

# Construction de tables horaires ferroviaires et mesure de leurs robustesses

Lucas Tabary, n° 49842

TIPE, session 2020-2021

10 juin 2021

Le réseau ferré en France, c'est :

- 4 000 000 d'utilisateurs par jour ;
- une organisation encore largement manuelle.

## Questionnement

Assurer la fiabilité de ces réseaux ?

- 1 Introduction au problème
- 2 Méthodes de résolution
  - Programmation linéaire
  - Algorithme génétique
- 3 Éléments d'études de la robustesse
  - Approche proposée
  - Étude des résultats
- 4 Conclusion

# Choix de modélisation du TTP

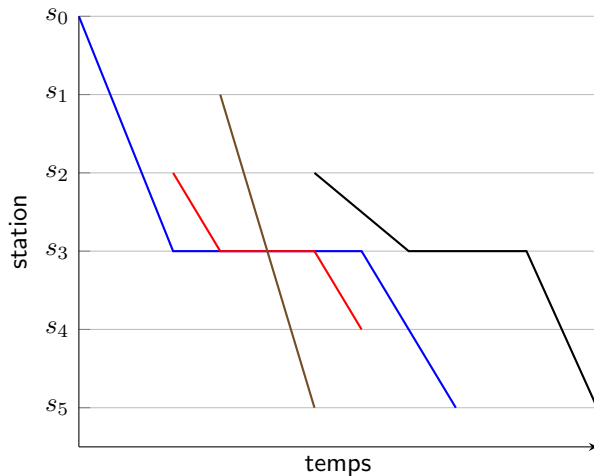
```
1  type probleme = {  
2    s: int; t: int;  
3    trajets: (int * int) array;  
4    gares: int array array;  
5    lignes: (int * int) array;  
6    types: int array  
7    (* ... *)  
8  }  
9  
10 type solution = {  
11   trajets: (int * int) array;  
12   horaires: (int * int) array array  
13 }
```

Description en OCaml

Problème  
d'optimisation  
combinatoire :  
paramètres d'études  
nécessairement  
précis.

- Étude d'une unique ligne ;
- Déplacement dans un seul sens, *etc.*

# Représentation usuelle



# Programmation linéaire : motivations

- Problème entièrement descriptible par des équations linéaires ;
- convergence en temps fini vers une solution optimale ;
- connaissance d'algorithmes efficaces (algorithme du simplexe).

## Exemple de condition

$$\forall t, \forall s, v_{\max, t}(\text{arr}_{t,s} - \text{dep}_{t,s}) \geq d_s$$

## Formalisation

$$\exists A, b, \mathcal{S} = \{x \mid Ax \leq b\}$$
$$x^* = \operatorname{argmin}_{x \in \mathcal{S}} c(x)$$

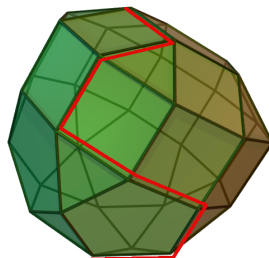


Figure – Approche pour 3 variables

# Algorithme génétique : motivations

Approche par biomimétisme :

- reproduction ;
- mutation ;
- sélection.

Intérêt calculatoire :

- temps polynomial assuré ;
- bons résultats expérimentaux.

# Algorithme génétique : Mise en place pour le TTP

**Limitation** type solution inadapté

**Choix** utilisation d'un type  
consignes pour  
construire des solutions

## Exemple

À partir de la consigne  
ci-dessous

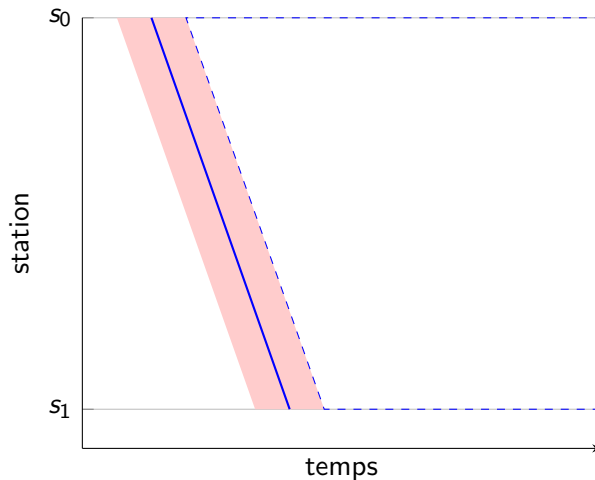
$t_3 > t_2 > t_0 > t_1$  au  
départ de  $s_2$ .

```
1 type consignes = {
2   priorites = int array array;
3   vmax = int array array
4 }
```

