Rapport de TP 4MMAOD : Génération d'ABR optimal

Samy Amraoui
samy.amraoui@ensimag.grenoble-inp.fr

Matthieu Charly-Desroches
matthieu-charly-desroches@ensimag.grenoble-inp.fr

April 12, 2017

1 Équation de Bellman

Question 1

Considérons l'ensemble d'élements $e_0 < e1 < \cdots < e_{n-1}$ associés aux probabilités $(p_i)_{0\leq i\leq n-1}$. Soit A un ABR *optimal* pour les (e_i), i.e. minimisant la quantité $\sum_{i=0}^{n-1} p_i\Delta_A(e_i)$. Soit B un sous-arbre de A qui contient les noeuds $e_0 < e1 < \cdots < e_{m-1}$ avec $m \leq n$, et supposons que ce dernier n'est pas optimal. Comme B est un sous-arbre, il existe une constance $\delta \in \mathbb{N}$ telle que pour tout $i \in [\![0, m-1]\!], \Delta_B(e_i) = \Delta_A(e_i) - \delta$. Or B n'est pas optimal, donc il existe un arbre C reprenant les noeuds de B tel que $\sum_{i=0}^{m-1} p_i\Delta_C(e_i) < \sum_{i=0}^{m-1} p_i\Delta_B(e_i)$. On a alors :

$$\sum_{i=0}^{m-1} p_i\Delta_C(e_i) + \delta\left(\sum_{i=0}^{m-1} p_i\right) + \sum_{i=m}^{n-1} p_i\Delta_A(e_i) < \sum_{i=0}^{m-1} p_i\Delta_A(e_i) + \sum_{i=m}^{n-1} p_i\Delta_A(e_i)$$

En remplaçant le sous-arbre B par C dans A, on en déduit que $\forall i \in [\![1, m-1]\!], \Delta_C(e_i) + \delta = \Delta_{A'}(e_i)$ où A' est le nouvel arbre formé, donc on a :

$$\sum_{i=0}^{m-1} p_i\Delta_{A'}(e_i) + \sum_{i=m}^{n-1} p_i\Delta_A(e_i) < \sum_{i=0}^{m-1} p_i\Delta_A(e_i) + \sum_{i=m}^{n-1} p_i\Delta_A(e_i)$$
$$\sum_{i=0}^{n-1} p_i\Delta_{A'}(e_i) < \sum_{i=0}^{n-1} p_i\Delta_A(e_i)$$

ce qui est contradictoire avec le fait que A est un ABR *optimal*.

Pour calculer le coût de recherche d'un élément e_i de manière récursive en partant de la racine de l'ABR, on remarque qu'il suffit de sommer p_i à chaque descente dans les sous-arbres gauches et droits tant qu'on ne l'a pas trouvé. On peut alors obtenir récursivement le produit $p_i\Delta_A(e_i)$. Soit $C_{i,j}$ le coût de calcul de l'ABR optimal pour les éléments $e_i < \cdots < e_j$. Considérons que e_r où $r \in [\![i, j]\!]$ soit le noeud racine de cet ABR. On en déduit alors que :

$$C_{i,j} = \min_{r=i\ldots j}\left\{\underbrace{C_{i,r-1} + \sum_{k=i}^{r-1} p_k}_{\text{sous-arbre gauche}} + p_r + \underbrace{C_{r+1} + \sum_{k=r+1}^{j} p_k}_{\text{sous-arbre droit}}\right\} = \sum_{k=i}^{j} p_k + \min_{r=i\ldots j}\{C_{i,r-1} + C_{r+1,j}\}$$

avec comme conditions aux bords : $\forall i \in [\![0, n-1]\!], C_{i,i} = p_i$ et $\forall i > j, C_{i,j} = 0$.

Question 2

Les calculs des sommes $\sum_{k=i}^{j} p_k$ pour $0 \leq i \leq j \leq n-1$ peut se faire en $\Theta(n^2)$ additions en stockant en mémoire les sous-sommes successives de la forme $\sum_{k=i}^{j} p_k = \sum_{k=i}^{j-1} p_k + p_j$. Pour le calcul récursif du coût minimum des sous-arbres, on utilise une boucle `for` sur tous les couples (i, j) avec $0 \leq i \leq j \leq n-1$ soit $\Theta(n^2)$ opérations à l'intérieur desquelles on va itérer sur l'indice r entre i et j qui sera retenu comme minimum, il s'agit d'une boucle `for` interne en $\Theta(n)$. On en déduit au total un coût en $\Theta(n^3)$ opérations.

Au niveau du coût en mémoire, l'algorithme nécessite de stocker les valeurs des $C_{i,j}$ où $0 \leq i \leq j \leq n-1$ dans une matrice triangulaire de taille $n \times n$ et les $\sum_{k=i}^{j} p_k$ où $0 \leq i \leq j \leq n-1$ de même dans une autre matrice triangulaire de taille $n \times n$, soit un coût total en mémoire en $\Theta(n^2)$.

