(i) in
612         let j = ref 0 in
613         while tbase.(!j) <> traj do
614           j:=!j+1;
615         done;
616         print_int !j;
617         print_string " : ";
618         print_int tab.(i);
619         print_newline()
620       end;
621   done
622
623
624
625
626 let temps_trajet (r:repartition_trajet)
627   (mf:mat_adj_fonction_temps)(tab_traj:tab_trajet)
628   (tab_routes:tab_route)(trajet:num_trajet) =
629   (*temps mais on arrête dès qu'il n'y a plus de voiture sur le
630    trajet considéré*)
631   let nb_routes = Array.length tab_routes in
632   let nb_traj = Array.length tab_traj in
633   let tps = ref 0. in
634   let tps_tot = ref 0. in
635   let init_route = Array.make nb_routes 0 in
636   let tab_t = Array.copy tab_traj in
637   let table = Array.make nb_routes None in
638   let utile = ref 0 in
639   let rep = Array.copy r in
640   rep.(trajet) <- rep.(trajet) + 1;
641   for i = 0 to nb_traj - 1 do
642     if rep.(i) <> 0 then utile := !utile + 1;
643     match tab_t.(i) with
644     |[] -> failwith "erreur1"
645     |e:: suiv ->
646       init_route.(e) <- init_route.(e) + rep.(i);
647     done;
648   let r_routes : repartition_route_direct = cree_fibo_vide !utile
649   in
650   for i = 0 to nb_routes - 1 do
651     let (d,a,_,_) = tab_routes.(i) in
652     match init_route.(i),mf.(d).(a) with
653     |0,_ -> ()
654     |n,None -> failwith "erreur2"
655     |n,Some(f) ->
656       let v = (i,n,Queue.create(),cree_fibo_vide !utile) in
657       ajoute_fibo v (f n) r_routes;

```



```

746     else table.(n_route) <- None
747   end;
748   if est_vide_fibo r_routes then continu := false
749   else if obtenir_min_fibo_prio r_routes > !tps then
    continu := false
750 done;
751 for i = 0 to nb_routes - 1 do
752   match dans_q.(i) with
753   |false -> ()
754   |true -> begin
755     dans_q.(i) <- false;
756     match table.(i) with
757     |None -> failwith "erreur11"
758     |Some e ->
759       match Hashtbl.find r_routes.table e with
760       |Vide -> failwith "erreur12"
761       |Noeud n ->
762         let (nr,nv,q,fib) = n.elem in
763         let traj = ref (Queue.pop q) in
764         let prem = !traj in
765         let somme = ref (nv + rep.(!traj)) in
766         Queue.add (!traj) q;
767         while Queue.top q <> prem do
768           traj := Queue.pop q;
769           Queue.add (!traj) q;
770           somme := !somme + rep.(!traj)
771         done;
772         supprime_element_fibo n.elem r_routes;
773         let (a,b,_,_) = tab_routes.(nr) in
774         match mf.(a).(b) with
775         |None -> failwith "erreur13"
776         |Some f ->
777           while not (Queue.is_empty q) do
778             ajoute_fibo (Queue.pop q) (!tps +. f !somme) fib
779           done;
780           let ppp = obtenir_min_fibo_prio fib in
781           ajoute_fibo (nr,!somme,q,fib) ppp r_routes;
782           begin
783             match table.(nr) with
784             |None -> failwith "erreur14"
785             |Some (a,b,c,d) -> table.(nr) <- Some (a,!
    somme,c,d)
786           end;
787           Hashtbl.remove r_routes.table e;
788           Hashtbl.add r_routes.table e (Noeud n);
789         end;
790       done;
791       if est_vide_fibo r_routes then stop := true
792     done;
793   !tps
794
795

```