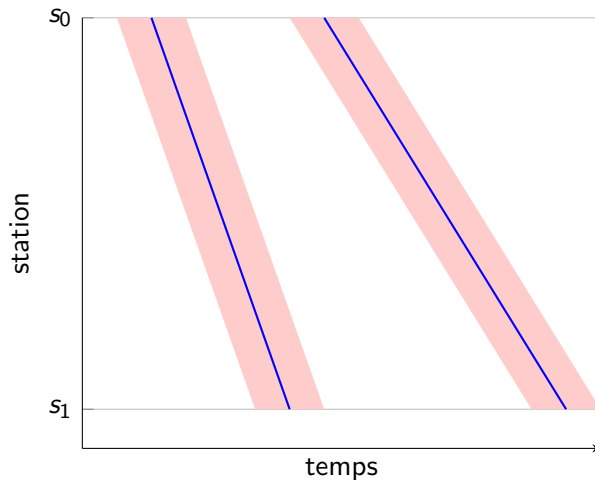
$s_0$	(0, 2)	(*, *)	(*, *)	(*, *)	$s_0$	1	3	0	2
$s_1$	(2, 4)	(8, 9)	(*, *)	(*, *)	$s_1$	3	1	2	0
$s_2$	(4, 6)	(9, 10)	(6, 10)	(14, 17)	$s_2$	3	2	0	1
$s_3$	(18, 20)	(10, 11)	(14, 16)	(21, 23)	$s_3$	0	1	2	3
$s_4$	(20, 22)	(11, 2)	(*, *)	(23, 25)	$s_4$	0	2	1	3



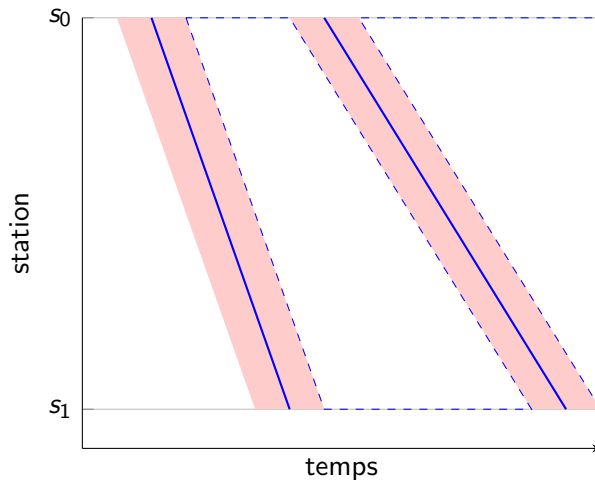
# Algorithme génétique : construction avec consignes



# Algorithme génétique : construction avec consignes



# Algorithme génétique : construction avec consignes



# Algorithme génétique : reproduction et mutation

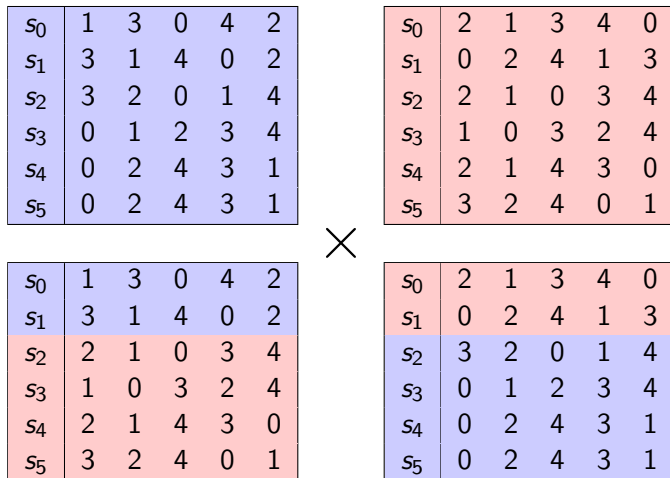


Figure – Reproduction de deux consignes

# Algorithme génétique : reproduction et mutation

**Reproduction** mélange des choix de priorité entre les parents

**Mutation** mélange sur un trajet entre deux stations des priorités, avec une certaine probabilité

$s_0$	1	3	0	4	2
$s_2$	3	1	4	0	2
$s_3$	3	2	0	1	4
$s_3$	0	1	2	3	4
$s_4$	0	2	4	3	1
$s_5$	0	2	4	3	1

→

$s_0$	1	3	0	4	2
$s_2$	3	1	4	0	2
$s_3$	2	1	0	3	4
$s_3$	0	1	2	3	4
$s_4$	0	2	4	3	1
$s_5$	0	2	4	3	1

Figure – Mutation d'une consigne

# Approches proposées

- Possibilités d'amélioration vis-à-vis de la robustesse ;
- Indicateur de ces améliorations ?

Approche par déviation : fragilité

$$\sigma \in \mathcal{S}, W_\sigma \subset \mathcal{S}'$$

$$F(\sigma) := \sum_{\tau \in W_\sigma} (c(\sigma) - c(\chi(\tau)))^2$$

Mesure par déviation	Dénombrement des points critiques
<i>ex-poste</i> : justifiée	<i>ex-ante</i> : approximation à justifier
Coût plus élevé, $O(nst \cdot c(s, t))$	Coût moindre, $O(st^2 \log t)$

