- TP Recuit Simulé -

Buche Cédric/Vincent Rodin

Novembre 2016

1 Objectif

L'objectif est de comprendre le fonctionnement et l'utilisation de la technique du recuit simulé. Il s'agit ici de minimiser des fonctions objectifs. La mise en œuvre de l'algorithme et l'effet des différents paramètres seront étudiés.

2 Recuit simulé

Pour des problèmes NP complets d'optimisation (comme le problème du voyageur de commerce), on ne connait pas d'algorithme polynomial permettant une résolution de façon optimale. On va donc chercher une solution approchée de cette optimum en utilisant des heuristiques. Le recuit simulé est un algorithme basé sur une heuristique permettant la recherche de solution à un problème donné. Il permet notament d'éviter les minima locaux mais nécessite un réglage minutieux de ses paramètres.

2.1 Algorithme

L'algorithme du recuit simulé peut être représenté de la manière suivante:

- Fournir une *Solution initiale* X (configuration initiale)
- $X_{opt} = X$
- $F_{opt} = F(X_{opt})$ ($F()$: Fonction objectif)
- Fournir une $Temperature\ Initiale$
- Tant que ($Temperature > Temperature\ Finale$) faire

```
{
    tant que (Repetition > MaxRepetitionTconstant) faire:
    {
        choisir Y dans le Voisinage de (X)
        calculer Df = F(Y) − F(X)

        Si DF < 0 alors
        {
            X = Y
            Si F(X) < F(X_opt) alors
            {
                X_opt = X
                F_opt = F(X_opt)
            }
        }
        Sinon
        {
            Tirer p dans [0, 1]
            Si p ≤ exp(−Df/T)
            {
                X = Y
            }
        }
    }
    T = g(T)
}
```

- Afficher X_{opt}

2.2 Paramètres

La technique du recuit simulé est soumise à plusieurs paramètres. Nous allons proposer à l'utilisateur de définir:

1. *Temperature Initiale* (TInit)
2. *Temperature Finale* (TFin)
3. *Alpha* permettant de faire décroître la température par le biais de $g()$ (Alpha)
4. *Amplitude* permettant de définir le voisinage (Ampli)
5. Nombre de transformations à température constante (MaxRepetitions)

3 Travail demandé

3.1 Recherche d'optimum d'une fonction

L'implémentation incomplète est disponible.
Le fichier se nomme `recuitFonctions.c` (répertoire Fonctions).

1. Algorithme
   Complétez le code du fichier en incorporant l'algorithme du recuit simulé. Ce dernier doit être le plus générique possible.
2. Manipulation
   Vous disposez de 2 fonctions polynomiales:
   (a) $f_1 = x^2$
   (b) $f_2 = x^2, x \leq 3$
   $f_2 = \frac{5}{49}(x-10)^2 + 4, x > 3$
   L'objectif est de tester l'algorithme en utilisant les fonctions ci-dessus. Il s'agit notamment d'étudier l'impact de la modification des différents paramètres sur la solution trouvée et sur la convergence de l'algorithme. En choisissant judicieusement les paramètres, montrez comment l'algorithme peut aboutir à un optimum local pour f_2.
   NB : On peut remarquer que f_1 admet un minimum unique global au point $(x = 0, f_1(0) = 0)$. La fonction f_2 possède un minimum global au point $(x = 0, f_2(0) = 0)$ et un minimum local au point $(x = 10, f_2(10) = 4)$.

L'évolution de la fonction de coût est visualisée pendant exécution. Après exécution, le fichier `Cout` *contient les valeurs des différents coûts au cours du temps.*

3.2 Le problème du voyageur de commerce

Un voyageur de commerce doit visiter une seule fois un ensemble de villes et revenir chez lui en minimisant la distance parcourue.
Vous trouverez l'implémentation d'une ville (définie par une position(x,y)) et d'une carte dans les fichiers `geo.h/geo.c`.
L'implémentation incomplète est disponible.
Le fichier se nomme `recuitVoyageur.c` (répertoire Voyageur). Dans ce fichier, vous trouverez notamment des fonctions de gestion de chemins du voyageur (`genereChemin`, `calculCoutChemin`, `dessineChemin` et `transformationChemin` (à faire!).

