INF8775 - Lab3 Analyses Asymptotiques

Fabrice Charbonneau, Félix Boulet et Giuseppe La Barbera

Avril 2019

Analyse asymptotique de la consommation de ressources pour l'algorithme utilisé pour résoudre le problème du TP3 du cours INF8775.

1 Code

Le pseudo code de l'algorithme génétique, en ne gardant que les parties pertinentes pour l'analyse, est le suivant :

1. ```
   Fonction genererPopulationAleatoire():
       POUR p individus:
           population.ajouter(genererIndividuAleatoire())
       RETOURNER population
   ```

2. ```
   Fonction genererIndividuAleatoire():
       individu.genes <- []
       TANT QUE total(piecesAUtiliser) > 0
           gene <- selectionnerModeleAleatoirement()
           ajouterModeleAIndividu(gene)
           mettreAJourPiecesAUtiliser(gene)
   ```

3. ```
   Fonction mutation(individu):
       individuInitial <- individu
       POUR i = 0, 1
           geneAjoute <- selectionnerGeneAleaValide(individu)
           geneEnleve <- selectionnerGeneAleaValide(individu)
           individu.genes.ajouter(geneAjoute)
           individu.genes.enlever(geneEnleve)
       remplirGenesGlouton(individu)
       SI individu.coutTotal < individuInitial.coutTotal
           RETOURNER individu
       SINON
           RETOURNER individuInitial
   ```

4. ```
   Fonction adaptation(survivants, population):
       POUR CHAQUE individu DANS POPULATION MOINS LES SURVIVANTS
           parent <- selectionnerIndividuAleatoire(survivants)
           ancienIndividu <- individu
           individu <- parent
           POUR i DE 0 A nombreAleatoireEntre(0, nModeles/5)
               geneEnleve <- selectionnerGeneAleaValide(individu
                   )
               individu.genes.enlever(geneEnleve)
           remplirGenesAleatoirement(individu)
           SI individu.coutTotal < individuInitial.coutTotal
               individu <- individu (conserver)
           SINON
               individu <- individuInitial
   ```

5. ```
   Fonction selectionnerSurvivants(population):
       trier(population) selon le coutTotal de chaque individu
       RETOURNER p/2 premiers elements de population
   ```

6. ```
   Fonction evolution(population):
       POUR CHAQUE individu DANS population
           mutation(individu)
       survivants <- selectionnerSurvivants(population)
       imprimer(survivants[0])
       adaptation(survivants, population)
   ```

7.
```
Fonction resoudreGenetique():
    population <- genererPopulationAleatoire()
    TANT QUE vrai:
        evolution(population)
```

2 Analyse

Afin d'obtenir une analyse de l'algorithme complet, nous allons analyser séparément chaque partie de l'algorithme.

Paramètres de l'algorithme :

— p : taille de la population / nombre d'individus (constant, mais peut varier dans l'algorithme)
— d : nombre de modèles possibles (relatif à la taille de l'exemplaire)
— g : nombre de types de pièces (relatif à la taille de l'exemplaire)
— n : nombre total de pièces à combler (relatif à la taille de l'exemplaire)

1. Pour la fonction genererPopulationAleatoire(), la complexité est dans l'ordre de $\Theta(p * g * n)$. En effet, on a une boucle qui itère sur p, le nombre d'individus à générer, et la logique à l'intérieur de la boucle (genererIndividuAleatoire()) dépend du nombre total de types de pièces et du nombre de pièces individuelles (remplir la solution jusqu'à ce que toutes les pièces soient utilisées dépend de n, car en pire cas, un modèle n'utilisera qu'une seule pièce). On doit faire la somme à chaque fois du nombre de pièces restantes à combler avec le gène (modèle) nouvellement ajouté à un individu.
2. Tel que mentionné pour la fonction précédente, la fonction genererIndividuAleatoire() se fait dans un temps dans l'ordre de $\Theta(g * n)$.
3. La fonction mutation(individu) se fait dans l'ordre de $\Theta(d * g * n)$, car la sélection aléatoire se fait en temps constant (autant pour le gène ajouté que celui enlevé), mais le coût total de l'individu doit être mis à jour par la suite en sommant le coût des modèles qu'il contient (d), et donc des pièces de chaque modèle (g). Le remplissage de l'individu de façon gloutonne (on remplit avec le modèle remplissant le plus possible la pièce ayant le plus grand nombre de pièces en demande) se fait en $\Theta(d * g * n)$, car on doit calculer le coût de chaque modèle possible comme s'il était ajouté aux gènes, avant d'en choisir un, et de recommencer tant que la solution n'est pas valide (comme la génération d'individu aléatoire).
4. Pour ce qui est de la fonction adaptation, on commence avec une boucle qui dépend de p. Dans cette boucle, après quelques opérations en temps constant, une boucle itère sur un nombre aléatoire mais borné selon le nombre de modèles d. Autrement dit, le nombre d'itérations suit une loi uniforme dont l'espérance est $(d/5)/2$ en moyenne, 1 en meilleur cas et d/5 en pire cas, bref la complexité de cette boucle est de $\Theta(d)$. Ensuite, à l'intérieur de cette boucle, des gènes sont enlevés en ajustant le coût des pièces et le coût total de l'individu, ce qui se fait au total en $\Theta(d * g * n)$

tout comme la mutation puisqu'on y effectue un remplissage de gènes également.

5. Pour la fonction selectionnerSurvivants, il s'agit simplement d'un tri en $\Theta(p\log(p))$. Retourner les $p/2$ premiers se fait en $\Theta(p)$ pour la copie, mais cette complexité est dominée par celle du tri.
6. En ce qui concerne la fonction evolution(population), elle se fait dans l'ordre de $\Theta(p*d*g*n)$ (mutation de p individus) + $\Theta(p\log(p))$ (sélection des survivants) + $\Theta(p*d*g*n)$ (adaptation). En pratique, les modèles contiennent beaucoup plus de pièces qu'une seule chacun, donc on peut s'attendre à une constante multiplicative assez petite petite affectant cette complexité.
7. Enfin, la fonction resoudreGenetique() est la fonction qui regroupe le tout en un seul bloc. D'abord, on y génère une population aléatoire en $\Theta(p*g*n)$. Ensuite, une boucle infinie appelle evolution(population) en $\Theta(p*d*g*n)$ + $\Theta((p\log(p))$, pour une complexité résultante de $\Omega(p*d*g*n+p\log(p))$, comme on ne sait pas quand sera arrêtée la boucle. En réalité, comme on contrôle la taille de la population (constante) et qu'elle ne dépend pas de l'exemplaire, on aura plutôt une complexité $\Omega(d*g*n)$