Figure – Comparaison des mesures

# Points critiques

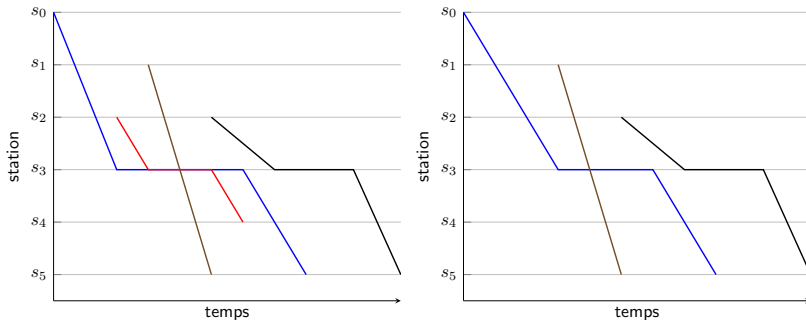


Figure – Mise en évidence par un retard d'un point critique

# Pertinence des points critiques

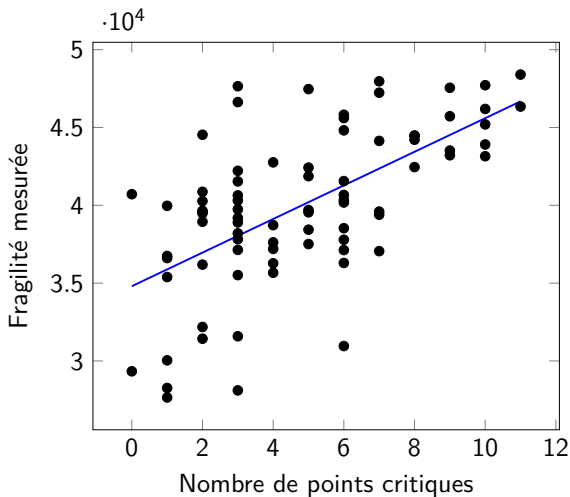


Figure – Recherche d'une corrélation entre les mesures ( $r = 0,62$ )






# Conclusion

- **Résultats**

- ajout d'autres indicateurs *ex-ante* ?
- choix de modélisation très limitants, pertinence dans un cas plus général ?
- piste intéressante dans son ensemble, chemins explorés par la littérature.

- **Limites & perspectives**

- trop peu de données d'exemple ;
- ajout d'autres éléments de modélisation (solutions cycliques, doubles voies, dépassements, etc.).

-  E. V. Andersson, A. Peterson, and J. Törnquist Krasemann.  
Quantifying railway timetable robustness in critical points.  
*Journal of Rail Transport Planning & Management*,  
3(3):95–110, 2013.  
Robust Rescheduling and Capacity Use.
-  D. Arenas, R. Chevrier, S. Hanafi, and J. Rodriguez.  
Solving the Train Timetabling Problem, a mathematical model  
and a genetic algorithm solution approach.  
In *6th International Conference on Railway Operations  
Modelling and Analysis (RailTokyo2015)*, Tokyo, Japan, Mar.  
2015.
-  A. Caprara, M. Fischetti, and P. Toth.  
Modeling and solving the train timetabling problem.  
*Operations Research*, 50(5):851–861, 2002.

# Définition du problème — TTP.ml

Lines 1–11 / 113

```
(* Représentation d'un TTP *)
* s est le nombre de gares, t le nombre de trains. c_arret et
  c_marche sont
* les coups à l'arrêt et en marche des trains (ne dépendent
  donc pas des trains)
* trajets indique la gare de départ et d'arrivée de chaque
  train. Le dernier trajet
* est donc le numéro de la dernière station, moins 1. *)
type probleme = {
  nom: string;
  s: int; t: int; c_arret: int; c_marche: int;
  k: int; (* coefficient pour calcul du coût *)
  priorites: int array;
  tmax: int; (* durée maximale d'une solution *)
```

# Définition du problème — TTP.ml

Lines 11–21 / 113

```
tmax: int; (* durée maximale d'une solution *)
departs_initiaux: int array; (* temps minimaux de départ des
    trains *)
trajets: (int * int) array; (* deb, fin *)
gares: int array array; (* temps_mini pour chaque train dans
    chaque gare *)
lignes: (int * int) array; (* distance minimale entre deux
    trains, longueur *)
types: int array (* vitesse maximale *)
}

(* Représentation d'une solution au TTP *
 * trajets indique pour chaque train les premières et dernières
   gares du trajet.
```

# Définition du problème — TTP.ml

Lines 21–31 / 113

```
* trajets indique pour chaque train les premières et dernières
   gares du trajet.
* Si celles-ci sont égales cela signifie que le train n'est
   pas programmé.
* Les valeurs de horaires.(i) "situées" avant la gare initiale
   ou après la
* gare finale sont arbitraires. *)
type solution = {
  nom: string;
  trajets: (int * int) array;
  horaires: (int * int) array array
}
```

# Définition du problème — TTP.ml

Lines 31–41 / 113

```
(* Vérifie qu'un problème est sémantiquement correct *)
let est_probleme {s; t; trajets; gares; lignes; c_marche;
  c_arret} =
  Array.length trajets = t && Array.length gares = s &&
  Array.length lignes = s - 1 && c_marche >= 0 && c_arret >= 0
  &&
  Array.length gares = s && Array.length gares.(0) = t

(* Vérifie qu'une solution à un TTP donné est sémantiquement
   correcte *)
let est_solution {s; t; trajets; depart_initiaux; gares;
  lignes; types; tmax} {trajets=trajets'; horaires} =
  (* vérifie que les trajets ne se font pas plus loin que prévu
   initialement *)
```

