Graphes et Recherche opérationnelle

Rapport final

Chef de projet :
Martin Carton
Responsable qualité :
Maxime Arthaud

Maxence Ahlouche
Korantin Auguste
Thomas Forgione
Thomas Wagner

20 décembre 2013

Table des matières

1 Introduction

Rappel : L'objectif de ce rapport final (CRF6) est de ***corriger les erreurs*** commises dans les rapports (CRFs) précédents (ndlr : il y en a !) et de faire la preuve que les travaux de l'équipe l'ont conduite à ***acquérir des connaissances théoriques*** dans les divers domaines abordés.

2 UA : Graphes

2.1 Modélisation mathématique

Nous avons choisi de représenter nos graphes comme une liste de sommets, chacun ayant une liste d'arêtes.

En mémoire, cette structure est donc constituée d'une liste de pointeurs vers des sommets. Les sommets contenant une liste de pointeurs vers des arêtes. Chaque arête ayant un pointeur vers chaque sommet extrêmité.

Dans la suite nous noterons n le nombre de sommets du graphe.

2.2 Problème du voyageur de commerce

2.2.1 Analyse mathématique

Le problème du voyageur de commerce consiste à chercher un chemin passant par tous les sommets du graphe, de longueur minimale. Ce problème peut s'illustrer par l'exemple d'une fraiseuse qui doit percer des trous dans une plaque le plus rapidement possible, ou encore par un car de touristes qui souhaiterait trouver l'itinéraire le plus rapide pour visiter un certain nombre de lieux.

On peut modéliser ce problème par un graphe complet, dont les arêtes ont un coup qui correspond à la distance entre chaque point, on cherche alors le cycle hamiltonien de coût minimal. On sait qu'un tel cycle existe car le graphe est complet.

Cependant trouver un tel cycle est un problème NP-difficile, et il n'existe donc pas d'algorithme efficace pour trouver ce cycle. En effet, la seule méthode exacte consisterait à tester toutes les chaines hamiltoniennes, et à prendre celle la plus courte, mais le nombre de chaines hamiltonniennes croît exponentiellement du nombre de sommets dans le graphe.

Nous allons donc nous concentrer sur les méthodes approchées de résolution, qui peuvent donner de très bons résultats tout en étant rapides. Toutefois, le résultat n'est donc pas forcément le plus court.

2.2.2 Heuristiques

Les heuristiques vont nous permettre de construire un chemin le plus court possible, de manière rapide, avec le moins de calcul possible. Étant donné qu'on est confronté à énormément de possibilitées pendant la recherche, ils vont permettre d'orienter cette dernière, en faisant des choix les plus judicieux possibles sur les possibilitées a explorer.

Exemple Un heuristique simple consiste à partir d'un sommet au hasard du graphe et d'aller au sommet le plus proche sur lequel on n'est pas encore passé (puis à retourner au sommet de départ pour boucler le cycle). Cet algorithme est en $O(n)$ et donc rapide. Mais il n'offre cependant aucune garantie de résultat, il existe même des graphes pour lesquels il donne le pire cycle.

Plus généralement, chercher parmi les p sommets les plus proches s'avère être une solution relativement efficace, avec une complexité en $O(p^n)$ (donc toujours exponentielle si $p \neq 1$).

Une méthode purement basée sur cet heuristique donnerait :

é

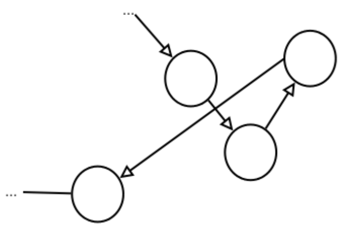
```
Entre : g (Graphe complet)
Sortie : (ûcot , cycle) ùo cycle est un cycle hamiltonien construit selon la én
cot = 0
cycle = ["un point de g au hasard"]

tant qu'il reste des points:
    # On ajoute au cycle le point suivant
    plus_proche = "point de g sur lequel on est pas encore épass le plus proch

    cot += "ûcot de plus_proche au dernier point du cycle"
    cycle = cycle :: plus_proche

# On ferme le cycleû
cot += "ûcot du dernier au premier point de cycle"
cycle = cycle :: "premier point de chaine"

retourner (ûcot , cycle)
```

2.2.3 Recherche locale

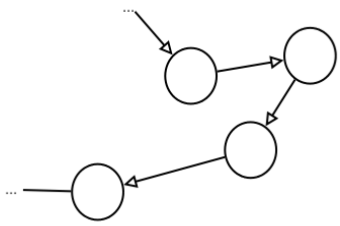
Les heuristiques nous donnant des solutions acceptables, choisies avec un minimum de « bon sens », il est ensuite possible de tenter d'améliorer ces solutions, via de la recherche locale. Partant d'une solution fournie, on va explorer les solutions voisines à cette dernière, afin de voire si on pourrait pas trouver des solutions encore meilleures parmi leur voisinage.

Exemple Un algorithme de recherche locale adapté au problème du voyageur de commerce est le 2-opt. Le principe du 2-opt consiste à tenter d'éliminer les « boucles » qui pourraient survenir dans le chemin, afin de le rendre plus court.

