

# Code source du programme.

Le programme a été séparé en six modules. Les modules les plus importants sont celui qui calcule le minorant ainsi que les fréquences idéales (cf première partie), celui qui implémente les algorithmes de dissémination de données et enfin celui qui simule un serveur traitant les requêtes de clients. Les autres modules sont un module qui regroupe les données communes aux autres modules, un module permettant de changer dynamiquement les données de la simulation et enfin un module qui permet de lancer le programme et de sauvegarder les résultats de la simulation dans des fichiers.

Chaque module est séparé en un fichier \*.mli, qui contient l'interface de ce module (ie : les données et les programmes accessibles à partir d'autres modules) et en un fichier \*.ml qui contient le corps des programmes. Ceci permet de faciliter la compilation du programme.

## D.1- Calcul du minorant :

```
(*****Fichiers Minorant.mli et Minorant.ml:*****)

value minorant : int -> float vect -> float vect -> float vect
and calc_min : int -> float vect -> float vect -> float
and calc_min_message : int -> float vect -> float vect -> int -> float;;

(*****Calcul du
minorant*****)

let Abs x=
  if x<.0. then -.x else x;;

(*Calcul de lambda dans le minorant, très rapide->ne pas hésiter à prendre
une précision élevée*)
(*Newton fonction, dérivée, approximation allouée, point de départ*)
let rec Newton f f' epsilon x=
  if Abs(f(x))<=.epsilon then x
  else Newton f f' epsilon (x-.f(x)/.f'(x));;

(*Calcule le minorant en relaxant les contraintes Taui>=1*)
(*minorant_relaxe nombre de canaux,table des popularités, table des coûts*)
let minorant_relaxe w p c=
  let m=vect_length p in
  let sigma=ref 0. in
  for i=0 to m-1 do
    sigma:=!sigma+.sqrt(p.(i)/.(2.*.c.(i)))
  done;
  (*Calcul de lambda, si jamais sigma n'est pas dans la frontière*)
  let lambda=if !sigma<=float_of_int(w) then 0.
    else ((*Comme la fonction objectif est convexe décroissante,
pour que Newton converge, il faut lui entrer
une valeur telle que f x soit < 0*)
    let essai=ref (float_of_int m /. (2.*.float_of_int
(w*w))) in
    let f=(fun x-> let r=ref 0. in for i=0 to m-1 do r:=!
r+.sqrt(p.(i)/.(2.*.c.(i)+.x)) done;
float_of_int w -. !r) in
    while f !essai >. 0. do essai:=!essai/.100. done;
  (*Appel de Newton avec une précision de 10-12*)
  Newton f (fun x-> let r=ref 0. in for i=0 to m-1 do
```

```

(2.*.c.(i)+.x)**(-1.5)/.2.
r:=!r+.sqrt(p.(i))*
done;
!r)
(10.**(-12.)) !essai) in
(*Calcul des Tau*)
let Tau=make_vect m 0. in
for i=0 to m-1 do
  Tau.(i)<-sqrt((2.*.c.(i)+.lambda)/.p.(i))
done;
Tau;;

(*Le type est utilisé par l'algo iter_minorant*)
type message={Popu:float;Cout:float};;

(*Lorsque minorant_relaxé renvoie un Tau<1, on vire le message et un canal
et on itère*)
(*iter_minorant nb_canaux, table des messages*)
let iter_minorant w m_vect=
  let m=vect_length m_vect in
  let Vire=make_vect m 0 in
  let Sol=make_vect m 0. in
  let nb_vire=ref 0 in
  let virer i=
    incr nb_vire;
    for j=i to m-1 do
      Vire.(j)<-Vire.(j)+1
    done in
  let b=ref false in
  while not !b do
    b:=true;
    (*On construit le tableau de message en enlevant ceux qui ont été
virés*)
    let P=make_vect (m- !nb_vire) 0. and C=make_vect (m- !nb_vire) 0. in
    for i=0 to m- !nb_vire-1 do
      P.(i)<-m_vect.(i+Vire.(i)).Popu;
      C.(i)<-m_vect.(i+Vire.(i)).Cout;
    done;
    let Tau=minorant_relaxe (w- !nb_vire) P C in
    for i=0 to m- !nb_vire-1 do
      if Tau.(i)<1. then (Sol.(i+Vire.(i))<-1.;virer (i+Vire.(i));b:=false)
    done;
    if !b then (
      (*On entre les derniers Tau* - ceux qui ne sont pas égaux à 1*)
      for i=0 to m- !nb_vire-1 do
        Sol.(i+Vire.(i))<-Tau.(i)
      done
    )
  done;
  Sol;;

(*Calcule les Tau, avec ou sans coût*)
(*minorant nb_canaux, popu, couts*)
let minorant w p c=
  let m=vect_length p in
  let vide={Popu=0.;Cout=0.} in
  let vect=make_vect m vide in
  for i=0 to m-1 do
    vect.(i)<-{Popu=p.(i);Cout=c.(i)}

```

```

done;
iter_minorant w vect;;

let minor_sans_cout w p=
  let m=vect_length p in
  minorant w p (make_vect m 0.);;

(*Calcul le minorant, avec ou sans coût de diffusion*)
let calc_min w p c=
  let m=vect_length p in
  let Tau=minorant w p c in
  let min=ref 1. in
  for i=0 to m-1 do
    min:=!min+.p.(i)*.Tau.(i)/.2.+c.(i)/.Tau.(i)
  done;
  !min;;

let calc_min_sans_cout w p=
  let m=vect_length p in
  let Tau=minor_sans_cout w p in
  let min=ref 1. in
  for i=0 to m-1 do
    min:=!min+.p.(i)*.Tau.(i)/.2.
  done;
  !min;;

(*Calcule le temps d'attente optimal pour un message donné*)
let calc_min_message w p c i=
  let m=vect_length p in
  let Tau=minorant w p c in
  1.+Tau.(i)/.2.+c.(i)/.Tau.(i);;