# Définition du problème — TTP.ml

Lines 41–51 / 113

```
(* vérifie que les trajets ne se font pas plus loin que prévu initialement *)
let est_trajet_reel (deb, fin) (deb', fin') =
  deb = deb' && deb' <= fin' && fin' <= fin
in
(* vérifie que chaque train ne roule pas plus vite que possible *)
let est_trajet_bonne_vitesse s' (deb, fin) vmax (dep, arr) =
  let longueur = snd lignes.(s') in
  (not (deb <= s' && s' < fin)) || (longueur <= (vmax * (arr - dep)) && arr < tmax)
in
(* vérifie qu'aucun train n'est trop proche d'un autre (nécessite un tri préalable) *)
let est_parcours_ligne_correct (marge, longueur) portion =
```

# Définition du problème — TTP.ml

Lines 51–61 / 113

```
let est_parcours_ligne_correct (marge, longueur) portion =  
  let aux (valid, dep, arr) (dep', arr') =  
    (valid && (dep' - dep >= marge) && (arr' - arr >= marge),  
      dep', arr')  
  and fst (a, b, c) = a in  
  (* On aimerait pouvoir utiliser - infini à la place de  
    -marge ci-dessous, car cela correspond à une  
    * inégalité toujours vérifiée, mais min_int ne convient  
      pas non plus (donne résultat aberrant) *)  
  fst (List.fold_left aux (true, -marge, -marge) portion)  
in  
let est_trajet_partie_route n_trajet n_train =  
  let (deb, fin) = trajets'.(n_train) in deb <= n_trajet &&  
    n_trajet < fin  
in
```



# Définition du problème — TTP.ml

Lines 61–71 / 113

```
in
let est_partie_bonne_heure n_train =
  let (deb, fin) = trajets.(n_train) in
  (deb = fin) || fst horaires.(deb).(n_train) >=
    departes_initiaux.(n_train)

in
let b = ref true in
  (* partie des vérifications qui se font sur une liste non
     triée *)
  Array.length horaires = s - 1 && Array.length horaires.(0) =
    t &&
  Array.length trajets = t &&
  Array.for_all2 est_trajet_reel trajets trajets' && (
  for n_train = 0 to t - 1 do
```

# Définition du problème — TTP.ml

Lines 71–81 / 113

```
for n_train = 0 to t - 1 do
  b := !b && est_partie_bonne_heure n_train
done;
for n_station = 0 to s - 2 do
  b := !b && Arraymod.for_all3 (est_trajet_bonne_vitesse
    n_station) trajets' types horaires.(n_station)
done; !b) &&
  (* partie des vérifications qui se font sur une liste triée *)
  (let nh = Array.mapi (fun i x -> Arraymod.liste_filtre
    (est_trajet_partie_route i) x) horaires in
    Array.for_all2 est_parcours_ligne_correct lignes nh)
```

# Définition du problème — TTP.ml

Lines 81–91 / 113

```
(* Détermine le coût d'une solution à un TTP donné, en la
    supposant correcte *)
* La manière dont le calcul est mené est peu naturelle car la
  structure de donnée
* est ici inadaptée: il faudrait ici avoir 'horaires' définie
  comme la transposée
* de la définition actuelle. *)
let cout {trajets; priorites; c_arret; c_marche; k}
    {trajets=trajets'; horaires} =
  let cout_trajet n_train (deb, fin) =
    let t_marche = ref 0 in (* Peu élégant mais plus rapide *)
    for i = deb to fin - 1 do
      t_marche := !t_marche + (fun (x, y) -> y - x)
        horaires.(i).(n_train)
    done;
```

# Définition du problème — TTP.ml

Lines 91–101 / 113

```
done;
let t_tot = (snd horaires.(fin - 1).(n_train)) - (fst
    horaires.(deb).(n_train)) in
    c_marche * !t_marche + c_arret * (t_tot - !t_marche)
in
let penalite_trajet_incomplet (deb, fin) (deb', fin') p =
    p * (fin' - fin) / (fin - deb)
in
(* On pénalise pour chaque train non programmé, en fonction
    de leur priorité *)
Arraymod.sum (Arraymod.map3 penalite_trajet_incomplet trajets
    trajets' priorites)
(* On détermine le temps d'activité de chaque train, en
    marche et à l'arrêt *)
+ k * Arraymod.sum (Array.mapi cout_trajet trajets)
```

# Définition du problème — TTP.ml

Lines 101–111 / 113

```
+ k * Arraymod.sum (Array.mapi cout_trajet trajets)
```

# Définition du problème — TTP.ml

Lines 111–121 / 113

# Algorithme génétique — genetique.ml

Lines 1–11 / 214

```
open TTP
```

```
(* Initialisation du module aléatoire *)
```

```
let _ = Random.self_init ()
```

```
(* D'une consigne on peut déduire une unique table horaire, à  
l'aide de appliquer_consigne
```

```
* priorites = [|2; 4; 5; ..|] si 2 est prioritaire devant 4,  
prioritaire devant 5, etc.
```

```
* vmax.(i).(t) est la vitesse maximale de t sur le trajet i  
(entre la gare i et i+1) *)
```

```
type consignes = {
```

```
  nom: string; (* nom de la solution *)
```

# Algorithme génétique — genetique.ml

Lines 11–21 / 214

```
nom: string; (* nom de la solution *)
priorites: int array array;
vmax: int array array;
mutable cout: int; (* cout de la solution associé *)
mutable est_solution_calculee: bool
}

(* Ensemble de consignes qui évolue à chaque étape de
   l'algorithme génétique *)
type population = {
  mutable n: int;
```



