TP - Compte-rendu
Méthode par séparation et évaluation : problème du voyageur de commerce

INF392 – Bertrand Chazot 24/03/12

Premier temps : Sans la relaxation

Nombre de nœuds traités pour les graphes dont une solution est trouvée en un temps raisonnable.

N° du graphe	5	10	15	20	22	23	25
Nb nœuds traités	8	511	9980	35137	22438	528589	3292515

Deuxième temps : Fonction de relaxation

Voici mon code pour la fonction de relaxation. Je me suis permis de passer en C99 pour améliorer la clarté ; le code est aussi présent avec le Makefile idoine dans l'archive jointe.

****

```
int relaxation(Arete ** liste, Arete ** liste0, int nb_liste, int nb_liste0) {

  double alpha = alpha_initial;

  for (int i = 0; i < nb_liste; i++) liste_relax[i] = liste[i];
  for (int i = 0; i < nb_liste0; i++) liste0_relax[i] = liste0[i];

  // debut de la  recherche du maximum de la fonction duale
  for (int num_iter = 1; num_iter <= nb_iter; num_iter++) {
        /* calcul de l'arbre qui minimise la fonction de Lagrange pour les
valeurs actuelles des lambdas ; la liste des aretes de cet arbre
           est dans un_arbre_relax (variable globale). */
        meilleur_un_arbre_relax(nb_liste, nb_liste0);

        // valeur de la fonction duale pour les valeurs actuelles des lambdas
        double valeur = valeur_fonction_duale();

        /* Regarder si on peut couper en comparant la valeur de la fonction
duale a la borne ; si oui, sortir ; on raffinera la comparaison en          utilisant le
fait que le probleme primal admet une solution entiere,          alors que les valeurs de
la fonction duale sont reelles.*/

        if(ceil(valeur) >= borne)
            return 1;

        /* Regarder si un_arbre_relax est un tour en utilisant le tableau global
degres_arbre_relax qui contient les degres des sommets de
           un_arbre_relax. */

        // est_ch == n ssi chemin hamiltonien ssi degres == 2 à tout les sommets
        int est_ch;
        for(est_ch = 0 ;
            (est_ch < n) && (degres_arbre_relax[est_ch] == 2) ;
            est_ch++);

        /* S'il s'agit d'un tour :
             calculer le poids de ce tour ;
             l'appel de poids(un_arbre_relax) renvoie ce poids.
             S'il le faut, mettre ce poids dans borne et sauvegarder
             le tour correspondant par l'instruction
             sauvegarde_solution(un_arbre_relax).
             Quitter la fonction relaxation en renvoyant la valeur appropriee.
             utiliser eventuellement :
             printf("borne = %d\n", borne);
             pour voir ou en est la borne.
```

```
        */
        if(est_ch == n) {
            int poids_1a = poids(un_arbre_relax);
            if(poids_1a < borne) {
              borne = poids_1a;
              sauvegarde_solution(un_arbre_relax);
              printf("borne = %d\n", borne);
            }
            return 0;
        }

        if (num_iter == nb_iter) break;
        /* Si les remarques precedentes n'ont pas permis de conclure,
             calculer de nouvelles valeurs des lambdas (voir le cours).
        */

        double norme_d2 = 0;
        for(int i = 0 ; i < n ; i++) {
            norme_d2+=(degres_arbre_relax[i]-2)*(degres_arbre_relax[i]-2);
        }

        double s = alpha * (borne - valeur)/norme_d2;
        for (int i = 0 ; i < n ; i++) {
            lambda[i] += s*(degres_arbre_relax[i]-2);
        }

        /* Ne pas trop se preoccuper de cette boucle qui sert a eviter  une divergence
             de la recherche de l'optimum,
             . */
        for (int i = 0; i < n; i++) {
            if ((lambda[i] > borne) || (lambda[i] < -borne)) lambda[i] = 0;
        }

        /* Faire decroitre la valeur de alpha (on peut eventuellement laisser
             toujours ce parametre a la valeur alpha_initial) */

        alpha *= 0.8;

    }
    return 0;
}
```