```

## D.2- Données communes aux modules:

```

(*****Fichiers Variables.mli et Variables.ml:*****)
(*Types*)
(*file_simul permet de calculer le temps moyen d'attente.
Premier sert pour les FIFOs, indique le premier message demandé*)
type file_simul= {mutable Nb_clients:int;mutable Attente_cum:float;mutable
Premier:int};;
exception Trouve of int;;

value nb_canaux : int ref
and nb_messages : int ref
and popularite : float vect ref
and popu_cum : float vect ref
and duree : float vect ref
and couts : float vect ref
and cumul : float vect -> float vect
and normalise : float vect -> float vect
and zipf : int -> float vect
and unif : int -> float -> float vect
and init_popularite : unit -> unit
and init_duree : unit -> unit
and init_couts : unit -> unit
and arrondir : float -> float
and puissance : int -> int -> int;;

(*****Variable contient les types, algo ou constantes utilisées
par plusieurs des modules*****)

```

```

(*Serveur*)
let nb_canaux= ref 1;;
let nb_messages= ref 1;;

(*Calcul des coûts et des popularités, dans un tableau*)
(*Messages de même longueur, distribution suivant la loi de Zipf*)
let popularite=ref (make_vect !nb_messages 0.);; (*la longueur de
popularite peut changer*)
let popu_cum=ref (make_vect !nb_messages 0.);; (*ce qui explique pourquoi
on prend une référence*)
let duree=ref (make_vect !nb_messages 1.);;
let couts=ref (make_vect !nb_messages 0.);;

(*Construction des tableaux*)
(*Construction d'une tableau de somme cumulée à partir d'un tableau de
float*)
let cumul p=
  let n=vect_length p in
  let p_cum=make_vect n 0. in
  p_cum.(0)<- p.(0);
  for i=1 to n-1 do
    p_cum.(i)<- p_cum.(i-1)+. p.(i);
  done;
  p_cum;;

(*Normalisation*)
let normalise p=
  let n=vect_length p in
  let r=ref 0. in
  for i=0 to n-1 do
    r:=!r+. p.(i)
  done;
  for i=0 to n-1 do
    p.(i)<-p.(i)/. !r
  done;
  p;;

(*Crée un tableau suivant la loi de Zipf:le nieme message a une popularite
proportionnelle à 1/n*)
let zipf n=
  let l=make_vect n 0. in
  for i=0 to n-1 do
    l.(i)<-1./.(float_of_int(i+1))
  done;
  l;;

(*unif crée un tableau de n éléments qui différent au plus de epsilon,sert
pour avoir des popularités uniformes*)
let unif n epsilon=
  let l=make_vect n 0. in
  let r=ref 0. in
  for i=0 to n-1 do
    l.(i)<-1./.(float_of_int n)+.random__float epsilon -. epsilon;
    if l.(i)<.0. then l.(i)<-0.;
    r:=!r+l.(i)
  done;
  l;;

```

```

let init_popularite()=
  popularite:=normalise(zipf !nb_messages);
  popu_cum:=cumul !popularite;;
init_popularite();;

let init_duree()=
  duree:=make_vect !nb_messages 1.;;

let init_couts()=
  couts:=make_vect !nb_messages 0.;;

(*Sous-prog utilisés*)
let arrondir x=floor(100.*x)/.100.;;
let rec puissance x y=
  let r=if y>0 then puissance x (y/2) else 1 in
  if y mod 2 = 1 then x*r*r
  else r*r;;

```

### D.3- Les algorithmes de dissémination de données:

```

(*****Fichiers Algorithmes.mli et Algorithmes.ml:*****)

#open "Variables";;

(*Pour différencier les types, on est obligé de noter différemment les
labels message*)
type message_dissemine={Message1:int;Place:float};; (*type utilisé par
l'algo de tri*)
type message_arbre={Message2:int;Freq:int};; (*type utilisé pour construire
l'arbre*)
type arbre=
  |Arbre of branche * branche
  |Feuille of message_arbre
and branche={mutable Couleur:bool;Branche:arbre};;

value ordonancement : message_dissemine vect ref
and T_nbor : float ref;;

value determine_message0 : file_simul vect -> int
and Tau : float vect ref
and init_Tau : unit -> unit
and init_proba : unit -> float vect
and determine_message1 : unit -> int
and determine_message2 : float vect -> int
and init_ordonancement : float -> unit
and determine_message3 : int -> int
and init_arbre : unit -> arbre
and determine_message4 : unit -> int
and determine_message5 : file_simul vect -> int -> int
and determine_message6 : unit -> int;;

#open "Variables";;
#open "Minorant";;

(*Algorithmes de dissémination de données*)
(*Variables utilisées:à besoin de Minorant.ml et de Variables.ml*)

(***** ALGO 0 *****)
(*Message à diffuser:méthode habituelle par FIFO*)

