INF4705: Analyse et conception d'algorithmes

Auteur: Jonathan Pelletier (1245014)
Date: 01/12/2011

Introduction

Dans le cadre de ce travail de laboratoire, les étudiants ont été invité à résoudre un problème d'optimisation combinatoire concernant la génération d'horaire pour l'orchestre symphonique de Montréal. Les solutions possibles à ce problème sont des suites qui représentrent l'orde dans lequel sont jouées des pièces pendant une répétition. Illustrons avec un exemple: si on a une répétition qui compte 9 pièces, une solution possible serait de les jouer dans cet ordre: 5,6,4,1,0,8,3,7,2. Sans entrer dans plus de détail sur la formulation du problème à résoudre (voir l'énoncé du devoir), il est possible de voir que la recherche de la solution optimale par énumération exhaustive est impossible. En effet, l'espace des états à explorer a une taille de n! ou n est le nombre de pièce à jouer.

Afin de résoudre ce problème en temps raisonnable, j'ai choisi d'utiliser un algorithme métaheuristique. Celui-ci s'inspire d'un réseau crystallin qui adopte la configuration qui minimise l'énergie du système losque celui-ci refroidit. L'algorithme que j'ai choisi est le recuit simulé.

Dans le reste de ce rapport vous trouverez:

1. Une présentation de l'algorithme et une analyse de complexité.
2. Une discussion sur les raisons du choix de cet algorithme.

Pour toutes questions relatives à l'utilisation du programme, veuillez vous référer au fichier README.rst qui fait office de manuel d'instruction.

Bonne correction !

Présentation de l'algorithme

Dans cette section, on présente brièvement les concepts de l'algorithme de recuit simulé. L'algorithme en pseudo-code est montré et on donne une analyse de la complexité algorithmique de son implémentation.

Description des concepts

L'algorithme de recuit simulé utilise les concepts suivants:

1. Énergie du système.
2. Température.

3. Probabilité de transition.

4. Voisinage d'un état.

5. Protocole de refroidissement.

Lorsqu'on démarre l'algorithme, on inisialise la température du système à une valeur très élevée. On choisi au hasard un état de départ et l'algorithme commence son exploration d'une suite d'états à partir de l'état courant. À chaque itération, on obtient un nouvel état de l'état courant en explorant aléatoirement son voisinage. Pour la résolution du problème, un voisin d'un état est obtenu en permutant l'ordre de deux pièces dans la solution. Une fois le voisin obtenu, on calcul l'énergie de ce nouvel état. L'énergie du système est la fonction que l'on doit minimiser. Dans le cadre de notre problème, il s'agit du coût du temps d'attente des solistes. En fonction de la valeur de la différence d'énergie entre l'état actuel et l'état voisin, on choisi d'effectuer une transition d'état selon une certaine distribution de probabilité que l'on notera $P(e,e',T)$ ou e et e' représentent l'énergie de l'état courant et de l'état voisin respectivement et T représente la température. Le rôle de la température est de permettre des transitions qui ne vont pas nécessairement améliorer le bilan énergétique du système lorsque celle-ci est haute. Au fur et à mesure ou la température diminue, la probabilité d'une transition vers un état qui ne minimise pas l'énergie devient de plus en plus faible. Le protocole de refroidissement représente la manière donc le système passe de sa température initiale élevé vers sa température finale de 0.

Voici maintenant résumé certains des paramètres que l'on a retenu pour notre implémentation de l'algorithme.

Probabilité de transition:

$$P = e^{\lambda(e-e')/T}$$

Ou λ est un paramètre ajustable, e est l'énergie de l'état courant, e', est l'énergie de l'état voisin et T est la température du système. Noter qu'on assigne une probabilité de transition égale à 1.0 si le nouvel état diminue l'énergie du système. Cette distribution intervient uniquement lorsque l'énergie du système est augmenté par la transition.

Protocole de refroidissement:

$$T = 0.5(T_i - T_f)(1 - \tanh(10.0(t - (t_0 + t_1)/2)/(t_1 - t_0))$$

ou T_i est la température initale du système, T_f est la température finale du système, t est le temps actuel, t_0 est le temps de départ de l'algorithme et t_1 est le temps de fin de l'algorithme.

Pseudo-code

Voici l'algorithme de recuit simulé en pseudo-code.

```
s <- s0; e <- E(s)
smeilleur <- s; emeilleur <- e
t <- t0
tant que t < t1
```

```
snouv <- voisin(s)
enouv <- E(snouv)
si P(e, enouv, T) > random() alors
    s <- snouv; e <-enouv
si enouv < emeilleur alors
    smeilleur <- snouv; emeilleur <- enouv
t <- t + dt
retour smeilleur, emeilleur
```

Analyse de complexité

En ce qui concerne l'analyse de complexité de l'algorithme, les fonctions qui permettent d'obtenir un état aléatoire et un état voisin s'exécutent en temps constant. Pour ce qui est de la fonction qui calcul l'énergie d'un état, elle met un temps en $O(nS)$ pour s'exécuter ou n est le nombre de pièces à jouer et S est le nombre de solistes. Le code des lignes 217-243 de schedule.py parcourent en effet un tableau de dimensions n * S. À chaque évaluation de l'énergie, on a donc à attendre un temps en $O(nS)$. Comme le nombre d'évaluations de l'énergie dépend du budget accordé pour le calcul, on peut affirmer que l'algorithme a une complexité en $O(nS)$

Motivation du choix de l'algorithme

J'ai choisi d'utiliser l'algorithme de recuit simulé pour plusieurs raisons. Voici un résumé des arguments qui militent en faveur de cet algorithme.

Simple: L'algorithme de recuit simulé est simple à implémenter et ne fait aucune supposition sur le problème à résoudre.

Bonne complexité algorithmique: L'efficacité de l'algorithme est vastement supérieure à celle d'une recherche exhaustive si une "bonne solution" est suffisante.

Idée originale: Il est très intéressant de voir un algorithme qui s'inspire du monde physique pour résoudre un problème d'optimisation. C'est donc en grande partie la curiosité face à l'algorithme qui a motivé mon choix.

Conclusion

Dans le cadre de ce travail pratique, j'ai implémenté l'algorithme de recuit simulé afin de résoudre le problème de génération de l'horaire pour l'orchestre symphonique de Montréal. L'algorithme est simple à implémenter et donne de meilleurs résultats qu'une simple recherche aléatoire.

Dans le futur, il pourrait être intéressant de comparer la performance de cet algorithme à celle d'autres algorithmes métaheuristiques comme ceux basés sur les colonies de fourmies, ou encore la recherche tabou.