OS01

RAPPORT DE PROJET

Relaxation Lagrangienne pour le problème du voyageur de commerce

Responsable d'UV:	Christian PRINS
Étudiant:	TA Tien Thanh
	TRAN Nguyen Anh Thu
	NGUYEN Trung Duong
Spécialité:	Master 2 OSS

Université de Technologie de Troyes

Semestre Automne 2012

I. Code-source du programme

[1]. Fonction «Random_TSP »

```
% ------------------------------------------------------------------------ %
% @Fonction     Random_TSP
% @Bref         Générer en mémoire un TSP aléatoire
% @Paramètres d'entrés
%               n               Nombre de noeuds
%               Xmax, Ymax      Largeur et longeur du rectangle danslequel
%                               répartis aléatoirement les noeuds
% @Paramètres de sortis
%               X, Y            Coordonnées des noeuds
% @Programmeur  NGUYEN Trung Duong - Étudiant Master OSS - UTT
% @Date         2013/01/11
% ------------------------------------------------------------------------ %

function [X,Y] = Random_TSP(n, Xmax, Ymax)

    X = Xmax*rand(1,n);
    Y = Xmax*rand(1,n);

end % end of function
```

[2]. Fonction «Read_TSP »

```
% ------------------------------------------------------------------------ %
% @Fonction     Read_TSP
% @Bref         Lire les données continues dans un fichier-texte de nom
%               donnés
% @Paramètres d'entrés
%               filename    Nom du fichier-texte
% @Paramètres de sortis
%               n           Nombre de noeuds
%               X, Y        Coordonnées des noeuds
% @Programmeur  NGUYEN Trung Duong - Étudiant Master OSS - UTT
% @Date         2013/01/11
% ------------------------------------------------------------------------ %

function [n, X, Y] = Read_TSP(filename)

A = textscan(filename,'%n %f %f');

n = size(A{1},1);

for i = 1:n
    X(i) = A{2}(i,1);
    Y(i) = A{3}(i,1);
end

end % end of function
```

[3]. Fonction «Compute_C »

```
% ------------------------------------------------------------------------ %
% @Fonction     Compute_C
% @Bref         Calculer la distance entre les noeuds
% @Paramètres d'entrés
%               n           Nombre de noeuds
%               X, Y        Coordonnées des noeuds
% @Paramètres de sortis
%               C           Matrice de distances entre les noeuds
% @Programmeur  NGUYEN Trung Duong – Étudiant Master OSS - UTT
% @Date         2013/01/11
% ------------------------------------------------------------------------ %

function C = Compute_C(n, X, Y)

for i = 1:n
    for j = 1:n
        C(i,j)= sqrt((X(i)-X(j))^2 + (Y(i)-Y(j))^2);
        C(i,j)= round(C(i,j));                    % arrondir la distance
    end
end

end % end of function
```

[4]. Fonction «Farthest_Insertion»

```
% ------------------------------------------------------------------------ %
% @Fonction      Farthest_Insertion
% @Bref          Application de la méthode heuristique Farthest_Insertion
%                pour calculer la distance totale du cycle parcouru
% @Paramètres d'entrés
%                n     Nombre de noeuds
%                C     Matrice de distances entre les noeuds
% @Paramètres de sortis
%                UB    Distance totale du cycle
%                T     Itinéraire du cycle (l'ordre de noeuds à traverser
%                      successivement)
% @Programmeur   NGUYEN Trung Duong – Étudiant Master OSS - UTT
% @Date          2013/01/11
% ------------------------------------------------------------------------ %

function [UB, T] = Farthest_Insertion(n, C)

cost = 0;
T(1) = 1;               % Cycle initial T =(1,1): ne contient que le dépôt
T(2) = 1;

for i = 2:n
    Free(i) = 1;        % le noeud i est dehors du cycle
    Free(1) = 0;
end

%----- boucle effectuant les (n-1) insertions -----%
for k = 1:(n-1)
```

```
    % Chercher bi(best i) le noeud libre le plus loin de T
    Dmax = -Inf('single');                  % infini négative
    for i = 2:n
        if (Free(i)==1)
            Dmin = Inf('single');           % infini positive

            for j = 1:k
                if (C(i,T(j))< Dmin)
                    Dmin = C(i,T(j));
                end
            end

            if(Dmin > Dmax)
               Dmax = Dmin;
               bi = i;
            end
        end
    end

    %----- Insère bi dans T en minimisant l'augmentation de longeur -----%
    MinCVar = Inf('single');    % Minimum cost variation

    for j = 1:k
        CVar = C(T(j), bi) + C(bi, T(j+1)) - C(T(j), T(j+1));
        if (CVar < MinCVar)
            MinCVar = CVar;
            bj = j;             % Meilleur position d'insertion
        end
    end

    % Décaler T pour faire un trou pour bi en T[bj+1]
    for i = (k+1):-1:(bj+1)
        T(i+1) = T(i);
    end

    T(bj+1) = bi;
    cost = cost + MinCVar;
    UB = cost;                  % Longueur totale du cycle
end % end for

end % end of function
```