```

```

(*Coût linéaire en le nombre de messages*)
let determine_message0 v=
  let r=ref v.(0).Premier and indice=ref 0 in
  for i=1 to !nb_messages-1 do
    if v.(i).Premier< !r then (r:=v.(i).Premier;indice:=i)
  done;
  if !r=max_int then -1 else !indice;;

(*Calcul des proportions idéales:sert à 1 2 3 et 4**)
let Tau=ref (minorant !nb_canaux !popularite !couts);;
let init_Tau()=
  Tau:=minorant !nb_canaux !popularite !couts;;

(*Note:pour les algos qui suivent, qui implémentent les coûts, il se peut
qu'aucun message ne soit diffusé
à l'instant T. L'algo renvoie alors le nombre -1*)
(***** ALGO 1*****
(*Probabiliste: message diffusé proportionnellement à sqrt p_i*)

(*Calcule les probabilités de diffusion et les probas cumulées de
diffusion*)
let proba=ref (make_vect !nb_messages 0.);;
let proba_cum=ref (make_vect !nb_messages 0.);;
let init_proba()=
  proba:=make_vect !nb_messages 0.;
  proba_cum:=make_vect !nb_messages 0.;
  !proba.(0)<-1./.(float_of_int !nb_canaux *. !Tau.(0));
  !proba_cum.(0)<- !proba.(0);
  for i=1 to !nb_messages-1 do
    !proba.(i)<-1./.(float_of_int !nb_canaux *. !Tau.(i));
    !proba_cum.(i)<- !proba_cum.(i-1)+. !proba.(i);
  done;
  !proba;;
init_proba();;

(*Diffuse aléatoirement un message suivant les probas plus haut*)
(*Coût linéaire en le nombre de messages*)
let determine_message1()=
  let r=random_float 1. in
  try (
    for i=0 to !nb_messages-1 do
      if !proba_cum.(i)>.r then raise (Trouve i)
    done;
    -1)
  with Trouve i->i;;

(***** ALGO 2*****
(*Dérandomisation gloutonne de 1*****

(*état est le tableau des sigma_i*)
(*Coût linéaire en le nombre de messages*)
let determine_message2 etat=
  let min=ref 0. and indice=ref (-1) in
  for i=0 to !nb_messages-1 do
    let r= !couts.(i)-. !popularite.(i)*.etat.(i)*. !Tau.(i) in
    if r<=. !min then (min:=r;indice:=i)
  done;

```

```

!indice;;

(*****Tri Fusion*****)
(*Sert pour 3 et 4. ppe x y renvoie le plus petit des éléments*)
let rec fusion ppe= fun
  |[] 12 -> 12
  |l1 [] -> l1
  |(p::q as l1) (u::v as l2)-> if ppe p u then p::(fusion ppe q l2) else
u::(fusion ppe l1 v);;
let rec partage = fun
  |[] -> ([],[])
  |[p]-> ([p],[])
  |(p::q::r) -> let (l1,l2)=partage r in (p::l1,q::l2);;
let rec tri_fusion ppe = fun
  |[]->[]
  |[p]->[p]
  |l->let (l1,l2)=partage l in fusion ppe (tri_fusion ppe l1) (tri_fusion
ppe l2);;

(***** ALGO 3*****
(*Nombre d'or*****
let T_nbor=ref 1000.;; (*période des messages*)

let theta=(1.+sqrt(5.))/2.;; (*nb d'or*)
let dec x=x-.floor(x);; (*x mod 1*)

(*tri avec tri_fusion*)
let inf1 x y=x.Place<. y.Place;;

(*Construit l'ordonnancement périodique*)
(*coût T log T où T est la période*)
let construct_period T=
  (*nonT est la fréquence de diffusion du message vide*)
  let nonT=
    let r=ref 0. in
    for i=0 to !nb_messages-1 do
      r:=!r+1./.!Tau.(i)
    done;
    int_of_float(T*(float_of_int !nb_canaux -. !r)) in
  (*list_mess contient la liste des messages qu'il faudra trier, chaque
message de la liste ayant une valeur
de type freq_cum*nb d'or mod 1*)
  let list_mess=ref [] in
  for j=1 to nonT do
    list_mess:={Message1=(-1);Place=dec(float_of_int j *. theta)}:: !
list_mess
  done;
  let freq_cum=ref nonT in
  for i=0 to !nb_messages-1 do
    (*ni est la fréquence de diffusion du message i*)
    let ni=int_of_float(T/. !Tau.(i)) in
    for j=1 to ni do
      list_mess:={Message1=i;Place=dec(float_of_int (!freq_cum+j) *.
theta)}:: !list_mess
    done;
    freq_cum:=!freq_cum+ni
  done;

```

```

tri_fusion inf1 !list_mess;;

print_string "\nCalcul en cours de l'ordonancement périodique...\n";;
let ordonancement=ref (vect_of_list (construct_period 1.));;
let init_ordonancement n=
  ordonancement:=vect_of_list (construct_period n);;

(*Donne le n_ième message à diffuser*)
(*Coût:1*)
let determine_message3 n=
  let m=vect_length !ordonancement in
  !ordonancement.(n mod m).Message1;;

(***** ALGO 4*****
*****Utilisation d'arbres*****
let message_arbre_vider()={Message2= -1;Freg=0};;
let inf2=(fun x y->x.Freg<y.Freg);;

(*Parcours donne la feuille indiquée par les couleurs tout en les
switchant*)
let rec parcours arb=match arb with
|Feuille i->i
|Arbre (g,d)->
  if g.Couleur then (g.Couleur<-false;d.Couleur<-true;parcours g.Branche)
  else (d.Couleur<-false;g.Couleur<-true;parcours d.Branche);;

(*Prend en argument une liste de puissances de 2 (dans Freg) triée par
ordre croissant et dont la somme est une puissance de
deux et renvoie deux listes de puissances de 2 dont les sommes sont
égales*)
let partage l=
  let sum=list_it (fun x y->x.Freg+y) l 0 in
  constr l [] 0
where rec constr a b s=
  if s=sum/2 then (a,rev b)
  else match a with
  |[]->failwith "Entrez des puissances de deux par ordre croissant dont
la somme est une puissance de deux!"
  |p::q -> constr q (p::b) (s+p.Freg);;

(*Construit l'arbre à partir d'une liste de puissances de 2 ... *)
let rec construct_arb = fun
  |p->Feuille p
  | l ->let (l1,l2)=partage l in Arbre ({Couleur=true;Branche=construct_arb
l1},
{Couleur=false;Branche=construct_arb l2});;

(*approche x par la puissance de 2 immédiatement supérieure*)
let approx x=
  let r=puissance 2 (int_of_float (log x /. log 2.)) in
  if r>=int_of_float (ceil x) then r else 2*r;;

(*Transforme un tableau en liste de puissances de 2 ... *)
let construct_message t trou=
  let n=vect_length t in
  let min=ref t.(0) in
  for i=1 to n-1 do
    if t.(i)<. !min then min:=t.(i)