1. Algorithme
   Afin de montrer que l'algorithme du recuit simulé est indépendant du problème, modifier la fonction de transformation `transformation` et la fonction de calcul du coût `f` pour les adapter au problème du voyageur de commerce. Vous pourrez vous aider notamment des fonctions de gestion de chemins.

2. Manipulation
   Il s'agit d'étudier l'impact de la modification des différents paramètres sur la solution trouvée et sur la convergence de l'algorithme. Il peut être intéressant de tester

   (a) différents schémas de décroissances de la température (linéaire, discret, exponentielle ...)

   (b) différents types de transformations (inversement, déplacement, échange ...)

   (c) ...

L'évolution de la fonction de coût est visualisée pendant exécution. Après exécution, le fichier `Cout` *contient les valeurs des différents coûts au cours du temps.*

4 Compte rendu à fournir

Un compte rendu sera à remettre. Il devra, en plus de vos analyses (évaluation de l'influence des différents paramètres, ...), comporter une version papier des fichiers `recuitFonctions.c` et `recuitVoyageur.c`

```
                    /* random.h : generation de nombres aleatoire */
#ifndef RANDOM_H
#define RANDOM_H

/*********************************************************************/
/********* Generation de nombres aleatoires ***************************/
/*********************************************************************/

extern void   initRandom(void);                 /* Initialisation generateur */
extern int    myRandomMinMax(int min, int max); /* Generation dans [min,max] */
extern double myRandom01(void);                 /* Generation dans [0.0,1.0] */

#endif /* RANDOM_H */
```