# Algorithme génétique — genetique.ml

Lines 21–31 / 214

```
mutable n: int;  
nmax: int;  
pop: consignes array  
}  
  
(* Consigne par défaut, remplit les cases de population non  
   utilisées *)  
let default = {  
  nom = "default";  
  priorites = [| [| 0 |] |];  
  vmax = [| [| 0 |] |];
```

# Algorithme génétique — genetique.ml

Lines 31–41 / 214

```
vmax = [| [| 0 |] |];  
cout = max_int;  
est_solution_calculée = true  
}  
  
(* Construit de manière déterministe une solution à partir d'un  
TTP et d'une consigne  
* comme détaillée dans le rapport, la solution obtenue est par  
construction une  
* solution valide, que l'on n'a pas besoin de tester. *)  
let appliquer_consigne {s; t; trajets; gares; departs_initiaux;  
lignes; tmax} {nom; priorites; vmax} =  
  let trajets' = Array.copy trajets and departs_min =  
    Array.copy departs_initiaux in
```

# Algorithme génétique — genetique.ml

Lines 41–51 / 214

```

let trajets' = Array.copy trajets and depart_min =
  Array.copy depart_initiaux in
let horaires = Array.make_matrix (s - 1) t (0, 0) in
(* ajouter_train ajoute un train sur le trajet au départ de
   la gare n_trajet et renvoie
   * la liste des espaces de parcours encore disponibles après
   ajout éventuel du train *)
let rec ajouter_train n_trajet l n_train =
  let (deb, fin) = trajets'.(n_train) and (marge, dist) =
    lignes.(n_trajet) in match l with
  | 1 when not (deb <= n_trajet && n_trajet < fin) -> 1
  | [] -> let (deb, _) = trajets'.(n_train) in
    trajets'.(n_train) <- (deb, n_trajet); []
  | (depmin, depmax, arrmin, arrmax) :: q ->

```

# Algorithme génétique — genetique.ml

Lines 51–61 / 214

```
| (depmin, depmax, arrmin, arrmax) :: q ->
  let dep = max depmin departs_min.(n_train) in
  let arr = dep + dist / vmax.(n_trajet).(n_train) in
  if dep <= depmax && arrmin <= arr && arr <= arrmax
  then begin
    let t_min = gares.(n_trajet + 1).(n_train) in
    horaires.(n_trajet).(n_train) <- (dep, arr);
    departs_min.(n_train) <- arr + t_min;
    if dep - marge <= depmin || arr - marge <= arrmin
      (* plus de place à gauche *)
    then (dep + marge, depmax, arr + marge, arrmax)
      :: q
    else if dep + marge >= depmax || arr + marge >=
      arrmax (* plus de place à droite *)
```

# Algorithme génétique — genetique.ml

Lines 61–71 / 214

```
    else if dep + marge >= depmax || arr + marge >=
      arrmax (* plus de place à droite *)
    then (depmin, dep - marge, arrmin, arr - marge)
      :: q
    else (depmin, dep - marge, arrmin, arr - marge)
      ::
        (dep + marge, depmax, arr + marge, arrmax)
      :: q
  end
  else (depmin, depmax, arrmin, arrmax) ::
    ajouter_train n_trajet q n_train
```

```
in
for s' = 0 to s - 2 do
  (* ajoute successivement chaque train sur le trajet au
    départ de la gare s' *)
  ignore (Array.fold_left (ajouter_train s') [(0, tmax, 0,
    tmax)] priorities.(s'))
done; {nom; trajets=trajets'; horaires}
```

# Algorithme génétique — genetique.ml

Lines 71–81 / 214

```
done; {nom; trajets=trajets'; horaires}
```

```
(* Étant donnée une consigne, fournit une autre consigne, qui a  
    subi une  
* mutation des priorités pour au plus chaque trajet. *)  
let une_mutation mut_par_ligne {nom; priorites; vmax} =  
  let priorites' = Arraymod.copym priorites and vmax' =  
    Arraymod.copym vmax in  
  let permutation a =  
    let n = Array.length a in  
    let i = Random.int n and j = Random.int n in  
    let t = a.(i) in
```

# Algorithme génétique — genetique.ml

Lines 81–91 / 214

```
    let t = a.(i) in
    a.(i) <- a.(j); a.(j) <- t
in
let choix = Random.int (Array.length priorites) in
for i = 0 to Random.int mut_par_ligne do
    permutation priorites'.(choix)
done;
{nom = nom ^ "m"; priorites = priorites'; vmax = vmax';
  cout = max_int; est_solution_calculee = false}
```

# Algorithme génétique — genetique.ml

Lines 91–101 / 214

```
let une_descendance {nom=nom1; priorites=priorites1; vmax=vmax1}
                    {nom=nom2; priorites=priorites2;
                     vmax=vmax2} =
  let n = Array.length priorites1 in
  let coupure = Random.int n in
  let priorites1' = Array.init n (fun i -> Array.copy (if i <
    coupure
      then priorites1.(i) else priorites2.(i)))
  and priorites2' = Array.init n (fun i -> Array.copy (if i >=
    coupure
      then priorites1.(i) else priorites2.(i)))
  and vmax1' = Array.init n (fun i -> Array.copy (if i < coupure
    then vmax1.(i) else vmax2.(i)))
```



