THÈSE

présentée

A L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

par

Francis PRUSKER

pour obtenir

LE GRADE DE DOCTEUR ES-SCIENCES

Spécialité : INFORMATIQUE

ASPECTS THÉORIQUES ET PRATIQUES DU TRI PAR FUSION SUR DISQUE

Soutenue 6-1-77 devant la Commission d'examen composée de :

MM. C. PAIR — Président

J. BERSTEL
D. COULON
M. MIGNOTTE
J. VUILLEMIN
— Examinateurs

THÈSE

présentée

A L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

par

Francis PRUSKER

pour obtenir

LE GRADE DE DOCTEUR ES-SCIENCES

Spécialité : INFORMATIQUE

ASPECTS THÉORIQUES ET PRATIQUES DU TRI PAR FUSION SUR DISQUE

Soutenue le 15 juin 1976 devant la Commission d'examen composée de :

MM.	C. PAIR	Président
	J. BERSTEL D. COULON M. MIGNOTTE J. VUILLEMIN	Examinateurs

Je remercie vivement Monsieur C. PAIR qui a suivi le travail présenté dans cette thèse et a bien voulu présider mon jury de thèse.

Je remercie Messieurs, J. BERSTEL, D. COULON, M. MIGNOTTE et J. VUILLEMIN d'avoir bien voulu participer au jury de cette thèse.

Je remercie L. HYAFIL et J. VUILLEMIN qui ont contribué à la recherche de certains résultats présentés dans ce travail.

TABLE DES MATIERES

INTRODUCTION

1. MOTIVATION

Jusqu'à ces dernières années, les bandes magnétiques étaient les seuls périphériques utilisés pour trier les grands fichiers ; naturellement, ce problème a été très étudié, les principaux résultats étant présentés par KNUTH [14].

Avec l'évolution de la technologie, les périphériques à accès direct, disques et tambours en particulier, sont de plus en plus utilisés : on peut estimer qu'à l'heure actuelle, les ordinateurs passent 10 % de leur temps à effectuer des tris sur disques (voir [6]). Cependant, bien que les bonnes stratégies pour ce type de tri externe diffèrent beaucoup de celles utilisées pour le tri sur bande magnétique, peu de recherches ont été publiées sur le sujet, comme en atteste encore KNUTH [13].

Le point de départ de cette thèse est la modélisation du tri-fusion sur disque faite par KNUTH [13] : la recherche de stratégies optimales de tri-fusion se ramène à un problème de recherche d'arbres minimisant une certaine fonction de coût. Ces arbres sont appelés arbres optimaux pour cette fonction de coût.

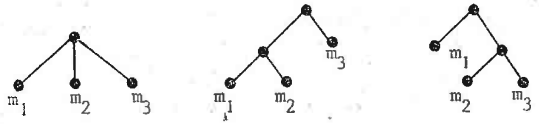
Dans une première partie, nous étudions les propriétés des arbres optimaux ainsi que la complexité des algorithmes de recherche pour une classe plus large de fonctions de coût. Sous cette forme plus générale, ce problème a des applications en théorie du codage (HUFFMAN [7], GALLAGER [5]) et en théorie des questionnaires (PICARD [17]), comme nous le montrons au paragraphe 7. Il s'applique également à des problèmes plus ésotériques, comme la construction de réseaux de permutations (PIPPENGER et VALIANT [18]).

Dans la seconde partie, les résultats obtenus nous permettent de proposer de bonnes stratégies de tri-fusion sur disque, adaptées aux configurations matérielles couramment utilisées. Nous évaluons également les performances de diverses configurations du point de vue du tri sur disque, permettant ainsi de faire des choix de configuration.

2. PRESENTATION DU TRI-FUSION SUR DISQUE

On peut distinguer deux étapes dans le tri d'un fichier sur périphérique externe. Dans une première étape, on génère une série de monotonies, ou "séquences initiales", que l'on range sur le périphérique. Dans une deuxième étape, que l'on appelle généralement "tri-fusion", ces monotonies sont fusionnée répétitivement, jusqu'à ce qu'on obtienne le fichier trié résultant. Cette deuxième phase peut être représentée par un "arbre de fusion", qui décrit dans quel ordre le processus doit s'exécuter.

Par exemple, si l'on a 3 monotonies m_1, m_2, m_3 à fusionner, les arbres de fus possibles sont :



Le troisième arbre décrit le processus suivant. Au début, les 3 monotonies sont rangées sur le disque. On effectue alors une "fusion élémentaire" de m_2 et m_3 pour obtenir une nouvelle monotonie, qui est le résultat du tri-interclassement de m_2 et m_3. A la fin de la fusion élémentaire, cette nouvelle monotonie est rangée sur disque. On effectue alors la fusion élémentaire de cette monotoni avec la monotonie m_1 pour obtenir le fichier trié résultant.

Nous décrirons au paragraphe 6 le processus de la fusion élémentaire. Disons simplement que le temps nécessaire pour l'effectuer peut être évalué en fonctio de la taille des monotonies à fusionner, de la stratégie d'allocation de mémoire centrale de l'ordinateur, de la configuration et des performances du matériel. Dans la suite, nous appellerons "degré" (ou "nombre de voies") de fusion le nombre de monotonies à interclasser au cours d'une fusion élémentaire.

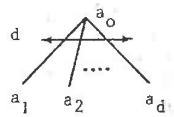
On peut donc associer à chaque arbre de fusion un temps (ou coût) de fusion, qui est égal à la somme des temps de chacune des fusions élémentaires à effectuer. Le temps de fusion élémentaire définit ainsi une fonction de coût sur les arbre de fusion.

3. LE MODELE DE KNUTH - TRAVAUX ANTERIEURS

L'approche de KNUTH est la suivante : étant donné n monotomies $m_1,\ldots,m_n$, quel est, parmi tous les arbres de fusion possibles, celui qui minimise le temp total de fusion ? Cet arbre est appelé arbre optimal pour la fonction de coût ainsi définie. Ce problème peut se formaliser de la manière suivante :

1) Pour une stratégie simple de fusion élémentaire, que nous décrirons en détail au paragraphe 6, le temps F d'une fusion élémentaire à d voies s'écrit :

(1) $F = (\alpha d + \beta)a_o$



$a_o = \sum_{1 \leq i \leq d} a_i$

Les constantes positives α et β dépendent du matériel utilisé. Les réels $a_1,\ldots,a_d$ sont les longueurs des monotonies à fusionner et a_o est la longueur de la monotonie résultante.

2) Evaluons maintenant le coût d'un arbre de fusion quelconque T. A chaque feuille f de T correspond une monotonie initiale ; nous noterons w(f), ou poids de la feuille f, la longueur de cette monotonie. A chaque noeud v de T correspond une fusion élémentaire, et donc une monotonie résultante ; nous noterons w(v), ou poids du noeud v, la longueur de cette monotonie. Remarquons que w(v) est la somme des poids des feuilles du sous-arbre de sommet v. Le coût de la fusion élémentaire correspondant au noeud v s'écrit donc :

(1) $(\alpha d(v) + \beta).w(v)$

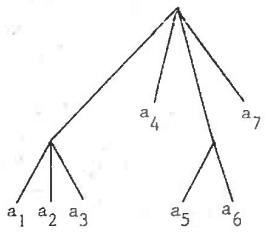
où d(v) désigne le degré du noeud v.

Le temps (ou coût) de fusion de l'arbre de fusion T est la somme des coûts des fusions élémentaires. La fonction de coût $\mathcal{C}$ sur les arbres de fusion s'écrit donc :

$$\mathcal{C}(T) = \sum_{v \in \mathcal{N}_T} \left[(\alpha d(v) + \beta).w(v) \right]$$

$\mathcal{N}_T$ désigne l'ensemble des noeuds de l'arbre T. Par exemple, le coût de l'arbre de fusion suivant pour 7 monotonies initiales de longueurs $a_1,\ldots,a_n$ est égal à :

T =



$$\mathcal{C}(T) = (a_1+a_2+a_3)(3\alpha+\beta) + (a_5+a_6)(2\alpha+\beta) + (a_1+\ldots+a_7)(4\alpha+\beta)$$

Le problème est donc le suivant : étant donné n monotonies initiales, trouver l'arbre optimal qui minimise le coût $\mathcal{C}(T)$. Naturellement, si nous choisissons une méthode de fusion élémentaire plus sophistiquée, nous obtiendrons une autre fonction de coût et les arbres optimaux correspondants seront différents.

Les travaux effectués jusqu'ici concernent principalement le cas où le temps de fusion élémentaire est donné par la formule (1), et où les monotonies initiales ont toutes même taille. On peut d'ailleurs supposer, sans faire de restriction, que cette taille est égale à l'unité de longueur. Le coût optimal dépend alors uniquement du nombre n de monotonies initiales. On peut donc le noter C(n). Nous montrons au chapitre 2 que C(n) est solution de la récurrence suivante :

$$\begin{cases} C(1) = 0 \\ C(n) = \underset{d\geq 2}{\text{Min}} \quad \underset{\sum_{1\leq i\leq d} n_i = n}{\text{Min}} \left[(\alpha d + \beta)n + \sum_{1\leq i\leq d} C(n_i) \right] \end{cases}$$

n, n_i et d sont des entiers positifs (les d-uples $n_1,\ldots,n_d$ représentent l'ensemble des partitions de n). La forme des arbres optimaux se déduit facilement de tout procédé permettant de calculer le coût optimal C(n).

Si $\alpha = 0$, l'arbre optimal est l'arbre plat. Si $\beta = 0$, KNUTH [13] donne une règle simple de formation des arbres optimaux. Dans les autres cas, les arbres optimaux sont très irréguliers. KNUTH [13] a proposé un algorithme de recherche de l'arbre optimal et de son coût C(n) en temps $O(n^2.\text{Log } n)$. VUILLEMIN et SCHLUMBERGER [20] ont montré que les degrés des noeuds des arbres optimaux sont bornés, mettant ainsi en évidence un algorithme de recherche en temps $O(n^2)$. HYAFIL, PRUSKER et VUILLEMIN ont borné inférieurement et supérieurement le coût optimal [8] et proposé un algorithme de recherche en temps O(Log n) quand la fonction $(\alpha d + \beta)/\text{Log } d$ a un seul minimum sur les entiers. Dans ce même cas, HYAFIL [10] a montré que si n est assez grand, il existe toujours un arbre optimal dont le degré au sommet est l'entier qui minimise $(\alpha d + \beta)/\text{Log } d$.

4. PRESENTATION DE LA PARTIE THEORIQUE

Nous généralisons d'abord ces résultats à une vaste classe de fonctions de coût, les rendant ainsi applicables aux problèmes cités au paragraphe 1 ainsi qu'à d'autres fonctions de coût liées au tri sur disque. Ces fonctions de coût $\mathcal{C}_\varphi$ sont définies par la donnée d'une fonction φ quelconque des entiers dans les réels positifs :

$$\mathcal{C}_\varphi(T) = \sum_{v\in\mathcal{V}_T} \left[\varphi(d(v)).w(v)\right] \qquad (3)$$

Les résultats sont obtenus par des méthodes analytiques, essentiellement différentes des techniques combinatoires utilisées en [8], [9], [10] et [20] dans le cas où $\varphi(d) = \alpha d+\beta$

Nous obtenons ensuite des résultats plus fins pour les fonctions de coût $\mathcal{C}_\varphi$ de la formule (3). Si la fonction $\varphi(d)/\text{Log } d$ a un seul minimum, nous présentons une règle relativement simple de formation des arbres optimaux permettant de calculer la forme et le coût des arbres optimaux en temps constant. Pour les autres fonctions de coût, nous montrons que, si n est assez grand, le degré du sommet des arbres optimaux est l'un des entiers qui minimisent $\varphi(d)/\text{Log } d$.

Nous répondons à diverses conjectures relatives à ce problème, ainsi qu'aux questions posées par KNUTH [13], p. 377, Ex. 9. En particulier, nous montrons qu'il n'existe pas toujours un arbre optimal dont la différence de profondeur des feuilles n'excède pas un, et que, pour certaines fonctions de coût, cette différence tend vers l'infini avec n.

Résumé de la partie théorique

Le premier chapitre précise la terminologie, indique les définitions et les propriétés élémentaires, et donne des conditions suffisantes sur φ pour que les degrés des arbres optimaux soient bornés.

Les quatres chapitres suivants traitent le cas où les feuilles des arbres optimaux ont toutes même poids. Le coût optimal C(n) est alors solution de la récurrence suivante :

$$\begin{cases} C(1) = 0 \\ C(n) = \min\limits_{d\geq 2} \min\limits_{\sum_{1\leq i\leq d} n_i = n} \left[\varphi(d)n + \sum_{1\leq i\leq d} C(n_i) \right] \end{cases}$$

Il s'agit d'étudier cette récurrence et les propriétés des arbres optimaux correspondants, ainsi que la complexité des algorithmes de recherche.

Au chapitre 2, nous bornons inférieurement et supérieurement le coût optimal et donnons quelques indications sur la forme des arbres optimaux, généralisant ainsi les résultats obtenus en [8] et [20].

Au chapitre 3, nous étudions les propriétés de l'enveloppe convexe de la fonction C(n), mettant en évidence la forme asymptotique de C(n). Nous sommes ensuite amenés à distinguer deux catégories de fonctions φ, pour lesquelles les propriétés des arbres optimaux sont assez différentes.

Pour la première catégorie de fonctions φ, nous montrons au chapitre 4 que le degré du sommet des arbres optimaux tend vers une limite avec n, généralisant ainsi le résultat obtenu en [9] et [10]. Nous présentons ensuite un algorithme de calcul des arbres optimaux et de leur coût en temps constant. Enfin, nous montrons que, pour une fonction de coût donnée, la différence de profondeur des feuilles des arbres optimaux est bornée, mais que cette borne n'est pas toujours égale à l'unité.

Au chapitre 5, nous étudions les propriétés des arbres optimaux pour la deuxième catégorie de fonctions φ. Nous montrons en particulier que la différence de profondeur des feuilles tend vers l'infini avec n.

Au chapitre 6, nous étudions la forme des arbres optimaux dans le cas où les feuilles ont des poids quelconques. Nous bornons inférieurement et supérieurement le coût optimal et présentons des algorithmes et heuristiques de calcul des arbres optimaux.

5. PRESENTATION DE LA PARTIE PRATIQUE

Il s'agit ici d'étudier les problèmes pratiques posés par le tri-fusion sur disque, et, plus généralement, sur tous les périphériques à accès direct (tambours, cartes magnétiques par exemple). Nous proposons diverses stratégies de tri-fusion selon le contrôle de l'utilisateur sur les entrées/sorties et selon la configuration matérielle utilisée. L'analyse des performances de ces diverses stratégies permet de faire des choix de configuration.

Au chapitre 7, nous présentons les caractéristiques de quelques périphériques à accès direct ainsi que les diverses configurations possibles. Selon le type de périphériques utilisés et les possibilités offertes par le système d'exploitation, nous distinguons deux méthodes principales de tri-fusion : la méthode à temps d'accès constant et la méthode optimisant les mouvements de bras du disque.

Au chapitre 8, nous étudions la méthode à temps d'accès constant. Elle s'applique si l'utilisateur dispose d'un tambour ou d'un disque à têtes fixes. Elle s'applique également pour un disque à têtes mobiles si l'utilisateur ne contrôle pas suffisamment les entrées/sorties pour optimiser les mouvements de bras du disque.

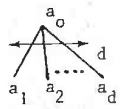
Au chapitre 9, nous étudions le tri-fusion avec optimisation du mouvement de bras. Nous présentons un algorithme d'optimisation des mouvements de bras quand l'utilisateur dispose d'un disque à têtes mobiles. Nous étudions également le cas où l'utilisateur dispose de deux disques à têtes mobiles et comparons les performances des diverses méthodes de tri-fusion.

6. MODELISATION DU TRI-FUSION SUR DISQUE

Nous présentons en détail le processus de tri-fusion et analysons son coût pour une stratégie simple, justifiant ainsi les formules (1) et (2) du paragrap

6.1. Description de la fusion élémentaire

Il s'agit de fusionner d monotonies $m_1, m_2, \ldots, m_d$ rangées sur disque en une seule monotonie m_o. A la fin de la fusion élémentaire, la monotonie m_o devra être rangée sur disque. Une fusion élémentaire à d voies peut être représentée de la manière suivante :



$$\sum_{1 \leqslant i \leqslant d} a_i = a_o$$

Nous appelons a_i la longueur de la monotonie m_i. Les monotonies $m_1, \ldots, m_d$ sont appelées "monotonies entrantes" et m_o "monotonie résultante".

Avant de commencer la fusion, il faut réserver sur le disque un emplacement de taille a_o pour la monotonie résultante m_o. Il faut également partager la place M disponible en mémoire centrale en (d+1) tampons de tailles $b_o, b_1, \ldots, b_d$. On a évidemment $\sum_{1 \leqslant i \leqslant d} b_i = M$.

A chaque monotonie entrante m_i correspond un tampon d'entrée de taille b_i ; à la monotonie résultante m_o correspond le tampon de sortie de taille b_o.

Nous représentons cela schématiquement sur la figure suivante :

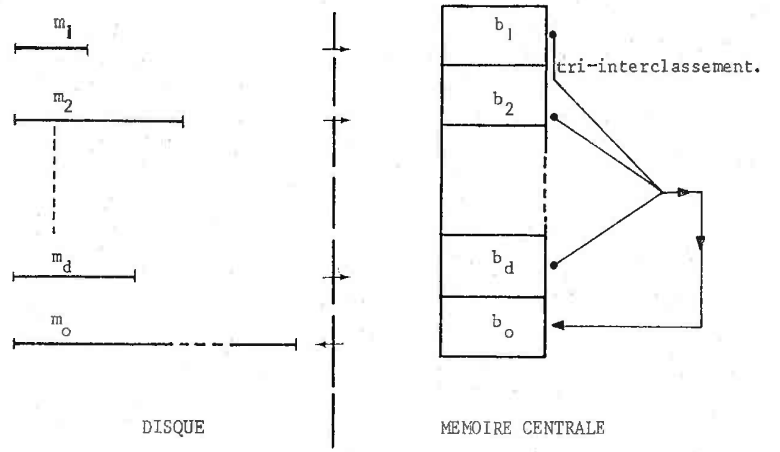


FIGURE 1

Au début, on remplit chaque tampon d'entrée avec le début de la monotonie correspondante ; le tampon de sortie est vide. On effectue alors en mémoire centrale la fusion des tampons d'entrée par un programme classique de tri-interclassement, tri dont on range le résultat au fur et à mesure dans le tampon de sortie. Ce tri se poursuit jusqu'à ce qu'un tampon d'entrée devienne vide ou que le tampon de sortie devienne plein. Dans le premier cas, on remplit le tampon vide avec la suite de la monotonie correspondante. Dans le deuxième cas, on vide le tampon de sortie à l'emplacement réservé à la monotonie résultante m_o.

Le temps F pris par la fusion élémentaire se décompose en trois parties :

1) temps total de traitement de l'ordinateur : Tm

2) temps total d'accès : pour chaque entrée/sortie, il y a un temps d'attent - ou "temps accès" - avant que le transfert puisse commencer (temps de positionnement du bras si le disque est à bras mobile plus temps de rotation du disque). Le nombre d'accès à la séquence m_i est $\lceil a_i/b_i \rceil$, c'est-à-dire l'entier égal ou immédiatement supérieur à a_i/b_i.

Dans la pratique, les monotonies étant beaucoup plus longues que le tampon qui leur correspond en mémoire centrale, on ne commettra qu'une erreur minime en remplaçant $\lceil a_i/b_i \rceil$ par a_i/b_i. Si Ta_i désigne le temps d'accès à la monotonie m_i, le temps total d'accès est égal à :

$$\sum_{0 \leq i \leq d} Ta_i \frac{a_i}{b_i}$$

3) temps total de transfert : chaque enregistrement étant lu depuis le disq puis écrit sur le disque, le temps total de transfert est égal à :

$$2\ Tt\ a_o$$

où T_t désigne le temps de transfert par unité de longueur.

Dans le cas général, le temps F d'une fusion élémentaire est donc égal à

$$F = 2\ Tt\ a_o + \sum_{o \leq i \leq d} Ta_i \frac{a_i}{b_i} + Tm \qquad (4)$$

6.2. Temps de fusion élémentaire pour une stratégie simple

Nous supposerons d'abord que le temps d'accès est constant. Dans le cas de tambours ou de disques à têtes fixes, le temps d'accès Ta est en moyenne la moitié du temps d'une rotation du disque. Dans le cas de disques à têtes mobiles, on est obligé de prendre pour temps d'accès une valeur moyenne Ta si l'utilisateur n'a aucun contrôle sur l'emplacement des monotonies sur disque (dans le cas contraire, il est possible d'optimiser le mouvement du bras du disque - voir chapitre 9).

Nous supposerons ensuite que les tampons en mémoire centrale ont tous même taille, soit M/(d+1). Nous traitons au chapitre 8 du choix d'une meilleu stratégie d'allocation de la mémoire centrale.

Enfin, nous ne tiendrons pas compte du temps de traitement en mémoire centrale, ce problème étant examiné au chapitre 8.

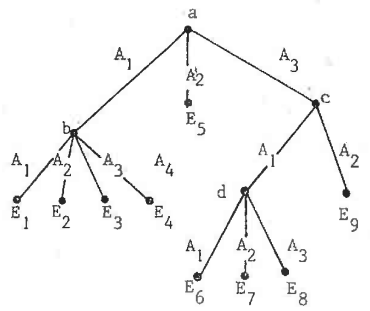
En utilisant la formule (4), un calcul simple montre qu'avec ces hypothèses, le temps de fusion élémentaire s'écrit :

$$F = (\alpha d + \beta).a_o \qquad \alpha = 2.\frac{Ta}{M} \qquad \beta = 2\left(\frac{Ta}{M} + Tt\right) \qquad (5)$$

ce qui correspond bien au temps de fusion élémentaire du paragraphe 3.

7. ANALOGIE AVEC LE CODAGE ET LA THEORIE DES QUESTIONNAIRES

Supposons que nous disposions d'un alphabet infini $A_1, A_2, \ldots, A_i, \ldots$ et que nous voulions coder dans cet alphabet n événements $E_1, \ldots, E_n$ de probabilités $p_1, \ldots, p_n$. Ce codage peut être représenté par un arbre à n feuilles (voir GALLAGER [5], HUFFMAN [7], PICARD [17]). L'arbre suivant représente un codage possible pour 9 événements $E_1, \ldots, E_9$.



Le code de l'événement E_1 est A_1A_1, celui de E_4 est A_1A_4, celui de E_5 est A_2, celui de E_7 est $A_3A_1A_2$, etc... Pour identifier l'événement E_4, il faudra d'abord poser la question correspondant au noeud a, qui a trois réponses possibles, puis la question b, qui a quatre réponses possibles. A chaque événement E_i correspond donc un temps d'identification $c(E_i)$ égal à la somme des temps pris par les questions nécessaires à l'identification de E_i.

Si $\varphi(d)$ est le temps pris par une question à d réponses, le temps d'identification de E_i est égal à :

$$c(E_i) = \sum_{v \in L(E_i)} \varphi(d(v))$$

où $L(E_i)$ désigne l'ensemble des noeuds sur le chemin qui mène du sommet à la feuille E_i. Le temps moyen d'identification $\mathcal{C}(T)$ correspondant à un arbre de codage T est donc :

$$\mathcal{C}(T) = \sum_{1 \leq i \leq n} p_i . c(E_i) \qquad (6)$$

Le problème est le suivant : trouver le codage optimal, c'est-à-dire l'arbre qui minimise le temps moyen d'identification $\mathcal{C}(T)$.

Reprenons les notations du paragraphe 3 en remplaçant les longueurs des n monotonies initiales par les probabilités des n événements. Un calcul simple montre que la formule (6) peut s'écrire :

$$\mathcal{C}(T) = \sum_{v \in \mathcal{N}_T} [\varphi(d(v)).w(v)]$$

Nous retrouvons la formule (4) : la fonction de coût sur les arbres est donc la même dans le cas du codage et dans le cas du tri-fusion. Le problème de la recherche de l'arbre optimal est donc le même dans les deux cas.

CHAPITRE I

DEFINITIONS ET PROPRIETES ELEMENTAIRES

Nous présentons ici les définitions ainsi que les propriété élémentaires des arbres de fusion. Le premier paragraphe est consacré à la terminologie que nous utiliserons sur les arbres et les transformations d'arbres. La plupart des termes sont classiques. Nous définissons ensuite la fonction de coût sur les arbres et évaluons le coût des arbres possédant certaines régularités. Nous donnons quelques propriétés élémentaires des arbres optimisant le coût et examinons les transformations d'arbres préservant l'optimalité. Nous disons enfin quelles conditions doit satisfaire la fonction de coût pour que les degrés des arbres optimaux soient bornés.

1.1. TERMINOLOGIE

Définition 1 : Un arbre de fusion T est une arborescence A dont les feuilles f ont des poids w(f) réels positifs ou nuls.

Par exemple :

A =

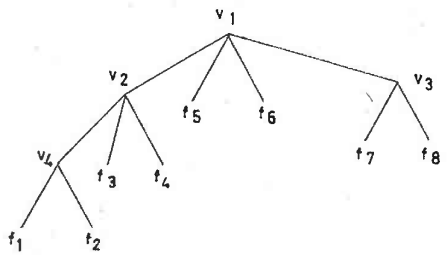


FIGURE 1

Dans la suite, nous parlerons le plus souvent d' "arbre", au lieu d'arbre de fusion. Généralement, un arbre sera représenté par une arborescence dont le feuilles sont valuées.

L'arbre T suivant est une valuation des feuilles de l'arborescence A de la figure 1.

T =

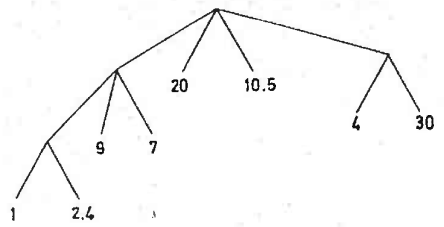


FIGURE 2

Nous emploierons les notations et la terminologie suivantes sur les arborescences et les arbres :

$\mathcal{N}_T$ désigne l'ensemble des noeuds de l'arbre ; sur l'exemple :

$$\mathcal{N}_T = \{v_1, v_2, v_3, v_4\}$$

$\mathcal{F}_T$ désigne l'ensemble des feuilles de l'arbre ; sur l'exemple :

$$\mathcal{F}_T = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8\}$$

S_T désigne la suite des poids des feuilles de l'arbre T, c'est-à-dire le n-uple constitué par les poids de ses n feuilles ; sur l'exemple $S_T = (4, 20, 7, 10.5, 2.4, 30, 9, 1)$. L'opération qui consiste à attribuer les poids d'un n-uple S aux n feuilles d'une arborescence sera appelé placement. On dira qu'on place S sur A.

ascendant d'un noeud ou d'une feuille : sur l'exemple, les ascendants du noeud v_4 sont les noeuds v_2 et v_1 ; les ascendants de la feuille f_2 sont les noeuds v_4, v_2 et v_1.

indépendance : un ensemble de noeuds ou de feuilles est dit indépendant si aucun d'eux n'est ascendant d'un autre ; sur l'exemple, $\{v_4, f_4, f_5, v_3\}$ est indépendant.

père d'un noeud ou d'une feuille : c'est l'ascendant direct ; sur l'exemple le père de v_3 est v_1 ; le père de f_4 est v_2.

$\mathcal{F}(v)$ désigne l'ensemble des fils du noeud v, c'est-à-dire les descendants directs de v ; sur l'exemple : $\mathcal{F}(v_2) = \{v_4, f_3, f_4\}$.

sommet de l'arbre : c'est le noeud qui ne possède pas de père.

noeud terminal : c'est un noeud dont tous les fils sont des feuilles de l'arbr

T(v) ou sous-arbre engendré par le noeud v : c'est le sous-arbre de sommet v dont les feuilles ont même poids que dans l'arbre initial. Sur l'exemple :

$T(v_2)$ =

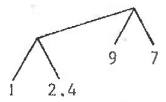


FIGURE 3

sous-arbres principaux : ce sont les sous-arbres engendrés par les fils du sommet de l'arbre, appelés noeuds principaux.

Définition 2 : On appelle arbre de fusion à poids égaux un arbre dont toutes les feuilles ont le même poids.

Définition 3 : Le poids w(v) d'un noeud v est la somme des poids des feuilles du sous-arbre T(v) engendré par le noeud v. Le poids d'un arbre T est la somme des poids de ses feuilles ; on le note w(T).

Sur l'exemple, on a $w(v_2) = 19.4$, $w(v_4) = 3.4$ et $w(T) = 83.9$

Définition 4 : Le degré d(v) d'un noeud v est égal au nombre de ses fils. Le degré d'un arbre T est le degré du sommet de cet arbre ; on le note d(T).

Sur l'exemple, on a $d(v_4) = 2$, $d(v_2) = 3$ et $d(T) = 4$

Définition 5 : On appelle profondeur d'un noeud (resp. d'une feuille) le nombre d'ascendants de ce noeud (resp. de cette feuille).

La profondeur du sommet est donc zéro. Sur l'exemple, la profondeur de v_3 est un, la profondeur de f_1 est trois.

Définition 6 : On appelle chemin L(v) (resp. L(f)) d'un noeud v (resp. d'une feuille f) l'ensemble des ascendants de ce noeud (resp. de cette feuille). La profondeur du chemin est la profondeur de ce noeud (resp. de cette feuille).

Sur l'exemple, le chemin de f_2 est $\{v_4, v_2, v_1\}$; le chemin de v_4 est $\{v_2, v_1\}$

1.1.1. TRANSFORMATIONS SUR LES ARBRES

Définition 7 : Nous définissons sur les arbres de fusion les transformations suivantes (T désigne l'arbre initial).

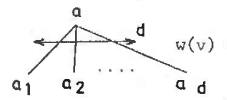
Contraction T/v d'un noeud v : c'est la transformation qui consiste à remplacer le noeud v par une feuille de poids w(v). On peut contracter plusieurs noeuds indépendants $v_1, v_2, \ldots, v_t$. On note la contraction $T/v_1, \ldots, v_t$.

Expansion d'une feuille f par un arbre T' (l'arbre T' doit avoir même poids que la feuille f) : c'est la transformation qui consiste à remplacer la feuille f par un noeud qui engendrera le sous-arbre T'.

Remplacement d'un noeud v par un arbre T' (l'arbre T' doit avoir le même poids que le noeud v) : c'est la transformation qui consiste à remplacer le sous-arbre engendré par v par le sous-arbre T'.

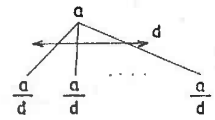
Equilibrage d'un noeud terminal v : c'est la transformation qui consiste à remplacer le noeud terminal v par un noeud terminal dont tous les fils ont même poids :

Noeud terminal initial :

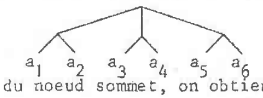


$w(v) = a = \sum_{1 \leq i \leq d}$

Noeud terminal transformé :

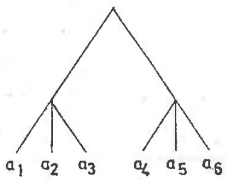


Filtrage d'un noeud v : si tous les fils de v sont des noeuds d même degré d, le filtrage consiste à remplacer le sous-arbre T(v) par un sous-arbre de degré d dont tous les fils ont degré d(v). Par exemple, si l'arbre initial est

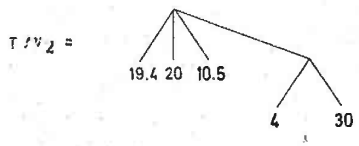


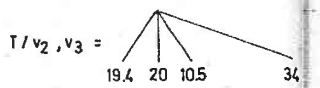
$d(v) = 3$ $d = 2$

par filtrage du noeud sommet, on obtient l'arbre :

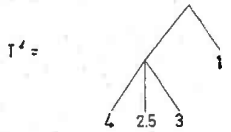


Toutes ces transformations laissent invariant le poids de l'arbre. Mais seul le filtrage laisse invariant sa suite de poids. Donnons quelques exem de transformations sur l'arbre T des figures 1 et 2. Voici deux exemples d contractions de T :

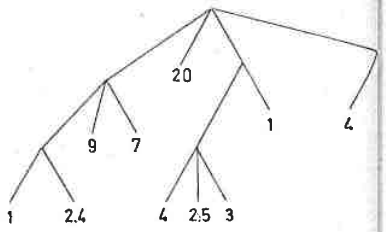




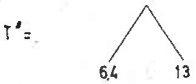
L'expansion de f_6 par l'arbre T' suivant donne l'arbre :



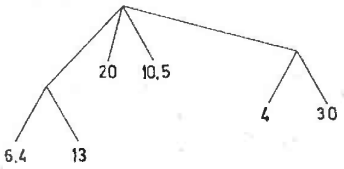
Expansion de f_6 par T' =



Le remplacement de v_2 par l'arbre T' suivant donne l'arbre :



Remplacement de v_2 par T' =



1.2. DEFINITION ET PROPRIETES ELEMENTAIRES DE LA FONCTION DE COUT

Définition 8 : Nous définissons sur les arbres la fonction de coût suivante :

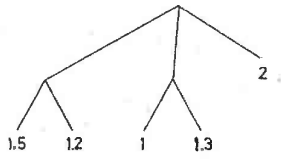
(1) $$\mathcal{C}_\varphi(T) = \sum_{v \in \mathcal{N}_T} \left[\varphi\left(d(v)\right) \cdot w(v)\right]$$

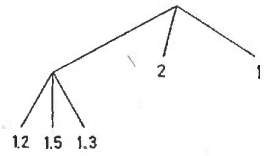
où φ est une fonction quelconque des entiers plus grands ou égaux à deux dans les réels strictement positifs. L'arbre réduit à une feuille a un coût nul.

Notons que la fonction de coût est entièrement définie par la donnée de la fonction φ. Généralement, nous supposerons la fonction φ donnée et nous noterons simplement $\mathcal{C}(T)$ le coût d'un arbre T.

Définition 9 : Deux arbres sont dits équivalents s'ils ont même coût et même suite de poids des feuilles.

Si $\varphi(d) = d+2$, les deux arbres suivants sont équivalents pour la suite (2,1.5,1,1.3,1.2) :





Nous examinons maintenant différentes méthodes de calcul du coût, ainsi que les modifications du coût entraînées par les transformations de la définition

Définition 10 : Le coût partiel c(v) d'un noeud v est égal à :

(2) $$c(v) = \sum_{u \in L(v)} \varphi\left(d(u)\right)$$

On définit de même le coût partiel d'une feuille.

Sur l'exemple de la figure 1, on a $c(f_8) = \varphi(2)+\varphi(4)$ et $c(v_4) = \varphi(3)+\varphi(4)$. Notons que le coût partiel ne dépend pas des poids.

Proposition 1 : Le coût $\mathcal{C}(T)$ d'un arbre T peut être calculé par les quatre formules suivantes :

(3) 1) $$\mathcal{C}(T) = \sum_{v \in \mathcal{N}_T} \left[\varphi\left(d(v)\right) w(v)\right]$$

(4) 2) $$\mathcal{C}(T) = \mathcal{C}(T/v) + \mathcal{C}\left(T(v)\right)$$

où v désigne un noeud quelconque de T. Cette formule est appelée formule des contractions.

(5) 3) $$\mathcal{C}(T) = \sum_{f \in \mathcal{F}_T} \left[c(f)\, w(f)\right]$$

Le coût d'un arbre est donc la somme des coûts partiels des feuilles pondérés par leurs poids. Cette formule est appelée formule des coûts partiels.

(6) 4) $$\mathcal{C}(T) = \varphi\left(d(T)\right) w(T) + \sum_{1 \leqslant i \leqslant d(T)} \mathcal{C}(T_i)$$

$T_1, T_2, \ldots, T_{d(T)}$ désignent les sous-arbres principaux de T.

Preuve :

La méthode des contractions revient à faire une partition des noeuds de $\mathcal{N}$ en deux sous-ensembles : le sous-ensemble des noeuds de T/v et celui des noeuds de T(v). On obtient ainsi la formule (4) à partir de la formule (3). On en déduit la formule (6).

Montrons maintenant la formule (5). La formule (3) de définition du coût peut s'écrire :

(7) $$\mathcal{C}(T) = \sum_{v \in \mathcal{N}_T} \left[\varphi\left(d(v)\right) \cdot \sum_{f \in T(v)} w(f)\right]$$

Considérons maintenant une feuille f. Dans la formule (7), w(f) est en facteur de tous les termes $\varphi\left(d(v)\right)$, où v est un ascendant de f. Si nous inversons l'ordre des sommations, nous obtenons donc :

$$\mathcal{C}(T) = \sum_{f \in \mathcal{F}_T} \left[w(f) \cdot \sum_{v \in L(f)} \varphi\left(d(v)\right)\right]$$

D'après la définition du coût partiel, on en déduit (5). ▨

Proposition 2 : Le filtrage et l'équilibrage laissent invariant le coût.

Preuve :

Pour l'équilibrage, c'est évident.
Pour le filtrage : soit v le noeud que l'on veut filtrer, d son degré, et d' le degré de ses fils. D'après (4), le coût de l'arbre T s'écrit :

$$\mathcal{C}(T) = \mathcal{C}(T/v) + \mathcal{C}\left(T(v)\right)$$

Si nous appelons $v_1, v_2, \ldots, v_q$ les petits-fils (fils de fils) de v ($q = d.d'$), le coût de T(v) s'écrit :

$$\mathcal{C}\left(T(v)\right) = \mathcal{C}\left(T(v)/v_1, \ldots, v_q\right) + \sum_{1 \leqslant i \leqslant q} \mathcal{C}\left(T(v_i)\right)$$

Calculons maintenant le coût de $T(v)/v_1, \ldots, v_q$ par la méthode des coûts partiels :

$$\mathcal{C}\left(T(v)/v_1, \ldots, v_q\right) = \sum_{1 \leqslant i \leqslant q} \left[w(v_i)\left(\varphi(d) + \varphi(d')\right)\right]$$

Ce qui s'écrit :

$$\mathcal{C}\left(T(v)/v_1, \ldots, v_q\right) = \left[\varphi(d) + \varphi(d')\right] w(v)$$

Le coût de l'arbre s'écrit donc :

$$\mathcal{C}(T) = \mathcal{C}(T/v) + \Big[\varphi(d)+\varphi(d')\Big].w(v) + \sum_{1\leqslant i\leqslant q} \mathcal{C}\Big(T(v_i)\Big)$$

Cette formule étant symétrique en d et d', l'opération de filtrage ne change pas le coût.

1.3. DEFINITION ET COUT DE QUELQUES ARBRES PARTICULIERS

Nous allons étudier ici quelques types d'arbres possédant certaines régularités.

Définition 11 : Un arbre est dit plat si toutes ses feuilles sont à la profondeur 1.

Le coût d'un arbre plat T est donc :

$$\mathcal{C}(T) = \varphi\Big(d(T)\Big).w(T) \tag{8}$$

L'arbre obtenu depuis un arbre quelconque par contraction de ses noeuds principaux sera appelé arbre plat sommital.

Définition 12 : Un arbre est dit régulier si toutes ses feuilles sont à la même profondeur et si tous les noeuds situés à une même profondeur ont même degré. On appelle décomposition d'un arbre régulier un ensemble noté $(d_1^{\ell_1},\ldots,d_t^{\ell_t})$ où les d_i sont tous différents, et où ℓ_i est égal au nombre de profondeurs où les noeuds ont degré d_i.

Par exemple, l'arbre régulier suivant a pour décomposition $(3^1,2^2)$.

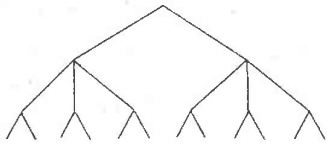


Fig. 3

Si n est le nombre de feuilles et p leur profondeur, on a

$$n = \prod_{1\leqslant i\leqslant t} d_i^{\ell_i} \quad \text{et} \quad p = \sum_{1\leqslant i\leqslant t} \ell_i \tag{9}$$

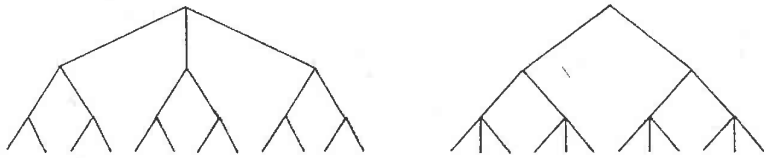
A toute décomposition de l'arbre régulier est donc associée une factorisation de n. La réciproque est évidemment fausse. D'autre part, il est évident que toutes les feuilles ont même coût partiel c :

$$c = \sum_{1\leqslant i\leqslant t} \Big[\ell_i\varphi(d_i)\Big] \tag{10}$$

Le coût d'un arbre T régulier de décomposition $(d_1^{\ell_1},\ldots,d_t^{\ell_t})$ est donc :

$$\mathcal{C}(T) = w(T).\sum_{1\leqslant i\leqslant t}\Big[\ell_i\varphi(d_i)\Big] \tag{11}$$

D'après cette formule, le placement des poids sur les feuilles n'influe pas sur le coût. De même, l'ordre des degrés des différentes profondeurs n'influe pas sur le coût. Quel que soit le placement des poids sur les feuilles, les deux arbres suivants ont donc même coût que l'arbre de la figure 3 :



Définition 13 : Un arbre d-parfait est un arbre régulier dont tous les noeuds ont même degré d.

La décomposition d'un arbre d-parfait est donc (d^p), où p est la profondeur des feuilles. Avec les mêmes notations que pour les arbres réguliers, nous avons :

$$n = d^p \tag{12}$$

$$c = p\,\varphi(d) \tag{13}$$

$$\mathcal{C}(T) = p.d^p\varphi(d) \tag{14}$$

Définition 14 : Un arbre est dit d-équilibré si son arborescence a la forme suivante : jusqu'à la profondeur (p-1), tous les noeuds ont degré d ; à la profondeur p, il y a r noeuds terminaux de degré d, un noeud terminal de degré s ($2 \leqslant s \leqslant d$), et ($d^p - r - 1$) feuilles. Si n est le nombre de feuilles de l'arbre, les entiers p, r et s sont déterminés de façon unique par :

$$d^p < n \leqslant d^{p+1} \tag{15}$$

$$n - d^p - 1 = (d-1)r + (s-2) \tag{16}$$

Les entiers r et (s-2) sont respectivement le quotient et le reste de la division de ($n-d^p-1$) par (d-1).

Notons que dans un arbre d-équilibré tous les noeuds, sauf peut-être un, ont degré d et que la différence de profondeur des feuilles n'excède pa un. Un arbre d-équilibré peut être représenté de la façon suivante :

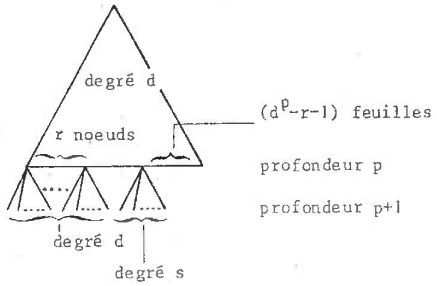
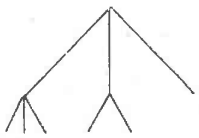


FIGURE 4

Par exemple, l'arbre 3-équilibré à 6 feuilles est :



$p = 1$
$r = 1$
$s = 2$

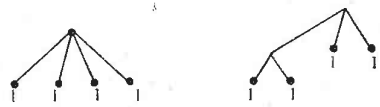
Un calcul simple montre que le coût $H_d(n)$ de l'arbre d-équilibré à n-feuilles de poids égaux à l'unité est :

$$H_d(n) = \varphi(d)\left[np + rd\right] + s\varphi(s) \tag{17}$$

1.4. DEFINITION ET PROPRIETES ELEMENTAIRES DES ARBRES OPTIMAUX

Définition 15 : Un arbre est dit optimal pour un n-uple $S = (a_1,\ldots,a_n)$ donné s'il minimise le coût des arbres ayant S pour suite de poids des feuilles. Le coût de l'arbre optimal est noté $C_\varphi(a_1,\ldots,a_n)$ ou $C_\varphi(S)$.

S'il n'y a pas ambiguité sur φ, nous noterons le coût optimal $C(a_1,\ldots,a_n)$ ou C(S). Il peut exister plusieurs arbres optimaux différents pour un même n-uple. Par exemple, si $\varphi(d) = d$, les deux arbres suivants sont optimaux pour le 4-uple (1,1,1,1)



Proposition 3 : Le coût optimal est "proportionnel" au n-uple. Plus précisément, si b est une constante réelle positive, on a :

$$C(b\,a_1,\ldots,b\,a_n) = b\,C(a_1,\ldots,a_n) \tag{18}$$

D'autre part, les arbres optimaux pour $(b\,a_1,\ldots,b\,a_n)$ se déduisen des arbres optimaux pour $(a_1,\ldots,a_n)$ en remplaçant le poids a_i de chaque feuille par $b.a_i$.

Preuve : Soit T un arbre optimal pour $(a_1,\ldots,a_n)$. Si nous remplaçons dans T le poids a_i de chaque feuille par $b.a_i$, nous obtenons un arbre T' pour $(b\,a_1,\ldots,b\,a_n)$ de coût :

$$\mathcal{C}(T') = b\,.\,\mathcal{C}(T) = b.C(a_1,\ldots,a_n)$$

L'arbre T' ayant $(b\,a_1,\ldots,b\,a_n)$ pour suite de poids des feuilles, nous avons :

$$\mathcal{C}(T') \geqslant C(b\,a_1,\ldots,b\,a_n)$$

On en déduit que :

$$b.C(a_1,\ldots,a_n) \geqslant C(b\,a_1,\ldots,b\,a_n)$$

Un raisonnement analogue (avec 1/b) montre l'inégalité contraire. On obtient donc bien l'égalité (18). Il en résulte que l'arbre T' est bien optimal pour $(b\ a_1,\ldots,b\ a_n)$.

1.4.1. HYPOTHESE DE CROISSANCE DE LA FONCTION DE COUT

Dans la pratique, il est évident que la fonction φ est croissante. Du point de vue théorique, si ce n'est pas le cas, on aboutit à une anomalie, comme le montre le lemme suivant :

Lemme 1 : (i) Si φ est croissante, on a :

$$C(a_1,\ldots,a_n) \leqslant C(a_1,\ldots,a_n,b) \qquad (b \geqslant 0)$$

(ii) Si φ n'est pas croissante, c'est-à-dire, s'il existe deux entiers d et d' vérifiant :

$d < d'$ et $\varphi(d) > \varphi(d')$

alors on a :

$$C(a_1,\ldots,a_n) > C(a_1,\ldots,a_n,\underbrace{\varepsilon,\ldots,\varepsilon}_{(d'-d)\text{ fois}})$$

chaque fois que l'arbre optimal correspondant au n-uple $(a_1,\ldots,a_n)$ a un noeud v de degré d et si ε vérifie :

$$\varepsilon < \frac{w(v)}{d'-d}\ \frac{\varphi(d)-\varphi(d')}{\varphi(d')} \qquad (19)$$

Preuve de (i) : Soit T' un arbre optimal correspondant à la suite $(a_1,\ldots,a_n,b)$. Considérons le noeud v qui est le père de la feuille de poids b. Si nous supprimons dans T' la feuille de poids v, le degré du noeud v passe de d à d-1. L'arbre T ainsi obtenu a donc pour coût :

$$\mathcal{C}(T) = \mathcal{C}(T') - \varphi(d).w(v) + \varphi(d-1).(w(v)-b)$$

Comme φ est croissante par hypothèse, on en déduit $\mathcal{C}(T) \leqslant \mathcal{C}(T')$. Comme par définition, on a $C(a_1,\ldots,a_n) \leqslant \mathcal{C}(T)$ et $\mathcal{C}(T') = C(a_1,\ldots,a_n)$ on en déduit bien :

$$C(a_1,\ldots,a_n) \leqslant C(a_1,\ldots,a_n,b)$$

Preuve de (ii) : Appelons T l'arbre optimal correspondant au n-uple $(a_1,\ldots,a_n)$. Complétons le noeud v de degré d par (d'-d) fils de poids ε. Le coût de l'arbre T' ainsi obtenu est égal à :

$$\mathcal{C}(T') = \mathcal{C}(T) - \varphi(d).w(v) + \varphi(d')\left[w(v)+(d'-d)\varepsilon\right]$$

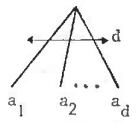
Comme par hypothèse $\varphi(d) > \varphi(d')$, on en déduit, d'après l'inégalité (19), que $\mathcal{C}(T') < \mathcal{C}(T)$. Comme par définition on a $C(a_1,\ldots,a_n,\varepsilon,\ldots,\varepsilon) \leqslant \mathcal{C}(T')$ et $\mathcal{C}(T) = C(a_1,\ldots,a_n)$, on en déduit bien que :

$$C(a_1,\ldots,a_n) \geqslant C(a_1,\ldots,a_n,\underbrace{\varepsilon,\ldots,\varepsilon}_{(d'-d)\text{ fois}})$$

Si φ n'est pas croissante, on a donc une anomalie : on peut améliorer le coût optimal correspondant à un n-uple en lui ajoutant des termes non nuls ! Pour éviter cette anomalie, la méthode la plus simple consiste à transformer la fonction φ en fonction croissante de la manière suivante. Chaque fois que l'on a :

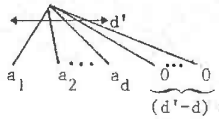
$$d < d' \quad \text{et} \quad \varphi(d) > \varphi(d')$$

on fait $\varphi(d) = \varphi(d')$. Un noeud de degré d sera alors représenté par :



coût : $\left(\sum_{1\leqslant i\leqslant d} a_i\right)\varphi(d')$

mais signifiera en réalité :



coût : $\left(\sum_{1\leqslant i\leqslant d} a_i\right)\varphi(d')$

Si nous appliquons cette transformation répétitivement, nous obtenons une fonction croissante, c'est-à-dire telle que :

$$d > d' \Longrightarrow \varphi(d) \geqslant \varphi(d').$$

Dans la suite, nous supposerons donc que la fonction φ est croissante. Cette hypothèse n'est pas essentielle à l'exposé, mais elle simplifie les démonstrations. Si la fonction φ n'est pas croissante, il faut modifier d'une constante les bornes supérieures des chapitres suivants, et la proposition 6.1.(ii) n'est plus valables.

1.4.2. TRANSFORMATIONS D'ARBRES CONSERVANT L'OPTIMALITE

Proposition 4 : (i) Toute contraction d'un arbre optimal est optimale.
(ii) Tout sous-arbre d'un arbre optimal est optimal.
(iii) Si dans un arbre optimal, on remplace un noeud v par un arbre équivalent à T(v), on obtient un arbre optim

Preuve : Considérons un arbre optimal T et un noeud v. D'après la proposition 1, le coût de T peut s'écrire :

$$\mathcal{C}(T) = \mathcal{C}(T/v) + \mathcal{C}(T(v))$$

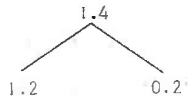
Cette formule montre que si l'on remplace T(v) par un arbre équivalent, l'arbre obtenu a même coût que T. Comme, d'autre part, sa suite de poids la même, l'arbre obtenu est équivalent à T. Il est donc optimal, ce qui démontre (iii).

Enfin , si T/v et T(v) n'étaient pas optimaux, on les remplacerait dans T par un arbre de coût inférieur, améliorant ainsi le coût $\mathcal{C}(T)$, ce qui contredit l'hypothèse d'optimalité de T.

Les contractions multiples sont aussi optimales.
Il est important de voir que la "réciproque" de ce lemme est fausse : si dans un arbre optimal, nous faisons l'expansion d'une feuille par un sous-arbre optimal, nous n'obtenons pas toujours un arbre optimal. Considérons par exemple, la fonction de coût $\varphi(d) = d$. Pour cette fonction de coût, les arbres T et T_1 suivants sont optimaux :

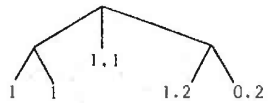
T =

T_1 =



Si dans T, nous faisons l'expansion de la feuille de poids 1.4 par T_1, nous obtenons l'arbre T' suivant qui n'est pas optimal :

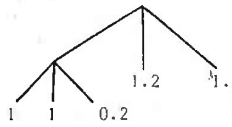
T' =



$\mathcal{C}(T') = 20.3$

En effet, pour la suite de poids (1,1,1.1,1.2,0.2), l'arbre optimal est l'arbre T" suivant :

T" =

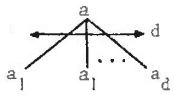


$\mathcal{C}(T") = 20.1$

Proposition 5 : L'arbre obtenu par équilibrage d'un arbre optimal est également optimal.

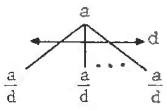
Preuve : Reprenons les notations de la définition 7 :

Noeud terminal initial



$w(v) = a = \sum_{1 \leq i \leq d} a_i$

Noeud terminal transformé



On peut supposer que $a_1 \leq a_2 \leq \ldots \leq a_d$. Appelons T l'arbre initial et S_T sa suite de poids. Appelons U' l'arbre transformé et S_U sa suite. D'après la proposition 2, on a :

$$\mathcal{C}(U') = \mathcal{C}(T) \tag{20}$$

Appelons U l'arbre optimal pour la suite S_U. Il est clair que :

$$\mathcal{C}(U) \leq \mathcal{C}(U') \tag{21}$$

Classons maintenant les d feuilles de poids a/d de l'arbre U par ordre décroissant de leurs coûts partiels (c'est-à-dire : $c(f_1) \geqslant c(f_2) \geqslant \ldots \geqslant c(f_d)$). Calculons le coût de U par la formule des coûts partiels :

$$\mathcal{C}_{(U)} = R + \frac{a}{d} \sum_{1 \leqslant i \leqslant d} c(f_i) \tag{22}$$

Dans cette formule, R désigne la sommation des coûts partiels pondérés sur toutes les feuilles sauf les feuilles f_i définies précédemment.

Construisons maintenant un arbre T' pour la suite initiale S_T de la manière suivante : on attribue à chaque feuille f_i de U le poids a_i. Comme T est optimal pour S_T, on a :

$$\mathcal{C}_{(T)} \leqslant \mathcal{C}_{(T')} \tag{23}$$

Comme précédemment, le coût de T' peut s'écrire :

$$\mathcal{C}_{(T')} = R + \sum_{1 \leqslant i \leqslant d} c(f_i) . a_i \tag{24}$$

D'après (22) et (24), la différence $\mathcal{C}_{(U)} - \mathcal{C}_{(T)}$ s'écrit :

$$\mathcal{C}_{(U)} - \mathcal{C}_{(T')} = \sum_{1 \leqslant i \leqslant d} c(f_i) \left(\frac{a}{d} - a_i \right) \tag{25}$$

Montrons que cette quantité est positive ou nulle. Les a_i étant croissant appelons k un indice tel que $a_k \leqslant \frac{a}{d}$ et $a_{k+1} \geqslant \frac{a}{d}$.
L'égalité (25) peut alors s'écrire :

$$\mathcal{C}_{(U)} - \mathcal{C}_{(T')} = \sum_{1 \leqslant i \leqslant k} c(f_i) \left(\frac{a}{d} - a_i \right) - \sum_{k+1 \leqslant i \leqslant d} c(f_i) \left(a_i - \right. \tag{26}$$

Les coûts partiels $c(f_i)$ étant décroissants avec i, on en déduit :

$$\mathcal{C}_{(U)} - \mathcal{C}_{(T')} \geqslant \left[c(f_k) . \sum_{1 \leqslant i \leqslant d} \left(\frac{a}{d} - a_i \right) \right] - \left[c(f_k) . \sum_{k+1 \leqslant i \leqslant d} \left(a_i - \right. \right. \tag{27}$$

Le second terme étant nul, on a bien :

$$\mathcal{C}_{(U)} \geqslant \mathcal{C}_{(T')} \tag{28}$$

Finalement, en combinant (20), (23) et (28), on obtient :

$$\mathcal{C}_{(U')} \leqslant \mathcal{C}_{(U)}.$$

L'inégalité (21) étant dans l'autre sens, on en déduit :

$$\mathcal{C}_{(U')} = \mathcal{C}_{(U)}.$$

L'arbre transformé U' est donc optimal. ▨

Bien entendu, la réciproque (avec opération de déséquilibrage) est fausse.

1.4.3. RELATION ENTRE POIDS ET COUTS PARTIELS DANS LES ARBRES OPTIMAUX

Dans un arbre optimal, il existe une relation simple entre les poids et les coûts partiels des noeuds et des feuilles. Cette relation nous permettra de placer les poids d'une suite S donnée sur les feuilles d'une arborescence A donnée de manière à minimiser le coût.

Proposition 6 : Pour toute paire de noeuds ou de feuilles v_1 et v_2 d'un arbre optimal, on a l'inégalité :

$$\left[c(v_1) - c(v_2) \right] . \left[w(v_1) - w(v_2) \right] \leqslant 0 \tag{29}$$

Preuve : Si l'un des deux est ascendant de l'autre, l'inégalité est évidente. Supposons donc v_1 et v_2 indépendants et contractons si nécessaire v_1 et v_2. D'après la proposition précédente, l'arbre U ainsi obtenu est également optimal. Calculons son coût par la méthode des coûts partiels :

$$\mathcal{C}_{(U)} = R + c(v_1) . w(v_1) + c(v_2) . w(v_2) \tag{30}$$

Dans cette formule, R désigne la sommation des coûts partiels pondérés sur toutes les feuilles sauf v_1 et v_2. Considérons l'arbre U' obtenu en échangeant les poids de v_1 et v_2.

Son coût est :

$$\mathcal{C}(U') = R + c(v_1).w(v_2) + c(v_2).w(v_1)$$

L'arbre U étant optimal, on a $\mathcal{C}(U) \leqslant \mathcal{C}(U')$. On en déduit l'inégalité (

▨

Corollaire 1 : Pour toute paire de noeuds ou de feuilles v_1 et v_2 d'un arbre optimal, on a les implications :

(31) $c(v_1) < c(v_2) \Longrightarrow w(v_1) \geqslant w(v_2)$

(32) $w(v_1) < w(v_2) \Longrightarrow c(v_1) \geqslant c(v_2)$

Corollaire 2 : Dans un arbre optimal, les poids des feuilles sont dans l'ordre inverse de leurs coûts partiels.

Corollaire 3 : Soient une arborescence A à n feuilles et une suite $S = (a_1,\ldots,a_n)$. L'arbre T de coût minimum ayant pour arborescence A et pour suite S est obtenu en plaçant les poids de S sur les feuilles de A, dans l'ordre inverse de leurs coûts partiels. Le coût de T est noté :

(33) $\mathcal{C}_A(S)$ ou $\mathcal{C}_A(a_1,\ldots,a_n)$

Preuve : Par l'absurde : s'il existait deux feuilles f_1 et f_2 telles que

$$w(f_1) > w(f_2) \quad \text{et} \quad c(f_1) > c(f_2),$$

on pourrait alors améliorer strictement le coût en échangeant les poids de f_1 et f_2 (cf. démonstration de la proposition 5).

▨

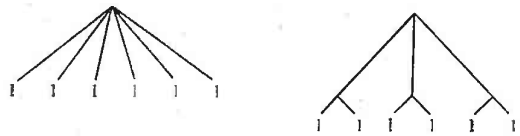
1.5. FONCTIONS DE COUT A DEGRE BORNE

Pour les fonctions de coût rencontrées dans la pratique, les degrés des noeuds des arbres optimaux ne sont pas arbitrairement grands (c'est le cas pour les fonctions de coût du tri sur disque). Nous montrons ici que les arbres optimaux à poids égaux ont leurs degrés bornés, alors c'est vrai pour les arbres optimaux correspondants à un n-uple quelconque. Nous donnons ensuite des conditions suffisantes pour que la fonction de coût soit à degré borné.

1.5.1. DEFINITION ET PROPRIETES

Définition 16 : Une fonction de coût est dite à degré borné par un entier d_o si pour tout n-uple, il existe un arbre optimal dont tous les noeuds ont un degré inférieur ou égal à d_o.
On appelle degré maximum le plus petit entier d_{max} ayant cette propriété.

Le fait que la fonction de coût possède un d_{max} n'entraîne pas que tous les arbres optimaux ont un degré inférieur ou égal à d_{max}. Il peut exister des arbres équivalents de degré supérieur. Par exemple, si $\varphi(d) = d+1$, on peut montrer que $d_{max} = 5$. Cependant, les deux arbres suivants sont optimaux, pour le 6-uple (1,1,1,1,1,1) et ont pour coût 4.2 :



Proposition 7 : Si les deux conditions suivantes sont réalisées :
1) pour tout n-uple, il existe un arbre optimal de degré au sommet inférieur ou égal à d_{max}
2) pour certains n-uple, il n'existe pas d'arbres optimaux de degré au sommet inférieur à d_{max}

alors la fonction de coût a pour degré maximum d_{max}.

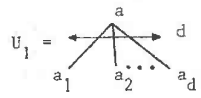
Preuve : Considérons un arbre optimal T quelconque pour un n-uple donné. Nous allons construire un arbre équivalent dont tous les noeuds ont un degré inférieur ou égal à d_{max}. Si le degré du sommet de T est supérieur à d_{max}, nous remplaçons T par un arbre équivalent de degré au sommet plus petit ou égal à d_{max} (c'est possible d'après la condition 1). Dans l'arbre ainsi obtenu, remplaçons si nécessaire les noeuds situés à la profondeur 1 par des arbres équivalents de degrés plus petits ou égaux à d_{max} (toujours d'après la condition 1). D'après la proposition 4, l'arbre résultant est optimal.

Sur ce nouvel arbre, on fait les mêmes opérations sur tous les noeuds situés à la profondeur 2, et ainsi de suite, jusqu'à ce qu'on obtienne un arbre optimal dont tous les noeuds ont un degré inférieur ou égal à d_{max}. La condition 2 assure que d_{max} est le plus petit entier possédant cette propriété.

Proposition 8 : Si la fonction de coût possède un degré maximum d_{max} pour tous les arbres à poids égaux, alors elle possède le même degré maximum d_{max} dans le cas général.

Preuve : Si pour tout n-uple, nous construisons un arbre optimal de degré au sommet plus petit ou égal à d_{max}, la condition 1 de la proposition 7 sera satisfaite. La condition 2 est satisfaite puisque, par hypothèse, pour certains n-uples à poids égaux, il n'existe pas d'arbres optimaux de degré au sommet inférieur à d_{max}. D'après la proposition précédente, la fonction de coût aura donc pour degré maximum d_{max}.

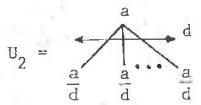
Soient donc un n-uple quelconque et un arbre optimal T correspondant. Si T a un degré au sommet d supérieur à d_{max}, nous allons construire un arbre équivalent de degré au sommet inférieur ou égal à d_{max}. Posons $a = w(T)$; appelons $T_1,\ldots,T_d$ les sous-arbres principaux de T et $a_1,\ldots,a_d$ leurs poids ; appelons U_1 l'arbre plat sommital de T :



D'après la proposition 1, le coût de T s'écrit :

$$\mathcal{C}(T) = \mathcal{C}(U_1) + \sum_{1 \leq i \leq d} \mathcal{C}(T_i)$$

D'après la proposition 5, l'arbre U_2 suivant obtenu par équilibrage de U_1, est également optimal :



Il est clair que :

$$\mathcal{C}(U_1) = \mathcal{C}(U_2) \tag{34}$$

L'arbre U_2 est un arbre optimal à poids égaux de degré au sommet d supérieur à d_{max}. Par hypothèse, il existe donc un arbre équivalent U_3 à d feuilles, de degré au sommet d' plus petit ou égal à d_{max}. On a donc :

$$\mathcal{C}(U_2) = \mathcal{C}(U_3) \tag{35}$$

Plaçons donc les poids $(a_1,\ldots,a_d)$ sur l'arborescence de U_3 dans l'ordre inverse des coûts partiels. Nous obtenons ainsi un arbre U_4. Par une méthode identique à celle de la proposition 5 (démonstration de l'inégalité (28)), on montre que :

$$\mathcal{C}(U_4) \leq \mathcal{C}(U_3) \tag{36}$$

La combinaison des formules (34), (35) et (36) montre que $\mathcal{C}(U_4) \leq \mathcal{C}(U_1)$. Mais les arbres U_4 et U_1 ont la même suite de poids $(a_1,\ldots,a_d)$. Comme U_1 est optimal, on en déduit que U_4 est également optimal. Mais le degré au sommet de U_4 est d', inférieur ou égal à d_{max}. Il suffit donc dans T de remplacer U_1 par U_4 (c'est-à-dire de faire dans U_4 l'expansion des feuilles de poids $a_1,\ldots,a_d$ par les sous-arbres $T_1,\ldots,T_d$) pour obtenir un arbre T' équivalent T et de degré au sommet plus petit ou égal à d_{max}. ▨

1.5.2. CONDITIONS SUFFISANTES POUR QU'UNE FONCTION DE COUT SOIT A DEGRE BORNE

Nous avons vu qu'il suffit de se limiter aux arbres optimaux à poids égaux. En fait, on peut se limiter aux arbres à poids égaux et unitaires. Nous allons donner une condition pour que les arbres de ce type ne soient pas optimaux si leurs degrés sont trop grands. A cette fin, il suffi de trouver une condition pour que les coûts des arbres de degrés trop grands soient supérieurs aux coûts de certains arbres, les arbres d-équilibrés. Nou allons donc commencer par étudier le coût des arbres d-équilibrés à poids égaux et unitaires.