Optimisation de fonctions !


```
#define VISUMEILLEUR 0  /* Si 1, visu de la meilleure solution connue    */
                        /* Si 0, visu en continu de la solution courante */
#include <math.h>
#include <stdlib.h>

#include "random.h"
#include "gnuplot.h"

double T;                   /* Temperature T        */
double Ti;                  /* Temperature initiale */
double Tf;                  /* Temperature finale   */

double amplitude;           /* Parametre d'amplitude dans transformation() */
double alpha;               /* Facteur de decroissance de la temperature   */

int    NbEssais;            /* Nb total de mouvements essayes              */
int    MaxRepetitions;      /* Nb. max de repetition a temp. constante     */

FILE* fdCout;
char* fileNameCout="Cout";

FILE *fdResults;
char* fileNameResults="Resultats";

FILE* fdGnuplotCout;

/*_________  Choix de la Fonction d'Evaluation  ____________________________*/
#define F(x)      f1_1(x)  /* ... et la lere fonction exemple (ci-dessous)   */
#define FNAME     "f1_1"   /* indiquer aussi son libelle (pour impressions)  */

/*_________  Etats du Recuit  ____________________________________________*/
double x0;                      /* Etat initial */
double x;                       /* Solution courante */
double y;                       /* Solution voisine */
double xopt;                    /* Solution optimale */
double fx, fy, fxopt;           /* Valeurs */

/*_________  Fonctions Exemples  (Fonction de coût) ____________________*/
double f1_1( double t ){ return t*t; }

double f1_2( double t ){

    if (t <= 3.) return t*t;

 /* if (t >  3.) */
     return (5.*(t-10.)*(t-10.))/49. + 4.;
}

/*_________  Voisinage (modification configuration)  ____________________*/
void transformation(void){
  y = ...;                      /* y est au voisinage de x suivant amplitude */
}


/*_________  Modification température  ____________________________________*/
double g(void) {
 return( ... ) ;            /* on decroit le température en utilisant alpha */
}
```

```
/*------ visu du Cout -------- */
void visualiserCout(FILE *fd, char *fileName, int affichageObligatoire)
{
 static int i=0;

 if (i%100==0 || affichageObligatoire)
 {
#if 0
  fprintf(fd,"plot \"%s\" with impulses\n",fileName);
  fflush(fd);
#else

  FILE *fdFileName;

  fdFileName=fopen(fileName,"r");
  if (fdFileName==NULL) {
    perror("fopen (dans visualiserCout)");
    fprintf(fd,"plot \"%s\" with impulses\n",fileName);
    fflush(fd);
  }
  else {

   long ind; double val;

   fprintf(fd,"plot '-' with impulses\n");

   fscanf(fdFileName,"%ld %lf",&ind,&val);
   if (ferror(fdFileName)) perror("fscanf (dans visualiserCout)");

   while (!feof(fdFileName) && !ferror(fdFileName))
   {
    fprintf(fd,"%ld %f\n",ind,val);
    fflush(fd);
    fscanf(fdFileName,"%ld %lf",&ind,&val);
    if (ferror(fdFileName)) perror("fscanf (dans visualiserCout)");
   }

   fprintf(fd,"e\n");
   fflush(fd);

   fclose(fdFileName);

  }
#endif

 }
 i++;
}

void ecrireCout(FILE *fdCout, int abscisse, double cout)
{
 if (fdCout==NULL) return;

 fprintf(fdCout,"%d %f\n",abscisse,cout);
 fflush(fdCout);
}
```



```
/*------ Sauvegarde fichier Resultats -------- */
void PrintParameters(FILE *fd)
{
 if (fd==NULL) return;

 fprintf(fd,"%-15s %s\n"    ,"FNAME",FNAME);
 fprintf(fd,"%-15s %-.2f\n","x0",x0);
 fprintf(fd,"%-15s %-.2f\n","FNAME(x0)",F(x0));
 fprintf(fd,"%-15s %-.2f\n","Ti",Ti);
 fprintf(fd,"%-15s %-.2f\n","Tf",Tf);
 fprintf(fd,"%-15s %-.2f\n","amplitude",amplitude);
 fprintf(fd,"%-15s %-.2f\n","alpha",alpha);
 fprintf(fd,"%-15s %d\n"    ,"MaxRepetitions",MaxRepetitions);

 fprintf(fd,"\n");
 fflush(fd);
}

void PrintTitleLine(FILE *fd)
{
 int n=11;

 if (fd==NULL) return;

 fprintf(fd,"%-*s",n,"T");
 fprintf(fd,"%-*s",n,"NbEssais");
 fprintf(fd,"%-*s",n," x");
 fprintf(fd,"%-*s",n," f(x)");
 fprintf(fd,"%-*s",n," xopt");
 fprintf(fd,"%-*s",n," f(xopt)");

 fprintf(fd,"\n");
 fflush(fd);
}

void PrintALine(FILE *fd)
{
 int n=11;

 if (fd==NULL) return;

 fprintf(fd,"%-*.2f",n,T);
 fprintf(fd,"%-*d",n,NbEssais);
 fprintf(fd,"%- *.2f",n,x);
 fprintf(fd,"%- *.2f",n,fx);