# Algorithme génétique — genetique.ml

Lines 101–111 / 214

```

    then vmax1.(i) else vmax2.(i)))
and vmax2' = Array.init n (fun i -> Array.copy (if i >=
    coupure
    then vmax1.(i) else vmax2.(i)))
in
({nom = nom1 ^ "g"; priorites = priorites1'; vmax = vmax1';
  cout = max_int; est_solution_calculée = false},
 {nom = nom2 ^ "g"; priorites = priorites2'; vmax = vmax2';
  cout = max_int; est_solution_calculée = false})

(* Mélange de Knuth, renvoie une permutation au hasard à n
   éléments *)
let melange_p n =
  let a = Array.init n (fun i -> i) in

```

# Algorithme génétique — genetique.ml

Lines 111–121 / 214

```
let a = Array.init n (fun i -> i) in
for i = n - 1 downto 1 do
  let j = Random.int (i + 1) in
  let t = a.(i) in
  a.(i) <- a.(j); a.(j) <- t
done; a
```

```
(* former des paires d'éléments tous distincts entre 0 et n - 1
   *)
```

```
let former_paires n =
  let a = melange_p n in
```

# Algorithme génétique — genetique.ml

Lines 121–131 / 214

```
let a = melange_p n in
let rec aux i =
  if i < n - 1
  then (a.(i), a.(i + 1)) :: aux (i + 2)
  else []
in aux 0
```

```
let generer_solutions_initiales (pop_init, rho, mut) {nom; s;
  t; priorites; types} =
  let p = Arraymod.indices_tries priorites in
  let priorites' = Array.init (s - 1) (fun i -> Array.copy p)
```

# Algorithme génétique — genetique.ml

Lines 131–141 / 214

```
let priorities' = Array.init (s - 1) (fun i -> Array.copy p)
and vmax = Array.init (s - 1) (fun i -> Array.copy types) in
let pop_init_copie = int_of_float (rho *. (float_of_int
    pop_init))
and pop = Array.make (4 * pop_init) default
and copie = {
    nom = nom ^ "-g";
    priorities = priorities';
    vmax;
    cout = max_int;
    est_solution_calculee = false
} in
```

# Algorithme génétique — genetique.ml

Lines 141–151 / 214

```
} in
for i = 0 to pop_init_copie - 1 do
  pop.(i) <- copie
done;
for i = pop_init_copie to pop_init - 1 do
  pop.(i) <- une_mutation mut copie
done;
{n = pop_init; nmax = 4 * pop_init; pop}

let cout_consignes probleme consignes =
```

# Algorithme génétique — genetique.ml

Lines 151–161 / 214

```
let cout_consignes probleme consignes =  
  if consignes.est_solution_calculée  
  then consignes.cout  
  else begin  
    let c = cout probleme (appliquer_consigne probleme  
                           consignes) in  
    consignes.est_solution_calculée <- true;  
    consignes.cout <- c; c  
  end
```

```
let compare_consignes probleme c1 c2 =
```

# Algorithme génétique — genetique.ml

Lines 161–171 / 214

```
let compare_consignes probleme c1 c2 =  
  let cc1 = cout_consignes probleme c1 and cc2 = cout_consignes  
    probleme c2 in  
  compare cc1 cc2  
  
(* Sélectionne, après tri, les rang_max meilleurs individus, et  
  remplace le reste  
  * par des solutions par défaut *)  
let selection rang_max probleme population =  
  Array.sort (compare_consignes probleme) population.pop;  
  population.n <- min population.n rang_max;  
  for i = population.n to population.nmax - 1 do
```

# Algorithme génétique — genetique.ml

Lines 171–181 / 214

```
for i = population.n to population.nmax - 1 do
  population.pop.(i) <- default
done
```

```
(* avec une certaine probabilité, modifie les solutions
   aléatoirement *)
```

```
let mutation (p_mut, mut_par_ligne) population =
  let nv = ref 0 and nn = population.n in
  for i = 0 to nn - 1 do
    if Random.float 1. <= p_mut then begin
      population.pop.(nn + !nv) <- une_mutation mut_par_ligne
      population.pop.(i);
```



# Algorithme génétique — genetique.ml

Lines 181–191 / 214

```
        population.pop.(nn + !nv) <- une_mutation mut_par_ligne
            population.pop.(i);
    incr nv
end
done; population.n <- nn + !nv
```

```
(* Ajoute de nouveaux individus en croisant les traits des
   individus présents *)
let reproduction population =
  let nn = population.n in
  let rec ajouter_descendance = function
    | [] -> ()
```

# Algorithme génétique — genetique.ml

Lines 191–201 / 214

```
| [] -> ()  
| (p, m) :: r ->  
    let papa = population.pop.(p) and mama =  
        population.pop.(m) in  
    let (alice, bob) = une_descendance papa mama in  
    population.pop.(nn) <- alice;  
    population.pop.(nn + 1) <- bob;  
    population.n <- nn + 2  
in ajouter_descendance (former_paires nn)  
  
(* Fait évoluer un ensemble de solutions selon le concept  
   d'algorithme génétique.  
   * Les paramètres sont:
```