Définition 17 : Nous définissons deux fonctions f_d et h_d sur le domaine réel $[1,\infty[$ de la manière suivante :

(37) (i) $f_d(x) = \frac{\varphi(d)}{\text{Log } d} \times \text{Log } x$

(ii) h_d est la fonction constituée par les segments de droite joignant les paires de points du plan $\left[d^p, f_d(d^p)\right]$ et $\left[d^{p+1}, f_d(d^{p+1})\right]$ pour p entier positif ou nul. Plus précisément :

(38) $\left\{ \right.$ pour $d^p \leqslant x \leqslant d^{p+1}$

(39) $h_d(x) = \varphi(d).\left[p\, d^p + (x-d^p)\left(p + \frac{d}{d-1}\right)\right]$

Lemme 2 : Le coût $H_d(n)$ d'un arbre d-équilibré à poids égaux et unitaires vérifie :

(40) (i) $H_d(n) < f_d(n) + \varphi(d).n$

(41) (ii) $H_d(n) \leqslant h_d(n) + K(d)$

où K(d) est le réel positif ou nul défini par :

(42) $K(d) = \underset{2<s\leqslant d}{\text{Max}} \left[(s-1)\left(\varphi(s)\,\frac{s}{s-1} - \varphi(d)\,\frac{d}{d-1}\right)\right]$

Preuve de (i) : Pour les arbres d-équilibrés, nous utilisons les notations de la définition 14 :

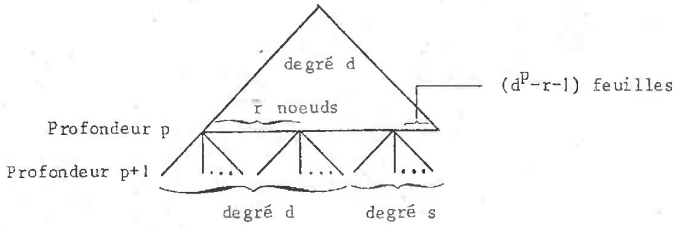


FIGURE 5

Si n est le nombre de feuilles, les entiers p, r et s sont déterminés par :

(43) $\left\{ \right.$ $d^p < n \leqslant d^{p+1}$

(44) $n-d^p-1 = (d-1).r + (s-2) \qquad 2 \leqslant s \leqslant d$

Majorons le coût $H_d(n)$ en utilisant la formule des coûts partiels. Les s feuilles correspondant au noeud terminal de degré s ont pour coût partiel $\left[p\,\varphi(d) + \varphi(s)\right]$. Les autres feuilles sont à la profondeur p ou p+1 ; leurs coûts partiels sont donc inférieurs ou égaux à $\left[(p+1)\,\varphi(d)\right]$. Le coût $H_d(n)$ est donc majoré par :

$$H_d(n) \leqslant s\left[p\,\varphi(d) + \varphi(s)\right] + (n-s)\,(p+1)\,\varphi(d)$$

ce qui s'écrit :

(45) $$H_d(n) \leqslant (p+1)\,\varphi(d)\,n + s\left[\varphi(s) - \varphi(d)\right]$$

Or, d'après l'inégalité (35), p est borné par :

$$p < \frac{\text{Log } n}{\text{Log } d}$$

En substituant dans (45), on obtient :

$$H_d(n) < \frac{\varphi(d)}{\text{Log } d}\, n \text{ Log } n + \varphi(d)\, n + s\left(\varphi(s)-\varphi(d)\right)$$

On en déduit l'inégalité (40), car φ est croissante.

Preuve de (ii) : D'après les formules (17) et (39), la différence $H_d - h_d$ s'écrit :

(46) $$H_d(n) - h_d(n) = s\,\varphi(s) - \left(\frac{s-1}{d-1}\right) d\,\varphi(d)$$

On en déduit immédiatement la formule (41).

▨

La deuxième majoration est évidemment plus fine que la première. Cependant, cette dernière a l'avantage d'être simple à formuler et sera utilisée le plus souvent dans la suite.

Remarquons que si la fonction $\varphi(d).d/(d-1)$ est croissante, on a :

$$K(d) = 0$$

C'est le cas pour les fonctions de coût du type tri sur disque si $\beta/\alpha < 1$.

Théorème 1 : Si la fonction φ satisfait l'une des conditions suivantes, alors la fonction de coût possède un degré maximum d_{max} inférieur à d_o :

(47) (i) $\forall n \geq d_o \qquad \varphi(n) > \frac{\varphi(d_1)}{\log d_1} \text{ Log } n + \varphi(d_1)$

(48) (ii) $\forall n \geq d_o \qquad n.\varphi(n) > h_{d_1}(n) + K(d_1)$

(d_o et d_1 sont des entiers plus grands ou égaux à deux)
De plus, tous les noeuds des arbres optimaux ont un degré inférieur à d_o.

Preuve : D'après le lemme précédent, l'une ou l'autre de ces deux conditions entraîne que :

(49) $\forall n \geq d_o \qquad n\varphi(n) > H_{d_1}(n)$

Montrons donc que les noeuds des arbres optimaux ont un degré inférieur à d_o, ce qui entraînera le degré maximum

Supposons qu'il existe un arbre optimal dont un noeud a un degré d supérieur ou égal à d_o. Considérons l'arbre engendré par ce noeud.

D'après la proposition 4, cet arbre est optimal. Toujours d'après cette même proposition, son arbre plat sommital est également optimal. D'après la proposition 5, l'arbre P obtenu par équilibrage de cet arbre plat est aussi optimal. Son coût est donc inférieur ou égal au coût de l'arbre d-équilibré à d feuilles de poids w(P)/d, ce qui s'écrit :

$$\varphi(d).w(P) \leq \frac{w(P)}{d} \cdot H_{d_1}(d)$$

ce qui entraîne :

$$d\varphi(d) \leq H_{d_1}(d)$$

Cette dernière inégalité contredit (49), puisque $d > d_o$. ▨

Corollaire 4 : Si la différence entre $\varphi(n)/\text{Log } n$ et son minimum est plus grande qu'une constante à partir d'un certain rang, alors la fonction de coût est à degré borné. Cette condition s'exprime de la manière suivante :

(50) $(\exists \varepsilon>0)(\exists N) \ (\forall_{n \geq N}) \qquad \frac{\varphi(n)}{\text{Log } n} \geq \varepsilon + \underset{d \geq 2}{\text{Min}} \frac{\varphi(d)}{\text{Log } n}$

Preuve : Il nous suffit de montrer que cette condition entraîne la proposition (i) du théorème 1. Cette dernière peut s'écrire :

(51) $(\exists d_1)(\exists d_o) \ \forall n \geq d_o \qquad \frac{\varphi(n)}{\text{Log } n} > \frac{\varphi(d_1)}{\text{Log } d_1} + \frac{\varphi(d_1)}{\text{Log } n}$

Nous allons donc calculer d_1 et d_o en fonction de ε et N. D'après (50), la fonction $\varphi(n)/\text{Log } n$ a un minimum pour un entier m plus petit que N. Posons :

(52) $d_1 = m$

(53) $d_o = \text{Max}\left(N, \left\lfloor e^{\varphi(d_1)/\varepsilon} \right\rfloor + 1\right)$

Comme $d_o \geq N$, la proposition (50) entraîne :

$$(54) \qquad \forall n \geq d_o \qquad \frac{\varphi(n)}{\text{Log } n} \geq \varepsilon + \frac{\varphi(d_1)}{\text{Log } d_1}$$

Or, d'après (53), on a :

$$\varepsilon > \frac{\varphi(d_1)}{\text{Log } d_o}$$

En combinant cette inégalité et l'inégalité (54), on obtient :

$$\forall n \geq d_o \qquad \frac{\varphi(n)}{\text{Log } n} > \frac{\varphi(d_1)}{\text{Log } d_1} + \frac{\varphi(d_1)}{\text{Log } d_o}$$

Comme $n \geq d_o$, cette dernière inégalité entraîne bien l'inégalité (51). ▨

Corollaire 5 : Si la fonction $\varphi(n)/\text{Log } n$ est croissante et non bornée à partir d'un certain rang, alors la fonction de coût est à degré borné.

Preuve : Il est évident que cette proposition entraîne le corollaire précédent. ▨

1.6. EXEMPLE DU TRI SUR DISQUE

Nous avons vu dans l'introduction que pour une modélisation simple du tri sur disque, les fonctions de coûts sont du type :

$$(55) \qquad \varphi(d) = \alpha\, d + \beta$$

où α et β sont des réels positifs ou nuls. Ces fonctions de coût nous serviront d'exemple dans la première partie. En fait, nous nous limiterons aux fonctions du type :

$$(56) \qquad \varphi(d) = d + \lambda$$

où λ est un réel positif ou nul.

En effet, il est facile de vérifier que la multiplication par une constante α d'une fonction φ de ce type ne change pas la forme des arbres optimaux. Seul le coût est multiplié par α. Plus précisément :

$$(57) \qquad \begin{cases} \mathcal{C}_{\alpha\varphi}(T) = \alpha . \mathcal{C}_{\varphi}(T) \\ C_{\alpha\varphi}(S) = \alpha . C_{\varphi}(S) \end{cases}$$

D'après le corollaire 5, les fonctions φ de ce type possèdent un degré maximum. En effet, la fonction $(n+\lambda)/\text{Log } n$ possède un (parfois deux) minimum entier et elle est croissante ensuite.

CHAPITRE 2

ARBRES OPTIMAUX A POIDS EGAUX

Dans ce chapitre, ainsi que dans les trois chapitres suivants, nous étudions les arbres optimaux à poids égaux. Il s'agit d'une part de mettre en évidence les propriétés des arbres optimaux et de leur coût, et d'autre part d'étudier la complexité des algorithmes de recherche des arbres optimaux. Nous montrerons au chapitre 4 que, pour la plupart des fonctions de coût, il existe un algorithme de recherche en temps constant.

Dans la suite, nous supposerons que les poids sont égaux à l'unité. D'après la proposition 1.3, cela n'est pas une restriction : l'arbre optimal pour le n-uple $(1,1,\ldots,1)$ est également optimal pour le n-uple $(b,b,\ldots,b)$, le coût étant proportionnel à b. Le coût optimal dépend alors uniquement de n. Nous le noterons C(n). Par ailleurs, les poids des feuilles étant toujours l'unité, les arbres seront entièrement décrits par la forme de leurs arborescences ; nous omettrons donc d'indiquer sur les figures le poids des feuilles.

Nous présentons d'abord une récurrence permettant de calculer C(n), coût des arbres optimaux à n feuilles de poids unitaire. Un algorithme de calcul de ces arbres et de leurs coûts est proposé, algorithme dont nous étudions la complexité en temps et en espace mémoire.

Au deuxième paragraphe, nous étudions la forme asymptotique du coût optimal C(n), ce qui nous donnera, au paragraphe 3, des indications sur la forme des arbres optimaux. Nous appliquons ensuite ces résultats à une certaine classe de fonctions de coût.

2.1. RECURRENCE ET ALGORITHMES DE CALCUL DES ARBRES OPTIMAUX ET DE LEURS COUTS

2.1.1. RECURRENCE DE CALCUL DU COUT OPTIMAL C(n)

Nous cherchons les arbres optimaux à n feuilles, ainsi que leur coût. L'arbre à une feuille est réduit à un point et son coût est nul par définition : C(1) = 0. Supposons que l'on connaisse les arbres optimaux à 1,2,3,...,n-1 feuilles, ainsi que leurs coûts.

Pour calculer l'arbre optimal à n feuilles, la méthode la plus simple consiste à examiner tous les arbres à n feuilles et à choisir celui de coût minimum. D'après la proposition 4, on peut limiter cette recherche aux arbres dont les sous-arbres principaux sont eux-mêmes optimaux. Soit T un tel arbre. Appelons d son degré au sommet, $(T_1,\ldots,T_d)$ ses sous-arbres principaux et $(n_1,\ldots,n_d)$ leurs poids. D'après la proposition 1.1, le coût de cet arbre est égal à :

$$\mathcal{C}(T) = \varphi(d)\, n + \sum_{1\leqslant i\leqslant d} \mathcal{C}(T_i)$$

Les arbres T_i étant optimaux, on a $\mathcal{C}(T_i) = C(n_i)$; d'où :

$$\mathcal{C}(T) = \varphi(d)\, n + \sum_{1\leqslant i\leqslant d} C(n_i)$$

Il s'agit donc de chercher l'arbre qui minimise $\mathcal{C}(T)$ pour tous les degrés d possible et pour toutes les partitions de n en $(n_1,\ldots,n_d)$. Le coût C(n) est donc solution de la récurrence suivante :

(1)
$$\boxed{\begin{cases} C(1) = 0 \\ C(n) = \underset{2\leqslant d\leqslant n}{\text{Min}} \left[\varphi(d)\, n + \underset{\sum_{1\leqslant i\leqslant d} n_i = n}{\text{Min}} \left(\sum_{1\leqslant i\leqslant d} C(n_i) \right) \right] \end{cases}}$$

où d et n_i sont des entiers positifs.

La fonction C(n) n'est définie que sur les entiers. On peut l'étendre, par interpolation linéaire, aux réels de $[1,\infty[$. Plus précisément, entre deux entiers n et n+1 la fonction C est le segment de droite joignant les points du plan $[n,C(n)]$ et $[n+1,C(n+1)]$.

On montre facilement que la fonction C(n) ainsi définie vérifie :

$$(2)\qquad \begin{cases} C(1) = 0 \\ C(x) \leqslant \min\limits_{d\geqslant 2}\left[\varphi(d).\,x + \min\limits_{\sum_{1\leqslant i\leqslant d} x_i = x}\left(\sum_{1\leqslant i\leqslant d} C(x_i)\right)\right] \end{cases}$$

où x et x_i sont des réels plus grands ou égaux à 1. Notons que pour un réel x donné on ne trouve généralement pas d'entier d et de réels x_i qui assurent l'égalité.

2.1.2. ALGORITHME DE CALCUL DES ARBRES OPTIMAUX ET DE LEURS COUTS

Les méthodes de programmation dynamique s'appliquent bien à la résolution de la récurrence précédente. En effet, définissons le tableau A(d,n) suivant :

$$(3)\qquad \begin{cases} A(d,n) = \min\limits_{\sum_{1\leqslant i\leqslant d} n_i = n}\left[\sum_{1\leqslant i\leqslant d} C(n_i)\right] & \text{pour } 2\leqslant d\leqslant n \\ A(1,n) = C(n) \end{cases}$$

En clair, A(d,n) minimise la somme des coûts de d arbres dont la somme des poids est n. Connaissant A(d,n) pour $2\leqslant d\leqslant n$, il est clair que le coût optimal C(n) s'écrit :

$$(4)\qquad C(n) = A(1,n) = \min_{2\leqslant d\leqslant n}\left[\varphi(d).\,n + A(d,n)\right]$$

Pour calculer A(d,n), il faut examiner tous les groupes de d arbres dont la somme des poids est n et choisir le groupe de coût minimum. On peut limiter cette recherche aux groupes dont les arbres constitutifs et tous les sous-groupes sont eux-mêmes optimaux. Considérons un tel groupe et isolons un de ses arbres de poids i. Cet arbre étant optimal, son coût est égal à C(i) = A(1,i). De même, la somme des coûts des (d-1) arbres restants est A(d-1,n-i).

La somme des coûts des arbres du groupe est donc :

$$(5)\qquad A(1,i) + A(d-1,n-i).$$

Pour obtenir A(d,n) il suffit de minimiser cette quantité quand i varie :

$$(6)\qquad A(d,n) = \min_{1\leqslant i\leqslant n/d}\left[A(1,i) + A(d-1,n-i)\right]$$

(par symétrie, on peut faire varier i de 1 à $\lfloor n/d \rfloor$ seulement).

L'algorithme suivant calcule C(N) = A(1,N) en utilisant les formules (4) et (6). Dans un souci de clarté, cet algorithme, ainsi que ceux que nous serons amenés à décrire par la suite, est écrit dans un ALGOL très approximatif. Nous nous permettons par exemple d'écrire $\min_{2\leqslant d\leqslant n}$. En tête de chaque algorithme, nous indiquerons la dimension des tableaux nécessaires.

```
   tableau A(N,N) ;
I1 pour n:=2 pas 1 jusqu'à N faire
   début
I2     pour d: = 2 pas 1 jusqu'à n faire
       début
I3         A(d,n):= Min [A(1,i) + A(d-1,n-i)]
                    1≤i≤n/d
       fin ;
I4     A(1,n):= Min [φ(d).n + A(d,n)]
                2≤d≤n
   fin
```

Cet algorithme calcule uniquement le coût optimal. Pour obtenir en même temps la forme des arbres optimaux, il suffit de garder les entiers i et d qui minimisent respectivement A(d,n) dans l'instruction I3 et A(1,n) dans l'instruction I4. A partir de ces informations, on reconstitue aisément la forme des arbres optimaux.

2.1.3. COMPLEXITE DE L'ALGORITHME

Evaluons la complexité de cet algorithme. Il utilise un espace mémoire de taille N^2 pour le tableau A. Evaluons le temps pris par cet algorithme, c'est-à-dire le nombre d'opérations nécessaires à l'exécution de l'instruction I1. Pour cela, nous comptons le nombre d'opérations en commençant par la boucle intérieure.

Instruction I3 : il s'agit de calculer un minimum parmi $\lfloor n/d \rfloor$ valeurs ; cette instruction nécessite donc de l'ordre de n/d opérations.

Instruction I2 : pour chaque d ($2 \leqslant d \leqslant n$), il faut exécuter l'instruction I3 ; le nombre d'opérations nécessaires à l'exécution de I2 est donc de l'ordre de :

$$\sum_{2 \leqslant d \leqslant n} \frac{n}{d} = n\left(\frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}\right)$$

On reconnaît la série harmonique. L'instruction I2 nécessite donc de l'ordre de n Log n opérations.

Instruction I1 : c'est une boucle pour. Pour chaque d, il faut effectuer successivement I2 et I4. Comme pour I3, il faut de l'ordre de n opérations pour exécuter I4. On peut négliger ce terme devant n Log n, nombre d'opérations nécessaires pour effectuer I2. Finalement, le nombre d'opérations nécessaires à l'exécution de I1 est donc :

$$\sum_{2 \leqslant n \leqslant N} n \text{ Log } n$$

c'est-à-dire de l'ordre de N^2. Log N opérations.

Nous dirons que l'algorithme est en espace mémoire $O(N^2)$ et en temps $O(N^2 . \text{Log } N)$. Plus précisément, cela signifie que le temps $\mathcal{C}(N)$ pris par cet algorithme pour calculer la forme et le coût des arbres optimaux à N feuilles est borné de la manière suivante :

$$c_1 N^2 \text{ Log } N \leqslant \mathcal{C}(N) \leqslant c_2 N^2 \text{ Log } N$$

où c_1 et c_2 sont des constantes réelles positives dépendant de l'implémentation de l'algorithme et des performances de l'ordinateur utilisé.

Pour évaluer c_1 et c_2, il faut évaluer avec précision le temps pris par chaque instruction et négliger les termes de second ordre, comme nous l'avons fait en négligeant le temps pris par l'instruction I4 devant celui pris par I2.
Pourquoi prendre cette mesure pour évaluer la complexité d'un algorithme ? La raison est la suivante : quelles que soient l'implémentation de l'algorithme et les performances de l'ordinateur, un algorithme en temps $O(N^2)$ est toujours plus rapide qu'un algorithme en $O(N^3)$ à partir d'un certain rang. C'est pourquoi nous nous attacherons dans la suite à trouver des algorithmes dont la mesure de complexité est la plus petite possible.

2.1.4. CAS DES FONCTIONS DE COUT A DEGRE BORNE

Dans ce cas, quel que soit n, il existe un arbre optimal dont tous les noeuds ont un degré inférieur ou égal à d_{max}. On peut donc limiter la recherche aux arbres de degré inférieur ou égal à d_{max}. Nous n'obtiendrons pas tous les arbres optimaux par cette méthode puisqu'il peut exister des arbres optimaux de degré supérieur à d_{max}. Mais le plus souvent, un seul suffit !

En modifiant l'algorithme précédent, on obtient l'algorithme :

```
                tableau A(d_max,N) ;
I1              pour  n:= 2 pas 1 jusqu'à N faire
                début
                    D:= min (n,d_max) ;
I2                  pour  d: = 2 pas 1 jusqu'à D faire
                    début
I3                      A(d,n) = Min  [A(1,i) + A(d-1,n-i)]
                                1≤i≤n/d
                    fin;
                    A(1,n): = Min  [φ(d).n + A(d,n)]
I4                            2≤d≤D
                fin
```

Evaluons la complexité de cet algorithme. Il utilise un tableau de dimension $N \geq d_{max}$. L'algorithme est donc de complexité O(N) en espace mémoire, ce qui constitue une amélioration considérable par rapport au précédent algorithme en $O(N^2)$. En effet, un algorithme utilisant un espace mémoire de dimension N^2 est pratiquement inutilisable pour les grandes valeurs de N.

En ce qui concerne le temps, nous l'analysons comme pour l'algorithme précédent :

Instruction I3 : n/d opérations

Instruction I2 : si n est plus grand que d_{max}, le nombre d'opérations nécessaire est :

$$\frac{n}{2} + \ldots + \frac{n}{d_{max}}$$

c'est-à-dire O(n) opérations

Instruction I1 : le nombre d'opérations nécessaire est de l'ordre de

$$\sum_{2 \leq n \leq N} n$$

c'est-à-dire de l'ordre de N^2 opérations

Finalement, si la fonction de coût est à degré borné, l'algorithme précédent permet de calculer les arbres optimaux en temps $O(N^2)$ et en espace mémoire O(N).

2.2. BORNE INFERIEURE ET SUPERIEURE DU COUT OPTIMAL C(n)

Nous nous proposons de donner une évaluation approximative du coût optimal qui est calculable rapidement. Dans le cas du tri sur disque, cela nous permettra d'évaluer rapidement les performances d'une configuration en fonction des caractéristiques du matériel.

Cela nous permettra aussi de comparer des configurations différentes sans être obligés à chaque fois d'utiliser les algorithmes précédents.

Il y a une autre motivation plus théorique. Nous verrons que les bornes inférieures et supérieures donnent des indications sur la forme des arbres optimaux. Ces indications nous permettrons dans la suite, de limiter de plus en plus la recherche des arbres optimaux à certains types d'arbres, et par là d'améliorer les algorithmes de recherche. Plus les bornes seront fines, plus les indications sur la forme des arbres optimaux deviendront précises, ce qui nous permettra d'améliorer les algorithmes. Les bornes suivantes constituent donc une première approximation et donnent une idée sur la forme des arbres et la forme asymptotique de C(n).

Pour la définition des bornes inférieures et supérieures, l'étude de la fonction $\varphi(n)/\text{Log } n$ joue un grand rôle. Nous avons déjà rencontré cette fonction au chapitre précédent (corollaires 1.4 et 1.5), à l'occasion de l'étude des fonctions de coût à degré borné. En fait, nous verrons plus loin que si la fonction φ croît "moins vite" que la fonction logarithme, alors les arbres optimaux sont les arbres plats. Pour l'instant, nous avons besoin des définitions suivantes concernant les minimums de la fonction $\varphi(n)/\text{Log } n$.

Définition 1 : La fonction $k_\varphi(n)$ des entiers positifs dans les réels positifs est définie par :

(7)
$$\begin{cases} k_\varphi(1) = \varphi(2) \,/\, \text{Log } 2 \\ k_\varphi(n) = \underset{2 \leq d \leq n}{\text{Min}} \dfrac{\varphi(d)}{\text{Log } d} \qquad \text{pour } n \geqslant 2 \end{cases}$$

On appelle "degré-limites pour n" les entiers minimisant la fonction $\varphi(d)/\log d$ sur l'intervalle $[2,n]$. On note $\mathcal{D}_\varphi(n)$ l'ensemble des degré-limites pour n.

L'ensemble $\mathcal{D}_\varphi(n)$ est donc défini par :

$$\mathcal{D}_\varphi(n) = \left\{ d \;\middle|\; (d \in N)_\wedge (2 \leq d \leq n)_\wedge \left(\frac{\varphi(d)}{\text{Log } d} = k_\varphi(n) \right) \right\}$$

La fonction $k_\varphi(n)$ est positive décroissante. Elle a donc une limite positive ou nulle.

Définition 2 : Nous définissons la constante réelle positive ou nulle k_φ par :

(8) $$k_\varphi = \lim_{n\to\infty} k_\varphi(n)$$

En d'autres termes, k_φ est la borne inférieure de l'ensemble dénombrable des réels $\varphi(n)/\text{Log } n$ pour $n \geqslant 2$. Nous verrons plus loin que les propriétés des arbres optimaux sont assez différentes selon que cette borne inférieure est atteinte ou non dans cet ensemble :

Définition 3 : On dit qu'une fonction de coût possède un degré-limite s'il existe un ou plusieurs entiers minimisant la fonction $\varphi(n)/\text{Log } n$ sur l'intervalle $[2,\infty[$. On appelle degré-Limites entiers. On note "a" le plus petit d'entre eux et $\mathcal{D}_\varphi$ leur ensemble.

D'après la définition 2, si la fonction de coût possède un degré-limite, on a :

(9) $$k_\varphi = \frac{\varphi(a)}{\text{Log } a} = \frac{\varphi(d)}{\text{Log } d} \qquad \text{avec } d \in \mathcal{D}_\varphi$$

Dans la suite, quand il n'y aura pas ambiguïté sur la fonction φ, nous omettrons les indices φ dans $k_\varphi(n)$, k_φ, $\mathcal{D}_\varphi(n)$ et $\mathcal{D}_\varphi$.
Nous pouvons maintenant mettre en évidence les bornes inférieures et supérieures suivantes :

Théorème 2 : Le coût optimal C(n) vérifie :

(10) $$k\, n \text{ Log } n \leqslant k(n)\, n \text{ Log } n \leqslant C(n) < \frac{\varphi(d)}{\text{Log } d}\, n \text{ Log } n + \varphi(d)\, n$$

(d est un entier quelconque plus grand ou égal à deux).

Preuve : Nous avons montré au lemme 1.1 que le coût $H_d(n)$ des arbres d-équilibrés à n feuilles de poids unitaire est majoré par :

$$H_d(n) < \frac{\varphi(d)}{\text{Log } d}\, n \text{ Log } n + \varphi(d).n$$

Comme par définition, on a : $C(n) \leqslant H_d(n)$, on en déduit la majoration.
La borne inférieure se montre par récurrence. Pour n = 1, c'est évident.
Considérons un arbre optimal de poids n. Appelons d son degré et $n_1,\ldots,n_d$ les poids de ses sous-arbres principaux. Le coût optimal peut s'écrire :

$$C(n) = \varphi(d)\, n + \sum_{1\leqslant i\leqslant d} C(n_i)$$

D'après l'hypothèse de récurrence, on en déduit :

$$C(n) \geqslant \varphi(d)\, n + \sum_{1\leqslant i\leqslant d} k(n_i)\, n_i \text{ Log } n_i$$

La fonction k(n) étant décroissante, on a :

$$C(n) \geqslant \varphi(d)\, n + k(n) \sum_{1\leqslant i\leqslant d} n_i \text{ Log } n_i$$

La fonction x Log x étant convexe, on en déduit :

$$C(n) \geqslant \varphi(d)\, n + k(n)\, d\, \frac{n}{d} \text{ Log } \frac{n}{d}$$

ce qui peut s'écrire :

$$C(n) \geqslant k(n)\, n \text{ Log } n + n\left[\varphi(d) - k(n) \text{ Log } d\right]$$

Comme on a $2 \leqslant d \leqslant n$, le terme entre crochets est positif ou nul d'après la définition 1. On en déduit la minoration :

$$C(n) \geqslant k(n)\, n \text{ Log } n$$

Comme par définition, $k(n) \geqslant k$, on a bien :

$$C(n) \geqslant k(n).\, n \,\mathrm{Log}\, n \geqslant k\, n \,\mathrm{Log}\, n$$

Corollaire 1 : Le coût optimal C(n) vérifie :

$$k(n)\, n \,\mathrm{Log}\, n \leqslant C(n) < k(n)\, n \,\mathrm{Log}\, n + \varphi(d).n$$

où d est un degré-limite pour n $\left(d \in \mathcal{D}(n)\right)$.

Preuve : Il suffit, dans l'inégalité (10), de prendre d dans $\mathcal{D}(n)$.

Cette formulation fait apparaître le même terme $\left[k(n)\, n \,\mathrm{Log}\, n\right]$ dans la borne inférieure et la borne supérieure. L'erreur relative commise en approximant C(n) par la borne inférieure $\left[k(n)\, n \,\mathrm{Log}\, n\right]$ est donc égale à :

$$\mathcal{E} = \frac{\mathrm{Log}\, d}{\mathrm{Log}\, n} \qquad \text{avec } d \in \mathcal{D}(n) \tag{12}$$

Cette erreur relative est donc inférieure ou égale à 1. Mais, comme $\mathcal{D}(n)$ varie avec n, nous ne sommes pas en mesure de dire si la borne inférieure est ou non une bonne approximation. Cependant, nous verrons plus loin que C(n) atteint la borne inférieure infiniment souvent.

2.3. PREMIERES INDICATIONS SUR LA FORME DES ARBRES OPTIMAUX

La borne inférieure du théorème précédent va nous fournir des indications précieuses sur la forme des arbres optimaux. Nous allons d'abord mettre en évidence les facteurs qui interviennent dans la différence entre le coût d'un arbre quelconque et cette borne inférieure. Comme les arbres optimaux sont ceux qui minimisent cette différence, cela nous donnera une idée de leurs propriétés. Nous examinons ensuite dans quelles conditions le coût d'un arbre optimal est précisément égal à cette borne inférieure.
Le lemme suivant nous donne un moyen d'évaluer la différence entre le coût d'un arbre quelconque et la borne inférieure.

Lemme 1 : Pour tout arbre T à n feuilles de poids unitaire, on a :

$$\mathcal{C}(T) - c.n.\mathrm{Log}\, n = \sum_{v \in \mathcal{N}_T} w(v) \left[\Delta_1(v) + \Delta_2(v)\right] \tag{13}$$

c est une constante réelle positive quelconque.
Δ_1 et Δ_2 sont des fonctions dépendant de φ et de c :

$$\Delta_1(v) = \varphi(d(v)) - c.\mathrm{Log}\, d(v) \tag{14}$$

$$\Delta_2(v) = c\left[\mathrm{Log}\, d(v) + \sum_{u \in \mathcal{F}(v)} \frac{w(u)}{w(v)} \cdot \mathrm{Log}\, \frac{w(u)}{w(v)}\right] \tag{15}$$

Preuve : Rappelons que $\mathcal{F}(v)$ désigne l'ensemble des fils de v.
Posons :

$$\Delta(v) = w(v)\left[\Delta_1(v) + \Delta_2(v)\right] \tag{16}$$

Un calcul simple montre que :

$$\Delta(v) = \varphi(d(v))\, w(v) - c.w(v).\mathrm{Log}\, w(v) + \sum_{u \in \mathcal{F}(v)} c\, w(u)\, \mathrm{Log}\, w(u) \tag{17}$$

Il nous faut donc montrer que pour tout arbre T à n feuilles de poids unitaire, on a :

$$\mathcal{C}(T) - c.n.\mathrm{Log}\, n = \sum_{v \in \mathcal{N}_T} \Delta(v) \tag{18}$$

La démonstration se fait par récurrence sur n. Un calcul simple montre que la formule (18) est vérifiée pour n = 2.
Considérons maintenant un arbre T quelconque à n feuilles. Appelons V son sommet, d son degré au sommet, $(T_1,\ldots,T_d)$ ses sous-arbres principaux de poids $(n_1,\ldots,n_d)$. D'après la formule (1.6), son coût peut s'écrire :

$$\mathcal{C}(T) = \varphi(d).n + \sum_{1 \leqslant i \leqslant d} \mathcal{C}(T_i)$$

D'après l'hypothèse de récurrence, on a :

$$\mathcal{C}(T_i) = c\, n_i \,\mathrm{Log}\, n_i + \sum_{v \in \mathcal{N}_{T_i}} \Delta(v)$$

La somme de tous les $\mathcal{C}(T_i)$ est donc égale à :

$$\sum_{1\leq i\leq d} \mathcal{C}(T_i) = \sum_{1\leq i\leq d} c\, n_i \,\mathrm{Log}\, n_i + \sum_{v\in\mathcal{N}_T} \Delta(v) - \Delta(V)$$

La différence à calculer s'écrit donc :

$$\mathcal{C}(T) - c.n.\mathrm{Log}\, n = \sum_{v\in\mathcal{N}_T} \Delta(v)$$

$$+\left[\varphi(d)n - c.n.\mathrm{Log}\, n + \sum_{1\leq i\leq d} c\, n_i \,\mathrm{Log}\, n_i - \Delta(\right.$$

D'après la définition de Δ, le terme entre crochets est nul, ce qui termine la démonstration. ▨

D'après ce lemme, la différence entre le coût d'un arbre quelconque à n feuilles de poids unitaire et la borne inférieure du théorème 2 peut donc s'écrire, si l'on pose $c = k(n)$:

$$\mathcal{C}(T) - k(n).n.\mathrm{Log}\, n = \sum_{v\in\mathcal{N}_T} \left[w(v).\Delta_1(v) + w(v).\Delta_2(v)\right] \tag{19}$$

avec :

$$\Delta_1(v) = \varphi(d(v)) - k(n)\ \mathrm{Log}\, d(v) \tag{20}$$

$$\Delta_2(v) = k(n)\left[\mathrm{Log}\, d(v) + \sum_{u\in\mathcal{F}(v)} \frac{w(u)}{w(v)} \mathrm{Log}\, \frac{w(u)}{w(v)}\right] \tag{21}$$

Examinons de plus près la signification de Δ_1 et Δ_2. Comme $d(v) \leq n$, $\Delta_1(v)$ est positif ou nul d'après la définition de $k(n)$ (définition 1). Il ne s'annule que si $d(v)$ est degré-limite pour n. Dans les autres cas, il est strictement positif.

Voyons maintenant Δ_2. La fonction $x \mathrm{Log}\, x$ étant convexe, $\Delta_2(v)$ est minimum et s'annule si tous les termes $w(u)/w(v)$ sont égaux, c'est-à-dire si $w(u)/w(v) = 1/d(v)$.

Le terme $\Delta_2(v)$ est donc nul si et seulement si tous les fils de v ont même poids. Dans les autres cas, il est positif. Le terme $\Delta_2(v)$ constitue en quelque sorte une mesure du déséquilibre des poids des fils du noeud v. Plus ce déséquilibre est grand, plus $\Delta_2(v)$ augmente. Nous résumons cela dans la proposition suivante :

Proposition 1 : La différence entre le coût d'un arbre quelconque T à poids égaux et unitaires et la borne inférieure est la somme sur chaque noeud v de l'arbre de deux termes $\Delta_1(v)$ et $\Delta_2(v)$ positifs ou nuls. Le premier terme s'annule si et seulement si le degré du noeud est degré-limite pour n, nombre de feuilles de l'arbre. Le deuxième terme mesure le déséquilibre des poids des fils du noeud ; il s'annule si et seulement si ces poids sont tous égaux. Une formulation précise est donnée par les formules (19), (20) et (21).

Les arbres optimaux étant ceux qui minimisent cette différence, ils tendent à avoir pour degrés le ou les degré -limites. Par ailleurs, ils tendent à être aussi équilibrés que possible. Nous disons tendance, car il n'est généralement pas possible de satisfaire ces deux conditions, et en particulier d'annuler simultanément Δ_1 et Δ_2. Par exemple, si n est premier, le seul arbre annulant Δ est l'arbre plat ; mais si n n'est pas degré-limite pour n, Δ n'est pas nul ; le coût de cet arbre plat est donc plus grand que la borne inférieure.
Mais il existe certains arbres qui annulent simultanément Δ_1 et Δ_2 sur chaque noeud :

Théorème 3 : Tout arbre régulier à n feuilles est optimal si ses degrés sont des degrés-limites pour n. Son coût est égal à la borne inférieure $k(n).n.\mathrm{Log}\, n$.

Preuve : Pour chaque noeud v, le terme $\Delta_1(v)$ est nul puisque d(v) est un degré-limite. D'autre part, le terme $\Delta_2(v)$ est également nul : en effet, les fils de v ont tous même poids puisque l'arbre est régulier. Le coût de ces arbres est donc égal à la borne inférieure. Ces arbres sont donc optimaux.

▨

Il existe une preuve directe de ce théorème. Calculons en effet le coût d'un arbre T régulier dont tous les degrés sont des degrés-limites pour n, nombre de feuilles de T. Appelons $(d_1^{\ell_1},\ldots,d_t^{\ell_t})$ la décomposition de T. D'après (1.11), son coût est égal à :

$$\mathcal{C}(T) = n \sum_{1\leq i\leq t} \left[\ell_i \varphi(d_i)\right]$$

Comme $n = d_1^{\ell_1} \times \ldots \times d_t^{\ell_1}$ et que par hypothèse, $\varphi(d_i) = k(n) \text{ Log } d_i$, on en déduit facilement que :

$$\mathcal{C}(T) = k(n)\, n \text{ Log } n$$

qui est précisément la borne inférieure.

Il est évident qu'il y a une infinité d'arbres réguliers verifiant les conditions de ce théorème. Il y a donc une infinité de valeurs de n pour lesquelles C(n) est égal à la bonne inférieure. Il semble donc que cette borne constitue une bonne approximation de C(n). Malheureusement ces valeurs de n peuvent être très espacées et, entre ces valeurs, C(n) peut s'éloigner infiniment de la borne inférieure. Il en est de même en ce qui concerne la forme des arbres : il n'y a pas de règles simples de formation des arbres entre ces valeurs de n.

Nous allons maintenant examiner certains arbres optimaux particuliers.

Corollaire 2 : Tout arbre d-parfait à d^p feuilles est optimal si d est un degré-limite pour d^p. Son coût est égal à la borne inférieure. Si d est l'unique degré-limite pour d^p, alors cet arbre est l'unique arbre optimal à d^p feuilles.

Preuve : La première partie de l'énoncé découle directement du théorème précédent, les arbres parfaits étant des arbres réguliers particuliers. Montrons la deuxième partie. Supposons que d est l'unique degré-limite pour d^p. Considérons un arbre T optimal à d^p feuilles. Son coût est égal à la borne inférieure. D'après la proposition 1, les termes $\Delta_1(v)$ et $\Delta_2(v)$ sont donc nuls pour chaque noeud v de T.

Comme Δ_1 est nul, tous les noeuds de T ont pour degré d puisque d est l'unique degré-limite. Comme Δ_2 est nul, les fils de chaque noeud de T ont même poids et toutes les feuilles de T sont donc situées à la même profondeur. L'arbre T est donc parfait. ▨

Corollaire 3 : Tout arbre plat à n feuilles est optimal si n est un degré-limite pour n. Son coût est égal à la borne inférieure. Si n est l'unique degré-limite pour n, alors l'arbre plat est l'unique arbre optimal à n feuilles.

Preuve : Il suffit de faire p = 1 dans le corollaire précédent. ▨

On peut exprimer ce corollaire de la manière suivante. Si pour n donné, on a :

$$\frac{\varphi(n)}{\text{Log } n} \leq \min_{2\leq d\leq n} \frac{\varphi(d)}{\text{Log } d} \qquad (22)$$

alors l'arbre plat de degré n est optimal. Si l'inégalité est stricte, cet arbre plat est l'unique arbre optimal. Il en résulte la proposition suivante :

Proposition 2 : Si la fonction $\varphi(n)$ / Log n est décroissante, alors les arbres plat sont optimaux quel que soit n et leur coût est égal à la borne inférieure. Si la décroissance est stricte, les arbres plats sont seuls optimaux.

2.4. CLASSIFICATION DES FONCTIONS DE COUT - HYPOTHESE DU DEGRE-LIMITE

Nous allons classer les fonctions de coût en quatre catégories incluses l'une dans l'autre :

1) Fonctions de coût quelconque
2) Fonctions de coût bornées inférieurement : ce sont les fonctions telles que k_φ n'est pas nul (voir définition 2). Le coût optimal est alors borné inférieurement par :

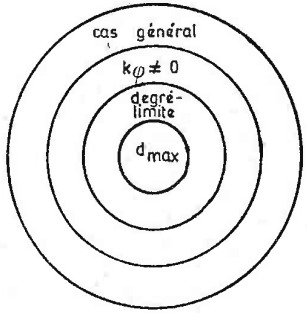
$$C(n) \geq k_\varphi\, n \text{ Log } n$$

3) Fonctions de coût possédant un degré-limite : ce sont les fonctions telles que $k\varphi$ est atteint pour un ensemble $\mathcal{D}$ d'entiers appelés degré-limites :

$$k\varphi = \frac{\varphi(d)}{\text{Log } d} \qquad \text{pour } d \in \mathcal{D}$$

4) Fonctions de coût possédant un degré maximum

Nous représentons les inclusions de ces quatre catégories sur le schéma suivant :



Les deux premières inclusions sont évidentes. Reste à montrer que :

Proposition 3 : Une fonction de coût ayant un degré maximum possède aussi un degré-limite. Si a designe le plus petit degré-limite, on a : $a \leqslant d_{max}$.

Preuve : Il suffit de montrer qu'une fonction n'ayant pas de degré-limite ne possède pas non plus de degré maximum. D'après la définition 3, une telle fonction vérifie :

$$(\forall N)\ (\exists n \geqslant N)\quad \frac{\varphi(n)}{\text{Log } n} < \min_{2 \leqslant d < n} \frac{\varphi(d)}{\text{Log } d}$$

D'après le corollaire 3, les seuls arbres optimaux pour les valeurs de n vérifiant l'inégalité ci-dessus sont les arbres plats. Leur degré au sommet étant précisément n, il en résulte que la fonction de coût ne possède pas de degré maximum.

D'après le corollaire 3, le seul arbre optimal ayant a feuilles est l'arbre plat ; on a donc $a \leqslant d_{max}$. ▨

Pour les trois premières catégories, il existe une condition nécessaire et suffisante simple les caractérisant. Pour les fonctions de coût à degré borné, il y a bien sûr la condition nécessaire et suffisante de définition. Mais cette condition n'est pas simple ! Il existe une condition nécessaire simple, celle de posséder un degré-limite, ce qui peut s'écrire :

$$(\exists a)(\forall n)\quad \frac{\varphi(a)}{\text{Log } a} \leqslant \frac{\varphi(n)}{\text{Log } n}$$

Il existe également une condition suffisante simple, celle du corollaire 1.4 :

$$(\exists a)\ (\exists \varepsilon>0)(\exists N)(\forall n \geqslant N)\quad \frac{\varphi(a)}{\text{Log } a} + \varepsilon \leqslant \frac{\varphi(n)}{\text{Log } n}$$

A cause de la constante ε, on n'a pas de condition nécessaire et suffisante simple.

2.4.1. FONCTIONS DE COUT LOGARITHMIQUES

Que se passe-t-il si $\varphi(n)/\text{Log } n$ est constant ? Si cette fonction est décroissante, il n'y a pas de degré-limite et donc pas de degré maximum, et que si elle est croissante, il y a un degré maximum (voir corollaire 1.5). Si cette fonction est constante, on a la situation intermédiaire ; la fonction de coût a un degré-limite, mais pas de degré maximum.

En effet, tous les degrés sont des degré-limites dans ce cas, puisque

$$\forall n \quad k\varphi = \varphi(n) / \text{Log } n$$

On en déduit facilement que tous les arbres plats sont optimaux puisque leurs coûts sont égaux à la borne inférieure $k\varphi$ n Log n. En fait, tous les arbres réguliers sont optimaux, d'après le théorème 3. Pour n donné, il y a autant d'arbres réguliers optimaux que de factorisations possibles de n. Si n est premier, le seul arbre optimal est donc l'arbre plat. La fonction de coût ne possède donc pas de degré maximum.

2.4.2. HYPOTHESE DU DEGRE-LIMITE

Du point de vue théorique et du point de vue pratique, les seules fonctions de coût intéressantes sont celles qui possèdent un degré maximum. Du point de vue théorique, les résultats montrés jusqu'ici sont valables dans le cas général. Mais il est difficile d'aller plus loin pour les fonctions de coût à degré non borné : les arbres plats étant seuls optimaux pour une infinité de valeurs de n, il n'est pas possible de trouver des régularités pour les arbres optimaux. Dans ce cas, le seul algorithme de recherche des arbres optimaux est l'algorithme en $0(n^2 \text{ Log } n)$

C'est pourquoi les résultats concernent maintenant principalement les fonctions de coût à degré maximum. Cependant, le chapitre suivant s'applique également si l'on suppose seulement l'existence d'un degré-limite. Pour l'instant, nous supposons donc un degré-limite et ferons l'hypothèse du degré maximum aux chapitres 4, 5 et 6.

Nous allons d'abord appliquer les résultats précédents quand il y a un degré-limite.

2.5. PROPRIETES DU COUT OPTIMAL ET DES ARBRES OPTIMAUX S'IL Y A UN DEGRE-LIMITE

Nous supposons donc dorénavant qu'il existe un ou plusieurs entiers, appelés degrés-limites, qui minimisent (n)/Log n. Rappelons la définition 3 :

(23) $$k = \min_{d \geq 2} \frac{\varphi(d)}{\text{Log } d}$$

(24) $$k = \frac{\varphi(a)}{\text{Log } a} = \frac{\varphi(d)}{\text{Log } d} \qquad \text{pour } d \in \mathcal{D}$$

$\mathcal{D}$ désigne l'ensemble des degrés-limites et a le plus petit d'entre eux. Les propriétés montrées aux paragraphes 2 et 3 s'appliquent et prennent une forme plus simple.

2.5.1. BORNE INFERIEURE ET SUPERIEURE - FORME DE C(n)

Proposition 4 : Si la fonction de coût possède un degré-limite, on a :

(25) $$k.n.\text{Log } n \leq C(n) < k\, n \text{ Log } n + \varphi(a)\, n$$

(26) $$C(d^p) = k\, d^p \text{ Log } d^p \qquad \text{pour } d \in \mathcal{D}$$

Rappelons que $\mathcal{D}$ désigne l'ensemble des degrés-limites et a le plus petit d'entre eux.

Preuve : Il suffit d'appliquer les théorèmes 2 et 3 avec d = a en remarquant que $\varphi(a)/\text{Log } a = k$. ▨

Cette proposition montre que $\lfloor k.n.\text{Log } n \rfloor$ est une bonne approximation de C(n). En effet, d'après (26), la borne inférieure $\lfloor k.n.\text{Log } n \rfloor$ est atteinte infiniment souvent. D'autre part, l'erreur relative commise en remplaçant C(n) par $\lfloor k\, n \text{ Log } n \rfloor$ est égale à :

(27) $$\mathcal{E} = \frac{\text{Log } a}{\text{Log } n}$$

L'erreur relative tend vers 0 avec n. On peut donc considérer que
le coût optimal C(n) est asymptotiquement égal à k.n.Log n.

Quelle est la forme de la courbe C(n) ? Nous savons qu'elle touche à la borne inférieure aux points d'abscisses $n = a^p \left(p \geq 0\right)$.
Que se passe-t-il entre ces points ? La courbe C(n) est inférieure, bien sûr, à la borne supérieure précédente. Mais il existe une meilleure borne supérieure. Nous avons vu au lemme 1.2, que le coût $H_a(n)$ d'un arbre a-équilibré à n feuilles de poids égaux et unitaires vérifie :

$$H_a(n) \leq h_a(n) + K(a)$$

La constante K(a) est définie en (1.42) ; la fonction h_a (voir définition 1.17) est constituée par les segments de droite joignant les paires de points du plan

$$\left[a^p, f_a(a^p) \right] \quad \text{et} \quad \left[a^{p+1}, f_a(a^{p+1}) \right]$$

où $f_a(x)$ est précisément la borne inférieure du théorème précédent.
Nous résumons cela sur le schéma suivan :

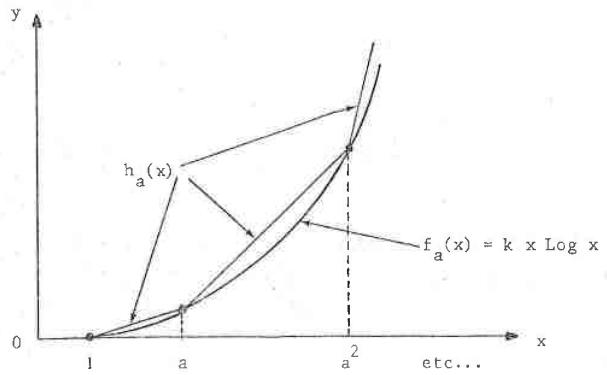


FIGURE 1

La fonction C(n) est supérieure à la borne inférieure $f_a(n)$ et inférieure à une constante près à $h_a(n)$. Notons que cette inégalité s'étend sans difficulté aux réels, C(x) étant définie par interpolation linéaire de C(n) (voir paragraphe 2.1.1).

La fonction C(n) est-elle plus proche de $h_a(n)$ ou de $f_a(n)$?
Nous verrons dans la suite que la plupart du temps C(n) s'éloigne infiniment de ces deux bornes quand n croît. C'est pourquoi nous serons amenés à définir de meilleures bornes au chapitre suivant.

2.5.2. FORME DES ARBRES OPTIMAUX

Dans le cas où la fonction de coût possède un degré-limite, les propriétés générales montrées plus haut se simplifient :

Proposition 5 : Tout arbre régulier est optimal si ses degrés sont des degrés-limites. Son coût est égal à la borne inférieure.

Preuve : Il suffit d'appliquer le théorème 3 en remarquant que les degrés-limites sont valables pour tout n. ▨

Donnons un exemple. Si $\varphi(d) = d + \lambda_o$, avec :

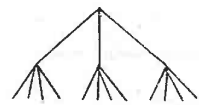
$$\lambda_o = \frac{4 \text{ Log } 3 - 3 \text{ Log } 4}{\text{Log } 4 - \text{Log } 3}$$

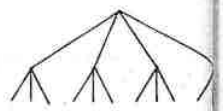
on verra que la fonction de coût possède deux degrés-limites, 3 et 4. Dans ce cas, cette proposition indique que pour $n = 3^p.4^q$, les arbres réguliers de décomposition $(3^p, 4^q)$ sont optimaux. Par exemple, les arbres réguliers suivants sont optimaux pour les valeurs de n indiquées :

n = 3

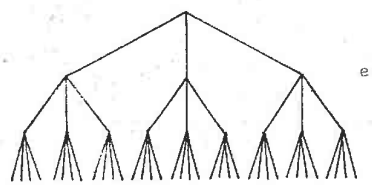
n = 4

$n = 12 = 3^1 . 4^1$ et

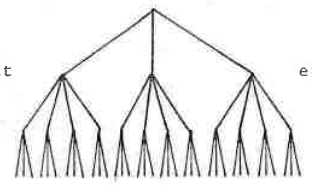




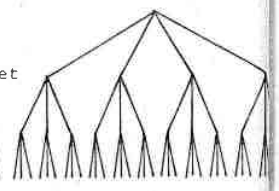
$n = 36 = 3^2 . 4^1$



et



et



Corollaire 4 : Tout arbre d-parfait est optimal si d est un degré-limite. Son coût est égal à la borne inférieure. Si le degré-limite est unique, alors cet arbre est l'unique arbre optimal.

Preuve : Cela découle du corollaire 2. ▨

Par exemple, si $\varphi(d) = d + 0.5$, on verra que la fonction de coût possède un degré-limite unique égal à trois. Les arbres 3-parfaits sont donc seuls optimaux pour $n = 3^p$ ($p \geqslant 0$).

2.6. EXEMPLE DES FONCTIONS DE COUT $\varphi(d) = d + \lambda$

Nous allons appliquer les résultats précédents aux fonctions de coût du type $\varphi(d) = d + \lambda$, où λ est un réel positif ou nul.

2.6.1. DEGRE MAXIMUM

Examinons d'abord la fonction $(x+\lambda)/\text{Log } x$. Quand λ est positif ou nul, elle possède un seul minimum réel x_o. D'après le corollaire 1.5, la fonct de coût possède donc un degré maximum. Par des méthodes combinatoires, VUILLEMIN et SCHLUMBERGER [20] ont montré qu'il est égal à :

$$d_{max} = \left\lceil \min_{\ell \geqslant 1} \left[\ell + \left(1 + \frac{1}{\ell}\right)(1+\lambda) \right] \right\rceil \qquad \ell \in N$$

Rappelons que la notation $\lceil x \rceil$ désigne l'entier égal ou immédiatement supérieure au réel x. On trouve également la démonstration de ce résultat dans KNUTH ([13], p. 372, théorème M). Le degré maximum est donc une fonction croissante de λ. Pour $\lambda = 0$, d_{max} est égal à trois. En minimisant la fonction réelle $\left[x + (1+1/x)(1+\lambda)\right]$, on voit que d_{max} est approximativement égal à :

(29) $$\boxed{d_{max} \simeq (1+\lambda) + 2\sqrt{1+\lambda}}$$

2.6.2. DEGRE-LIMITE

Les fonctions de coût du type $\varphi(d) = d+\lambda$ ont également un degré-limite car la fonction $(x+\lambda)/\text{Log } x$ a un seul minimum réel x_o. Le degré-limite est donc généralement unique et égal, soit à $\lfloor x_o \rfloor$, soit à $\lceil x_o \rceil$. Il y a deux degrés-limites si la fonctions $(x+\lambda)/\text{Log } x$ a même valeur pour $x = \lfloor x_o \rfloor$ et $x = \lceil x_o \rceil$. Ces valeurs étant consécutives, il y a deux degré -limites, d et d+1, si :

$$k = \frac{\varphi(d)}{\text{Log } d} = \frac{\varphi(d+1)}{\text{Log } d+1}$$

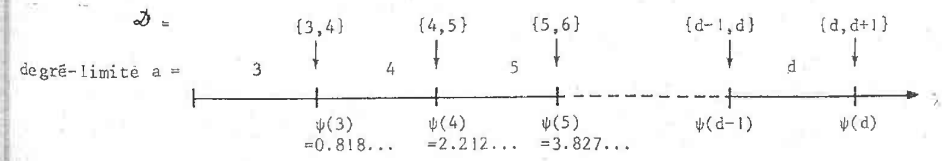
ce qui implique :

(30) $$\lambda = \psi(d) \qquad \psi(d) = \frac{(d+1)\text{ Log } d - d\text{ Log}(d+1)}{\text{Log}(d+1) - \text{Log } d}$$

On en deduit facilement que :

(31) $$\begin{cases} \text{si } \psi(d-1) < \lambda \leqslant \psi(d) & \text{alors } a = d \\ \text{si } \lambda = \psi(d) & \text{alors } \mathcal{D} = \{d, d+1\} \end{cases}$$

Cela peut se traduire sur le schéma suivant :



Un calcul simple montre que le minimum réel x_o est solution de :

(32) $$\lambda = x_o \text{ Log } x_o - x_o$$

et que la valeur de la fonction $(x+\lambda)/\text{Log } x$ en x_o est précisément x_o.
On en déduit que k est approximativement égal au degré-limite a.

L'égalité se produit quand :

(33) $$\lambda = a.\text{Log } a - a$$

Dans le cas général, on a évidemment :

(34) $$k = \frac{\varphi(a)+\lambda}{\text{Log } a}$$

2.6.3. CAS $\varphi(d) = d$

D'après les résultats précédents, on a :

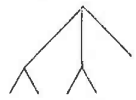
$$a = d_{max} = 3$$

Dans ce cas, KNUTH ([13] , Théorème L) a montré que les arbres optimaux ont la forme suivante :

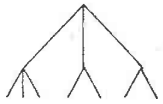
soit p l'entier tel que : $3^p \leqslant n < 3^{p+1}$
construire un arbre 3-parfait de profondeur p
si $n > 3^p$, remplacer chaque feuille par un noeud terminal binaire : jusqu'à ce qu'il y ait n feuilles, ou bien 2.3^p feuilles,
si $n > 2.3^p$, remplacer chaque noeud terminal binaire par un noeud terminal ternaire, jusqu'à ce qu'on obtienne n feuilles.

Par exemple :

si n = 5, l'arbre optimal est :



si n = 7, l'arbre optimal est :



On en déduit que le coût optimal est donné par la formule suivante :

(35)
$$\begin{cases} 3^p \leqslant n < 2.3^p & C(n) = (3p+4)n - 4.3^p \\ 2.3^p \leqslant n < 3^{p+1} & C(n) = (3p+5)n - 2.3^{p+1} \end{cases}$$

On voit facilement que C(n) est une ligne brisée convexe dont les points anguleux ont pour abscisses 3^p et 2.3^p :

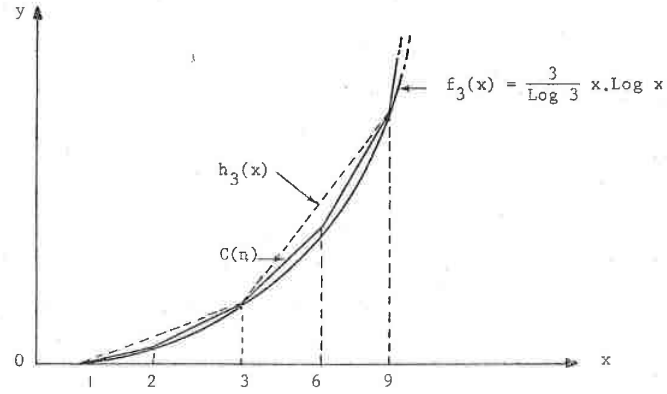


FIGURE 2

On peut remarquer que C(n) s'éloigne infiniment de $f_3(n)$ et $h_3(n)$ pour $n = 2.3^p$:

$$f_3(2.3^p) = 6p.3^p + 6(\text{Log}_3 2).3^p$$

$$C(2.3^p) = 6p.3^p + 4.3^p$$

$$h_3(2.3^p) = 6p.3^p + \frac{9}{2}.3^p$$

2.6.4. CAS GENERAL : $\varphi(d) = d+\lambda$

Nous ne retrouvons pas les belles régularités du cas précédent. Les arbres ont des formes irrégulières, ainsi que les courbes C(n). En particulier, la fonction C(n) n'est jamais convexe si $\lambda > 0$, comme nous le verrons au chapitre suivant. Nous indiquons ci-dessous quelques arbres optimaux et leurs coûts pour $\varphi(d) = d+1$ et $\varphi(d) = d+0.5$.

Cas $\varphi(d) = d+1$

On a $d_{max} = 5$ et le degré-limite est unique : a = 4

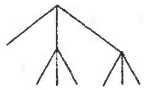
C(2) = 6

C(3) = 12

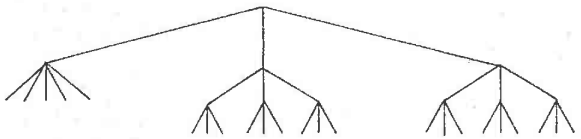
C(4) = 20

C(5) = 30

C(6) = 42



C(7) = 52



C(23) = 266

Cas $\varphi(d) = d+0.5$

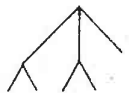
On a $d_{max} = 4$ et a = 3. Nous tracerons en 3.3. la courbe C(n) et étudierons plus particulièrement ce cas.

C(2) = 5

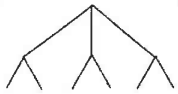
C(3) = 10.5

C(4) = 18

Pour n = 5, il y a 3 arbres optimaux équivalents :



C(5) = 27.5



C(6) = 36

C(7) = 45

C(8) = 54

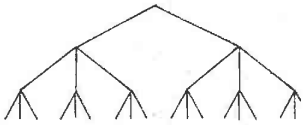
C(9) = 63

C(11) = 85

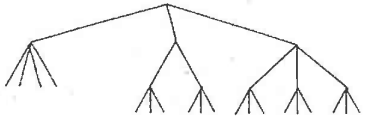
C(12) = 96

C(14) = 120

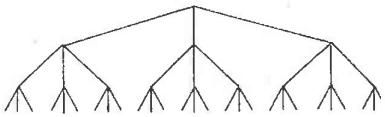
C(16) = 144



C(18) = 171



C(19) = 183.5



C(27) = 283.5

Nous constatons que les arbres 3-parfaits sont bien optimaux. Remarquons que pour n = 16, l'arbre 4-parfait est optimal.

2.6.5. QUELQUES QUESTIONS

Dans les exemples précédents, les différences de profondeur des feuilles n'excèdent jamais un. Il est vrai qu'il s'agit seulement de petites valeurs de n. Mais même pour de grandes valeurs de n, cette propriété reste vraie si $\lambda = 0$ ou $\lambda = 0.5$.

Par ailleurs, cette propriété semble une conséquence logique de la proposition 1, à savoir que les arbres optimaux ont tendance à être équilibrés. En fait, nous verrons que dans le cas général, la différence des profondeurs n'est pas bornée : il existe des fonctions φ pour lesquelles cette différence croît infiniment avec le nombre de feuilles (voir chapitre 5). Il en est de même pour les différences de poids des sous-arbres principaux.

Une autre question se pose : la proposition 1 montre que les arbres optimaux ont tendance à avoir pour degré les degré-limites. Mais nous avons vu sur les exemples précédents que les degrés peuvent être différents des degré -limites. Par exemple, pour $\varphi(d) = d+0.5$, on peut montrer que le seul arbre optimal à 16 feuilles est l'arbre 4-parfait de profondeur 2. Or, le degré-limite est unique et égal à 3 dans ce cas.
Nous verrons aux chapitres 4 et 5 que les arbres ont pour degré au sommet le (ou les) degré -limite(s) seulement à partir d'un certain rang.

CHAPITRE 3

ETUDE DE L'ENVELOPPE CONVEXE DU COUT OPTIMAL C(n)

Nous supposons à partir de maintenant que la fonction de coût possède un degré-limite, c'est-à-dire qu'il existe des entiers, appelés degré -limites, minimisant $\varphi(n)/\text{Log } n$:

(1) $$\text{pour } d \in \mathcal{D} \quad k = \frac{\varphi(d)}{\text{Log } d} \leqslant \frac{\varphi(n)}{\text{Log } n}$$

Nous avons montré au chapitre précédent (proposition 2.4) qu'avec cette hypothèse, le coût optimal C(n) est encadré par :

(2) $$k\, n \text{ Log } n \leqslant C(n) < k\, n \text{ Log } n + \varphi(a)\, n$$

où a désigne le plus petit degré-limite. Nous en avons déduit des indications sur la forme des arbres optimaux par la méthode suivante : chaque fois que l'on peut construire un arbre à n feuilles de coût $k.n.\text{Log } n$, alors cet arbre est optimal d'après la formule (2).

Nous allons étudier dans ce chapitre une nouvelle minoration convexe E(n) de C(n), qui est meilleure que la fonction convexe $[k\, n \text{ Log } n]$. La fonction E(n) est plus difficile à calculer mais donne des indications plus précises sur la forme des arbres optimaux. Nous montrons également que, dans un certains sens, les fonctions C et E sont asymptotiquement égales.

Dans la suite, nous poserons :

(3) $$G(x) = k \times \text{Log } x$$

3.1. DEFINITION ET PROPRIETES DE L'ENVELOPPE CONVEXE E

Définition 1 : Nous appelons poche p, ou P_p (p entier, $p \geq 0$) l'intervall réel $\left[a^p, a^{p+1}\right]$. Si n est un entier positif, nous appelons poche de n l'intervalle P_p tel que $n \in P_p$ et $n \neq a^{p+1}$.

Dans la suite, nous noterons $C_p(x)$ la partie de C(x) définie sur l'intervalle P_p.

Définition 2 : Nous appelons $E_p(x)$ la fonction réelle définie sur P_p de la manière suivante :

- considérons l'enveloppe convexe des points P(n) du plan d'abscisse n et d'ordonnée C(n), où n est un entier de la poche p.
- E_p est la ligne brisée convexe constituée par la partie de l'enveloppe convexe située sous le segment d'extrêmités $P(a^p)$ et $P(a^{p+1})$

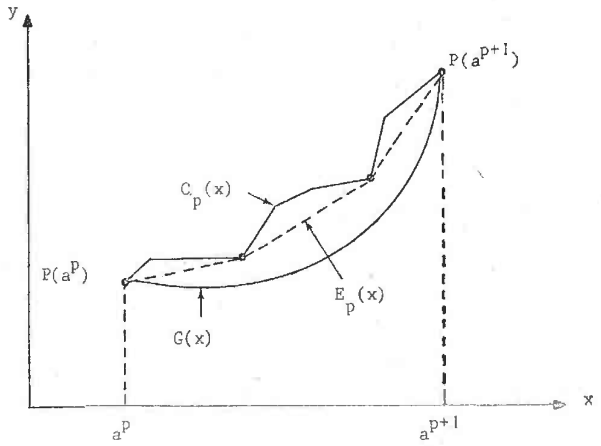


FIGURE 1

Lemme 1 : (i) Les points anguleux de E_p ont une abscisse entière e et on a $E_p(e) = C(e)$

(4) (ii) On a : $G(x) \leq E_p(x) \leq C(x)$ pour x réel $\in P_p$

(5) $G(a^p) = E_p(a^p) = C(a^p)$

(6) $G(a^{p+1}) = E_p(a^{p+1}) = C(a^{p+1})$

Preuve : La proposition (i) découle de la définition de E_p. D'après la proposition 2.4, les fonctions G et C sont égales en a^p et a^{p+1} ; on en déduit (5) et (6).

Montrons la formule (4). D'après la définition de E_p, si n est entier, on a $E_p(n) \leq C(n)$. Entre les valeurs entières, E_p et C sont des segments de droite ; on a donc $E_p(x) \leq C(x)$ pour x réel. Il reste à montrer que $G(x) \leq E_p(x)$. Si e est l'abscisse d'un point anguleux de E_p, on a $E_p(e) = C(e)$ d'après (i). D'après la proposition 2.4, on en déduit que $E_p(e) \geq G(e)$. Entre les points anguleux, E_p est un segment de droite et G est convexe ; on a donc $E_p(x) \leq C(x)$ pour x réel quelconque de la poche p. ▨

Nous sommes maintenant en mesure de définir l'enveloppe convexe E de C.

Définition 3 : Nous appelons enveloppe E de C la fonction définie sur les réels plus grands ou égaux à 1 par réunion des fonctions E_p ($p \geq 0$). Plus précisément, si $x \in P_p$ alors $E(x) = E_p(x)$.