 fprintf(fd,"%- *.2f",n,xopt);
 fprintf(fd,"%- *.2f",n,fxopt);
 fprintf(fd,"\n");

 fflush(fd);
}
```

```
/*_______  Initialisation  ____________________________________________*/
int main(void)
{
 char rep;

 initRandom();

 printf("f(x)=%s\n", FNAME );

 printf("Etat initial (x0) ?\n");
 scanf("%lf", &x0);

 printf("TInit ?\n");
 scanf("%lf",&Ti);

 printf("TFin ?\n");
 scanf("%lf",&Tf);

 printf("Alpha ?\n");
 scanf("%lf",&alpha);

 printf("Ampli ?\n");
 scanf("%lf",&amplitude);

 printf("MaxRepetitions ?\n");
 scanf("%d",&MaxRepetitions);

 PrintParameters(stdout);

 do {
     printf("Sauvegarde des resultats dans un fichier? (o/n)\n");
     scanf("%c",&rep);
     while (rep=='\n') { scanf("%c",&rep); }
 }while (rep!='o' && rep!='n' && rep!='O' && rep!='N');

 if (rep=='n' || rep=='N') fdResults=NULL;
 else {
       fdResults= fopen(fileNameResults, "w");
       if (fdResults==NULL) {
         fprintf(stderr,"Probleme sur fopen(\"%s\",\"w\")\n",fileNameResults);
       }
 }

 /*  Si on veut avoir un en-tete d'identification */
 PrintParameters(fdResults);
 PrintTitleLine(fdResults);

 fdCout=fopen(fileNameCout,"w");  /* Ouverture du fichier pour les couts */
 if (fdCout==NULL) {
   fprintf(stderr,"Probleme sur fopen(\"%s\",\"w\")\n",fileNameCout);
   fprintf(stderr,"=> Arret du programme\n");
   fclose(fdResults);
   exit(EXIT_FAILURE);
 }

 fdGnuplotCout=openGnuplot(NULL);            /* pipe + fork pour visu    */
 if (fdGnuplotCout==NULL) {
   fprintf(stderr,"Probleme sur openGnuplot => Arret du programme\n");
   fclose(fdResults);
   fclose(fdCout);
   exit(EXIT_FAILURE);
 }
```



```
/*__ Recuit Simule  ____________________________________________________*/

    x        = xopt  = ... ;         /* Configuration initiale */
    fx       = fxopt = ... ;         /* Cout initial           */
    T        = ... ;                 /* Temperature initiale   */
    NbEssais = 0;

    ecrireCout(fdCout,NbEssais,fx);
    visualiserCout(fdGnuplotCout,fileNameCout,1);
    PrintALine(fdResults); /* Sauvegarde configuration initiale       */

    while ( ... ) {                            /* 1er critere d'arret */

       int    rep;    /* Nb de repetitions a temperature constante    */
       double p, Df;  /* p: pour tirage aleatoire, Df: delta fonction */

       for(rep=0; rep<MaxRepetitions; rep++){ /* 2eme critere d'arret */

          ... ;                    /* transformation => y, voisin de x */
          fy = ... ;

          Df = ... ;

          if ( .... ) {                   /* Descente !!              */
                x  = ... ;                /* y devient l'etat courant */
                fx = ... ;
                if( ... ){                /* Mise a jour optimum ?    */
                    xopt  = ... ;
                    fxopt = ... ;
#if VISUMEILLEUR==1
             ecrireCout(fdCout,NbEssais,fxopt);
             visualiserCout(fdGnuplotCout,fileNameCout,1);
#endif
                }
          }
          else {                          /* Remontee : acceptee ??   */
                 p = ... ;
                 if ( ... ) {
                    x  = ... ;            /* y devient l'etat courant */
                    fx = ... ;
                 }
          }

          NbEssais++;

#if VISUMEILLEUR!=1
             ecrireCout(fdCout,NbEssais,fx);
             visualiserCout(fdGnuplotCout,fileNameCout,0);
#endif
             PrintALine(fdResults); /* Sauvegarde resulats courants */
       }

       T = ... ;                          /* modifier la temperature */

    } /* end while */

 printf("Temp=%f\n",T);
 visualiserCout(fdGnuplotCout,fileNameCout,1);
 if (fdResults!=NULL) fclose(fdResults);
 fclose(fdCout); closeGnuplot(fdGnuplotCout);
 exit(EXIT_SUCCESS);
}
```

```
#ifndef PARAM_H
#define PARAM_H

#define  NBVILLES     30
#define COTECARTE     10

/*****************************************************************/
/************ Verification contraintes sur les defines ***********/
/*****************************************************************/


#if NBVILLES>COTECARTE*COTECARTE
#error "Attention: NBVILLES>COTECARTE*COTECARTE"
#endif

#endif /* PARAM_H */
```

