INF226

Problème du voyager de commerce

Travaux Pratique 1 - Première semestre de 2012

Professeur: Olivier Hudry

Tiago CHEDRAUOI SILVA Casier: 214

Mars 22, 2012

Matériel

Pour le travaux pratique on a utilise un ordinateur un peut lent :
- Intel(R) Core(TM)2 Duo CPU T5250 @ 1.50GHz
- cache size : 2048 KB
- Ram : 3Gb

Changements

On a inclue la bibliothèque “math.h” et on a changé le Makefile pour inclure le flag -lm necessaire pour compiler cette bibliothèque.

Partie 1

TAB. 1: Nombre de noeuds visités sans sans utiliser la relaxation lagrangienne.

Graphe	Nombre de noeuds	Temps	Poids
g5	8	0.000000 secondes	30
g10	511	0.000000 secondes	35
g15	9980	0.028995 secondes	20
g20	35137	0.158975 secondes	52
g22	23438	0.121981 secondes	120
g23	528589	2.809572 secondes	269
g25	3292515	20.954814 secondes	237
g30	Temps non raisonnables	-	-
g50	Temps non raisonnables	-	-
g100	Temps non raisonnables	-	-

Partie 2

Le code pour ces calculs est au dessous.

Relaxation

```
1  int relaxation(Arete ** liste, Arete **
       liste0, int nb_liste, int
      nb_liste0)
2  {
3    int sommet;
4    int num_iter = 1, i, k;
5    double valeur;
6    double alpha = alpha_initial;
7    double T;
8    int P, D;
9    int tour;
10   int norme=0;
11   double s;
12
```

```
13  for (i = 0; i < nb_liste; i++)
        liste_relax[i] = liste[i];
14  for (i = 0; i < nb_liste0; i++)
        liste0_relax[i] = liste0[i];
15
16   // debut de la  recherche du maximum
         de la fonstion duale
17  for (num_iter = 1; num_iter <=
        nb_iter; num_iter++)
18    {
19      /* calcul de l'arbre qui minimise
            la fonction de Lagrange
20   pour les valeurs actuelles des
        lambdas ;
21   la liste des aretes de cet arbre
22   est dans un_arbre_relax (variable
        globale). */
23      meilleur_un_arbre_relax(nb_liste,
            nb_liste0);
24
25      // valeur de la fonction duale
            pour les valeurs actuelles
            des lambdas
26      valeur = valeur_fonction_duale();
27
28      /* Regarder si on peut couper en
            comparant la valeur de la
29   fonction duale a la borne ; si oui,
        sortir ;
30   on raffinera la comparaison en
        utilisant le fait que
31   le probleme primal admet une
        solution entiere, alors que
32   les valeurs de la fonction duale
        sont reelles (si la borne vaut
        40
33   et que valeur vaut 39.001, on ne
        peut pas faire moins que 40
34   et on peut couper). */
35
36      // A COMPLETER
37      //TCS
38      if(ceil(valeur)>=borne)
39  return 1;
40
41      /* Regarder si un_arbre_relax est
            un tour en utilisant
42   le tableau global degres_arbre_relax
43   qui contient les degres des sommets
        de un_arbre_relax. */
44
45      // A COMPLETER
46      // mes degrees doivent etre egal
            a 2
47      tour=1;
48      for(i=0;i<n;i++){
49  if(degres_arbre_relax[i]!=2)
50    tour = 0;
51      }
52      /* S'il s'agit d'un tour :
53   calculer le poids de ce tour ;
54   l'appel de poids(un_arbre_relax)
        renvoie ce poids.
55   S'il le faut, mettre ce poids dans
        borne et sauvegarder
56   le tour correspondant par l'
        instruction
57   sauvegarde_solution(un_arbre_relax).
58   Quitter la fonction relaxation en
        renvoyant la valeur appropriee.
59   utiliser eventuellement :
60        printf("borne = %d\n", borne);
61   pour voir ou en est la borne.
62      */
63      // A COMPLETER
64      if(tour){
65  int varpoids = poids(un_arbre_relax);
66  if(varpoids<borne){
67    borne = varpoids; /*provavelemnte
          errado =P*/
68    sauvegarde_solution(un_arbre_relax)
          ;
69    printf("borne = %d\n", borne);
70    return 1;
71  }
72      }
73
74
75      if (num_iter == nb_iter) break;
76      /* Si les remarques precedentes n
            'ont pas permis de conclure,
77   calculer de nouvelles valeurs des
        lambdas (voir le cours).
78    */
79      for(i=0;i<n;i++){
80  norme = norme + pow(
        degres_arbre_relax[i]-2,2);
81      }
82      // comentar se gama = 2
83      //norme = sqrt(norme);
84
85      s = alpha*(borne-valeur)/(norme);
86      for(i=0;i<n;i++){
87  lambda[i]=lambda[i]+s*(
        degres_arbre_relax[i]-2);
88      }
89      // A COMPLETER
90
91      /* Ne pas trop se preoccuper de
            cette boucle qui sert a
            eviter  une divergence
92   de la recherche de l'optimum,
93      . */
94      for (i = 0; i < n; i++)
95  {
```

```
96      if ((lambda[i] > borne) || (lambda[
            i] < -borne)) lambda[i] = 0;
97    }
98
99        /* Faire decroitre la valeur de
              alpha (on peut eventuellement
               laisser
100     toujours ce parametre a la valeur
            alpha_initial) */
101     // A COMPLETER EVENTUELLEMENT
102
103
104       }
105     return 0;
106   }
```