[5]. Fonction «One_Tree »

```
% ------------------------------------------------------------------------ %
% @Fonction     One_Tree
% @Bref         Application de l'algorithme 1-arbre optimal pour calculer la
%               distance(le coût)totale du trajet parcouru
% @Paramètres d'entrés
%               n     Nombre de noeuds
%               C     Matrice de distances entre les noeuds
% @Paramètres de sortis
%               h     Distance(coût)totale du trajet parcouru
%               A     Liste des arêtes
%               D     Tableau de degrés des noeuds
% @Programmeur  NGUYEN Trung Duong - Étudiant Master OSS - UTT
```

```
% @Date          2013/01/11
% ------------------------------------------------------------------------ %
function [h, A, D] = One_Tree(n, C)

h = 0;                                  %cout initial = 0
S = zeros(1,n);
S(2)= 1;

for k = 1:(n-2)
    Cmin = Inf('single');
    for i = 1:n
        if (S(i)== 1)
            for j = 2:n
                if (S(j)==0)
                    if(C(i,j)< Cmin)
                       Cmin = C(i,j);
                       imin = i;
                       jmin = j;
                    end
                end
            end
        end
    end

    S(jmin)= 1;
    h = h + Cmin;
    A(1,k) = imin;
    A(2,k) = jmin;
end

%----- Ajouter la plus petite arête incidente a 1 -----%
Cmin = Inf('single');
for j = 2:n
    if (C(1,j)< Cmin)
        Cmin = C(1,j);
        jmin = j;
    end
end
h = h + Cmin;
A(1,n-1) = 1;
A(2,n-1) = jmin;

%----- Ajouter la 2e plus petite arête incidente a 1 -----%
Cmin = Inf('single');
for i = 2:n
    if ((C(1,i) < Cmin) & (i ~= jmin))
        Cmin = C(1,i);
        imin = i;
    end
end
h = h + Cmin;
A(1,n) = 1;
A(2,n) = imin;

%----- Établir le tableauau de degrés de chaque noeud ----%
D = zeros (2,n);
for  i = 1:n
    D(1,i) = i;
    X = (A(1,:)== i);
    Y = (A(2,:)== i);
```

```
    D(2,i) = sum(X,2) + sum(Y,2);
end

end % end of function
```

[6]. Fonction «Sous_Gradient»

```
% ------------------------------------------------------------------------ %
% @Fonction     Sous_Gradient
% @Bref         Application de la méthode Sous_Gradient pour calculer la
%               distance(le coût)totale du trajet parcouru
% @Paramètres d'entrés
%               n     Nombre de noeuds
%               C     Matrice de distances entre les noeuds
%               UB    Distance totale du cycle
%              lamda  Valeur du pas utilisée dans l'algorithme sous_gradient
% @Paramètres de sortis
%               h     Distance(coût)totale du trajet parcouru
%               A     Liste des arêtes
%               D     Tableau de degrés des noeuds
% @Programmeur  NGUYEN Trung Duong - Étudiant Master OSS - UTT
% @Date         2013/01/11
% ------------------------------------------------------------------------ %

function [h, A, D] = Sous_Gradient (n, C, UB, lamda)

LB = 0.00;
k = 0; kmax = 1000;                     % compteur d'itération
esl = 0.01;

for i = 1:n
    Pi(i) = 0.00;                       % multiplicateurs de départ
end

TSP = 0;
Norm = 0.00;

while (TSP == 0)
      k = k + 1;

      % lamda = 1/k;
      % lamda = delta * alpha^k;

      for i = 1:n
          for j = 1:n
              NewC(i,j) = C(i,j) + Pi(i) + Pi(j);
          end
      end
      [h, A, D] = One_Tree(n, NewC);
      % h: cout, A: liste des arêtes, D: tableau de degrés

      SumPi = 0.00;
      for i = 1:n
          SumPi = SumPi + Pi(i);
      end
```

```
      if ((h - 2*SumPi) > LB)
          LB = h - 2*SumPi;
      end
% Ici on est a l'optimum si LB = UB ou si One_Tree crée un cycle %
% ---------------------------------------------------------------%

      TSP = 0;
      Temp = repmat(2,1,n);

      if ((TSP == 0) && (LB < UB))
         Norm = 0.00;
         for i = 1:n
             Norm = Norm + (D(2,i)-2)^2;
         end

         Norm = sqrt(Norm);
         if(Norm > esl)
            %----- Mise à jour des multiplicateurs -----%
            for i = 1:n
                Pi(i) = Pi(i) + lamda*(D(2,i) - 2)/ Norm;
            end
         end
      end

      if ((D(2,:)== Temp)|(k == kmax)|(LB > UB-esl)|(Norm < esl))
         TSP = 1;
      end
      % (D(2,:)= Temp) veut dit que tous les noeuds ont un degré égale à 2
end % end while

end % end of function
```