TSP Voyageur de commerce !!


```
                        /* geo.h : la geographie */

#ifndef GEO_H
#define GEO_H

#include "params.h"      /* Pour NBVILLES */

/* On utilise NBVILLES (genereCarte) */

/*****************************************************************/
/********* Les Villes ********************************************/
/*****************************************************************/

typedef struct { int x;
                 int y;
               }        Ville;

extern void   genereVille(Ville *ville, int coteCarte);
extern void   printVille(const Ville *ville);
extern void   dessineVille(FILE* flot, const Ville *ville);
extern void   dessineUneSeuleVille(FILE* flot, const Ville *ville);
extern double distanceVilles(const Ville *ville1, const Ville *ville2);

/*****************************************************************/
/********* Une Carte *********************************************/
/*****************************************************************/

typedef struct { Ville  villes[NBVILLES];
                 int    nbVilles;
               }                          Carte;

extern void genereCarte(Carte *carte, int coteCarte);
extern void printCarte(const Carte *carte);
extern void dessineCarte(FILE* flot,const Carte *carte);

#endif /* GEO_H */
```

```
                    /* random.h : generation de nombres aleatoire */
#ifndef RANDOM_H
#define RANDOM_H

/*********************************************************************/
/********* Generation de nombres aleatoires ***************************/
/*********************************************************************/

extern void   initRandom(void);                 /* Initialisation generateur */
extern int    myRandomMinMax(int min, int max); /* Generation dans [min,max] */
extern double myRandom01(void);                 /* Generation dans [0.0,1.0] */

#endif /* RANDOM_H */
```



```
#define VISUMEILLEUR 1  /* Si 1, visu de la meilleure solution connue    */
                        /* Si 0, visu en continu de la solution courante */
#include <math.h>
#include <stdlib.h>

#include "random.h"
#include "gnuplot.h"
#include "geo.h"

#include "params.h"   /* Pour NBVILLES et COTECARTE */

#if NBVILLES==8

 Carte carte={{{5,2},{7,3},{8,5},{7,7},{5,8},{3,7},{2,5},{3,3}}, 8};

#elif NBVILLES==16

 Carte carte={{{5,2},{6,2},{7,3},{8,4},
               {8,5},{8,6},{7,7},{6,8},
               {5,8},{4,8},{3,7},{2,6},
               {2,5},{2,4},{3,3},{4,2}}, 16};

#elif NBVILLES==30

 Carte carte={{{8,4},{4,1},{1,9},{7,6},{2,1},
               {9,8},{3,5},{4,6},{5,9},{3,9},
               {0,4},{8,5},{5,2},{6,1},{7,8},
               {2,6},{3,6},{6,5},{1,8},{7,1},
               {7,0},{7,3},{0,0},{3,1},{5,1},
               {6,0},{3,3},{1,1},{1,4},{1,6}}, 30};
#else
 Carte carte;
#endif

typedef struct { int parcours[NBVILLES];
                 int nbVilles;
               } Chemin;

/*--------------- Des fonctions de gestion de Chemins ----------------*/

void genereChemin(Chemin *chemin)
{
 int nbVillesGenerees=0;

 chemin->nbVilles=NBVILLES;

 while (nbVillesGenerees!=chemin->nbVilles)
 {
   int entier=myRandomMinMax(0,chemin->nbVilles-1);

   /* Il faut rechercher si l'entier est deja present dans le parcours */

   int i=0,trouve=0;
   for(i=0;i<nbVillesGenerees;i++)
   {
    if (chemin->parcours[i]==entier) trouve=1;
   }
   if (!trouve) chemin->parcours[nbVillesGenerees++]=entier;
 }
}
```

```
/*--*/
double calculCoutChemin(Chemin chemin)
{