Cette définition a un sens puisque E_p et E_{p+1} sont égales en a^{p+1} qui est l'intersection de leurs domaines de définition P_p et P_{p+1}.

Proposition 1 : (i) E est convexe

(ii) Les points anguleux de E ont une abscisse entière e, et on a : $E(e) = C(e)$

(7) (iii) On a : $G(x) \leq E(x) \leq C(x)$

$G(a^p) = E(a^p) = C(a^p)$ p entier, $p \geq 0$

(8) (iiii) Si $n = \prod_{1 \leq i \leq t} d_i^{\ell_i}$ avec $d_i \in \mathcal{D}$ et $\ell_i \in N^+$

alors n est l'abscisse d'un point anguleux et on a $G(n) = E(n) = C(n)$.

Preuve : Les propositions (ii) et (iii) résultent du lemme 1. Pour (i), comme les fonctions E_p sont convexes à l'intérieur de leurs domaines de définition, il reste à montrer la convexité aux "points de raccordement" a^p. Appelons ℓ' la pente du dernier segment de E_{p-1} et ℓ'' la pente du premier segment de E_p. Il faut montrer que :

$$\ell' \leq \ell''$$

Appelons ℓ la pente de la tangente à la fonction G en a^p.

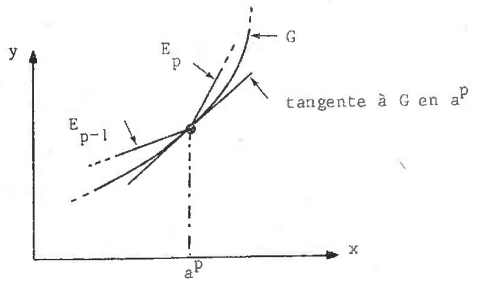


FIGURE 2

D'une part, G, E_p et E_{p-1} sont égales en a^p. D'autre part, E_p et E_{p-1} sont plus grandes ou égales à la fonction G qui est convexe. Il en résulte que :

$$\ell' < \ell < \ell''$$

L'inégalité $\ell' < \ell''$ étant stricte, les entiers a^p sont des abscisses de points anguleux de E.

Montrons (iiii). D'après la proposition 2.5, l'arbre régulier à n feuilles de décomposition $(d_1^{\ell_1}, \ldots, d_t^{\ell_t})$ est optimal et son coût vérifie

$$C(n) = G(n).$$

Par un raisonnement analogue au précédent, on en déduit que n est l'abscisse d'un point anguleux de E. ▨

Définition 4 : Nous appelons arbres correspondant aux points anguleux de E les arbres optimaux dont le poids est l'abscisse d'un point anguleux.

D'après la proposition précédente, le coût de ces arbres est égal à E(n), où n est leur nombre de feuilles. Nous examinons dans la suite la forme de ces arbres.

On peut faire une première constatation : d'après la proposition précédente, les arbres réguliers dont les degrés sont les degré-limites correspondent à des points anguleux.

Nous verrons au paragraphe suivant qu'il existe d'autres arbres correspondant aux points anguleux de E. Nous montrons d'abord que E possède la propriété suivante :

Proposition 2 : (i) Pour tous x et x_i réels et d entier vérifiant :

$$\sum_{1 \leq i \leq d} x_i = x \qquad x, x_i \geq 1 \qquad d \geq 2$$

On a :

(9) $$E(x) \leq \varphi(d)x + \sum_{1 \leq i \leq d} E(x_i)$$

(ii) Pour tous x réel et d entier vérifiant
$x/d > 1 \quad d \geq 2$
On a :

(10) $$E(x) \leq \varphi(d)x + d.E(x/d)$$

Preuve : La fonction E étant convexe, (i) résulte de (ii). Montrons donc la proposition (ii). Appelons e_1 et e_2 les abscisses des points anguleux les plus proches de x/d $(e_1 \leq x/d \leq e_2)$ et posons :

(11) $$x/d = \alpha_1 e_1 + \alpha_2 e_2 \qquad \text{avec } \alpha_1+\alpha_2 = 1$$

La fonction E étant un segment de droite entre e_1 et e_2, on a :

(12) $$E(x/d) = \alpha_1 E(e_1) + \alpha_2 E(e_2)$$

D'autre part, la fonction E étant convexe, on a :

(13) $$E(x) \leq \alpha_1 E(d\, e_1) + \alpha_2 E(d\, e_2)$$

D'après la proposition précédente, on a $E(d\, e_1) \leq C(d\, e_1)$.
D'après la définition de C, on a : $C(d\, e_1) \leq d\, e_1 \varphi(d) + d(e_1)$.
Comme e_1 est l'abscisse d'un point anguleux, on a $C(e_1) = E(e_1)$ d'après la proposition précédente. En combinant ces formules, on obtient :

$$E(d\, e_1) \leq d.e_1 \varphi(d) + d\, E(e_1)$$

On montre de même que :

$$E(d\, e_2) \leq d\, e_2 \varphi(d) + d\, E(e_2)$$

En combinant ces deux dernières inégalités avec l'inégalité (13), on obtient :

$$E(x) \leq \varphi(d).x + d\left[\alpha_1 E(e_1) + \alpha_2 E(e_2)\right]$$

D'après l'égalité (12), on a bien :

$$E(x) \leq \varphi(d)\, x + d.E(x/d)$$

On peut remarquer que les trois fonctions C, G et E vérifient :

$$g(x) \leq \varphi(d)\, x + \sum_{1 \leq i \leq d} g(x_i)$$

avec les mêmes conditions que dans la proposition 2.(i). Si l'on impose l'égalité pour certains x et x_i entiers, on obtient la définition de C. Si l'on impose que g est convexe, on obtient des bornes inférieures de C, comme G et E, mais il n'y a plus en général d'entiers assurant l'égalité. En ce qui concerne la fonction G, l'égalité est assurée pour d = a et x_i = x/d. (voir lemme 2(iii) au paragraphe 5). Nous étudions plus en détail les fonctions g de ce type au chapitre 6.

3.2. FORME DES ARBRES OPTIMAUX CORRESPONDANT AUX POINTS ANGULEUX DE E

De même que G, la fonction E est une fonction minorante convexe de C. Comme au chapitre précédent, nous examinons maintenant la forme des arbres optimaux quand C = E, c'est-à-dire aux points anguleux.

Théorème 4 :

(i) Dans un arbre optimal correspondant à un point anguleux de E, tous les sous-arbres correspondent à un point anguleux et les fils d'un même noeud de l'arbre ont tous même poids.

(ii) A un point anguleux correspond au moins un arbre régulier optimal.

Preuve de (i) : Soient n l'abscisse d'un point anguleux et T un arbre optimal correspondant. Appelons d son degré au sommet et $n_1,\ldots,n_d$ les poids de ses sous-arbres principaux. Pour montrer (i), il suffit de montrer que :

(14) $$n_1 = \ldots = n_d = n/d$$

(15) n/d est l'abscisse d'un point anguleux.

En effet, d'après (15), les sous-arbres principaux correspondent eux-mêm à des points anguleux. Ils possèdent donc eux-mêmes les propriétés (14)et En continuant ainsi, on montre bien (i).

Montrons d'abord (14) :

Les sous-arbres principaux ont tous même poids n/d.

L'arbre T étant optimal, on a :

$$C(n) = \varphi(d)\, n + \sum_{1 \leqslant i \leqslant d} C(n_i)$$

Comme $C \geqslant E$, on en déduit :

$$C(n) \geqslant \varphi(d)\, n + \sum_{1 \leqslant i \leqslant d} E(n_i) \tag{16}$$

D'autre part, d'après la proposition 2, on a :

$$E(d.n_i) \leqslant \varphi(d)\, d\, n_i + d.E(n_i)$$

Si l'on ajoute ces inégalités membre à membre pour $1 \leqslant i \leqslant d$, on obtient :

$$\varphi(d)\, n + \sum_{1 \leqslant i \leqslant d} E(n_i) \geqslant \frac{1}{d} \sum_{1 \leqslant i \leqslant d} E(d\, n_i) \tag{17}$$

D'après la convexité de E, on a :

$$\frac{1}{d} \sum_{1 \leqslant i \leqslant d} E(d\, n_i) \geqslant E(n) \tag{18}$$

On déduit des inégalités (16), (17) et (18) que :

$$C(n) \geqslant \frac{1}{d} \sum_{1 \leqslant i \leqslant d} E(d\, n_i) \geqslant E(n)$$

Comme n est un point anguleux, on a $E(n) = C(n)$; on déduit que :

$$E(n) = \frac{1}{d} \sum_{1 \leqslant i \leqslant d} E(d\, n_i)$$

La fonction E étant convexe et n étant un point anguleux de E, cette dernière égalité n'est possible que si $n = dn_i$. On a donc bien $n_i = n/d$. Montrons maintenant (15)

n/d est l'abscisse d'un point anguleux.

Appelons e_1 et e_2 les abscisses des points anguleux les plus proches de n/d ($e_1 \leqslant n/d \leqslant e_2$; $e_1 \neq e_2$). Posons :

$$n/d = \alpha_1 e_1 + \alpha_2 e_2 \qquad \alpha_1 + \alpha_2 = 1 \qquad \alpha_1 \geqslant 0 \qquad \alpha_2 \geqslant 0$$

D'après la proposition 2, on a :

$$E(n) \geqslant \varphi(d)\, n + d\, E(n/d)$$

Comme E est une droite entre e_1 et e_2, on a : $E(n/d) = \alpha_1 E(e_1) + \alpha_2 E(e_2)$.

L'inégalité précédente peut donc s'écrire :

$$E(n) \geqslant \alpha_1 \left[d\, e_1\, \varphi(d) + d.E(e_1) \right] + \alpha_2 \left[d\, e_2\, \varphi(d) + d\, E(e_2) \right]$$

D'après la proposition 2, chacun des termes entre crochets est supérieur ou égal à $E(d\, e_1)$ et $E(d\, e_2)$ respectivement. D'où :

$$E(n) \geqslant \alpha_1\, E(d\, e_1) + \alpha_2\, E(d\, e_2)$$

La fonction E étant convexe et n étant un point anguleux de E, cette dernière égalité n'est possible que si α_1 ou α_2 est nul, c'est-à-dire si n/d est confondu avec e_2 ou e_1. Dans les deux cas, n/d est un point. point anguleux de E.

Preuve de (ii) : La démonstration se fait par récurrence sur n. C'est trivial pour n = 1, l'arbre étant réduit à un point Considérons maintenant un arbre optimal T correspondant à un point anguleux d'abscisse n. D'après (i), ses sous-arbres principaux ont tous même poids et correspondent à un même point anguleux de E.

Par hypothèse de récurrence, à ce point anguleux correspond au moins un arbre optimal régulier. Si nous remplaçons les sous-arbres principaux de T par cet arbre régulier, nous obtenons un arbre régulier de même coût que T, donc optimal pour n.

Notons qu'à un même point anguleux peuvent correspondre des arbres réguliers de décomposition différentes. Dans la suite, nous dirons souvent "décomposition d'un point anguleux" au lieu de "décomposition d'un arbre régulier correspondant à un point anguleux".

D'après la proposition 1.(iiii), les arbre réguliers dont les degrés sont des degré-limites correspondent à des points anguleux de E. Le théorème précédent met en évidence d'autres arbres réguliers correspond à des points anguleux, comme nous le verrons sur l'exemple du paragraphe suivant.

Corollaire 1 : Si le degré d figure dans une décomposition d'un point anguleux d'abscisse n, alors :

(19) $E(n) = \varphi(d)\, n + d\, E(n/d)$

(20) $C(n) = \varphi(d)\, n + d\, C(n/d)$

Preuve : Considérons l'arbre régulier correspondant à cette décompositi Par hypothèse, à une certaine profondeur, tous les noeuds de l'arbre ont degré d. Par filtrages successifs, on peut construire un arbre régulier de degré au sommet d. Le filtrage ne changeant pas le coût (proposition 1.2), cet arbre régulier est optimal. On en déduit (20). D'après le théorème précédent, on a $C(n/d) = E(n/d)$; on en déduit (19), puisque $C(n) = E(n)$ par hypothèse.

3.3. EXEMPLE DE LA FONCTION DE COUT $\varphi(d) = d+0.5$

Nous avons représenté sur la figure 3, les fonctions C, E et G dans le où $\varphi(d) = d+0.5$. Nous avons déjà étudié ce cas en 2.6.4. et donné la f et le coût des arbres optimaux pour quelques valeurs de n. Rappelons qu

$$a = 3 \qquad d_{max} = 4 \qquad k = 3.5/\mathrm{Log}\ 3$$

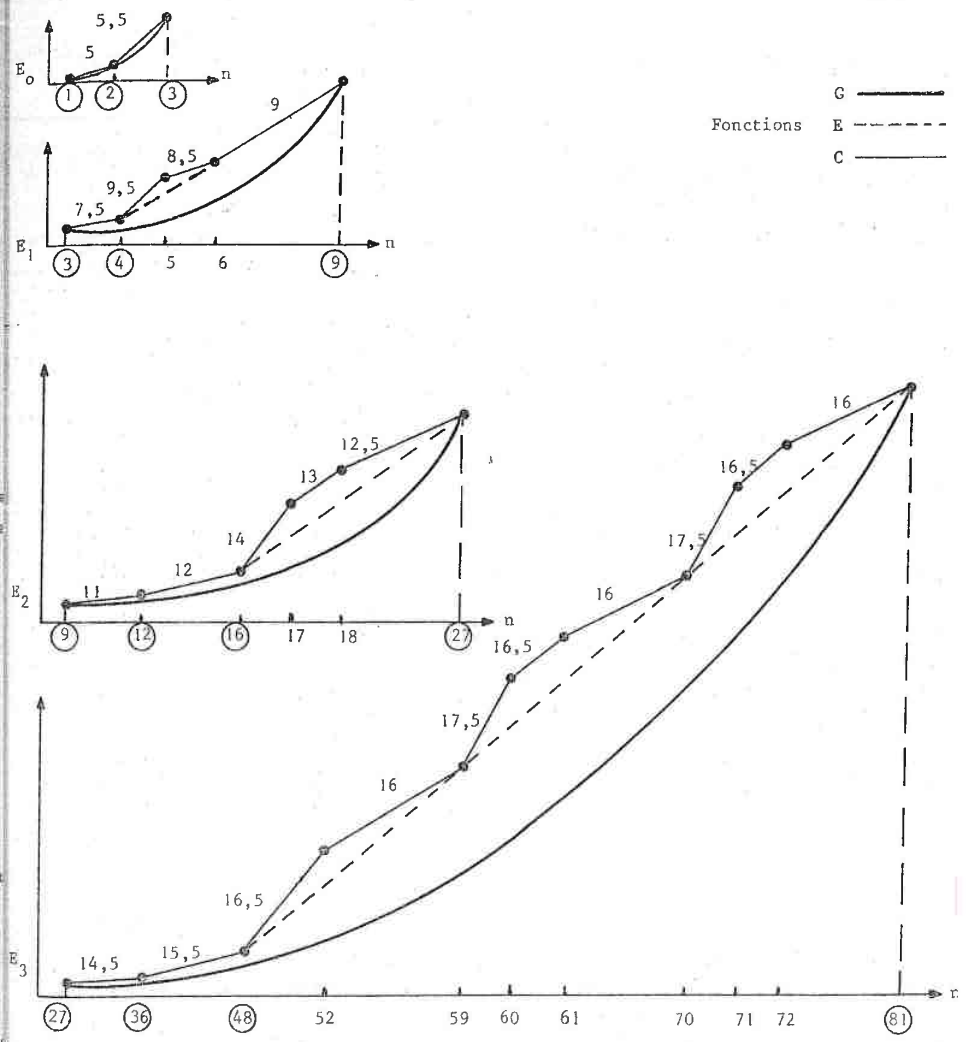


FIGURE 3

Ce dessin ne respecte pas les échelles . Pour des raisons de présentation, nous avons représenté séparément E_o, E_1, E_2 et E_3. Nous avons entouré les abscisses des points anguleux de E et indiqué les pentes des segments de droite de C.

Chaque fois que E et C sont confondues (par exemple sur l'intervalle $[27,48]$), nous n'avons pas tracé de pointillés.

Nous indiquons dans le tableau ci-dessous l'abscisse des points anguleux ainsi que la décomposition et le coût des arbres réguliers correspondant

n = abscisses des points anguleux	Décomposition	Coût optimal C(n) = E(n)
1		0
2	2^1	5
3	3^1	10.5
4	4^1	18
9	3^2	63
12	$3^1,4^1$	96
16	4^2	144
27	3^3	283.5
36	$3^2,4$	414
48	$3^1,4^2$	600
81	3^4	1134

TABLEAU 1

Notons que 6 n'est pas l'abscisse d'un point anguleux, bien qu'on ait C(6) = E(6) ; en effet, E est une droite sur l'intervalle $[4,9]$.
De même 59 et 70 ne sont pas des abscisses de points anguleux.
On peut remarquer que les abscisses des points anguleux de E_3 se déduisent de celles de E_2 par une multiplication par 3. Un examen plus fin montre que :

$$E_3(x) = \varphi(3)x + 3.E_2(x/3)$$

On peut se demander si E_4 ne se déduit pas de E_3 par la même transforma

La réponse est oui : nous montrerons au chapitre suivant que si la fonction de coût a un seul degré-limite, les fonctions E_p se déduisent l'une de l'autre par des transformations de ce type à partir d'un certain rang.

3.4. CALCUL DE L'ENVELOPPE CONVEXE

Nous allons donner des indications facilitant le calcul de l'enveloppe convexe E.

Proposition 3 : Pour calculer l'enveloppe convexe, il suffit de se limiter aux arbres réguliers. Plus précisément : E est l'enveloppe convexe de C', où C'(n) minimise le coût des arbres réguliers à n feuilles.

Preuve : Appelons E' l'enveloppe convexe de C' et montrons que E' = E. Par définition de C, on a C' ⩾ C ; on en déduit :

(21) $\quad E' \geqslant E$

D'après le théorème 4.(ii), à chaque point anguleux de E d'abscisse n correspond un arbre régulier optimal. On a donc E(n) = C'(n) aux points anguleux de E. Comme E' est l'enveloppe convexe de C', on en déduit :

(22) $\quad E \geqslant E'$

On a donc bien E = E', d'après (21) et (22). ▨

La proposition suivante va restreindre la classe des arbres réguliers à examiner pour calculer l'enveloppe convexe :

Proposition 4 : Les décompositions $(d_1^{\ell_1},\ldots,d_t^{\ell_t})$ des arbres réguliers optimaux obéissent à la règle suivante : si d_i n'est pas un degré-limite alors ℓ_i est borné par une constante $L(d_i)$:

(23) $$L(d_i) = \frac{\varphi(a).}{\varphi(d_i)-k.\mathrm{Log}\ d_i}$$

Preuve : Appelons n le poids de l'arbre régulier optimal. Son coût s'écrit :

$$C(n) = n \sum_{1 \leq i \leq t} \ell_i \cdot \varphi(d_i)$$

D'après la proposition 2.4, on a :

$$C(n) < k\, n \operatorname{Log} n + \varphi(a)\, n$$

ce qui s'écrit :

$$n \sum_{1 \leq i \leq t} \ell_i\, \varphi(d_i) < k\, n \sum_{1 \leq i \leq t} \ell_i \operatorname{Log} d_i + \varphi(a)\, n$$

soit :

$$\sum_{1 \leq i \leq t} \ell_i \left[\varphi(d_i) - k \operatorname{Log} d_i \right] < \varphi(a)$$

D'après la définition de k, les termes entre crochets sont positifs si d_i n'est pas un degré-limite. On en déduit $\ell_i < L(d_i)$.

Proposition 5 : Si la fonction de coût a un degré maximum d_{max}, à chaque point anguleux correspond au moins un arbre régulier dont tous les degrés sont plus petits ou égaux à d_{max}.

Preuve : Il suffit de refaire la preuve de la proposition (ii) du théorème 4 en choisissant à chaque fois un arbre optimal de degré au sommet inférieur ou égal à d_{max}.

Corollaire 2 : Si la fonction de coût possède un degré maximum, on peut calculer l'enveloppe convexe E sur $\left[1, a^p\right]$ de la manière suivante :

trouver l'enveloppe convexe inférieure des points :

$(1,0)\ (X,Y)$ $\quad \begin{cases} X = d_1^{\ell_1} \times \ldots \times d_t^{\ell_t} \\ Y = X \sum_{1 \leq i \leq t} \ell_i\, \varphi(d_i) \end{cases}$

avec $\begin{cases} 1 \leq X \leq a^p \\ 2 \leq d_i \leq d_{max} \\ 0 < \ell_i < L(d_i) \quad \left(L(d_i) \text{ de la proposition 4}\right) \\ d_i \text{ et } \ell_i \text{ entiers} \end{cases}$

Preuve : Il suffit d'appliquer la proposition 3 avec les restrictions des propositions 4 et 5.

Dans le cas où il y a un seul degré-limite, nous verrons au chapitre 4, une méthode plus rapide permettant de calculer E en temps fini sur $[1, \infty[$

3.5. FORME ASYMPTOTIQUE DU COUT OPTIMAL ET DE L'ENVELOPPE CONVEXE

Nous avons vu au chapitre précédent que la fonction $G(x) = k \times \operatorname{Log} x$ est asymptotiquement égale à C. La fonction E est une meilleure approximation de C puisque l'on a $G \leq E \leq C$.
Nous allons donner une méthode permettant de comparer asymptotiquement ces trois fonctions et montrer que les transformées $\mathcal{E}_p$ et $\mathcal{C}_p$ de E et C sont asymptotiquement égales.

3.5.1. DEFINITION ET PROPRIETES DES TRANFORMEES $\mathcal{E}_p$ ET $\mathcal{C}_p$

On a vu sur l'exemple du paragraphe 3 que E_3 se déduit de E_2 par la transformation :

$$E_3(x) = \varphi(a)x + a \cdot E_2(x/a) \qquad \text{avec } a = 3.$$

Cette transformation "ramène" une fonction définie sur l'intervalle P_2 à une fonction définie sur P_3. On peut définir aussi la transformation inverse qui fait passer de P_3 à P_2. L'idée est de "ramener" ainsi C_p et E_p définies sur l'intervalle P_p à des fonctions $\mathcal{C}_p$ et $\mathcal{E}_p$ définies sur $P_o = [1,a]$. Ceci afin de pouvoir comparer C_p et E_p quand p varie. C'est pourquoi nous définissons la transformation suivante :

Définition 5 : Nous appelons $\mathcal{C}$ la transformation linéaire définie par la matrice suivante :

(24) $$\mathcal{C} = \begin{bmatrix} a & 0 \\ a.\varphi(a) & a \end{bmatrix}$$

Nous poserons :

$$\begin{bmatrix} \mathcal{C}_X(x) \\ \mathcal{C}_Y(x,y) \end{bmatrix} = \mathcal{C} \begin{bmatrix} x \\ y \end{bmatrix}$$

c'est-à-dire :

(25) $$\mathcal{C}_X(x) = a.x$$

(26) $$\mathcal{C}_Y(x,y) = a\varphi(a)\, x + a\, y$$

Si $\mathcal{H}$ désigne un ensemble de points du plan, nous noterons $\mathcal{C}(\mathcal{H})$ l'ensemble des points transformés par $\mathcal{C}$.
En particulier :
- Si I est un intervalle réel $[\ell,\ell']$, nous noterons $\mathcal{C}(I)$ l'intervalle transformé $[a\ell, a\ell']$
- Si f est une fonction définie sur I, nous appelons $\mathcal{C}(f)$ la fonction définie sur $\mathcal{C}(I)$.
  Plus précisément :

(27) $x \in \mathcal{C}(I)$ $\qquad \mathcal{C}(f)(x) = \mathcal{C}_Y\big(x/a\,,\ f(x/a)\big)$

Lemme 2 : (i) La transformation $\mathcal{C}$ est inversible :

$$\mathcal{C}^{-1} = \begin{bmatrix} 1/a & 0 \\ -\varphi(a)/a & 1/a \end{bmatrix}$$

(ii) Si p est un entier relatif quelconque, on a :

$$\mathcal{C}^p = a^p \begin{bmatrix} 1 & 0 \\ p\varphi(a) & 1 \end{bmatrix}$$

(28) $$\mathcal{C}^p_X(x) = a^p x \qquad \mathcal{C}^p_Y(x,y) = a^p\big(p\varphi(a)x + y\big)$$

(29) $$\mathcal{C}^p_Y(x,y') - \mathcal{C}^p_Y(x,y) = a^p(y'-y)$$

(30) $$y \leqslant y' \Longrightarrow \mathcal{C}^p_X(x,y) \leqslant \mathcal{C}^p_Y(x,y')$$

(iii) Si f est convexe, $\mathcal{C}(f)$ est convexe

(iiii) La fonction $G(x) = k\ x\ \mathrm{Log}\ x$ est invariante par $\mathcal{C}^p$.

Preuve : Il suffit de vérifier par un calcul simple. ▨

Nous allons maintenant "ramener" C_p et E_p sur l'intervalle P_o.

Définition 6 : Nous définissons les "transformées" $\mathcal{E}_p$ et $\mathcal{C}_p$ de E et C de la manière suivante :

$$\mathcal{C}_p = \mathcal{C}^{-p}(C_p)$$

$$\mathcal{E}_p = \mathcal{C}^{-p}(E_p)$$

$\mathcal{E}_p$ et $\mathcal{C}_p$ sont des fonctions réelles sur l'intervalle $[1,a]$
On a donc :

(31) $$\mathcal{E}_p(x) = \frac{E_p(a^p x)}{a^p} - p.x.\varphi(a)$$

(32) $$\mathcal{C}_p(x) = \frac{C_p(a^p x)}{a^p} - p.x.\varphi(a)$$

et inversement :

$$E_p(n) = a^p \mathcal{E}_p(n/a^p) + p\, n\, \varphi(a) \tag{33}$$

$$C_p(n) = a^p \mathcal{C}_p(n/a^p) + p\, n\, \varphi(a) \tag{34}$$

Comme C est définie sur les réels par interpolation linéaire entre les entiers, il en résulte que $\mathcal{C}_p$ est une ligne brisée constituée de segments de droite sur les intervalles $\left[\frac{n}{a^p}, \frac{n+1}{a^p}\right]$ $\left(n \text{ et } n+1 \in \left[a^p, a^{p+1}\right]\right)$.
Il en est de même pour $\mathcal{E}_p$.

Le lemme suivant va nous permettre de comparer les fonctions $\mathcal{C}_p$, $\mathcal{E}_p$ et

Lemme 3 :

(i) $\mathcal{C}_p(x) \geq \mathcal{E}_p(x) \geq G(x) = k\, x\, \mathrm{Log}\, x$

(ii) $\mathcal{C}_p(1) = \mathcal{E}_p(1) = G(1) = 0$

$\mathcal{C}_p(a) = \mathcal{E}_p(a) = G(a) = k\, a\, \mathrm{Log}\, a$

(iii) $\mathcal{C}_p$ est convexe

(iiii) $\mathcal{E}_{p+1} \leq \mathcal{E}_p$ et $\mathcal{C}_{p+1} \leq \mathcal{C}_p$

Preuve : Les propositions (i), (ii) et (iii) découlent respectivement de (iii), (iiii), et (i) de la proposition 1. Montrons (iiii).

Calculons $\mathcal{C}_p(x) - \mathcal{C}_{p+1}(x)$:

$$\mathcal{C}_p(x) - \mathcal{C}_{p+1}(x) = \frac{1}{a^{p+1}}\left[a^{p+1} \times \varphi(a) + a.C(a^p x) - C(a^{p+1}\right.$$

Posons $y = a^{p+1}x$; on en déduit :

$$\mathcal{C}_p(x) - \mathcal{C}_{p+1}(x) = \frac{1}{a^{p+1}}\left[y\, \varphi(a) + a\, C(y/a) - C(y)\right]$$

D'après la définition du coût optimal et la formule (2.2), le terme entre crochets est positif ou nul. On a donc bien $\mathcal{C}_{p+1} \leq \mathcal{C}_p$.
On montrerait de même, à l'aide de la formule (10), que $\mathcal{E}_{p+1} \leq \mathcal{E}_p$. ▨

3.5.2. EGALITE ASYMPTOTIQUE DE $\mathcal{C}_p$ ET $\mathcal{E}_p$

Nous voulons montrer que $\mathcal{C}_p$ et $\mathcal{E}_p$ convergent vers une même fonction $\mathcal{E}$, généralement différente de G. L'égalité asymptotique de C et E est donc plus forte que l'égalité asymptotique entre C et G que nous avons mise en évidence au chapitre précédent (voir 2.5.1).

Nous avons besoin, dans ce but, d'une majoration simple de C(n). Pour cela, nous allons construire un arbre à n feuilles qui n'est pas forcément optimal mais dont le coût est facile à calculer. Cette majoration nous sera utile au chapitre suivant.

Définition 7 : Si $n \geq a^{p'+1}$ (p' entier ≥ 0), nous appelons $R_{p'}(n)$ l'arbre à n feuilles suivant : soit p la poche de n ($a^p \leq n < a^{p+1}$). Par définition, on a $p > p'$.
Posons $m = n/a^{p-p'}$.
On a $m \in P_{p'}$. Appelons e_1 et e_2 les abscisses des points anguleux de $E_{p'}$ les plus proches de m ($e_1 \leq m < e_2$). Par définition, $(n - a^{p-p'}.e_1)$ est positif. Appelons q et r le quotient et le reste de la division de $(n - a^{p-p'} e_1)$ par $(e_2 - e_1)$:

$$n - a^{p-p'} e_1 = (e_2 - e_1)q + r \qquad 0 \leq r < e_2 - e_1 \tag{35}$$

Posons :

$$\begin{cases} b_1 = a^{p-p'} - q - 1 \\ b_2 = q \\ n' = e_1 + r \end{cases} \tag{36}$$

L'arbre $R_{p'}(n)$ a la forme suivante :

1) Jusqu'à la profondeur (p-p'-1) comprise, tous les noeuds ont degré a.

2) A la profondeur p-p', il y a $a^{p-p'}$ noeuds qui ont les poids suivants :

$$\begin{cases} b_1 \text{ noeuds engendrant l'arbre optimal de poids } e_1 \\ b_2 \text{ noeuds engendrant l'arbre optimal de poids } e_2 \\ 1 \text{ noeud engendrant l'arbre optimal de poids } n' \end{cases}$$

L'arbre $R_{p'}(n)$ peut être représenté de la manière suivante :

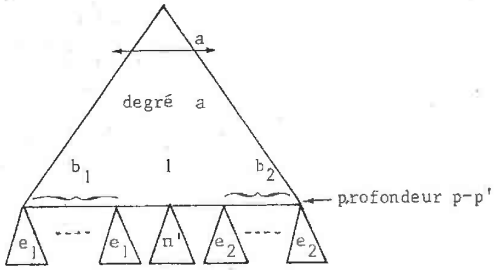


FIGURE 4.

La définition est cohérente : il suffit de vérifier que :

$$(37) \qquad \begin{cases} b_1 \geq 0 \qquad b_2 \geq 0 \\ a^{p-p'} = b_1 + b_2 + 1 \\ n = b_1 e_1 + b_2 e_2 + n' \end{cases}$$

Lemme 4 : Avec les notations de la définition 7 ($m = n/a^{p-p'}$), on a :

$$(38) \qquad \mathcal{C}\left(R_{p'}(n)\right) = (p-p')\varphi(a)n + b_1 E(e_1) + b_2 E(e_2) + C(n')$$

$$(39) \qquad \mathcal{C}\left(R_{p'}(n)\right) = \mathcal{C}_Y^{p-p'}\left(m, E(m)\right) = C(n') - E(n')$$

Preuve : Pour obtenir (38), il suffit de calculer le coût de l'arbre en remarquant que E et C sont égales aux points anguleux e_1 et e_2.
Pour obtenir (39), calculons d'abord E(m). D'après (37), on a :

$$m = a^{p-p'}\left[b_1 e_1 + b_2 e_2 + n'\right]$$

Comme E est une droite entre e_1 et e_2, on en déduit :

$$(40) \qquad E(m) = a^{p'-p}\left[b_1 E(e_1) + b_2 E(e_2) + E(n')\right]$$

Or, d'après (28), on a :

$$(41) \qquad \mathcal{C}_Y^{p-p'}\left(m, E(m)\right) = a^{p-p'}\left[(p-p')\varphi(a)m + E(m)\right]$$

D'après (40) et (41), on a :

$$(42) \qquad \mathcal{C}_Y^{p-p'}\left(m, E(m)\right) = (p-p')\varphi(a)n + b_1 E(e_1) + b_2 E(e_2) + E(n'$$

En soustrayant membre à membre les égalités (38) et (42), on obtient bien la formule (39) : ▨

Définition 8 : Si f et g sont deux fonctions réelles définies sur un même intervalle I, nous noterons $|f-g|$ leur différence maximum

$$|f-g| = \operatorname*{Max}_{x \in I}\left[f(x)-g(x)\right]$$

Lemme 5 : On a : $|\mathcal{C}^p(f-g)| = |f-g|a^p$

Preuve : Cela découle de la formule (29). ▨

Lemme 6 : Si p et p' sont deux entiers positifs ou nuls quelconques vérifiant $p \geq p'$, alors :

$$|\mathcal{C}_p - \mathcal{C}_{p'}| \leq \frac{|\mathcal{C}_{p'} - \mathcal{E}_{p'}|}{a^{p-p'}}$$

Preuve : C'est évident si $p = p'$. Supposons donc $p > p'$. Soit $n \in P_p$. Comme $C(n) \leq \mathcal{C}\left(R_{p'}(n)\right)$, on déduit du lemme 4 que :

(43) $$C(n) - \mathcal{C}_Y^{p-p'}\left(m, E(m)\right) \leq C(n') - E(n')$$

où n' est l'entier de $P_{p'}$ de la définition 7, et $m = n/a^{p-p'}$.
On montre que le premier membre de l'inégalité (43) vaut :

$$a^p\left[\mathcal{C}_p(n/a^p) - \mathcal{E}_{p'}(n/a^p)\right]$$

et que le deuxième membre de (43) est égal à :

$$a^{p'}\left[\mathcal{C}_{p'}(n'/a^{p'}) - \mathcal{E}_{p'}(n'/a^{p'})\right]$$

On déduit donc de (43) que :

$$\mathcal{C}_p(n/a^p) - \mathcal{E}_{p'}(n/a^p) \leq a^{p'-p}\left[\mathcal{C}_{p'}(n'/a^{p'}) - \mathcal{E}_{p'}(n'/a^{p'})\right]$$

D'après la définition 8, on a :

$$\mathcal{C}_{p'}(n'/a^{p'}) - \mathcal{E}_{p'}(n'/a^{p'}) \leq |\mathcal{C}_{p'} - \mathcal{E}_{p'}|$$

Comme $\mathcal{C}_p$ et $\mathcal{E}_{p'}$ sont des segments de droite sur les intervalles $\left[\frac{n}{a^p}, \frac{n}{a^p}\right.$ (n et n+1 $\in P_p$), on en déduit bien :

$$\left|\mathcal{C}_p - \mathcal{E}_{p'}\right| \leq \frac{|\mathcal{C}_{p'} - \mathcal{E}_{p'}|}{a^{p-p'}}$$

Nous sommes maintenant en mesure de montrer que $\mathcal{C}_p$ et $\mathcal{E}_p$ convergent vers la même limite :

Théorème 5 : $\mathcal{E}_p$ et $\mathcal{C}_p$ convergent uniformément vers la même limite $\mathcal{E}$. La fonction $\mathcal{E}$ est continue, bornée, convexe, et vérifie :

(44) $$\mathcal{E}(x) \geq G(x) = k \times \text{Log } x$$

(45) $$\mathcal{E}(1) = G(1) = 0$$

(46) $$\mathcal{E}(a) = G(a) = a.\varphi(x)$$

Preuve : D'après le lemme 3, si x est fixé, $\mathcal{C}_p(x)$ est décroissant avec p et borné inférieurement par G(x). La suite $\mathcal{C}_p(x)$ tend donc vers une limite $\mathcal{C}(x) \geq G(x)$. Les fonctions continues $\mathcal{C}_p$ sont donc bornées inférieurement par G et convergent en décroissant sur l'intervalle fermé $[1,a]$. La convergence est donc uniforme sur cet intervalle et la fonction $\mathcal{C}$ est continue, bornée, supérieure ou égale à G.

De la même manière, $\mathcal{E}_p$ converge uniformément vers une fonction $\mathcal{E}$ continue, bornée, supérieure ou égale à G.

Les fonctions $\mathcal{E}_p$ étant convexes, leur limite $\mathcal{E}$ est aussi convexe.
Les égalités (45) et (46) découlent du lemme 3.(ii).
Reste à montrer que $\mathcal{C} = \mathcal{E}$. Comme $\mathcal{C}_p \geq \mathcal{E}_p$ (voir lemme 3.(i)), on en déduit que $\mathcal{C} \geq \mathcal{E}$. Il nous suffit donc de montrer que la différence $\mathcal{C}(x) - \mathcal{E}(x)$ peut être rendue aussi petite que possible. Cette différence peut s'écrire :

(47) $$\mathcal{C}(x) - \mathcal{E}(x) = \left[\mathcal{C}(x) - \mathcal{C}_p(x)\right] + \left[\mathcal{C}_p(x) - \mathcal{E}_{p'}(x)\right] + \left[\mathcal{E}_{p'}(x) - \mathcal{E}(x)\right]$$

p et p' sont deux entiers vérifiant $p \geq p' \geq 0$. D'après le lemme précédent, on a :

(48) $$\mathcal{C}_p(x) - \mathcal{E}_{p'}(x) \leq \frac{|\mathcal{C}_{p'} - \mathcal{E}_{p'}|}{a^{p-p'}}$$

Si l'on choisit p et p' suffisamment grands, le premier et le troisième terme entre crochets peuvent être rendus aussi petits que possible. D'après (48), pour rendre le deuxième terme aussi petit que possible, il suffit que la différence (p-p') soit suffisamment grande. Finalement, la différence $\mathcal{C}(x) - \mathcal{E}(x)$ peut être rendue aussi petite que possible. On a donc bien $\mathcal{C} = \mathcal{E}$.

CHAPITRE 4

CAS OU IL Y A UN SEUL DEGRE-LIMITE

Ce chapitre et le chapitre suivant sont consacrés à l'étude des arbres optimaux à poids égaux quand la fonction de coût est à degré borné, c'est-à-dire quand il existe pour tout n un arbre optimal de poids n dont les degrés sont au plus égaux à une constante d_{max}. Il faut distinguer deux cas, suivant que la fonction de coût possède un seul ou plusieurs degré-limites, c'est-à-dire un ou plusieurs entiers vérifiant :

$$\frac{\varphi(d)}{\text{Log } d} \leqslant \frac{\varphi(n)}{\text{Log } n} \qquad \text{pour tout } n.$$

Nous avons vu au chapitre précédent que les transformées $\mathcal{C}_p$ et $\mathcal{E}_p$ du coût optimal C et de son enveloppe convexe E convergent vers la même limite $\mathcal{E}$. La convergence se fait de façon très différente suivant qu'il y a un seul ou plusieurs degré-limites. Dans le premier cas, $\mathcal{E}_p = \mathcal{E}$ à partir d'un certain rang ; dans le deuxième cas, on a $\mathcal{E} = G\left(G(x) = k \times \text{Log } x\right)$. Il en résulte que les propriétés des arbres optimaux sont très différentes. En particulier, nous montrerons que dans le premier cas, les arbres optimaux et leurs coûts sont calculables en temps fini.

Nous étudierons au chapitre 5 le cas où il y a plusieurs degré-limites. Dans ce chapitre, nous supposons donc que la fonction de coût possède un degré maximum d_{max} et un seul degré-limite a. En fait, les résultats de ce chapitre restent valables s'il y a plusieurs degré-limites qui sont des puissances entières d'un même nombre entier. Les démonstrations s'étendent facilement à ce cas.

Rappelons que :

(1) $$k = \frac{\varphi(a)}{\text{Log } a} \leqslant \frac{\varphi(n)}{\text{Log } n}$$

(2) $$a \leqslant d_{max}$$

(3) $$G(n) \leqslant E(n) \leqslant C(n) \qquad G(x) = k \times \text{Log } x$$

Nous allons d'abord montrer que l'enveloppe convexe E a une forme régulière à partir d'un certain rang, c'est-à-dire que la limite $\mathcal{E}$ des transformées $\mathcal{E}_p$ de E est atteinte à partir d'un certain rang p_1. Nous en déduirons un procédé de calcul de E en temps fini et montrerons que E est une excellente approximation de C : en effet, C et E ne diffèrent que d'une constante. Nous donnerons également une heuristique quasi-optimale qui calcule des arbres dont le coût ne diffère du coût optimal que d'une constante.

Au paragraphe 3, nous nous intéresserons aux degrés des arbres optimaux. Nous avons vu au chapitre 2, que les arbres a-parfaits sont optimaux et, plus généralement, que les arbres ont tendance à avoir pour degré le degré-limite a. Le théorème du degré-limite va préciser cette notion : à partir d'un certain poids N_2, il existe toujours un arbre optimal dont le degré au sommet est le degré-limite a. C'est ce qui justifie l'expressi "degré-limite".

Au paragraphe 5, nous mettrons en évidence des propriétés des arbres optimaux qui nous permettrons de calculer leur forme et leur coût en temps fini.

Au paragraphe 6, nous verrons qu'il existe toujours des arbres optimaux dont la différence de profondeur des feuilles est bornée par une constant Mais, contrairement à la conjecture de KNUTH ([13], p. 377, Ex. 9), cette borne n'est pas toujours égale à un. Autrement dit, il n'existe pas toujours un arbre optimal dont la différence des profondeurs des feuilles n'excède pas un.

4.1. FORME ASYMPTOTIQUE DE E

Nous voulons montrer que les transformées $\mathcal{E}_p$ de E sont égales à leur limite $\mathcal{E}$ à partir d'un certain rang p_1. Nous en déduirons que les fonctions E_p sont régulières, c'est-à-dire qu'elles se déduisent l'une de l'autre par une transformation $\mathcal{P}^q$ (voir définition 3.5 La définition suivante va nous permettre d'établir des correspondances entre les E_p :

Définition 1 : Pour p et i entiers ($p \geqslant 0$, $i \geqslant 1$), nous définissons :

$S_{p,i}$: le i-ième segment de E_p

$e_{p,i}$ et $e'_{p,i}$: les abscisses des points anguleux qui sont les extrémités de $S_{p,i}$

$I_{p,i}$: le p-ième intervalle de E_p ; c'est l'intervalle réel $I_{p,i} = \left[e_{p,i}, e'_{p,i} \right]$

$\left| I_{p,i} \right| = e'_{p,i} - e_{p,i}$

$D_{p,i}(x)$: l'équation de la droite prolongeant $S_{p,i}$.

Si $n \in \left[e_{p,i}, e'_{p,i} \right[$, on dit que $I_{p,i}$ est l'intervalle de n et que $S_{p,i}$ est le segment de n

Notons qu'on a toujours $E(x) \geqslant D_{p,i}(x)$, puisque E est convexe.

4.1.1. REGULARITE DE E

Nous allons d'abord traiter le cas où les seuls points anguleux de E sont ceux d'abscisses a^p.

Lemme 1 : Si à chaque point anguleux correspond au moins un arbre régulier ayant un noeud de degré a, alors les points anguleux ont pour abscisses a^p et l'enveloppe convexe E est la fonction h_a.

Preuve : La fonction h_a a été définie en 1.17 : elle est constituée des segments joignants les points $\left(a^p, C(a^p)\right)$ et $\left(a^{p+1}, C(a^{p+1})\right)$ (voir figure 2.1). Soit n l'abscisse d'un point anguleux. D'après l'hypothèse, il lui correspond au moins un arbre régulier ayant un noeud de degré a. Par filtrages successifs, on peut transformer cet arbre en un arbre régulier, également optimal, mais de degré au sommet a. D'après le théorème 4, les sous-arbres principaux ont tous même poids et correspondent eux-mêmes à des points anguleux. On a donc $n = a.n'$, où n' est l'abscisse d'un point anguleux. En appliquant ce raisonnement à n' et en continuant ainsi, on voit finalement que $n = a^p$. ▨

Théorème 6 : A partir d'une certaine poche p_1, E_p a même forme que E_{p_1} à une transformation $\mathcal{C}$ près. Plus précisément :

(4) $$\forall p \geqslant p_1 \quad \begin{cases} E_p = \mathcal{C}^{p-p_1}(E_{p_1}) \\ \mathcal{E}_p = \mathcal{E}_{p_1} = \mathcal{E} \end{cases}$$

p_1 est la poche contenant le point anguleux d'abscisse N_1 la plus grande de l'ensemble $\mathcal{P}$ suivant : $\mathcal{P}$ est l'ensemble des points anguleux pour lesquels tous les arbres réguliers correspondants n'ont aucun noeud de degré a.

Preuve : Si $\mathcal{P}$ est réduit au seul point anguleux (1,0), alors, d'après le lemme précédent, l'enveloppe convexe est la fonction h_a. On vérifie facilement que dans ce cas $E_p = \mathcal{C}^p(E_o)$, et que pour tout p, $\mathcal{E}_p = \mathcal{E} = E_o$. La fonction E_o est le segment de droite joignant les points (1,0) et $\left(a, a\,\varphi(a)\right)$.

Dans le cas général, montrons d'abord que les abscisses des points anguleux de $\mathcal{P}$ sont bornées. D'après la proposition 3.5, à un point anguleux de $\mathcal{P}$ correspond toujours un arbre régulier de décomposition $(d_1^{\ell_1}, \ldots, d_t^{\ell_t})$, avec $2 \leqslant d_i \leqslant d_{max}$. Par hypothèse, les d_i sont différents de a. D'après la proposition 3.4, les abscisses des points anguleux de $\mathcal{P}$ sont donc bornées par :

(6) $$N_1' = \prod_{\substack{2 \leqslant d \leqslant d_{max} \\ d \neq a}} d^{L(d)}$$

Appelons donc N_1 l'abscisse de $\mathcal{P}$ la plus grande et p_1 la poche correspondante.

Comparons $\mathcal{E}_{p_1+1}$ et $\mathcal{E}_{p_1}$. Considérons un point anguleux de E_{p_1+1} d'abscisse e. Comme $e > N_1$, il existe une décomposition de e comportant le degré a. D'après le corollaire 3.1, e/a est donc l'abscisse d'un point anguleux de E_{p_1} et on a :

$$E_{p_1+1}(e) = \varphi(a)e + a.E_{p_1}(e/a)$$

A chaque point anguleux de E_{p_1+1} correspond donc par $\mathcal{C}^{-1}$ un point anguleux de E_{p_1}. On en déduit que les points anguleux de $\mathcal{E}_{p_1+1}$ sont également des points anguleux de $\mathcal{E}_p$. Comme $\mathcal{E}_{p_1}$ et $\mathcal{E}_{p_1+1}$ sont des lignes brisées convexes, on a donc :

$$\mathcal{E}_{p_1+1} \geqslant \mathcal{E}_p$$

Mais, d'après le lemme 3.3, on a $\mathcal{E}_{p_1+1} \leqslant \mathcal{E}_{p_1}$, on en déduit que $\mathcal{E}_{p_1+1} = \mathcal{E}_p$. On montrerait de même que $\mathcal{E}_{p_1+2} = \mathcal{E}_{p_1+1}$, et, plus généralement, l'égalité (5). La formule (4) en résulte d'après les définitions de $\mathcal{C}$ et $\mathcal{E}_p$. ▨

D'après ce théorème, une fois que l'on a déterminé E_{p_1}, on en déduit E_p pour $p > p_1$ ainsi que la limite $\mathcal{E}$. Plus précisément :

Corollaire 1 : Si $n \in P_p$, avec $p \geqslant p_1$, on a:

(7) $$E_p(n) = (p-p_1)\,\varphi(a)n + a^{p-p_1} E_{p_1}(n/a^{p-p_1})$$

(8) $$\mathcal{E}(x) = (E_{p_1}(a^{p_1}x))/a^{p_1} - p_1 x\,\varphi(a)$$

(9) $$E(n) = p\,\varphi(a)n + a^p.\mathcal{E}(n/a^p)$$

Preuve : Ces formules découlent directement du théorème précédent et des définitions de $\mathscr{E}$ et $\mathcal{E}$. ▨

Sur la figure suivante, nous avons représenté très schématiquement les fonctions $\mathcal{E}$, E_{p_1}, $E_{p'}$ et E_p, pour $p > p' > p_1$.

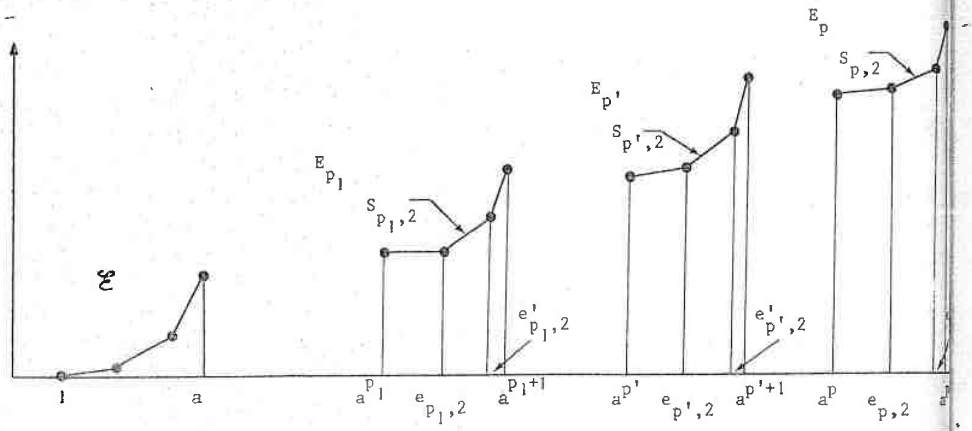


FIGURE 1.

Ce dessin ne respecte pas les échelles. Nous avons représenté le segment $S_{p_1,2}$, c'est-à-dire le deuxième segment de E_{p_1}. Comme les fonctions E_p, $E_{p'}$ et E_{p_1} se déduisent l'une de l'autre par une transformation $\mathscr{E}$, ces trois fonctions ont le même nombre de segments et la transformation $\mathscr{E}$ fait correspondre le deuxième segment de E_{p_1} aux deuxièmes segments de E_p et $E_{p'}$. Nous sommes donc amenés à définir des "segments" correspondants :

Définition 2 : Si $I_{p,i}$ est l'intervalle de n (c'est-à-dire $e_{p,i} \leqslant n < e'_{p,i}$), nous appelons $I_{p',i}$ l'intervalle correspondant de n dans la poche p' (p et p' $\geqslant p_1$). Plus généralement :

$n \in I_{p,i}$	$I_{p,i}$	$S_{p,i}$	$e_{p,i}$	$e'_{p,i}$	$D_{p,i}$
correspondants dans $P_{p'}$	$I_{p',i}$	$S_{p',i}$	$e_{p',i}$	$e'_{p',i}$	$D_{p',i}$

Corollaire 2 : Si p et p' sont des entiers plus grands ou égaux à p_1, on a

(i) $S_{p,i} = \mathscr{E}^{p-p'}(S_{p',i})$

$e_{p,i} = a^{p-p'} \cdot e_{p',i}$

$e'_{p,i} = a^{p-p'} \cdot e'_{p',i}$

$|I_{p,i}| = a^{p-p'} |I_{p',i}|$

$D_{p,i} = \mathscr{E}^{p-p'}(D_{p',i})$

(ii) Si $n \in I_{p,i}$ et $p \geqslant p'$, alors :

(10) $$E_p(n) = (p-p')\varphi(a)n + a^{p-p'} \cdot E_{p'}(n/a^{p-p'})$$

(11) $$\begin{cases} E_p(n) = (p-p')\varphi(a)n + \sum_{1 \leqslant n_j \leqslant a^{p-p'}} \left[D_{p',i}(n_j) \right] \\ \sum_{1 \leqslant n_j \leqslant a^{p-p'}} n_j = n \end{cases}$$

Preuve : Toutes ces formules, sauf (11), sont des conséquences du théorème 6 et du corollaire précédent. Pour la formule (11), il suffit de remarquer que $D_{p',i}$ est une droite, et qu'en conséquence :

(12) $$\sum_{1 \leqslant n_j \leqslant a^{p-p'}} \left[D_{p',i}(n_j) \right] = a^{p-p'} \cdot D_{p',i}(n/a^{p-p'})$$

Comme $e_{p',i} \leqslant n/a^{p-p'} < e'_{p',i}$, on a :

(13) $$D_{p',i}(n/a^{p-p'}) = E_{p'}(n/a^{p-p'})$$

En substituant (12) et (13) dans la formule (11), on obtient bien (10).

▨

Le théorème 6 a comme conséquence immédiate la propriété suivante : si $n \in I_{p,i}$, alors le coût optimal est égal à E(n) chaque fois que $(n-e_{p,i})$ est un multiple de $|I_{p_1,i}|$. Pour cela, il suffit de montrer que le coût de l'arbre $R_{p_1}(n)$ est égal à E(n). Montrons d'abord que si n est grand, le coût de l'arbre $R_{p_1}(n)$ (voir définition 3.7) se calcule simplement :

Corollaire 3 : Si $n \geqslant a^{p_1+1}$, alors le coût de l'arbre $R_{p_1}(n)$ vérifie :

$$\mathcal{C}\left(R_{p_1}(n)\right) - E(n) = C(n') - E(n') \tag{14}$$

où n' est défini en 3.7.

Preuve : D'après le lemme 3.4, si l'on pose $m = n/a^{p-p_1}$, le coût de cet arbre vérifie :

$$\mathcal{C}\left(R_{p_1}(n)\right) - \mathcal{C}_Y^{p-p_1}\left(m,E(m)\right) = C(n') - E(n')$$

Un calcul simple montre que :

$$\mathcal{C}_Y^{p-p_1}\left(m,E(m)\right) = (p-p_1)\varphi(a)n + a^{p-p_1}.E\left(n/a^{p-p_1}\right)$$

D'après le corollaire 2, formule (10), le deuxième membre est égal à $E_p(n)$. On en déduit bien (14).

▨

Corollaire 4 : Si $n \in I_{p,i}$ avec $p > p_1$, alors $|I_{p,i}|/a$ est entier et on a :

(i) $C(n) = E(n)$ pour $n = e_{p,i} + j|I_{p,i}|/a \quad (0 \leqslant j \leqslant a)$

(ii) $C(n) = E(n)$ pour $n = e_{p,i} + j|I_{p_1,i}| \quad (0 \leqslant j \leqslant a^{p-p_1})$

Preuve : D'abord $|I_{p,i}|/a$ est entier puisque d'après le corollaire 2, on a :

$$|I_{p,i}|/a = |I_{p-1,i}| \quad \text{pour} \quad p-1 \geqslant p_1$$

La proposition (i) est une conséquence de (ii) puisque :

$$|I_{p,i}|/a = |I_{p_1,i}|.a^{p-p_1-1}$$

Pour montrer (ii), il suffit de montrer que le coût de l'arbre $R_{p_1}(n)$ est égal à E(n). D'après la définition 3.7, n' est égal à :

$$n' = e_{p_1,i} + \text{reste}\left[\left(n - a^{p-p_1}.e_{p_1,i}\right),\left(e'_{p_1,i} - e_{p_1,i}\right)\right]$$

ce qui s'écrit :

$$n' = e_{p_1,i} + \text{reste}\left[\left(n-e_{p,i}\right),|I_{p_1,i}|\right]$$

Par hypothèse, le reste est donc nul et $n' = e_{p_1,i}$. Comme $e_{p_1,i}$ est l'abscisse d'un point anguleux, on a :

$$C(n') = E(n')$$

On en déduite que $\mathcal{C}\left(R_{p_1}(n)\right) = E(n)$ d'après le corollaire précédent. L'arbre $R_{p_1}(n)$ est donc optimal et on a bien : C(n) = E(n).

▨

4.1.2. CALCUL DE E ET $\mathcal{E}$ EN TEMPS FINI

Les résultats précédents nous permettent de donner un procédé de calcul de E et $\mathcal{E}$ en temps fini :

Proposition 1 : Les enveloppes convexes E et $\mathcal{E}$ sont calculables en temps fini de la manière suivante :

1) Déterminer N'_1 par la formule (6) et la poche p'_1 correspondante. Calculer l'enveloppe convexe E sur l'intervalle $\left[1, a^{p'_1+1}\right]$ à l'aide du corollaire 3.2.

2) Identifier la poche p_1 ($p_1 \leqslant p'_1$) à partir de laquelle E est régulière.

3) Pour $p \geq p_1$, E_p est donné par la formule :

$$E_p(n) = (p-p_1)\varphi(a)n + a^{p-p_1} \cdot E_{p_1}\left(n/a^{p-p_1}\right)$$

4) l'enveloppe convexe asymptotique est donnée par :

$$\mathcal{E}(x) = \frac{E_{p_1}\left(a^{p_1}.x\right)}{a^{p_1}} - p_1.x.\varphi(a)$$

Preuve : Comme N_1' est borné, la phase 1 prend un temps fini. Il en est de même pour les phases 2, 3 et 4 (les formules sont celles du corollaire

4.2. FORME ASYMPTOTIQUE DE C

Nous avons vu au théorème 5 que les transformées $\mathcal{E}_p$ et $\mathcal{E}_p$ tendent vers la même limite. Mais cela ne signifie pas que C_p et E_p ne s'éloignent pas infiniment l'une de l'autre pour certaines valeurs de n.

Nous montrons ici une propriété beaucoup plus forte, à savoir que C et E ne diffèrent que d'une constante. Nous en déduirons un procédé heuristique calculant des arbres quasi-optimaux en temps fini.

4.2.1. LA DIFFERENCE C-E EST BORNEE

Théorème 7 : La différence $C(x) - E(x)$ est bornée :

(15) $\forall$ x réel ≥ 1 $\quad C(x)-E(x) \leq B$

$$\text{avec } B = \max_{1 \leq n \leq a^{p_1+1}} \left[C(n)-E(n)\right]$$

Preuve : Comme E et C sont des segments de droite entre les valeurs entières, il suffit de montrer (15) pour x entier. Soit donc $n > a^{p_1+1}$. Considérons l'arbre $R_{p_1}(n)$ de la définition 3.7. D'après le corollaire 3, son coût vérifie :

$$\mathcal{E}\left(R_{p_1}(n)\right) - E(n) = C(n')-E(n')$$

avec $n' \in P_{p_1}$.

Par hypothèse, on a donc $C(n') - E(n') \leq B$. Comme d'autre part, $C(n) \leq \mathcal{E}\left(R_{p_1}(n)\right)$, on en déduit bien (15).

Ce théorème nous montre que l'enveloppe convexe E est une excellente approximation du coût optimal ; contrairement à la fonction $G(x) = k \times \text{Log } x$ qui pouvait s'éloigner infiniment de C(x) (voir l'exemple 2.6.3).
Aux paragraphes 4 et 5, nous déduirons de cette propriété des indications très précises sur la forme des arbres optimaux. Pour l'instant, nous allons en déduire l'heuristique suivante.

4.2.2. HEURISTIQUE QUASI-OPTIMALE CALCULABLE EN TEMPS FINI

Définition 3 : L'heuristique $\mathcal{H}$ calcule les arbres $\mathcal{A}(n)$ suivants :

1) Calculer E (voir proposition 1) etles arbres réguliers optimaux correspondants aux points anguleux de E_{p_1}.

2) Pour $1 \leq n \leq a^{p_1+1}$: si n est un point anguleux de E_{p_1}, l'arbre $\mathcal{A}(n)$ est l'arbre régulier optimal correspondant ; son coût est donc égal à E(n).
Dans les autres cas, on calcule des arbres, optimaux ou non, par n'importe quelle méthode.

3) Pour $n > a^{p_1+1}$: l'arbre $\mathcal{A}(n)$ a même forme que l'arbre $R_{p_1}(n)$ défini en 3.7, sauf que l'on remplace le sous-arbre optimal de poids n' par l'arbre $\mathcal{A}(n')$ calculé à la phase 2.

Proposition 2 : L'heuristique $\mathcal{H}$ possède les propriétés suivantes :

(i) Elle est calculable en temps fini.
(ii) Si $n > a^{p_1+1}$, elle calcule des arbres de coût :

(16) $$\mathcal{C}\left(\mathcal{A}(n)\right) = E(n) + \mathcal{C}\left(\mathcal{A}(n')\right) - E(n')$$

avec $n' \in P_{p_1}$ (n'a été défini en 3.7).

(iii) La différence entre le coût des arbres $\mathcal{A}(n)$ et le coût optimal est bornée par une constante :

(17) $$(\forall n) \quad \mathcal{C}\left(\mathcal{A}(n)\right) - C(n) \leqslant B'$$

$$\text{avec } B' = \operatorname{Max}_{1 \leqslant n \leqslant a^{p_1+1}} \left[\mathcal{C}\left(\mathcal{A}(n)\right) - E(n) \right]$$

Preuve : (i) Nous avons vu à la proposition 1 que E et p_1 sont calculables en temps fini. Comme p_1 est fini, la phase 2 prend un temps fini quel que soit le procédé utilisé. Si $n > a^{p_1+1}$, le calcul de la forme de l'arbre est immédiate, d'après la définition 3. Le calcul du coût de l'arbre est immédiat par la formule (16). Cette heuristique est donc bien calculable en temps fini.

(ii) Un calcul identique à celui du corollaire 3 démontre la formule (16).

(iii) Supposons $n > a^{p_1+1}$. Comme $C \geqslant E$, on a :

$$\mathcal{C}\left(\mathcal{A}(n)\right) - C(n) \leqslant \mathcal{C}\left(\mathcal{A}(n)\right) - E(n)$$

D'après (16), on en déduit :

$$\mathcal{C}\left(\mathcal{A}(n)\right) - C(n) \leqslant \mathcal{C}\left(\mathcal{A}(n')\right) - E(n')$$

Comme $n' \leqslant a^{p_1+1}$, le deuxième membre de l'inégalité est inférieur ou égal à B', ce qui entraîne la formule (17).

▨

Cette heuristique est quasi-optimale puisque le coût de ses arbres ne diffère du coût optimal que par une constante. Naturellement, plus les arbres calculés à la phase 2 sont bons, plus petite est la constante En particulier, si à la phase 2, on prend les arbres optimaux, B' sera égale à la constante B du théorème 7.

4.3. THEOREME DU DEGRE-LIMITE

Nous nous proposons de montrer qu'asymptotiquement le degré au sommet des arbres tend vers le degré-limite a. Plus précisément, nous allons voir que si n est assez grand, il existe toujours un arbre optimal de poids n dont le degré au sommet est égal à a. C'est pourquoi, il nous faut introduire la notion de degré possible pour un arbre.

Définition 4 : On dit que le degré d est possible pour n s'il existe un arbre optimal à n feuilles de degré au sommet d. Sinon, on dit que le degré d est impossible pour n.
Par extension, on dit que le degré d est possible pour un arbre si le degré au sommet de cet arbre est d ou s'il existe un arbre équivalent de degré au sommet d. Sinon, on dit que le degré d est impossible pour cet arbre.

Proposition 3 : Si le degré d est impossible pour un arbre, alors le degré d est impossible pour au moins un des ses sous-arbres principaux.

Preuve : La preuve se fait par l'absurde. Soit T un arbre de degré d impossible. Si le degré d était possible pour tous ses sous-arbres principaux, on pourrait les remplacer dans T par des arbres équivalents de degré d. On obtiendrait ainsi un arbre T' équivalent à T. En filtrant le sommet de T', on obtiendrait un arbre T" équivalent à T, donc optimal, mais de degré au sommet d, ce qui contredit l'hypothèse.

▨

Lemme 2 : Soit un arbre optimal de poids n. Soient d son degré et $n_1, \ldots, n_d$ les poids de ses sous-arbres principaux. Si une droite D quelconque vérifie :

(18) $$D \leqslant E \quad \text{et} \quad C(n) \leqslant D(n) + b$$

alors la droite D' suivante :

(19) $$D'(x) = D(dx)/d - x\varphi(d)$$

vérifie :

(20) $$D' \leqslant E \quad \text{et} \quad \sum_{1 \leqslant i \leqslant d} \left[C(n_i) - D'(n_i) \right] \leqslant b$$

Preuve : L'inégalité $D \leqslant E$ signifie $D(x) \leqslant E(x)$ pour $x \geqslant 1$.
Cette inégalité est possible puisque E est convexe. La définition de D' peut s'écrire :

$$D(x) = \varphi(d)x + d.D'(x/d) \tag{21}$$

D'autre part, on sait, d'après la proposition 3.2, que :

$$E(x) \leqslant \varphi(d)x + d\,E(x/d) \tag{22}$$

Comme $D \leqslant E$, on déduit de (21) et (22) que :

$$\varphi(d)x + d.D'(x/d) \leqslant \varphi(d)x + d.E(x/d)$$

ce qui entraîne bien $D' \leqslant E$.

Comme D' est une droite, on peut écrire, d'après (21) :

$$D(n) = \varphi(d)\,n + \sum_{1\leqslant i\leqslant d} D'(n_i) \tag{23}$$

D'autre part, l'arbre de départ étant optimal, on a:

$$C(n) = \varphi(d)\,n + \sum_{1\leqslant i\leqslant d} C(n_i) \tag{24}$$

Par hypothèse, $C(n) \leqslant D(n)+b$. On a donc, d'après (23) et (24) :

$$\sum_{1\leqslant i\leqslant d} C(n_i) \leqslant \sum_{1\leqslant i\leqslant d} D'(n_i) + b$$

ce qui est équivalent à (20). ▨

Lemme 3 : Dans un arbre optimal, les degrés des noeuds du chemin d'une feuille quelconque vérifient :

$$\left[\sum_{0\leqslant i\leqslant p-1} \varphi(d_i) - B\right].m \leqslant E(m) \qquad m = \prod_{o\leqslant i\leqslant p-1} d_i \tag{25}$$

B est la constante du théorème 6 ; p est la profondeur de la feuille ; d_i est le degré du noeud situé à la profondeur i sur le chemin.