```

797 let liste_rep_ego (mf:mat_adj_fonction_temps)
  (tab_traj:tab_trajet)(tab_routes:tab_route)(voit_max:int) :
  repartition_trajet array=
798 let nb_traj = Array.length tab_traj in
799 let rep : repartition_trajet = Array.make nb_traj 0 in
800 let mini = Array.make nb_traj 0. in
801 for i = 0 to nb_traj - 1 do
802   rep.(i) <- 1;
803   mini.(i) <- temps rep mf tab_traj tab_routes 1;
804   rep.(i) <- 0;
805 done;
806 let repo = Array.copy rep in
807 let tab_rep = Array.make (voit_max+1) (Array.copy rep) in
808 if nb_traj <> 0 then
  begin
    for e = 1 to voit_max do (* Pour toutes les voitures *)
      (*print_int e;
      print_newline();*)
      let temps_min = ref (Float.infinity) in
      let trajet_min = ref(-1) in
      for i = 0 to nb_traj - 1 do (* Pour tous les trajets *)
        if !temps_min >= mini.(i) then begin
          let tps = begin
            try temps_trajet rep mf tab_traj tab_routes i with
            |Probleme -> Float.infinity
          end in
          temps_min := Float.min (!temps_min) tps;
          if (!temps_min) = tps then trajet_min := i;
          if tps <> Float.infinity then mini.(i) <- tps;
        end;
      done;
      rep.(!trajet_min) <- rep.(!trajet_min) + 1;
      tab_rep.(e) <- Array.copy rep;
    done;
  end;
  tab_rep
832 let liste_rep_social (mf:mat_adj_fonction_temps)
  (tab_traj:tab_trajet)(tab_routes:tab_route)(voit_max:int):
  repartition_trajet array=
833 let nb_traj = Array.length tab_traj in
834 let rep : repartition_trajet = Array.make nb_traj 0 in
835 let mini = Array.make nb_traj 0. in
836 for i = 0 to nb_traj - 1 do
837   rep.(i) <- 1;
838   mini.(i) <- temps rep mf tab_traj tab_routes 1;
839   rep.(i) <- 0;
840 done;
841 let min = Array.copy mini in
842 let tab_rep = Array.make (voit_max+1) (Array.copy rep) in
843 if nb_traj <> 0 then
  begin

```

```

844
845 begin
846 let temps_min = ref (Float.infinity) in
847 let trajet_min = ref(0) in
848 for e = 1 to voit_max do (* Pour toutes les voitures *)
849   (*print_int e;
850   print_newline();*)
851   let repo = Array.copy rep in
852   repo.(!trajet_min) <- repo.(!trajet_min) + 1;
853   temps_min := begin
854     try temps repo mf tab_traj tab_routes e with
855       |Probleme -> Float.infinity
856   end;
857   for i = 0 to nb_traj - 1 do (* Pour tous les trajets *)
858     if !temps_min >= min.(i) then begin
859       let rep_bis = Array.copy rep in
860       rep_bis.(i) <- rep_bis.(i) + 1;
861       let tps =
862         begin
863           try temps rep_bis mf tab_traj tab_routes e with
864             |Probleme -> Float.infinity
865           end in
866           temps_min := Float.min (!temps_min) tps;
867           if (!temps_min) = tps then trajet_min := i
868           end;
869       done;
870       for i = 0 to nb_traj - 1 do
871         min.(i) <- (!temps_min *. (float_of_int(e)) +. mini.
872           (i)) /. (float_of_int(e+1));
873       done;
874       rep.(!trajet_min) <- rep.(!trajet_min) + 1;
875       tab_rep.(e) <- Array.copy rep;
876     done;
877   end;
878   tab_rep
879
880
881
882 let liste_prix_anarchie (rep_e:repartition_trajet array)
883   (rep_s:repartition_trajet array)(mf:mat_adj_fonction_temps)
884   (tab_traj:tab_trajet)(tab_routes:tab_route) : (int*float) array =
885   let t = Array.make (Array.length rep_e) (0,0.) in
886   for i = 0 to (Array.length rep_e) - 1 do
887     let temps_e = try temps rep_e.(i) mf tab_traj tab_routes i
888     with
889       |Probleme -> 1. in
890       let temps_s = try temps rep_s.(i) mf tab_traj tab_routes i
891     with
892       |Probleme -> 1. in
893       t.(i) <- (i,temps_e/.temps_s)
894   done;
895
896 (*
897  let liste_diff_prix_anarchie_sans_i (r_route:list_routes)
898    (voit_max:int)(i:int) : (int*float) list =
899    let g = graphe_trajet r_route in
900    let g_sans_i = graphe_trajet_sans_i r_route i in
901    let lt = rem_liste (liste_trajet g) in
902    let lt_sans_i = rem_liste (liste_trajet g) in
903    let ltre = liste_temps_rep_ego (mat_temps_route g) lt 1
904      nb_voiture in
905    let ltre_sans_i = liste_temps_rep_social (mat_temps_route
906      g_sans_i) lt_sans_i 1 nb_voiture in
907    let ltrs = liste_temps_rep_social (mat_temps_route g) lt 1
908      nb_voiture in
909    let ltrs_sans_i = liste_temps_rep_social (mat_temps_route
910      g_sans_i) lt_sans_i 1 nb_voiture in
911    let lpa = liste_prix_anarchie ltre ltrs in
912    let lpa_sans_i = liste_prix_anarchie ltre_sans_i ltrs_sans_i in
913    let liste = ref [] in
914    for i = 0 to voit_max-1 do
915      liste:=(i,(snd(List.nth lpa_sans_i i))-.(snd((List.nth lpa
916      i))))::!liste
917    done;
918    List.rev !liste
919  *)
920
921
922
923
924 let prix_anarchie_max (lpa:(int*float) array) : (int*float) =
925   let max = ref 0. in
926   let max_nb = ref 0 in
927   for i = 1 to (Array.length lpa)-1 do
928     let (nb,t) = lpa.(i) in
929     if t > !max
930       then begin max:=t; max_nb:=nb end
931     else ()
932   done;
933   (!max_nb,!max)
934
935
936
937 let prix_anarchie_moy (lpa:(int*float) array) : float =
938   let som = ref 0. in
939   for i = 1 to (Array.length lpa) - 1 do
940     let (_,t) = lpa.(i) in
941     som := !som +. t
942   done;
943   !som /. float_of_int(Array.length lpa - 1)

```