# Algorithme génétique — genetique.ml

Lines 201–211 / 214

```
* Les paramètres sont:  
* nb_gen: nombre de générations (itérations)  
* pop_init: population initiale  
* rho: part de la population initiale qui utilise la méthode  
  de copie  
* p_mut: probabilité pour un individu de muter pour en donner  
  un autre  
* mut_par_ligne: nombre de transpositions maximal pouvant  
  s'appliquer à  
*           une liste de priorité pendant une mutation *)  
let genetique (nb_gen, pop_init, rho, p_mut, mut_par_ligne)  
  probleme =  
  let p = generer_solutions_initiales (pop_init, rho, 3 *  
    mut_par_ligne) probleme in  
  for i = 1 to nb_gen do begin  
    reproduction p;
```

# Algorithme génétique — genetique.ml

Lines 211–221 / 214

```
reproduction p;  
mutation (p_mut, mut_par_ligne) p;  
selection pop_init probleme p  
end done; p
```

# Mesures de robustesse — robuste.ml

Lines 1–11 / 100

```
open TTP
```

```
let _ = Random.self_init ()
```

```
(* Associe à toute solution une n-fragilité  
 * tc est le temps caractéristique des perturbations *)  
let fragilite n (probleme, {trajets; horaires}) =  
  (* plus grande des marges du problème: *)  
  let tc = fst (Array.fold_left max (min_int, min_int)  
    probleme.lignes) in  
  let c = cout probleme {nom=""; trajets; horaires} in
```

# Mesures de robustesse — robustesse.ml

Lines 11–21 / 100

```
let c = cout probleme {nom=""; trajets; horaires} in
let t = probleme.t in
let f = ref 0 in
for i = 1 to n do
  let n_train_retard = Random.int t in
  let (deb, fin) = trajets.(n_train_retard) in
  if fin > deb then begin
    let n_trajet_retarde = deb + (Random.int (fin - deb)) in
    let vmax = probleme.types.(n_train_retard) in
    let trajets' = Array.copy trajets
    and horaires' = Arraymod.copypm horaires in
```

# Mesures de robustesse — robuste.ml

Lines 21–31 / 100

```
and horaires' = Arraymod.copym horaires in

let (depi, arri) =
  horaires.(n_trajet_retarde).(n_train_retard) in
horaires'.(n_trajet_retarde).(n_train_retard) <- (depi,
  arri + tc);

let rec ajouter_retard n_trajet = if n_trajet < fin then
  begin
    let (marge, dist) = probleme.lignes.(n_trajet_retarde)
      in
    let (_, arrp) = horaires'.(n_trajet -
      1).(n_train_retard)
    and (dep, arr) = horaires.(n_trajet).(n_train_retard)
    and tmin = probleme.gares.(n_trajet).(n_train_retard) in
    if dep - arrp < tmin then begin (* le retard n'est pas
      amorti par le temps d'arrêt en gare *)
```

# Mesures de robustesse — robuste.ml

Lines 31–41 / 100

```

if dep - arrp < tmin then begin (* le retard n'est pas
    amorti par le temps d'arrêt en gare *)
let dep' = arr + tmin in
    (* on essaie de rattraper le retard dans le cas où le
    train n'était pas
    * à vitesse maximale: *)
let arr' = if vmax * (arr - dep') >= dist then arr
    else dep' + dist / vmax in

let est_train_retarde_prioritaire n_autre_train
    (depo, arro) =
    if (n_autre_train <> n_train_retard) && (abs (depo
        - dep') < marge || abs (arro - arr') < marge)
    then
    if probleme.priorites.(n_autre_train) <
        probleme.priorites.(n_train_retard) then
    let (debo, fino) = trajets'.(n_autre_train) in
    trajets'.(n_autre_train) <- (debo, n_trajet);
    true

```



# Mesures de robustesse — robustesse.ml

Lines 41–51 / 100

```
        trajets'.(n_autre_train) <- (debo, n_trajet);
        true
    else false
else true
in
if Arraymod.for_alli est_train_retarde_prioritaire
    horaires.(n_trajet) then begin
    horaires'.(n_trajet).(n_train_retard) <- (dep',
        arr');
    ajouter_retard (n_trajet + 1)
    end
end
end in ajouter_retard (n_trajet_retarde + 1);
let c' = cout probleme {nom=""; trajets=trajets';
    horaires=horaires'} in
```

# Mesures de robustesse — robustesse.ml

Lines 51–61 / 100

```

    let c' = cout probleme {nom=""; trajets=trajets';
        horaires=horaires'} in
    f := !f + abs (c - c')
end
done; (float_of_int !f) /. (float_of_int n)

```

```

(* Associe à chaque solution son nombre de points critiques
 * tc est l'intervalle de temps dans lequel on cherche des
   paires
 * de points dits critiques *)
let npc (probleme, {trajets; horaires}) =
  (* plus grande des marges du problème: *)

```

# Mesures de robustesse — robustesse.ml

Lines 61–71 / 100

```
(* plus grande des marges du problème: *)
let tc = fst (Array.fold_left max (min_int, min_int)
  probleme.lignes) in
let est_trajet_partie_route n_trajet n_train =
  let (deb, fin) = trajets.(n_train) in deb <= n_trajet &&
    n_trajet < fin
in
let horaires_triees = Array.mapi (fun i x ->
  Arraymod.liste_filtre (est_trajet_partie_route i) x)
  horaires in
let rec compter_npc = function
  | [] | [_] -> 0
  | (dep1, arr1) :: (dep2, arr2) :: q -> (if dep2 - dep1 < tc
    then 1 else 0)
    + (compter_npc ((dep2, arr2) :: q))
in
```

