

Methodes Formelles

TP Recuit Simulé

Valentin Guerlesquin

12 décembre 2006

1 Fonction

Cette partie du TP permet la mise en oeuvre de l'algorithme de recuit simulé, tel qu'il est décrit dans le sujet du TP. A partir de l'implémentation incomplète du problème, il est assez simple de finaliser celle ci à l'aide du sujet de TP.

L'étude de l'influence des différents paramètres nous amènent à confirmer la pertinence des chiffres donnés dans le cours :

- Un facteur de modification de la température proche de 90 %
- Un nombre d'essai par palier de l'ordre de 2500 (en prenant 50 paliers)

Pour obtenir un nombre de palier à 50, on prends une température finale à 1 et la température finale à 50. L'algorithme fonctionne correctement ainsi, donc on conserve ces valeurs.

2 Voyageur

On reprends le même algorithme que précédement pour résoudre le problème d'optimisation du voyageur de commerce. L'algorithme reste donc le même, seuls changent la fonction de transformation et la fonction de calcul de coût.

2.1 Situation initiale

On commence avec la fonction de transformation suivante, qui réalise `amplitude` permutations dans parcours.

```
void transformationParcours(int parcoursY[NBVILLES],int parcoursX[NBVILLES],int amplitude)
{
    int ville_1, ville_2, ville_temp, nb_iter, nbperm;
    /* On nous donne un parcoursX, on le copie dans parcoursY, et on le transforme */

    /* copie ... */
    copieParcours(parcoursY, parcoursX);
    for(nb_iter = 0; nb_iter < amplitude; nb_iter++)
    {
        ville_1 = myRandomMinMax(0, NBVILLES-1);
        /* assurons nous que l'on permutte pas la meme chose */
        for(ville_2 = ville_1; ville_2 == ville_1; ville_2 = myRandomMinMax(0, NBVILLES-1)) {;}
        /* permutation */
        ville_temp = parcoursY[ville_1];
        parcoursY[ville_1] = parcoursY[ville_2];
        parcoursY[ville_2] = ville_temp;
    }
}
```

Un autre élément qu'il serait interessant de faire varier est la décroissance de la température. On choisit pour le moment une décroissance linéaire.

```

/*_____ Modification température _____*/
double g(void){
    return( T*alpha ); /* on décroît le température en utilisant alpha */
}

```

2.2 Résultats obtenus

On testera nos paramètres et les fonctions avec un nombre constant de villes. Les paramètres fournis au lancement du programme sont :

TInit 50, pplus ou moins arbitrairement

TFin 1, histoire que le nombre de pallier (Tinit/Tfin) soit égale à 50, conformément aux valeurs optimales données dans le cours.

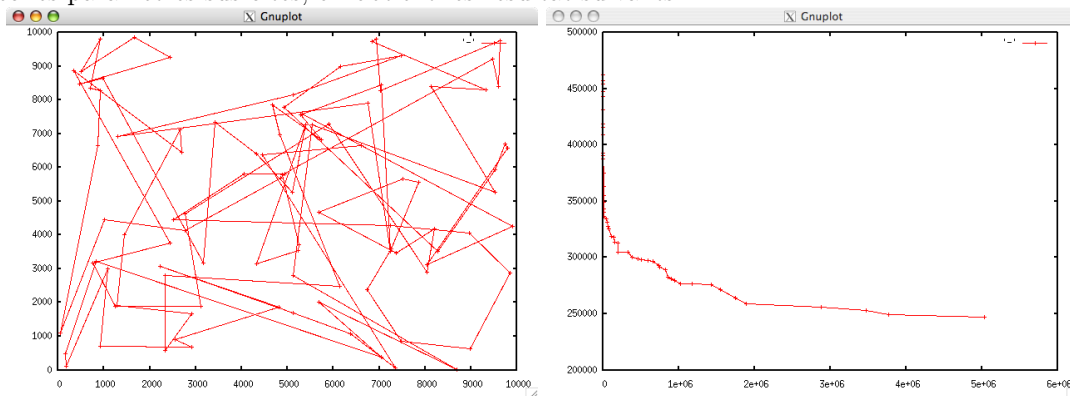
Alpha 0,93 car le cours nous indique cela, alors, pour le moment, suivons le cours !

Amplitude On choisit 10% du nombre de villes, plus ou moins arbitrairement. On se dit que si ce chiffre est trop bas, il faudra beaucoup trop de temps à l'algorithme pour optimiser la solution, et que si il est trop haut, alors l'algorithme risque de s'éloigner de la solution courante de façon trop importante.

NMaxEcc Le cours nous dit qu'il est bon que cette valeur prenne $(\frac{N}{2})^2$, où N est le nombre d'éléments (le nombre de villes). Alors on fixe cette valeur à 2500.