 double cout=0.0;
 int    i=0;

 const Ville *villeInitiale,*villePrecedente,*villeCourante;

 cout=0.0;                  /* Si 1 seule ville, cout: 0.0 */
 if (chemin.nbVilles!=1)
 {
  villeInitiale=&carte.villes[chemin.parcours[0]];
  villePrecedente=villeInitiale;

  for(i=1;i<chemin.nbVilles;i++)
  {
   villeCourante=&carte.villes[chemin.parcours[i]];

   cout+=distanceVilles(villePrecedente,villeCourante);

   villePrecedente=villeCourante;
  }

  /* Si 2 villes, distance premiere/derniere Ville deja calculee ... */
  if (chemin.nbVilles!=2)
  {
   cout+=distanceVilles(villePrecedente,villeInitiale);
  }
 }

 return cout;
}
/*--*/
/*
void copieChemin(Chemin *cheminDst, Chemin cheminSrc)
{
 *cheminDst = cheminSrc;
}
*/
/*--*/
void dessineChemin(FILE* flot, Chemin chemin)
{
 int    i=0;

 beginPointsToGnuplot(flot,"linespoint");

 for(i=0;i<chemin.nbVilles-1;i++)
 {
  vectorGnuplot(flot, carte.villes[chemin.parcours[i]].x,
                      carte.villes[chemin.parcours[i]].y,
                      carte.villes[chemin.parcours[i+1]].x,
                      carte.villes[chemin.parcours[i+1]].y);
 }

 vectorGnuplot(flot, carte.villes[chemin.parcours[chemin.nbVilles-1]].x,
                     carte.villes[chemin.parcours[chemin.nbVilles-1]].y,
                     carte.villes[chemin.parcours[0]].x,
                     carte.villes[chemin.parcours[0]].y);

 endPointsToGnuplot(flot);
}
```



```
/*--*/
void transformationChemin(Chemin *cheminY,
                          Chemin  cheminX, int amplitude)
{
 ...
}


/*--------------- Parametres de controle du recuit --------------------*/

double T;                  /* Temperature T        */
double Ti;                 /* Temperature initiale */
double Tf;                 /* Temperature finale   */

int    amplitude;          /* Parametre d'amplitude dans transformation() */
double alpha;              /* Facteur de decroissance de la temperature   */

int    NbEssais;           /* Nb total de mouvements essayes              */
int    MaxRepetitions;     /* Nb. max de repetition a temp. constante     */

FILE* fdCout;
char* fileNameCout="Cout";

FILE *fdResults;
char* fileNameResults="Resultats";

FILE* fdGnuplotCout;
FILE* fdGnuplotChemin;

/*_________  Choix de la Fonction d'Evaluation  ____________________________*/
#define F(x)     f(x)      /* la fonction de cout (ci-dessous)             */
#define FNAME    "f"       /* indiquer aussi son libelle (pour impressions)*/

/*_________  Etats du Recuit  ____________________________________________*/

Chemin x0;                /* Etat initial      */
Chemin x;                 /* Solution courante */
Chemin y;                 /* Solution voisine  */
Chemin xopt;              /* Solution optimale */
double fx, fy, fxopt;     /* Valeurs */

/*_________  Fonctions Exemples  (Fonction de coût) ____________________*/
double f(Chemin chemin)
{
 return ... ;
}


/*_________  Voisinage (modification configuration)  ____________________*/
void transformation(void) {  /* y est au voisinage de x suivant amplitude */
  transformationChemin(&y,x,amplitude);                  /* Doonnneeee ! */
}


/*_________  Modification temperature  ____________________________________*/
double g(void) {
 return( ... ) ;          /* on decroit le temperature en utilisant alpha */
}
```

```
/*------ visu du Cout -------- */
void visualiserCout(FILE *fd, char *fileName, int affichageObligatoire)
{
 static int i=0;

 char type[]="linespoints";