2 Principe de notre programme

Le programme se base sur le principe que tout sous-arbre d'un ABR optimal est lui-même un ABR optimal. Initialement, après avoir calculer les fréquences de tirage de chaque élément, le programme parcours détermine la racine de l'ABR optimal en parcourant l'ensemble des sommets avec une boucle `for` et en calculant à chaque fois le coût engendré par le choix de chaque noeud pour racine. Après avoir choisi la racine, il détermine récursivement les autres noeuds de l'ABR (qui sont les racines des sous-arbres de l'ABR principal) en itérant sur les noeuds restants à gauche et à droite de la racine (diviser pour régner). À chaque étape de la récursivité, le programme actualise le coût de l'ABR optimal et la matrice des racines `root`, où `root[i][j]` désigne la racine de l'ABR optimal formé des noeuds i à j. Enfin, le programme construit récursivement l'ABR optimal `BSTtree` en parcourant `root`.

3 Analyse du coût théorique

3.1 Nombre d'opérations en pire cas : $\Theta(n^3)$

Justification : La fonction `init_freqmatrix` initialise la matrice `freqmat` en affectant à `freqmat[i][j]` la valeur $\sum_{k=i}^{j} f(k)$ pour $i \leq j$. L'algorithme utilise le principe de mémoïsation : il réutilise la valeur de $\sum_{k=i}^{j-1} f(k)$ pour calculer $\sum_{k=i}^{j} f(k) = \sum_{k=i}^{j-1} f(k) + f(j)$. Les `freqmat[i][j]` sont calculés à l'aide de deux boucles `for` imbriquées qui parcourt la partie triangulaire supérieure de la matrice, soit $\Theta(n^2/2)$ additions.

La fonction `optCost` avec comme paramètres i et $j > i$ effectue $\Theta(j-i)$ opérations. Donc l'appel de `optCost` dans `optBSTcost` effectue j opérations soit au total :

$$\begin{aligned}
\sum_{j=0}^{n-1}\sum_{i=0}^{n-j} j &= \sum_{j=0}^{n-1} j(n-j+1) \\
&= \sum_{j=0}^{n-1}(n+1)j - \sum_{j=0}^{n-1} j^2 \\
&= \frac{n(n-1)(n+1)}{2} - \frac{n(n-1)(2n-1)}{6} \\
&= \frac{1}{6}\left[3n(n-1)(n+1) - n(n-1)(2n-1)\right] \\
&= \frac{1}{6}\left[3n^3 - 3n - 2n^3 + 3n^2 - n\right] \\
&= \frac{n^3 + n^2 - 4n}{6}
\end{aligned}$$

donc un coût théorique en $\Theta(n^3)$. La fonction `BSTcreate` se contente de parcourir récursivement l'ensemble des racines de la matrice `root`, ce qui coûte $\Theta(n)$ opérations élémentaires (c'est le nombre de noeuds de l'ABR).

On en conclut que le coût théorique est en $\Theta(n^3)$.

3.2 Place mémoire requise : $\Theta(n^2)$

Justification : En plus des variables locales des différentes fonctions, le programme utilise le tableau `BSTtree[][2]` de $2n$ cases, ainsi que les matrices `freqmatrix`, `costmatrix` et `root` de taille n^2 soit une utilisation mémoire totale en $3n^2 + 2n + O(1) = \Theta(n^2)$

3.3 Nombre de défauts de cache sur le modèle CO : $\Theta(n^2)$

Justification : Soit Z la taille du cache et L la taille d'un bloc, c'est-à-dire que le cache contient Z/L blocs (modèle CO). Dans la fonction `init_freqmatrix`, la partie triangulaire supérieure de la matrice `freqmat` est parcourue selon les lignes, ce qui engendre $\frac{n}{L}$ défauts de cache. Une exécution de la boucle `for` de la fonction `optCost` avec comme paramètres i et $j \geq i$ provoque $\frac{j-i}{L}$ défauts de cache, soit au total après exécution de `optBSTcost` :

$$\sum_{0 \leq i \leq j \leq n} \frac{j-i}{L} = \frac{1}{L}\sum_{k=0}^{n} k = \frac{n(n+1)}{2L}$$

défauts de cache. On en déduit au total $\Theta(n^2)$ défauts de cache.

4 Compte rendu d'expérimentation

4.1 Conditions expérimentaless

4.1.1 Description synthétique de la machine

Le programme a été lancé sur une machine avec ArchLinux, et compilé avec gcc 6.2.1. L'ordinateur possède 4GB de RAM, un processeur i5-1.6GHz avec deux coeurs physiques et l'hyperthreading activé. L'ordinateur n'avait pas de connexion Internet, très peu de processus en train de tourner, en soit il a été monopolisé pour ces tests. Le temps a été mesuré à l'aide de la commande `/usr/bin/time` en utilisant le temps réel qui s'est déroulé.

4.1.2 Méthode utilisée pour les mesures de temps

Afin d'effectuer les mesures de temps nous avons appelé la fonction `/usr/bin/time` pour mesurer le temps d'exécution de chaque appel. Nous appelions la fonction principale en passant en paramètre le nombre d'éléments dans le fichier et le fichier contenant les fréquences. Nous exécutons les 5 itérations du `benchmark1` puis les 5 du `benchmark2`, et ainsi de suite.