NMaxRep Fixé à dix millions. Pourquoi pas ... de toute façon, n'est pas utilisé. Ce paramètre permet à l'utilisateur d'exprimer toute sa fantaisie !

Avec les paramètres sus-cités, on obtient les résultat suivants¹ :



Le parcours final n'est manifestement pas une approximation satisfaisante de la solution optimale. La fonction de coût fait apparaitre qu'aucune solution n'a été trouvée entre la 5 millionième et la 10 millionième itération.

2.3 Le problème de l'amplitude

Tel qu'elle est écrite, la fonction de transformation effectue sur la solution un nombre constant de permutations. Cette approche peut paraître choquante car, à partir d'une certaine température, modifier le parcours de 10% risque fort d'ammener à une détérioration de la solution. Or, plus la température diminue, moins la régression est autorisée.

On propose donc la chose suivante : le nombre de transformations à effectuer sera un nombre tiré aléatoirement entre 1 et l'amplitude. Le code de cette nouvelle solution est assez simple :

```

void transformationParcours(int parcoursY[NBVILLES],int parcoursX[NBVILLES],int amplitude)
{
    int ville_1, ville_2, ville_temp, nb_iter, nbperm;
    /* On nous donne un parcoursX, on le copie dans parcoursY, et on le transforme */

    /* copie ... */

```

¹Les résultats présentés sont conforme à la moyenne de l'ensembles des essais effectués avec les même paramètres. Avec une répartition de villes aléatoire, et un chemin initiale également aléatoire, il est bon de faire plusieurs essai pour ne pas tirer de conclusions trop hâtives.

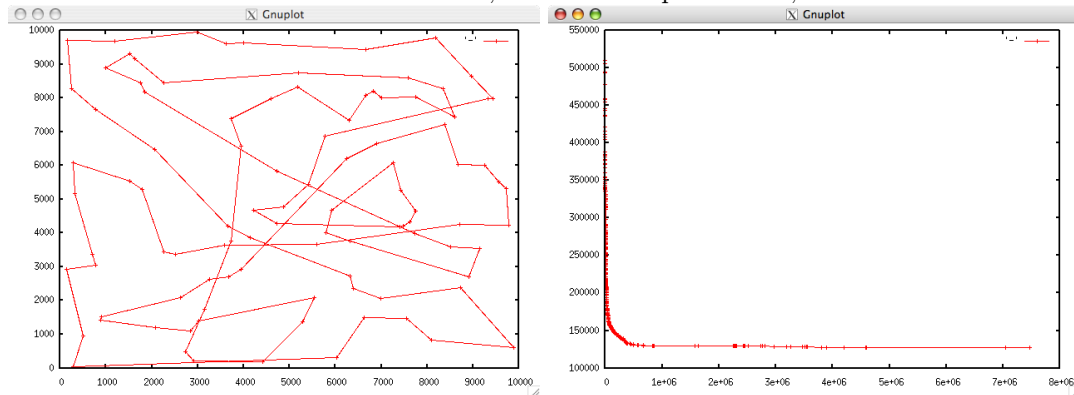
```

copieParcours(parcoursY, parcoursX);
initRandom();
nbperm = myRandomMinMax(1, amplitude);
for(nb_iter = 0; nb_iter < nbperm; nb_iter++)
{
    ville_1 = myRandomMinMax(0, NBVILLES-1);
    /* assurons nous que l'on permutte pas la meme chose */
    for(ville_2 = ville_1; ville_2 == ville_1; ville_2 = myRandomMinMax(0, NBVILLES-1)) {;}
    /* permutation */
    ville_temp = parcoursY[ville_1];
    parcoursY[ville_1] = parcoursY[ville_2];
    parcoursY[ville_2] = ville_temp;
}
}

```

2.4 Résultats obtenus

Avec une telle fonction de transformation, et les mêmes paramètres, on obtient des résultats de ce type :



Le résultat semble bien plus proche d'une solution optimale que le précédent, et on remarque que la fonction de coût décroît bien plus rapidement. Elle atteint assez rapidement un seuil bas que l'on devine être le coût optimal.

2.5 D'autres possibilités pour la fonction de transformation

Les tests effectués en rendant l'amplitude de modification directement dépendante de la température courante ont été particulièrement décevants. Cette méthode ne permet pas d'obtenir une approximation de la solution optimale.

Par manque de temps, il n'a pas été possible de tester d'autres méthodes de transformation. Cependant, en interrogeant nos voisins, ceux-ci ont testé une méthode d'inversion de segments. Deux bornes sont choisies aléatoirement, et l'ordre de parcours à l'intérieur de ce segment est inversé.

D'après leurs dires, cette méthode serait efficace, mais bougrement coûteuse en temps d'exécution. On comprend aisément cette seconde observation.