```

```

done;
for i=0 to n-1 do
  t.(i)<-t.(i)/. !min
done;
let l=ref [] and puiss_cum=ref 0 in
for i=0 to n-1 do
  let r=approx t.(i) in
  l:= {Message2=i;Freq=r} :: !l;
  puiss_cum:=!puiss_cum+r
done;
(*on rajoute les messages vides, si les coûts de diffusion sont non nuls
Sinon, on voit qu'il vaut mieux rajouter de vrais messages.
L'algo comblera les trous par le message trou sauf si trou=-2 auquel
cas il
  comble les trous en allant du premier message au dernier - cela
fonctionne mieux si la distribution est de Zipf*)
let rest=ref ((approx (float_of_int !puiss_cum)) - !puiss_cum) in
let compteur=ref 1 and num=ref 0 in
while !rest>0 do
  if !rest mod 2=1 then (if trou= -2 then (l:={Message2= !num;Freq= !
compteur}:: !l;incr num)
                        else l:={Message2= trou;Freq= !
compteur}:: !l);
  compteur:=!compteur*2;rest:=!rest/2;
done;
tri_fusion inf2 !l;;

let arb=ref (Feuille (message_arbre_vider()));;
let init_arbre()=
  let n=vect_length !Tau in
  let freq=make_vect n 0. in
  for i=0 to n-1 do
    freq.(i)<-1./.(float_of_int !nb_canaux *. !Tau.(i));
  done;
  (*Si les couts de diff ne sont pas nuls, on comble les trous par le
message vide*)
  if !couts=make_vect !nb_messages 0. then arb:=construct_arb
(construct_message freq (-2))
                        else arb:=construct_arb
(construct_message freq (-1));
  !arb;;

let determine_message4()=(parcours !arb).Message2;;

(***** ALGO 5*****
*****GLOUTON Dynamique*****)

let determine_message5 tabl t= (*tabl représente le tableau des files
d'attentes, cf simul, t représente le temps*)
  let max=ref 0. and indice=ref (-1) in
  for i=0 to !nb_messages-1 do
    let r=sqrt(float_of_int(tabl.(i).Nb_clients*(t+1)) -. tabl.
(i).Attente_cum) -. !couts.(i) in
    if r>=. !max then (max:=r;indice:=i)
  done;
  !indice;;

(***** ALGO 6 *****
(* Algorithme très mauvais utilisé pour le télétexte*)

```

```
(*Cet algo consiste à diffuser cycliquement les messages!  
D'où un temps d'attente horrible égal à nombre de messages/2*)
```

```
let message_actuel=ref (-1);;  
let determine_message6()=  
  incr message_actuel;  
  !message_actuel mod !nb_messages;;
```

## D.4- Pour changer dynamiquement la popularité des messages :

```
(*****Fichiers ChangeDyna.mli et ChangeDyna.ml:*****)
```

```
value reload : float -> unit  
and change_popu : float vect -> float vect  
and change_couts : float vect -> float vect ref;;
```

```
(**Cette partie permet de modifier la popularité des messages et le coût  
sans avoir à tout réinitialiser**)
```

```
#open "Variables";;  
#open "Algorithmes";;
```

```
(*Pour remettre les variables à jour en cas de changement de nb_messages ou  
nb_canaux*)
```

```
let reload T=  
  init_popularite(); (*les popularités suivent la loi de Zipf*)  
  init_couts(); (*les coûts sont nuls*)  
  init_Tau(); (*issu du minorant*)  
  init_proba(); (*pour l'algo random*)  
  init_ordonancement T; (*pour l'algo du nombre d'or*)  
  init_arbre() (*pour l'algo se servant des arbres*);  
  print_string "Variables initialisées avec succès!";;
```

```
(*Pour pouvoir tester les algos sur d'autres distributions que Zipf*)  
(*Pour pouvoir étudier les algos dynamiques*)
```

```
let change_popu l=  
  nb_messages:=vect_length l;  
  popularite:=normalise l;  
  popu_cum:=cumul !popularite;  
  (*Réinitialisation des algos qui dépendent évidemment tous de la  
popularité!*)  
  init_couts(); (*on réinitialise les coûts à 0 parceque le nombre des  
messages peut avoir changer*)  
  init_Tau();  
  init_proba();  
  init_ordonancement (!T_nbor);  
  init_arbre();  
  !popularite;;
```

```
let change_couts c =  
  if vect_length c <> !nb_messages then failwith "Erreur : il faut entrer  
un nombre de coûts égal au nombre de messages!";  
  couts:=c;  
  init_Tau();  
  init_proba();  
  init_ordonancement (float_of_int(vect_length !ordonancement));  
  init_arbre();
```

```
couts;;
```