 if (i%10==0 || affichageObligatoire)
 {
#if 0
  fprintf(fd,"plot \"%s\" with %s\n",fileName,type);
  fflush(fd);
#else

  FILE *fdFileName;

  fdFileName=fopen(fileName,"r");
  if (fdFileName==NULL) {
    perror("fopen (dans visualiserCout)");
    fprintf(fd,"plot \"%s\" with %s\n",fileName,type);
    fflush(fd);
  }
  else {

   long ind; double val;

   fprintf(fd,"plot '-' with %s\n",type);

   fscanf(fdFileName,"%ld %lf",&ind,&val);
   if (ferror(fdFileName)) perror("fscanf (dans visualiserCout)");

   while (!feof(fdFileName) && !ferror(fdFileName))
   {
    fprintf(fd,"%ld %f\n",ind,val);
    fflush(fd);
    fscanf(fdFileName,"%ld %lf",&ind,&val);
    if (ferror(fdFileName)) perror("fscanf (dans visualiserCout)");
   }

   fprintf(fd,"e\n");
   fflush(fd);

   fclose(fdFileName);

  }
#endif

 }
 i++;
}

void ecrireCout(FILE *fdCout, int abscisse, double cout)
{
 if (fdCout==NULL) return;

 fprintf(fdCout,"%d %f\n",abscisse,cout);
 fflush(fdCout);
}
```



```
/*------ Sauvegarde fichier Resultats -------- */
void PrintParameters(FILE *fd)
{
 if (fd==NULL) return;

 fprintf(fd,"---> Parametres\n");
 fprintf(fd,"%-15s %s\n"    ,"FNAME",FNAME);
 fprintf(fd,"%-15s %s\n","x0","...");
 fprintf(fd,"%-15s %-.2f\n","FNAME(x0)",F(x0));
 fprintf(fd,"%-15s %-.2f\n","Ti",Ti);
 fprintf(fd,"%-15s %-.2f\n","Tf",Tf);
 fprintf(fd,"%-15s %d\n","amplitude",amplitude);
 fprintf(fd,"%-15s %-.2f\n","alpha",alpha);
 fprintf(fd,"%-15s %d\n"    ,"MaxRepetitions",MaxRepetitions);

 fprintf(fd,"<---\n");
 fflush(fd);
}

void PrintTitleLine(FILE *fd)
{
 int n=11;

 if (fd==NULL) return;

 fprintf(fd,"%-*s",n,"T");
 fprintf(fd,"%-*s",n,"NbEssais");
 fprintf(fd,"%-*s",n," x");
 fprintf(fd,"%-*s",n," f(x)");
 fprintf(fd,"%-*s",n," xopt");
 fprintf(fd,"%-*s",n," f(xopt)");

 fprintf(fd,"\n");
 fflush(fd);
}

void PrintALine(FILE *fd)
{
 int n=11;

 if (fd==NULL) return;

 fprintf(fd,"%-*.2f",n,T);
 fprintf(fd,"%-*d",n,NbEssais);
 fprintf(fd,"%-*s",n," ...");
 fprintf(fd,"%- *.2f",n,fx);
 fprintf(fd,"%-*s",n," ...");
 fprintf(fd,"%- *.2f",n,fxopt);
 fprintf(fd,"\n");

 fflush(fd);
}
```

```
/*_______  Initialisation  _____________________________________________*/
int main(void)
{
 char rep;

 initRandom();

#if NBVILLES!=8 && NBVILLES!=16 && NBVILLES!=30
 genereCarte(&carte,COTECARTE);
#endif

/*
 printf("f(x)=%s\n", FNAME );

 printf("Etat initial (x0) ?\n");
 scanf("%lf", &x0);
*/

 genereChemin(&x0);  /* Configuration initiale */

 printf("TInit ?\n");
 scanf("%lf",&Ti);

 printf("TFin ?\n");
 scanf("%lf",&Tf);

 printf("Alpha ?\n");
 scanf("%lf",&alpha);

 printf("Ampli (un int) ?\n");
 scanf("%d",&amplitude);

 printf("MaxRepetitions ?\n");
 scanf("%d",&MaxRepetitions);

 PrintParameters(stdout);

 do {
     printf("Sauvegarde des resultats dans un fichier? (o/n)\n");
     scanf("%c",&rep);
     while (rep=='\n') { scanf("%c",&rep); }
 }while (rep!='o' && rep!='n' && rep!='O' && rep!='N');

 if (rep=='n' || rep=='N') fdResults=NULL;
 else {
       fdResults= fopen(fileNameResults, "w");
       if (fdResults==NULL) {
         fprintf(stderr,
                 "Probleme sur fopen(\"%s\",\"w\")\n",fileNameResults);
       }
 }

 /*  si on veut avoir un en-tete d'identification */
 PrintParameters(fdResults);
 PrintTitleLine(fdResults);
```



```
 fdCout=fopen(fileNameCout,"w");  /* Ouverture du fichier pour les couts */
 if (fdCout==NULL) {
   fprintf(stderr,"Probleme sur fopen(\"%s\",\"w\")\n",fileNameCout);
   fprintf(stderr,"=> Arret du programme\n");