- Pour les règles R2 et R3, il y a une petite modification dans la fonction Sous_Gradient ( ). La valeur « lamda » est mise à jour dans le boucle « while – end ». Donc, elle n'est plus un paramètre d'entré, pré-défini à l'extérieur de la fonction comme dans le cas de la règle R1.

  - ✓ *Règle R2:*

```
function [h, A, D] = Sous_GradientR2 (n, C, UB)
...
while (TSP == 0)
      k = k + 1;

      lamda = 1/k;
...
end

end % end of function
```

  - ✓ *Règle R3: $\lambda = \sigma.\alpha^k$ avec $\sigma = 50$, $\alpha = 0.9$*

```
function [h, A, D] = Sous_GradientR3 (n, C, UB)
...
while (TSP == 0)
      k = k + 1;
```

```
      lamda = 50*0.9^k;

...
end

end % end of function
```

- Nous ajoutons également deux fonctions qui servent à tracer le trajet trouvé en utilisant les méthodes «Farthest_Insertion» et «Sous_Gradient ».

[7]. Fonction «plot_Cycle»

```
% ------------------------------------------------------------------- %
% @Fonction      plot_Cycle
% @Bref          Tracer le cycle trouvé en utilisant la méthode
%                Farthest_Insertion
% @Paramètres d'entrés
%                X, Y  Coordonnées des noeuds
%                T     Itinéraire du cycle (l'ordre de noeuds à traverser
%                      successivement)
% @Programmeur   NGUYEN Trung Duong - Étudiant Master OSS - UTT
% @Date          2013/01/11
% ------------------------------------------------------------------- %

function plot_Cycle(X, Y, T)

for i = 1:size(T,2)
    newX(i) = X(T(i));
    newY(i) = Y(T(i));
end

% Localiser le dépôt et les noeuds
plot(X,Y,'o','MarkerEdgeColor','k',...
             'MarkerFaceColor','g',...
             'MarkerSize',10);
grid
hold on

plot(X(1),Y(1),'o','MarkerEdgeColor','k',...
               'MarkerFaceColor','b',...
               'MarkerSize',12);
hold on

% Tracer le cycle
line (newX, newY,'Color','r','LineWidth',2);

end % end of function
```

[8]. Fonction «plot_Aretes»

```
% ------------------------------------------------------------------------ %
% @Fonction      plot_Aretes
% @Bref          Tracer le trajet trouvé en utilisant la méthode
%                sous_gradient
% @Paramètres d'entrés
%                X, Y  Coordonnées des noeuds
%                A     Liste des arêtes
% @Programmeur   NGUYEN Trung Duong - Étudiant Master OSS - UTT
% @Date          2013/01/11
% ------------------------------------------------------------------------ %

function plot_Aretes(X, Y, A)

% Localiser le dépôt et les noeuds
plot(X,Y,'o','MarkerEdgeColor','k',...
             'MarkerFaceColor','g',...
             'MarkerSize',10);
grid
hold on

plot(X(1),Y(1),'o','MarkerEdgeColor','k',...
               'MarkerFaceColor','b',...
               'MarkerSize',12);
hold on

% Tracer les arêtes
for i = 1:size(A,2)
    for j = 1:2
        newX(1,j) = X(A(j,i));
        newY(1,j) = Y(A(j,i));
    end
        line(newX,newY,'Color','r','LineWidth',2);
end

end % end of function
```