## D.5- Simulation d'un serveur:

```
(*****Fichiers Simulation.mli et Simulation.ml:*****)
```

```
type popu_dyna={Popu: float vect;Duree: int};;
```

```
(*Constantes permettant d'indiquer quel algo on veut utiliser*)
```

```
value FIFO : int  
and RAND : int  
and GLOUTON : int  
and NBOR : int  
and ARBRE : int  
and DYNA : int  
and TELETEXT : int;;
```

```
value simul : int -> int -> int -> float  
and simul2 : int -> int -> float -> int -> float  
and theorie : int -> int -> float  
and theorie2 : int -> int -> int list * int vect * float vect * float  
and simul_all : int -> int -> unit  
and simul_dyna : popu_dyna vect -> int -> int -> float -> int -> float  
and simul_all_dyna : popu_dyna vect -> int -> float -> int -> unit;;
```

```
#open "Variables";;  
#open "Minorant";;  
#open "Algorithmes";;  
#open "ChangeDyna";;
```

```
(*Simulation du log des clients*)  
(*Variables utilisées:à besoin de Minorant.ml et de Variables.ml (et de  
launch.ml pour la partie dynamique*)
```

```
(***** Simulation  
*****)
```

```
(*Utilisé pour la simu. Les clients arrivent en un temps continu, d'où le  
type de Arrivee.
```

```
Numéro est utilisé pour l'algo FIFO*)  
type client={Arrivee : float;Numero : int};;
```

```
(*Sélection d'un message voulu pour un client*)
```

```
let rand_mess()=  
  let r=random__float 1. in  
  try (  
    for i=0 to !nb_messages-1 do  
      if !popu_cum.(i)>.r then raise (Trouve i)  
    done;  
    !nb_messages-1)  
  with Trouve i->i;;
```

```
(*Gestion des files d'attentes*)
```

```
let file_simul_vide()={Nb_clients=0;Attente_cum=0.;Premier=max_int};;
```

```
(*Ajoute un client à une file d'attente. Coût:1*)
```

```
let add f c=  
  f.Nb_clients<-f.Nb_clients+1;  
  f.Attente_cum<-f.Attente_cum+.c.Arrivee;  
  if c.Numero<f.Premier then f.Premier<-c.Numero;;
```

```

(*Calcule le nombre de clients dans le tableau des files d'attentes*)
let nombre_de_clients v=
  let r=ref 0 in
  for i=0 to !nb_messages-1 do
    r:=!r+v.(i).Nb_clients
  done;
  !r;;

(*Ajoute l'attente cumulée, f représentant la file d'attente pour le message
diffusé*)
(*Coût:1*)
let add_attente f t=
  float_of_int(f.Nb_clients*(t+2)) -. f.Attente_cum;; (*t+2 en prenant en
compte le temps de téléchargement*)

(*AJoute l'attente cumulée des clients n'ayant pas été servis à la fin de
la simulation*)
(*tps_attente tableau des files d'attentes, temps; Coût:linéaire en le
nombre de clients*)
let tps_attente_restant v t=
  let r=ref 0. in
  for i=0 to !nb_messages-1 do
    r:=!r+.add_attente (v.(i)) t
  done;
  !r;;
(***** La simulation:
*****)

let FIFO=0;;
let RAND=1;;
let GLOUTON=2;;
let NBOR=3;;
let ARBRE=4;;
let DYNA=5;;
let TELETEXT=6;;

let imprimer commentaire i=print_string commentaire;print_float (arrondir
i);print_newline();;
let imprimer_int commentaire i=print_string commentaire;print_int
i;print_newline();;

(*simul algorithme, longueur de la simulation, nombre de clients se loguant
par unité de temps.
La variation du nombre_de_clients ne fait varier le temps d'attente que
pour FIFO et GLOUTON_DYNA
->utilité de la dissémination de données*)
let simul algo long_simu freq_pers=
  let tabl_attente=make_vect !nb_messages (file_simul_vide()) in
  let etat=make_vect !nb_messages 0. in (*etat sert à glouton, représente
la durée depuis la dernière diffusion*)
  (*Initialisation de la table d'attente, l'initiation plus haut ne
créant qu'une seule référence*)
  for z=0 to !nb_messages-1 do
    tabl_attente.(z)<-file_simul_vide()
  done;
  let nb_clients=ref 0 and tps_attente=ref 0. in (*Pour calculer le temps
d'attente moyen*)
  for i=0 to long_simu do
    (*log des clients*)
    for j=1 to freq_pers do

```

```

        add tabl_attente.(rand_mess()) {Arrivee=float_of_int(i)
+.random__float 1.;Numero= !nb_clients};
        incr nb_clients
        done;
        (*Actualisation de l'état*)
        for j=0 to !nb_messages-1 do
            etat.(j)<-etat.(j)+.1.
        done;
        (*Diffusion dans les canaux*)
        for j=0 to !nb_canaux-1 do
            let message=(if algo=1 then determine_message1() else
                if algo=2 then determine_message2 etat else
                if algo=3 then determine_message3 (i* !nb_canaux+j)
else
                if algo=4 then determine_message4() else
                if algo=5 then determine_message5 tabl_attente i else
                if algo=6 then determine_message6 () else
                determine_message0 tabl_attente) in

            if message<> -1 then (
                etat.(message)<-0.;
                tmps_attente:=!tmps_attente+.add_attente (tabl_attente.(message))
i +. !couts.(message);
                tabl_attente.(message)<-file_simul_vide();
            )
        done;
        done;
        (*Résultats*)
        (*On a deux méthodes de calcul du temps de service moyen, qui
convergent toutes les deux pour une durée de simu
assez grande, mais la deuxième est beaucoup plus fiable.
imprimer "Temps d'attente moyen actuel: " (!tmps_attente /.
float_of_int (!nb_clients-nombre_de_clients tabl_attente));*)
        (!tmps_attente+. tps_attente_restant tabl_attente long_simu) /.
float_of_int !nb_clients;;

(*simul2 est exactement comme simul, sauf que le nombre de clients se
loguant dans une unité de temps n'est
plus entier mais probabiliste d'espérance freq_pers.
simul2 est un peu plus lent, c'est pourquoi j'ai conservé simul1
-importance de la rapidité pour des
simus se déroulant sur un long intervalle de temps
enfin, simul2 permet également de présenter les résultats par intervalles
de longueurs de temps, ce
qui est utile pour tester la variation du temps d'attente lorsque le
comportement des clients change*)
let simul2 algo long_simu freq_pers pallier=
    let client_cum=ref 0. in (*utilisé pour calculer le nombre de clients se
loguant en une unité de temps*)
    let tabl_attente=make_vect !nb_messages (file_simul_vide()) in
    (*Initialisation de la table d'attente, l'initiation plus haut ne créant
qu'une seule référence*)
    for z=0 to !nb_messages-1 do
        tabl_attente.(z)<-file_simul_vide()
    done;
    let etat=make_vect !nb_messages 0. in (*etat sert à glouton*)
    let nb_clients=ref 0 and tmps_attente=ref 0. in
    for i=1 to long_simu do
        (*log des clients*)
        client_cum:=!client_cum+.random__float (2.*.freq_pers);
        while !client_cum>.1. do