Preuve : Appelons n_i $(0 \leqslant i \leqslant p)$ le poids du noeud situé à la profondeur i. Remarquons que n_o est le poids de l'arbre et que $n_p = 1$. Soit une droite D_o quelconque vérifiant :

$$D_o \leqslant E \quad \text{et} \quad C(n_o) \leqslant D_o(n_o) + B$$

Remarquons qu'une telle droite existe toujours : d'après le théorème 6, on peut prendre la droite prolongeant le segment de E correspondant à n_o. Considérons la droite D_1 suivante :

$$D_1(x) = D_o(d_o x)/d_o - x\varphi(d_o)$$

D'après le lemme précédent, on a:

$$D_1 \leqslant E \quad \text{et} \quad C(n_1) \leqslant D(n_1)+B$$

On peut donc réappliquer le lemme précédent à tous les noeuds du chemin. On obtient une suite de droite D_i ayant les propriétés suivantes :

$$\text{pour } 1\leqslant i\leqslant p \quad \begin{cases} D_i \leqslant E \quad \text{et} \quad C(n_i) \leqslant D_i(n_i) + B \\ D_i(x) = D_{i-1}(d_{i-1}x)/d_{i-1} - \varphi(d_i)x \end{cases} \tag{26}$$

Comme $m = \prod_{0\leqslant i\leqslant p-1} d_i$, on en déduit que :

$$D_o(x) = x \sum_{0\leqslant i\leqslant p-1} \varphi(d_i) + m.D_p(x/m)$$

Ecrivons que $D_o(m) \leqslant E(m)$:

$$m\left[\sum_{0\leqslant i\leqslant p-1} \varphi(d_i) + D_p(1)\right] \leqslant E(m) \tag{27}$$

D'après (26), nous savons que $D_p(n_p) \geqslant C(n_p) - B$. Comme $n_p = 1$, on en déduit que $D_p(1) \geqslant -B$. En substituant dans (27), on obtient bien (25).

▨

Avant de montrer le théorème du degré-limite, nous mettons en évidence la relation suivante entre le poids des arbres et leurs profondeurs.

Proposition 4 : Les poids n des arbres optimaux dont tous les noeuds ont des degrés inférieurs ou égaux à d_{max} et qui possèdent une feuille à la profondeur p sont bornés supérieurement par la fonction de p suivante :

$$n \leqslant e^{(p+1).\varphi(d_{max})/k} \tag{28}$$

Preuve : Considérons un tel arbre optimal. Appelons n son poids et f la feuille à la profondeur p. Soit f' une feuille quelconque, f' pouvant être confondue avec f. Appelons u le père de f'. Comme $w(u) > w(f)$, d'après le corollaire 1.1, les coûts partiels vérifient $c(u) \leqslant c(f)$. Comme par définition, $c(f') = \varphi\left(d(u)\right) + c(u)$, on en déduit :

$$c(f') \leqslant \varphi\left(d(u)\right) + c(f)$$

La fonction φ étant croissante, on a : $\varphi\left(d(w)\right) \leqslant \varphi(d_{max})$ et $c(f) \leqslant p.\varphi(d_{max})$, puisque la feuille f est à la profondeur p. D'où :

$$c(f') \leqslant (p+1).\varphi(d_{max})$$

Cette dernière inégalité étant valable pour toute feuille f' de l'arbre, on en déduit, d'après la formule des coûts partiels :

$$C(n) \leqslant n(p+1).\varphi(d_{max})$$

Comme d'après la proposition 2.4, on a $k\ n \operatorname{Log} n \leqslant C(n)$, on en déduit :

$$k\ n \operatorname{Log} n \leqslant n(p+1)\ \varphi(d_{max})$$

Ce qui entraîne la formule (28).

▨

Théorème du "degré-limite" (8) : A partir d'un certain poids N_2, il existe au moins un arbre optimal dont le degré au sommet est le degré-limite a.

Preuve : Nous allons majorer les valeurs de n pour lesquelles le degré a est impossible. Soit n une telle valeur. La fonction de coût possédant un degré maximum d_{max}, il existe un arbre de poids n dont tous les noeuds ont un degré plus petit ou égal à d_{max}. Par hypothèse, son degré au sommet d_o est différent de a, et, d'après la proposition 3, le degré a est impossible pour un de ses sous-arbres principaux. Appelons d_1 le degré au sommet de ce sous-arbre. En réappliquant la proposition 3, on construit un chemin de l'arbre dont tous les noeuds ont un degré différent de a. D'après le lemme 3, si B désigne la constante du théorème 7 et p la profondeur du chemin, on a :

$$m.\left[\sum_{0\leqslant i\leqslant p-1} \varphi(d_i) - B\right] \leqslant E(m) \tag{29}$$

avec :

$$m = \prod_{0\leqslant i\leqslant p-1} d_i \tag{30}$$

Comme $E(m) \leqslant C(m)$ (proposition 3.1) et $C(m) < k\ m \operatorname{Log} m + \varphi(a)\ m$ (proposition 2.4), on en déduit :

$$E(m) < k\ m \operatorname{Log} m + \varphi(a)\ m \tag{31}$$

En combinant (29), (30) et (31), on obtient :

$$\sum_{0\leqslant i\leqslant p-1} \left[\varphi(d_i) - k \operatorname{Log} d_i\right] < \varphi(a) + B \tag{32}$$

Mais les d_i étant tous différents de a, les termes entre-crochets sont tous strictement positifs.

Si nous posons :

$$\mu = \min_{\substack{2 \leq d \leq d_{max} \\ d \neq a}} \left[\varphi(d_i) - k \operatorname{Log} d_i \right]$$

l'inégalité (32) entraîne :

$$p < \left(\varphi(a)+B \right) / \mu$$

D'après le lemme précédent, on en déduit :

$$n > e^{\left[\frac{\varphi(d_{max})}{k} \left(1 + \frac{\varphi(a)+B}{\mu} \right) \right]} \qquad (33)$$

Donc si n dépasse cette valeur, le degré-limite a sera toujours possible pour n : il existera toujours un arbre de degré au sommet a. C'est ce qui justifie l'expression "degré-limite".

Définition 5 : Nous appelons N_2 la plus petite valeur de n à partir de laquelle le degré-limite a est toujours possible. Nous appelons p_2 la poche de N_2.

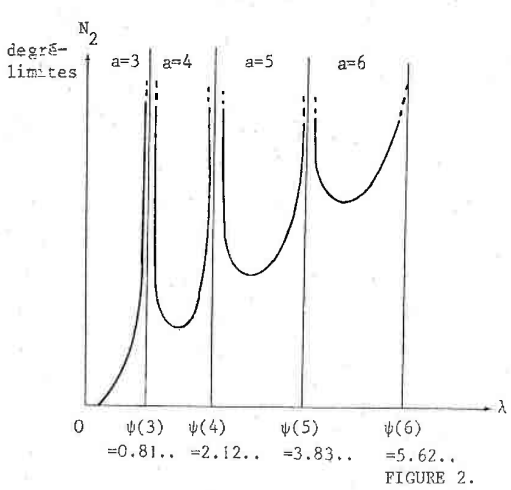
Il faut bien comprendre le sens de ce théorème : il ne signifie pas qu'à partir d'un certain rang tous les arbres optimaux ont un degré au sommet a. Nous verrons sur l'exemple suivant qu'il existe des arbres optimaux de degré au sommet différent de a, même si n est arbitrairement grand.

4.4. EXEMPLE DES FONCTIONS DE COUT $\varphi(d) = d+\lambda$

D'après le théorème précédent, la valeur de N_2, rang à partir duquel le degré-limite a est possible, ne dépend que de la donnée de la fonction de coût, c'est-à-dire de φ et, dans notre cas, uniquement de λ. Nous allons d'abord voir comment varie N_2 avec λ.

4.4.1. VALEURS DE N_2

Nous donnons ci-dessous l'allure générale de la courbe N_2 en fonction de λ. Nous détaillons plus pour $\lambda \in \left[0, \psi(3) \right]$.



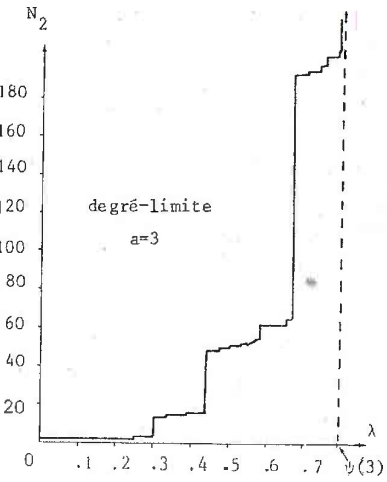


FIGURE 2.

Les valeurs $\psi(3)$, $\psi(4)$, ... sont les valeurs de λ pour lesquelles il y a plusieurs degré-limites (voir 2.6.2). On remarque que si λ est proche d'une valeur $\psi(d)$, N_2 tend vers l'infini. Ce cas sera étudié au chapitre suivant. Voici quelques valeurs de N_2 :

λ	N_2
0	3
1/4	5
3/10	14
1/3	16
7/18	17
4/9	50
9/19	51
1/2	52

Par exemple :
si $0 < \lambda \leq 1/4$ alors $N_2 = 5$

Remarque :
si $\lambda > 0.797$ alors $N_2 \geq 5000$

TABLEAU I.

4.4.2. CAS $\varphi(d) = d$

Nous avons vu en 2.6.3 que KNUTH ([13] , Théorème L) avait donné une méthode simple de calcul des arbres optimaux et de leurs coûts. Nous pouvons maintenant montrer ce résultat par une autre méthode : Considérons l'ensemble $\mathcal{P}$ des points anguleux pour lesquels tous les arbres réguliers correspondants n'ont pas de noeud de degré a = 3 (voir théorème 6). Comme dans notre cas $d_{max} = 3$, les seuls points anguleux de $\mathcal{P}$ sont ceux d'abscisses 1 et 2. On a donc $N_1 = 2$ et $p_1 = 0$. On en déduit, d'après le théorème 6, que :

$$E_p = \mathcal{C}^p(E_o)$$

On vérifie facilement que les arbres définis en 2.6.3 ont pour coût E(n). Ils sont donc optimaux. On a donc dans ce cas :

$$\mathcal{E} = \mathcal{E}_p = \mathcal{C}_p = E_o = C_o$$

et les points anguleux de E_p sont :

abscisses	3^p	2.3^p	3^{p+1}
ordonnées	$p.3^{p+1}$	$(6p+4).3^p$	$(p+1).3^p$

4.4.3. CAS $\varphi(d) = d+0.5$

Nous avons déjà étudié ce cas en 2.6.4 et en 3.3.
Rappelons que :

$$a = 3 \qquad d_{max} = 4 \qquad k = 3.5 / \text{Log } 3$$

Sur la figure 3.3, on peut constater que le dernier point anguleux dont la décomposition ne comporte pas le degré 3 a pour abscisse $N_1 = 16$. On a donc $p_1 = 2$.

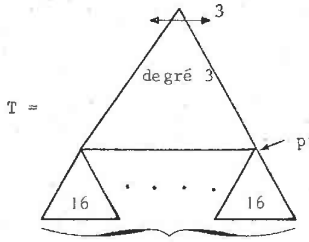
On en déduit que $\mathcal{E}_2$ est défini par les points anguleux :

$$(1,0) \quad \left(\frac{4}{3},\frac{4}{3}\right) \quad \left(\frac{16}{9},\frac{32}{9}\right) \quad (3,10.5)$$

et que les points anguleux de E_p ($p \geq 2$) ont pour abscisses et ordonnées :

3^p	4.3^{p-1}	16.3^{p-2}	3^{p+1}
$p\ 3^p \varphi(3)$	$4.3^{p-1}\left[(p-1)\varphi(3)+\varphi(4)\right]$	$16.3^{p-2}\left[(p-2)\varphi(3)+2\varphi(4)\right]$	$(p+1)3^{p+1}\varphi(3)$

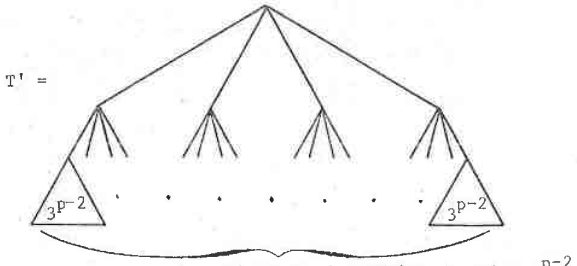
L'arbre régulier optimal de poids 16.3^{p-2} a la forme suivante :



3^{p-2} arbres 4-parfaits de poids 16

FIGURE 4.

Remarquons que cet arbre est équivalent à l'arbre T' suivant :

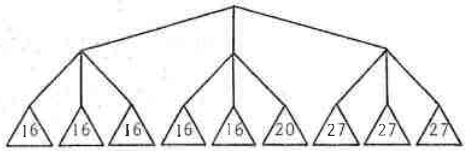


16 arbres 3-parfaits de poids 3^{p-2}

FIGURE 5.

Donc, même si n est grand, on pourra toujours trouver des arbres de degré au sommet 4, donc de degré différent du degré-limite 3.

On voit sur cet exemple que, par filtrages successifs, on peut amener le degré-limite a au sommet de l'arbre et amener les degrés différents de a vers les profondeurs de l'arbre. Nous montrerons au paragraphe suivant que cette technique s'applique aussi pour les arbres optimaux ne correspondant pas à des points anguleux. Sur notre exemple, pour n = 181, un arbre optimal possible est :



n = 16 arbre 4-parfait
n = 27 arbre 3-parfait
n = 20 l'arbre optimal es

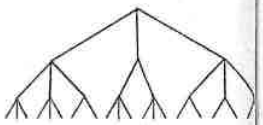


FIGURE 6.

4.5. FORMULE DE CALCUL EN TEMPS FINI DES ARBRES OPTIMAUX ET DE LEURS COUTS

Nous avons vu sur l'exemple précédent qu'il existe des arbres optimaux pour lesquels les noeuds ont pour degré le degré-limite aux petites profondeurs. Nous allons préciser cette notion et montrer qu'à partir d'un certain rang la forme des arbres est suffisamment régulière pour être calculable en temps constant.

4.5.1. PREMIERES INDICATIONS

Nous allons d'abord montrer qu'il existe des arbres optimaux pour lesquels la différence des poids des sous-arbres principaux n'est pas arbitrairement grande. Pour cela, il nous faut définir la notion de "voisinage".

Définition 6 : Nous appelons i-ième voisinage de la poche p l'intervalle réel suivant :

$$V_{p,i} = \left[\ell(V_{p,i}), \ell'(V_{p,i})\right]$$

$\ell(V_{p,i})$ et $\ell'(V_{p,i})$ sont les abscisses des points d'intersection de $\left[D_{p,i}(x)+B\right]$ avec E(x). S'il n'y a qu'un seul point d'intersection, $\ell'(V_{p,i})$ est son abscisse et $\ell(V_{p,i}) = 1$.
La constante B est la borne du théorème 7.

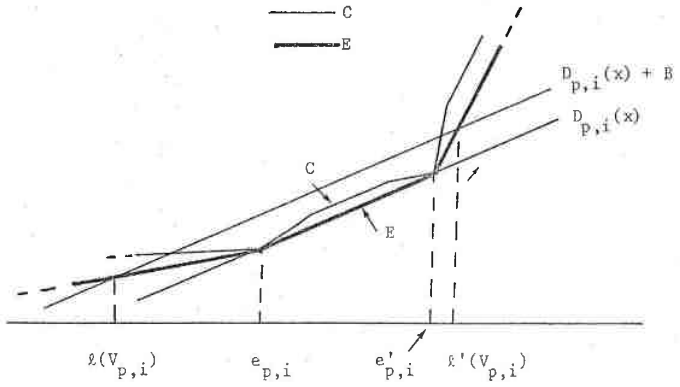


FIGURE 7

Comme à la définition 2, à un voisinage $V_{p,i}$ de la poche p, nous ferons correspondre les voisinages $V_{p',i}$ dans les poches p' (si p et p' $\geq p_1$).
D'autre part, on peut remarquer que l'intervalle $V_{p,i}$ contient strictement $I_{p,i} = \left[e_{p,i}, e'_{p,i}\right]$ et que $\ell(V_{p,i})$ et $\ell'(V_{p,i})$ sont croissants avec p et i.

Dans la suite, quand ce ce ne sera pas nécessaire, nous omettrons l'indice i indiquant le numéro du segment dans la poche.

Lemme 4 : Si n vérifie $C(n) \leq D_{p,i}(n) + B$ alors $n \in V_{p,i}$

Preuve : Comme $C(n) \geqslant E(n)$, on a :

$$E(n) \leqslant D_{p,i}(n) + B$$

D'après la définition précédente, cela n'est possible que si $n \in V_{p,i}$. ▨

Définition 7 : Nous appelons p_3 le plus petit entier tel que :

(34) $$\ell(V_{p_3+1,1}) \geqslant N_2$$ (l'entier N_2 est celui de la définitio

Il nous faut montrer que p_3 existe ! Si p est plus grand que p_2, appelons le premier segment de E_p et S'_{p-1} le dernier segment de E_{p-1}. Considérons les droites correspondantes D_p et D'_{p-1}. Ces droites se coupent au point $\left(a^p, G(a^p)\right)$. Leurs équations s'écrivent donc :

$$D_p(x) = G\left(a^p\right) + \alpha_p\left(x-a^p\right)$$

$$D'_{p-1}(x) = G\left(a^p\right) + \alpha'_{p-1}\left(x-a^p\right)$$

où α_p et α'_{p-1} désignent leurs pentes respectives. Par définition, $\ell(V_p)$ vérifie :

$$E\left(\ell(V_p)\right) = D_p\left(\ell(V_p) + B\right)$$

Comme $D'_{p-1} \leqslant E$, on en déduit :

$$D'_{p-1}\left(\ell(V_p)\right) \leqslant D_p\left(\ell(V_p)\right) + B$$

ce qui s'écrit :

$$\ell(V_p) \geqslant a^p - \frac{B}{\alpha_p - \alpha'_{p-1}}$$

Un calcul simple montre que la différence des pentes $(\alpha_p - \alpha'_{p-1})$ est consta quand $p > p_1$. On peut donc choisir p suffisamment grand pour que $\ell(V_p) \geqslant N_2$.

Comme V_p désigne en fait $V_{p,1}$, premier voisinage de la poche p, on trouvera bien un entier p_3 vérifiant (34).

Remarquons que $p_3 \geqslant p_2$ puisque p_2 désigne la poche de N_2.

Lemme 5 : Si $n \in V_{p,i}$, avec $p > p_3$, alors le degré-limite a est possible pour n.

Preuve : Comme $\ell(V_{p,i})$ est croissant avec p et i, on a :

$$\ell(V_{p,i}) \geqslant \ell(V_{p_3+1,i})$$

Comme par définition $n \geqslant \ell(V_{p,i})$, on en déduit, d'après (34) :

$$n \geqslant N_2$$

D'après le théorème 8, le degré-limite a est donc possible pour n. ▨

Nous sommes maintenant en mesure de donner une première indication sur la forme des arbres optimaux.

Proposition 5 : Si $n \geqslant a^{p_3+1}$ et si $I_{p,i}$ désigne l'intervalle de n, alors il existe un arbre optimal ayant la forme suivante :

1) Tous les noeuds de profondeur plus petite ou égale à $p-p_3-1$ ont pour degré a.
2) Les sous-arbres optimaux dont le sommet est à la profondeur $p-p_3$ ont leur poids dans $V_{p_3,i}$.

Preuve : Dans cette preuve, nous omettrons l'indice i qui désigne le numéro du segment. Nous écrirons donc I_p pour $I_{p,i}$, D_p pour $D_{p,i}$, V_p pour $V_{p,i}$ etc... Nous allons construire un arbre optimal ayant cette forme. Comme $p > p_3$, il existe, d'après le lemme précédent, un arbre optimal T de degré au sommet a.

Comme D_p désigne la droite prolongeant le segment S_p de n, on a :

$$D_p \leqslant E$$

et, d'après le théorème 7 :

$$C(n) \leqslant D_p(n) + B$$

D'après le lemme 2, les poids n_1 des noeuds situés à la profondeur 1 vérifient donc :

$$C(n_1) \leqslant D_{p-1}(n_1) + B$$

En effet, les droites D_p et D_{p-1} correspondent pour d = a, aux droites D et D' du lemme 2. D'après le lemme 4, les poids n_1 sont donc dans V_{p-1}. Si $p-1 = p_3$, alors l'arbre T répond aux conditions de la proposition 5 et la démonstration est terminée.

Si $p-1 > p_3$, alors le degré a est possible pour les sous-arbres principaux puisque leurs poids sont dans V_{p-1} (d'après le lemme 5). On peut donc remplacer, si nécessaire, ces sous-arbres par des sous-arbres optimaux de degré au sommet a. On obtient ainsi un nouvel arbre optimal dont les noeuds situés aux profondeurs 0 et 1 ont pour degré a. D'autre part, comme $D_{p-1} \leqslant E$ et $C(n_1) \leqslant D_{p-1}(n_1)+B$ (n_1 est le poids d'un noeud quelconque de profondeur 1), on peut réappliquer les lemmes 2 et 4. On montre ainsi que les poids des noeuds situés à la profondeur 2 dans le nouvel arbre optimal sont dans V_{p-2}.

En continuant ce processus, on obtient un arbre satisfaisant les conditions de la proposition 5. On est obligé de s'arrêter quand on a montré que les poids des noeuds de profondeur $p-p_3$ sont dans V_{p_3}. En effet, rien n'assure que le degré a est possible pour $n \in V_{p_3}$. ▨

Cette proposition nous permet de restreindre la classe des arbres à explorer pour trouver l'arbre optimal. La recherche du coût optimal s'en trouve également simplifiée, comme le montre la proposition suivante.

Proposition 6 : Si $n \geqslant a^{p_3+1}$ et si $I_{p,i}$ désigne l'intervalle de n, le coût optimal est donné par la formule :

$$(35) \qquad \begin{cases} C(n) = (p-p_3).\varphi(a).n + \text{Min} \left[\sum_{1 \leqslant j \leqslant a^{p-p_3}} C(n_j) \right] \\ \sum_{1 \leqslant j \leqslant a^{p-p_3}} n_j = n \qquad \text{avec } n_j \in V_{p_3,i} \end{cases}$$

Preuve : Il suffit de calculer le coût d'un arbre quelconque ayant la forme décrite dans la proposition 5. Comme d'après cette proposition il existe un arbre optimal de cette forme, il suffit de minimiser le coût pour obtenir la formule (35). ▨

4.5.2. FORMULE DE CALCUL EN TEMPS FINI

A ce point de l'exposé, nous avons montré qu'il existe des arbres optimaux ayant la forme suivante, si $n \geqslant a^{p_3+1}$

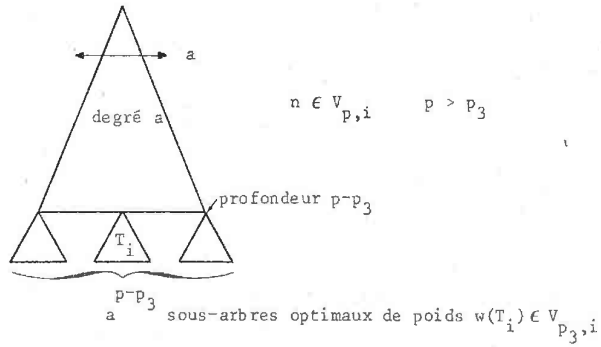


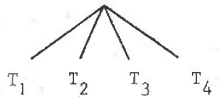
FIGURE 8.

Nous allons maintenant nous attacher à trouver une formule donnant les poids des sous-arbres T_i.

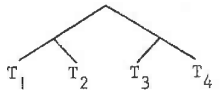
Nous supposons d'abord que le degré-limite a est au moins égal à 3. Cela nous permettra de simplifier les démonstrations. Si le degré-limite est 2, il suffit de remplacer la fonction φ par la fonction φ_1 suivante :

$$\begin{cases} \varphi_1(d) = \varphi(d) & \text{pour } d \neq 4 \\ \varphi_1(4) = 2\varphi(2) \end{cases}$$

On montre facilement que les arbres optimaux pour φ se déduisent des arbres optimaux pour φ_1 en remplaçant les noeuds de degré 4



par le sous-arbre :



Pour la fonction de coût φ_1, les degrés 2 et 4 sont des degré-limites puisque :

$$\frac{\varphi_1(2)}{\text{Log } 2} = \frac{\varphi_1(4)}{\text{Log } 4} \leq \frac{\varphi(n)}{\text{Log } n} \qquad \text{pour tout } n.$$

Bien que 2 soit le plus petit degré-limite, nous poserons a = 4. Tous les résultats montrés jusqu'ici restent valables. Les résultats suivants s'appliquent donc pour φ_1. On en déduit les arbres optimaux pour φ par la transformation ci-dessus .

Supposons donc $a \geq 3$. Nous allons mettre en évidence une certaine régularité de C(n), grâce au lemme suivant dont la démonstration figure en annexe.

Lemme 6 : Soient des entiers positifs a, L, ℓ, ℓ' ($a \geq 3$) et la suite de fonctions g_q suivantes (q entier ≥ 0)

1) g_o est une fonction bornée quelconque de W_o dans les réels positifs ou nuls :
W_o = ensemble des entiers relatifs de $[-\ell, L+\ell']$
g_o vérifie :

(36)
$$\begin{cases} g_o(0) = g_o(L) = 0 \\ \text{pour } m < 0 \text{ ou } m > L,\ g_o(m) > 0 \end{cases}$$

2) pour $q \geq 1$, g_q est une fonction de W_q dans les réels positifs ou nuls, définie par :
W_q = ensemble des entiers de $[0, L\, a^q]$

(37)
$$\begin{cases} g_q(m) = \text{Min} \sum_{1 \leq i \leq a^q} g_o(m_i) \\ \sum_{1 \leq i \leq a^q} m_i = m \qquad m_i \in W_o \end{cases}$$

Alors les fonctions g_q possèdent une certaine régularité à partir d'un certain rang. Plus précisément :
$\exists Q \geq 1$

(38) $$(\forall q)(\forall q') \qquad q > q' \geq Q$$

(39) $$g_q(m) = g_{q'}(m') \qquad (m \in W_q\ ,\ m' \in W_{q'})$$

et m' se déduit de m par :

(40)
$$m \in \begin{cases} [0, \gamma] \\ ]\gamma\ ,\ a^q L - \gamma[\\ [a^q L - \gamma\ ,\ a^q L] \end{cases} \qquad m' = \begin{cases} m \\ j\gamma + \text{reste}(m, \gamma) \\ a^{q'} L + n - a^q L \end{cases}$$

γ et j sont des entiers vérifiant :
$\gamma = a^{q'-1} L$ et $1 \leq j \leq a-2$

L'entier Q est le plus petit entier vérifiant :

$$a^{Q-1} > \left\lceil s_1 + \frac{\mu_1}{\mu_2}\left(1 + \frac{\mathrm{Max}(\ell,\ell')}{L}\right)\right\rceil$$

avec :

$s_1 = \sum_{m \in Z} \left[p(m)-1\right]$

Z est l'ensemble des zéros de g_o différents de 0 et L ; $p(m) = \frac{\mathrm{P.P.C.M.}(m,L)}{m}$

μ_1 = maximum de g_o

μ_2 = minimum non nul de g_o

Preuve : La démonstration de ce lemme, qui est assez longue et technique, est donnée en annexe.

Nous en déduisons une certaine régularité de C :

Lemme 7 : A chaque segment i de E_{p_1} correspond un entier P_i tel que si $p > \pi \geq P_i$ et $n \in I_{p,i}$ alors :

(41) $C(n) = E(n) + C(n') - E(n')$ avec $n' \in I_{\pi,i}$ et n' se déduit de n par les formules suivantes :

(42) $$\pi \in \begin{cases} \left[e_{p,i}, e_{p,i}+\gamma\right] \\ \left]e_{p,i}+\gamma, e'_{p,i}-\gamma\right[\\ \left[e'_{p,i}-\gamma, e'_{p,i}\right] \end{cases} \qquad n' = \begin{cases} e_{\pi,i} + n - e_{p,i} \\ e_{\pi,i} + j\gamma + \mathrm{reste} \\ \quad + \mathrm{reste}(n-e_{p,i},\gamma) \\ e'_{\pi,i} + n - e'_{p,i} \end{cases}$$

j et γ sont des entiers vérifiant $1 \leq j \leq a-2$ et $\gamma = |I_{\pi,i}|/a$.

Preuve : Pour alléger la preuve, nous supprimons l'indice i indiquant le numéro du segment : nous écrirons I_p pour $I_{p,i}$, V_p pour $V_{p,i}$, V_{p_3} pour $V_{p_3,i}$ etc...

Supposons tout d'abord $p > p_3$. D'après la proposition 6, le coût optimal C(n) s'écrit :

(43) $$\begin{cases} C(n) = (p-p_3).\varphi(a).n + \mathrm{Min} \sum_{1\leq i\leq a} {}^{p-p_3} C(n_i) \\ \sum_{1\leq i\leq a} {}^{p-p_3} n_i = n \qquad \text{avec } n_i \in V_{p_3} \end{cases}$$ (44)

D'après le corollaire 2 et la formule (44), $E_p(n)$ est égal à :

(45) $$E_p(n) = (p-p_3)\,\varphi(a)\,n + \sum_{1\leq i\leq a} {}^{p-p_3} D_{p_3}(n_i)$$

Posons :

(46) $$A_p(n) = C_p(n) - D_p(n)$$

D'après (43), (44) et (45), $A_p(n)$ vérifie :

(47) $$\begin{cases} A_p(n) = \mathrm{Min} \sum_{1\leq i\leq a} {}^{p-p_3} A_{p_3}(n_i) \\ \sum_{1\leq i\leq a} {}^{p-p_3} n_i = n \qquad \text{avec } n_i \in V_{p_3} \end{cases}$$ (48)

Définition de g_o : Nous allons définir une fonction g_o satisfaisant les conditions du lemme 6. Posons :

(49) $$\begin{cases} L = e'_{p_3} - e_{p_3} = |I_{p_3}| \\ \ell = \left\lfloor e_{p_3} - \ell\left(V_{p_3}\right)\right\rfloor \\ \ell' = \left\lfloor \ell'\left(V_{p_3}\right) - e'_{p_3}\right\rfloor \\ g_o(m) = A_{p_3}\left(e_{p_3}+m\right) \end{cases}$$

La définition de g_o est cohérente puisque :

$$m \in W_o \Longrightarrow \left(e_{p_3} + m\right) \in V_{p_3} \tag{50}$$

$$m \in V_{p_3} \Longrightarrow \left(n - e_{p_3}\right) \in W_o \tag{51}$$

On vérifie facilement que :

$$A_{p_3}(n) \geqslant 0 \qquad \text{pour } e_{p_3} < n < e'_{p_3}$$

$$A_{p_3}\left(e_{p_3}\right) = A_{p_3}\left(e'_{p_3}\right) = 0$$

$$A_{p_3}(n) > 0 \qquad \text{pour } n < e_{p_3} \text{ ou } n > e'_{p_3}$$

La fonction g_o satisfait bien les conditions du lemme 6.

Correspondance avec $A_p(n)$

Si dans les formules (47) et (48), on pose $n_i = e_{p_3} + m_i$ et $n = e_p + m$, elles deviennent, d'après (49), (50) et (51) :

$$\begin{cases} A_p(e_p+m) = \operatorname{Min}\left[\sum_{1 \leqslant i \leqslant a} {}_{p-p_3}\, g_o(m_i)\right] \\ \sum_{1 \leqslant i \leqslant a} {}_{p-p_3}\, m_i = m \qquad \text{avec } m_i \in W_o \end{cases}$$

Mais ces formules correspondent exactement à la définition de g_q dans le lemme 6. On a donc :

$$\text{pour } n \in I_p \qquad A_p(n) = g_{p-p_3}(n-e_p) \tag{52}$$

et inversement :

$$\text{pour } m \in W_q \qquad g_q(m) = A_{p_3+q}\left(e_{p_3+q}+m\right) \tag{53}$$

Application du lemme 6

Soit Q l'entier du lemme 6 correspondant à la fonction g_o définie en (49)

Posons :

$$P_i = p_3 + Q$$

Considérons deux entiers p et π vérifiant $p > \pi \geqslant P_i$. Appliquons le lemme 6 avec $q = p-p_3$ et $q' = \pi-p_3$. On a donc :

$$g_{p-p_3}(m) = g_{\pi-p_3}(m') \qquad \left(m \in W_{p-p_3},\ m' \in W_{\pi-p_3} \right)$$

m' se déduit de m par :

$$m \in \begin{cases} [0,\gamma] \\ ]\gamma,\ a^{p-p_3}L - \gamma[\\ [a^{p-p_3}L - \gamma,\ a^{p-p_3}L] \end{cases} \qquad m' = \begin{cases} m \\ j\gamma + \text{reste } (m,\gamma) \\ a^{\pi-p_3}L + n - a^{p-p_3}L \end{cases}$$

j et γ sont des entiers vérifiant $\gamma = a^{\pi-p_3-1}L$ et $1 \leqslant j \leqslant a-2$

Si nous posons maintenant $m = n-e_p$ et $m' = n'-e_\pi$, nous obtenons les formules suivantes, en utilisant la formule de conversion (53) :

$$A_p(n) = A_\pi(n) \qquad (n \in I_p,\quad n' \in I_\pi)$$

n' se déduit de n par :

$$n \in \begin{cases} [e_p,\ e_p + \gamma] \\ ]e_p + \gamma,\ e_p + a^{p-p_3}L - \gamma[\\ [e_p + a^{p-p_3}L - \gamma,\ e_p + a^{p-p_3}L] \end{cases} \qquad n' = \begin{cases} e_\pi + m \\ e_\pi + j\gamma + \text{reste } (n-e_p,\gamma) \\ e_\pi + a^{\pi-p_3}L + n - a^{p-p_3}L \end{cases}$$

D'après le corollaire 2 et la définition de L, on a :

$$\begin{cases} e_p + a^{p-p_3}L = e'_p \\ e_\pi + a^{\pi-p_3}L = e'_\pi \\ \gamma = a^{\pi-p_3-1}L = |I_\pi|/a \end{cases}$$

En substituant ces formules dans les précédentes et en se souvenant que $A(n) = C(n) - E(n)$, on obtient bien les formules (41) et (42) à démontrer. Vérifions que γ est bien entier : en effet, on sait que $\pi \geqslant P_i > p_3 \geqslant p_2 \geqslant p_1$; d'où $\pi > p_1$; d'après le corollaire 2, on a donc

$$|I_\pi|/a = |I_{p_1}| a^{\pi - p_1 - 1}$$

Comme $|I_{p_1}|$ est entier et $\pi - p_1 - 1 \geqslant 0$, γ est bien entier. ▨

Nous en déduisons que le coût des arbres optimaux est calculable par une formule simple à partir d'un certain rang.

Proposition 7 : Il existe un entier p_4, $(p_4 > p_3)$, tel que, si $n \geqslant a^{p_4+1}$ et $n \in I_{p,i}$ alors :

(54) $C(n) = E(n) + C(n') - E(n')$ avec $n' \in I_{p_4,i}$

et n' se déduit de n par les formules suivantes :

(55)
$$n \in \begin{cases} \left[e_{p,i},\ e_{p,i} + \gamma\right] \\ \left]e_{p,i}+\gamma, e'_{p,i} - \gamma\right[\\ \left[e'_{p,i} - \gamma,\ e'_{p,i}\right] \end{cases} \qquad n' = \begin{cases} e_{p_4,i} + n - e_{p,i} \\ e_{p_4,i} + j\gamma + \text{reste } (n - e_{p,i}, \gamma) \\ e'_{p_4,i} + n - e'_{p,i} \end{cases}$$

(56) j et γ sont entiers ; $\gamma = |I_{p_4,i}|/a$; $1 \leqslant j \leqslant a-2$

Preuve : A chaque segment i de E_{p_1} correspond l'entier P_i du lemme précédent. Soit p_4 le plus grand des P_i. Sur chaque segment i de E_{p_1}, on peut appliquer le lemme précédent avec $\pi = p_4$. Comme $p > p_4 \geqslant P_i$ on déduit les formules (54) et (55) des formules (41) et (42). ▨

Remarque : D'après la preuve, les formules (54), (55) et (56) restent valables si on remplace p_4 par un entier plus grand.

La formule (54) nous donne un procédé simple de calcul du coût. Nous allons maintenant construire des arbres dont le coût vérifie (54) ; ces arbres seront donc optimaux. Leur forme est donnée par le théorème suivant :

Théorème 9 : Si $n \geqslant a^{p_4+1}$ (p_4 est défini à la proposition 7), et si $I_{p,i}$ est l'intervalle de n, il existe un arbre optimal T de poids n ayant la forme suivante :

1) Jusqu'à la profondeur $p - p_4 + 1$, tous les noeuds ont pour degré le degré-limite a

2) A la profondeur $p - p_4$, il y a a^{p-p_4} noeuds qui sont sommets de sous-arbres optimaux : il y a $(a^{p-p_4} - 2)$ noeuds de poids $e_{p_4,i}$ ou $e'_{p_4,i}$, dont M noeuds de poids $e_{p_4,i}$. Les poids des deux noeuds restants, n_1 et n_2, sont dans $I_{p_4,i}$ et on a $C(n_2) = E(n_2)$.

(57) Le coût optimal $C(n)$ est égal à :
$C(n) = E(n) + C(n_1) - E(n_1)$

Cet arbre est entièrement décrit par la donnée de M, n_1 et n_2. Ces entiers sont donnés par les formules suivantes :

Posons :

$\gamma = |I_{p_4,i}|/a$ (γ est entier car $p_4 > p_1$)

$q = \text{quotient } (n - e_{p,i}, \gamma)$

$r = \text{reste } (n - e_{p,i}, \gamma)$

Nous distinguons 3 cas suivant les valeurs de q :

A) $q = 0$ $\quad \left(n \in \left[e_{p,i},\ e_{p,i} + \gamma \right[\right)$

(58)
$$\begin{cases} M = a^{p-p_4} - 2 \\ n_1 = e_{p_4,i} + n - e_{p,i} \\ n_2 = e_{p_4,i} \end{cases}$$

B) $q \leqslant a^{p-p_4+1} - a$ $\quad \left(n \in \left[e_{p,i}+\gamma,\ e'_{p,i} - (a-1)\gamma \right[\right)$

Posons q' = quotient $(q-1, a)$
r' = reste $(q-1, a)$

(59)
$$\begin{cases} M = a^{p-p_4} - q' - 2 \\ n_1 = e_{p_4,i} + \gamma + r \\ n_2 = e_{p_4,i} + r'\gamma \end{cases}$$

C) $q > a^{p-p_4+1} - a$ $\quad \left(n \in \left[e'_{p,i} - (a-1)\gamma\ ,\ e'_{p,i} \right[\right)$

(60)
$$\begin{cases} M = 0 \\ n_1 = e'_{p_4,i} + n - e'_{p,i} \\ n_2 = e'_{p_4,i} \end{cases}$$

La forme de l'arbre est indiquée sur le schéma suivant :

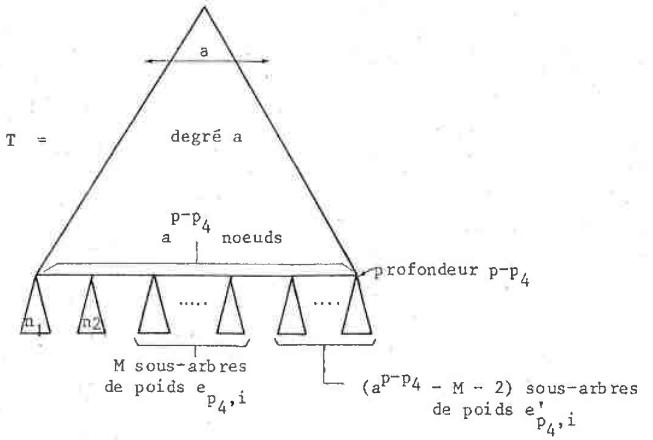


FIGURE 9.

Preuve : On vérifie facilement dans chaque cas que M, n_1 et n_2 satisfont aux conditions du théorème, c'est-à-dire :

(61)
$$\begin{cases} n = M.e_{p_4,i} + (a^{p-p_4} - M - 2).e'_{p_4,i} + n_1 + n_2 \\ 0 \leqslant M \leqslant a^{p-p_4} - 2 \\ n_1 \text{ et } n_2 \in \left[e_{p_4,i},\ e'_{p_4,i} \right] \\ C(n_2) = E(n_2) \end{cases}$$

L'égalité $C(n_2) = E(n_2)$ résulte du corollaire 4, puisque $\gamma = |I_{p_4,i}|/a$.

Il reste à montrer que l'arbre T est optimal. Calculons son coût :

(62) $$\mathcal{C}(T) = (p-p_4).\varphi(a).n + M.C(e_{p_4,i}) + (a^{p-p_4}-M-2).C(e'_{p_4,i}) + C(n_1) + C(n_2)$$

Calculons maintenant E(n). Comme tous les noeuds situés à la profonde
$(p-p_4)$ ont leurs poids n_j dans $I_{p_4,i}$, on a $D_{p_4,i}(n_j) = E(n_j)$.

D'après le corollaire 2, formule (11), E(n) s'écrit donc :

(63) $$E(n) = (p-p_4).\varphi(a).n + M.E(e_{p_4,i}) + (a^{p-p_4}-M-2).E(e'_{p_4,i}$$
$$+ E(n_1) + E(n_2)$$

Comme les fonctions C et E sont égales en $e_{p_4,i}$, $e'_{p_4,i}$ et n_2, on déduit de (62) et (63) que :

$$\mathcal{C}(T) - E(n) = C(n_1) - E(n_1) \qquad \text{avec } n_1 \in I_{p_4,i}$$

Cette formule est analogue à la formule (54) de la proposition 7. Si nous montrons que $n_1 = n'$, nous en déduirons que $\mathcal{C}(T) = C(n)$, et donc que l'arbre T du théorème est bien optimal.

Nous allons donc montrer cas par cas que $n_1 = n'$.

1) Cas A $\quad n \in \left[e_{p,i},\ e_{p,i} + \gamma\right[$

C'est évident dans ce cas car $n_1 = n'$.

2) Cas B $\quad n \in \left[e_{p,i} + \gamma,\ e'_{p,i} - (a-1).\gamma\right[$

L'entier n_1 est égal à l'entier n' correspondant à j = 1

3) Cas C $\quad n \in \left[e'_{p,i} - (a-1)\gamma,\ e'_{p,i}\right[$

Il faut distinguer deux sous-cas :

- Cas C.1 $\quad n \in \left[e'_{p,i} - (a-1)\gamma\ ,\ e'_{p,i} - \gamma\right[$

on vérifie que n_1 peut s'écrire :

$$n_1 = e_{p_4,i} + (q - a^{p-p_4+1} + a).\gamma + \text{reste } (n-e_{p,i},\gamma)$$
avec $1 \leqslant q - a^{p-p_4+1} + a \leqslant a-2$

L'entier n_1 est donc égal à l'entier n' pour $j = q - a^{p-p_4+1}\Big| + a$

- Cas C.2 $\quad n \in \left[e'_{p,i}-\gamma,\ e'_{p,i}\right[$

C'est évident dans ce cas car $n_1 = n'$. ▨

4.5.3. ALGORITHME DE CALCUL EN TEMPS FINI

La forme des arbres est donc "simple" à partir d'un certain rang. La forme et le coût des arbres se calcule immédiatement si l'on connaît C(n) et E(n) pour $n \leqslant a^{p_4+1}$. On en déduit :

Théorème du temps fini (10). La forme et le coût des arbres optimaux peuvent être calculés en temps fini par la méthode suivante :

1) Calculer E (proposition 1)
2) Déterminer N_2 (théorème 8)
3) Déterminer p_3 (définition 7)
4) Déterminer p_4 (lemme 6, proposition 7)
5) Calculer la forme et le coût C(n) des arbres optimaux pour $n \in \left[1,\ a^{p_4+1}\right]$ (Algorithme 2.1.4)
6) Si $n > a^{p_4+1}$, la forme et le coût C(n) des arbres optimaux sont donnés par les formules du théorème 9.

Preuve : Il est évident que toutes ces opérations prennent un temps fini, qui ne dépend que de la donnée de φ. ▨

Algorithme de calcul en temps fini

Si les cinq premières étapes ont été franchies, l'algorithme suivant calcule les arbres pour $n > a^{p_4+1}$. Plus précisément, il a pour résultats M, n_1, n_2, qui déterminent la forme de l'arbre optimal (voir théorème 9), ainsi que le coût optimal, noté C_n.

Nous supposons donc donnés :

- a et $\varphi(a)$: a est le degré-limite
- p_4 : l'entier déterminé à la proposition 7
- s : le nombre de segments de E_{p_1} (donc de E_{p_4})
- C(n) et la forme des arbres optimaux pour $1 \leqslant n \leqslant a^{p_4+1}$
- X(i) $(1 \leqslant i \leqslant s+1)$: abscisse du i-ième point anguleux de E_{p_4} $\left(X(1) = a^{p_4} \; ; \; X(s+1) = a^{p_4+1}\right)$

Nous indiquons les commentaires entre accolades.

```
tableau  C(a^p4) ;

tableau  X(s+1) ;

p: = ⌊Log n / Log a⌋ ;  {détermination de la poche de n}

                        {calcul de e_p4,i  et  e'_p4,i}
A: = a^(p-p4) ;

m: = n/A ;

i: = 1 ;

tant que m ⩾ X(i+1) faire i: = i+1 ;

e_p4 : = X(i) ; e'_p4 : = X(i+1) ;

e_p: = A.e_p4 ; e'_p = A.e'_p4 ;

                        {calcul de M, n_1 et n_2}

γ: = (e'_p4 - e_p4)/a ;

q: = quotient (n-e_p,γ) ;
```



```
si q = 0 alors
début
        M: = A-2 ;
        n_1: = e_p4 + n - e_p ;
        n_2: = e_p4 ;
        allera suite ;
fin ;

si q ⩽ a.A - a alors
début
        r: = reste (n-e_p,γ) ;
        q': = quotient (q-1,a) ; r': = reste (q-1,a) ;
        M: = A - q' - 2 ;
        n_1: = e_p4 + γ + r ;
        n_2: = e_p4 + r'.γ ;
fin
sinon
début
        M: = 0
        n_1: = e'_p4 + n - e'_p ;
        n_2:= e'_p4 ;
fin ;

suite :                 {calcul du coût optimal C_n}
    pente: = (C(e'_p4)-C(e_p4))/(e'_p4 -e_p4) ;

    E_n1 : = C(e_p4) + pente.(n_1-e_p4) ;
    E_m: = C(e_p4) + pente.(m-e_p4) ;
    E_n: = (p-p_4).φ(a).n + A.E_m ;
    C_n: = E_n + C(n_1) - E_n1 ;
```

Notons que cet algorithme ne comporte pas de tableau pour la fonction E(n). La donnée des points anguleux de E_{p_4} suffit. C'est pourquoi nous écrivons E_{n_1}, E_m, E_n pour $E(n_1)$, $E(m)$, $E(n)$. Pour le calcul de E_n, nous avons utilisé le corollaire 2, formule (10). D'autre part, pour ne pas introduire de complications inutiles, nous avons supprimé l'indice indiquant le numéro de segment : nous écrivons e_p pour $e_{p,i}$, e_{p_4} pour $e_{p_4,i}$, etc...

Cet algorithme a pour résultats M, n_1, n_2 et le coût optimal C_n.
D'après le théorème 9, on en déduit la forme d'un arbre optimal possible.

L'efficacité de cet algorithme dépend de l'ordre de grandeur de p_4.
En effet, il faut tout de même calculer C(n) par l'algorithme classique pour $1 \leqslant n \leqslant a^{p_4+1}$. Nous avons vu que p_4 vérifie :

$$p_4 > p_3 \geqslant p_2 \geqslant p_1$$

La poche p_1 est celle à partir de laquelle l'enveloppe convexe E est régulière. La poche p_2 est celle qui contient N_2, rang à partir duquel le degré-limite a est possible pour tout n.
L'entier p_3 est tel que le degré-limite a soit possible dans tous les voisinages des segments de E_{p_3+1}. Notons que, généralement, $p_3=p_2$.
La poche p_4 est celle à partir de laquelle la fonction C elle-même devient régulière. En ce qui concerne les fonctions $\varphi(d) = d+\lambda$, on a généralement $p_4 \leqslant p_2 + 3$.

L'efficacité de l'algorithme est donc conditionnée par l'ordre de grandeur p_2, c'est-à-dire de N_2.

Cas des fonctions de coût $\varphi(d) = d+\lambda$

Reportons-nous à la figure 2. Si λ est plus petit que $\psi(3)$ et pas trop proche de $\psi(3)$, N_2 n'est pas très grand :

$$\text{pour } \lambda < 0.796 \quad N_2 \leqslant 215 \quad p_4 \leqslant 8$$

Mais si λ est proche de $\psi(3)$, alors N_2 tend vers l'infini :

$$\text{pour } \lambda > 0.797 \quad N_2 \geqslant 5000 \quad p_4 \geqslant 10$$

Il faut alors calculer C(n) jusqu'à : $3^{11} = 177\,147$!

Cas $\varphi(d) = d+0.5$

Nous avons déjà vu que dans ce cas $p_2 = 3$ et $N_2 = 52$.
On montre que $p_3 = 3$ et $p_4 = 4$. L'algorithme précédent fonctionne donc avec les données suivantes :

```
a: = 3 ; φ(a): = 3.5 ;
p4: = 4 ;
s: = 3 ;
tableau C(a^(p4+1)) ;
X: = (81,108,144,243) ;
```

Le tableau C, de dimension 243, est supposé calculé par l'algorithme classique. Remarquons que dans ce cas, l'algorithme en temps constant est parfaitement utilisable.

4.6. DIFFERENCES DE POIDS ET DE PROFONDEURS BORNEES

Nous allons voir que pour une fonction φ donnée ayant un seul degré-limite les différences de poids et de profondeurs des arbres optimaux sont bornées Avant d'aller plus avant, il nous faut bien préciser que cette propriété n'est vraie, pout tout n, que pour certains arbres optimaux. Il y a, en effet un grand nombre d'arbres optimaux pour n donné, qui ne vérifient pas tous ces propriétés.

4.6.1. DIFFERENCES DE POIDS BORNEES

Nous allons montrer qu'il existe pour tout n des arbres optimaux de poids n dont la différence des poids des sous-arbres principaux est bornée. Il nous faut d'abord donner la définition suivante :

Définition 7 : Pour φ et n donnés, nous appelons $\mathcal{E}_\varphi(n)$ l'ensemble des arbres à n feuilles optimaux pour la fonction de coût φ. Pour un arbre T, nous appelons Q(T) la différence entre le poids du sous-arbre principal de plus grand poids et le poids du sous-arbre principal de plus petit poids.

Nous définissons :

(64) $$Q_\varphi(n) = \min_{T \in \mathcal{C}_\varphi(n)} Q(T)$$

La quantité $Q\varphi(n)$ correspond donc à l'arbre optimal à n feuilles qui minimise la différence Q(T).

Lemme 8 : Si $n > a^{p_4+1}$ et si $I_{p,i}$ désigne l'intervalle de n, alors il existe un arbre optimal de poids n dont la différence des poids des sous-arbres principaux est bornée par $|I_{p_4,i}|$. L'entier p_4 est défini au théorème 9. Plus précisément :

$$n \in I_{p,i} \qquad p > p_4 \implies Q_\varphi(n) \leq |I_{p_4,i}|$$

Preuve : Reportons-nous au théorème 9 et à la figure 9 qui décrit la forme d'un arbre optimal T possible pour $n > a^{p_4+1}$.
Considérons l'arbre T' obtenu par contraction des noeuds situés à la profondeur $(p-p_4)$. D'après la figure 9, l'arbre T' est a-parfait et les poids de ses A feuilles sont dans l'intervalle $I_{p_4,i}$ $(A = a^{p-p_4})$. Appelons n_q leurs poids classés par ordre croissant :

$$e_{p_4,i} \leq n_1 \leq \ldots \leq n_j \leq n_{j+1} \leq \ldots \leq n_A \leq e'_{p_4,i}$$

On peut partitionner les poids n_q en a classes (de numéro 1,2,...,a) telles que la différence entre la somme de poids de deux classes quelconques n'excède pas $|I_{p_4,i}| = e'_{p_4,i} - e_{p_4,i}$. Pour cela, il suffit d'attribuer le poids n_q à la classe de numéro (q mod a).

Appelons $T'_1,\ldots,T'_d$ les sous-arbres principaux de T' et attribuons aux feuilles de T_j les A poids de la classe j. Dans l'arbre T" ainsi obtenu, faisons l'expansion de chaque feuille de poids n_j par l'arbre optimal à n_j feuilles. Nous obtenons ainsi un arbre T"'qui satisfait les conditions du lemme.

En effet, les poids de chacun de ses sous-arbres principaux sont égaux à la somme des poids de chacune des a classes définies précédemment. D'autre part, l'arbre T"' est optimal puisque nous avons seulement changé l'ordre des sous-arbres dont les sommets sont à la profondeur $p-p_4$.

▨

Proposition 8 : Pour une fonction de coût φ donnée, pour tout n, il existe un arbre optimal de poids n dont la différence de poids des sous-arbres principaux est bornée par une constante Q_φ. Cette propriété peut se traduire par la formule suivante :

$$(\forall n)(\exists T)\ T \in \mathcal{C}_\varphi(n) \qquad Q(T) \leq Q_\varphi$$

ou bien par la formule suivante :

$$(\forall n) \quad Q_\varphi(n) \leq Q_\varphi$$

La constante Q_φ vérifie :

$$Q_1 = \max_{1 \leq n \leq a^{p_4+1}} Q_\varphi(n)$$

$$Q_2 = \max_i |I_{p_4,i}|$$

$$Q_\varphi \leq \max(Q_1, Q_2)$$

Preuve : Si $n \leq a^{p_4+1}$, la propriété résulte de la définition de Q_1.
Si $n > a^{p_4+1}$, la propriété résulte du lemme précédent, puisque Q_2 est le plus grand intervalle de E_{p_4}.

▨

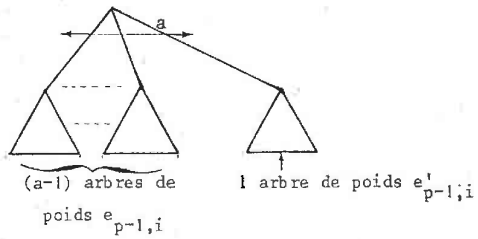
Il existe par contre des arbres optimaux dont la différence des poids des sous-arbres principaux est arbitrairement grande, et cela, pour toutes les fonctions φ possédant un seul degré-limite.

Considérons en effet un segment i de E_{p_1} et les valeurs de n suivantes, pour $p > p_1$:

$$n = e_{p,i} + |I_{p,i}|/a$$

D'après le corollaire 4, pour ces valeurs de n, on a C(n) = E(n).
On voit facilement qu'un arbre optimal possible est :

T =



La différence des poids entre les sous-arbres principaux est égale à :

$$Q(T) = e'_{p-1,i} - e_{p-1,i} = |I_{p_1,i}|.a^{p-p_1-1}$$

Comme p peut être arbitrairement grand, la différence de poids Q(T) peut être arbitrairement grande.
Notons cependant que la différence des poids ne peut être rendue arbitrairement grande pour toutes les valeurs de n : si n est l'abscisse d'un point anguleux, cette différence est nulle pour tous les arbres optimaux de poids n, d'après le théorème 4.

4.6.2. DIFFERENCES DE PROFONDEURS BORNEES

Nous avons besoin de la définition suivante :

Définition 8 : Pour un arbre donné T, nous appelons P(T) la profondeur de la feuille de profondeur minimum ; nous appelons G(T) la profondeur maximum et $\Delta(T) = G(T) - P(T)$.

Pour φ et n donnés, nous définissons :

(65) $$P_\varphi(n) = \min_{T \in \mathcal{C}_\varphi(n)} P(T)$$

(66) $$G_\varphi(n) = \max_{T \in \mathcal{C}_\varphi(n)} G(T)$$

(67) $$\Delta_\varphi(n) = \min_{T \in \mathcal{C}_\varphi(n)} \Delta(T)$$

Remarquons que $\Delta_\varphi(n) \leqslant G_\varphi(n) - P_\varphi(n)$. Mais il n'y a pas toujours égalité, les profondeurs maximum et minimum pouvant provenir d'arbres optimaux différents.

Proposition 9 : Pour une fonction de coût φ donnée, il existe pour tout n un arbre optimal T de poids n dont la différence maximum de profondeur des feuilles $\Delta(T)$ est bornée par une constante Δ_φ. Cette propriété peut se traduire par la formule suivante :

$$(\forall n)\ (\exists T)\ T \in \mathcal{C}_\varphi(n) \qquad \Delta(T) \leqslant \Delta_\varphi$$

ou bien par la formule suivante :

$$(\forall n)\ \Delta_\varphi(n) \leqslant \Delta_\varphi$$

La constante Δ_φ vérifie :

$$\Delta_1 = \max_{1 \leqslant n \leqslant a} p_{4+1}\ \Delta_\varphi(n)$$

$$\Delta_2 = \left[\max_{n \in P_{p_4}} G_\varphi(n)\right] - \left[\min_{n \in P_{p_4}} P_\varphi(n)\right]$$

$$\Delta_\varphi \leqslant \max(\Delta_1, \Delta_2)$$

(Rappelons que $P_{p_4} = \left[a^{p_4}, a^{p_4+1}\right]$).

Preuve : Si $n \leq a^{p_4+1}$, c'est évident d'après la définition de Δ_1. Supposons $n > a^{p_4+1}$. Considérons l'arbre T optimal du théorème 9 (voir figure 9). On sait que tous les sous-arbres optimaux dont le sommet est à la profondeur $p-p_4$ ont leurs poids dans $I_{p_4,i}$, donc dans P_{p_4}. Il en résulte que la différence maximum de profondeur des feuilles de T est inférieure ou égale à Δ_2, donc à Δ_φ. ▨

La constante Δ_φ est-elle toujours égale à un ? Si φ est de la forme $\varphi(d) = d + \lambda$, on peut vérifier que $\Delta_\varphi = 1$ pour $\lambda \leq 1/4$ (voir KNUTH [13], p. 377, Ex. 9). Pour ces valeurs de λ, il existe toujours pour tout n un arbre optimal dont la différence de profondeurs des feuilles n'excède pas 1. Dans le cas général, c'est faux. Par exemple, si φ est définie par :

$$\varphi(d) = d+0.81 \quad \text{alors} \quad \Delta_\varphi = 2$$

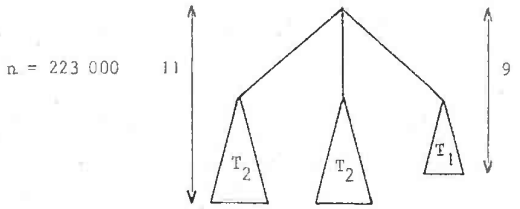
Par exemple, $\Delta_\varphi(223\ 000)=2$. Cela signifie qu'il n'existe pas d'arbre optimal à 223 000 feuilles dont la différence des profondeurs soit 0 ou 1. En effet, pour n = 223 000, l'arbre optimal est unique et à la forme suivante :

Pour cette fonction φ, le degré-limite unique est égal à 3 ; les valeurs n_1 et n_2 suivantes sont les abscisses de deux points anguleux consécutifs de la poche $P_{10} = [3^{10}, 3^{11}]$. Nous indiquons la décomposition des arbres réguliers T_1 et T_2 correspondants :

$$n_1 = 4^8 = 65\ 536 \quad \text{décomposition de } T_1 : (4^8)$$

$$n_2 = 4.3^9 = 78\ 732 \quad \text{décomposition de } T_2 : (3^9, 4^1).$$

On peut vérifier que pour $n = n_1 + 2\, n_2$, le seul arbre optimal T est :



Toutes les feuilles de T_1 étant à la profondeur 8 et celles de T_2 à la profondeur 10. On a bien $\Delta_\varphi(223\ 000) = 2$, puisque T est le seul arbre optimal pour n = 223 000.

Plus généralement, il existe des fonctions φ pour lesquelles Δ_φ est arbitrairement grand : considérons les fonctions φ du type $\varphi(d) = d+\lambda$, avec $\lambda < \psi(3) = 0.81884 \ldots$ (rappelons que si $\lambda = \psi(3)$, il y a deux degré -limites, 3 et 4). On peut montrer que si λ se rapproche de $\psi(3)$, alors Δ_φ augmente. Par exemple, si :

$$\varphi(d) = d + 0.\ 81863 \qquad \text{alors } \Delta_\varphi = 6.$$

Cela signifie que pour cette fonction φ, il existe des valeurs de n pour lesquelles tous les arbres optimaux T ont une différence de profondeur des feuilles $\Delta(T)$ égale à 6.

Nous verrons au chapitre suivant que si $\lambda = \psi(3)$, c'est-à-dire s'il y a plusieurs degré -limites, la différence de profondeur des feuilles n'est pas bornée.

CHAPITRE 5

CAS OU IL Y A PLUSIEURS DEGRE-LIMITES

5.1. PRESENTATION ET RAPPEL DES RESULTATS

Comme au chapitre précédent, nous étudions toujours les arbres optimaux à poids égaux quand la fonction de coût est à degré borné, c'est-à-dire quand il existe pour tout n un arbre optimal de poids n dont les degrés sont au plus égaux à une constante d_{max}. Le chapitre précédent était consacré au cas où la fonction de coût possède un seul degré-limite, ou plusieurs degré-limites qui sont puissances entière d'un même nombre entier. Dans ce chapitre, nous supposons donc qu'il existe plusieurs degré-limites dont deux au moins, a et b (a<b), ne sont pas puissances entière d'un même nombre entier. Cette condition peut s'écrire :

(1) $\quad \text{Log } b/\text{Log } a$ irrationnel

Rappelons d'abord les principales propriétés montrées aux chapitres 1, 2 et 3 qui restent valables dans ce cas :

1) L'ensemble $\mathcal{D}$ des degré -limites (a, b $\in \mathcal{D}$) est l'ensemble des entiers d vérifiant (définition 2.3) :

$$k = \frac{\varphi(d)}{\text{Log } d} \leqslant \frac{\varphi(n)}{\text{Log } n} \qquad \text{pour tout } n$$

Si l'on pose $k = \varphi(a)/\text{Log } a$ et $G(x) = k.x.\text{Log } x$, on a :

$$G(n) \leqslant E(n) \leqslant C(n) < G(n) + \varphi(\mathbb{a})\ n$$

où C désigne le coût optimal et E l'enveloppe convexe de C (propositions 2.4 et 3.1).

2) Les arbres réguliers dont les degrés sont les degré-limites sont optimaux et leur coût est égal à la borne inférieure G (proposition 2.5).

3) Les transformées $\mathcal{C}_p$ et $\mathcal{E}_p$ de C_p et E_p convergent uniformément vers la même fonction continue et convexe $\mathcal{E}$ (théorème 5).

Récapitulons maintenant les résultats obtenus au chapitre précédent, dans le cas où il y a un seul degré-limite.

1) L'enveloppe convexe E est régulière à partir d'un certain rang. Plus précisément, si $p \geqslant p_1$, les fonctions E_p se déduisent de E_{p_1} par la transformation :

$$E_p = \mathcal{C}^{p-p_1}(E_{p_1})$$

et la limite $\mathcal{E}$ des transformées $\mathcal{E}_p$ est égale à $\mathcal{E}_{p_1}$ (théorème 6).

2) Si $n \geqslant N_2$ (la constante N_2 ne dépend que de φ), alors il existe toujours un arbre optimal à n feuilles dont le degré au sommet est le degré-limite a (théorème 8).

3) Il existe une formule permettant de calculer la forme et le coût C(n) des arbres optimaux en temps fini (théorème 9).

4) Pour tout n, il existe un arbre optimal dont la différence de poids des sous-arbres principaux est bornée et dont la différence de profondeur des feuilles est bornée (propositions 4.8 et 4.9).

Que deviennent ces propriétés dans le cas qui nous intéresse maintenant ? Nous supposons l'existence de plusieurs degré-limites dont deux au moins, a et b, vérifient l'hypothèse (1).
La première propriété n'est plus vraie, les fonctions $\mathcal{C}_p$ et $\mathcal{E}_p$ évoluent constamment vers une limite que nous montrerons être égale à $G(x) = k \times \mathrm{Log}\, x$.

Pour ce qui est de la deuxième propriété, nous montrerons ici une propriété plus forte : à partir d'un certain rang N_2, les arbres de poids supérieurs à N_2 ont obligatoirement pour degré au sommet l'un des degré-limites. Alors que dans l'autre cas, il existait des arbres optimaux de degré différent du degré-limite a pour des valeurs de n arbitrairement grandes (voir 4.4.3).

En ce qui concerne la complexité du problème de la recherche des arbres optimaux et de leurs coûts, il semble très improbable qu'il existe un algorithme en temps fini dans le cas qui nous intéresse. Le meilleur algorithme connu reste l'algorithme en temps $O(n^2)$ et en espace mémoire $O(n)$ décrit en 2.1.4.

En ce qui concerne les différences de poids et de profondeurs, nous montrons qu'elles ne sont pas bornées. Plus précisément, pour tout entier N, on peut trouver des valeurs de n pour lesquelles tous les arbres optimaux de poids n ont une différence de profondeur des feuilles supérieure à N. Il en est de même pour la différence des poids des sous-arbres principaux.