****

Troisième temps : Réglage des paramètres

Résultats obtenus par le script test.sh

Tests sur le graph g50 ; Borne initial: 5000
---

*Recherche du meilleur alpha avec 10 itérations
Alphas testés: 0.1 0.2 0.4 0.7 0.8 0.9 1.0 1.2 1.3 1.4 1.5 1.6 1.7 2 2.5 3

Alpha=0.1:
Nombre de noeuds 25397
duree = 2.508156 secondes

Alpha=0.2:
Nombre de noeuds 10158
duree = 1.148071 secondes

Alpha=0.4:
Nombre de noeuds 7806

duree = 0.944059 secondes

Alpha=0.7:
Nombre de noeuds 5248
duree = 0.644040 secondes

Alpha=0.8:
Nombre de noeuds 5113
duree = 0.628039 secondes

Alpha=0.9:
Nombre de noeuds 5210
duree = 0.688043 secondes

Alpha=1.0:
Nombre de noeuds 4088
duree = 0.544034 secondes

Alpha=1.2:
Nombre de noeuds 4609
duree = 0.628039 secondes

Alpha=1.3:
Nombre de noeuds 4888
duree = 0.660041 secondes

Alpha=1.4:
Nombre de noeuds 4068
duree = 0.592037 secondes

Alpha=1.5:
Nombre de noeuds 4381
duree = 0.620038 secondes

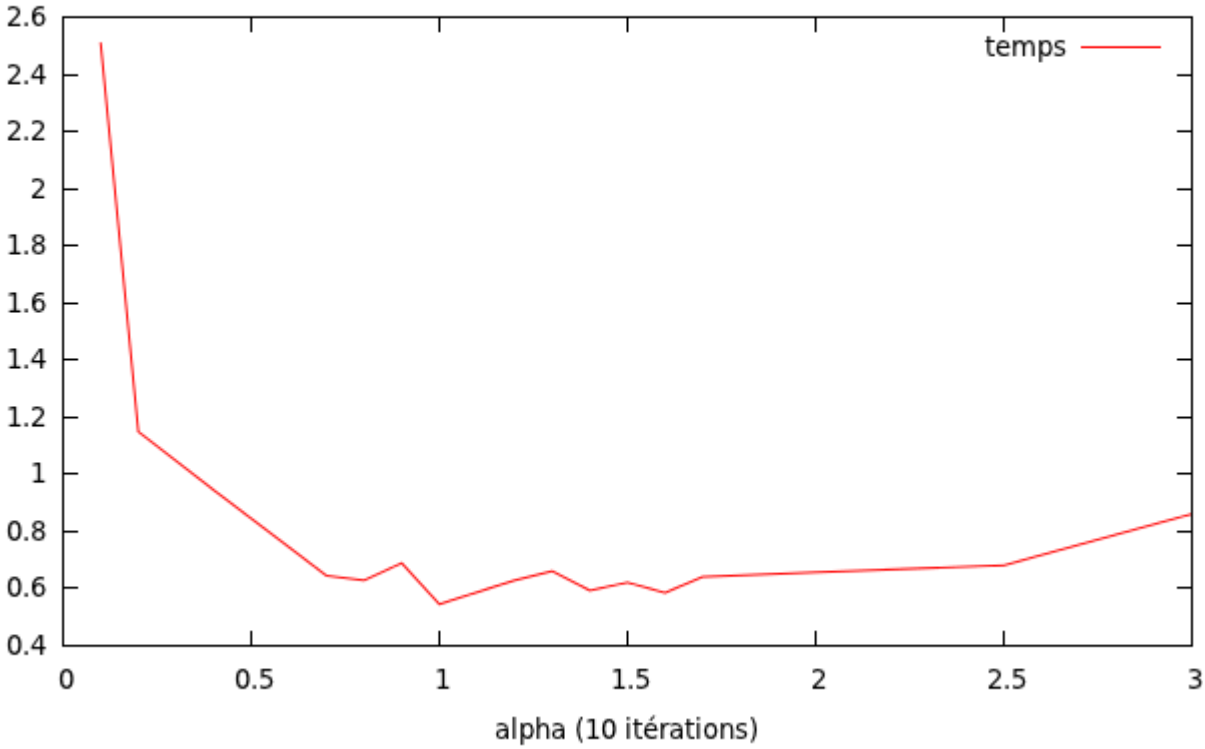
Alpha=1.6:
Nombre de noeuds 4145
duree = 0.584036 secondes