```

937
938
939
940 let prix_anarchie_modif (lpa:(int*float) array) : int =
941   let nb_min = ref (Array.length lpa) in
942   for i = 1 to (Array.length lpa)-1 do
943     let (v,t) = lpa.(i) in
944       if t>=1. && v < !nb_min then nb_min := v
945   done;
946   !nb_min
947
948
949
950 let mini (mf:mat_adj_fonction_temps)(tab_trajets:tab_trajet)
951   (tab_routes:tab_route) =
952   let nb_trajets = Array.length tab_trajets in
953   let rep : repartition_trajet = Array.make nb_trajets 0 in
954   let min = Array.make nb_trajets (0.,0) in
955   for i = 0 to nb_trajets - 1 do
956     rep.(i) <- 1;
957     min.(i) <- (temps rep mf tab_trajets tab_routes 1,i);
958     rep.(i) <- 0;
959   done;
960   Array.sort compare min;
961   min
962
963 let coupe_tab (tab:(float*int) array)(n:int) =
964   let t = Array.make n 0 in
965   for i = 0 to n-1 do
966     let _,j = tab.(i) in
967     t.(i) <- j;
968   done;
969   t
970
971
972
973 (*34 - 31 - 1750 voit*)
974
975 let dep = 34
976 let arr = 31
977 let voit_tot = 1750
978 let g = graphe_trajet tab_routes
979 let mf = mat_temps_route g tab_routes
980 let t_traj = tab_trajet g dep arr tab_routes
981 let m_traj = coupe_tab (mini mf t_traj tab_routes) 50
982 let tab_traj = Array.init 50 (fun x -> t_traj.((m_traj).(x)))
983 let ltre = liste_rep_ego mf tab_traj tab_routes voit_tot
984 let ltrs = liste_rep_social mf tab_traj tab_routes voit_tot
985 let lpa = liste_prix_anarchie ltre ltrs mf tab_traj tab_routes
986
987 let trsi (i:int) =
988   let t = Array.make (Array.length tab_routes - 1) (0,0,0.,0) in
989   for j = 0 to Array.length tab_routes - 1 do
990     if j < i then t.(j) <- tab_routes.(j)
991     else if j = i then ()
992     else t.(j-1) <- tab_routes.(j)
993   done;
994   t
995
996
997
998
999
1000 let i_min (n:int)(v:int) =
1001   let min_t = ref Float.infinity in
1002   let min_i = ref (-1) in
1003   for i = 0 to n do
1004     let tab_routes_sans_i = trsi i in
1005     let g_sans_i = graphe_trajet tab_routes_sans_i in
1006     let t_traj_sans_i = tab_trajet g_sans_i dep arr
1007       tab_routes_sans_i in
1008     let m_traj_sans_i = coupe_tab (mini mf t_traj_sans_i
1009       tab_routes_sans_i) 50 in
1010     let tab_traj_sans_i = Array.init 50 (fun x -> t_traj_sans_i.
1011       ((m_traj_sans_i).(x))) in
1012     let ltre_sans_i = liste_rep_ego mf tab_traj_sans_i
1013       tab_routes_sans_i v in
1014     let t = temps ltre_sans_i.(v) mf tab_traj_sans_i
1015       tab_routes_sans_i v in
1016     print_int i;
1017     print_string " : ";
1018     print_float t;
1019     print_newline();
1020     if t < !min_t then begin
1021       min_t:=t;
1022       min_i:=i;
1023     end;
1024   done;
1025   !min_i,(!min_t-.(temps ltre.(v) mf tab_traj tab_routes v))
1026
1027
1028
1029 let i_min2 (n:int)(v:int) =
1030   let tab_routes_sans_i = trsi n in
1031   let g_sans_i = graphe_trajet tab_routes_sans_i in
1032   let t_traj_sans_i = tab_trajet g_sans_i dep arr
1033     tab_routes_sans_i in
1034   let m_traj_sans_i = coupe_tab (mini mf t_traj_sans_i
1035     tab_routes_sans_i) 50 in
1036   let tab_traj_sans_i = Array.init 50 (fun x -> t_traj_sans_i.
1037     ((m_traj_sans_i).(x))) in
1038   let ltre_sans_i = liste_rep_ego mf tab_traj_sans_i

```