5.2. FORME ASYMPTOTIQUE DU COUT OPTIMAL ET DE L'ENVELOPPE CONVEXE

Nous voulons montrer que E et C tendent vers la borne inférieure G au sens suivant : leurs transformées $\mathcal{E}_p$ et $\mathcal{C}_p$ définies sur $[1,a]$ tendent vers G. Il nous faut d'abord établir le lemme suivant :

Lemme 1 : Si a et b sont des entiers ($a < b$) vérifiant la condition alors :

$$(2)\qquad (\forall \eta > 0)(\forall x \in [1,a])\ (\exists q \in \mathbb{N}^+)\ \left| x - \frac{b^q}{a^{p(q)}} \right| < \eta$$

avec $p(q) = \lfloor q \operatorname{Log}_a b \rfloor$

(x et η sont des réels)

Preuve : Si l'on passe aux logarithmes, on vérifie facilement que la condition suivante est une condition suffisante de (2)

$$(3)\qquad (\forall \eta > 0)(\forall x \in [1,a])\ (\exists q \in \mathbb{N}^+)$$
$$\left| \operatorname{Log}_a x - \left(q \operatorname{Log}_a b - \lfloor q \operatorname{Log}_a b \rfloor \right) \right| < \frac{\eta}{a \operatorname{Log} a}$$

Posons $z = \operatorname{Log}_a b$; z est irrationnel d'après la condition (1). Posons $y = \operatorname{Log}_a x$; y est une réel de $[0,1]$ puisque $x \in [1,a.]$ Une condition suffisante de (3) est donc que :

$$(4)\qquad \begin{cases} (\forall z > 0,\ z \text{ irrationnel})(\forall \varepsilon > 0)(\forall y \in [0,1])\ (\exists q \in \mathbb{N}^+) \\ \qquad \left| y - (qz - \lfloor qz \rfloor) \right| < \varepsilon \end{cases}$$

Cela revient à montrer que $(qz - \lfloor qz \rfloor)$ est dense dans $[0,1]$ quand z est un irrationnel et q varie sur les entiers. C'est un exercice de taupe bien connu ! ▨

Proposition 1 : Les transformées $\mathcal{C}_p$ et $\mathcal{E}_p$ de C_p et E_p tendent vers $G(x) = k.x.\operatorname{Log} x$ quand p tend vers l'infini.

Preuve : D'après le théorème 5, nous savons que $\mathcal{C}_p$ et $\mathcal{E}_p$ tendent vers la même limite $\mathcal{E}$. Pour montrer que $\mathcal{E} = G$, il suffit de montrer que $|\mathcal{E}_p(x) - G(x)|$ peut être rendu plus petit que ε, le terme positif ε pouvant être aussi petit que possible. On sait que :

$$(5)\qquad |\mathcal{E}_p(x)-G(x)| \leq |\mathcal{E}_p(x)-\mathcal{E}(x)| + |\mathcal{E}(x)-\mathcal{E}(x')| + |\mathcal{E}(x')-G(x')| + |G(x')-G(x)|$$

Les fonctions $\mathcal{E}$ et G étant continues sur l'intervalle fermé borné $[1,a]$, elles sont uniformément continues. Il est donc possible de choisir η tel que :

$$(6)\qquad (\forall x')\ |x-x'| < \eta \quad |\mathcal{E}(x)-\mathcal{E}(x')| < \varepsilon/3 \quad |G(x)-G(x')| < \varepsilon/3$$

Choisissons maintenant x'. D'après le lemme précédent, il existe un entier q tel que :

$$(7)\qquad \left| x - \frac{b^q}{a^{p(q)}} \right| < \eta$$

avec :

(8) $$p(q) = \lfloor q \operatorname{Log}_a b \rfloor$$

Posons $x' = b^q / a^{p(q)}$. D'après la proposition 2.5, b étant un degré-limite, on a $E(b^q) = G(b^q)$. On en déduit, d'après (8), que $\mathcal{E}_p(x') = G(x')$.
D'après le théorème 5, on sait que $\mathcal{E}_p(x') \geqslant \mathcal{E}(x') \geqslant G(x')$. On a donc :

(9) $$|\mathcal{E}(x') - G(x')| = 0$$

Reste le terme $\mathcal{E}_p(x) - \mathcal{E}(x)$. Comme $\mathcal{E}_p(x)$ converge vers $\mathcal{E}(x)$, il existe p' tel que :

(10) $$\forall_p > p' \quad |\mathcal{E}_p(x) - \mathcal{E}(x)| < \varepsilon/3$$

D'après (5), (6), (9) et (10), on a finalement :

$$|\mathcal{E}(x) - G(x)| < \varepsilon$$ ▨

On peut donc dire que C et E tendent vers $G(x) = k \times \operatorname{Log} x$.
Mais cela ne signifie pas que la différence C-E est bornée, ou que la différence C-G est bornée !

5.3. THEOREME DU DEGRE-LIMITE

Théorème 11 : A partir d'un certain rang N_2, tous les arbres optimaux de poids $n \geqslant N_2$ ont pour degré au sommet l'un des degré-limites.

Preuve : Considérons un arbre optimal de poids n et de degré au sommet d. Nous allons montrer que si n est assez grand, d est égal à l'un des degré-limites.
D'après la proposition 2.1, et la formule (2.20), on a :

(11) $$C(n) - k\, n \operatorname{Log} n \geqslant n\left[\varphi(d) - k \operatorname{Log} d\right]$$

Appelons p la poche de n ($a^p \leqslant n < a^{p+1}$). D'après la proposition 3.6, formule 3.(34), on a:

(12) $$C(n) = a^p\, \mathcal{C}_p(n/a^p) + p\, n\, \varphi(a)$$

Posons $x = n/a^p$. D'après (11) et (12), on a:

$$\mathcal{C}_p(x) - k \times \operatorname{Log} x \geqslant x\left[\varphi(d) - k \operatorname{Log} d\right]$$

D'après la proposition précédente, $\mathcal{C}_p(x)$ tend vers $k \times \operatorname{Log} x$.
Donc, si p est assez grand, c'est-à-dire si n est assez grand, le terme entre-crochets doit être nul, ce qui entraîne que d est un degré-limite. Il existe donc un entier N_2 tel que tous les arbres optimaux de poids supérieur ou égal à N_2 ont pour degré au sommet l'un des degré-limites. ▨

Dans le cas où il y a un seul degré-limite a, nous avons vu qu'à partir d'un certain rang N_2 il existe un arbre optimal de degré au sommet a.
Dans le cas où il y a plusieurs degré-limites, nous avons une propriété duale : à partir d'un certain rang N_2, tous les arbres optimaux ont pour degré au sommet l'un des degré-limites.

Quelles sont les valeurs de N_2 s'il y a plusieurs degré-limites ?
Prenons l'exemple des fonctions de coût du type $\varphi(d) = d+\lambda$. Nous avons vu en 4.4.1, que dans le cas où il y a un seul degré-limite, N_2 tend vers l'infini quand λ approche des valeurs $\psi(d)$ (voir définition en 2.6.2) pour lesquelles il y a plusieurs degré-limites. En fait, quand $\lambda = \psi(d)$, les valeurs de N_2 ne sont pas très élevées. Si $\lambda = \psi(3)$, par exemple, N_2 est égal à 7 ! Bien entendu, N_2 a un autre sens : cela signifie que pour cette fonction de coût, les arbres optimaux de poids supérieur ou égal à $N_2 = 7$ ont pour degré au sommet l'un des degré-limites, c'est-à-dire 3 ou 4.

5.3.1. FORME DES ARBRES CORRESPONDANT AUX POINTS ANGULEUX DE E. CALCUL DE E

Nous avons déjà vu (proposition 3.1.(iiii)) que les arbre réguliers dont les degrés sont tous des degré-limites correspondent à des points anguleux de E, c'est-à-dire que leur poids est l'abscisse d'un point anguleux de E.

Si n est plus grand ou égal à N_2, la réciproque est vraie :

Proposition 2 : Si n est l'abscisse d'un point anguleux de E et si $n \geqslant N_2$, alors tous les degrés des arbres réguliers correspondants sont des degré-limites.

Preuve : Supposons qu'il existe un arbre régulier correspondant à un point anguleux d'abscisse $n \geqslant N_2$ et tel qu'à une profondeur donnée le degré d de ses noeuds ne soit pas un degré-limite. L'ordre des degrés des différentes profondeurs n'influant pas sur le coût d'un arbre régulier, cet arbre est équivalent à un arbre optimal de degré au sommet d. Comme $n \geqslant N_2$, c'est contradictoire avec le théorème précédent. ▨

Corollaire 1 : Si n est l'abscisse d'un point anguleux et si $n \geqslant N_2$, alors n est égal à un produit de puissances entières de degré-limites :

$$n = \prod_{1 \leqslant i \leqslant t} d_i^{\ell_i} \qquad d_i \in \mathcal{D} \qquad \ell_i \in N^+ \tag{13}$$

Preuve : Cela découle directement de la proposition précédente. ▨

Au chapitre précédent, nous avons vu que l'enveloppe convexe E est calculable en temps fini s'il y a un seul degré-limite. Dans notre cas, ce n'est évidemment plus vrai puisque nous avons vu au paragraphe 2 que $\mathcal{E}_p$ évolue constamment et tend asymptotiquement vers G. Cependant, la formule (13) nous donne un moyen de calcul rapide des points anguleux de E d'abscisse supérieure ou égale à N_2.

5.4. DIFFERENCES DE POIDS ET DE PROFONDEURS NON BORNEES

Reportons-nous à la définition 4.8. Pour un arbre T donné, nous avons appelé $\Delta(T)$ la différence maximum de profondeur de ses feuilles. Pour une fonction de coût φ donnée et pour n donné, nous avons appelé $\Delta_\varphi(n)$ le minimum de Δ sur tous les arbres de poids n optimaux pour la fonction de coût φ.

Nous avons défini de même Q(T) et $Q_\varphi(n)$ pour la différence de poids des sous-arbres principaux (voir définition 4.7). Nous avons montré que s'il y a un seul degré-limite, $\Delta_\varphi(n)$ et $Q_\varphi(n)$ sont bornés respectivement par des constantes Δ_φ et Q_φ (propositions 4.8 et 4.9).

Nous allons montrer que s'il y a plusieurs degré limites (avec l'hypothèse (1)), $\Delta_\varphi(n)$ et $Q_\varphi(n)$ ne sont pas bornés. Qu'est ce que cela signifie exactement ? Pour les différences de profondeur, cela peut s'exprimer par la formule suivante :

$$\forall N \qquad \exists n \qquad \Delta_\varphi(n) \geqslant N$$

ou bien, si $\mathcal{C}_\varphi(n)$ désigne l'ensemble des arbres optimaux de poids n :

$$\forall N \qquad \exists n \qquad \forall T \in \mathcal{C}_\varphi(n) \qquad \Delta(T) \geqslant N$$

Cela signifie que pour tout N arbitrairement grand, on peut trouver des valeurs de n telles que tous les arbres optimaux de poids n ont une différence Δ de profondeur des feuilles supérieure à N. De même, les différences de poids peuvent être rendues arbitrairement grandes.

Cette propriété est assez surprenante. En effet, elle semble en contradiction avec la proposition 2.1, qui indique que les arbres optimaux ont tendance à être équilibrés. Nous n'avons pas trouvé d'explication réellement satisfaisante à cette contradiction.
Nous allons faire la démonstration dans le cas où la fonction de coût ne possède que deux degré-limites a et b (a < b) vérifiant l'hypothèse (1). La présence d'autres degré-limites ne ferait que compliquer inutilement la démonstration. La méthode de preuve est la suivante : trouver des valeurs de n pour lesquelles l'arbre optimal est unique et les différences Δ et Q arbitrairement grandes. Nous avons besoin du lemme suivant :

Lemme 2 : Soit n tel que le degré d soit possible pour n et tel que n et n/d ne soient pas des abscisses de points anguleux de E. Appelons e_1 et e les abscisses des points anguleux les plus proches de n :
$e_1 < n < e$

Appelons e'_1 et e' les abscisses des points anguleux les plus proches de n/d :

(14) $\quad e'_1 < n/d < e'$

Alors :

(i) $\quad d.e' > e \quad \Rightarrow \quad C(n) > E(n)$

(ii) $\quad d.e'_1 < e_1 \quad \Rightarrow \quad C(n) > E(n)$

Preuve : Nous ferons la démonstration pour (i) seulement. La démonstration de (ii) est du même type. Comme d est possible pour n (voir définition 4.4), on a, pour certains $n_1, \ldots, n_d$:

$$C(n) = \varphi(d).n + \sum_{1 \leqslant i \leqslant d} C(n_i)$$

Comme $C \geqslant E$ et comme E est convexe, on en déduit :

(15) $\quad C(n) \geqslant \varphi(d).n + d.E(n/d)$

Posons :

(16) $\quad n/d = \alpha_1.e'_1 + \alpha.e' \qquad \alpha_1 + \alpha = 1$

Comme E est une droite entre les points anguleux consécutifs d'abscisses e'_1 et e_1, on a :

(17) $\quad E(n/d) = \alpha_1.E(e'_1) + \alpha.E(e')$

On déduit de (15), (16) et (17) que :

$$C(n) \geqslant \alpha_1 \left[\varphi(d).d.e'_1 + d.E(e'_1)\right] + \alpha \left[\varphi(d).d.e' + d.E(e')\right]$$

D'après la proposition 3.2, on en déduit :

(18) $\quad C(n) \geqslant \alpha_1.E(d\,e'_1) + \alpha.E(d\,e')$

Appelons D(x) l'équation de la droite passant par les points de E d'abscisse $(d\,e'_1)$ et $(d\,e')$.

D'après (16), D(n) est égal à :

$$D(n) = \alpha_1.D(d\,e'_1) + \alpha.D(d\,e')$$

D'après (18), on en déduit :

(19) $\quad C(n) \geqslant D(n)$

D'autre part, d'après les hypothèses du lemme, on a :

$$d\,e'_1 < n < e < d\,e'$$

Comme il y a un point anguleux d'abscisse e entre $(d\,e'_1)$ et $(d\,e')$, la droite D est strictement supérieure à E entre $(d\,e'_1)$ et $(d\,e')$.
On a donc : $D(n) > E(n)$. On en déduit bien, d'après (19) que :

$$C(n) > E(n).$$

La figure suivante aidera à comprendre cette preuve :

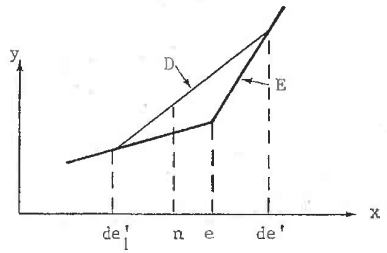


FIGURE 1.

La définition suivante va nous permettre de trouver des valeurs de n pour lesquelles les arbres optimaux sont uniques et ont une grande différence de profondeurs des feuilles.

Définition 1 : (i) Pour q entier non nul, nous définissons :

(20) $$p(q) = \lfloor q \operatorname{Log}_a b \rfloor$$

(21) $$y(q) = q \operatorname{Log}_a b - p(q)$$

(ii) Nous appelons Q l'ensemble des entiers q tels que

(22) $$a^{p(q)}/b \geqslant N_2$$

(23) $$y(q) < \operatorname*{Min}_{1 \leqslant s \leqslant q-1} y(s)$$

Notons que :

$$y(q) \in [0,1[$$

$$a^{p(q)} < b^q < a^{p(q)+1}$$

Notons d'autre part: que l'ensemble Q comporte une infinité d'entiers. En effet, y(q) s'écrit :

$$y(q) = q\,z - \lfloor q\,z \rfloor \qquad \text{avec } z = \operatorname{Log}_a b$$

Nous avons vu dans la preuve du lemme 1 que y(q) est dense dans $[0,1]$ quand q varie. Il y a donc une infinité de valeurs de q pour lesquelles y(q) est aussi proche que possible de zéro. L'ensemble Q est donc infini.

Lemme 3: Si $q \in Q$ et si a et b sont les seuls degré-limites, on a :

(i) $a^{p(q)}$ et b^q sont les abscisses de points anguleux consécutifs.

(ii) $a^{p(q)+1}$ et $(a.b^q)$ sont les abscisses de points anguleux consécutifs.

(iii) Si $n \in]a^{p(q)}, b^q[$ alors C(n) > E(n)

Preuve de (i) et (ii) : Nous ferons la démonstration uniquement pour $a^{p(q)}$ et b^q. La preuve est identique dans l'autre cas. La démonstration se fait par l'absurde. Supposons qu'il existe un point anguleux d'abscisse n entre $a^{p(q)}$ et b^q. D'après (22), n est plus grand que N_2. D'après le corollaire 1, comme il y n'y a que deux degré-limites a et b, n est de la forme :

$$n = a^r b^s \qquad (r \text{ et } s \text{ entiers})$$

On a donc :

$$a^{p(q)} < a^r b^s < b^q \qquad \text{et} \qquad s < q$$

En prenant le logarithme, on obtient :

$$p(q) < r + s \operatorname{Log}_a b < q \operatorname{Log}_a b$$

Avec les notations de la définition 1, cela s'écrit :

$$[p(q) - r - p(s)] < y(s) < [p(q) - r - p(s)] + y(q)$$

On en déduit donc y(s) < y(q). Comme s < q, c'est contradictoire avec l'hypothèse $q \in Q$.

Preuve de (iii) : Soit n vérifiant :

(24) $$a^{p(q)} < n < b^q$$

Comme $n > N_2$ (d'après (22)), seuls les degré-limites a et b sont possibles pour n.

Considérons d'abord le cas où a est possible pour n.
Appelons e' l'abscisse du point anguleux le plus proche de n/a et tel que :

(25) $$n/a \leqslant e'$$

D'après (22), on a $e' \geqslant N_2$. D'après le corollaire 1, l'entier e' est donc de la forme :

$$e' = a^r b^s \qquad (r \text{ et } s \text{ entiers})$$

On a donc

$$ae' = a^{r+1} b^s$$

D'après la proposition 3.1.(iiii), l'entier (ae') est donc l'abscisse d'un point anguleux de E. Comme $a^{p(q)} < n \leqslant ae'$ et que $a^{p(q)}$ et b^q sont les abscisses de points anguleux consécutifs, on en déduit que :

$$b^q \leqslant a^{r+1} b^s$$

Mais l'égalité est impossible car elle contredirait l'hypothèse (1) sur a et b (on aurait en effet : $b^{q-s} = a^{r+1}$). On a donc :

(26) $$ae' > b^q$$

De (24) et (26), on déduit que :

(27) $$n/a < e'$$

L'entier n/a n'est donc pas l'abscisse d'un point anguleux. D'autre part, l'entier n n'est pas non plus l'abscisse d'un point anguleux. D'après les inégalités (24), (26) et (27), nous pouvons donc appliquer le lemme 2.(i) avec $d = a$ et $e = b^q$. On a donc bien :

$$C(n) > E(n)$$

Dans le cas où le degré b est possible pour n, la démonstration est identique, en utilisant le lemme 2.(ii).

▨

Proposition 3 : Si a et b sont les seuls degré-limites, alors, pour les valeurs de n suivantes :

(28) $$n = j\, b^q + (a-j)\, a^{p(q)} \quad \text{avec} \begin{cases} q \in Q \\ j \text{ entier} \\ 1 \leqslant j \leqslant a-1 \end{cases}$$

On a :

(29) $$\Delta_\varphi(n) = p(q) - q$$

(30) $$Q_\varphi(n) = b^q - a^{p(q)}$$

L'arbre T optimal de poids n est unique à une permutation près des sous-arbres principaux. Son coût vérifie $C(n) = E(n)$ et il a la forme suivante :

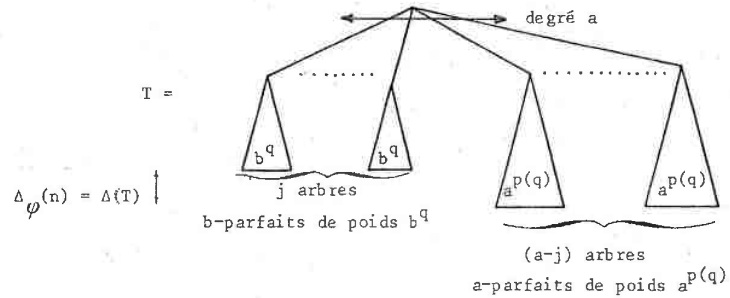


FIGURE 2.

Preuve : La preuve se décompose en plusieurs parties. Nous allons d'abord montrer que l'arbre T ci-dessus est optimal pour les valeurs de n définies en (28). Nous montrerons ensuite que pour ces valeurs de n, il n'existe pas d'arbre optimal de degré au sommet b. Nous en déduirons enfin que T est l'unique arbre optimal pour ces valeurs de n.

Nous allons d'abord mettre en évidence une propriété de D(x), droite passant par les points de E d'abscisses $a^{p(q)}$ et b^q. Comme $a^{p(q)}$ et $b^{(q)}$ sont les abscisses de points anguleux consécutifs de E (lemme 3.(i)), on a :

(31) si $x \in [a^{p(q)}, b^q]$ alors $E(x) = D(x)$

(32) si $x \notin [a^{p(q)}, b^q]$ alors $E(x) > D(x)$

Comme $C(x) \geqslant E(x)$, on en déduit que :

(33) $$C(x) \geqslant D(x)$$

Montrons qu'il n'y a égalité que si x est égal à $a^{p(q)}$ ou b^q.
D'après (32) et (33), on a :

(34) si $x \notin [a^{p(q)}, b^q]$ alors $C(x) > D(x)$

D'après le lemme 3.(iii), on a :

(35) si $x \in]a^{p(q)}, b^q[$ alors $C(x) > E(x)$

puisque C et E sont des interpolations linéaires entre les valeurs entières.

On déduit de (31) et (35) que :

(36) si $x \in]a^{p(q)}, b^q[$ alors $C(x) > D(x)$

Finalement, d'après (34) et (36), on a :

(37) $C(x) = D(x) \Longrightarrow x = a^{p(q)}$ ou $x = b^q$

(Rappelons que C et E sont égaux en $a^{p(q)}$ et b^q puisque ce sont des abscisses de points anguleux).

L'arbre T est optimal pour n et C(n) = E(n)

Calculons le coût de T :

$$\mathcal{C}(T) = \varphi(a).n + j.C(b^q) + (a-j).C(a^{p(q)})$$

D'après la définition de n, on en déduit :

$$\mathcal{C}(T) = j\left[\varphi(a).a.b^q + a.C(b^q)\right] + (a-j)\left[\varphi(a).a.a^{p(q)} + a.C(a^{p(q)})\right]$$

Mais les termes entre-crochets sont précisément les coûts des arbres optimaux de poids $(a.b^q)$ et $\left(a^{p(q)+1}\right)$. On a donc :

$$\mathcal{C}(T) = j.C(a\,b^q) + (a-j).C\left(a^{p(q)+1}\right)$$

Comme ces arbres correspondent à des points anguleux, on en déduit :

$$\mathcal{C}(T) = j.E(a.b^q) + (a-j).E\left(a^{p(q)+1}\right)$$

Les entiers $(a.b^q)$ et $\left(a^{p(q)+1}\right)$ étant les abscisses de points anguleux consécutifs de l'enveloppe convexe E, on en déduit, d'après (28), que le point $\left(n, \mathcal{C}(T)\right)$ est sur le segment de E joignant ces points anguleux. On a donc $\mathcal{C}(T) = E(n)$. L'arbre T est donc optimal et on a :

(38) $C(n) = E(n)$

Le degré b est impossible pour n

La démonstration se fait par l'absurde. Nous allons montrer que si le degré b était possible pour n, on aurait $C(n) > E(n)$, ce qui contredit (38).

Supposons donc que le degré b soit possible pour n et appelons e_1' l'abscisse du point anguleux le plus proche de n/b et tel que :

$$e_1' \leqslant n/b$$

La suite du raisonnement est la même que pour la preuve du lemme 3.(iii). On montre que n/b n'est pas l'abscisse d'un point anguleux (c'est-à-dire que $e_1' < n/b$) et que :

$$be_1' < a^{p(q)+1}$$

Comme n n'est pas l'abscisse d'un point anguleux, on peut appliquer le lemme 2.(ii) avec d = b et $e_1 = a^{p(q)+1}$. On en déduit que $C(n) > E(n)$, ce qui contredit (38).

L'arbre T est l'unique arbre optimal de poids n

Comme le degré b est impossible pour n et que $n \geqslant N_2$, tous les arbres optimaux de poids n ont pour degré au sommet a.

Appelons T' l'un deux et appelons $n_1,\ldots,n_a$ les poids de ses sous-arbres principaux. Comme T' est optimal, on a :

$$C(n) = \varphi(a).n + \sum_{1\leqslant i\leqslant a} C(n_i) \qquad \text{avec} \qquad \sum_{1\leqslant i\leqslant a} n_i = n \tag{39}$$

Calculons maintenant E(n). D'après (28), n est dans l'intervalle

$$n \in \left[a^{p(q)+1},\ a.b^q\right]$$

On en déduit facilement que :

$$E(n) = \varphi(a).n + a.D(n/a)$$

où D est la droite définie précédemment. On en déduit que :

$$E(n) = \varphi(a).n + \sum_{1\leqslant i\leqslant a} D(n_i) \tag{40}$$

D'après (39) et (40), on a :

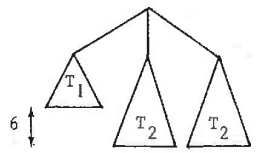
$$\sum_{1\leqslant i\leqslant a} \left[C(n_i)-D(n_i)\right] = 0$$

D'après (33), on en déduit que $C(n_i) = D(n_i)$. D'après (37), les entiers n_i sont donc égaux, soit à $a^{p(q)}$, soit à b^q. L'arbre T est donc l'unique arbre optimal, à une permutation près des sous-arbres principaux. ▨

Bien entendu, il existe d'autres valeurs de n pour lesquelles les différences de profondeurs et de poids des sous-arbres principaux sont arbitrairement grandes.
Donnons un exemple. Si $\varphi(d) = \lambda+\psi(3)$ (voir 2.6.2 et 4.4.1) il y a deux degré-limites : 3 et 4. L'entier q = 23 fait partie de l'ensemble Q de la définition 1 et on a p(q) = 29. D'après la proposition précédente,

pour $n = 4^{23} + 2.3^{29}$, l'unique arbre optimal est :

$\Delta_\varphi(n) = 6$
$Q_\varphi(n) = 4^{23} - 3^{29}$



où T_1 est l'arbre 4-parfait de profondeur 23 et T_2 l'arbre 3-parfait de profondeur 29.

Théorème 12 : Si la fonction de coût φ possède plusieurs degré-limites, vérifiant l'hypothèse(1), les différences de profondeurs des feuilles $\Delta_\varphi(n)$ et les différences de poids $Q_\varphi(n)$ ne sont pas bornées. Plus précisément :

$\forall N \quad \exists n \qquad \Delta_\varphi(n) \geqslant N$
$\forall N \quad \exists n \qquad Q_\varphi(n) \geqslant N$

Preuve : Dans le cas où il y a deux degré-limites a et b (a < b) vérifiant l'hypothèse (1), la proposition précédente nous indique que pour les valeurs de n suivantes :

$$n = b^q + (a-1).a^{p(q)} \qquad \text{avec} \quad \begin{cases} q \in Q \text{ (définition 1)} \\ p(q) = \left\lfloor q \operatorname{Log}_a b \right\rfloor \end{cases} \tag{41}$$

on a :

$$\Delta_\varphi(n) = p(q) - q \quad \text{et} \quad Q_\varphi(n) = b^q - a^{p(q)}$$

Montrons que pour ces valeurs de n la différence de profondeurs $\Delta_\varphi(n)$ n'est pas bornée. La démonstration est analogue pour $Q_\varphi(n)$. On a :

$$\Delta_\varphi(n) = \left\lfloor q.\operatorname{Log}_a b \right\rfloor - q$$

On en déduit :

$$\Delta_\varphi(n) > q.\operatorname{Log}_a b - 1 - q$$

ce qui s'écrit :

$$\Delta_\varphi(n) > q.(\operatorname{Log}_a b - 1) - 1 \tag{42}$$

L'ensemble Q étant infini, les entiers q de l'ensemble Q peuvent être arbitrairement grands. D'après (42), on en déduit que $\Delta_\varphi(n)$ n'est pas borné.

La démonstration se généralise au cas où il y a plusieurs degré-limites dont deux au moins vérifient l'hypothèse (1). ▨

CHAPITRE 6

ARBRES OPTIMAUX A POIDS INEGAUX

6.1. PRESENTATION

Dans les chapitres précédents, nous avons étudié la forme et le coût des arbres optimaux à poids égaux. Nous nous plaçons ici dans le cas général où l'on cherche la forme et le coût des arbres optimaux pour une suite de poids quelconque $S = (a_1,\ldots,a_n)$. Nous supposons toujours que la fonction de coût est à degré borné, c'est-à-dire qu'il existe, pour toutn-uple S, un arbre optimal dont tous les degrés sont inférieurs ou égaux à un entier d_{max}. Au chapitre I, le théorème I et les corollaires 1.4 et 1.5 donnent des conditions suffisantes sur φ pour que la fonction de coût soit à degré borné.

Il s'agit donc, pour une suite $S = (a_1,\ldots,a_n)$ de réels positifs ou nuls, de trouver l'arbre de suite S de coût minimum.

Nous étudions d'abord comment varie le coût optimal C(S) avec le n-uple S . A poids total constant, nous verrons que plus le n-uple est "équilibré", plus le coût optimal augmente.
Nous avons vu que pour les arbres à poids égaux le coût optimal est borné inférieurement et supérieurement. Nous montrerons que l'on peut étendre ce résultat aux cas des arbres à poids inégaux.

Nous avons vu aux chapitres précédents que les arbres optimaux ont tendance à avoir pour degré les entiers d qui minimisent $\varphi(n)/\text{Log } n$:

$$(1) \qquad \forall n \text{ entier} \qquad \frac{\varphi(d)}{\text{Log } d} \leqslant \frac{\varphi(n)}{\text{Log } n}$$

Ces entiers sont appelés degré-limites et leur ensemble est noté $\mathcal{D}$.
Nous ne distinguerons pas ici les cas où il y a un seul ou plusieurs degré-limites. Mais nous montrerons que dans le cas général, les arbres optimaux ont également tendance à avoir pour degré les degré-limites.

En particulier, l'algorithme de HUFFMAN [7], utilisé avec l'un de ces degrés constitue une bonne heuristique.

En ce qui concerne la complexité de l'algorithme de recherche de l'arbre optimal, nous n'avons pu mettre en évidence un algorithme en temps polynomial en n. Cependant, pour certains n-uples, il est possible de calculer rapidement l'arbre optimal, c'est-à-dire en temps polynomial - ou même linéaire - en fonction de n. Nous indiquerons également des heuristiques.

Rappelons que a désigne le plus petit degré-limite et que

$$k = \frac{\varphi(a)}{\text{Log } a} = \frac{\varphi(d)}{\text{Log } d} \quad \text{pour } d \in \mathcal{D} \tag{2}$$

Définition 1 : Pour un n-uple $S = (a_1,\ldots,a_n)$ nous définissons :

- le minimum m(S) de S : $m(S) = \underset{1\leqslant i\leqslant n}{\text{Min}}\ a_i$
- le poids N(S) de S : $N(S) = \sum_{1\leqslant i\leqslant n} a_i$
- l'entropie H(S) de S : $H(S) = \sum_{1\leqslant i\leqslant n} \frac{a_i}{N(S)} \text{Log} \frac{N(S)}{a_i}$

6.2. VARIATION DU COUT OPTIMAL AVEC LE N-UPLE

Il s'agit d'étudier les facteurs qui interviennent dans la valeur du coût optimal. Dans certains cas, cela permet de comparer le coût optimal correspondant à deux n-uples donnés. Si l'on connaît le coût optimal de l'un deux, on en déduit des indications sur celui de l'autre. Les règles suivantes peuvent donc servir à trouver de bonnes approximation du coût optimal :

Proposition 1 :

(i) $\forall c > 0 \quad C(c.a_1,\ldots,c.a_n) = c.C(a_1,\ldots,a_n)$

(ii) $\forall b \geqslant 0 \quad C(a_1,\ldots,a_n) \leqslant C(a_1,\ldots,a_n,b)$

(iii) $\forall c > 0 \quad C(a_1,\ldots,a_i,\ldots a_n) < C(a_1,\ldots,a_i+c,\ldots,a_n)$

Preuve : La formule (i) résulte de la proposition 1.3 ; la formule (ii) découle du lemme 1.1, puisque la fonction φ est supposée croissante. Montrons (iii) : si dans l'arbre optimal pour la suite $(a_1,\ldots,a_i+c,\ldots,a_n)$ nous remplaçons la feuille de poids a_i+c par une feuille de poids a_i, nous obtenons un arbre de coût moindre pour la suite $(a_1,\ldots,a_i,\ldots,a_n)$; il en résulte (iii).

▨

Proposition 2 : A poids constant, plus le n-uple est équilibré, plus le coût optimal augmente. Plus précisément :

si : $a'_i + a'_j = a_i + a_j$

$|a'_j - a'_i| \leqslant |a_j - a_i|$

alors :

$C(a_1,\ldots,a_i,\ldots,a_j,\ldots,a_n) \leqslant C(a_1,\ldots,a'_i,\ldots,a'_j,\ldots,a_n)$

Preuve : C'est évident si $a_i = a_j$. Supposons donc $a_i < a_j$ et $a'_i \leqslant a'_j$. On a donc $a_i \leqslant a'_i$. Posons :

$S = (a_1,..,a_i,..,a_j,\ldots,a_n)$

$S' = (a_1,\ldots,a'_i,\ldots,a'_j,\ldots,a_n)$

Appelons T' l'arbre optimal pour S' et c'_i et c'_j les coûts partiels des feuilles de poids a'_i et a'_j dans T'. D'après le corollaire 1.1, comme $a'_i \leqslant a'_j$, on a $c'_i \geqslant c'_j$ (si $a'_i = a'_j$, il suffit d'échanger les indices i et j). Remplaçons dans T' les feuilles de poids a'_i et a'_j par des feuilles de poids a_i et a_j respectivement. On obtient ainsi un arbre T_1 dont le coût vérifie :

$$\mathcal{C}(T_1) - \mathcal{C}(T') = (a_i - a'_i)(c'_i - c'_j)$$

Comme $a_i \leqslant a'_i$ et $c'_i \geqslant c'_j$, on en déduit que :

$$\mathcal{C}(T_1) \leqslant \mathcal{C}(T')$$

Comme par définition $C(S') = \mathcal{C}(T')$ et $C(S) \leqslant \mathcal{C}(T_1)$, on en déduit bien $C(S) \leqslant C(S')$.

▨

Corollaire 1 : Pour tout n-uple $S = (a_1,\dots,a_n)$, on a :

$$C(a_1,\dots,a_n) \leqslant C\left(\frac{N(S)}{n},\dots,\frac{N(S)}{n}\right) \tag{3}$$

Preuve : On peut passer du n-uple $(a_1,,,,,a_n)$ au n-uple $\left(N(S)/n,\dots,N(S)/n\right)$ par une suite de transformations du type de celle décrite à la proposition 2. Comme, d'après cette proposition, le coût augmente à chaque fois, on en déduit bien (3). ▨

6.3. BORNE INFERIEURE DU COUT OPTIMAL

Aux chapitres 2 et 3, nous avons vu que l'enveloppe convexe E et la fonction G sont des bornes inférieures du coût optimal C, pour les arbres optimaux à poids égaux. Nous verrons qu'à partir des fonctions G et E, on peut construire des bornes inférieures du coût optimal $C(a_1,\dots,a_n)$ dans le cas général. Nous allons d'abord donner une méthode générale de construction de bornes inférieures.

6.3.1. FORME GENERALE DES BORNES INFERIEURES

Définition 2 : A toute fonction g des réels dans les réels, nous faisons correspondre les trois fonctions suivantes :

(i) une fonction Z_g sur les n-uples

$$Z_g(a_1,\dots,a_n) = g\left(\sum_{1\leqslant i\leqslant n} a_i\right) - \sum_{1\leqslant i\leqslant n} g(a_i) \tag{4}$$

(ii) une fonction W_g sur les noeuds v d'un arbre

$$W_g(v) = \varphi\left(d(v)\right).w(v) - g\left(w(v)\right) + \sum_{u\in\mathcal{F}(v)} g\left(w(u)\right) \tag{5}$$

(iii) une fonction Z'_g sur les arbres

$$Z'_g(T) = \sum_{v\in\mathcal{N}_T} W_g(v) \tag{6}$$

On peut remarquer que W_g et Z'_g dépendent de la donnée de la fonction de coût, c'est-à-dire de φ.

Lemme 1 : Le coût $\mathcal{C}(T)$ d'un arbre quelconque vérifie :

$$\mathcal{C}(T) = Z_g(S_T) + Z'_g(T) \tag{7}$$

Preuve : Il suffit de calculer $Z'_g(T)$ et de remarquer que la somme sur tous les noeuds de l'arbre du premier terme de $W_g(v)$ est égale à $\mathcal{C}(T)$ et que la somme sur tous les noeuds de l'arbre du deuxième et du troisième terme est égale à $(-Z_g(S_T))$. ▨

Nous définissons maintenant une classe de fonctions qui vont nous permettre de construire des bornes inférieures.

Définition 3 : (i) On appelle $\mathcal{G}$ l'ensemble des fonctions réelles g qui satisfont les conditions suivantes :

$$\begin{cases} g(x) \leqslant \varphi(d).x + \sum\limits_{1\leqslant i\leqslant d} g(x_i) \\ \text{avec} \quad \sum\limits_{1\leqslant i\leqslant d} x_i = x \\ d \text{ entier}, \quad 2 \leqslant d \leqslant d_{max} \end{cases} \tag{8}$$

(ii) On appelle $\mathcal{G}(S)$ le sous-ensemble des fonctions de $\mathcal{G}$ dont l'intervalle de définition contient $\left[m(S), N(S)\right]$

Lemme 2 : Si $g \in \mathcal{G}(S)$, si T est un arbre quelconque de suite S et v un noeud quelconque de T, on a :

$$W_g(v) \geqslant 0 \qquad \text{et} \qquad Z'_g(T) \geqslant 0$$

Preuve : Si l'on écrit la formule (8) avec $x = w(v)$ et $x_i = w(u)$ $\left(\text{pour } 1\leqslant i\leqslant d \text{ et } u\in\mathcal{F}(v)\right)$, on voit que $W_g(v)$ est positif ou nul. D'après (6), on en déduit que $Z'_g(T) \geqslant 0$. ▨

Théorème 12 : Pour tout n-uple S, pour toute fonction g de $\mathcal{G}(S)$ le coût optimal C(S) est minoré par $Z_g(S)$:

(9) $$\forall S \quad \forall g \in \mathcal{G}(S) \qquad C(S) \geqslant Z_g(S)$$

ce qui s'écrit :

(10) $$C(a_1,\ldots,a_n) \geqslant g\left(\sum_{1\leqslant i\leqslant d} a_i\right) - \sum_{1\leqslant i\leqslant d} g(a_i)$$

Preuve : Considérons un arbre T quelconque de suite S. D'après le lemme 1, on a :

$$\mathcal{C}(T) = Z_g(S) + Z'_g(T)$$

D'après le lemme précédent, on a $Z'_g(T) \geqslant 0$. On en déduit :

(11) $$\mathcal{C}(T) \geqslant Z_g(S).$$

Comme T est un arbre quelconque de suite S, cette inégalité reste valable pour l'arbre optimal. ▨

Pour une suite S donnée, il y a une infinité de fonctions g dans $\mathcal{G}(S)$. Le coût optimal C(S) est borné inférieurement par tous les termes $Z_g(S)$ correspondants. Il en résulte que si nous trouvons une fonction $g \in \mathcal{G}(S)$ et un arbre T dont le coût est égal à $Z_g(S)$ alors l'arbre T est optimal pour S.

Définition 4 : On dit qu'une fonction g est optimale pour une suite S si

$$g \in \mathcal{G}(S) \quad , \quad C(S) = Z_g(S)$$

On dit qu'une fonction g est optimale pour un arbre T si g est optimale pour S_T.

Lemme 3 : Si g est optimale pour un arbre T, alors T est optimal et $W_g(v)$ est nul sur tous les noeuds v de T.

Preuve : Si g est optimale pour T, alors $\mathcal{C}(T) = Z_g(S_T)$. D'après le théorème précédent, l'arbre T est donc optimal. D'après le lemme 1, on en déduit que $Z'_g(T) = 0$, et, d'après le lemme 2, que $W_g(v) = 0$ sur tous les noeuds de T. ▨

6.3.2. EXEMPLES DE BORNES INFERIEURES DU COUT OPTIMAL

Dans le cas des arbres à poids égaux, chaque fois que le coût optimal C(n) est égal à l'une des bornes inférieures E(n) ou G(n), les arbres optimaux correspondants ont des propriétés particulières. Dans notre cas, il y a une infinité de bornes inférieures $Z_g(S)$ possible. Nous allons montrer que l'on peut prendre pour g les fonctions C, E et G et étudierons la forme des arbres optimaux pour lesquels ces fonctions sont optimales (au sens de la définition 4).

Corollaire 2 : Si $m(S) \geqslant 1$, alors $C(S) \geqslant Z_E(S)$, où E désigne l'enveloppe convexe définie en 3.3.

Preuve : D'après la proposition 3.2.(i), la fonction E satisfait la condition (8). On a donc $E \in \mathcal{G}$. Comme E est définie sur $[1,\infty[$, on a bien $E \in \mathcal{G}(S)$ puisque $m(S) \geqslant 1$. Il suffit alors d'appliquer le théorème 12. ▨

Corollaire 3 : Si $m(S) \geqslant 1$, alors $C(S) \geqslant Z_C(S)$, où C est le coût optimal des arbres à poids égaux et unitaires.

Preuve : D'après la formule (2.2), la fonction C satisfait la condition (8). La suite de la preuve est identique à celle du corollaire 2. ▨

Ce corollaire peut s'écrire :

(12) $$C(a_1,\ldots,a_n) \geqslant C\left(\sum_{1\leqslant i\leqslant n} a_i\right) - \sum_{1\leqslant i\leqslant n} C(a_i)$$

Pour quels arbres la fonction C est-elle optimale ? D'abord pour les contractions d'arbres optimaux à poids égaux. Considérons en effet, un arbre optimal T à n feuilles de poids égaux et unitaires.

Contractons le par un ensemble de noeuds indépendants $v_1,\ldots,v_t$, de poids $n_1,\ldots,n_t$. D'après la proposition 1.4, l'arbre T' obtenu est également optimal pour une certaine suite S' de poids n et on a :

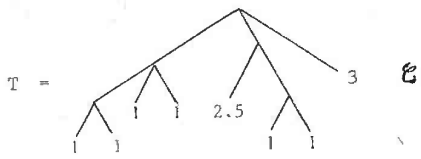
$$\mathcal{C}(T) = \mathcal{C}(T') + \sum_{1\leqslant i\leqslant t} \mathcal{C}(T(v_i))$$

ce qui s'écrit :

$$C(S') = C(n) - \sum_{1\leqslant i\leqslant t} C(n_i) \qquad (13)$$

Le deuxième membre de l'égalité (13) étant égal à $Z_C(S')$, la fonction C est bien optimale pour l'arbre T'.

Mais la fonction C peut être optimale pour des arbres qui ne sont pas des contractions d'arbres optimaux à poids égaux. Par exemple, si $\varphi(d) = d$, l'arbre suivant est optimal pour la suite S = (1,1,1,1,1,1,2.5,3)



$\mathcal{C}(T) = C(S) = 63.5$

FIGURE 1.

Si l'on se reporte à 2.6.3, on voit que cet arbre n'est contraction d'au arbre optimal à poids égaux. Et pourtant, la fonction C est optimale pou c'est-à-dire : $Z_C(S) = \mathcal{C}(T)$. En effet, à l'aide des formules (2.35), on vérifie que :

$$Z_C(S) = C(11.5) - C(3) - C(2.5) = 63.5$$

Cela nous montre d'une part que les arbres optimaux pur un n-uple quelconque ne sont pas tous des contractions d'arbres optimaux à poids égaux, et d'autre part, que la recherche d'une fonction g optimale pour un arbre T donné est un moyen efficace de tester l'optimalité de T. Malheureusement, comme nous le verrons plus loin, il existe des arbres optimaux auxquels ne correspond aucune fonction g optimale.

Corollaire 4 : Pour tout n-uple S, on a :

$$C(S) \geqslant Z_G(S) \qquad \text{avec } G(x) = k\ x\ \mathrm{Log}\ x$$

$$C(S) \geqslant k\ N(S)\ H(S) \qquad (14)$$

Preuve : La fonction G est la borne inférieure des chapitres précédents et H est l'entropie de la définition 1. On vérifie facilement que :

$$Z_G(S) = k\ N(S)\ H(S)$$

Pour terminer la preuve, il nous faut donc vérifier que G satisfait la condition (8), c'est-à-dire :

$$\begin{cases} k\ x\ \mathrm{Log}\ x \leqslant \varphi(d)\ x + \sum_{1\leqslant i\leqslant d} k\ x_i\ \mathrm{Log}\ x_i \\ \text{avec} \sum_{1\leqslant i\leqslant d} x_i = x \\ d \text{ entier, } 2 \leqslant d \leqslant d_{max} \end{cases} \qquad (15)$$

La fonction $x.\mathrm{Log}\ x$ étant convexe, on a :

$$\sum_{1\leqslant i\leqslant d} x_i\ \mathrm{Log}\ x_i \geqslant d\ \frac{x}{d}\ \mathrm{Log}\ \frac{x}{d}$$

On en déduit que :

$$\varphi(d).x + \sum_{1\leqslant i\leqslant d} k\ x_i\ \mathrm{Log}\ x_i \geqslant k\ x\ \mathrm{Log}\ x + x\left[\varphi(d) - k\ \mathrm{Log}\ d\right] \qquad (16)$$

Le terme entre-crochets étant positif ou nul par définition de k, on en déduit bien (15). ▨

Pour quels arbres la fonction G est-elle optimale ? Calculons la différence entre le coût d'un arbre T quelconque et la borne inférieure $Z_G(S_T)$. D'après le lemme 1, et la définition 2, cette différence s'écrit :

$$\mathcal{C}(T) - Z_G(S_T) = \sum_{v\in\mathcal{N}_T} W_G(v). \qquad (17)$$

Par un calcul analogue à celui effectué au lemme 2.1, on montre que :

$$W_G(v) = w(v)\left[\Delta_1(v) + \Delta_2(v)\right] \tag{18}$$

$$\Delta_1(v) = \varphi\left(d(v)\right) - k \operatorname{Log} d(v) \tag{19}$$

$$\Delta_2(v) = k\left[\operatorname{Log} d(v) + \sum_{u \in \mathcal{F}(v)} \frac{w(u)}{w(v)} \operatorname{Log} \frac{w(u)}{w(v)}\right] \tag{20}$$

Par définition de k, on a $\Delta_1(v) \geq 0$, et $\Delta_1(v)$ ne s'annule que si d(v) est un degré-limite. D'autre part, la fonction x.Log x étant convexe, on a $\Delta_2(v) \geq 0$, et $\Delta_2(v)$ ne s'annule que si tous les fils de v ont même poids w(v)/d.

La fonction G sera donc optimale pour tout arbre T vérifiant les deux conditions suivantes :

1) tous les degrés de T sont des degré-limites,
2) les fils de chaque noeud ont même poids.

S'il n'y a qu'un seul degré-limite a, G sera optimale uniquement pour les contractions d'arbres a-parfaits.

6.3.3. EXISTE-T-IL UNE BORNE INFERIEURE "UNIVERSELLE" ?

Existe-t-il une fonction g telle que pour tout n-uple S on ait : $C(S) = Z_g(S)$? Si c'était vrai, le calcul du coût optimal serait immédiat. Mais la réponse est non. On peut alors se demander si à tout n-uple S correspond une fonction g optimale pour S. La réponse est encore non, comme le montre le contre-exemple suivant .
Nous montrerons en 6.6.3, que l'arbre T suivant est optimal si $\varphi(d) = d$ pour le 4-uple (1,1,2,4).

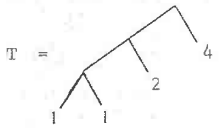


FIGURE 2.

D'après le lemme 3, une fonction g optimale pour T doit vérifier $W_g(v) = 0$ sur chacun des noeuds de T ; cela s'écrit :

$$g(2) = 4 + 2.g(1)$$
$$g(4) = 8 + 2.g(2)$$
$$g(8) = 16 + 2.g(4)$$

ce qui entraîne :

$$g(8) = 48 + 8\, g(1) \tag{21}$$

D'après la définition de g, on doit avoir :

$$g(3) \leq 3.\varphi(3) + 3.g(1)$$
$$g(8) \leq 8.\varphi(3) + g(3) + g(3) + g(2)$$

On déduit des deux inégalités précédentes que :

$$g(8) \leq 46 + 8\, g(1)$$

ce qui est contradictoire avec (21). Au 4-uple (1,1,2,4) ne correspond donc aucune fonction g optimale si $\varphi(d) = d$.

6.4. BORNE SUPERIEURE DU COUT OPTIMAL - ANALOGIE AVEC LE CODAGE

HUFFMAN [7] (voir aussi GALLAGER [5], p. 50) a donné une borne inférieure et supérieure du coût optimal quand on se limite à une certaine classe d'arbres, les arbres d-aires :

Définition 5 :

(i) Nous appelons arbres d-aires, les arbres dont tous les noeuds, sauf peut-être un noeud terminal, ont pour degré d. Plus précisément, si le nombre de feuilles n vérifie :

$$(n-1) \equiv 0 \mod (d-1)$$

alors tous les noeuds ont degré d. Sinon, il y a un noeud terminal de degré $\left[1 + \text{reste}\,(n-1, d-1)\right]$

(ii) L'arbre d-aire de coût minimum pour une suite S donnée est appelé arbre d-aire optimal. Son coût est noté $C_d(S)$.

Théorème 13 (HUFFMAN) : Pour une suite S quelconque, on a :

$$\frac{\varphi(d)}{\text{Log } d} . N(S) . H(S) \leqslant C_d(S) < \frac{\varphi(d)}{\text{Log } d} . N(S) . H(S) + \varphi(d) . N(S) \tag{22}$$

En fait, HUFFMAN cherchait à minimiser la longueur moyenne du code pour un alphabet à d lettres. Dans ce cas, $\varphi(d) = 1$ et comme les a_i sont des probabilités, on a $N(S) = 1$. Le théorème de Huffman s'écrit donc :

$$\frac{H(S)}{\text{Log } d} \leqslant C_d(S) < \frac{H(S)}{\text{Log } d} + 1 \qquad \text{avec } H(S) = \sum_{1 \leqslant i \leqslant n} a_i \text{ Log}$$

HUFFMAN a proposé un algorithme permettant de calculer rapidement l'arbre d-aire optimal et le coût correspondant (voir 6.7.1).

Remarquons que la borne inférieure donnée par ce théorème a même forme que celle du corollaire 4 (formule (14)). Si nous nous limitons aux arbres d-aires, leur coût est borné inférieurement par :

$$\frac{\varphi(d)}{\text{Log } d} N(S) \; H(S) \tag{23}$$

Si nous ne faisons aucune restriction sur la forme des arbres, le coût est borné inférieurement par :

$$k \; N(S) \; H(S) \qquad \text{avec } k = \min_{d \geqslant 2} \frac{\varphi(d)}{\text{Log } d} \tag{24}$$

La borne supérieure est évidemment valable dans le cas général :

Corollaire 5 : Pour un n-uple S quelconque, on a :

$$\text{(i)} \quad C(S) < \frac{\varphi(d)}{\text{Log } d} \; N(S) \; H(S) + \varphi(d) \; N(S) \tag{25}$$

(où d est entier, $d \geqslant 2$)

$$\text{(ii)} \quad k . N(S) . H(S) \leqslant C(S) < k . N(S) . H(S) + \varphi(a) . N(S) \tag{26}$$

Preuve : La proposition (i) résulte directement du théorème de HUFFMAN, puisque $C(S) \leqslant C_d(S)$. Pour (ii), il suffit de faire $d = a$ dans les formules (14) et (25). ▨

On peut remarquer que la borne supérieure varie avec d. Généralement, la plus petite borne supérieure sera donnée pour $d = a$.
Le terme $k \; N(S) \; H(S)$ constitue-t-il une bonne approximation du coût optimal ? L'erreur relative $\mathcal{E}$ commise en approximant le coût optimal par la borne inférieure est égale à :

$$\mathcal{E} = \frac{\text{Log } a}{H(S)} \tag{27}$$

Plus S est équilibré , et plus H(S) augmente. Donc, plus les suites sont équilibrées, et plus la borne inférieure est une bonne approximation de C(S). Si tous les poids sont égaux, l'erreur relative est égale à :

$$\mathcal{E} = \frac{\text{Log } a}{\text{Log } n} \tag{28}$$

6.5. CALCUL DES ARBRES OPTIMAUX ET DE LEURS COUTS

6.5.1. ALGORITHME DE CALCUL

Soit un n-uple S donné. Pour calculer la forme et le coût de l'arbre optimal correspondant à S, on peut explorer toutes les arborescences possibles à n feuilles. Pour chaque arborescence, il faut placer les poids du n-uple S sur les feuilles dans l'ordre inverse de leurs coûts partiels, de manière à minimiser le coût (voir le corollaire 1.3). Enfin, parmi les arbres ainsi obtenus, il faut choisir celui de coût minimum. C'est cette méthode que nous allons employer, en utilisant la proposition suivante :

Proposition 3 : Pour tout n-uple, il existe un arbre optimal où les plus petits poids du n-uple sont placés sur les feuilles qui sont les filles d'un même noeud terminal.

Preuve : Considérons l'arbre optimal T pour ce n-uple et son arborescence A. Considérons un noeud terminal de A dont les filles sont les feuilles de plus grand coût partiel. Plaçons les plus petits poids du n-uple sur ces feuilles et plaçons les poids restants sur les autres feuilles de A dans l'ordre inverse des coûts partiels. L'arbre T' ainsi obtenu satisfait les conditions de la proposition. D'après le corollaire 1.3, on a : $\mathcal{C}(T') = \mathcal{C}_A(S) \leqslant \mathcal{C}(T)$. Comme T est optimal pour S, on en déduit que T' est optimal.

▨

L'algorithme de calcul du coût optimal utilise une récurrence sur n, nombre de poids du n-uple. Pour n = 1, le coût optimal est nul. Supposons que nous sachions calculer le coût optimal pour tous les 1-uple, 2-uples,... (n-1)-uples. Considérons maintenant un n-uple $(a_1,\ldots,a_n)$ avec $n > d_{max}$, où les a_i sont ordonnés par poids croissants. Soit un arbre optimal correspondant vérifiant la propriété de la proposition 3 et dont tous les degrés sont plus petits ou égaux à d_{max}. Soit d le degré du noeud terminal qui regroupe les plus petits poids du n-uple. D'après la formule des contractions, le coût de cet arbre est égal à :

$$\varphi(d).\sum_{1\leqslant i\leqslant d} a_i + C\left(\sum_{1\leqslant i\leqslant d} a_i\ ,\ a_{d+1},\ldots,a_n\right) \qquad (29)$$

avec $2\leqslant d\leqslant d_{max}$

D'après l'hypothèse de récurrence, nous savons évaluer le deuxième terme de (29), puisqu'il s'agit de calculer le coût optimal d'un (n-d+1)-uple. Le coût optimal est donc obtenu en minimisant (29) quand d varie entre 2 et d_{max} :

$$C(a_1,\ldots,a_n) = \min_{2\leqslant d\leqslant d_{max}}\left[\varphi(d).\sum_{1\leqslant i\leqslant d} a_i + C\left(\sum_{1\leqslant i\leqslant d} a_i,\ a_{d+1},\ldots,a_n\right)\right] \qquad (30)$$

C'est le procédé qu'utilise l'algorithme récursif suivant :

données : entier d_{max} ; fonction φ.
appel: le tableau TA, de dimension n, contient les n poids du n-uple triés par ordre croissant de poids.

pour $1\leqslant i\leqslant n$ $\quad TA(i) = a_i$

résultat : le résultat de la procédure est C, le coût optimal correspondant au n-uple.
Fonction utilisée : la fonction somme(d,TA) a pour résultat la somme des d premiers éléments du tableau TA. La fonction remplacer(d,TA) a pour argument un entier d et un tableau TA trié par ordre croissant de poids . Elle a pour résultat un tableau trié où les d premiers éléments de TA ont été remplacés par leur somme. Par exemple :

TA = (1,2,3,4,9) $\qquad$ remplacer (3,TA) = (4,6,9)

La procédure récursive C est la suivante :

```
réel procédure C(TA,n) ; valeur TA,n ; réel tableau TA ; entier n ;
début
    entier d,E ;
    si n = 1 alors C: = 0
    sinon
    début
        C:=∞
        pour d: = 2 pas 1 jusqua min (n,d_max) faire
        début
            E: = φ(d).somme(d,TA)
                + C(remplacer(d,TA),(n-d+1)) ;
            C: = min(E,C) ;
        fin
    fin
fin
```

Pour obtenir la forme de l'arbre optimal, il suffit de garder trace des entiers d qui minimise l'expression E, c'est-à-dire l'expression (29).

Cet algorithme est récursif. Il est évident que son temps de calcul est exponentiel en fonction de n. Il est donc inutilisable pour les grandes valeurs de n : il faut environ une heure d'ordinateur IRIS-80 pour calculer le coût optimal si n = 60 !

6.5.2. COMPLEXITE INTRINSEQUE DU PROBLEME DE RECHERCHE DE L'ARBRE OPTIMAL

Existe-t-il des algorithmes non exponentiels ? Nous verrons que dans certains cas particuliers, il est possible de calculer l'arbre optimal rapidement, c'est-à-dire en temps polynomial, ou même linéaire, en fonction de n. Mais dans le cas général ?
Jusqu'à présent, nous nous sommes posés le problème suivant :

Problème 1 : Soient une fonction φ et une suite $S = (a_1, \ldots, a_n)$, trouver l'arbre de suite S qui minimise le coût.

Considérons le problème suivant :

Problème 2 : Soient une fonction φ, une suite $S = (a_1, \ldots, a_n)$ et une constante réelle c. Existe-t-il un arbre T de suite S dont le coût vérifie $\mathcal{C}(T) \leqslant c$?

Il est évident que le problème 2 est plus simple que le problème 1, car si on connaît l'arbre optimal pour S, on peut immédiatement répondre à la deuxième question. Essayons d'analyser la complexité du deuxième problème.

Tout d'abord, le problème 2 est NP, ce qui veut dire "non déterministe polynomial" (voir COOK [3] et KARP [12]). Cela signifie que la vérification peut être faite en temps polynomial en n : c'est évident dans notre cas puisqu'il suffit de calculer le coût de T et de comparer $\mathcal{C}(T)$ et c.

Le problème 2 lui-même peut-il être résolu en temps polynomial ? Ce serait évidemment le cas s'il existait une fonction g "universelle", c'est-à-dire telle que pour toute suite S on ait $Z_g(S) = C(S)$ (voir 6.3.3).

Mais il n'existe pas de telles fonctions g et nous n'avons pu mettre en évidence un algorithme non exponentiel pour le problème 2.
Le problème 2 est-il alors intrinsèquement exponentiel ? Jusqu'à maintenant, on n'a jamais pu montrer qu'un problème NP est intrinsèquement exponentiel (voir COOK [3] et KARP [12]). Mais on a mis en évidence une vaste classe de problèmes NP, appelés problèmes NP-complets, qui sont équivalents entre eux, au sens suivant : si l'un d'eux est intrinsèquement exponentiel, alors ils le sont tous ; si l'un deux peut être résolu par un algorithme polynomial, alors tous peuvent l'être également. Nous pensons que le problème 2 est NP-complet, mais nous n'avons pu le montrer.

6.6. INDICATIONS SUR LA FORME DES ARBRES OPTIMAUX

Nous étudions d'abord la forme générale des arbres optimaux (degrés, différence de poids des fils d'un même noeud). Nous verrons ensuite que les arborescences optimales pour les n-uples à poids égaux restent optimales pour les n-uples équilibrés. Enfin, pour les n-uples très déséquilibrés, nous verrons que les arbres "linéaires" sont optimaux.

6.6.1. FORME GENERALE

Nous avons vu en 6.3.2. (formules (17), (18), (19) et (20)), que la différence entre le coût d'un arbre quelconque T et la borne inférieure $Z_G(S_T)$ s'écrit :

$$\mathcal{C}(T) - Z_G(S_T) = \sum_{v \in \mathcal{N}_T} w(v)\left[\Delta_1(v) + \Delta_2(v)\right]$$

avec :

$$\Delta_1(v) = \varphi(d(v)) - k \operatorname{Log} d(v)$$

$$\Delta_2(v) = k\left[\operatorname{Log} d(v) + \sum_{u \in \mathcal{F}(v)} \frac{w(u)}{w(v)} \operatorname{Log} \frac{w(u)}{w(v)}\right]$$

Nous avons vu que $\Delta_1(v)$ et $\Delta_2(v)$ sont positifs ou nuls.

Le terme $\Delta_1(v)$ ne s'annule que si d est un degré-limite.
Le terme $\Delta_2(v)$ ne s'annule que si tous les fils de v ont même poids.

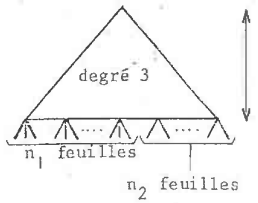
En conséquence, les degrés des arbres optimaux tendent à être les degré-limites, et les arbres optimaux tendent à être aussi équilibrés que possible. Mais cela ne signifie pas que les différences de poids des sous-arbres principaux sont bornées ! (cf. 5.4)

6.6.2. CAS DES N-UPLES EQUILIBRES

Nous allons montrer que les arbres optimaux à poids égaux restent optimaux pour les n-uples équilibrés. Nous le montrerons pour deux fonctions de coût particulières, mais ce résultat s'étend à d'autres fonctions de coût.

Proposition 4 : Si $\varphi(d) = d$ et si le rapport du plus grand au plus petit poids du n-uple $S = (a_1,\dots,a_n)$ est inférieur à 3/2, l'arbre optimal T pour S est obtenu en plaçant les poids a_i de S sur l'arborescence A dans l'ordre inverse des coûts partiels. L'arborescence A est celle de l'arbre optimal à n feuilles de même poids.

Preuve : Supposons tout d'abord que $2.3^p \leqslant n \leqslant 3^{p+1}$. D'après 2.6.3, l'arborescence A a la forme suivante : jusqu'à la profondeur p-1, tous les noeuds ont degré 3 ; à la profondeur p, il y a $(n-2.3^p)$ noeuds ternaires et $(3^{p+1}-n)$ noeuds binaires :



profondeur p arborescence A

Il y a donc :

$n_1 = 3(n-2.3^p)$ feuilles "ternaires"
$n_2 = 2(3^{p+1}-n)$ feuilles "binaires"

Nous allons supposer d'autre part, que les n_1 plus petits poids du n-uple $(a_1 \dots a_n)$ sont dans l'intervalle $[1,2]$ et les n_2 plus grands poids dans l'intervalle $[2,3]$. En effet, on peut toujours se ramener à ce cas en multipliant le n-uple par une constante, puisque le rapport du plus grand au plus petit poids du n-uple est inférieur à 3/2 (inférieur donc à 2/1 et 3/2, rapport entre les extrêmités des intervalles $[1,2]$ et $[2,3]$). D'après la proposition 1.3, la forme de l'arbre optimal ne change pas dans cette transformation. Le résultat sera donc valable pou tout n-uple dont le rapport du plus grand au plus petit poids est inférieur à 3/2.
Appelons B la somme des n_1 plus petits poids et H la somme des n_2 plus grands poids.
Calculons le coût de l'arbre T obtenu en plaçant sur A les poids dans l'ordre inverse des coûts partiels. Appelons N le poids du n-uple ; un calcul simple montre que :

$$\mathcal{C}(T) = 3.p.N + 2.H + 3.B$$

soit :

$$\mathcal{C}(T) = 3(p+1)N - H \tag{31}$$

Calculons maintenant la borne inférieure $Z_C(S)$ du corollaire 3. Les formules (2.3.5) sont évidemment valables pour les réels. Elles s'écrivent :

$$\begin{cases} \text{si} \quad 3^p \leqslant x \leqslant 2.3^p & \text{alors} \quad C(x) = (3p+4)x - 4.3^p \\ \text{si} \quad 2.3^p \leqslant x \leqslant 3^{p+1} & \text{alors} \quad C(x) = (3p+5)x - 2.3^{p+1} \end{cases} \tag{32}$$

Evaluons C(N). Comme les n_1 plus petits poids sont compris entre 1 et 2, nous avons : $n_1 \leqslant B \leqslant 2n_1$. De même : $2n_2 \leqslant H \leqslant 3n_2$. Comme N = B+H, nous en déduisons :

$$n_1 + 2n_2 \leqslant N \leqslant 2n_1 + 3n_2$$

On en déduit que :

$$3^{p+1} \leqslant N \leqslant 2.3^{p+1}$$

D'où :

$$C(N) = \Big(3(p+1)+4\Big)N - 4.3^{p+1}$$

Calculons maintenant $\sum_{1\leqslant i\leqslant n} C(a_i)$ en partitionnant les a_i entre les n_1 plus petits poids et les n_2 plus grands. Nous obtenons, d'après (32) :

$$\sum_{1\leqslant i\leqslant n} C(a_i) = \Big[4\ B - 4\ n_1\Big] + \Big[5\ H - 6\ n_2\Big]$$

On en déduit :

$$Z_C(S) = C(N) - \sum_{1\leqslant i\leqslant n} C(a_i) = 3(p+1)N - H$$

La borne inférieur $Z_C(S)$ est donc égale au coût de l'arbre T (formule (31)). L'arbre T est donc optimal.

Dans le cas où $3^p \leqslant n \leqslant 2.3^p$, une démonstration analogue prouve le résultat. ▨

Les arborescences des arbres optimaux à poids égaux restent donc optimales si les n-uples ne sont pas trop déséquilibrés. Il en est de même dans le cas suivant :

Proposition 5 : Si $\varphi(d) = d+\lambda$, avec $\lambda \in \left]-2, -1/2\right]$, et si le rapport du plus grand au plus petit poids du n-uple $S = (a_1,\ldots,a_n)$ est inférieur à 2, l'arbre optimal T pour S est obtenu en plaçant les poids a_i de S sur l'arborescence 2-équilibrée à n feuilles dans l'ordre inverse des coûts partiels.