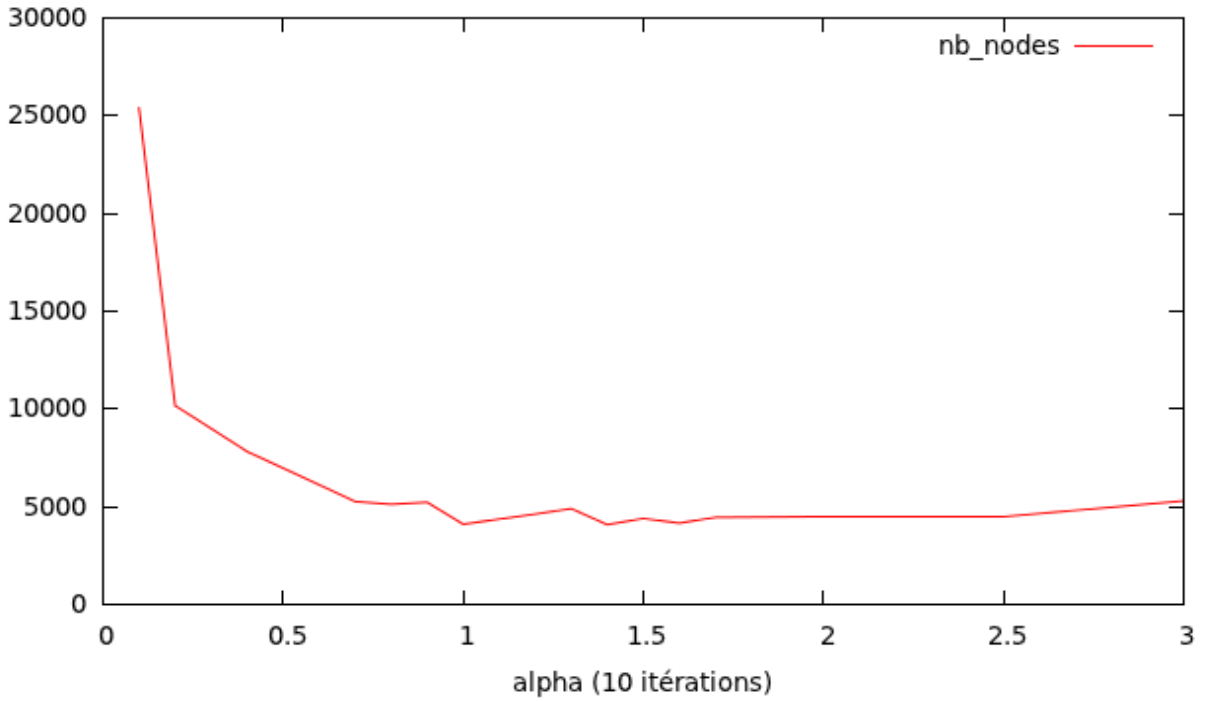
Alpha=1.7:
Nombre de noeuds 4438
duree = 0.640040 secondes

Alpha=2:
Nombre de noeuds 4480
duree = 0.656041 secondes

Alpha=2.5:
Nombre de noeuds 4474
duree = 0.680042 secondes

Alpha=3:
Nombre de noeuds 5285
duree = 0.860053 secondes





Résultats: meilleur temps (0.544034) pour alpha=1.0

*Recherche du meilleur nombre d'itérations avec alpha=1.0
Valeurs testées: 5 8 9 10 11 12 15 20 30 50 100