# Mesures de robustesse — robustesse.ml

Lines 71–81 / 100

```

in
Array.fold_left (+) 0 (Array.map compter_npc horaires_triees)

(* Détermine deux tableaux (np-fragilité et npc) pour un
   tableau de
* solutions (jointes avec leurs problèmes), les enregistre à
* l'emplacement 'fichier', puis effectue une régression
   linéaire
* pour obtenir le coefficient de corrélation. *)
let construire_relation fichier np a_solutions =
  let a_fragilite = Array.map (fragilite np) a_solutions
  and a_npc = Array.map (fun a -> float_of_int (npc a))
    a_solutions in

```

# Mesures de robustesse — robustesse.ml

Lines 81–91 / 100

```
and a_npc = Array.map (fun a -> float_of_int (npc a))
    a_solutions in
let n = Array.length a_npc in
let oc = open_out fichier in
for i = 0 to n - 1 do
    Printf.fprintf oc "%f□%f\n" a_npc.(i) a_fragilite.(i)
done; close_out oc;
(* Calculs usuels pour une régression linéaire: *)
let covfn = Arraymod.cov a_fragilite a_npc
and varf = Arraymod.var a_fragilite
and varn = Arraymod.var a_npc
and moyf = Arraymod.moy a_fragilite
```

# Mesures de robustesse — robustesse.ml

Lines 91–101 / 100

```
and moyf = Arraymod.moy a_fragilite
and moyn = Arraymod.moy a_npc
in
let rr = (covfn *. covfn) /. (varf *. varn) in
let a = covfn /. varn in
let b = moyf -. a *. moyn in
print_string ("r^2=" ^ (string_of_float rr) ^ ", a=" ^
              (string_of_float a) ^ ", b=" ^
              (string_of_float b));
(rr, a, b)
```

# Extension de Array — arraymod.ml

Lines 1–11 / 79

```
open Array
```

```
(* Extension du module Array  
 * La structure des fonctions est inspirée du code source  
 * originale des fonctions qui leur ressemblent (map2 pour  
 * map3 par exemple). *)
```

```
let map3 f a b c =  
  let la = length a in  
  let lb = length b in  
  let lc = length c in
```

# Extension de Array — arraymod.ml

Lines 11–21 / 79

```
let lc = length c in
if la <> lb || lb <> lc then
  invalid_arg "Arraymod.map3: arrays must have the same
    length"
else begin
  if la = 0 then [[]] else begin
    let r = make la (f (unsafe_get a 0) (unsafe_get b 0)
      (unsafe_get c 0)) in
    for i = 1 to la - 1 do
      unsafe_set r i (f (unsafe_get a i) (unsafe_get b i)
        (unsafe_get c i))
    done;
    r
  end
end
```



# Extension de Array — arraymod.ml

Lines 21–31 / 79

```
    end
  end

let for_alli p l =
  let n = length l in
  let rec loop i =
    if i = n then true
    else if p i (unsafe_get l i) then loop (succ i)
    else false in
  loop 0
```

# Extension de Array — arraymod.ml

Lines 31–41 / 79

```
let for_all3 p l1 l2 l3 =  
  let n1 = length l1  
  and n2 = length l2  
  and n3 = length l3 in  
  if n1 <> n2 || n2 <> n3 then invalid_arg "Arraymod.for_all3"  
  else let rec loop i =  
    if i = n1 then true  
  else if p (unsafe_get l1 i) (unsafe_get l2 i) (unsafe_get l3  
    i) then loop (succ i)  
    else false in  
  loop 0
```

# Extension de Array — arraymod.ml

Lines 41–51 / 79

```
    loop 0

let sum = fold_left (+) 0
let sum' = fold_left (+.) 0.

(* moyenne d'un tableau de flottants *)
let moy a =
  let n = length a in
  (sum' a) /. (float_of_int n)

(* covariance de deux tableaux de flottants *)
```

# Extension de Array — arraymod.ml

Lines 51–61 / 79

```
(* covariance de deux tableaux de flottants *)
let cov a b =
  let aa = moy a and bb = moy b in
  let ab = Array.map2 (fun x y -> (x -. aa) *. (y -. bb)) a b in
  moy ab

let var a = cov a a

(* renvoie la liste des indices associés au plus grandes
    valeurs,
    * dans l'ordre décroissant *)
let indices_tries a =
```

# Extension de Array — arraymod.ml

Lines 61–71 / 79

```
let indices_tries a =  
  let a' = mapi (fun i e -> (e, i)) a in  
  let compare' x y = - (compare x y) in  
  sort compare' a'; map (fun (e, i) -> i) a'  
  
(* copie "en profondeur" un tableau de tableaux *)  
let copym (a: 'a array array) =  
  Array.init (Array.length a) (fun i -> Array.copy a.(i))  
  
(* conserve dans une liste les éléments d'un tableau  
 * dont les indices vérifient un prédicat *)
```

# Extension de Array — arraymod.ml

Lines 71–81 / 79

```
* dont les indices vérifient un prédicat *)  
let liste_filtre predicat a =  
  let n = Array.length a in  
  let rec aux i = if i < n  
    then if predicat i  
      then a.(i) :: aux (i + 1)  
      else aux (i + 1)  
    else []  
  in List.sort compare (aux 0) (* la liste renvoyée est triée *)
```