Preuve : L'arborescence 2-équilibrée est définie en 1.14. On montre facilement que si $\varphi(d) = d+\lambda$ avec $\lambda \in \left]-2, -1/2\right]$, les arbres optimaux à poids égaux sont les arbres 2-équilibrés. Le reste de la démonstration est analogue à celle de la proposition 4. ▨

6.6.3. CAS DES N-UPLES DESEQUILIBRES

Si les n-uples sont très déséquilibrés, les arbres "linéaires" sont optimaux : un arbre linéaire à n feuilles a pour profondeur maximum n-1 et tous ses noeuds ont pour degré 2. Par exemple, l'arbre linéaire à 6 feuilles est :

Nous nous limitons au cas $\varphi(d) = d+\lambda$. Mais la méthode de démonstration s'applique à d'autres fonctions de coût.
Il nous faut d'abord établir le lemme suivant :

Lemme 3 : Si $\varphi(d) = d+\lambda$ et si l'arbre plat de degré d est optimal pour le d-uple $(a_1,\ldots,a_d)$, alors tout arbre plat de degré d' < d est optimal pour tout d'-uple extrait de $(a_1,\ldots,a_d)$.

Preuve : Appelons T' l'arbre plat de degré d' et appelons b' son poids. Appelons l'arbre de degré d et N son poids. Posons $d_1 = d-d'$. On vérifie que :

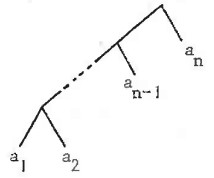
$$\mathcal{C}(T) = \Big[\mathcal{C}(T')+d'(N-b')\Big] + \Big[(d_1+\lambda)(N-b')+d_1 b'\Big] \tag{33}$$

Si T' n'était pas optimal, il existerait un arbre T" de coût inférieur à $\mathcal{C}(T')$ et de degré d" < d'. Remplaçons dans T l'arbre T' par l'arbre T". On obtient ainsi un arbre de coût inférieur à $\mathcal{C}(T)$: en effet, dans la formule (33), le premier terme entre-crochets diminue, le deuxième reste invariant. Cela contredit l'hypothèse d'optimalité de T. ▨

Proposition 6 : Si $\varphi(d) = d+\lambda$ et si le n-uple $S = (a_1,\ldots,a_n)$ vérifie :

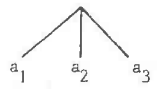
(34) pour $3 \leqslant i \leqslant n$ $\quad a_i > (1+\lambda) \sum_{1 \leqslant j < i} a_j$

alors l'arbre linéaire suivant est optimal :



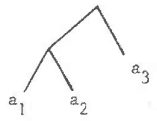
Preuve : D'après la proposition 3, il existe un arbre optimal où les plus petits poids du n-uple sont les fils d'un même noeud terminal. Appelons d son degré. Montrons d'abord que d est différent de 3. Si d était égal à 3, l'arbre T_2 suivant serait optimal :

T_2 =



or, ce n'est pas le cas : on peut vérifier à l'aide de (34) que son coût est strictement supérieur au coût de l'arbre T_1 suivant :

T_1 =



Montrons que d = 2. Si d était supérieur ou égal à 4, on en déduirait d'après le lemme précédent, que T_3 est optimal, ce qui n'est pas le cas. Comme d = 2, nous sommes donc amenés, comme pour l'algorithme décrit en 6.5.1, à chercher l'arbre optimal pour le (n-1)-uple $(a_1+a_2, a_3, \ldots, a_d)$. Comme ce (n-1)-uple satisfait les hypothèses (34), l'arbre linéaire est bien optimal.

6.7. HEURISTIQUES

L'algorithme proposé en 6.5.1 est inutilisable en général, puisqu'il prend un temps exponentiel en n pour le calcul de l'arbre optimal correspondant à un n-uple donné. Les deux heuristiques suivantes donnent des moyens pratiques et rapides de calculer des arbres approximativement optimaux.

6.7.1. ALGORITHME DE HUFFMAN

Cet algorithme dû à HUFFMAN [7] calcule le meilleur arbre d-aire, c'est-à-dire l'arbre dont tous les noeuds ont degré d, sauf peut-être un noeud terminal (définition 5). On procède comme dans l'algorithme général exposé en 6.5.1, sauf qu'à chaque fois on regroupe toujours les d plus petits poids sur les feuilles d'un même noeud terminal. L'algorithme est le suivant :

données : entier d ; fonction φ.
appel : le tableau TA, de dimension n, contient les n poids du n-uple triés par ordre croissant de poids :
résultat : H, coût de l'arbre d-aire optimal.
fonctions utilisées : les fonctions somme et remplacer sont décrites en 6.5.1.

```
r: = reste (n-1,d-1)
si r = 0 alors H: = 0
sinon
début
        H: = φ(r+1).somme(r+1,TA) ;
        TA: = remplacer(r+1,TA) ;
        n: = n-r ;
fin ;
tant que n > 1 faire
début
        H: = H + φ(d).somme(d,TA) ;
        TA: = remplacer(d,TA) ;
        n: = n-d+1 ;
fin
```

Cet algorithme n'est pas récursif. Il utilise un espace mémoire de l'ordre de O(n). Pour ce qui est du temps, la boucle tant que est parcourue environ $\left((n-1)/(d-1)\right)$ fois. Dans cette boucle, seule la fonction remplacer prend un temps significatif : elle effectue la somme des d premiers éléments et insère cette somme dans la suite déjà triée des éléments restants ; cette fonction effectue donc de l'ordre de Log n opérations. La complexité en temps de l'algorithme est donc O(n Log n).

Dans la suite nous appelons $H_d(S)$ le coût de l'arbre calculé par cet algorithme. HUFFMAN a montré que $H_d(S)$ satisfait les bornes inférieures et supérieures du théorème 13 :

Proposition 7 (HUFFMAN) : Pour un n-uple $S = (a_1,\ldots,a_n)$, on a les propriétés :

(i) Si $n-1 \equiv 0 \mod.(d-1)$, l'algorithme précédent calcule l'arbre d-aire de coût minimum pour S :

$$n-1 \equiv 0 \mod.(d-1) \quad \Longrightarrow \quad H_d(S) = C_d(S)$$

(ii) Pour d entier, $d \geqslant 2$,

$$H_d(S) < \frac{\varphi(d)}{\text{Log } d}\ N(S)\ H(S) + \varphi(d)\ N(S) \tag{35}$$

Preuve de (i) : Une démonstration analogue à celle de la proposition 3 montre que, pour tout n-uple, il existe un arbre d-aire optimal où les plus petits poids du n-uple sont regroupés sur un même noeud terminal. D'autre part, si $n-1 \equiv 0 \mod.(d-1)$, tous les noeuds de cet arbre ont pour degré d. Le noeud terminal précédent regroupe donc les d plus petits poids et cet arbre est calculé par l'algorithme de HUFFMAN. Son coût $C_d(S))$ est donc égal à $H_d(S)$.

Preuve de (ii) : Si $n-1 \equiv 0 \mod (d-1)$, c'est évident d'après le théorème 13 puisque $H_d(S) = C_d(S)$. Sinon, complétons S par des poids nuls de manière à obtenir un n'-uple S' tel que :

$$n'-1 \equiv 0 \mod.(d-1)$$

L'inégalité (35) est valable pour S'. Comme :

$$H_d(S) \leqslant H_d(S') \qquad N(S) = N(S') \qquad H(S) = H(S')$$

l'inégalité (35) est donc valable pour S.

L'algorithme de HUFFMAN nous donne une heuristique dans le cas général : il suffit de choisir l'arbre minimisant $H_d(S)$ quand $2 \leqslant d \leqslant d_{max}$. Généralement, les bons degrés d sont ceux qui minimisent $\varphi(d)/\text{Log } d$, c'est-à-dire les degré-limites. En effet, le premier terme de la borne supérieure (35) est généralement d'un ordre de magnitude plus grand que le second.

6.7.2. HEURISTIQUE DES N-UPLES ORDONNES

Pour chercher l'arbre optimal pour un n-uple S, la méthode de l'algorithme général consiste à examiner toutes les arborescences et à placer les poids de S sur ces arborescences dans l'ordre inverse des coûts partiels. L'heuristique que nous présentons ici consiste à faire l'inverse : nous allons fixer l'ordre des poids des feuilles $a_1,\ldots,a_n$ et chercher l'arbre de coût minimum tel que les feuilles de poids respectifs $a_1,\ldots,a_n$ soient placées sur l'arbre de la gauche vers la droite. Nous appelerons cet arbre l'arbre optimal ordonné pour le n-uple ordonné $a_1,\ldots,a_n$.

L'algorithme utilise le tableau A à trois dimensions suivant :

A(d,i,j) est le coût minimum de d arbres juxtaposés dont les feuilles ont pour poids, de gauche à droite :

$$a_i,\ a_{i+1},\ldots,a_j \tag{36}$$

En particulier, A(1,i,j) est le coût de l'arbre optimal ordonné dont les feuilles ont leurs poids dans l'ordre (36). D'après cette définition, il est clair que le tableau A vérifie :

$$\text{pour } d \geqslant 2 \qquad A(d,i,j) = \min_{i \leqslant k \leqslant j-d+1} \left[A(1,i,k) + A(d-1,k+1,j) \right]$$

$$\text{et :} \quad \begin{cases} A(1,i,j) = \min_{2 \leqslant d \leqslant D} \left[\varphi(d) \left(\sum_{i \leqslant k \leqslant j} a_k \right) + A(d,i,j) \right] \\ \text{avec } D = \text{Min}(d_{max}, j-i+1) \end{cases}$$

On en déduit l'algorithme suivant, où l'on a posé $\ell = j-i$:

données : entier d_{max} ; fonction φ.
appel : le tableau TA, de dimension n, contient les n poids du n-uple ordonné selon l'ordre choisi.
résultat : A(1,1,n) , coût de l'arbre optimal ordonné
fonction utilisée : la fonction sigma(TA,i,j) avec i < j, a pour résultat la somme TA(i) + ... + TA(j)

```
tableau A(d_max,n,n) ;
pour i: = 1 pas 1 jusqua n faire A(1,i,i): = 0 ;
pour ℓ: = 2 pas 1 jusqua n-1 faire
début
    pour i: = 1 pas 1 jusqua n-ℓ faire
    début
        D: = min (d_max,ℓ) ;
        pour d: = 2 pas 1 jusqua D faire

        A(d,i,i+ℓ): =   Min   [A(1,i,k) + A(d-1,k+1,i+ℓ)] ;
                    i≤k≤i+ℓ-d+1

        A(1,i,i+ℓ): =  Min   [φ(d).sigma(TA,i,j) + A(d,i,i+ℓ)]
                     2≤d≤D

    fin
fin
```

On vérifie facilement que la complexité de cet algorithme est $O(n^3)$ en temps et $O(n^2)$ en espace mémoire, puisque le tableau A est de dimension (d_{max}, n^2).
Le coût de l'arbre calculé par cet algorithme peut être éloigné du coût optimal : en particulier rien n'assure que son coût soit plus petit que la borne supérieure du corollaire 5. Cependant, on constate que si les poids TA(i) sont ordonnés par ordre croissant, cette heuristique donne généralement de bons résultats.
L'intérêt principal de cette heuristique est qu'elle peut être combinée avec l'algorithme de HUFFMAN.

Soit en effet, un n-uple $S = (a_1, \ldots, a_n)$. Avec l'algorithme de HUFFMAN, nous obtenons un arbre pour la suite S dont le coût est plus petit que la borne supérieure du théorème 13. Appelons $a_{\tau(1)}, a_{\tau(2)}, \ldots, a_{\tau(n)}$ les poids des feuilles de cet arbre ordonnées de la gauche vers la droite (τ est une permutation de $1,2,\ldots,n$). Appliquons maintenant au n-uple ordonné $\left(a_{\tau(1)}, a_{\tau(2)}, \ldots, a_{\tau(n)}\right)$ la deuxième heuristique. Nous obtenons ainsi un nouvel arbre pour la suite S dont le coût est inférieur ou égal au précédent, puisque c'est le meilleur arbre pour ce n-uple ordonné. Ce procédé donne une bonne approximation de l'arbre optimal.

CHAPITRE 7

MODELISATION DU TRI-FUSION

Le but de cette étude est de proposer des stratégies efficaces pour fusionner un certain nombre de monotonies de grandes tailles quand l'utilisateur dispose de périphériques à accès direct, disques et tambours en particulier. Rappelons qu'une monotonie est une suite d'enregistrements triés suivant une certaine clé.
Nous proposons des stratégies optimales et sous-optimales et évaluons leurs performances pour les diverses configurations matérielles couramment utilisées. Nous définissons également une vitesse de tri-fusion, ne dépenda que des caractéristiques matérielles, qui constitue une bonne mesure des performances d'une configuration du point de vue du tri-fusion. Cette vitesse de tri-fusion permet de comparer les performances de diverses configurations et peut aider dans le choix d'une configuration et d'une méthode de tri-fusion adaptée au tri sur disque.

7.1. PRESENTATION DU TRI-FUSION SUR DISQUE

7.1.1. CADRE DE L'ETUDE

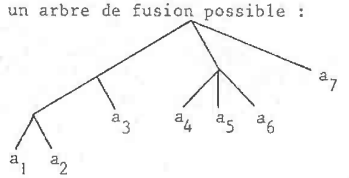
Tout d'abord, nous supposons que les monotonies à fusionner sont initialement placées sur disque et que le fichier trié résultant doit être également placé sur disque. Dans le cas où les périphériques de départ et d'arrivée ne sont pas des périphériques à accès direct, les stratégies optimales de fusion se déduisent facilement des stratégies optimales proposées.
D'autre part, nous supposons que la place M disponible en mémoire centrale de l'ordinateur reste fixe tout au long du processus de tri-fusion. C'est évidemment le cas si l'utilisateur est seul à travailler sur l'ordinateur. C'est également le cas dans certains systèmes à partage de ressources où chaque utilisateur dispose d'une partition fixe de la mémoire centrale.

Notons que dans ce cas, l'optimisation du temps de fusion n'a de sens que du point de vue du système, puisque le processus de tri-fusion sera interrompu par les autres utilisateurs partageant l'ordinateur.
Nous supposons également que le processus de tri-fusion s'exécute séquentiellement, c'est-à-dire que les lectures et les écritures sur disques ne s'effectuent pas en parallèle. C'est le cas si l'utilisateur dispose d'un seul disque. S'il dispose de plusieurs disques, il est généralement possible de faire des entrées/sorties en parallèle. Cette méthode permet d'améliorer sensiblement les temps de fusion. Mais il est difficile de proposer des stratégies de tri-fusion qui soient efficaces dans tous les cas, car les temps de fusion dépendent alors de l'ordre respectif des enregistrements des différentes monotonies.
Enfin, nous supposons que les tailles des monotonies à fusionner sont grandes par rapport à la place disponible en mémoire centrale. Si ce n'est pas le cas, il est inutile d'utiliser un périphérique pour effectuer la fusion ! Cependant, il faut considérer le cas où les tailles des monotonies initiales sont plus petites que la place disponible en mémoire centrale, alors que la taille du fichier trié résultant est beaucoup plus grande.

7.1.2. METHODE DE TRI-FUSION - OPTIMISATION

Description du processus de tri-fusion

Nous avons vu dans l'introduction que le processus de tri-fusion peut être représenté par un arbre de fusion. Soient par exemple 7 monotonies $m_1,\ldots,m_7$, de tailles respectives $a_1,\ldots,a_7$ à fusionner. Le schéma suivant représente un arbre de fusion possible :

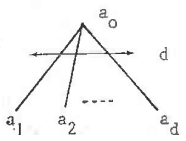


A chaque feuille de poids a_i correspond la monotonie initiale m_i de taille a_i

L'arbre précédent décrit le processus de tri-fusion suivant : effectuer la fusion élémentaire de m_1 et m_2 ; on obtient ainsi une nouvelle monotonie de taille $a_1 + a_2$; effectuer ensuite la fusion élémentaire de cette monotonie avec m_3 ; et ainsi de suite, jusqu'à obtenir le fichier trié résultant. Les fusions élémentaires étant effectuées successivement, le temps total de fusion est égal à la somme des temps pris par chacune des fusions élémentaires.

Temps de fusion élémentaire

A chaque noeud de l'arbre correspond une fusion élémentaire et donc une monotonie résultante dont la longueur est la somme des poids des feuilles du sous-arbre correspondant à ce noeud. Une fusion élémentaire peut être représentée de la manière suivante :



$$a_o = \sum_{1 \leq i \leq d} a_i$$

Les valeurs $a_1,\ldots,a_d$ sont les tailles respectives des monotonies "entrantes" $m_1,\ldots,m_d$ à fusionner et a_o est la taille de la monotonie résultante m_o. L'entier d est le degré ou "nombre de voies" de la fusion élémentaire.
Nous avons décrit en détail dans l'introduction le processus de fusion élémentaire. Si toutes les entrées/sorties s'effectuent séquentiellement, le temps F pris par la fusion élémentaire est égal à :

$$(1) \qquad F = 2\,Tt\,a_o + \sum_{0 \leq i \leq d} Ta_i \left\lceil a_i/b_i \right\rceil + Tm$$

Tt est le temps de transfert par unité de longueur ; Ta_i est le temps moyen d'accès à la monotonie m_i ; Tm est le temps de traitement en mémoire centrale. A chaque monotonie m_i correspond en mémoire centrale un tampon de taille b_i. Bien entendu, nous avons intérêt à utiliser toute la place M disponible en mémoire centrale, si bien que :

$$(2) \qquad M = \sum_{0 \leq i \leq d} b_i$$

Recherche des stratégies optimales de fusion

Pour une configuration donnée, cette recherche se décompose en deux étapes:

1) L'optimisation du temps de fusion élémentaire : il s'agit d'abord d'évaluer Tt, Ta_i et Tm en fonction de l'ordinateur utilisé, des caractéristiques matérielles des disques, et de la méthode de fusion élémentaire choisie (voir paragraphes suivants). Une fois ces paramètres connus, il reste à partitionner la place M disponible en mémoire centrale en (d+1) tampons $b_o,b_1,\ldots,b_d$ de manière à minimiser F. Nous obtenons ainsi un temps F de fusion élémentaire qui est fonction des tailles $a_1,\ldots,a_d$ des monotonies entrantes.

2) La recherche des arbres de fusion optimaux représentant les stratégies optimales de fusion. Le temps de fusion élémentaire calculé ci-dessus définit une fonction de coût sur les arbres de fusion qui est la somme des temps de fusions élémentaires représentées par chacun des noeuds de l'arbre. Etant donné n monotonies initiales de tailles $a_1,\ldots,a_n$, il s'agit de trouver l'arbre de fusion à n feuilles de poids $a_1,\ldots,a_n$ qui minimise ce coût ; cet arbre est appelé arbre optimal pour le n-uple $S=(a_1,\ldots,a_n)$.
Dans la suite de ce chapitre, nous étudions les divers paramètres qui interviennent dans le temps de fusion élémentaire, ainsi que les diverses méthodes de fusion élémentaires possibles.

7.2. CARACTERISTIQUES DES PERIPHERIQUES A ACCES DIRECT

Nous décrivons d'abord le fonctionnement et les caractéristiques des périphériques usuels à accès direct. Nous évaluons ensuite les temps moyens d'accès et de transfert quand l'utilisateur a un contrôle minimum sur les entrées/sorties, en particulier quand il ne peut choisir l'emplacement des monotonies résultantes sur le périphérique. Nous indiquons enfin comment améliorer les temps d'accès si l'utilisateur a un contrôle plus grand sur les entrées/sorties.
Dans cette étude, nous nous limiterons à deux types de périphériques à accès direct : les disques à têtes mobiles d'une part, les disques à têtes fixes et les tambours d'autre part. Mais les modélisations que nous proposons sont valables pour d'autres périphériques à accès direct, les cartes magnétiques par exemple.

7.2.1. DISQUES A TETES MOBILES

Ces disques sont généralement constitués de une ou plusieurs surfaces circulaires empilées en rotation très rapide . Chaque surface est divisée en un certain nombre de pistes circulaires sur lesquelles se trouvent les données. Un bras mobile, comprenant une ou plusieurs têtes de lecture/écriture, sert à lire ou écrire les données sur ces pistes, de telle sorte que toute l'information stockée sur une piste peut être lue en une révolution du disque ; de même pour l'écriture. Le bras mobile se déplace radialement, de piste en piste, mais ce déplacement prend du temps. Un ensemble de pistes qui peut être lu ou sur lequel on peut écrire sans déplacer le bras est appelé un cylindre . La figure suivante présente un disque possédant une seule tête de lecture/écriture par surface :

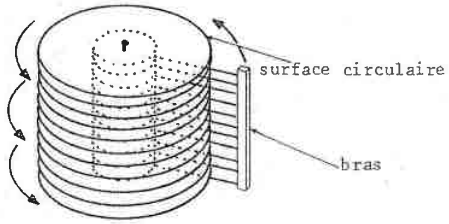


FIGURE 1

Disque à bras mobile

Les pointillés indiquent le cylindre correspondant à la position actuelle du bras.

1) Evaluation du temps d'accès moyen Ta : le temps d'accès est la somme du temps de positionnement du bras sur le cylindre plus le délai rotationnel d'attente pour que la tête de lecture/écriture atteigne le point désiré de la piste.

En moyenne, le délai rotationnel est égal à la moitié du temps d'une révolution du disque. Nous l'appelons γ. L'évaluation du temps de positionnement du bras est plus complexe : le temps de déplacement du bras d'un cylindre à un autre augmente avec la distance qui les sépare, mais il n'y a pas proportionnalité,comme nous le verrons au chapitre 9.

Si l'utilisateur n'a aucun contrôle sur l'emplacement de ses monotonies sur disque, il faut supposer qu'elles sont réparties de façon aléatoire. Le temps moyen Tb de positionnement du bras est alors le temps moyen de déplacement entre deux points aléatoires du disque. Le temps Tb est généralement fourni par le constructeur.

Finalement, le temps d'accès moyen Ta est égal à :

$$Ta = Tb + \gamma \qquad (3)$$

2) Evaluation du temps de transfert moyen Tt

A l'intérieur d'un cylindre, le bras n'a pas à se déplacer. Le temps de transfert Ct à l'intérieur d'un cylindre est donc égal à la durée d'une révolution divisée par la capacité d'une piste. Ce temps est généralement fourni par le constructeur. Si les données à transférer chevauchent deux cylindres, il faut ajouter au temps de transfert proprement dit le temps θ de déplacement du bras d'un cylindre au voisin. Pour le transfert d'une séquence de taille N, la somme des temps de déplacement du bras est en moyenne $N.\theta/\rho$, où ρ est la capacité d'un cylindre. Le temps de transfert moyen Tt est donc égal à :

$$Tt = Ct + \theta/\rho \qquad (4)$$

7.2.2 - DISQUES A TETES FIXES - TAMBOURS

Les disques à têtes fixes sont identiques aux disques à têtes mobiles, sauf qu'il y a une tête de lecture/écriture par piste. C'est également le cas pour les tambours, bien que leur fonctionnement soit différent. Dans les deux cas, il n'y a donc pas de temps de positionnement du bras. On a donc :

$$Ta = \gamma \qquad Tt = Ct$$

7.2.3. CARACTERISTIQUES DE QUELQUES DISQUES ET TAMBOURS

Le tableau suivant présente les caractéristiques d'un tambour et de quelques disques. Ces caractéristiques proviennent des manuels CII et IBM. Le premier périphérique est un tambour et les deux suivants sont des disques DIAD à têtes fixes. Les trois derniers sont des disques à têtes mobiles : le MD 100, le MD 50 et le DIMAS. Les unités utilisées sont la milliseconde et le K-octet (= 1024 octets).

	capacité	Ta	Tt	Vt	γ	TM	Tb	Ct	ρ	θ
TAMBOUR IBM 2301	4000	8.65	0.81	1.23	8.65	3450				
DIAD-G grande vitesse CII	5248	20.03	0.49	2.05	20.03	1497				
DIAD-M moyenne vitesse CII	2880	20.03	7.12	0.14	20.03	1497				
MD-100 CII	84436	38.33	1.56	0.64	8.33	3600	30	1.51	209	10
MD-50 CII	48000	47.50	4.22	0.24	12.50	2400	35	1.17	120	6
DIMAS CII	24000	92.50	4.37	0.23	12.50	2400	80	4.17	120	24

TABLEAU 1

Caractéristiques de quelques disques et tambours
exprimées en millisecondes et en K-octets

Les colonnes indiquent dans l'ordre :

la capacité du disque
Ta : le temps d'accès moyen
Tt : le temps de transfert moyen
Vt : la vitesse de transfert moyenne (Vt = 1/Tt)
γ : le délai rotationnel moyen ($\gamma = \frac{1}{2}$ 60000/TM)
TM : le nombre de tours/minute

Pour les disques à têtes mobiles, nous indiquons également :
Tb : le temps moyen de positionnement du bras
Ct : le temps de transfert à l'intérieur d'un cylindre.
ρ : la capacité d'un cylindre
θ : le temps de déplacement du bras d'un cylindre au voisin

Dans les chapitres suivants, nous étudions les performances de ces disques du point de vue du tri-fusion, en fonction des diverses configurations possibles.

7.3. METHODES D'AMELIORATION DES TEMPS D'ACCES

Le tableau 1 indique les temps d'accès et de transfert moyens quand l'utilisateur a un contrôle minimum sur les entrées/sorties. Naturellement plus ce contrôle est grand, et plus ces temps peuvent être améliorés.
En ce qui concerne le temps de transfert, la seule amélioration possible serait d'éviter les déplacements de bras d'un cylindre au voisin au cours d'une même entrée/sortie. En effet, les tailles b_i des tampons en mémoire centrale sont toujours plus petites que la capacité d'un cylindre. Si l'utilisateur peut diviser les monotonies m_i correspondantes en séquences de taille b_i de manière à ce que chaque séquence soit contenue dans un cylindre, le temps de transfert est alors égal à Ct au lieu de Tt. Mais, d'une part, cela est difficile à réaliser, et, d'autre part, la diminution des temps de fusion qui en résulte est très faible, même pour le DIMAS. Il n'est donc pas possible en général d'améliorer les temps de transfert. Par contre, les procédés suivants améliorent beaucoup les temps d'accès, et, par là, les temps de fusion.

Amélioration du temps de positionnement du bras

Supposons que la taille du fichier résultant soit nettement plus faible que la capacité du disque. Même si l'utilisateur ne peut choisir l'emplacement de ses monotonies sur disque, il est parfois possible de n'utiliser qu'une partie du disque, c'est-à-dire un certain nombre de cylindres contigus, au cours du processus de tri-fusion. Ce procédé permet de diminuer très sensiblement les temps de fusion, comme nous le montrons en 8.3.4.

Optimisation du mouvement de bras

Supposons que l'utilisateur dispose de disques à bras mobile et qu'il a la possibilité de choisir l'emplacement de ses monotonies sur disque. Dans ce cas, il a évidemment intérêt à rapprocher sur le disque les monotonies entrantes et la monotonie résultante, de manière à minimiser les temps de déplacement du bras au cours des fusions élémentaires.

Plaçons nous dans le cas d'un seul disque. Si les monotonies entrantes et la monotonie résultante sont placées de manière contiguë sur le disque, la longueur totale de l'espace disque nécessaire pour la fusion élémentaire est égale à $2a_o$, si a_o est la longueur de la monotonie résultante. Si la longueur totale $2a_o$ est comprise entre $(q-1)\rho$ et $q\rho$, où ρ est la capacité d'un cylindre, le bras ne se déplacera au maximum que de $(q+1)$ cylindres contigus au cours de la fusion élémentaire, au lieu de se déplacer sur tout le disque. Le chapitre 9 est consacré à l'étude de cette optimisation du mouvement du bras.

Amélioration du délai rotationnel

Cette amélioration a été suggérée par KNUTH [13] . Elle consiste à commencer la lecture ou l'écriture d'une séquence le plus tôt possible, sans attendre que le début de la séquence vienne se placer sous la tête de lecture/écriture.

Par exemple,la lecture d'une séquence occupant toute une piste commencera à partir de la position de la tête de lecture sur la piste au moment où le bras est positionné ; on remplit ainsi la fin du tampon correspondant en mémoire centrale, puis le début du tampon.

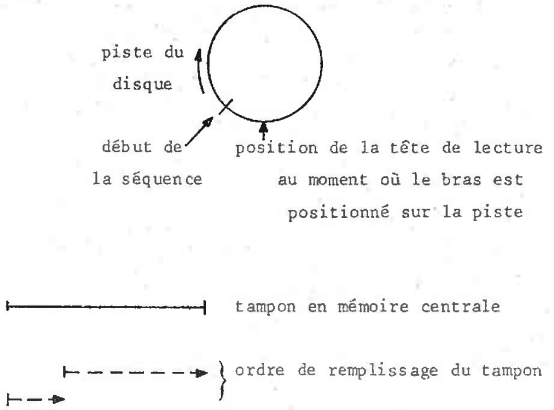


FIGURE 2

Cette méthode permet d'éliminer presque totalement le délai rotationnel si les longueurs des séquences sont des multiples de la capacité d'une piste. Cependant, elle est légèrement utopique, car, à notre connaissance, avec les disques actuels, les entrées/sorties ne peuvent commencer qu'à partir de l'adresse disque indiquée dans l'ordre d'entrée/sortie. Nous indiquons néanmoins en 8.3.4 les améliorations des temps de fusion que pourrait apporter ce procédé.

7.4. LES DEUX PRINCIPALES METHODES DE TRI-FUSION

Aux paragraphes précédents, nous avons vu qu'il existe de nombreuses méthodes de fusion, selon les caractéristiques des disques, selon le contrôle de l'utilisateur sur les entrées/sorties. A chacune d'entre elles correspondent des temps de fusion élémentaire différents, et donc des stratégies optimales de tri-fusion différentes. Du point de vue de la modélisation, on peut classer ces méthodes en deux catégories principales : les stratégies à temps d'accès constant, les stratégies à temps d'accès optimisé.

7.4.1. TRI-FUSION A TEMPS D'ACCES CONSTANT

Ce modèle, qui sera étudié au chapitre 8, s'applique dans le cas où l'utilisateur n'a aucun moyen d'influer sur les temps d'accès. Le temps d'accès Ta est alors une moyenne. Si nous disposons de tambours ou de disques à têtes fixes, le temps d'accès moyen Ta est égal au délai rotationnel moyen γ. Pour des disques à têtes mobiles, ce modèle s'applique si l'utilisateur n'a aucun contrôle sur l'emplacement des monotonies sur disque et si elles sont réparties de façon aléatoire sur le disque. Le temps d'accès moyen Ta est alors le temps d'accès aléatoire calculé en 7.2.1.
Naturellement, le temps d'accès moyen Ta peut être réduit si l'on utilise les procédés du paragraphe précédent qui améliorent le temps de positionnement du bras ou le délai rotationnel. Mais cela ne change pas la modélisation. Par ailleurs, on peut remarquer que le nombre de disques importe peu dans ce modèle puisque les temps d'accès et de transfert restent les mêmes quel que soit le nombre de disques.

7.4.2. TRI-FUSION AVEC OPTIMISATION DU MOUVEMENT DE BRAS

Nous avons vu en 7.3 que si l'utilisateur dispose de disques à têtes mobiles et contrôle l'emplacement des monotonies sur disque, il a intérêt à "rapprocher" les monotonies de manière à minimiser les temps de déplacement du bras. Au chapitre 9, nous présentons des algorithmes optimisant les mouvements du bras pour le tri-fusion si l'utilisateur dispose de un ou deux disques. La modélisation dans ce cas est entièrement différente de la précédente. En particulier, les stratégies optimales et les temps de fusion dépendent du nombre de disques utilisés.

7.5. EVALUATION DU TEMPS DE TRAITEMENT EN MEMOIRE CENTRALE

Nous avons vu que le temps de fusion élémentaire est égal à :

$$F = 2\,Tt\,a_o + \sum_{o \leq i \leq d} Ta_i \lceil a_i / b_i \rceil + Tm$$

Jusqu'ici, nous avons étudié uniquement les paramètres qui interviennent dans les temps d'entrée/sortie. Bien que le temps Tm de traitement en mémoire centrale soit généralement beaucoup plus petit, il est tout de même nécessaire de l'évaluer, ne serait-ce que pour savoir dans quel cas il est négligeable. Le temps Tm comporte d'une part le temps de tri-interclassement des d monotonies à fusionner, et d'autre part le temps d'activation des entrées/sorties.

1) Temps de tri en mémoire centrale

Appelons E le nombre d'enregistrements à interclasser et L leur longueur. Si a_o désigne la longueur de la monotonie résultante, on a donc : $a_o = L \times E$. Le travail de l'algorithme de tri consiste à sélectionner l'enregistrement à transférer dans le tampon de sortie, puis à effectuer le transfert.

Le temps total de transfert dépend uniquement de la longueur de la monotonie résultante. Il est égal à :

$$Mt.a_o$$

où Mt est le temps de transfert à l'intérieur de la mémoire centrale de l'ordinateur.

Le temps de tri proprement dit dépend uniquement du nombre E d'enregistrements et du temps pris par la comparaison de deux clés. Il est égal à :

$$(c_1 + c_2 \log_2 d).E$$

d est le nombre de monotonies à fusionner ; c_1 et c_2 sont des constantes dépendant des performances de l'ordinateur utilisé. Le terme en $\log_2 d$ vient de ce que les algorithmes de tri-interclassement utilisent un arbre de sélection de profondeur moyenne $\log_2 d$.

2) Temps d'activation des entrées/sorties

Pour chaque entrée/sortie, il y a un délai Ma entre le moment où le programme donne l'ordre d'entrée/sortie et le moment où l'ordre est reçu par le périphérique. Si l'utilisateur programme lui-même les entrées/sorties, ce délai est négligeable.

Mais si l'utilisateur passe par l'intermédiaire du système d'exploitation le délai Ma peut atteindre une milliseconde. Dans ce cas, le temps total d'activation au cours de la fusion élémentaire est égal au nombre total d'entrées/sorties multiplié par Ma :

$$Ma \cdot \sum_{o \leq i \leq d} a_i/b_i$$

Le temps total de traitement en mémoire centrale est donc égal à :

$$Tm = a_o \left[Mt + (c_1 + c_2 \, Log_2 d)/L \right] + Ma \cdot \sum_{o \leq i \leq d} \lceil a_i/b_i \rceil$$

le temps de fusion élémentaire est donc égal à :

$$F = a_o \left[2\,Tt + Mt + (c_1 + c_2 \, Log_2 d)/L \right] + \sum_{o \leq i \leq d} (Ta_i + Ma) \lceil a_i/b$$

Quel est l'ordre de grandeur du temps de traitement Tm en mémoire centrale par rapport au temps d'entrée/sortie ? Il dépend évidemment des performances de l'ordinateur et des périphériques. Considérons le cas suivant : sur un ordinateur CII IRIS-80, si l'utilisateur passe par le système d'exploitation pour ses entrées/sortie, et si les clés sont des nombres à comparer, les constantes précédentes et le temps de traitement en mémoire centrale sont égaux à :

(5)
$$Mt = 0.33 \qquad c_1 = 30.10^{-3} \qquad c_2 = 10.10^{-3} \qquad Ma = 0.5$$

$$Tm = a_o \left[0.33 + (30 + 10 \, Log_2 d) \, \frac{10^{-3}}{L} \right] + 0.5 \sum_{o \leq i \leq d} \lceil a_i/b_i \rceil$$

Rappelons que les unités sont toujours la milliseconde et le K-octet. Remarquons tout d'abord que le temps d'activation des entrées/sorties est négligeable devant le temps d'accès au périphérique, sauf peut-être si l'on utilise un tambour : le délai Ma s'ajoute en effet au temps d'accès Ta_i qui est de l'ordre de 20 à 100 millisecondes pour des disques. Considérons maintenant le temps de tri en mémoire centrale : on remarque

tout de suite qu'il est loin d'être négligeable pour les périphériques à grande cadence de transfert (tambours, DIAD grande vitesse, MD-100). En effet, le temps de transfert de ces périphériques (de 0.49 à 1.56 millisecondes par K-octet) est du même ordre de grandeur que le temps de transfert Mt en mémoire centrale (0.33 ms/K-octet). Si l'on utilise un périphérique à cadence de transfert moins rapide (MD-50, DIMAS ou DIAD moyenne vitesse), le temps de tri en mémoire centrale n'est négligeable que si la longueur L des enregistrements est grande , de l'ordre de un K-octet. En effet, le nombre d de voies de fusion dépassant rarement la centaine, le temps de tri interne est alors de l'ordre de 0.4 a_o et peut être négligé devant le temps total de transfert 2 Tt a_o, qui varie de 8.4 a_o pour le MD-50 à 14.2 a_o pour le DIAD moyenne vitesse.

En conclusion, sur un IRIS-80, le temps de traitement en mémoire centrale n'est généralement pas négligeable devant le temps d'entrée/sortie. Notons que même si l'on dispose d'un ordinateur plus rapide, il y a toujours des cas où le temps de traitement en mémoire centrale n'est pas négligeable : périphérique à grande cadence de transfert ; enregistrements de petites tailles ; grand nombre de voies de fusion ; temps de comparaison de deux clés important (rappelons que dans l'exemple ci-dessus les clés à comparer étaient des nombres; s'il s'agit de trier des chaînes de caractère par ordre alphabétique, le temps de comparaison de deux clés est beaucoup plus long et les constantes c_1 et c_2 beaucoup plus grandes).

Si le temps de traitement en mémoire centrale n'est pas négligeable, nous poserons :

(6)
$$F = 2.T't \, a_o + \sum_{o \leq i \leq d} T'a_i \, \lceil a_i/b_i \rceil + c.a_o . \, Log \, d$$

avec :

(7)
$$\begin{cases} T't = Tt + \frac{1}{2} (Mt + c_1/L) \\ T'a_i = Ta_i + Ma \\ c = c_2/(L \times Log\ 2) \end{cases}$$

Nous montrerons en 8.4. que la valeur de c n'a pratiquement aucune influence sur les stratégies optimales : seuls les temps de fusion varient avec c. Dans la suite, nous nous plaçons donc dans le cas où le temps de traitement de l'ordinateur est négligeable. Les résultats sur la forme des arbres optimaux restent valables si ce temps n'est pas négligeable.

CHAPITRE 8

TRI-FUSION A TEMPS D'ACCES CONSTANT

Dans ce chapitre, nous supposons que le temps d'accès Ta au périphérique est constant. Nous proposons diverses stratégies de tri-fusion selon les caractéristiques du matériel et selon le contrôle de l'utilisateur sur les entrées/sorties. Nous étudions les propriétés des stratégies optimales et les temps de fusion correspondants. Nous définissons une mesure de performance pour une configuration donnée, ce qui nous permet de comparer l'efficacité de diverses configurations ainsi que l'influence des divers paramètres sur les temps de fusion (temps d'accès et de transfert, place disponible en mémoire centrale, temps de traitement de l'ordinateur).

Rappelons que cette modélisation est valable si l'utilisateur dispose de disques à têtes fixes ou de tambours, ou s'il dispose de disques à têtes mobiles sans possibilité de choisir l'emplacement des monotonies sur disque. Dans tous les cas, le temps d'accès Ta est une moyenne ; la validité de la modélisation dépend donc de la validité de cette moyenne.

Pour des disques à têtes fixes, le temps d'accès est égal au délai rotationnel. Dans ce cas, on peut considérer que γ , moitié du temps de rotation, constitue une bonne mesure du délai rotationnel moyen, et donc du temps d'accès moyen Ta. Mais dans le cas de disques à têtes mobiles, il faut ajouter au délai rotationnel le temps de positionnement du bras : le temps d'accès moyen Ta est donc égal à Tb + γ, où Tb est le temps moyen de positionnement du bras si les monotonies sont réparties de façon aléatoire sur le disque. Il faut que cette condition soit effectivement réalisée pour que la modélisation soit valide. Sinon, il faut mesurer expérimentalement Tb.

Au paragraphe 1 , nous indiquons comment allouer la mémoire centrale de manière à minimiser le temps de fusion élémentaire. Au paragraphe 2, nous présentons les algorithmes calculant les stratégies optimales et étudions la forme des arbres optimaux correspondants. Quand les monotonies intiales ont même taille, nous présentons une heuristique très simple calculant des stratégies quasi-optimales à 1 % près.

Au paragraphe 3, nous étudions les temps de fusion correspondant aux stratégies optimales. Nous présentons une formule simple permettant de calculer approximativement les temps de fusion, ainsi qu'une vitesse de tri-fusion mesurant les performances d'une configuration donnée. Nous étudions l'influence des divers paramètres sur les performances et montrons en particulier que le choix du nombre de voies dans les fusions élémentaires a une influence déterminante. Au paragraphe 4, nous proposons des stratégies efficaces quand le temps de traitement en mémoire centrale n'est pas négligeable.

8.1. ALLOCATION DE LA MEMOIRE CENTRALE. TEMPS DE FUSION ELEMENTAIRE

8.1.1. METHODE OPTIMALE D'ALLOCATION

Si le temps d'accès Ta au périphérique est constant et si le temps de traitement en mémoire centrale est négligeable, le temps F d'une fusion élémentaire à d voies est égal à :

$$(1) \qquad F = 2\,Tt\,a_o + Ta \sum_{o<i\leq d} a_i/b_i \qquad\qquad a_o = \sum_{1\leq i\leq d} a_i$$

Rappelons que $a_1,\ldots,a_d$ sont les tailles des monotonies à fusionner et que a_o est la taille de la monotonie résultante. Par rapport à la formule 7.(1), nous avons remplacé $\lceil a_i/b_i \rceil$ par a_i/b_i en supposant que a_i est grand devant b_i. Nous examinerons en 8.1.5 dans quelles conditions l'erreur ainsi commise est acceptable.

Comme Tt, Ta, $a_1,\ldots,a_d$ sont supposés donnés, le problème est de choisir les tailles b_i des tampons en mémoire centrale de manière à minimiser F. Le calcul a été fait par FERGUSON [4]. Comme la somme des tailles des tampons $b_o,b_1,\ldots,b_d$ est égale à la place M disponible en mémoire centrale, les extrêmums de F (considéré comme fonction de b_o, $b_1,\ldots,b_d$) vérifient :

$$\frac{\partial F}{\partial b_o} = \frac{\partial F}{\partial b_1} = \ldots = \frac{\partial F}{\partial b_d}$$

c'est-à-dire :

$$\frac{b_o}{\sqrt{a_o}} = \frac{b_1}{\sqrt{a_1}} = \ldots = \frac{b_d}{\sqrt{a_d}}$$

On vérifie facilement que cet extrêmum est un minimum quand les b_i sont positifs et que leur somme est égale à M. Le temps de fusion et la taille des tampons correspondants sont égaux à :

$$(2) \qquad F_1(a_1,\ldots,a_d) = 2\,Tt\,a_o + \frac{Ta}{M}\left[\sum_{o<i\leq d} \sqrt{a_i}\right]^2$$

$$(3) \qquad b_i = M \frac{\sqrt{a_i}}{\sum_{o<i\leq d} \sqrt{a_i}}$$

Bien que minimisant le temps de fusion élémentaire, cette méthode présente un inconvénient majeur : le calcul des arbres optimaux correspondants prend des temps prohibitifs : il faut une heure d'IRIS-80 pour calculer la stratégie optimale pour soixante monotonies initiales ! Et cela pour un gain de temps généralement faible par rapport à la méthode que nous décrivons maintenant.

8.1.2. METHODE SIMPLIFIEE D'ALLOCATION

Cette méthode consiste à partager la place M disponible en mémoire centrale en d tampons de même taille b pour les monotonies entrantes et un tampon de taille B pour la monotonie résultante. On a évidemment M = B + db. Le temps de fusion élémentaire devient alors :

$$F = 2\,Tt\,a_o + Ta\left[\frac{a_o}{B} + \frac{a_o}{b}\right]$$

Le temps de fusion minimum et la taille des tampons correspondants sont égaux à :

$$(4) \qquad F(a_1,\ldots,a_d) = a_o\left[2\,Tt + \frac{Ta}{M}(1 + \sqrt{d})^2\right]$$

$$(5) \qquad b = M \frac{1}{d + \sqrt{d}} \qquad\qquad B = M \frac{\sqrt{d}}{d + \sqrt{d}}$$

On a évidemment :

$$F_1(a_1,\ldots,a_d) \leq F(a_1,\ldots,a_d)$$

l'égalité n'ayant lieu que si toutes les monotonies entrantes ont même taille.

Avec cette méthode, le temps de fusion élémentaire ne dépend que de a_o et de d. Nous le noterons $F(a_o,d)$:

$$(6)\quad \begin{cases} F(a_o,d) = \varphi(d)\ a_o & a_o = \sum_{1\leq i\leq d} a_i \\ \varphi(d) = 2\ Tt + \frac{Ta}{M}(1+\sqrt{d})^2 \end{cases}$$

8.1.3. COMPARAISON DES DEUX METHODES D'ALLOCATION

Il nous faut comparer les temps - ou coûts - de fusion des arbres optimaux correspondant à chacune des deux méthodes. Pour la méthode optimale, appelons $C_1(a_1,\ldots,a_n)$ le coût de l'arbre de fusion optimal pour n nonotonies de longueurs $a_1,\ldots,a_n$. On définit de même $C(a_1,\ldots,a_n)$ pour la méthode simplifiée. On a évidemment :

$$C_1(a_1,\ldots,a_n) \leq C(a_1,\ldots,a_n)$$

D'après (6), le temps de fusion élémentaire de la méthode simplifiée définit une fonction de coût sur les arbres de fusion du type étudié dans la partie théorique. D'après le corollaire 6.4, le coût optimal $C(a_1,\ldots,a_n)$ est minoré par :

$$k\left[N \text{ Log } N - \sum_{1\leq i\leq n} a_i \text{ Log } a_i\right] \leq C(a_1,\ldots,a_n)$$

où N est la somme des a_i et k le minimum de $\varphi(d)/\text{Log } d$ pour d entier. Cette borne inférieure est atteinte pour une infinité de n-uples $S = (a_1,\ldots,a_n)$ et constitue généralement une bonne minoration de $C(a_1,\ldots,a_n)$. Or, HYAFIL, PRUSKER et VUILLEMIN ont montré (voir [11]) que cette borne inférieure est également valable pour C_1, c'est-à-dire que :

$$k\left[N \text{ Log } N - \sum_{1\leq i\leq n} a_i \text{ Log } a_i\right] \leq C_1(a_1,\ldots,a_n) \leq C(a_1,\ldots,a_n)$$

Il y a donc de bonnes raisons de penser que la méthode simplifiée d'allocation est en fait quasi-optimale. Dans la pratique, l'erreur relative commise en utilisant cette méthode au lieu de la méthode optimale dépend du n-uple : si tous a_i sont égaux cette erreur ne dépasse jamais 0.1 %. Si le n-uple n'est pas trop déséquilibré (rapport du plus grand au plus petit a_i inférieur à 2), cette erreur ne dépasse pas 1 %. Si le n-uple est très déséquilibré, cette erreur peut atteindre 20 %. La méthode la plus simple pour pallier cet inconvénient est de calculer les arbres optimaux correspondant à la méthode simplifiée d'allocation ; on obtient ainsi rapidement un arbre de fusion ; mais ensuite, on alloue les tampons en mémoire centrale selon la méthode optimale. L'erreur relative passe alors à 8 %, ce qui est acceptable.
Dans la suite, le temps de fusion élémentaire sera donc donné par la formule (6). Une fois déterminé l'arbre optimal, il faut examiner pour chaque fusion élémentaire quelle méthode d'allocation employer : si les monotonies entrantes sont très déséquilibrées en taille, on alloue la mémoire centrale selon la méthode optimale ; sinon, on choisit la méthode simplifiée.

8.1.4. AJUSTEMENT DE LA TAILLE DES TAMPONS

Quelle que soit la méthode d'allocation employée, on peut généralement améliorer le temps de fusion élémentaire en modifiant légèrement la taille des tampons. Soit m_i une monotonie entrante ou la monotonie résultante, a_i sa longueur et b_i la taille du tampon correspondant, taille déterminée par l'une des deux méthodes d'allocation précédentes. Le nombre d'accès à cette monotonie est $\lceil a_i/b_i \rceil$, entier égal ou immédiatement supérieur à (a_i/b_i). La méthode d'amélioration consiste à modifier les tailles des tampons de manière à diminuer le nombre d'accès. Remarquons tout d'abord que le nombre d'accès à la monotonie m_i ne change pas si l'on alloue à cette monotonie un tampon plus petit de taille :

$$(7)\quad b_i' = a_i/\lceil a_i/b_i \rceil$$

Par contre, si nous allouons à cette monotonie un tampon plus grand de taille :

$$b_i'' = a_i / \left(\lceil a_i / b_i \rceil - 1 \right) \tag{8}$$

le nombre d'accès à cette monotonie diminue d'une unité. A chaque monotonie correspond donc une place mémoire "excédentaire" $(b_i - b_i')$ et une place mémoire "utile" $b_i'' - b_i$. Si nous ajustons la taille de certains tampons à b_i', nous libérons une certaine place mémoire sans changer le nombre d'accès. Si nous utilisons cette place mémoire pour augmenter la taille des autres tampons et les ajuster à b_i'', nous diminuons le nombre total d'accès au cours de la fusion élémentaire. La règle suivante indique quels tampons ajuster à b_i' et quels tampons ajuster à b_i'' de manière à gagner le maximum de temps d'accès.

Règle d'ajustement des tampons

1) A chaque tampon de taille b_i, faisons correspondre :

$$e_i = b_i'' - b_i'$$

2) Classons les (d+1) tampons par ordre croissant des $e_i (e_o \leq e_1 \leq \ldots \leq e_d)$. Posons $f_i = e_o + \ldots + e_i$ et $f_{-1} = 0$.

3) Si k est l'entier vérifiant :

$$f_{k-1} \leq \sum_{o \leq i \leq d} (b_i - b_i') < f_k \tag{9}$$

alors on peut gagner k temps d'accès en ajustant les tailles des k premiers tampons à b_i'' $(o \leq i < k)$ et en ajustant les tailles des tampons restants à b_i' $(k \leq i \leq d)$. Cette règle permet de gagner le maximum de temps d'accès.

preuve : Montrons d'abord que cette méthode permet de gagner k temps d'accès, l'entier k étant déterminé par la formule (9). Si k n'est pas nul, cette formule peut s'écrire :

$$\sum_{o \leq i \leq k-1} b_i'' - b_i' \leq \sum_{o \leq i \leq d} b - b_i' < \sum_{o \leq i \leq k} b_i'' - b_i'$$

Retranchons $\sum_{o \leq i \leq k-1} b_i - b_i'$ à la première inégalité. Nous obtenons :

$$\sum_{o \leq i \leq k-1} b_i'' - b_i \leq \sum_{k \leq i \leq d} b - b_i'$$

En clair, cette inégalité signifie que la place mémoire utile des k premiers tampons est inférieure ou égale à la place mémoire excédentaire des tampons restants. En utilisant la même taille mémoire M, la méthode d'ajustement fait donc gagner k temps d'accès.

Montrons qu'il n'est pas possible de gagner plus de k temps d'accès. Considérons un ajustement permettant de gagner k' temps d'accès et soit I l'ensemble des k' tampons ajustés à b_i'', et J l'ensemble des tampons restants. La place mémoire utilisée ne devant pas augmenter, on a :

$$\sum_{i \in I} b_i'' - b_i \leq \sum_{i \in J} b_i - b_i'$$

Ajoutons $\sum_{i \in I} b_i - b_i'$ à cette inégalité ; nous obtenons :

$$\sum_{i \in I} e_i \leq \sum_{o \leq i \leq d} b_i - b_i'$$

Les tampons étant classés par ordre croissant des e_i, on en déduit :

$$f_{k'-1} \leq \sum_{o \leq i \leq d} b_i - b_i'$$

D'après (9), on en déduit que $k' \leq k$. On ne peut donc gagner plus de k temps d'accès. ▨

On vérifie facilement que cette méthode permet de diminuer le nombre d'accès de d unités au maximum. Le gain relatif est donc au plus égal à $(d.Ta/\varphi(d).a_o)$. Il est d'autant plus important que la taille de la

monotonie résultante est petite. En particulier, si la taille de la monotonie résultante est de l'ordre de la place M disponible en mémoire centrale, nous montrons ci-dessous qu'il est indispensable d'ajuster la taille des tampons pour que la modélisation reste valide.

8.1.5. VALIDITE DE LA MODELISATION

Pour évaluer le temps de fusion élémentaire correspondant à une allocation donnée des tampons en mémoire, nous avons approximé le nombre d'accès $\lceil a_i/b_i \rceil$ à chaque monotonie par a_i/b_i. Le temps de fusion élémentaire modélisé de la formule (6) sous-estime donc le temps de fusion réel. Nous allons évaluer l'erreur relative ainsi commise en fonction du rapport a_o/M entre la taille de la monotonie résultante et la place disponible en mémoire centrale. Nous indiquons également comment calculer le temps de fusion élémentaire quand l'approximation précédente n'est plus valable. L'erreur relative E est égale à :

$$E = \Delta . Ta / \varphi(d) . a_o$$

Δ est la différence entre le nombre réel d'accès et le nombre d'accès de la modélisation :

$$\Delta = \sum_{o \leq i \leq d} \left[\lceil a_i/b_i \rceil - a_i/b_i \right]$$

$$\Delta = \left\lceil \frac{a_o}{B} \right\rceil + \sum_{1 \leq i \leq d} \left\lceil \frac{a_i}{b} \right\rceil = a_o . (1+\sqrt{d})^2 / M$$

(pour les tailles b et B des tampons, voir la formule (5)) ; Comme $\Delta \leq d+1$, l'erreur relative vérifie :

$$E \leq \frac{M}{a_o} \left[\frac{d+1}{2Tt.M/Ta + (1+\sqrt{d})^2} \right]$$

En pratique, le nombre de voies de fusion dans les stratégies optimales étant borné, le terme entre crochets est inférieur à 1/2 :

$$E \leq \frac{1}{2\ a_o/M}$$

Le tableau suivant indique l'erreur relative maximale en fonction de a_o/M :

a_o/M	1	2.5	5	10	25	50	> 50
E	50 %	20 %	10 %	5 %	2 %	1 %	< 1 %

TABLEAU 1

Validité du temps de fusion

Si a_o/M est plus grand que 25, l'erreur relative est inférieure à 2 %. Par contre, si ce rapport est plus petit que 5, l'erreur dépasse 10 % pour atteindre 50 et 100 % si a_o est du même ordre de grandeur que M. Dans ce cas, il est indispensable d'améliorer l'allocation des tampons en mémoire centrale de manière à diminuer le temps de fusion élémentaire. Nous disposons de deux méthodes :

1) ajuster la taille des tampons,

2) utiliser la méthode optimale d'allocation.

Dans la plupart des cas, il suffit d'ajuster la taille des tampons à partir de la méthode simplifiée d'allocation. La méthode optimale d'allocation ne devient utile que si les monotonies entrantes sont de tailles très différentes (dans ce cas, il faut également ajuster la taille des tampons). D'une manière générale, ces procédés permettent de diminuer le nombre d'accès et donc de diminuer l'erreur relative qui est alors au plus égale à :

$$E \leq \frac{1}{4(a_o/M)^2}$$

Le tableau suivant indique l'erreur relative si l'on utilise les procédés d'amélioration ci-dessus :

a_o/M	1	1.1	1.6	2	2.5	3.5	5
E	25 %	20 %	10 %	6 %	4 %	2 %	1 %

TABLEAU 2

Validité du temps de fusion si l'on ajuste les tampons

Si a_o/M est plus grand que 1.6, l'erreur relative est inférieure à 10 %. On peut donc considérer que la modélisation est valide dès que a_o/M dépasse cette valeur.

Cas des monotonies de même ordre de grandeur que la place mémoire disponible.

Théoriquement, la modélisation n'est plus valide dans ce cas, puisque l'erreur sur le temps de fusion élémentaire dépasse 10 %. En pratique, si nous acceptons une erreur de 20 %, on peut considérer que le temps de fusion élémentaire est donné par la formule :

$$F(a_o,d) = \mathrm{Max}\left[\varphi(d).a_o,\ 2Tt + (d+1).T_a\right] \qquad (10)$$

Naturellement, les méthodes d'allocation de la mémoire centrale sont différentes de celles exposées jusqu'ici. En particulier, si a_o/M est plus petit que 1/2, la meilleure façon d'allouer la mémoire centrale est d'attribuer à chaque monotonie un tampon de même taille que la monotonie (la somme des longueurs de la monotonie résultante et des monotonies entrantes étant inférieure à M). Il y a donc seulement un accès seulement par monotonie et le temps de fusion élémentaire est égal à :

$$2.Tt.a_o + (d+1).Ta \qquad (11)$$

On peut remarquer que le temps de fusion élémentaire est toujours au moins égal à cette valeur puisqu'il y a au moins un accès par monotonie. C'est ce qui explique la formule (10).

Quelles méthodes d'allocation employer pour assurer que $F(a_o,d)$ approxime le temps de fusion élémentaire à 20 % près quand $a_o/M \leq 1.1$?

Si $a_o/M \leq 1/2$, l'erreur relative est nulle si l'on utilise la méthode ci-dessus. Dans les autres cas, si les monotonies entrantes ont approximativement même taille, il suffit d'ajuster la taille des tampons à partir de la méthode simplifiée d'allocation. Si les monotonies initiales sont de tailles très différentes, il faut parfois utiliser des méthodes "ad hoc" qui dépendent de chaque cas particulier. Par exemple, si $1/2 \leq a_o/M \leq 2/3$, les tailles des tampons correspondant aux monotonies $(m_o,m_1,\ldots,m_d)$ doivent être égales à $(a_o/2,\ a_1,\ldots,a_d)$.

Il ne faudrait pas croire qu'une erreur de 20 % sur le temps de fusion élémentaire soit catastrophique pour la modélisation. D'une part, cette erreur est une erreur maximale et non une erreur moyenne. D'autre part, pour un arbre de fusion quelconque, cette erreur ne concerne généralement que les fusions initiales, c'est-à-dire les fusions où certaines monotonies entrantes sont des monotonies initiales. Pour les autres fusions élémentaires, l'erreur est largement inférieure à 20 %.

Validité de la modélisation

En résumé, la modélisation du temps de fusion élémentaire de la formule (6) est bonne à 10 % près dès que a_o/M dépasse 1.6. Si $a_o/M \geqslant 2$, elle est valable à 6 % près. Si $a_o/M \leqslant 1.6$, la modélisation du temps de fusion de la formule (10) est bonne à 20 % près.

Dans la suite, nous supposons généralement que le temps de fusion est égal à $\varphi(d)a_o$, c'est-à-dire que :

$$a_o/M \geqslant (d+1)\Big/(1+\sqrt{d})^2 \qquad (12)$$

Le deuxième membre de cette inégalité étant plus petit que un, on peut considérer que le temps de fusion est égal à $\varphi(d)a_o$ dès que la taille a_o de la monotonie résultante est supérieure à la place M disponible en mémoire centrale.

8.2. LES STRATEGIES OPTIMALES DE TRI-FUSION

Nous avons vu que le temps de fusion élémentaire de d monotonies de tailles $a_1,\ldots,a_d$ suffisamment grandes par rapport à M (voir (12)) est égal à :

(13)
$$\boxed{F(a_o,d) = \varphi(d).a_o} \qquad a_o = \sum_{1\leqslant i\leqslant d} a_i$$
$$\boxed{\varphi(d) = 2\,Tt + \frac{Ta}{M}(1+\sqrt{d})^2}$$

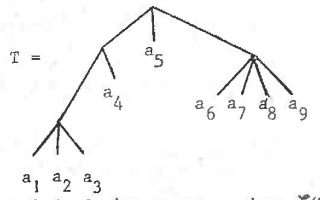
Dans la suite, nous poserons :

(14)
$$\boxed{\beta = 2\,Tt} \quad \boxed{\alpha = Ta/M} \quad \boxed{\lambda = \beta/\alpha} \quad \boxed{\xi(d) = (1+\sqrt{d})^2}$$

On a donc :

$$\boxed{\varphi(d) = \alpha(\lambda + \xi(d))}$$

Ce temps de fusion élémentaire définit une fonction de coût $\mathcal{C}(T)$ sur les arbres de fusion T, qui est la somme des temps des fusions élémentaires représentées par chaque noeud de l'arbre. Soit T l'arbre de fusion suivant pour 9 monotonies initiales de longueurs $a_1,\ldots,a_9$:



le temps total de fusion correspondant $\mathcal{C}(T)$ est égal à :

$$\mathcal{C}(T) = (a_1 + a_2 + a_3)\,\varphi(3) + (a_1 + a_2 + a_3 + a_4)\,\varphi(2)$$
$$+ (a_6 + a_7 + a_8 + a_9)\,\varphi(4) + (a_1 + \ldots + a_9)\,\varphi(3)$$

Trouver la stratégie optimale de fusion pour le 9-uple $(a_1,\ldots,a_9)$ revient donc à trouver l'arbre de fusion optimal qui minimise le coût précédent. On définit de même la stratégie optimale pour un n-uple $(a_1,\ldots,a_n)$. Ce problème a été étudié dans la partie théorique. Nous rappelons ci-dessous quelques définitions. Pour plus de précisions, le lecteur pourra se reporter au chapitre 1.

Définitions : Pour une fonction φ donnée, nous définissons :

1 : $C(a_1,\ldots,a_n)$ ou C(S) : coût optimal pour $S = (a_1,\ldots,a_n)$

2 : C(n,s) : coût optimal pour n monotonies initiales de longueur s.

3 : C(n) = C(n,1).

4 : le degré-limite a : c'est l'entier qui minimise $\varphi(d)/\text{Log } d$.

5 : la constante $k = \varphi(a)/\text{Log } a$

6 : les constantes K et K' qui maximisent et minimisent A(d) pour d entier $(2 \leqslant d \leqslant a)$:

$$A(d) = (d-1)\left[\varphi(d)\,\frac{d}{d-1} - \varphi(a)\,\frac{a}{a-1}\right]$$

7 : la fonction G(x) = k x Log x

8 : la fonction h(x) constituée par les segments de droite joignant les points du plan $\left[a^t,\ G(a^t)\right]$ et $\left[a^{t+1}, G(a^{t+1})\right]$ pour t entier positif ou nul.

Le degré-limite a est celui de la définition 2.3. Remarquons que nous sommes dans le cas étudié au chapitre 4 où il y a un seul degré-limite : d'après la formule (13), la fonction $\varphi(x)/\text{Log } x$ a un seul minimum réel ; elle a donc, soit un seul minimum entier, soit deux minimums entiers consécutifs ; on peut écarter ici ce dernier cas car il ne se produit que si λ est irrationnel (voir 8.2.2.), ce qui est impossible dans une implémentation sur ordinateur.

8.2.1. CALCUL DES STRATEGIES OPTIMALES. HEURISTIQUES

Pour le calcul des stratégies optimales et sous-optimales quand les monotonies initiales sont de tailles inégales, nous renvoyons le lecteur à l'algorithme et aux heuristiques présentés au chapitre 6, paragraphes 5 et 7. Si les monotonies initiales ont même taille, l'algorithme de calcul des arbres optimaux est décrit en 2.1.4. (nous indiquons plus loin en 8.2.2. comment calculer la borne d_{max} sur le nombre de voies de fusion). En fait, cet algorithme calcule seulement le coût optimal pour n monotonies initiales de taille unitaire. Le coût optimal pour n monotonies de tailles s est obtenu en multipliant le résultat par s.
D'autre part, pour obtenir la forme de l'arbre optimal, il suffit d'imprimer aux instructions I_3 et I_4 les valeurs de i et d qui minimisent respectivement A(d,n) et A(1,n). Si les monotonies initiales ont même taille, nous présentons plus loin deux heuristiques très simples qui donnent de bons résultats.
Nous avons vu au paragraphe précédent que le temps de fusion élémentaire n'est égal à $\varphi(d)a_o$ que si a_o est suffisamment grand par rapport à M (voir formule (12)). Si ce n'est pas le cas, il faut remplacer dans les algorithmes et heuristiques les termes de la forme $\varphi(x)y$ par le terme F(y,x) de la formule (10).

8.2.2. FORME DES ARBRES OPTIMAUX. DEGRE MAXIMUM. DEGRE-LIMITE

En général, il n'y a pas de règles simples permettant de déterminer les stratégies optimales (c'est-à-dire les arbres optimaux correspondant à une fonction φ donnée): la seule méthode consiste à utiliser les algorithmes précédents. Cependant, les arbres optimaux obéissent à certaines règles et possèdent certaines propriétés que nous décrivons dans ce paragraphe.

Tout d'abord, certaines transformations préservent l'optimalité : ces transformations opèrent, soit sur les arbres de fusion, soit sur les n-uples, soit sur la fonction φ. Elles permettent à partir d'une stratégie optimale pour un n-uple et une fonction φ donnée d'en déduire beaucoup d'autres.

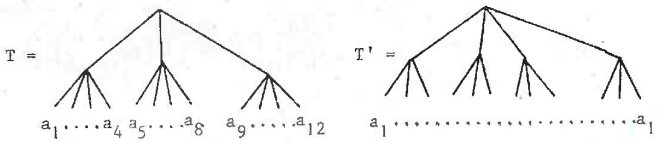
Transformations préservant l'optimalité

Si nous multiplions la longueur des monotonies initiales par une constante, la forme de l'arbre optimal ne change pas ; seul le coût optimal est multiplié par cette constante (proposition 1.3) :

$$C(ca_1,\ldots,ca_n) = c.C(a_1,\ldots,a_n)$$

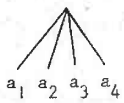
$$C(n,s) = s.C(n,1) = s.C(n)$$

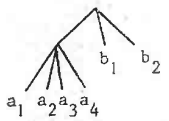
En particulier, l'arbre optimal est toujours le même pour n monotonies initiales de même taille, et cela quelle que soit cette taille. De la même manière, la forme des arbres optimaux ne change pas si l'on multiplie la fonction de coût φ par une constante : on peut donc remplacer $\varphi(d)$ par $\left[\lambda + (1+\sqrt{d})^2\right]$; la forme des arbres optimaux ne dépend donc que du n-uple et de λ.
La transformation d'arbre suivante laisse invariant le coût d'un arbre quelconque : cette transformation, appelée filtrage, consiste à échanger le degré d d'un noeud avec le degré d' de ses fils si ceux-ci ont tous d' pour degré (proposition 1.2). Par exemple, les deux arbres suivants ont même coût pour le 12-uple $(a_1,\ldots,a_{12})$:



Donc si T est optimal, T' l'est également, et réciproquement.