```

```

    add tabl_attente.(rand_mess()) {Arrivee=float_of_int(i)
+.random__float 1.;Numero= !nb_clients};
    incr nb_clients;
    client_cum:=!client_cum-.1.
done;
(*Actualisation de l'état*)
for j=0 to !nb_messages-1 do
    etat.(j)<-etat.(j)+.1.
done;
(*Diffusion dans les canaux*)
for j=0 to !nb_canaux-1 do
    let message=(if algo=1 then determine_message1() else
                if algo=2 then determine_message2 etat else
                if algo=3 then determine_message3 (i* !nb_canaux+j)
else
                if algo=4 then determine_message4() else
                if algo=5 then determine_message5 tabl_attente i else
                if algo=6 then determine_message6 () else
                determine_message0 tabl_attente) in

    if message<> -1 then (
        etat.(message)<-0.;
        tmps_attente:=!tmps_attente+.add_attente (tabl_attente.(message))
i +. !couts.(message);
        tabl_attente.(message)<-file_simul_vide();
    )
done;
(*résultats intermédiaires:*)
if i mod pallier=0 then (
    imprimer_int "Pallier :" i;
    imprimer_int "Clients restant à servir:" (nombre_de_clients
tabl_attente);
    imprimer "Attente restante moyenne:" ((tps_attente_restant
tabl_attente i)/.float_of_int(nombre_de_clients tabl_attente));
    print_newline());
done;
(*Résultats*)
(*imprimer "Temps d'attente moyen actuel: " (!tmps_attente /.
float_of_int (!nb_clients-nombre_de_clients tabl_attente));*)
(!tmps_attente+. tps_attente_restant tabl_attente long_simu) /.
float_of_int !nb_clients
(*tmps_attente représente l'attente moyenne de service pour un client,
le reste représente l'attente moyenne des
clients n'ayant pas encore été servis*);;

(***** Attente moyenne théorique *****)
(*Tous les algos de dissémination de donnés n'ont pas besoin de connaître
quels clients sont logués.
L'algo ci-dessous calcule le temps d'attente en arrière moyen. On peut
montrer que ce temps converge vers le
temps de service. C'est ce que l'on observe en pratique lorsqu'on compare
théorie et simul pour une longueur de
simulation d'environ 100*nb_messages
En fait, theorie calcule le coût du chemin du graphe des états*)

(*théorie algo à utiliser, longueur de la simulation*)
let theorie algo long=
    (*etat.(i) est le temps d'attente en arrière du message i*)
    let etat=make_vect !nb_messages 0. in
    let cout_tot=ref 0. in
    for i=0 to long do

```

```

(*Actualisation de l'état*)
for j=0 to !nb_messages-1 do
  etat.(j)<-etat.(j)+.1.
done;
(*Diffusion dans les canaux*)
for j=0 to !nb_canaux-1 do
  let message=(if algo=2 then determine_message2 etat else
               if algo=3 then determine_message3 (i* !nb_canaux+j)
else
               if algo=4 then determine_message4 () else
               if algo=6 then determine_message6 () else
               determine_message1()) in
  if message<> -1 then (etat.(message)<-0.;cout_tot:=!cout_tot+. !
couts.(message));
  done;
  cout_tot:=!cout_tot+.1.5; (*+1 pour le temps de téléchargement, +1/2
à cause de la continuité du temps*)
  for j=0 to !nb_messages-1 do
    cout_tot:=!cout_tot +. !popularite.(j)*.etat.(j)
  done;
done;
!cout_tot/.float_of_int long;;

(*comme simul2, theorie2 est exactement comme theorie sauf qu'il donne des
résultats plus complets
en particulier il donne la liste des messages diffuser par l'algo et le
temps d'attente pour chaque message,
ce qui permet d'étudier le comportement égalitariste ou non des
différents algos...*)
let theorie2 algo long=
  (*etat.(i) est le temps d'attente en arrière du message i*)
  let etat=make_vect !nb_messages 0. in
  let cout_tot=ref 0. in
  let messages_diffusés=ref [] in (*contient la liste des messages
diffusés*)
  let nb_message_vect=make_vect !nb_messages 0
  and cout_message=make_vect !nb_messages 0. in (*pour avoir le temps
d'attente de chaque message*)
  for i=0 to long do
    (*Actualisation de l'état*)
    for j=0 to !nb_messages-1 do
      etat.(j)<-etat.(j)+.1.
    done;
    (*Diffusion dans les canaux*)
    for j=0 to !nb_canaux-1 do
      let message=(if algo=2 then determine_message2 etat else
                   if algo=3 then determine_message3 (i* !nb_canaux+j)
else
                   if algo=4 then determine_message4 () else
                   if algo=6 then determine_message6 () else
                   determine_message1()) in
      if message<> -1 then (etat.(message)<-0.;cout_tot:=!cout_tot+. !
couts.(message);
                          nb_message_vect.(message)<-nb_message_vect.
(message)+1;
                          messages_diffusés:=message:: !
messages_diffusés);
      done;
      cout_tot:=!cout_tot+.1.5; (*+1 pour le temps de téléchargement, +1/2
à cause de la continuité du temps*)
      for j=0 to !nb_messages-1 do