II. Résultats des tests

II.1 Tâche 5

Fichier-test	Optimum	Farthest_ Insertion (UB)	Sous_Gradient					
			R1 (λ = 1)		**R2**		**R3 (σ= 50, α=0.9)**	
			LB	**(*)**	**LB**	**(*)**	**LB**	**(*)**
eil51	426	464	420.6162	-9.35 %	408.3243	-11.9941 %	422.1912	-9.0105 %
eil76	538	592	534.7368	-9.6728 %	507.3219	-14.3037 %	536.9004	-9.3074 %
pr76	108159	119692	93518	-21.8675 %	90145	-24.6858 %	91818	-23.2880 %
st70	675	719	668.9329	-6.9634 %	603.63632	-16.0455 %	671	-6.6759 %
bier127	118282	129723	99989	-22.9213 %	95389	-26.4672%	98012	-24.4448 %

*(*) Écart entre la valeur LB («h» obtenue de la fonction Sous_Gradient( )) et la valeur UB, par exemple: - 9.6728 % signifie que LB = (1- 9.6728/100)*UB.*

- Le programme ci-dessous réalise les travaux dans la tâche 5 (pour le cas du règle R1 typiquement)

```
% ------------------------------------------------------------------------ %
% @Fonction    Main1
% @Bref        [1]. Déterminer la valeur de UB et de LB(h)
%              [2]. Déterminer l'écart entre les deux en %
%
% @Programmeur NGUYEN Trung Duong - Student Master OSS - UTT
% @Date        2013/01/11
% ------------------------------------------------------------------------ %

filename = fopen('C:\Users\DUONG\Desktop\OS01 projet\TSPLIB
simplified\eil51.txt');

[n, X, Y] = Read_TSP(filename);
C = Compute_C(n, X, Y);
[UB, T] = Farthest_Insertion(n, C);

lamda = 1;                  % valeur du pas utilisée dans l'algorithme
                            % sous_gradient (règle R1)
[h, A, D] = Sous_Gradient(n, C, UB, lamda);

ecart = (1 - h/UB)*100;     % écart entre LB(h) et UB
```

II.2 Tâche 6

Règle	Écart moyen entre LB et UB	Nombre de problèmes où le 1-arbre est une solution optimale du TSP
R1 (λ = 1)	-4.4640 %	13
R2	-9.8384 %	1
R3 (σ= 50, α=0.9)	-3.6222 %	42

- Pour les travaux dans la tâche 6, dans la fonction «Compute_C( )» nous devons éliminer l'instruction qui sert à arrondir la distance

```
    C(i,j)= round(C(i,j));
```

- Le programme ci-dessous réalise les travaux dans la tâche 6 (pour le cas du règle R1 typiquement)

```
% ------------------------------------------------------------------- %
% @Fonction    Main2
% @Bref        [1]. Déterminer l'écart moyen entre LB et UB en %
%              [2]. Déterminer le nombre de problèmes où le 1-arbre est
%              une solution optimale du TSP
%
% @Programmeur NGUYEN Trung Duong - Student Master OSS - UTT
% @Date        2013/01/11
% ------------------------------------------------------------------- %

n = 30;                     % le problème TSP contient 30 noeuds
Xmax = 50; Ymax = 50;       % les noeuds répartis dans un rectangle (50x50)

k = 0;                      % compteur du nombre mentionné dans [2]

lamda = 1;                  % valeur du pas utilisée dans l'algorithme
                            % sous_gradient (règle R1)

for i = 1:100               % 100 problèmes à évaluer
    [X,Y] = Random_TSP(n, Xmax, Ymax);
    C = Compute_C(n, X, Y);
    [UB, T] = Farthest_Insertion(n, C);
    [h, A, D] = Sous_Gradient(n, C, UB, lamda);

    ecart(i) = (1 - h/UB)*100;      % écart entre LB(h) et UB

    %----- Examiner si tous les noeuds ont un degré égale à 2 ----%
    if sum(D(2,:)== 2) == n
       k = k + 1;                   % si Qui, la fonction Sous_Gradient
                                    % donne une solution optimale du TSP
    end
end % end for

moyen = sum(ecart)/100;             % écart moyen entre LB(h) et UB pour
                                    % les 100 problèmes
```