Par ailleurs, les sous-arbres et les contractions d'un arbre optimal sont également optimaux (proposition 1.4). Par exemple, si l'arbre T précédent est optimal, alors le sous-arbre T_1 et la contraction T_2 ci-dessous sont également optimaux :

T_1 = 

T_2 = 

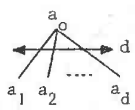
$b_1 = a_5 + \ldots +$

$b_2 = a_9 + \ldots +$

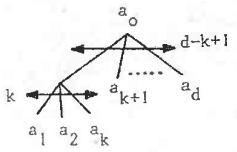
Degré-maximum

Nous allons montrer que, pour une fonction φ donnée, le nombre de voies de fusion des stratégies optimales est au plus égal à un entier d_{max} ne dépendant que de φ. Cette propriété résulte du corollaire 1.5 puisque la fonction $\varphi(d)/\text{Log } d$ est croissante et non bornée à partir d'un certain rang. Mais il existe une démonstration directe qui permet en plus de calculer d_{max}.

Considérons un noeud quelconque d'un arbre optimal de degré d et de poids a_o. Soient $a_1,\ldots,a_d$ les poids de ses fils, classés par ordre croissant de poids. Les sous-arbres et contractions d'un arbre optimal étant optimaux, l'arbre suivant est optimal :



Son coût C_1 est égal à $F(a_o,d)$. Considérons maintenant l'arbre suivant :



$2 \leqslant k \leqslant d-1$

Comme $a_1,..,a_k$ sont les k plus petits poids et que le temps de fusion élémentaire F est croissant par rapport à a_o, le coût C_2 de cet arbre vérifie :

$$C_2 \leqslant F(a_o,\ d-k+1) + F(\tfrac{k}{d}\, a_o,\ k)$$

Le premier arbre étant optimal, on a $C_1 \leqslant C_2$; on en déduit :

(15) $\quad \forall k \quad 2 \leqslant k \leqslant d-1 \quad F(a_o,d) \leqslant F(a_o,d-k+1) + F(\frac{k}{d}\, a_o,k)$

soit :

(16) $\quad \forall k \quad 2 \leqslant k \leqslant d-1 \quad d\left[\xi(d) - \xi(d-k+1)\right] \leqslant k\left[\lambda + \xi(k)\right]$

Tout entier d qui est le degré d'un noeud d'un arbre optimal doit vérifier cette condition. On vérifie facilement qu'il existe un plus grand entier d_o vérifiant cette condition et que tous les entiers plus petits que d_o la vérifient également. Il existe donc une borne d_{max} sur les degrés des noeuds des arbres optimaux qui ne dépend que de λ. En pratique, d_{max} est égal à d_o. Le tableau suivant indique les valeurs de d_{max} en fonction de λ.

d_{max}	λ	d_{max}	λ	d_{max}	λ
8	0.7	16	7.0	24	13.7
9	1.4	17	7.8	25	14.6
10	2.2	18	8.6	26	15.5
11	2.9	19	9.5	27	16.3
12	3.7	20	10.3	28	17.2
13	4.5	21	11.1	29	18.0
14	5.3	22	12.0	30	18.9
15	6.2	23	12.8	31	19.8

TABLEAU 3

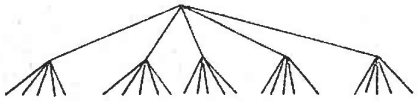
Degré maximum en fonction de λ

Bien entendu, les valeur de λ sont approchées. Nous voyons que d_{max} est une fonction croissante de λ. Par exemple, le degré maximum est 19 pour $8.6 < \lambda < 9.5$. Nous donnons un tableau plus complet de d_{max} en fonction de λ en annexe et indiquons plus loin les valeurs de d_{max} pour quelques configurations courantes.

Pour le calcul du degré maximum, nous avons supposé que $F(a_o,d)$ est égal à $\varphi(d)a_o$. Si ce n'est pas le cas, c'est-à-dire si les monotonies sont de petites tailles, on calcule de la même manière le degré maximum en remplaçant dans (15) les termes $F(a_o,d)$ par l'expression (10). Les valeurs obtenues sont plus grandes que celles du tableau 3 et dépendent évidemment de la taille des monotonies initiales.

Degré-limite

Le degré-limite a de la définition 4 peut être considéré comme le nombre de voies privilégié correspondant à une fonction φ, c'est-à-dire à une configuration donnée.
En effet, d'après le corollaire 2.4, les arbres a-parfaits sont optimaux pour a^p monotonies initiales de même taille. Si nous utilisons par exemple le disque DIMAS avec une place mémoire de 20 K-octets, le degré-limite correspondant est 5. Pour 25 monotonies initiales de même taille, l'arbre de fusion optimal est donc l'arbre 5-parfait suivant :



Comme la contraction préserve l'optimalité, on en déduit que, pour la même configuration, l'arbre suivant est optimal pour 10 monotonies de longueur s et 3 monotonies de longueur 5s :

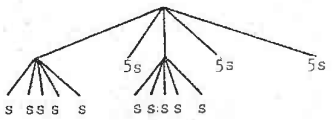


FIGURE 1

Enfin, d'une manière générale, les arbres optimaux ont tendance à avoir pour degré le degré-limite a (proposition 2.1 ; voir aussi le paragraphe 6.6.1.). En particulier, si les n monotonies initiales ont même taille, le degré au sommet des arbres optimaux tend vers le degré-limite a quand n augmente (théorème 8). Le degré-limite représente donc le nombre de voies moyen des arbres optimaux pour une configuration donnée.
Nous montrons en 8.3.2 que le degré-limite se calcule comme suit. Posons :

$$\psi(d) = \frac{\xi(d+1)\ \mathrm{Log}\ d - \xi(d)\ \mathrm{Log}\ (d+1)}{\mathrm{Log}\ (d+1) - \mathrm{Log}\ d} \qquad (17)$$

Le degré-limite a est défini par l'inégalité :

$$\psi(a-1) < \lambda < \psi(a) \qquad (18)$$

Remarquons qu'il n'y a jamais égalité car les valeurs de $\psi(d)$ sont irrationnelles. Nous indiquons ci-dessous les valeurs de $\psi(a)$ en fonction de λ:

a	$\psi(a)$	a	$\psi(a)$	a	$\psi(a)$
4	0.1	12	19.9	20	45.0
5	2.1	13	22.8	21	48.4
6	4.3	14	25.8	22	51.8
7	6.6	15	28.9	23	55.3
8	9.0	16	32.0	24	58.8
9	11.6	17	35.2	25	62.3
10	14.3	18	38.4	26	65.9
11	17.1	19	41.7	27	69.5

TABLEAU 4

Degré-limite a en fonction de λ

Le degré-limite est donc une fonction croissante de λ. Il est égal par exemple à 17 si $32.0 < \lambda < 35.2$. Nous donnons en annexe un tableau plus complet de la fonction ψ.

Degré-maximum, degré-limite pour quelques configurations

Le tableau 5 indique les valeurs de λ, a et d_{max} pour les 6 disques présentés au chapitre 7, quand la place mémoire disponible varie de 20 à 1000 K-octets. Nous avons vu que le degré-limite a et le degré-maximum d_{max} sont des fonctions croissantes de λ, c'est-à-dire de Tt.M/Ta. Ces deux degrés augmentent donc avec le temps de transfert et la place mémoire et diminuent avec le temps d'accès. Sauf pour le DIAD moyenne vitesse, le facteur prépondérant est la taille mémoire : pour les petites tailles mémoire, le nombre de voies moyen (égal au degré-limite a) est de l'ordre de la dizaine et le nombre de voies maximum est de l'ordre de la vingtaine ; pour les grandes tailles mémoires, le nombre de voies moyen et le nombre de voies maximum sont de l'ordre de 50 et 200 respectivement. Pour le DIAD moyenne vitesse, les nombres de voies moyens et maximums sont nettement plus importants, car le temps de transfert est relativement lent.

8.2.3. MONOTONIES INITIALES DE MEME TAILLE. FORME DES ARBRES OPTIMAUX. HEURISTIQUES

En pratique, on a souvent à considérer le cas où les monotonies initiales ont toutes même taille, ou approximativement même taille. Cela arrive en particulier si les monotonies initiales sont générées à partir du fichier que l'on désire trier. Il est donc utile d'étudier ce cas de plus près, d'autant que cette étude donnera une vue plus claire sur la forme et les propriétés des arbres optimaux dans le cas général. Si les monotonies initiales ont même taille, nous savons que la forme des arbres optimaux est indépendante de la taille des monotonies. Pour une configuration donnée, la forme des arbres optimaux dépend donc uniquement du nombre n de monotonies initiales. Nous allons d'abord étudier la valeur du degré ou sommet des arbres optimaux en fonction de n pour la configuration suivante : un disque DIMAS utilisé avec 20 K-octets en mémoire centrale.

	Ta	Tt	M = 20	50	100	200	300	500	1000
TAMBOUR 2301	8.65	0.81	6 3.8 13	9 9.4 19	12 18.8 30	18 37.6 51	24 56.4 72	34 94.0 112	57 188.0 211
DIAD grande vitesse	20.03	0.49	5 1.0 9	6 2.4 11	7 4.9 14	9 9.7 14	11 14.6 26	14 24.4 37	22 48.7 64
DIAD moyenne vitesse	20.03	7.12	10 14.2 25	18 35.6 49	28 71.1 88	46 142.2 163	63 213.3 238	94 355.6 385	164 711.1 763
MD-100	38.33	1.56	5 1.6 10	6 4.1 13	8 8.2 18	11 16.3 27	14 24.5 37	19 40.8 55	31 81.5 99
MD-50	47.50	4.22	6 3.5 12	8 8.9 19	12 17.8 29	18 35.5 49	23 53.3 69	33 88.8 107	55 177.6 200
DIMAS	92.50	4.37	5 1.9 10	7 4.7 14	9 9.4 19	12 18.9 30	15 28.3 41	21 47.2 62	34 94.4 113

TABLEAU 5

Valeurs de λ, a, d_{max} pour quelques configurations

Dans ce cas, nous savons (voir tableau 5) que le nombre de voies de fusion est au plus égal à 10 dans les stratégies optimales et que le degré-limite est 5. Pour n = 5, 25, ..., 5^p, l'arbre optimal est donc l'arbre 5-parfait de profondeur p (voir 8.2.2.).

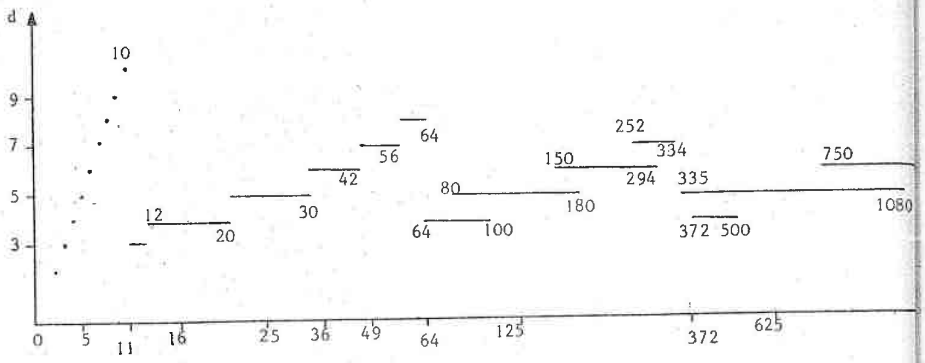


FIGURE 2

Degré au sommet d des arbres optimaux pour n monotonies initiales de même taille (disque DIMAS avec 20 K-octets)

Sur la courbe ci-dessus, on constate que l'arbre optimal est l'arbre plat jusqu'à $n = d_{max} = 10$ (cette propriété est valable dans le cas général). Pour n = 11, le degré au sommet passe brusquement à 3 et augmente jusqu'à 8 pour n = 64. Pour cette valeur de n, il y a deux arbres optimaux équivalents de degrés au sommets 8 et 4. De n = 65 à n = 334, le degré au sommet augmente régulièrement de 4 à 7. L'amplitude des oscillations diminue ensuite. On peut montrer qu'à partir de 300.000 monotonies initiales, il existe toujours un arbre optimal de degré au sommet égal au degré-limite 5 (cette propriété est vérifiée dans le cas général ; voir théorème 8). Par ailleurs, on vérifie que le degré au sommet est égal à 5 si n est une puissance de 5.

On peut remarquer que pour certaines valeurs de n, il y a plusieurs degrés au sommet possibles correspondant à plusieurs arbres optimaux. Par exemple si n = 90, un arbre optimal possible est :

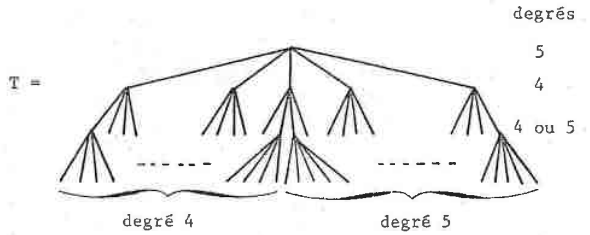


FIGURE 3

Si nous filtrons le sommet de cet arbre, nous obtenons un arbre équivalent, donc optimal, mais de degré au sommet 4 :

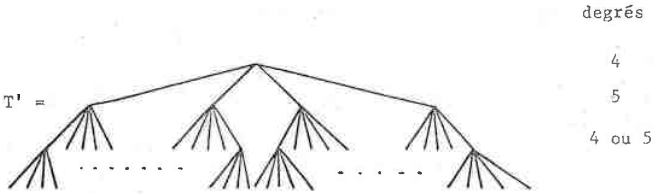


FIGURE 3 bis

En fait, à partir de la courbe précédente, on peut reconstituer dans la plupart des cas la forme des arbres optimaux. En effet, comme dans le cas n = 90, le degré des noeuds des arbres optimaux ne diffèrent le plus souvent que de un. Nous allons préciser cette notion à l'aide de la définition suivante :

Définition 1 : Un arbre est dit régulier si toutes ses feuilles sont à la même profondeur p et si les noeuds situés à une même profondeur ont tous même degré. Un arbre est dit pseudo-régulier si cette dernière propriété n'est vérifiée qu'aux profondeurs 1, 2, ..., p-2.

Par exemple, l'arbre T de la figure 3 est pseudo-régulier : toutes ses feuilles sont à la profondeur 2 et les noeuds situés à la profondeur 1 ont tous pour degré 4. Dans la suite, nous appelerons arbre à deux degrés un arbre dont les degrés sont égaux soit à un entier unique, soit à deux entiers consécutifs. Nous dirons par exemple que l'arbre T précédent est pseudo-régulier à deux degrés : 4 et 5.

Arbres réguliers et pseudo-réguliers à deux degrés optimaux

D'une manière générale, si les monotonies initiales ont même taille, les arbres optimaux sont le plus souvent des arbres réguliers ou pseudo-réguliers à deux degrés. Cette propriété est la conséquence d'une propriété plus générale que nous indiquons ci-dessous et dont la démonstration figure en annexe :

Propriété 1 : Soit un arbre optimal pour une fonction de coût du type (1). Si tous les fils d'un même noeud de degré d ont même degré d', alors la différence $|d-d'|$ n'excède pas un.

On en déduit facilement que les arbres réguliers et pseudo-réguliers optimaux sont forcément "à deux degrés". Si nous reprenons l'exemple de la figure 2, les arbres réguliers suivants sont optimaux pour n = 80 et n = 100 :

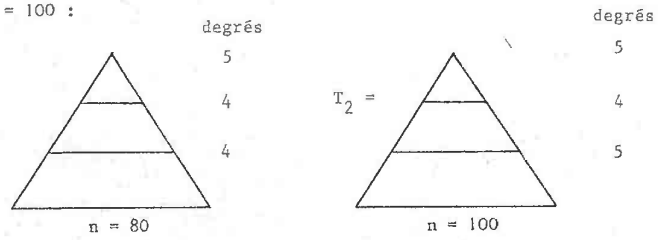


FIGURE 4

Le degré au sommet de T_1 est égal à 5 ; les 5 noeuds situés à la profondeur 1 ont pour degré 4 et les 20 noeuds situés à la profondeur 2 ont pour degré 4. Comme le filtrage ne change pas les coûts, l'ordre des degrés des différentes profondeurs est indifférent et les arbres réguliers sont entièrement décrits par leur décomposition. Nous indiquons ci-dessous les arbres réguliers optimaux dans l'exemple précédent pour n allant de 11 à 125 :

n	décomposition	n	décomposition
16	4^2	49	7^2
20	$4^1,5^1$	56	$7^1,8^1$
25	5^2	64	8^2
30	$5^1,6^1$	80	$4^2,5^1$
36	6^2	100	$4^1,5^2$
42	$6^1,7^1$	125	5^3

4^3

Entre ces valeurs de n, les arbres optimaux sont pseudo-réguliers à deux degrés. Par exemple, pour n = 90, l'arbre optimal se déduit facilement des arbres réguliers optimaux pour n = 80 et n = 100 : il suffit de modifier les degrés des noeuds situés à la profondeur 2. Ceci est valable pour tous les arbres optimaux entre 80 et 100 : le degré au sommet est 5 ; les noeuds de profondeur 1 ont pour degré 4 ; à la profondeur 2, il y a q noeuds de degré 5 et r noeuds de degré 4, les entiers q et r étant solution de :

$$q + r = 20 \qquad 5q + 4r = n$$

Bien entendu, les arbres optimaux ne sont pas toujours réguliers ou pseudo-réguliers. Mais c'est très souvent le cas, en particulier pour les petites valeurs de n : dans l'exemple précédent, pour $2 \leq n \leq 5^5=3125$, les arbres optimaux sont réguliers ou pseudo-réguliers sauf pour les valeurs suivantes de n :

$$n = 11, \quad 335 \leq n \leq 371, \quad 1804 \leq n \leq 1970$$

On en déduit que l'heuristique suivante sera pratiquement toujours optimale :

Heuristique à deux degrés.

Cette heuristique est une amélioration de la méthode de BLACK [1]. Elle consiste à prendre comme stratégie de fusion le meilleur arbre pseudo-régulier pour n donné.
Considérons un entier p quelconque vérifiant :

$$(19) \qquad \frac{\text{Log } n}{\text{Log } d_{max}} < p \leqslant \frac{\text{Log } n}{\text{Log } 2}$$

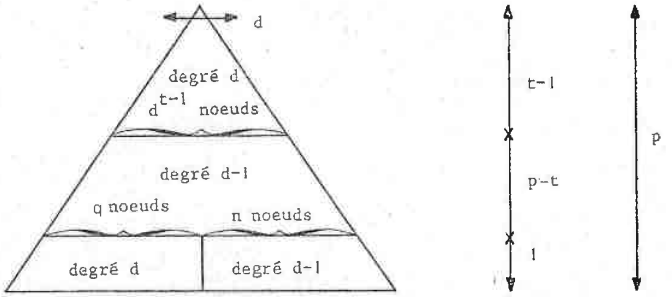
A p, nous faisons correspondre de manière unique un arbre pseudo-régulier profondeur p. Les degrés de ses noeuds sont égaux à d ou d-1, l'entier d étant déterminé par :

$$(d-1)^p < n \leqslant d^p$$

Soient t, q et r les entiers définis par :

$$\begin{cases} d^{t-1}(d-1)^{p-t+1} < n \leqslant d^t(d-1)^{p-t} \\ q = n - d^{t-1}(d-1)^{p-t+1} \\ r = d^t(d-1)^{p-t} - n \end{cases}$$

L'arbre pseudo-régulier a la forme suivante :



De la profondeur 0 à la profondeur t-2, tous les noeuds ont degré d ; de la profondeur t-1 à la profondeur p-2, tous les noeuds ont degré d-1 ; à la profondeur p-1, il a q noeuds de degré d et r noeuds de degré d-1.
Calculons le coût H(n,p) de cet arbre par la méthode des coûts partiels (voir proposition 1.1) en supposant que les monotonies initiales sont de longueur unitaire. Nous obtenons :

$$H(n,p) = \varphi(d)\left[ns - r\right] + \varphi(d-1)\left[n(p-s) + r\right]$$

L'heuristique à deux degrés consiste à choisir comme stratégie de fusion celle qui minimise H(n,p) quand p varie. Si p_o est l'entier vérifiant :

$$H(n,p_o) = \min_p H(n,p)$$

le coût de l'arbre calculé par cette heuristique sera égal à $s.H(n,p_o)$ pour n monotonies initiales de longueur s.
Cette heuristique donne d'excellents résultats : très souvent, elle calcule l'arbre optimal ; si ce n'est pas le cas, la différence relative avec le coût optimal n'excède pas 1 %.

Heuristique du degré-limite

Cette heuristique consiste à prendre comme arbre de fusion les arbres a-équilibrés, l'entier a étant le degré-limite. Rappelons que, dans un arbre a-équilibré, tous les noeuds, sauf peut-être un, ont pour degré a, et la différence de profondeurs des feuilles n'excède pas un (voir définition 1.14). L'avantage de cette méthode est sa simplicité : tous les noeuds étant de degré a, l'allocation des tampons en mémoire centrale sera toujours la même tout au long de processus de tri-fusion.
Calculons le coût A(n,s) des arbres ainsi obtenus pour n monotonies de longueur s. On a évidemment A(n,s) = s.A(n,1). D'autre part, si l'on fait d = a dans la formule 1.(46), on obtient :

$$(20) \qquad A(n,1) = h(n) + (d-1)\left[\varphi(d)\frac{d}{d-1} - \varphi(a)\frac{a}{a-1}\right]$$

d est un entier vérifiant $2 \leqslant d \leqslant a$ et h la fonction de la définition 8 (si l'arbre a-équilibré possède un noeud de degré différent de a, l'en-

tier d est le degré de ce noeud). D'après la définition 6, on a donc :

$$h(n) + K' \leqslant A(n,1) \leqslant h(n) + K \tag{21}$$

Sur la figure 5 du paragraphe suivant, nous avons tracé la courbe h(n). D'autre part, nous verrons que le coût optimal vérifie :

$$G(n) \leqslant C(n,1) \leqslant h(n) + K$$

On en déduit que la différence relative entre le coût des arbres calculés par cette heuristique et le coût optimal n'est jamais très grande et tend vers zéro quand le nombre de monotonies initiales augmente. Néanmoins, cette heuristique est moins bonne que la précédente : elle ne calcule les arbres optimaux que si $n = a^p$; la perte en pourcentage peut atteindre 50 % si $n < a$; elle dépasse rarement 10 % si $n > a$ et 5 % si $n > a^2$.

8.3. TEMPS DE FUSION OPTIMAUX. ETUDE DES PERFORMANCES EN FONCTION DE LA CONFIGURATION

Pour une configuration donnée, nous présentons une méthode permettant d'évaluer approximativement les temps de fusion optimaux en fonction du nombre et des longueurs des monotonies initiales. Nous introduisons ensuite une mesure des performances d'une configuration du point de vue du tri-fusion. Cette mesure nous permettra de comparer diverses configurations et d'étudier l'influence respective des divers paramètres sur les performances : temps d'accès et de transfert, place mémoire disponible, nombre de voies de fusion, méthode d'allocation de la mémoire.

8.3.1. TEMPS DE FUSION OPTIMAUX

Pour calculer le temps de fusion optimal $C(a_1,\ldots,a_n)$ correspondant à n monotonies initiales de longueurs $a_1,\ldots,a_n$, la seule méthode consiste à utiliser les algorithmes décrits précédemment. Il est cependant possible d'avoir une idée des temps de fusion optimaux par la formule suivante (voir corollaire 6.5) :

$$k\left[N \operatorname{Log} N - \sum_{1\leqslant i\leqslant n} a_i \operatorname{Log} a_i\right] \leqslant C(a_1,\ldots,a_n) < k\left[N \operatorname{Log} N - \sum_{1\leqslant i\leqslant n} a_i \operatorname{Log} a_i\right] + \varphi(a)N \tag{22}$$

N est égal à la somme des a_i et k est égal à $\varphi(a)/\operatorname{Log} a$ (l'entier a est le degré-limite). Nous verrons plus loin que la borne inférieure de la formule (22) constitue généralement une bonne estimation du coût optimal. On peut vérifier que le coût optimal est égal à cette borne inférieure si et seulement si les longueurs des monotonies initiales peuvent se mettre sous la forme $a_i = a^{-p_i}.N$. L'arbre optimal est alors une contraction d'arbre a-parfait, la monotonie de longueur a_i étant à la profondeur p_i. Dans l'exemple de la figure 1, les longueurs des monotonies peuvent se mettre sous la forme :

$$s = a^{-2}.N \qquad 5s = a^{-1}.N \qquad (a = 5,\ N = 25s)$$

L'arbre optimal est bien une contraction d'arbre 5-parfait et son coût est égal à la borne inférieure précédente.

Si les n monotonies initiales ont même longueur s, le coût optimal est proportionnel à s ($C(n,s) = s.C(n,1) = s.C(n)$), la formule (22) devient donc :

$$k\, n \operatorname{Log} n \leqslant C(n) < k\, n \operatorname{Log} n + \varphi(a)N \tag{23}$$

En fait, il existe une meilleure borne supérieure de C(n) : le coût optimal étant inférieur par définition au coût des arbres a-équilibrés, on déduit de la formule (21) que :

$$C(n) \leqslant h(n) + K$$

Sur la figure suivante, nous avons représenté les fonctions k n log n et k(n) qui encadrent C(n) :

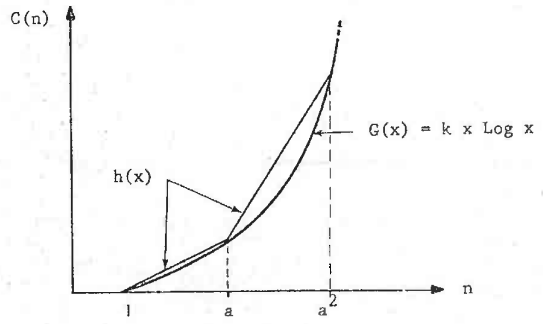


FIGURE 5

Borne inférieure et supérieure de C(n)

Si $n = a^p$, l'arbre optimal est l'arbre a-parfait de profondeur p et le coût optimal correspondant est égal à la borne inférieure G(n).
Le tableau suivant permet de se faire une idée plus précise des temps de fusion optimaux quand les monotonies initiales ont même taille. Ce tableau indique les valeurs de C(n) quand on utilise un disque MD-100 avec une place mémoire variant de 20 à 1000 K-octets. Les colonnes indiquent successivement la taille mémoire M, λ, le degré-limite a, k et les valeurs de C(n) exprimées en secondes, pour n variant de 20 à 300. La deuxième ligne indique la différence en pourcentage par rapport à la borne inférieure $\lfloor k\ n\ \mathrm{Log}\ n \rfloor$. La troisième ligne indique successivement le degré au sommet et la profondeur maximum de l'arbre optimal. Par exemple, pour M = 20, C(50,s) est égal à 2.88s secondes ; la différence en pourcentage par rapport à $\lfloor k\ n\ \mathrm{Log}\ n \rfloor$ est de 2.2 % ; le degré au sommet de l'arbre optimal correspondant est 8 et sa profondeur maximum 2.

M	λ	a	k	n = 20	50	100	300
20	1.63	5	14.41	0.87 0.9 5 2	2.88 2.2 8 2	6.68 0.6 5 3	25.05 1.6 7 3
50	4.08	6	6.84	0.42 3.4 5 2	1.34 0.2 8 2	3.23 2.7 5 3	11.71 0.1 7 3
100	8.16	8	4.20	0.27 8.9 5 2	0.83 0.5 8 2	1.95 0.9 10 2	7.26 0.9 7 3
200	16.31	11	2.79	0.18 6.0 20 1	0.57 4.2 8 2	1.29 0.2 10 2	4.94 3.4 18 2
300	24.47	14	2.27	0.14 2.1 20 1	0.48 8.8 8 2	1.07 2.0 10 2	3.92 0.8 18 2
500	40.78	19	1.81	0.11 0.0 20 1	0.41 14.7 50 1	0.89 6.9 10 2	3.10 0.2 18 2
1000	81.55	31	1.39	0.09 2.5 20 1	0.28 3.3 50 1	0.76 18.3 10 2	2.49 4.5 18 2

TABLEAU 6

Temps de fusion optimaux pour un disque MD-100

8.3.2. VITESSE ET TEMPS DE TRI-FUSION D'UNE CONFIGURATION

Examinons la borne inférieure du coût optimal de la formule (22) : elle est égale au produit de deux facteurs ; le premier, k, dépend exclusivement de la configuration (voir définition 5) ; le deuxième terme dépend exclusivement du nombre et des longueurs des monotonies à fusionner. Si cette borne inférieure est une bonne estimation du coût optimal, alors la constante k constitue une bonne mesure des performances d'une configuration.

D'après la formule (22), l'erreur $E(a_1,\ldots,a_n)$ commise en approximant le coût optimal par la borne inférieure est plus petite que :

$$E(a_1,\ldots,a_n) < \frac{\text{Log } a}{\sum_{1\leq i\leq n} (a_i/N) \text{ Log } (N/a_i)}$$

La fonction x log x étant convexe, plus les tailles des monotonies initiales sont équilibrées, et plus cette erreur diminue. Si les monotonies initiales ont même taille, cette erreur devient :

$$E(n) < \text{Log } a / \text{Log } n$$

Elle tend vers zéro quand le nombre de monotonies initiales augmente. Sur l'exemple du tableau 6, l'erreur E(n) ne dépasse pas 18.3 % et est généralement inférieure à 5 %. Si les monotonies initiales sont de tailles différentes, l'erreur peut être plus importante, mais, en moyenne, la borne inférieure est une bonne approximation du coût optimal.

On peut donc considérer que k est une bonne mesure des performances d'une configuration. Dans la suite, nous appelerons k le temps de tri-fusion de la configuration, exprimé en millisecondes par K-octets ; nous appelerons v=1/k la vitesse de tri-fusion. Ces constantes sont définies par la formule :

$$(24) \qquad k = \frac{1}{v} = \frac{2.\text{Tt} + (\text{Ta}/\text{M})(1+\sqrt{a})^2}{\text{Log } a} = \min_{d\geq 2} \frac{\varphi(d)}{\text{Log } d}$$

Rappelons que a est le degré-limite de la configuration.

Calcul de la vitesse de tri-fusion et du degré-limite

La formule (24) peut s'écrire, avec les notations (14) :

$$(25) \qquad \frac{k}{\alpha} = \min_{d\geq 2} \frac{\lambda + \xi(d)}{\text{Log } d}$$

Soit $D_d(\lambda)$ la droite d'équation :

$$D_d(\lambda) = \frac{\lambda + \xi(d)}{\text{Log } d}$$

la formule (25) devient :

$$k/\alpha = \min_{d\geq 2} D_d(\lambda)$$

On vérifie facilement que les droites D_d définissent la ligne brisée concave suivante :

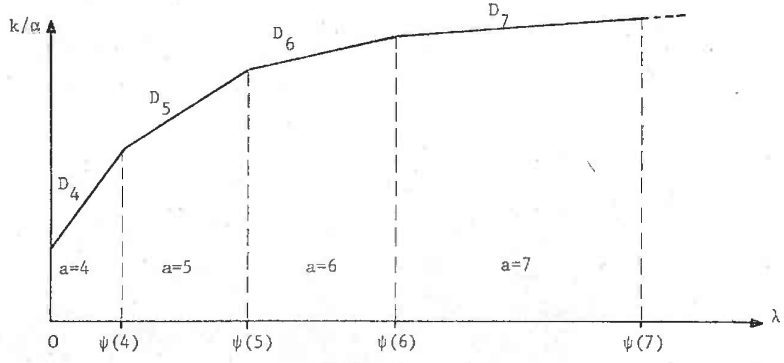


FIGURE 6

Temps de tri-fusion en fonction de λ

D'après la définition de $D_d(\lambda)$, l'abscisse $\psi(d)$ du point d'intersection de D_d et D_{d+1} est égale à :

$$(26) \qquad \psi(d) = \frac{\xi(d+1) \text{ Log } d - \xi(d) \text{ Log } (d+1)}{\text{Log } (d+1) - \text{Log } d}$$

Pour λ fixé, le terme $D_d(\lambda)$ est donc minimum pour d = a, l'entier a étant défini par :

(27) $\psi(a-1) < \lambda < \psi(a)$

Cette inégalité détermine le degré-limite a. On en déduit k et v par la formule (24). Pour les valeurs de $\psi(d)$, voir le tableau 4 et en annexe.

8.3.3. COMPARAISON DES PERFORMANCES DE DIVERSES CONFIGURATIONS

La vitesse et le temps de tri-fusion ne dépendent que des caractéristiques matérielles de la configuration : place mémoire disponible, temps de transfert, temps d'accès. Il s'agit ici d'étudier l'influence respective de ces divers facteurs sur les performances.

On peut faire une première constatation : le temps d'accès et la place mémoire jouent des rôles symétriques : seul le rapport Ta/M intervient dans le temps de fusion élémentaire et dans la vitesse de tri-fusion. Autrement dit, diviser par deux le temps d'accès a le même effet sur les performances que doubler la place mémoire disponible.

La figure 7 indique les temps de tri-fusion en fonction de la taille mémoire M pour les six disques présentés au chapitre 7. On remarque tout d'abord que les temps de tri-fusion diminuent très vite avec M pour les petites valeurs de M. Dès que la place mémoire dépasse 200 K-octets (300 K pour le DIMAS) les temps de tri-fusion diminuent beaucoup plus lentement avec M : nous avons indiqué par un point sur les courbes les valeurs de M pour lesquelles les temps de tri-fusion sont inférieurs de 20 % aux temps de tri-fusion pour M = 500 K. Pour tous les disques, les tailles mémoires correspondantes sont de l'ordre de 300 K.

Examinons maintenant l'influence du temps de transfert et du temps d'accès du périphérique. Classons les périphériques par temps de tri-fusion croissants pour les diverses tailles mémoires. Pour les petites tailles mémoires, ce classement correspond exactement au classement par temps d'accès croissants. A partir de 200 K-octets, il correspond au classement par temps de transfert croissants (si l'on excepte le DIMAS).

k \ M	20	50	100	200	300	500	1000
DIMAS	35.5	17.1	10.7	7.2	5.9	4.7	3.7
DIAD-Moy.	13.7	8.7	6.7	5.3	4.7	4.1	3.5
MD-50	20.5	10.7	7.2	5.2	4.4	3.6	2.9
MD-100	14.4	6.8	4.2	2.8	2.3	3.3	1.4
TAMBOUR	3.8	2.0	1.9	0.9	0.8	0.7	0.6
DIAD-Gd.	7.1	3.2	1.4	1.1	0.9	0.7	0.5

	Ta	Tt
DIMAS	92.50	4.37
DIAD-Moy.	20.03	7.12
MD-50	47.50	4.22
MD-100	38.33	1.56
TAMBOUR	8.65	0.81
DIAD-Gd.	20.03	0.49

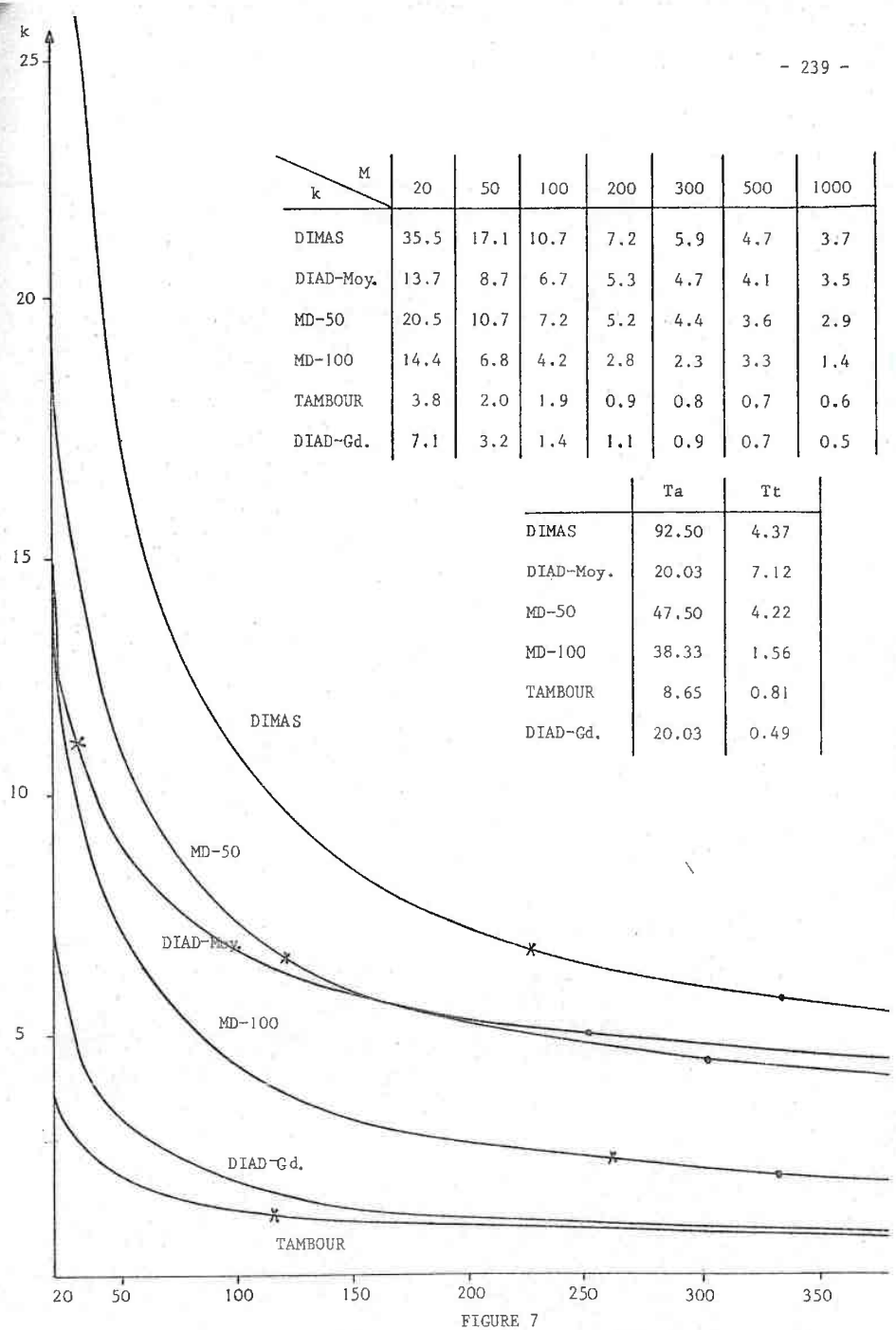


FIGURE 7

Temps de tri-fusion en fonction de la place mémoire

D'une manière générale, le temps d'accès est donc le facteur prépondérant pour les petites tailles mémoire ; pour les grandes tailles mémoire, c'est le temps de transfert. On peut d'ailleurs remarquer que les disques à tête mobile ayant une grande vitesse de transfert sont meilleurs à mémoire égale que certains disques à tête fixe dès que la mémoire est raisonnablement grande. Par exemple, le disque à tête mobile MD-50 est meilleur que le disque DIAD moyenne vitesse dès que M dépasse 170 K. La même remarque vaut pour le DIAD grande vitesse dont les performances sont comparables à celles du TAMBOUR dès que M dépasse 200 K ; et cela bien que le temps d'accès du DIAD soit trois fois plus grand que celui du tambour. En résumé, dès que la place mémoire dépasse 200 K-octets, il est préférable d'utiliser des disques ayant une grande vitesse de transfert.

Variations relatives des performances avec les caractéristiques de la configuration.

Afin de préciser l'influence respective de la place mémoire et des temps de transfert et d'accès sur les performances, nous indiquons ci-dessous les variations relatives de k et v avec M, Tt et Ta. En valeur absolue, elles sont égales à :

$$\frac{dk}{k} \Big/ \frac{dTt}{Tt} = \frac{1}{1+A} \qquad \begin{cases} A = (1+\sqrt{a})^2/\lambda \\ \lambda = 2.Tt.M/Ta \end{cases}$$

$$\frac{dk}{k} \Big/ \frac{dTa}{Ta} = \frac{dk}{k} \Big/ \frac{dM}{M} = \frac{A}{1+A}$$

L'entier a est le degré-limite. Notons que les variations relatives de la vitesse et du temps de tri-fusion sont les mêmes puisque dk/k et dv/v sont égaux en valeur absolue.
On constate que les variations relatives par rapport à Ta et M sont égales ce qui est normal puisque les performances ne dépendent que de Ta/M.
D'autre part, ces formules indiquent que la somme des variations relatives par rapport à Tt et Ta, ou par rapport à Tt et M, est égale à l'unité. En conséquence, si 1/(1+A) est plus grand que 1/2, c'est-à-dire si A < 1, le temps de transfert a plus d'influence sur les performances que le le temps d'accès ou la place mémoire. Si A > 1, c'est le contraire.

On peut montrer que A est une fonction décroissante de λ qui est égale à un pour $\lambda = (1+\sqrt{13})^2 = 10.60 \ldots$ Pour cette valeur de λ, le degré-limite a est égal à 13. En conséquence :

$$\begin{cases} \text{si } \lambda > 10.6 \text{ ou si } a > 13, \text{ il est préférable d'améliorer Tt} \\ \text{si } \lambda < 10.6 \text{ ou si } a < 13, \text{ il est préférable d'améliorer Ta ou M} \end{cases}$$

Sur les courbes de la figure 7, nous avons marqué d'une croix les valeurs de M au delà desquelles l'influence du temps de transfert sur les performances est prépondérante.

8.3.4. AMELIORATIONS DES TEMPS D'ACCES

Nous avons examiné en 7.3. diverses méthodes qui permettent de diminuer les temps d'accès au périphérique. La méthode d'optimisation du mouvement du bras sera étudiée à part au chapitre 9. Nous étudions ici l'incidence sur les performances des deux autres méthodes : la première diminue les temps de positionnement du bras, la seconde améliore le délai rotationnel.

Amélioration des temps de positionnement du bras

Cette méthode s'applique pour des disques à bras mobile et consiste à n'utiliser qu'une partie du disque comme espace de travail au cours du tri-fusion dans le cas où la taille du fichier trié résultant est nettement plus petite que la capacité du disque. Comme le bras ne se déplace que sur une partie du disque, les temps moyens Tb de positionnement du bras sont plus petits. Le tableau suivant indique les temps d'accès moyens Ta correspondants si l'on utilise qu'une fraction d'un disque DIMAS :

Fraction du disque utilisée	espace de travail (K-octets)	Tb	Ta	Gain sur le temps d'accès
tout	24.000	80	92.5	
moitié	12.000	61	73.5	21 %
quart	6.000	50	62.5	32 %
huitième	3.000	41	53.5	42 %

TABLEAU 7

Valeur des temps d'accès si l'on utilise qu'une partie du disque DIMAS

Le temps moyen de déplacement du bras est calculé à partir des temps de déplacements du bras en supposant que les accès sont aléatoires à l'intérieur de l'espace disque utilisé (voir chapitre suivant : paragraphe 1, figures 2 et 3). Pour obtenir les temps d'accès moyens, il faut ajouter le délai rotationnel (12.5 ms). La dernière colonne indique les améliorations des temps d'accès.
Le tableau suivant indique les gains en pourcentage sur les temps de tri-fusion quand on n'utilise qu'une partie du disque :

Fraction du disque utilisée	M = 50	200	500	1000
tout	17.1	7.2	4.7	3.7
moitié	15 %	11 %	8 %	7 %
quart	24 %	17 %	14 %	12 %
huitième	31 %	23 %	18 %	16 %

TABLEAU 8

Temps de tri-fusion si l'on utilise qu'une partie du disque DIMAS

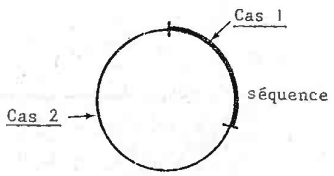
La première ligne indique les temps de tri-fusion quand on utilise la totalité du disque. Comme le laissaient prévoir les règles du paragraphe précédent, les gains sont d'autant plus grands que la place mémoire est petite. D'une manière générale, ils ne dépassent pas 20 %, sauf dans le cas où l'espace disque utilisé et la mémoire disponible sont petits. Dans tous les cas, les gains ne sont jamais en proportion de l'amélioration du temps d'accès (voir tableau 7).
En résumé, cette méthode n'améliore pas de façon décisive les temps de fusion. C'est encore plus net pour les disques dont les temps de déplacement du bras sont plus petits (disques MD-100 et MD-50 par exemple).

Amélioration du délai rotationnel

Cette méthode consiste à diminuer le délai rotationnel en commençant les entrées/sorties le plus tôt possible, sans attendre que la tête de lecture/écriture soit positionnée en début de séquence. Nous nous limiterons au cas de disques à têtes fixes ; pour les disques à têtes mobiles, cette méthode ne diminue que faiblement les temps d'accès, car le délai rotationnel γ est petit devant le temps moyen Tb de déplacement du bras (voir tableau 7.1).
Nous avons vu en 7.3 que si les longueurs des séquences à lire où à écrire sont des multiples de la capacité d'une piste de disque, cette méthode permet en théorie d'éliminer le délai rotationnel. En pratique, cela n'est pas possible : chaque piste est divisée en un certain nombre de secteurs et les entrées/sorties ne peuvent commencer qu'à un début de secteur. Si S est le nombre de secteurs par pistes et γ la moitié de la durée d'une révolution, le temps d'accès est en moyenne égal à :

$$\gamma/S$$

Par rapport à la méthode classique, le temps d'accès moyen est donc divisé par le nombre S de secteurs, qui est de l'ordre de 5 à 20. L'amélioration du temps d'accès est donc très importante ; mais il faut bien avoir à l'esprit les conditions à satisfaire pour que les longueurs des séquences soient toujours des multiples de la capacité d'une piste au cours du processus de tri-fusion : il faut bien sûr que les longueurs des monotonies initiales soient des multiples de la capacité d'une piste ; mais il faut aussi que les tailles des tampons en mémoire centrale vérifient cette condition.
Supposons maintenant que les longueurs x des séquences soient plus petites que la capacité P d'une piste et calculons le temps d'accès moyen. Il faut distinguer deux cas suivant que la tête de lecture est ou non positionnée sur la séquence au moment de l'ordre d'entrée. Dans le premier cas, la lecture de la fin de la séquence peut commencer tout de suite, mais il y a un temps d'attente égal à $2\gamma(P-x)/P$ avant qu'on puisse lire le début de la séquence. Dans le deuxième cas, le temps d'attente avant que la lecture puisse commencer est égal en moyenne à $\gamma(P-x)/P$.



x = longueur de la séquence

P = capacité d'une piste

2γ = temps d'une révolution

Il y a en moyenne x/P chances d'être dans le cas 1 et (P-x)/P chances d'être dans le cas 2. Le temps d'attente moyen pour lire une séquence de longueur x est donc égal à :

$$\frac{x}{p}\left[2\gamma(P-x)/P\right] + \frac{P-x}{P}\left[\gamma(P-x)/P\right]$$

Pour obtenir le temps d'accès moyen à une séquence de longueur quelconque x plus petite que P, il suffit d'intégrer cette expression entre 0 et P. Finalement, le temps d'accès moyen est égal à :

$$\frac{2}{3}\gamma$$

Si les séquences sont plus longues que la capacité d'une piste, on peut généralement diminuer encore le temps d'accès moyen. Mais on ne peut plus donner de méthode générale d'amélioration dans ce cas : en effet, chaque séquence est alors divisée en un certain nombre de sous-séquences réparties sur différentes pistes du disque ; les méthodes d'amélioration du délai rotationnel dépendent donc du placement des sous-séquences et des caractéristiques des disques (nombre de têtes mobiles, nombre de pistes par cylindre, organisation du disque, etc...).
Le tableau suivant indique les gains sur les temps de tri-fusion qui résultent des procédés précédents pour un disque DIAD moyenne vitesse. La durée d'une révolution étant de 40 ms, le temps d'accès moyen est de 13.34 ms si la longueur des séquences est quelconque. Dans le cas où les longueurs des séquences sont des multiples de la capacité d'une piste (soit 5760 octets), le temps d'accès moyen est de 1.25 ms ; en effet, chaque piste comprend 16 secteurs de 360 octets.

Ta	M = 50	200	500	1000
20	8.7	5.3	4.1	3.5
1.25	57 %	46 %	41 %	36 %
13.34	15 %	11 %	9 %	8 %

TABLEAU 9

Temps de tri-fusion avec amélioration du délai rotationnel

La première ligne indique les temps de tri-fusion sans amélioration. La deuxième ligne indique les gains de performances si les longueurs des séquences sont des multiples de la capacité d'une piste. La troisième ligne indique les gains dans le cas général. Ils sont en moyenne de 10 % dans ce dernier cas, ce qui est assez faible si l'on considère les difficultés d'implémentation de cette amélioration. Par contre, ces difficultés se justifient amplement dans le premier cas puisque les gains sont de l'ordre de 50 %.

8.3.5. INFLUENCE DU NOMBRE DE VOIES ET DE L'ALLOCATION DE LA MEMOIRE CENTRALE

Pour mettre en évidence l'influence respective de ces deux facteurs sur les performances, nous allons comparer les performances des stratégies proposées à celles de stratégies non optimales. Pour mettre en évidence l'influence de l'allocation de la mémoire centrale, nous allons comparer les performances de la méthode d'allocation proposée en 8.1. avec la méthode la plus simple possible, celle qui consiste à allouer des tampons de même taille à toutes les monotonies. Pour mettre en évidence l'influence du nombre de voies, nous allons comparer les performances des stratégies optimales proposées avec les stratgies où le nombre de voies des fusions élémentaires est limité à trois.

Etude des performances si les tampons en mémoire centrale ont tous même taille

Dans l'introduction (formule (5)) nous avons calculé le temps de fusion élémentaire dans ce cas. Il est égal à :

$$\begin{cases} F_1(a_o,d) = \varphi_1(d).a_o \\ \varphi_1(d) = 2(Tt + Ta/M) + 2(Ta/M)d \end{cases}$$

Ce temps de fusion élémentaire définit une fonction de coût sur les arbres du même type que celle de la formule (13). Les résultats exposés dans ce chapitre restent donc valables si l'on remplace la fonction φ par la fonction φ_1 précédente. En particulier, on peut définir un temps de tri-fusion k_1 qui mesure les performances des stratégies optimales correspondant à cette nouvelle méthode d'allocation :

$$k_1 = \underset{d\geq 2}{\text{Min}}\ \varphi_1(d)/\text{Log } d$$

En utilisant les notations (14), cela s'écrit :

$$k_1 = \alpha.\underset{d\geq 2}{\text{Min}}\ (\lambda + 2 + 2d)/\text{Log } d$$

Le temps de tri-fusion k_1 est évidemment supérieur au temps de tri-fusion k qui correspond à la méthode d'allocation proposée au paragraphe I

$$k = \alpha.\underset{d\geq 2}{\text{Min}}\ (\lambda + (1+\sqrt{d})^2)/\text{Log } d$$

La perte de performance quand on utilise la méthode simple d'allocation est égale à $(k_1 - k)/k$. Elle ne dépend donc que de λ. Un calcul simple montre qu'elle est de l'ordre de 12 % pour les petites valeurs de λ et qu'elle ne dépasse pas 18 % quand λ augmente. Cette perte de performance est relativement faible. On peut donc considérer que le choix de la méthode d'allocation de la mémoire centrale n'a pas une grande influence sur les performances. Il n'en va pas de même en ce qui concerne le choix du nombre de voies, comme nous le montrons ci-dessous.

Influence du nombre de voies des fusions élémentaires

Nous avons indiqué au paragraphe 2 qu'à chaque configuration correspondait un nombre de voies privilégié : le degré-limite. D'une part, le nombre de voies de fusion dans les stratégies optimales est en moyenne égal à ce degré-limite. D'autre part, si le nombre de voies des fusions élémentaires est toujours égal au degré-limite, on obtient de bonnes stratégies de tri-fusion (cf. algorithme de Huffman exposé en 6.7 ; cf. heuristique du degré-limite si les monotonies initiales ont même taille). Or, une légende tenace veut que les bonnes stratégies de tri-fusion soient obtenues avec des fusions élémentaires de degré trois. Nous allons montrer que c'est complètement faux.

Appelons $C_3(a_1,\ldots,a_n)$ le coût minimum des arbres de fusion quand le nombre de voies est limité à trois. On peut montrer sans difficulté que le coût $C_3(a_1,\ldots,a_n)$ vérifie la formule (22) avec a = 3 et $k = k_3 = \varphi(3)/\text{Log } 3$. La constante k_3 mesure donc les performances quand le nombre de voies est limité à trois. Le tableau suivant indique le rapport k_3/k entre le temps de tri-fusion k_3 de la méthode à degré trois et le temps de tri-fusion k de la méthode exposée dans ce chapitre :

M =	50	200	500	1000
TAMBOUR	1.3	1.8	2.3	2.7
DIAD-grande vitesse	1.1	1.3	1.6	1.9
DIAD-moyenne vitesse	1.8	2.6	3.2	3.7
MD-100	1.2	1.5	1.9	2.2
MD-50	1.3	1.8	2.3	2.7
DIMAS	1.2	1.5	1.9	2.3

TABLEAU 10

Temps de tri-fusion pour les statégies à degré 3.

Pour les petites tailles mémoires, le rapport k_3/k est de l'ordre de 1.2. Pour M = 200 K-octets, il est de l'ordre de 1.5. Si M dépasse 500 K-octets il est de l'ordre de 2 ou 3. D'une manière générale, la méthode à degré

trois est donc très mauvaise puisqu'elle multiplie les temps de fusion par un facteur variant entre 1.5 et 3. A contrario, on en déduit que le choix du nombre de voies des fusions élémentaires a une influence déterminante sur les temps de fusion.

8.4. INFLUENCE DU TEMPS DE TRAITEMENT DE L'ORDINATEUR SUR LES PERFORMANCES.

Nous allons examiner comment sont modifiées les stratégies optimales ainsi que les temps de fusion quand le temps Tm de traitement de l'ordinateur n'est pas négligeable devant le temps d'entrée/sortie. Nous avons vu en 7.5 que le temps de fusion élémentaire est alors égal à :

$$F = 2.T't.a_o + T'a \sum_{o\leq i\leq d} \lceil a_i/b_i \rceil + c.a_o.\text{Log } d \qquad (28)$$

avec :

$$\begin{cases} T't = Tt + \frac{1}{2}(Mt + c_1/L) \\ T'a = Ta + Ma \\ c = c_2/(L.\text{Log } 2) \end{cases} \qquad (29)$$

Rappelons que $a_1,\ldots,a_d$ sont les tailles des monotonies entrantes, a_o est la taille de la monotonie résultante, et $b_o,\ldots,b_d$ les tailles des tampons correspondants en mémoire centrale. Par ailleurs, c_1 et c_2 sont des constantes dépendant des performances de l'ordinateur et du temps de comparaison de deux clés ; Mt est le temps de transfert en mémoire centrale ; Ma est le délai d'activation des entrées/sorties.

On peut remarquer que, pour une allocation mémoire donnée, le nombre total d'accès est le même que dans le cas où le temps de traitement de l'ordinateur est négligeable. (voir formue 8.(1)). Tous les résultats du paragraphe 8.1 sont donc valables dans le cas présent (allocation de la mémoire centrale, ajustement de la taille des tampons, validité de la modélisation). D'après la formule 8.(4), le temps de fusion élémentaire est donc égal à :

$$F'(a_o,d) = \varphi'(d).a_o$$

$$\varphi'(d) = 2\,T't + \frac{T'a}{M}(1+\sqrt{d})^2 + c.\text{Log } d \qquad (30)$$

La fonction de coût $F'(a_o,d)$ étant du type étudié dans la partie théorique, tous les résultats exposés dans ce chapitre restent valables. En particulier, on peut associer à cette fonction de coût un temps de tri-fusion k' qui mesure les performances des stratégies optimales correspondantes. On peut également lui associer un degré-limite a', nombre de voies de fusion moyen de ces stratégies. Les constantes k' et a' sont solution de :

$$k' = \frac{\varphi'(a')}{\text{Log } a'} = \min_{d\geq 2} \frac{\varphi'(d)}{\text{Log } d} = c + \min_{d\geq 2} \left[\frac{2\,T't + (T'a/M)(1+\sqrt{d})^2}{\text{Log } d} \right] \qquad (31)$$

On voit que la valeur du degré-limite est indépendante de la constante c. Le calcul du degré-limite s'effectue donc comme en 8.3.2 : il suffit de calculer $\lambda' = 2\,T't.M/\,T'a$ et de se reporter au tableau 4. On en déduit k' par (31).

Comme dans le cas classique, le nombre de voies de fusion dans les stratégies optimales est borné. Le calcul du degré maximum est analogue : il suffit de remplacer F par F' dans la formule (15). Là encore, la valeur de c n'a que peu d'influence : si les autres paramètres sont constants, le degré maximum n'augmente que très lentement avec c. En résumé, on peut donc dire que la valeur de c n'a pratiquement aucune influence sur la forme des arbres optimaux.

Que deviennent les temps de fusion quand le temps de traitement de l'ordinateur n'est pas négligeable ? Considérons l'exemple du paragraphe 7.5 et supposons que l'utilisateur programme lui-même les entrées/sorties (c'est-à-dire Ma = 0) ; rappelons que l'ordinateur est un IRIS-80 et que les clés à comparer sont des nombres. Afin que le temps de tri interne soit non négligeable, nous supposons d'autre part que les enregistrements ont pour longueur une centaine d'octets (c'est-à-dire L = 0.1). D'après les formules 7.(5) et 7.(7), les constantes de la formule (30) sont alors égales à :

$$T't = Tt + 0.3 \qquad T'a = Ta \qquad c = 0.15$$

Le tableau suivant indique l'augmentation en pourcentage des temps de tri-fusion par rapport au cas où le temps de traitement de l'ordinateur est négligeable (c'est-à-dire par rapport aux temps de tri-fusion de la figure 7). La deuxième ligne indique successivement le degré-limite a' dans le cas présent et le degré-limite a du tableau 5.
On constate sur ce tableau que plus la configuration est rapide plus les augmentations des temps de tri-fusion sont grandes. Si l'augmentation est pratiquement négligeable pour le DIMAS, elle atteint 50 % à 60 % pour le tambour et le DIAD grande vitesse.

M =	50	200	500	1000
TAMBOUR	21 % 10 9	36 % 22 18	45 % 43 34	52 % 73 57
DIAD grande vitesse	15 % 6 6	36 % 11 9	52 % 19 14	62 % 30 22
DIAD moyenne vitesse	4 % 18 18	6 % 48 46	7 % 97 94	8 % 170 164
MD-100	7 % 7 6	14 % 12 11	19 % 22 19	23 % 35 31
MD-50	4 % 9 8	7 % 18 18	8 % 34 33	11 % 58 55
DIMAS	2 % 7 7	5 % 13 12	7 % 22 21	8 % 36 34

TABLEAU 11

Influence du temps de traitement de l'ordinateur sur les performances et la forme des stratégies optimales.

Pour la taille mémoire, on constate que l'augmentation varie du simple au double entre M = 50 et M = 1000. En ce qui concerne le temps de transfert les plus grandes augmentations sont enregistrées pour le DIAD grande vitesse, disque qui a la plus grande cadence de transfert. En ce qui concerne le nombre de voies moyen, on peut dire qu'il est relativement peu sensible au temps de traitement de l'ordinateur.

En conclusion, ce tableau met en évidence qu'il est indispensable d'évaluer précisément le temps de traitement de l'ordinateur si l'on veut trouver les bonnes stratégies optimales et calculer des temps de fusion qui aient un sens.

CHAPITRE 9

TRI-FUSION AVEC OPTIMISATION DU MOUVEMENT DE BRAS

Les stratégies de tri-fusion proposées au chapitre précédent sont valables dans le cas général. Cependant, il est possible de les améliorer quand l'utilisateur dispose de disques à bras mobile et a la possibilité de choisir l'emplacement des monotonies sur disque. Dans ce chapitre, nous présentons des algorithmes d'optimisation du mouvement de bras quand l'utilisateur dispose d'un seul ou de deux disques. Nous étudions les stratégies optimales correspondantes, ainsi que l'amélioration des performances par rapport au cas du chapitre précédent où les mouvements du bras n'étaient pas optimisés.

Au chapitre 7, nous avons indiqué que les temps de déplacement du bras, et donc les temps d'accès, pouvaient être fortement réduits dans le cas d'un seul disque si l'on rapprochait les monotonies entrantes et la monotonie résultante sur le disque pour chaque fusion élémentaire. Au paragraphe 1, nous indiquons comment il faut placer les monotonies sur disque pour optimiser les mouvements de bras au cours d'une fusion élémentaire. Nous indiquons également comment allouer la mémoire centrale de manière à minimiser le temps de fusion élémentaire.

Au paragraphe 2, nous présentons des algorithmes d'optimisation du mouvement de bras pour un ou deux disques ; pour un arbre de fusion donné, ces algorithmes indiquent dans quel ordre effectuer les fusions élémentaires et comment placer les monotonies résultantes sur disque de manière à ce que les mouvements de bras soient optimisés au cours de chacune des fusions élémentaires de l'arbre de fusion.

Au paragraphe 3, nous étudions les stratégies optimales correspondant à ce modèle : ces stratégies diffèrent sensiblement de celles présentées au chapitre précédent dans le cas où le temps d'accès est constant. En particulier, il n'y a plus de nombre de voies privilégié correspondant à une configuration : le nombre de voies diminue en effet avec la taille des monotonies à fusionner. D'autre part, nous présentons une heuristique calculant des stratégies quasi-optimales à 1% près. Enfin, nous évaluons l'amélioration des performances apportée par l'optimisation du mouvement de bras : dans certains cas, les temps de fusion sont divisés par un facteur deux (pour un seul disque) ou trois (si l'on dispose de deux disques).

9.1. TEMPS DE FUSION ELEMENTAIRE SI LE MOUVEMENT DU BRAS EST OPTIMISE

Dans la suite, pour indiquer les placements des monotonies sur disque, nous représentons le disque par une droite : à chaque cylindre du disque correspond un segment sur cette droite, segment qui est divisé lui-même en sous-segments correspondant aux différentes pistes du cylindre. Par exemple, si l'utilisateur dispose d'un seul disque, le schéma suivant représente une fusion élémentaire contiguë, c'est-à-dire telle que la monotonie résultante m_o et les monotonies entrantes $m_1,\ldots,m_d$ soient contiguës sur le disque :

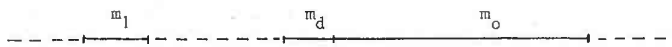


FIGURE 1
Placement optimum des monotonies au cours
d'une fusion élémentaire pour un seul disque

Peut-on considérer que ce placement optimise les mouvements du bras au cours d'une fusion élémentaire ? En fait, tous les placements contigus sont équivalents : on pourrait par exemple placer la monotonie résultante au milieu des monotonies entrantes. En effet, on peut considérer qu'au cours de la fusion élémentaire, les déplacements du bras du disque sont aléatoires à l'intérieur de l'espace disque de taille $2a_o$ occupé par les monotonies (a_o est la taille de la monotonie résultante m_o). Le temps moyen d'accès au cours de la fusion élémentaire est donc le même pour toutes les monotonies ; il est égal au délai rotationnel plus le temps moyen des déplacements aléatoires du bras à l'intérieur d'un espace disque de taille $2a_o$. Nous indiquons maintenant comment évaluer ce temps moyen.

9.1.1. TEMPS D'ACCES MOYEN A L'INTERIEUR D'UN ESPACE DISQUE DONNE

Nous calculons ce temps moyen pour un disque DIMAS[1] mais la méthode est valable pour un disque à têtes mobiles quelconque. La courbe suivante représent T, temps en millisecondes de déplacement du bras en fonction du nombre c de cylindres parcourus pour un disque DIMAS.

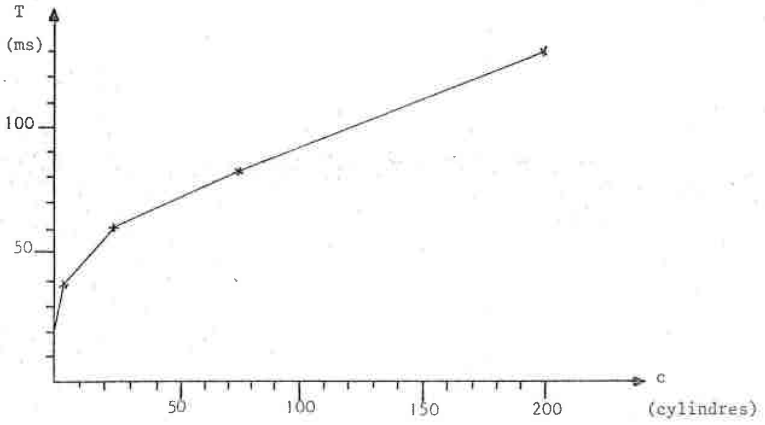


FIGURE 2

Temps de déplacement du bras

Remarquons que le temps de déplacement du bras n'est pas une fonction linéaire de la distance : en effet, pour effectuer un déplacement, le br prend une grande vitesse initiale, puis il ralentit au fur et à mesure qu'il se rapproche du cylindre à atteindre.

(1) Les courbes des figures 2 et 3 nous ont été fournies par la CII.