4.2 Mesures expérimentales

	temps min	temps max	temps moyen
benchmark1	0.002s	0.002s	0.002s
benchmark2	0.002s	0.002s	0.003s
benchmark3	3.35s	3.46s	3.386s
benchmark4	34.21s	38.11s	35.754s
benchmark5	119.36s	133.78	129.966s
benchmark6	557.71s	618.27s	599.534s

Figure 1: Mesures des temps minimum, maximum et moyen de 5 exécutions pour les 6 benchmarks.

4.3 Analyse des résultats expérimentaux

On a representé les résultats des benchmarks dans le tableau de la figure **??** puis on trace le temps en fonction de n sur le graphique de la figure **??**. On remarque alors que les points décrivent la courbe représentative d'une fonction proportionnelle à la fonction $f : x \mapsto x^3$. On en déduit que le coût expérimental en nombre d'opérations est bien en $\Theta(n^3)$

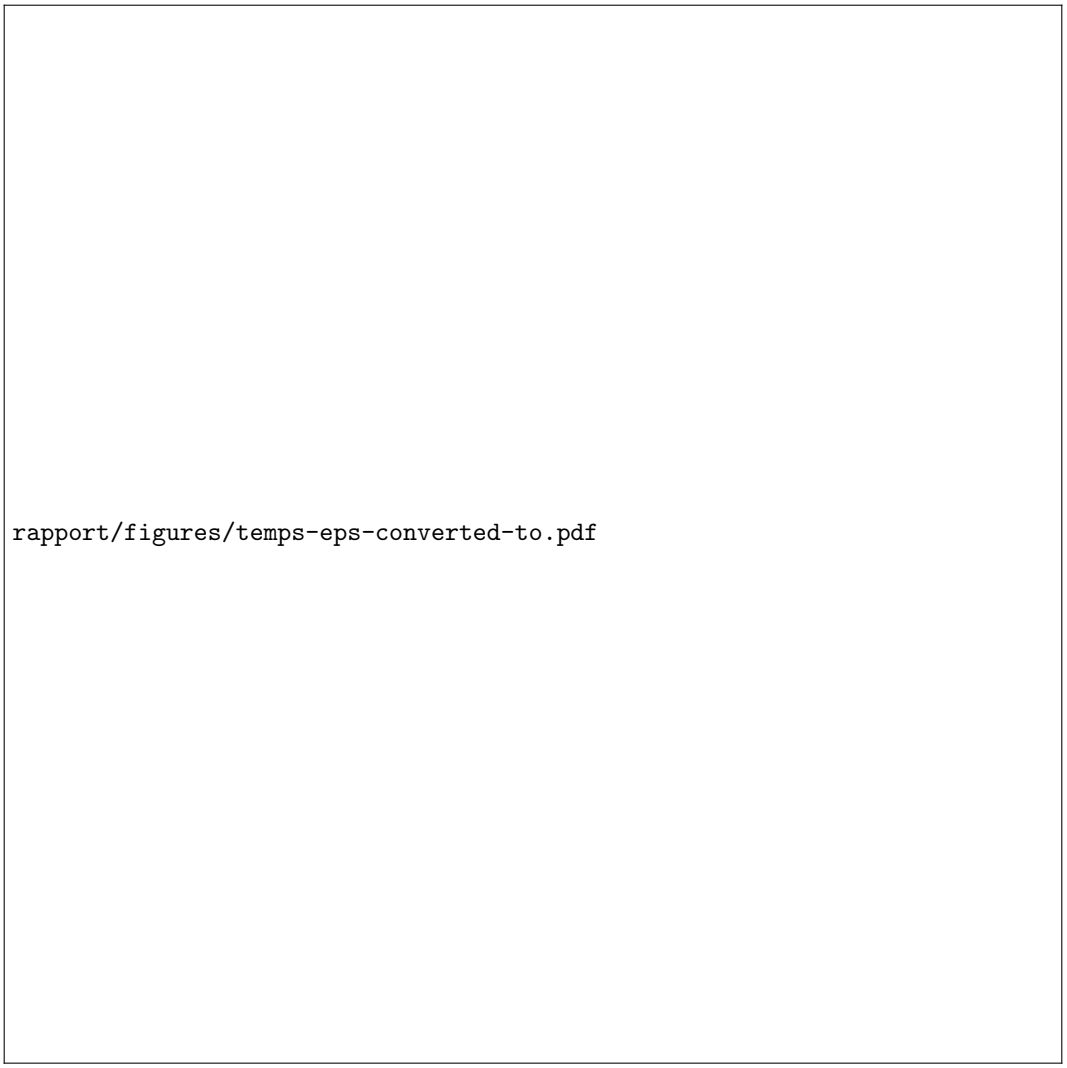

Figure 2: Temps d'exécution des benchmarks en fonction du nombre de noeuds n de l'ABR optimal