```

1030 (*let ltrs_sans_i = liste_rep_social mf tab_traj_sans_i
1031   tab_routes_sans_i v in
1032   *)
1033
1034
1035 let ltt (r:repartition_trajet array)(mf:mat_adj_fonction_temps)
1036   (tab_traj:tab_trajet)(tab_routes:tab_route) =
1037   let t = Array.make (Array.length r) (0,0.) in
1038   for i = 0 to Array.length r - 1 do
1039     t.(i) <- i,temps r.(i) mf tab_traj tab_routes i;
1040   done;
1041   t
1042   (*
1043     let ltrs_sans_i = liste_rep_social mf_sans_i tab_traj_sans_i
1044       tab_routes_sans_i nb_voit
1045     let lpa_sans_i = liste_prix_anarchie ltre_sans_i ltrs_sans_i
1046       mf_sans_i tab_traj_sans_i tab_routes_sans_i
1047     let (pama_v_sans_i,pama_t_sans_i) = prix_anarchie_max
1048       lpa_sans_i
1049     let pamo_sans_i = prix_anarchie_moy lpa_sans_i
1050   *)
1051
1052 let graphique (lpa:(int*float) array) =
1053   let (pama_v,pama_t) = prix_anarchie_max lpa in
1054   let pamo = prix_anarchie_moy lpa in
1055   let padiff = prix_anarchie_modif lpa in
1056   let ec = if (padiff<300) then 0 else padiff/2 in
1057   let padiff2 = if ec = 0 then 0 else padiff in
1058
1059 let u = Graphics.open_graph "" in
1060 u;
1061 let mv x y = Graphics.moveto x y in
1062 let mv2 x y = Graphics.lineto x y in
1063
1064 let f1 x =
1065   mv 90 (150+x*150);
1066   mv2 110 (150+x*150) in
1067
1068 let f2 x =
1069   mv (200+x*100) 110;
1070   mv2 (200+x*100) 90 in
1071
1072 let calc_x a =
1073   int_of_float(100. +. (float_of_int(a-
1074     padiff2+ec)/.float_of_int(voit_tot-padiff2+ec))*1200.) in
1075
1076 let calc_y t =
1077   int_of_float(150. +. (t-.1.)*(600.)/(pama_t-.1.)) in
1078
1079 let x_max = (calc_x pama_v) in
1080 let y_max = (calc_y pama_t) + 20 in
1081
1082
1083
1084 let param =
1085   Graphics.set_window_title "Prix de l'anarchie en fonction du
1086   nombre de voitures";
1087   Graphics.resize_window 1500 900;
1088   mv 100 100;
1089   mv2 1400 100;
1090   mv 1360 120;
1091   mv 1360 80;
1092   mv 1400 100;
1093   mv 100 800;
1094   mv2 80 760;
1095   mv 120 760;
1096   mv2 100 100;
1097   for i=0 to 4 do
1098     f1 i;
1099     mv (Graphics.current_x ()-45) (Graphics.current_y ()-5);
1100     Graphics.draw_string(string_of_float(1.+
1101       (((Float.round(pama_t*.100.))/.100.)-.1.)/.
1102       4.)*.float_of_int(i)));
1103   done;
1104   for j=0 to 11 do
1105     f2 j;
1106     mv (Graphics.current_x ()-5) (Graphics.current_y ()-20);
1107     Graphics.draw_string(string_of_int((voit_tot-
1108       padiff2+ec)/12)*(j+1)+(padiff2-ec)));
1109   done;
1110   mv 95 75;
1111   Graphics.draw_string(string_of_int(padiff2-ec));
1112   mv 1300 50;
1113   Graphics.draw_string("Nombre de voitures");
1114   mv 65 830;
1115   Graphics.draw_string("Prix de l'anarchie");
1116   mv 100 150;
1117   Graphics.set_color 180 in
1118 param;
1119 let flechemax =
1120   mv x_max y_max;
1121   Graphics.lineto(x_max-20)(y_max+30);
1122   mv x_max y_max;
1123   Graphics.lineto(x_max+20)(y_max+30);

```