   fclose(fdResults);
   exit(EXIT_FAILURE);
 }

 fdGnuplotCout=openGnuplot(NULL);             /* pipe + fork pour visu    */
 if (fdGnuplotCout==NULL) {
   fprintf(stderr,"Probleme sur openGnuplot => Arret du programme\n");
   fclose(fdResults);
   fclose(fdCout);
   exit(EXIT_FAILURE);
 }

 fdGnuplotChemin=openGnuplot(NULL);          /* pipe + fork pour visu    */
 if (fdGnuplotCout==NULL) {
   fprintf(stderr,"Probleme sur openGnuplot => Arret du programme\n");
   fclose(fdResults);
   fclose(fdCout);
   closeGnuplot(fdGnuplotCout);
   exit(EXIT_FAILURE);
 }

/*
   Recuit simule....

                A suivre....

                  A suivre....

                    A suivre....

                      A suivre....

                        A suivre....

                A suivre....

                  A suivre....

                    A suivre....

                      A suivre....

                        A suivre....

                A suivre....

                  A suivre....

                    A suivre....

                      A suivre....

                        A suivre....

*/
```

```
/*__ Recuit Simule _____________________________________________*/

    x        = xopt  = ... ;         /* configuration initiale */
    fx       = fxopt = ... ;         /* cout initial           */

    T        = ...;                  /* Temperature initiale   */
    NbEssais = 0;

    ecrireCout(fdCout,NbEssais,fx);
    visualiserCout(fdGnuplotCout,fileNameCout,1);

    dessineChemin(fdGnuplotChemin,x);

    PrintALine(fdResults);  /* Sauvegarde configuration initiale

    while ( ...) {                              /* 1er critere d'arret

       int    rep;    /* Nb de repetitions a temperature constante
       double p, Df;  /* p: pour tirage aleatoire, Df: delta fonction

     for(rep=0; rep<MaxRepetitions ; rep++){ /* 2eme critere d'arret

          ... ;                         /* transformation => y, voisin de x
          fy = ... ;

          Df = ... ;

          if ( ... ) {                      /* Descente !!              */
                x  = ... ;                  /* y devient l'etat courant */
                fx = ... ;
                if( ... ) {                 /* Mise a jour optimum ?    */
                    xopt  = ... ;
                    fxopt = ... ;
#if VISUMEILLEUR==1
                  ecrireCout(fdCout,NbEssais,fxopt);
                  visualiserCout(fdGnuplotCout,fileNameCout,1);
                  dessineChemin(fdGnuplotChemin,xopt);
#endif
             }
          }
          else {                            /* Remontee : acceptee ??   */
                 p = ... ;
                 if ( ...) {
                    x  = ... ;              /* y devient l'etat courant */
                    fx = ... ;
                 }
          }

          NbEssais++;

#if VISUMEILLEUR!=1
          ecrireCout(fdCout,NbEssais,fx);
          visualiserCout(fdGnuplotCout,fileNameCout,0);
          dessineChemin(fdGnuplotChemin,x);
#endif
          PrintALine(fdResults);     /* Sauvegarde resulats courants */
      }

      T = ... ;                          /* modifier la temperature */

    } /* end while */
```



```
 printf("Temp=%f\n",T);

 visualiserCout(fdGnuplotCout,fileNameCout,1);
 dessineChemin(fdGnuplotChemin,xopt);
 printf("Cout optimal=%f\n",fxopt);

 if (fdResults!=NULL) fclose(fdResults);

 fclose(fdCout);
 closeGnuplot(fdGnuplotCout);
 closeGnuplot(fdGnuplotChemin);

 exit(EXIT_SUCCESS);
}
```