5 itératons
Nombre de noeuds 8416
duree = 0.804050 secondes

8 itératons
Nombre de noeuds 6046
duree = 0.744046 secondes

9 itératons
Nombre de noeuds 5354

duree = 0.692043 secondes

10 itératons
Nombre de noeuds 4088
duree = 0.556034 secondes

11 itératons
Nombre de noeuds 4677
duree = 0.644040 secondes

12 itératons
Nombre de noeuds 4505
duree = 0.636039 secondes

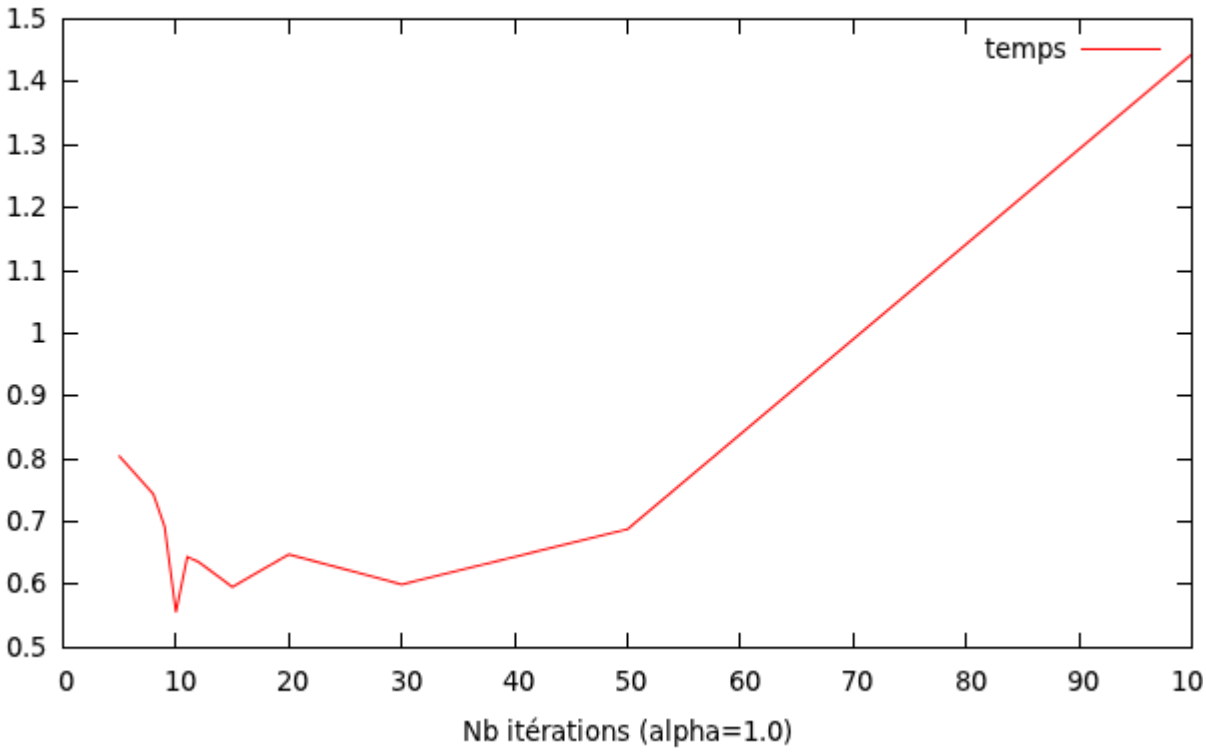
15 itératons
Nombre de noeuds 3519
duree = 0.596037 secondes