Partie 3

En utilisant le programme de relaxation, on voit que le temps nécessaire pour trouver la bonne réponse a diminué de façon significative. Par exemple, le graphe g30,g50 et g100 ont donné une réponse en quelques secondes, cela n'était pas possible avant.

Graphe	Nombre de noeuds	Temps	Poids
g5	4	0.000000 secondes	30
g10	36	0.000999 secondes	35
g15	96	0.002999 secondes	20
g20	71	0.004999 secondes	52
g22	59	0.003999 secondes	120
g23	301	0.015997 secondes	269
g25	234	0.015997 secondes	237
g30	514	0.048992 secondes	212
g50	4583	1.046840 secondes	214
g100	54262	46.290962 secondes	283

TAB. 2: Nombre de noeuds visités avec la relaxation lagrangienne (alpha = 1.5 nombre interactions = 10 borne = 5000 gama=2).

Après voir que la relaxation a amélioré la vitesse, on doit chercher pour les meilleurs paramètres pour notre relaxation. Ainsi, on a changé l'alpha (pas), le nombre itérations et la valeur de la borne.

Alpha	Temps
0.1	5.158214 secondes
0.3	3.011542 secondes
0.5	1.501771 secondes
0.75	1.334797 secondes
1	1.197817 secondes
1.3	1.146825 secondes
1.4	1.134827 secondes
1.5	1.038842 secondes
1.6	1.123829 secondes
1.8	1.087834 secondes
2	1.334797 secondes
2.5	1.309800 secondes
3	1.485774 secondes

TAB. 3: Comparaison alpha : Graphe g50 :Nombre de noeuds visités avec la relaxation lagrangienne (nombre interactions = 10 borne = 5000 gama=2).

Pour l'alpha, on a trouve : $Alpha = 1.5$.

Nb iteration	Temps
2	3.753429 secondes
3	2.238659 secondes
4	1.715739 secondes
5	1.467776 secondes
10	1.032842 secondes
12	1.087834 secondes
15	0.981850 secondes
18	1.432782 secondes
20	1.126828 secondes
25	1.495772 secondes
50	1.861716 secondes
100	1.781729 secondes

TAB. 4: Comparaison nb iteration : Graphe g50 :Nombre de noeuds visités avec la relaxation lagrangienne (nombre interactions = 10 borne = 5000 gama=2).

Pour le nombre d'interaction, on a trouve : $interation = 15$.

Borne	Temps
300	0.820875 secondes
500	0.720891 secondes
800	1.254809 secondes
1000	1.071837 secondes
2000	1.188819 secondes
3000	1.153824 secondes
5000	0.990849 secondes
10000	0.953854 secondes
20000	1.417784 secondes
100000	1.083835 secondes

TAB. 5: Comparaison des bornes : Graphe g50 :Nombre de noeuds visités avec la relaxation lagrangienne (alpha = 1.5 nombre interactions = 15 gama=2).

Pour la valeur de la borne, on a trouve : $borne = 500$.
En choisissant les meilleurs paramètres on a ajoute le code suivante, pour changer γ :
– norme = pow(norme,gama) ;
En choisissant un γ égal a 1 est très lent.

γ	Temps
1	Temps non raisonnables
1.5	Temps non raisonnables
2	0.707892 secondes
2.5	2.824570 secondes
3	0.706892 secondes
4	10.137458 secondes

TAB. 6: Comparaison des γ - Graphe g50 :alpha = 1.5 nombre interactions = 15 borne=500.