```

```

        cout_tot:=!cout_tot +. !popularite.(j)*.etat.(j);
        cout_message.(j)<-cout_message.(j) +. etat.(j);
    done;
done;
for j=0 to !nb_messages-1 do
    cout_message.(j)<-cout_message.(j) /. float_of_int long +. 1.5 (*+1
pour le temps de téléchargement, +1/2 à cause de la continuité du temps*)
done;
    (rev !messages_diffusés, nb_message_vect, cout_message, !
cout_tot/.float_of_int long);;

let simul_all long freq=
    print_string "\n";
    imprimer "Minorant: " (calc_min !nb_canaux !popularite !couts);
    print_string "FIFO:\n";
    imprimer " Temps d'attente moyen asymptotique: " (simul FIFO long
freq);
    print_string "RAND:\n";
    imprimer " Théorie: " (theorie RAND long);
    imprimer " Temps d'attente moyen asymptotique: " (simul RAND long
freq);
    print_string "GLOUTON:\n";
    imprimer " Théorie: " (theorie GLOUTON long);
    imprimer " Temps d'attente moyen asymptotique: " (simul GLOUTON long
freq);
    print_string "NB OR:\n";
    imprimer " Théorie: " (theorie NBOR long);
    imprimer " Temps d'attente moyen asymptotique: " (simul NBOR long
freq);
    print_string "ARBRE:\n";
    imprimer " Théorie: " (theorie ARBRE long);
    imprimer " Temps d'attente moyen asymptotique: " (simul ARBRE long
freq);
    print_string "GLOUTON DYNAMIQUE:\n";
    imprimer " Temps d'attente moyen asymptotique: " (simul DYNA long
freq);
    print_string "TELETEXTE:\n";
    imprimer " Théorie: " (theorie TELETEXT long);
    imprimer " Temps d'attente moyen asymptotique: " (simul TELETEXT long
freq);;

(*****Simulation
dynamique*****)
(*simul_dyna permet de simuler des variations de popularité*)

let simul_dyna popu_vect algo longueur freq_pers pallier=

    let sortie1=open_out ("C:\Temp\ " ^(string_of_int algo)^ "Pallier.txt")
in
    let sortie2=open_out ("C:\Temp\ " ^(string_of_int algo)^
"ClientsRestants.txt") in
    let sortie3=open_out ("C:\Temp\ " ^(string_of_int algo)^
"TmpsAttente.txt") in
    let sortie4=open_out ("C:\Temp\ " ^(string_of_int algo)^
"Popularité.txt") in

    let m=vect_length popu_vect in
    nb_messages:=vect_length popu_vect.(0).Popu;
    let duree=ref 0 in

```

```

for i=0 to m-1 do
  duree:=!duree+popu_vect.(i).Duree
done;
(*Calcul de la popularité moyenne*)
let popu_moyen=make_vect !nb_messages 0. in
for i=0 to !nb_messages-1 do
  popu_moyen.(i)<-
    (let r=ref 0. in
     for j=0 to m-1 do
       r:=!r+.popu_vect.(j).Popu.(i)*.float_of_int(popu_vect.(j).Duree)/.
(float_of_int !duree)
     done;
    !r)
  done;
  change_popu popu_moyen;
  (*tabl_temps_attente et tabl_clients permettent de calculer le temps de
service à chaque changement
de la distribution des popularités*)
  let tabl_temps_attente=make_vect (m+1) 0. and tabl_clients=make_vect
(m+1) 0 in

  (*Simulation proprement dite, basée sur Simul2
On ne peut utiliser directement simul2 car cela reviendrait à vider les
files d'attentes à
chaque changement de popularité*)
  let client_cum=ref 0. in (*utilisé pour calculer le nombre de clients se
loguant en une unité de temps*)
  let tabl_attente=make_vect !nb_messages (file_simul_vide()) in
  (*Initialisation de la table d'attente, l'initiation plus haut ne créant
qu'une seule référence*)
  for z=0 to !nb_messages-1 do
    tabl_attente.(z)<-file_simul_vide()
  done;
  let etat=make_vect !nb_messages 0. in (*etat sert à glouton*)
  let nb_clients=ref 0 and tmps_attente=ref 0. in
  (*popularité ne sert que pour les algos et le calcul du minorant.
Lorsqu'un client se logue, pour savoir quel
message il demande, on utilise popu_cum. Donc ici on se sert de
popu_moyen pour les algos, et à chaque changement
de distribution de popularité on change uniquement popu_cum, ce qui
change uniquement les habitudes des clients
dans la simulation*)
  let popu_num=ref 0 in let popu_change=ref (popu_vect.(!
popu_num).Duree*longueur/ !duree) in
  for i=1 to longueur do
    (*log des clients*)
    client_cum:=!client_cum+.random__float (2.*.freq_pers);
    while !client_cum>.1. do
      add tabl_attente.(rand_mess()) {Arrivee=float_of_int(i)
+.random__float 1.;Numero= !nb_clients};
      incr nb_clients;
      client_cum:=!client_cum-.1.
    done;
    (*Actualisation de l'état*)
    for j=0 to !nb_messages-1 do
      etat.(j)<-etat.(j)+.1.
    done;
    (*Diffusion dans les canaux*)
    for j=0 to !nb_canaux-1 do
      let message=(if algo=1 then determine_message1 () else
if algo=2 then determine_message2 etat else