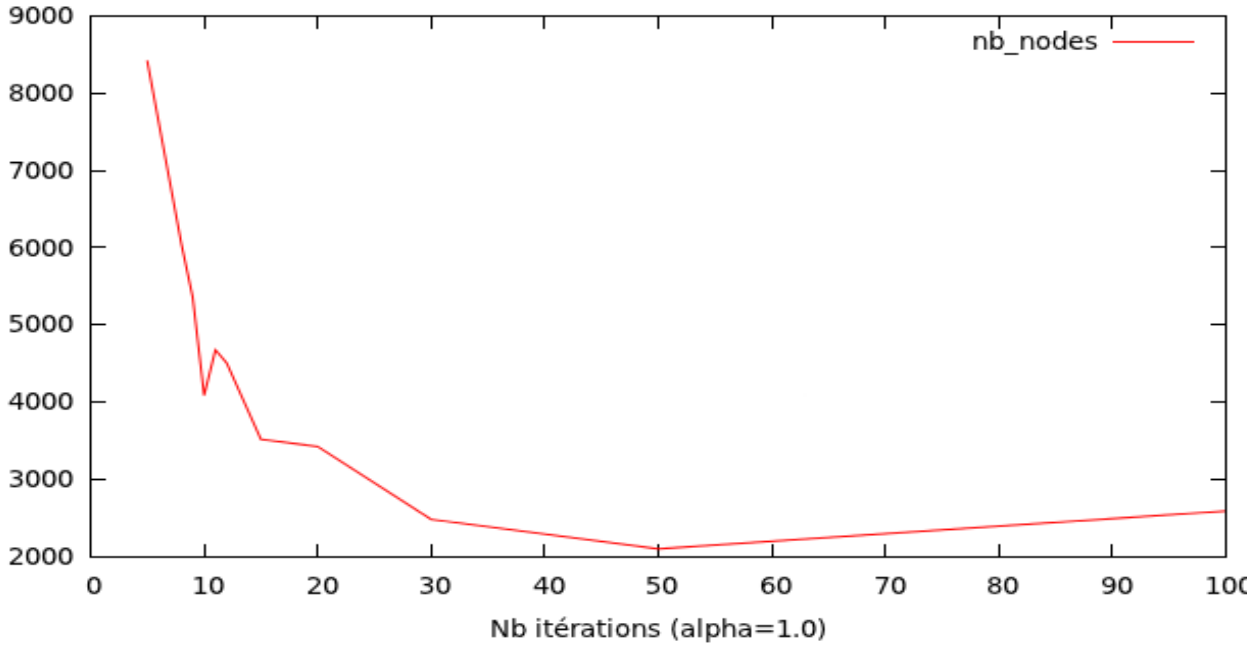
20 itératons
Nombre de noeuds 3426
duree = 0.648040 secondes

30 itératons
Nombre de noeuds 2482
duree = 0.600037 secondes

50 itératons
Nombre de noeuds 2100
duree = 0.688043 secondes

100 itératons
Nombre de noeuds 2590
duree = 1.444090 secondes





Résultats: meilleur temps (0.544034) pour alpha=1.0 et 10 itérations

Avec g100 :
Résultats: meilleur temps (13.592849) pour alpha=0.8 et 10 itérations

Conclusion :

La relaxation lagrangienne apporte un gain conséquent. On trouve un optimal pour g100 en moins de 15s alors qu'il fallait (beaucoup?) plus de 5 minutes sans.

Lorsqu'on augmente le nombre d'itérations sur la relaxation lagrangienne, le nombre de nœuds décroit mais le temps d'exécution augmente si ce nombre est trop grand car on passe trop de temps à chaque relaxation.

La valeur optimale de alpha semble bien augmenter avec la taille des graphes.

Pour gamma la meilleur valeur semble 2. Avec 1 g50 et g100 ne *semble* pas finir, avec 3 ils prennent plus de temps. De plus 2 permet d'éviter une racine pour le calcule de la norme (gain de 17% en temps, entre norme^gamma avec gamma=2 et juste le calcul du carré de la norme).