```
1122 Graphics.lineto(x_max+20)(y_max+30);
1123 mv x_max y_max;
1124 Graphics.lineto(x_max)(y_max+60);
1125 mv (x_max - 115) (y_max + 100);
1126 Graphics.draw_string("Le prix de l'anarchie maximum est : ");
1127 Graphics.draw_string(string_of_float((Float.round(pama_t*.
100.))/.100.));
1128 mv (x_max - 115) (y_max + 80);
1129 Graphics.draw_string("Il est atteint pour ");
1130 Graphics.draw_string(string_of_int(pama_v));
1131 Graphics.draw_string(" voitures.") in
1132 flechemax;
1133 let param2 =
1134 mv 100 150;
1135 let lpa_der = ref 1. in
1136 for i = (padiff2-ec) to (voit_tot-1) do
1137 let (v,tt) = lpa.(i) in
1138 let t = if tt < 1. then !lpa_der else begin lpa_der:=tt; tt
1139 end in
1140 if (i mod ((voit_tot-padiff2+ec)/240 + 1) = 0) then
1141 mv2 (calc_x v) (calc_y (t));
1142 if (i mod ((voit_tot-padiff2+ec)/240 + 1) = 0) then begin
1143 Graphics.set_color 255;
1144 Graphics.fill_circle (calc_x v) (calc_y (t)) 5;
1145 Graphics.set_color 180;
1146 end
1147 done in
1148 param2;
1149 ()
1150 let graphique2 (lt1:(int*float) array) (lt2:(int*float) array) =
1151 let n_v = Array.length lt1 in
1152 let ec = 0 in
1153 let padiff2 = 0 in
1154 let a =
1155 let aa = Array.make (Array.length lt1) 0. in
1156 for i = 0 to (n_v-1) do
1157 let (v,tt) = lt1.(i) in
1158 let (_,tt2) = lt2.(i) in
1159 aa.(i) <- tt/.tt2;
1160 done;
1161 aa in
1162 let tmax =
1163 let aa = a in
1164 let m = ref Float.neg_infinity in
1165 for i = 0 to n_v - 1 do
1166 m:=max aa.(i) !m;
1167 done;
1168 !m
1169 in
1170
1171
```