```

```

        if algo=3 then determine_message3 (i* !nb_canaux+j) else
        if algo=4 then determine_message4 () else
        if algo=5 then determine_message5 tabl_attente i else
        if algo=6 then determine_message6 () else
            determine_message0 tabl_attente) in
    if message<> -1 then (
        etat.(message)<-0.;
        tmps_attente:=!tmps_attente+.add_attente (tabl_attente.(message)) i
+. !couts.(message);
        tabl_attente.(message)<-file_simul_vide();
    )
done;
(*résultats intermédiaires:*)
if i mod pallier=0 then (
    imprimer_int "Pallier :" i;
    imprimer_int "Clients restant à servir:" (nombre_de_clients
tabl_attente);
    imprimer "Attente restante moyenne:" ((tps_attente_restant
tabl_attente i)/.float_of_int(nombre_de_clients tabl_attente));
    print_newline();

    output_string sortie1 (string_of_int i);output_string sortie1 "\n";
    output_string sortie2 (string_of_int ((nombre_de_clients
tabl_attente)));output_string sortie2 "\n";
    output_string sortie3 (string_of_float ((tps_attente_restant
tabl_attente i)/.float_of_int(nombre_de_clients
tabl_attente)));output_string sortie3 "\n"
);

    (*changement de la popularité des messages:*)
    if i= !popu_change then (
        tabl_clients.(!popu_num+1)<- !nb_clients - tabl_clients.(!popu_num)-
(nombre_de_clients tabl_attente);
        tabl_temps_attente.(!popu_num+1)<- !tmps_attente+.
(tps_attente_restant tabl_attente i)-. tabl_temps_attente.(!popu_num);
        imprimer (" Popularité n°"^string_of_int (!popu_num+1)^" ")
            (tabl_temps_attente.(!popu_num+1)/. (float_of_int
(tabl_clients.(!popu_num+1)+(nombre_de_clients tabl_attente))));
        print_newline();

        output_string sortie4 (string_of_float (tabl_temps_attente.(!
popu_num+1)/. (float_of_int (tabl_clients.(!popu_num+1)+(nombre_de_clients
tabl_attente))));output_string sortie4 "\n";

        if i<>longueur then (
            popu_cum:=cumul(normalise popu_vect.(!popu_num).Popu);
            incr popu_num;popu_change:= !popu_change + popu_vect.(!
popu_num).Duree*longueur/ !duree))
        done;
        (*Résultats*)
        output_string sortie4 (string_of_float ((!tmps_attente+.
(tps_attente_restant tabl_attente longueur)) /. (float_of_int !
nb_clients)));output_string sortie4 "\n";
        close_out sortie1;
        close_out sortie2;
        close_out sortie3;
        close_out sortie4;

        (!tmps_attente+. (tps_attente_restant tabl_attente longueur)) /.
(float_of_int !nb_clients);;

```

```

let simul_all_dyna popu_vect longueur nb_clients pallier=
  print_string "FIFO:\n";
  imprimer " Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
FIFO longueur nb_clients pallier);
  print_string "\nRAND:\n";
  imprimer " Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
RAND longueur nb_clients pallier);
  print_string "\nGLOUTON:\n";
  imprimer " Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
GLOUTON longueur nb_clients pallier);
  print_string "\nNB OR:\n";
  imprimer " Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
NBOR longueur nb_clients pallier);
  print_string "\nARBRE:\n";
  imprimer " Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
ARBRE longueur nb_clients pallier);
  print_string "\nGLOUTON DYNAMIQUE:\n";
  imprimer " Temps d'attente moyen asymptotique: " (simul_dyna popu_vect
DYNA longueur nb_clients pallier);;

```

## D.6- Pour lancer le programme:

```
(*****Fichiers Launch.mli et Launch.ml:***** *****)
```

```

value launch : unit -> unit
and result : int -> int -> float -> int -> int -> int -> float
and enregistre : int -> int -> int -> unit;;

```

```
(*****Permet de charger les modules et d'initialiser les
constantes*****)
```

```

directory "C:\Documents and Settings\Dams\Mes documents\Type\Dissémination
de Données";;
#open "Variables";;
#open "Algorithmes";;
#open "ChangeDyna";;
#open "Simulation";;

```

```

random__init (int_of_float (sys__time()));;
let io s=print_string s;print_newline();
      int_of_string(read_line());;
let io_float s=print_string s;print_newline();
              float_of_string(read_line());;

```

```
(*Pour initialiser les variables*)
```

```

let launch()=
  nb_canaux:=io "Entrez le nombre de canaux: ";
  nb_messages:=io "Entrez le nombre de messages: ";
  T_nbor:=io_float "Période de l'ordonancement? ";
  reload !T_nbor;
  print_string "\nPour démarrer la simulation: simul_all longueur
nb_de_clients";
  print_string "\nRemarque: Pour avoir des résultats fiables, il est
recommandé que";
  print_string "\nlongueur=10*nb_messages voire 100*nb_messages";
  print_string "\nLe minorant est asymptotique est n'est valable que pour
les algorithmes de dissémination de données";
  print_string "\nDes algorithmes peuvent donc avoir un temps de service
inférieur au minorant!";;

```

```

(* Comme launch sauf qu'on entre directement la valeur des variables en
argument *)
let result nbcanaux nbmessages Tnbor algo longsimu freqsimu=
  nb_canaux:=nbcanaux;
  nb_messages:=nbmessages;
  reload Tnbor;
  simul algo longsimu freqsimu;;

(**Utilisé pour faire des courbes à partir des données de la simulation**)
let enregistre debut fin step=
  let sortiel=open_out "C:\Temp\Simul2\Nombre de clients1.txt" in
  let sortie2=open_out "C:\Temp\Simul2\Glouton1.txt" in
  let pos_GLOBAL=ref debut in
  while !pos_GLOBAL<fin do
    output_string sortiel (string_of_int !pos_GLOBAL);output_string sortiel
"\n";
    let r=result 1 100 (float_of_int(!pos_GLOBAL*10)) GLOUTON 10000 !
pos_GLOBAL in
    output_string sortie2 (string_of_float (arrondir r));output_string
sortie2 "\n";
    pos_GLOBAL:=!pos_GLOBAL+step;
  done;
close_out sortiel;
close_out sortie2;;

print_string "\nPour lancer: launch()\n";;

```