Ainsi, partant du chemin suivant (qu'on obtendrait logiquement en suivant l'heuristique consistant à aller sur le sommet le plus près) :



On obtiendrait ceci, en éliminant le croisement :



L'algorithme pour le 2-opt est le suivant :

```
é
Entre : un cycle hamiltonien (liste de sommets) et son ûcot
Sortie : un cycle hamiltonien et son ûcot éinfrieur ou égal au coup d'éentre

pour chaque couple de points (a, b) dans le cycle:û
    nouveau_cot = ûcot
                    − "ûcot de a à son successeur dans le cycle"
                    − "ûcot de b à son successeur dans le cycle"
                    + "ûcot de a à b"
                    + "ûcot du successeur de a et au successeur de b dans le

    si ûnouveau_cot < ûcot:û
        cot = ûnouveau_cot
        cycle = cycle écre en échangeant a et b dans cycle

retourner (ûcot, cycle)
```

Il est donc en $O(n^2)$ du nombre de sommets du cycle.

L'application du 2-opt sur le chemin obtenu via un heuristique simple peut donner des résultats plus proches de la solution optimale qu'on pourrait le penser, et la combinaison des deux est donc une bonne méthode.

2.2.4 Métaheuristiques

Plutôt que d'utiliser un simple heuristique pour trouver une solution à priori plutôt bonne, puis d'y appliquer une recherche locale pour tenter de l'améliorer encore, il est possible d'utiliser des « métaheuristiques ». Ces algorithmes vont avoir besoin d'heuristiques et de recherche locale, mais vont s'en servir en boucle, pour tenter sans cesse de trouver une solution meilleure.

Ils vont partir explorer différentes parties de l'espace, souvent en guidant leur exploration grâce à l'heuristique, et en essayant de retomber sur des parties de l'espace les plus intéressantes possibles grâce aux algorithmes de recherche locale.

Il existe énormément de métaheuristiques. En voici quelques uns :

Recherche locale itérée méta-heuristique très simple consistant à utiliser un heuristique puis appliquer de la recherche locale pour améliorer son résultat. Ensuite, on perturbe légèrement ce résultat, on applique à nouveau une recherche locale et on recommence.

Recherche tabou Amélioration de la recherche locale itérée, qui va utiliser une « liste tabou » banissant toute recherche autour des zones de l'espace déjà explorées.

Recuit simulé explore d'abord l'espace sans se restreindre aux parties donnant des solutions efficaces, puis se restreint de plus en plus au voisinage des solutions les plus efficaces. Converge donc vers les solutions les plus efficaces trouvées, puis relâche les contraintes et explore autour de ces dernières, quitte à trouver des solutions vraiment moins efficace. Recommence à se contraindre aux plus efficaces, etc...

Algorithmes génétiques imitent de la sélection naturelle, avec une population de solutions qui évoluent en mutant et en s'échangeant leurs caractéristiques entre elles. On peut même faire évoluer des populations séparément avec les modèles en îles, pour avoir plusieurs populations très différentes.

Colonies de fourmis imitent la encore la nature en simulant des phéromones déposées par des fourmis virtuelles, qui orientent la recherche au fil du temps.

...

2.2.5 Conclusion

Il est intéressant de constater que les heuristiques, méthodes de recherche locales, ainsi que les méta-heuristiques sont des choses extrêmement générales, utilisées pour résoudre énormément de problèmes demandant d'explorer un espace extrêmement grand.

Elles ne sont donc pas propres au voyageur du commerce, même si on a vu comment, dans le cas précis du voyageur du commerce, on pouvait obtenir des résultats corrects en se passant de métaheuristiques. On pourrait donc améliorer ces résultats en en utilisant.

3 UA : Programmation linéaire

4 UA : Jeux

4.1 Shifumi

Une stratégie simple et efficace à laquelle on pourrait penser pour gagner au Shifumi serait de jouer de manière aléatoire.

Et en effet, il s'avère que si les deux joueurs jouent de manière équiprobable, on a affaire à un équilibre de Nash : aucun changement de stratégie de la part d'un joueur ne pourra lui permettre d'augmenter ses chances de gagner.

De plus, si un adversaire ne joue pas de manière aléatoire (ou augmente la probabilité de jouer un certain élément), alors on pourra prévoir ce qu'il va jouer et donc trouver une stratégie qui pourra le battre. Les humains étant très mauvais pour jouer de manière aléatoire, il est assez facile d'écrire une stratégie permettant de les battre.

4.1.1 Stratégie développée

Afin de démontrer qu'un adversaire ne jouant pas aléatoirement est facile à battre, nous avons développé une stratégie qui se base sur des chaînes de Markov : en se basant sur les derniers éléments joués, elle regarde dans l'historique pour voir l'élément qui était joué le plus souvent par l'adversaire après les derniers coups joués.

Cette stratégie s'avère vraiment efficace contre un joueur humain. Toutefois, elle est prévisible : si on sait qu'on a affaire à une telle stratégie, on peut jouer de manière à la battre.

C'est pour cela qu'une stratégie aléatoire est la seule pouvant maximiser nos gains dans le pire des cas.

4.1.2 Variantes

Toutes les variantes du Shifumi qui consistent à rajouter des éléments pour obtenir un nombre d'éléments pair (par exemple pierre/papier/ciseaux/puits) vont créer un déséquilibre, car un élément sera moins efficace contre les autres. L'équilibre de Nash du jeu va alors consister à ne jamais jouer cet élément.

Si le nombre d'éléments est impair, alors le jeu pourra être équilibré, comme un Shifumi classique.

5 UA : Processus stochastiques

6 UA : Ingénierie robotique

7 Bilan

TODO : Un bilan devra être présenté dans ce dernier rapport final sur la façon de travailler de l'équipe au cours de toutes les UAs. Une autoévaluation devra y être conduite de sorte à faire apparaitre les points positifs et/ou les disfonctionnements apparus dans l'équipe.