Calculons le temps moyen Tb(c) des déplacements aléatoires du bras à l'intérieur d'un espace disque de longueur égale à la capacité de c cylindres. Cet espace disque s'étend sur c+1 cylindres contigus. Le temps moyen Tb(c) est donc égal à la moyenne des temps de déplacement à l'intérieur de ces c+1 cylindres en considérant comme équiprobables tous les déplacements possibles. Il y a en tout $(c+1)^2$ déplacements possibles, qui se décomposent de la manière suivante :

nombre de déplacements	distance parcourue (en nombre de cylindres)
c+1	0
2c	1
2(c-1)	2
2(c+1-i)	i
2	c

Tous ces déplacements étant équiprobables et, compte tenu du fait que T(0) est nul, le temps moyen Tb(c) des déplacements aléatoires est égal à :

$$Tb(c) = \frac{2 \cdot \sum_{1 \leq i \leq c} (c+1-i)T(i)}{(c+1)^2}$$

La courbe suivant représente les temps moyens Tb(e) des déplacements aléatoires du bras à l'intérieur d'un espace disque de taille e ; cette courbe est calculée à partir de la courbe T(c) de la figure 2 à l'aide de la formule précédente. Les longueurs des espaces disques sont exprimées cette fois en K-octets (la capacité d'un cylindre du DIMAS est de 120 K-octets) :

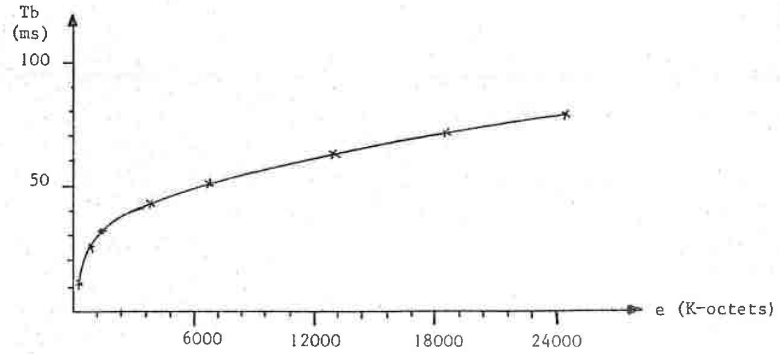


FIGURE 3

Temps moyens des déplacements aléatoires
à l'intérieur d'un espace disque

Nous précisons ci-dessous les temps moyens Tb pour certaines valeurs de e (exprimé en centaines de K-octets) :

e	1.2	6	12	36	60	120	180	240
Tb(e)	11	24	31	43	50	61	70	80

Quand la totalité du disque est utilisée, on retrouve bien la valeur moyenne de 80 ms du tableau 7.1.
Pour obtenir le temps d'accès moyen Ta(e) à l'intérieur d'un espace disque de taille e, il suffit d'ajouter le délai rotationnel moyen γ (égal à 12.5 ms pour le DIMAS) :

(1) $$Ta(e) = \gamma + Tb(e)$$

9.1.2. TEMPS DE FUSION ELEMENTAIRE. ALLOCATION DE LA MEMOIRE CENTRALE

Cas d'un seul disque

Les monotonies étant contiguës sur le disque, elles occupent un espace disque de taille $2a_o$ (a_o est la taille de la monotonie résultante). Le temps d'accès moyen au cours de la fusion élémentaire est donc égal à $Ta(2a_o)$. D'après la formule 7.(1), le temps de fusion élémentaire est alors égal à :

(2) $$F_1 = 2Tt.a_o + Ta(2a_o) \sum_{o \leq i < d} \lceil a_i/b_i \rceil + Tm$$

Nous ne traiterons pas dans ce chapitre le cas où il faut tenir compte du temps Tm de traitement en mémoire centrale : les techniques présentées au chapitre précédent restent en effet valables (voir 8.4). Si Tm est nul, l'expression du temps de fusion élémentaire est la même que dans le cas où le temps d'accès est constant (voir formule 8.(1)) : on a simplement remplacé Ta par $Ta(2a_o)$. Tous les résultats du paragraphe 8.1. sont donc valables dans le cas présent (allocation de la mémoire centrale, ajustement de la taille des tampons, validité de la modélisation). En particulier, on partagera la place M disponible en mémoire centrale en d tampons de taille b pour les monotonies entrantes et un tampon de taille B pour la monotonie résultante :

$$b = M \frac{1}{d + \sqrt{d}} \qquad B = M \frac{\sqrt{d}}{d + \sqrt{d}}$$

D'après la formule 8.(4), le temps de fusion élémentaire est égal à :

(3) $$F_1(a_o,d) = a_o \left[2Tt + \frac{Ta(2a_o)}{M} (1 + \sqrt{d})^2 \right]$$

D'après la formule 8.(10), si les longueurs des monotonies sont du même ordre de grandeur que M, le temps de fusion élémentaire est égal, à 20 % près, à :

(4) $$F_1(a_o,d) = 2Tt.a_o + Max \left[a_o \frac{Ta(2a_o)}{M} (1 + \sqrt{d})^2, (d+1)Ta(2a_o) \right]$$

Cas de deux disques

Si l'utilisateur dispose de deux disques identiques, le placement suivant des monotonies optimise les mouvements de bras au cours d'une fusion élémentaire :

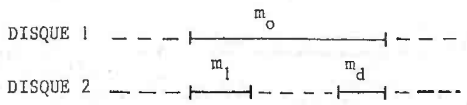


FIGURE 4

Placement optimum des monotonies au cours d'une fusion élémentaire pour deux disques

Montrons que tout autre placement des monotonies est moins bon. Appelons T1 (resp. T2) le temps d'accès moyen aux monotonies placées sur le premier disque (resp. sur le deuxième disque). Pour le placement de la figure 4, le temps d'accès T2 est égal à $Ta(a_o)$ puisque la somme des longueurs des monotonies entrantes est égale à la taille a_o de la monotonie résultante ; le temps d'accès T1 est égal au délai rotationnel γ puisque les sorties sur l'emplacement réservé à la monotonie résultante se feront séquentiellement (naturellement, le bras du disque 1 doit se déplacer de temps en temps d'un cylindre au cylindre voisin ; mais ce temps de déplacement est inclus dans le temps de transfert, comme nous l'avons indiqué en 7.2.1). Pour le placement précédent, les temps d'accès sont donc :

(5) $\qquad T1 = \gamma \qquad\qquad T2 = Ta(a_o)$

Considérons maintenant un placement quelconque des monotonies. Deux cas peuvent se présenter :

1) Une monotonie est isolée, par exemple sur le premier disque. Si L est la longueur de cette monotonie, les temps d'accès sont :

$$T1 = \gamma \qquad\qquad T2 = Ta(2a_o - L)$$

Comme L est plus petit que a_o, le temps d'accès T2 est supérieur à celui de la formule (5). Ce placement est donc moins bon que celui de la figure

2) Dans les autres cas, on peut supposer que la monotonie résultante est placée sur le deuxième disque. Si L est la somme des longueurs des monotonies placées sur ce disque, les temps d'accès sont :

$$T1 = Ta(2a_o - L) \qquad T2 = Ta(L)$$

D'après (1), T1 est supérieur à γ; d'autre part, comme L est supérieur à a_o, on a : $Ta(L) > Ta(a_o)$. Les temps d'accès étant supérieurs à ceux de la formule (5), ce placement est également moins bon que celui de la figure 4. ▨

Calculons le temps de fusion élémentaire correspondant au placement de la figure 4. Si l'on néglige le temps de traitement de l'ordinateur dans la formule 7.(1), il est égal à :

$$F_2 = 2Tt.a_o + \gamma.\left\lceil a_o/b_o \right\rceil + Ta(a_o).\sum_{1 \leqslant i \leqslant d} \left\lceil a_i/b_i \right\rceil$$

Le temps de fusion élémentaire F_2 pour deux disques diffère du temps F_1 pour un seul disque car le temps d'accès à la monotonie résultante diffère du temps d'accès aux monotonies entrantes. Néanmoins, les résultats du paragraphe 8.1. restent valables, avec quelques restrictions toutefois. Au cours de la fusion élémentaire, on partagera de la même manière la place M disponible en mémoire centrale en d tampons de tailles b' pour les monotonies entrantes et un tampon de taille B' pour la monotonie résultante ; mais les tailles des tampons ne sont plus les mêmes que dans le cas précédent. En effet, si les tailles des monotonies sont grandes par rapport à M, le temps de fusion F_2 est égal à :

$$F_2 = 2Tt.a_o + a_o\left[\gamma/B' + Ta(a_o)/b'\right]$$

On vérifie facilement que F_2 est minimum si :

$$b' = M/(d + \sqrt{d}/\delta) \qquad B' = M/(1 + \sqrt{d}.\delta) \qquad \delta = \sqrt{\frac{Ta(a_o)}{\gamma}}$$

On peut remarquer que b' est supérieur à b, et B' inférieur a B (puisque $\delta > 1$) ; cela vient du fait que le temps d'accès à la monotonie résultante est plus faible que le temps d'accès aux monotonies entrantes. Pour la même raison, si l'ajustement de la taille des tampons est nécessaire, il suffit de se limiter aux tampons d'entrée.

Le temps de fusion correspondant est égal à :

$$(6) \qquad F_2(a_o,d) = a_o \cdot \left[2Tt + \frac{1}{M}\left(\sqrt{\gamma} + \sqrt{d}\ \sqrt{Ta(a_o)}\right)^2 \right]$$

Si les longueurs des monotonies sont du même ordre de grandeur que M, le temps de fusion élémentaire est égal, à 20 % près, à :

$$(7) \qquad F_2(a_o,d) = 2Tt.a_o + Max\left[\frac{a_o}{M}\left(\sqrt{\gamma} + \sqrt{d}\sqrt{Ta(a_o)}\right)^2,\ \gamma + d.Ta(a_o) \right]$$

9.2. ALGORITHMES D'OPTIMISATION DU MOUVEMENT DE BRAS

Pour que les temps de fusions élémentaires calculés ci-dessus aient un sens, encore faut-il qu'il soit possible d'organiser le tri-fusion de manière à ce que la contiguïté de chaque fusion élémentaire soit assurée. Nous allons montrer que c'est possible, pour un ou deux disques, quel que soit l'arbre de fusion. Plus précisément, nous présentons des algorithmes qui, pour un arbre de fusion donné, indiquent dans quel ordre effectuer les fusions élémentaires et comment placer les monotonies résultantes sur disque pour assurer la contiguïté de presque toutes les fusions. Nous disons "presque toutes", car, pour les fusions initiales, c'est-à-dire celles ou interviennent des monotonies initiales, il n'est évidemment pas possible d'assurer la contiguïté si les monotonies initiales sont placées au départ n'importe où sur le disque. Nous envisagerons plusieurs cas pour les monotonies initiales :

Cas 1 : Elles sont placées sans ordre sur disque avant le début du tri-fusion.

Cas 2 : Elles sont contiguës sur disque et placées dans l'ordre, de gauche à droite, des feuilles correspondantes de l'arbre de fusion ; notons que cette dernière condition est automatiquement réalisée si les monotonies initiales ont même taille.

Cas 3 : Les monotonies initiales sont générées par l'utilisateur avant le début du processus de tri-fusion à partir du fichier initial à trier.

Cas 4 : Comme le cas 3, sauf que les phases de génération peuvent alterner avec les phases de fusion élémentaire au cours du processus de tri-fusion.

9.2.1. ALGORITHME D'OPTIMISATION DU MOUVEMENT DE BRAS POUR UN SEUL DISQUE

La procédure suivante permet d'effectuer le tri-fusion correspondant à un arbre de fusion donné en assurant la contiguïté des fusions élémentaires selon le schéma de la figure 1. Cette procédure, appelée TRIFUSION1 , a pour argument un noeud M d'un arbre de fusion ; elle effectue le tri-fusion correspondant au sous-arbre de sommet M en optimisant les mouvements de bras et place la monotonie résultante correspondant à M à l'adresse disque désignée par la variable p à l'appel de la procédure (dans la suite, M désignera indifféremment un noeud de l'arbre ou la monotonie résultante correspondante) ; au retour de la procédure, la variable p pointe sur la fin de l'emplacement réservé à M. Le programme suivant effectue donc le tri-fusion correspondant à un arbre T de fusion :

```
{ p: = adresse disque où doit être placé le fichier tri résultant
{ TRIFUSION1(sommet de T) ;
```

La procédure TRIFUSION1 s'écrit :

```
procédure TRIFUSION1(M) ;
début
    p: = p + taille (M) ;
    pour chaque F ∈ fils (M) faire
        si F ≠ feuille alors TRIFUSION1(F)  sinon GEN(F) ;
    si M = noeud initial alors FUSELEM1(M)  sinon FUSINIT1(M) ;
fin
```

Rappelons que les noeuds initiaux sont ceux dont un fils au moins est une feuille ; les noeuds initiaux correspondent donc aux fusions initiales, et les feuilles aux monotonies initiales. Cette procédure utilise les primitives suivantes :

FUSELEM1(M) : cette primitive effectue la fusion élémentaire des monotonies correspondant aux fils de M ; à l'appel de cette primitive, les monotonies entrantes sont à gauche de l'adresse B pointée par p ; au retour, p pointe sur A, c'est-à-dire sur la fin de la monotonie résultante M :

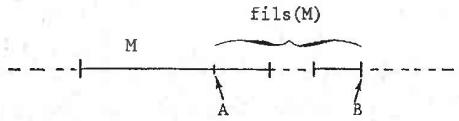


FIGURE 5

GEN(M) : cette primitive a pour argument une monotonie initiale M; elle a plusieurs actions possibles :

1) Dans le cas 4 (les phases de génération peuvent alterner avec les phases de fusion élémentaire), cette primitive génère la monotonie M à l'emplacement p sur disque.

2) Dans les autres cas, il y a deux possibilités : soit recopier la monotonie M à l'emplacement p, soit laisser la monotonie à l'endroit où elle est.

Dans les cas où M est placée à l'emplacement p, la variable p doit pointer sur la fin de cet emplacement au retour de GEN.

FUSINIT1(M) : cette primitive effectue la fusion élémentaire quand l'une au moins des monotonies entrantes est une monotonie initiale. Naturellement, si les monotonies initiales ont été générées ou recopiées par GEN, cette primitive à la même action que FUSELEM1. Sinon, seules les monotonies entrantes qui ne sont pas des monotonies initiales sont placées entre A et B (voir figure 5). Dans tous les cas, la variable p pointe sur B à l'appel de FUSINIT1, et sur A au retour.

Comment fonctionne la procédure TRIFUSION1 ? Elle réserve d'abord la place sur disque pour la monotonie résultante M. La procédure TRIFUSION1 est appelée ensuite récursivement sur chacun des fils de M de manière à placer sur disque les monotonies entrantes correspondant aux fils de M, selon le schéma de la figure 5. Enfin, la dernière instruction effectue la fusion élémentaire. Il est clair que cet algorithme assure la contiguïté de toutes les fusions, sauf peut être des fusions initiales.

Considérons l'exemple suivant où nous supposons pour simplifier que les fusions initiales sont contiguës. Soit à effectuer le tri-fusion correspondant à l'arbre de fusion suivant :

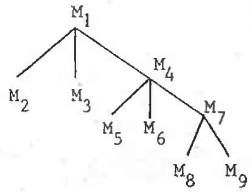
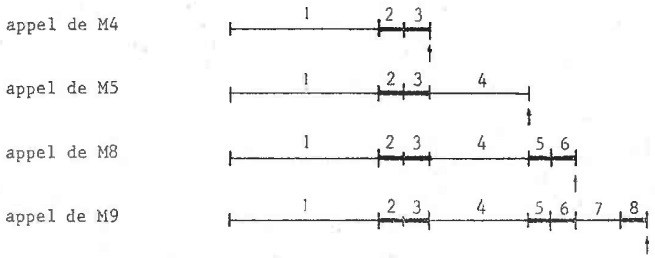


FIGURE 6

Nous indiquons ci-dessous l'état du disque à certaines phases du processus de tri-fusion (appel de M signifie appel de TRIFUSION1(M) ou de GEN(M) selon que M correspond à un noeud ou à une feuille) ; les monotonies sont représentées par leurs numéros :



Les traits pleins indiquent les monotonies effectivement présentes sur le disque ; dans le cas contraire, la place est simplement réservée. Les flèches indiquent les adresses disque pointées par la variable p.

Contiguïté des fusions initiales

D'après ce qui précède, la contiguïté des fusions initiales ne peut être assurée sans recopie que si les phases de génération peuvent alterner avec les phases de fusion (cas 4). En effet, on constate sur l'exemple précédent que l'algorithme réutilise le même espace disque pour différentes monotonies. Il n'est donc pas possible d'assurer la contiguïté des fusions initiales si les monotonies initiales sont générées avant le tri-fusion. Dans les autres cas, la méthode qui consiste à recopier les monotonies initiales pour assurer la contiguïté n'est généralement pas bonne (sauf si les monotonies initiales sont placées sur un périphérique plus lent : bande magnétique par exemple). Il est préférable de ne pas faire de recopies, quitte à effectuer des fusions initiales non contiguës.

9.2.2. ALGORITHME D'OPTIMISATION DU MOUVEMENT DE BRAS POUR DEUX DISQUES

La procédure suivante permet d'effectuer le tri-fusion correspondant à un arbre de fusion donné en assurant la contiguïté des fusions élémentaires selon le schéma de la figure 4. Les deux disques jouant un rôle symétrique, il y a deux types de fusion élémentaire, suivant que la monotonie résultante est sur le premier ou le deuxième disque. Dans la suite, le numéro i désignera l'un des disques et dif(i) l'autre disque.
La procédure TRIFUSION2 a pour arguments un noeud M d'un arbre de fusion et le numéro i de l'un des disques. Elle effectue le tri-fusion correspondant au sous-arbre de sommet M et place la monotonie résultante M à l'emplacement désigné par p(i) sur le disque i. Au retour de la procédure, la variable p(i) pointe sur la fin de l'emplacement réservé à M.
Le programme suivant effectue donc le tri-fusion correspondant à un arbre T de fusion :

```
{ p(1): = adresse disque où doit être placé le fichier trié
          résultant ;
  p(2): = début de l'espace de travail du deuxième disque ;
  TRIFUSION2(sommet de T,1) ;
```

La procédure TRIFUSION2 s'écrit :

```
procédure TRIFUSION2(M,i) ;
début
    pour chaque F ∈ fils(M) faire
        si F ≠ feuille alors TRIFUSION2(F,dif(i))
                                        sinon GEN(F,dif(i)) ;
    si M = noeud initial alors FUSELEM2(M,i)
                                        sinon FUSINIT2(M,i) ;
    p(i): = p(i) + taille(M) ;
fin
```

Cette procédure utilise les primitives suivantes :

FUSELEM2(M,i) : cette primitive effectue la fusion élémentaire des monotonies correspondant aux fils de M et place la monotonie résultante à l'adresse A pointée par p(i) sur le disque i ; à l'appel de la primitive, les monotonies entrantes sont à gauche de l'adresse B' pointée par p(dif(i)) sur le disque dif(i) ; au retour de la primitive, p(i) pointe sur B et p(dif(i)) sur A' :

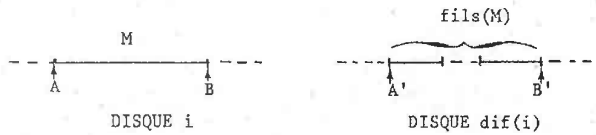
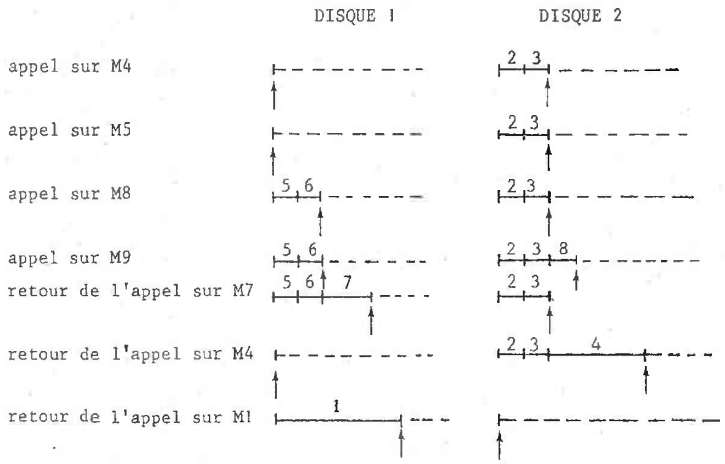


FIGURE 7

GEN(F,i) : : l'action de cette primitive est identique à celle de GEN(F), à ceci près que la monotonie initiale F peut être placée sur l'un ou l'autre disque, selon la valeur de i.

FUSINIT2(M,i): cette primitive est identique à FUSELEM2, avec les mêmes restrictions que pour la primitive FUSINIT1.

Pour expliquer comment fonctionne la procédure TRIFUSION2, nous reprenons l'exemple de la figure 6 en indiquant l'état des deux disques à certaines phases du tri-fusion :

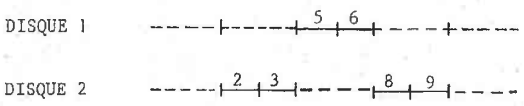


Les flèches indiquent les adresses disque pointées par p(1) et p(2).
Les montonies indiquées sont présentes sur disque puisqu'il n'y a pas de "réservation de place" dans ce cas.

Contiguïté des fusions initiales

Si nous sommes dans lescas 1 et 4 pour les monotonies initiales, les mêmes remarques s'appliquent que dans le cas d'un seul disque. Par contre dans le cas 3, il est toujours possible d'assurer la contiguïté des fusio

initiales. Il suffit de générer au départ ces monotonies à l'emplacement qu'elles doivent occuper au cours du tri-fusion. Dans l'exemple de la figure 6, le placement suivant assure la contiguïté des fusions initiales



On remarque que les monotonies initiales de profondeur impaire sont sur le deuxième disque et celles de profondeur paire sur le premier disque. En conséquence, dans le cas 2, c'est-à-dire quand les monotonies initiales sont contiguës sur un même disque et placées dans l'ordre des feuilles de l'arbre de fusion, il est possible d'assurer la contiguïté des fusions initiales si les feuilles de l'arbre sont toutes à la même profondeur. Dans le cas général, sauf coïncidence extraordinaire, les monotonies initiales ne sont pas dans l'ordre désiré. Par contre si elles ont toute même taille, cette condition est automatiquement réalisée. Il est donc intéressant dans ce cas de chercher des arbres de fusion dont toutes les feuilles ont même profondeur.
En résumé, la contiguïté des fusions initiales peut être assurée dans les cas suivants :

Cas	1	2	3	4
un disque	non	non	non	oui
deux disques	non	oui(*)	oui	oui

TABLEAU 1

Contiguïté des fusions initiales

Dans la suite, pour simplifier le calcul des temps de fusion, nous nous placerons toujours dans le cas où les fusions initiales sont contiguës.

(*) si les feuilles de l'arbre de fusion sont à la même profondeur

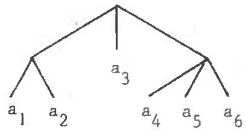
9.3. LES STRATEGIES OPTIMALES. COMPARAISON DES TEMPS DE FUSION

Nous avons vu que si la longueur a_o de la monotonie résultante est assez grande par rapport à la place M disponible en mémoire centrale, les temps des fusions élémentaires contiguës pour un ou deux disques sont égaux à :

$$(8) \qquad F_1(a_o,d) = a_o.\left[2Tt + \frac{Ta(2a_o)}{M}\,(1+\sqrt{d})^2\right]$$

$$(9) \qquad F_2(a_o,d) = a_o.\left[2Tt + \frac{1}{M}\left(\sqrt{\gamma} + \sqrt{d}\,\sqrt{Ta(a_o)}\right)^2\right]$$

Comme dans le cas où le temps d'accès est constant, les temps de fusion précédents définissent des fonctions de coût $\mathcal{C}_1(T)$ et $\mathcal{C}_2(T)$ sur les arbres de fusion. Soit par exemple l'arbre de fusion T suivant pour 6 monotonies initiales de longueurs $a_1,\dots a_6$:



Pour un seul disque, le temps total de fusion $\mathcal{C}_1(T)$ correspondant à cet arbre est égal à :

$$\mathcal{C}_1(T) = (a_1+a_2).F_1(a_1+a_2,2) + (a_4+a_5+a_6).F_1(a_4+a_5+a_6,3) + (a_1+..+a_6).F_1(a_1+..+a_6,3).$$

Pour deux disques, il suffit de remplacer F_1 par F_2 pour obtenir $\mathcal{C}_2(T)$. Trouver la stratégie optimale pour un n-uple $(a_1,\dots,a_n)$ revient donc à trouver l'arbre de fusion optimal qui minimise $\mathcal{C}_1(T)$ (resp. $\mathcal{C}_2(T)$) pour un disque (resp. deux disques) ; nous appellerons $C_1(a_1,\dots,a_n)$ (resp. $C_2(a_1,\dots,a_n)$) le coût optimal ; on définit de même $C_1(n,s)$ et $C_2(n,s)$ pour n monotonies de même longueur s.
Bien entendu, les stratégies et les coûts optimaux ne sont pas les mêmes pour un ou deux disques. Mais les propriétés des arbres optimaux correspondants sont assez voisines, si bien que nous traiterons les deux cas ensemble ; dans la suite $F(a_o,d)$ désignera indifféremment $F_1(a_o,d)$ ou $F_2(a_o,d)$; de même pour $\mathcal{C}$ et C.

9.3.1. CALCUL DES STRATEGIES OPTIMALES. HEURISTIQUES

Si les monotonies initiales sont de tailles différentes, les algorithmes et heuristiques présentés au chapitre 6, paragraphe 5 et 7 restent valables si l'on remplace les termes de la forme $\varphi(x).y$ par $F(y,x)$. Nous indiquons plus loin comment calculer la borne d_{max} sur le nombre de voies de fusion. Si les monotonies initiales ont même taille s, l'algorithme 2.1.4 est également valable si l'on remplace le terme $\varphi(d)n$ de l'instruction I4 par $F(ns,d)$. Mais attention ! si les monotonies initiales ont même taille, les heuristiques présentées au chapitre précédent en 8.2.3 ne sont plus valables, les arbres optimaux ayant des formes différentes. Nous présentons plus loin une très bonne heuristique dans ce cas.

9.3.2. FORME DES ARBRES OPTIMAUX. DEGRE MAXIMUM

Quelles sont les propriétés des stratégies optimales ainsi calculées ? Il est évident que le coût optimal reste proportionnel à la fonction de coût : si l'on multiplie F par une constante, la forme des arbres optimaux ne change pas ; seul le coût optimal est multiplié par cette constante. Mais, contrairement au cas où le temps d'accès est constant, le coût optimal n'est plus proportionnel au n-uple, c'est-à-dire aux longueurs des monotonies initiales ; en effet, le temps de fusion élémentaire n'est pas proportionnel à a_o à cause des termes $Ta(2a_o)$ et $Ta(a_o)$. En conséquence, la forme des arbres optimaux va dépendre de la taille des monotonies initiales. En particulier, pour n monotonies initiales de même taille s, la forme des arbres optimaux dépend non seulement de n, mais aussi de la longueur s des monotonies.

D'une manière générale, la plupart des propriétés indiquées au chapitre précédent ne sont plus valables (filtrage, nombre de voies privilégié d'une configuration, etc...). Les seules propriétés qui s'appliquent sont les suivantes : les contractionset les sous-arbres d'un arbre optimal sont optimaux, le nombre de voies des fusions élémentaires est borné, mais cette borne dépend maintenant de la longueur des monotonies.

Nombre de voies de fusion. Degré maximum

Dans une stratégie optimale, considérons une fusion élémentaire à d voies dont la longueur de la monotonie résultante est a_o. Comme en 8.2.2, formule (15), on montre que d et a_o vérifient :

$$\forall k \quad 2 \leqslant k \leqslant d-1 \qquad F(a_o,d) \leqslant F(a_o,d-k+1) + F(\tfrac{k}{d}\, a_o,k)$$

ce qui s'écrit :

$$\forall k \quad 2 \leqslant k \leqslant d-1 \qquad d.\left[f(a_o,d) - f(a_o,d-k+1)\right] \leqslant k\left[2Tt.M + f(\tfrac{k}{d}\, a_o,k)\right] \tag{10}$$

$$f(a_o,d) = \begin{cases} Ta(2a_o).(1+\sqrt{d})^2 & \text{pour un disque} \\ \left[\sqrt{Tt} + \sqrt{d}\,\sqrt{Ta(a_o)}\right]^2 & \text{pour deux disques} \end{cases} \tag{11, 12}$$

avec :

Comme dans le cas où le temps d'accès est constant, on peut montrer que d est borné par un entier d_{max1} (resp. d_{max2}) dans le cas d'un seul disque (resp. deux disques). Ces bornes dépendent d'une part de la configuration, c'est-à-dire du produit Tt.M, et d'autre part de la longueur a_o de la monotonie résultante par l'intermédiaire de la fonction Ta. Pour une configuration donnée, nous noterons donc ces bornes $d_{max1}(a_o)$ et $d_{max2}(a_o)$. Remarquons que, contrairement au cas où le temps d'accès est constant, il n'existe pas pour une configuration donnée de degré maximum valable quelles que soient les longueurs des monotonies. Nous savons seulement que dans les stratégies optimales, le nombre de voies de chaque fusion élémentaire est borné par un entier dépendant de la longueur de la monotonie résultante.

Comme dans le cas où le temps d'accès est constant, le degré maximum augmente avec le temps de transfert et la taille mémoire, et diminue avec le temps d'accès. Comme Ta est une fonction croissante de a_o, le degré maximum diminue avec la taille de la monotonie résultante.

En général, les fonctions d_{max1} et d_{max2} sont approximativement égales pour une configuration donnée : la fonction d_{max2} étant légèrement supérieure pour les valeurs courantes de a_o, Tt et M. Quand a_o augmente, on peut montrer que d_{max1} tend vers trois et que d_{max2} tend vers deux, et cela quelles que soient les valeurs de Tt et M. Ceci n'a qu'un intérêt

théorique car les valeurs correspondantes de a_o dépassent très largement la capacité de tous les disques connus ! Mais ce résultat nous montre qu'il n'y a pas de degré privilégié correspondant à une configuration, contrairement au cas où le temps d'accès est constant.
Il n'est donc pas possible d'indiquer des valeurs, même approximatives, pour les degrés des arbres optimaux correspondant à une configuration donnée. Tout dépend de la taille des monotonies initiales : plus elles sont grandes, et plus les degrés des arbres optimaux correspondants sont petits (et inversement). Cette propriété est également valable à l'intérieur d'un même arbre optimal : plus les noeuds sont proches du sommet de l'arbre, et plus les degrés diminuent, puisque la taille des monotonies résultantes augmente quand on se rapproche du sommet de l'arbre. Autrement dit, les degrés des noeuds des arbres optimaux augmentent avec la profondeur, comme nous le montrons en 9.3.2 quand les monotonies initiales ont même taille.
Les fonctions d_{max1} et d_{max2} indiquent des bornes sur le nombre de voies d'une fusion élémentaire seulement. Mais pour utiliser les algorithmes décrits plus haut, nous avons besoin d'une borne sur les degrés de tous les noeuds de l'arbre optimal correspondant à n monotonies initiales de longueurs $a_1,\ldots,a_n$. Appelons Ad la somme des longueurs des d plus petites monotonies initiales. La méthode la plus simple consiste à prendre comme borne $d_{max}(A2)$ (la fonction d_{max} désigne d_{max1} ou d_{max2} selon les cas) ; en effet, la taille des monotonies résultantes est toujours au moins égale à A2. Naturellement cette borne est généralement très large. Une méthode plus sophistiquée consiste à prendre comme borne le plus petit entier d tel que $d_{max}(Ad) \leqslant d$; si, dans l'arbre optimal correspondant, il existait un noeud de degré d' supérieur à d, la longueur a_o de la monotonie résultante correspondante serait plus grande que Ad ; on aurait donc : $d' \leqslant d_{max}(a_o) \leqslant d_{max}(Ad_o) \leqslant d$, ce qui est impossible.

9.3.3. MONOTONIES INITIALES DE MEME TAILLE

L'étude du cas où les monotonies initiales ont même taille permet de mieux saisir les propriétés des arbres optimaux dans le cas général. Nous allons considérer la même configuration qu'au chapitre précédent (disque DIMAS avec 20 K-octets en mémoire centrale) et étudier les propriétés des arbres optimaux correspondants dans le cas où l'on optimise les mouvements du bras pour un seul disque. Les résultats présentés sont valables pour d'autres configurations et s'appliquent également dans le cas de deux disques.

La forme et le coût des arbres optimaux dépendant de la longueur s des monotonies initiales, nous sommes obligés de choisir une valeur pour s. Nous prendrons s = 5 K-octets. Comme au chapitre précédent, nous étudions d'abord la valeur du degré au sommet des arbres optimaux en fonction du nombre n de monotonies initiales.

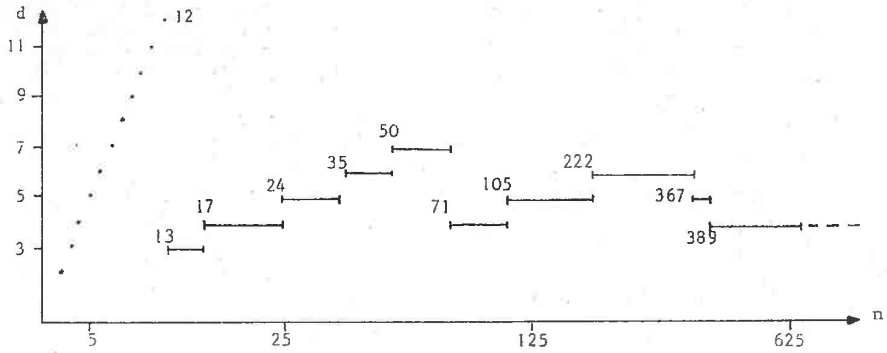


FIGURE 8

Degré au sommet des arbres optimaux pour n monotonies de taille 5 K-octets avec mouvement de bras optimisé pour un seul disque DIMAS (M= 20 K-octets)

Sur la courbe ci-dessus, on constate que l'arbre optimal est plat jusqu'à n = 12. Pour n = 13, le degré au sommet passe brusquement à 3 et augmente jusqu'à 7 pour n = 70. De n = 71 à n = 366, le degré augmente de 4 à 6. Comme dans le cas où le temps d'accès est constant (voir figure 8.2.), l'amplitude des oscillations diminue ensuite. Mais dans l'exemple de la figure 8.2, les degrés oscillent autour d'une valeur moyenne égale au degré-limite 5 et tendent vers cette valeur quand n augmente. Dans le cas présent au contraire, les valeurs extrêmes des oscillations diminuent simultanément de la manière suivante :

(2,12) (3,7) (4,6) (4,5) (3,5) (3,4) (3)

On peut montrer qu'à partir de un million de monotonies initiales, le degré au sommet des arbres optimaux est égal à trois. Ceci est valable quelles que soient la configuration et les longueurs des monotonies

initiales. On peut donc dire qu'il y a un "degré-limite" égal à trois. Dans le cas de deux disques, le "degré-limite" est égal à deux.
On peut remarquer que, contrairement au cas où le temps d'accès est constant, il n'y a jamais plusieurs degrés au sommet possibles pour une même valeur de n. Considérons par exemple l'arbre suivant qui est optimal pour n = 190 :

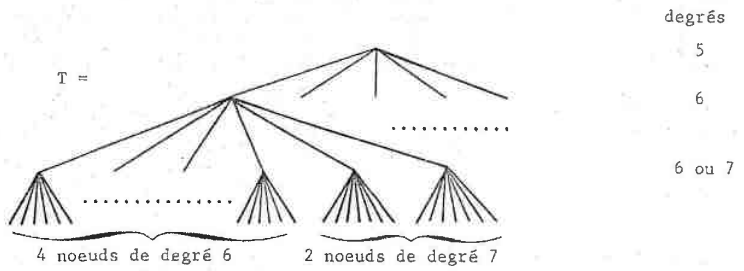


FIGURE 9

Arbre optimal pour n = 190

Le degré au sommet est égal à 5 ; les sous-arbres principaux ont chacun 38 feuilles et leur degré est égal à 6 ; sur la figure, nous n'avons représenté que le sous-arbre principal de gauche ; les autres sont identiques.
Pour n = 190, il y a un seul degré au sommet possible, qui est égal à 5; on peut même montrer que l'arbre optimal précédent est unique pour n = 190. D'une manière générale, pour la plupart des valeurs de n, l'arbre optimal est unique. Il y a deux raisons à cela :

1) Le filtrage change le coût des arbres (rappelons que le filtrage consiste à échanger le degré d'un noeud avec celui de ses fils si ceux-ci ont tous même degré ; voir 8.2.3). Par exemple, si nous filtrons le sommet de l'arbre T de la figure 9, nous obtenons un arbre T' de coût supérieur à celui de T. Cela est également valable pour deux disques : on a $\mathcal{C}_2(T') > \mathcal{C}_2(T)$, alors que pour le modèle à temps d'accès constant, le filtrage ne change pas le coût : $\mathcal{C}(T) = \mathcal{C}(T')$.

2) Les arbres optimaux sont aussi équilibrés que possible : Considérons l'arbre T précédent ; chacun des sous-arbres principaux a 6 noeuds pour fils, dont 4 de degré 6 et 2 de degré 7. Modifions les deux sous-arbres principaux de gauche de la manière suivante : attribuons 6 noeuds de degré 6 au premier ; attribuons au deuxième sous-arbre principal 2 noeuds de degré 6 et 4 noeuds de degré 7. Nous obtenons ainsi un arbre T' qui a également 190 feuilles, mais dont le coût est supérieur à celui de T. Cela est également valable pour deux disques : on a $\mathcal{C}_2(T') > \mathcal{C}_2(T)$, alors que cette opération ne change pas le coût pour le modèle à temps d'accès constant : $\mathcal{C}(T) = \mathcal{C}(T')$. Dans la suite nous appelerons cette propriété la propriété d'"équilibrage" des noeuds de plus grande profondeur.

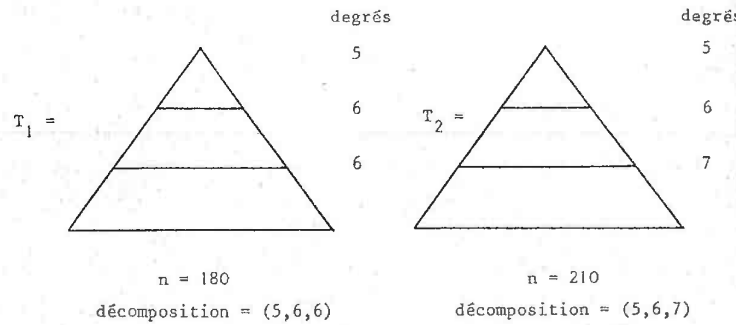
Arbres réguliers et pseudo-réguliers croissants optimaux

Définition 1 : Un arbre est dit pseudo-régulier croissant si :

1) toutes ses feuilles sont à la même profondeur p,
2) les noeuds situés à une même profondeur ont même degré, sauf à la plus grande profondeur (c'est-à-dire à la profondeur p-1),
3) les degrés des noeuds sont croissants (non strictement) avec la profondeur.
4) à la profondeur p-1, les noeuds n'ont que deux degrés possibles, d et d-1, qui sont répartis de la manière la plus équilibrée possible sur les noeuds de profondeur p-2.

Un arbre est dit régulier croissant si les noeuds de profondeur p ont aussi même degré.

D'une manière générale, si les monotonies initiales ont même taille, les arbres optimaux sont le plus souvent des arbres réguliers ou pseudo-réguliers croissants. Si nous reprenons l'exemple de la figure 8, les arbres réguliers croissants suivants sont optimaux pour n = 180 et n = 210 :



n = 180
décomposition = (5,6,6)

n = 210
décomposition = (5,6,7)

FIGURE 10

Le degré au sommet de T_2 est égal à 5 ; les 5 noeuds de profondeur 1 ont pour degré 6 et les 30 noeuds de profondeur 2 ont pour degré 7. Nous avons vu que l'ordre des degrés des différentes profondeurs a son importance : dans les décompositions, on lit de la gauche vers la droite les degrés par profondeur croissante. Nous indiquons ci-dessous les arbres réguliers optimaux dans l'exemple précédent pour n variant de 13 à 500 :

n	décomposition	n	décomposition
15	3,5	100	4,5,5
20	4,5	125	5,5,5
25	5,5	150	5,5,6
30	5,6	180	5,6,6
36	6,6	210	5,6,7
42	6,7	252	6,6,7
48	6,8	288	6,6,8
56	7,8	336	6,7,8
63	7,9	400	4,4,5,5
70	7,10	500	4,5,5,5
80	4,4,5		

Entre ces valeurs de n, les arbres optimaux sont le plus souvent pseudo-réguliers croissants. Par exemple, pour n = 190, l'arbre optimal T de la figure 9 se déduit facilement des arbres réguliers optimaux T1 et T2 pour n = 180 et n = 210 (voir figure 10) : il suffit de modifier les degrés des noeuds situés à la profondeur 2. Ceci est valable pour tous les arbres optimaux entre 180 et 210 : le degré au sommet est 5 ; les noeuds situés à la profondeur 1 ont pour degré 6 ; à la profondeur 2 il y a q noeuds de degré 7 et r noeuds de degré 6, ces entiers étant solution de :

$$q + r = 30 \qquad 7q + 6r = n$$

Mais attention, il ne faut pas oublier de placer à la profondeur 2 les noeuds de degré 6 ou 7 de la manière la plus équilibrée possible. Bien entendu, les arbres optimaux ne sont pas toujours des arbres pseudo-réguliers ou réguliers croissants. Mais l'heuristique qui consiste à choisir pour n donné l'arbre pseudo-régulier croissant de coût minimum donne d'excellents résultats : dans les cas où elle ne calcule pas l'arbre optimal, la différence relative avec le coût optimal n'excède pas 1 %. D'autre part, dans les arbres calculés par cette heuristique, les feuilles sont toutes à la même profondeur. Cette propriété est très intéressante si l'utilisateur dispose de deux disques : les algorithmes de tri-fusion sont alors très simples et la contiguïté des fusions initiales est assurée dans tous les cas (voir 9.2.2).

9.3.4. COMPARAISON DES TEMPS DE FUSION OPTIMAUX

Pour calculer les temps de fusion optimaux $C_1(a_1,\ldots,a_n)$ et $C_2(a_1,\ldots,a_n)$ pour n monotonies initiales de tailles $a_1,\ldots,a_n$, il n'y a pas d'autre méthode que d'utiliser les algorithmes décrits précédemment. Il n'existe pas en effet de mesure de performance permettant de se faire une idée approximative des temps de fusion si l'on optimise les mouvement de bras avec un ou deux disques. Bien entendu, on a l'inégalité :

$$C_2(a_1,\ldots,a_n) < C_1(a_1,\ldots,a_n) < C(a_1,\ldots,a_n)$$

Nous allons comparer les temps de fusion optimaux pour des monotonies initiales de tailles égales quand l'utilisateur dispose d'un disque DIMAS. Pour ce disque, le temps de transfert est de 4.37 millisecondes par K-octets ; le temps d'accès moyen Ta(e) a été calculé en 9.1.1 (voir figure 3 et formule (1)).

Les courbes de la figure 11 représentent les temps de fusion en secondes pour n monotonies initiales de longueur 50 K-octets quand la place M disponible en mémoire centrale est de 300 K-octets. Rappelons que C(n,50) est le temps de fusion sans optimisation du mouvement de bras calculé au chapitre 8 avec un temps d'accès moyen de 92.5 ms. Les courbes $C_1(n,50)$ et $C_2(n,50)$ représentent respectivement les temps de fusion avec optimisation du mouvement de bras pour un ou deux disques. Le tableau annexe précise pour certaines valeurs de n les temps de fusion quand il n'y a pas optimisation du mouvement de bras, ainsi que les gains de temps quand il y a optimisation.

On constate que dans ce cas, le gain de temps par rapport à la méthode sans optimisation est relativement faible : de l'ordre de 17 % avec un seul disque ; de l'ordre de 30 % avec deux disques. Cela vient du fait que la taille des monotonies initiales est relativement grande. Si elle est plus petite, le gain peut être beaucoup plus important. Considérons en effet le cas où la longueur des monotonies initiales est égale à 5 K-octets. Supposons en outre que la place disponible en mémoire centrale soit de 20 K-octets (nous avons étudié en 9.3.3 la forme des arbres optimaux dans ce cas si l'utilisateur optimise les mouvements de bras avec un seul disque). On constate sur la figure 12 que les gains par rapport à la méthode sans optimisation sont très intéressants : on gagne en moyenne un facteur deux dans le cas d'un seul disque, et un facteur trois pour deux disques. On remarque également que les gains diminuent avec le nombre de monotonies : en effet, la taille des monotonies résultantes augmente avec le nombre de monotonies et les temps d'accès moyens augmentent en proportion.

En résumé, on peut dire que l'optimisation du mouvement de bras n'a d'intérêt que si les tailles des monotonies initiales sont petites par rapport à la capacité du disque, et si les temps moyens de déplacement du bras sont importants : si l'on utilise un disque MD-50 ou MD-100, les gains sont beaucoup plus faibles qu'avec un DIMAS, car les temps moyens de déplacement du bras pour ces disques (respectivement 35 et 30 millisecondes) sont nettement moins importants que celui du DIMAS (80 millisecondes ; voir tableau 7.1).

temps de fusion (en secondes)

n =	50	100	200	300	400	500
C	64	141	314	509	719	943
C/C_1	1.16	1.17	1.17	1.16	1.16	1.15
C/C_2	1.34	1.29	1.31	1.32	1.32	1.31

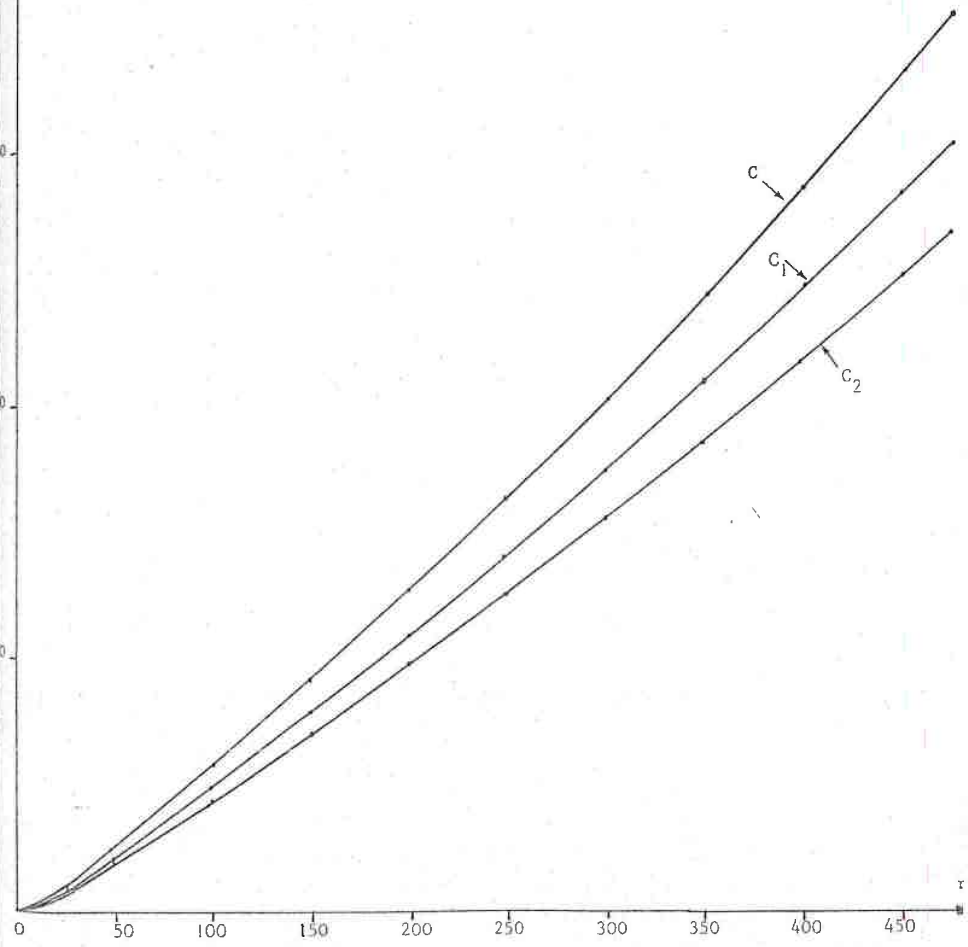
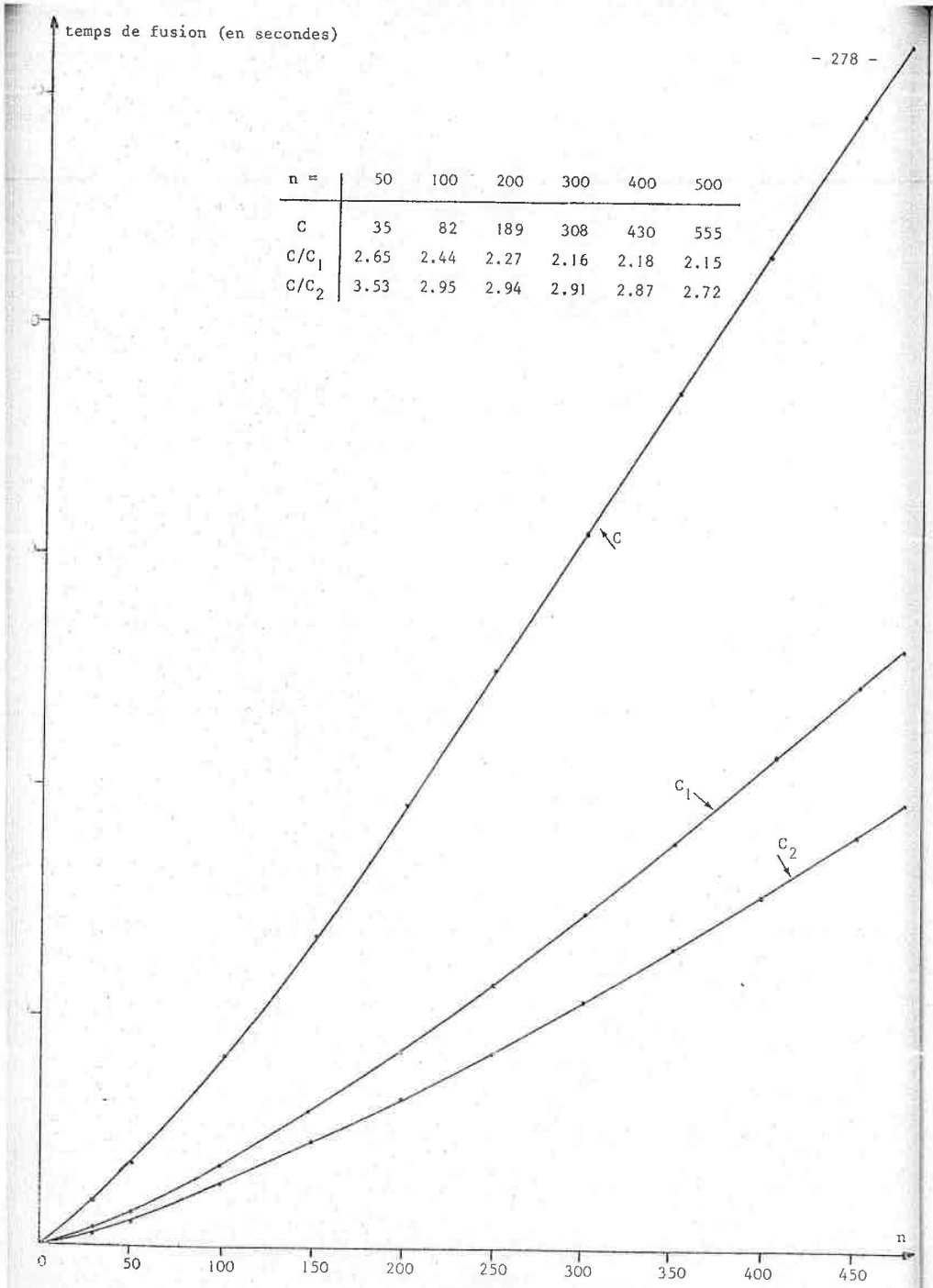


FIGURE 11

Comparaison des temps de fusion pour n monotonies de taille 50K (M=300)



n =	50	100	200	300	400	500
C	35	82	189	308	430	555
C/C_1	2.65	2.44	2.27	2.16	2.18	2.15
C/C_2	3.53	2.95	2.94	2.91	2.87	2.72

FIGURE 12

Comparaison des temps de fusion pour n monotonies de taille 5K (M=20)

ANNEXE 1

DEMONSTRATION DU LEMME 4.6

Le lemme 6 du chapitre 4 sert à montrer que le coût optimal C(n) possède une certaine régularité à partir d'un certain rang. On en déduit une formule relativement simple de calcul des arbres optimaux et de leurs coûts.

Nous étudions donc la suite de fonctions g_p suivantes et montrons que si p est plus grand qu'un certain entier p', g_p se déduit facilement de $g_{p'}$.

Définition 1 : Nous définissons la suite de fonction g_p suivantes pour p entier positif ou nul :

Soient des entiers positifs a, L, ℓ, ℓ' ($a \geq 3$) et la fonction g_o suivante :

1) g_o est une fonction bornée quelconque de W_o dans les réels positifs ou nuls:

W_o = ensemble des entiers relatifs de $\left[-\ell\ ,\ L+\ell'\right]$

g_o vérifie :

(1) $g_o(0) = g_o(L) = 0$

(2) pour $m < 0$ ou $m > L$, $g_o(m) > 0$

2) pour $p \geq 1$, g_p est la fonction de W_p dans les réels positifs ou nuls définie par :

W_p = ensemble des entiers de $\left[0, L.a^p\right]$

(3)
$$\begin{cases} g_p(n) = \text{Min} \sum_{1 \leq i \leq a^p} g_o(m_i) \\ \sum_{1 \leq i \leq a^p} m_i = n \end{cases}$$

Définition 2 : Nous définissons les deux sous-ensembles de W_o suivants :

$$Z = \{m \in W_o \mid m \neq 0,\ m \neq L,\ g_o(m) = 0\}$$

$$D = \{m \in W_o \mid g_o(m) > 0\}$$

L'ensemble Z est l'ensemble des zéros de g_o. D'après le définition de g_o, on a :

(4) $$Z \subset]0,L[$$

Remarquons que les quatre sous-ensembles Z, D, {0}, {L} constituent une partition de W_o.

Définition 3 : Pour $m \in Z$, nous définissons les entiers p(m) et p'(m) suivants :

(5) $$p(m).m = p'(m).L = \text{P.P.C.M.}(m,L)$$

(P.P.C.M. signifie : "plus petit commun multiple")

Comme $m \in Z$, on a d'après (4) :

(6) $$0 < p'(m) < p(m)$$

Définition 4 : Nous appelons $\mathcal{P}_p(n)$ ou p-partition de n un ensemble de a^p entiers tel que :

$$n = \sum_{m \in \mathcal{P}_p(n)} m \qquad \text{avec } m \in W_o$$

Notons qu'un même entier peut être répété plusieurs fois dans $\mathcal{P}_p(n)$.

A toute p-partition $\mathcal{P}_p(n)$, nous faisons correspondre :

$S = \{\mathcal{P}_p(n) \cap Z\}$

$T = \{\mathcal{P}_p(n) \cap D\}$

u = nombre d'éléments de $\mathcal{P}_p(n)$ égaux à L

v = nombre d'éléments de $\mathcal{P}_p(n)$ égaux à 0

Notons que la p-parititon $\mathcal{P}_p(n)$ est entièrement décrite par le quadruplet (S,T,u,v). Dans la suite, nous écrirons $\mathcal{P}_p(n) = (S,T,u,v)$. Si nous posons

$$\mathcal{S}(S) = \sum_{m \in S} m \qquad \mathcal{S}(T) = \sum_{m \in T} m$$

on a évidemment :

(7) $$n = \mathcal{S}(S) + \mathcal{S}(T) + u.L$$

(8) $$a^p = |S| + |T| + u + v \qquad (|S| = \text{nombre d'éléments de S})$$

On vérifie facilement que les formules (7) et (8) sont des conditions nécessaires et suffisantes pour qu'un quadruplet (S,T,u,v) soit une p-partition de n.

L'ensemble S sera souvent décrit par une fonction N sur Z : N(m) est égal au nombre d'éléments de S égaux à m. On a donc :

(9) $$|S| = \sum_{m \in Z} N(m) \qquad \mathcal{S}(S) = \sum_{m \in Z} N(m)$$

Nous décrirons parfois une p-partition par un quadruplet (N,T,u,v).

Défintion 5 : Nous définissons le coût d'une p-partition $\mathcal{P}_p(n) = (S,T,u,v)$ par :

$$\mathcal{C}\left(\mathcal{P}_p(n)\right) = \sum_{m \in \mathcal{P}_p(n)} g_o(m) = \sum_{m \in T} g_o(m)$$

Nous appelons p-partition optimale pour n celle qui minimise ce coût. Deux p-partitions de n sont dites équivalentes si elles ont même coût.

D'après la définition de g_p, le coût de la p-partition optimale pour n est égale à $g_p(n)$.

Lemme 1 : A tout p-partition $\mathcal{P}_p(n) = (S,T,u,v)$ correspond une p-partition équivalente $\mathcal{P}'_p(n) = (S',T,u',v')$ telle que :

(11) $$|S'| \leqslant s_1 \quad \text{avec} \quad s_1 = \sum_{m \in Z} \left[p(m)-1 \right]$$

$\left(p(m) \text{ a été défini en 3}\right)$.

Preuve : Appelons N et N' les fonctions correspondant respectivement à S et S'. Nous allons définir N', u' et v' de la manière suivante :

• N'(m) est le reste de la division entière de N(m) par p(m) (voir définition 3). Plus précisément :

(12) $$N(m) = q(m).p(m) + N'(m)$$
$$0 \leqslant N'(m) \leqslant p(m) - 1$$

(13) • $$u' = u + \sum_{m \in Z} q(m).p'(m)$$

(14) • $$v' = v + \sum_{m \in Z} \left(p(m)-p'(m)\right).q(m)$$

En utilisant les formules (5), (6), (12), (13) et (14), on vérifie facilement que (N',T,u',v') est une p-partition de n. Comme T reste le même, $\mathcal{P}'_p(n)$ est équivalent à $\mathcal{P}_p(n)$. D'autre part, on sait que :

$$|S'| = \sum_{m \in Z} N'(m)$$

Comme $N'(m) \leqslant p(m)-1$, on en déduit que $|S'| \leqslant s_1$.

Lemme 2 : Pour $p \geqslant p'$, $n \in W_p$, $n' \in W_{p'}$, on a :

(15) (i) $g_p(n') \leqslant g_{p'}(n')$.

(16) (ii) $g_p(n) \leqslant$ (reste (n,L)

(iii) si $n \equiv 0 \bmod.(L)$ alors $g_p(n) = 0$

Preuve de (i) : Soit (S,T,u,v) une p'-partition optimale de n'. On vérifie facilement que $(S,T,u,v+a^p-a^{p'})$ est une p-partition de n' de même coût que la précédente. On en déduit (15).

Preuve de (ii) : Appelons q et r le quotient et le reste de la division de n par L. Soit (S,T,u,v) une p'-partition optimale de r. On vérifie facilement que :

$$(S,T,u+q,v+a^p-a^{p'}-q)$$

est une p-partion de n de même coût que la précédente. On en déduit (16).

Preuve de (iii) : D'après (ii), on a $g_p(n) \leqslant g_o(0) = 0$. On en déduit que $g_p(n) = 0$.

Lemme 3 : Si (S,T,u,v) est une p-partition optimale de n, on a :

$$|T| < \mu_1/\mu_2$$

où μ_1 est le maximum de g_o et μ_2 est le minimum non nul de g_o sur D.

Preuve : D'après (10), on a $g_p(n) = \sum_{m \in T} g_o(m)$. On en déduit que :

(17) $$g_p(n) \geqslant |T|\mu_2$$

D'après le lemme précédent, on sait que $g_p(n) \leqslant g_o(\text{reste } (n,L))$. On en déduit que :

$$g_p(n) \leqslant \mu_1$$

D'après cette inégalité et l'inégalité (17), on a bien $|T| \leqslant \mu_1/\mu_2$.

Lemme 4 : Il existe toujours une p-partition optimale (S,T,u,v) de n telle que :

(18) $u \geq n/L - R$

(19) $v \geq a^p - n/L - R$

(20) avec : $R = \left\lceil \left(s_1 + \frac{\mu_1}{\mu_2} \right) \left(1 + \frac{\mathrm{Max}(\ell,\ell')}{L} \right) \right\rceil$

(s_1 est défini au lemme 1, μ_1 et μ_2 au lemme 3).

Preuve : D'après le lemme 1, il existe toujours une p-partition optimale (S,T,u,v) de n telle que ;

(21) $|S| \leq s_1$

Nous allons montrer que u et v vérifient (18) et (19). D'après cette dernière inégalité, et le lemme 3, on a :

(22) $|S| + |T| \leq s_1 + \mu_1/\mu_2$

Si nous posons $A = s_1 + \mu_1/\mu_2$, on en déduit que :

(23) $A\,\ell \leq \mathcal{S}(S) + \mathcal{S}(T) \leq A\,(L+\ell')$

D'après (7), on en déduit que :

$$n \leq A.(L+\ell') + u.L$$

ce qui s'écrit :

$$u \geq \frac{n}{L} - A.\left(1 + \frac{\ell'}{L}\right)$$

Cette dernière inégalité entraîne bien la minoration de u de l'inégalité (18). En utilisant les formules (7), (22), (23), on montre de même la minoration de v. ▨

Lemme 5 : Si les entiers n et n' vérifient :

(24) $\begin{cases} n, n' \in \left[RL,\ a^p L - RL \right] & \text{(R est défini par (20))} \\ n \equiv n' \mod.(L) \end{cases}$

alors : $g_p(n) = g_p(n')$

Preuve : Il suffit de faire la démonstration dans le cas où $n' = n+L$. Si n ou n' est égal à l'une des bornes, on a $n \equiv n' \equiv 0 \mod (L)$. D'après le lemme 2.(iii), on a bien $g_p(n) = g_p(n')$

Supposons maintenant que n et (n+L) sont différents des bornes. On en déduit que :

(25) $(n+L)/R - R > 0$

(26) $a^p - N/L - R > 0$

D'après cette dernière inégalité, et le lemme précédent, il existe une p-partition optimale (S,T,u,v) de n où v est strictement positif. On en déduit que (S,T,u+1,v-1) est une p-partition de (n+L) de même coût que la précédente. On en déduit que $g_p(n+L) \leq g_p(n)$. Un raisonnement analogue utilisant l'inégalité (25) montre que $g_p(n+L) \geq g_p(n)$. On en déduit que $g_p(n) = g_p(n+L)$. ▨

Lemme 6 : A partir d'un certain rang p', $g_p(n)$ se déduit facilement de $g_{p'}(n)$ si n est dans un voisinage borné de zéro ou a^p. Plus précisément : pour tout entier R', si p' vérifie

(27) $a^{p'} \geq R+R'$

alors pout tout entier $p \geq p'$, on a :

(28) $n \in \left[0, R'L\right] \Rightarrow g_p(n) = g_{p'}(n)$

(29) $n \in \left[a^p.L - R'L,\ a^pL \right] \Rightarrow g_p(n) = g_{p'}(a^{p'}.L+n-a^pL)$

Preuve : Nous ne le montrons que pour $n \in [0, R'L]$. La démonstration est analogue dans l'autre cas. D'après le lemme 4, il existe une p-partition optimale (S,T,u,v) de n, où v vérifie l'inégalité (19). Comme $n \leqslant R'L$, on en déduit que :

$$v \geqslant a^p - R' - R$$

ce qui entraîne, d'après (27) :

$$v - a^p + a^{p'} \geqslant 0$$

On en déduit que $(S,T,u,v-a^p+a^{p'})$ constitue une p'-partition possible de n de même coût que la p-partition optimale de n. On a donc $g_{p'}(n) \leqslant g_p(n)$. Comme p est plus grand que p', d'après le lemme 2.(i) on a aussi l'inégalité contraire. On en déduit bien que $g_p(n) = g_{p'}(n)$. ▨

Nous sommes maintenant en mesure de montrer le lemme 6 du chapitre 4.

Lemme 4.6 : Si l'entier Q vérifie $a^{Q-1} > R$, alors :

$$(\forall p)(\forall p') \qquad p > p' \geqslant Q$$

(30) $$g_p(n) = g_{p'}(n')$$

et n' se déduit de n par :

$$n \in \begin{cases} [0, \gamma] \\ ]\gamma, a^pL - \gamma[\\ [a^pL - \gamma, a^pL] \end{cases} \qquad n' = \begin{cases} n \\ j.\gamma + \text{reste}(n, \gamma) \\ a^{p'}L + n - a^pL \end{cases}$$

j et γ sont des entiers vérifiant :

$$\gamma = a^{p'-1}.L \qquad \text{et} \qquad 1 \leqslant j \leqslant a-2$$

Preuve : Posons :

$$R' = a^{p'-1}$$

On a donc :

$$\gamma = R'L$$

Comme $p' \geqslant Q$ et que $a^{Q-1} > R$, on vérifie facilement que :

$$a^{p'} \geqslant R+R'$$

Donc, si n est proche de zéro ou de $(a^p.L)$, l'égalité (30) découle directement du lemme 6. Reste le cas où $n \in]\gamma, a^pL - \gamma[$. Posons :

$$r = \text{rest}(n,L) \qquad 0 \leqslant r \leqslant L$$

Comme $p' \geqslant Q$ et que $a^{Q-1} > R$, on en déduit que $\gamma > RL$. On a donc :

$$RL \leqslant \gamma - L + r < n < a^p - RL$$

Comme $n \equiv \gamma - L + r \mod.(L)$, on a, d'après le lemme 5 :

(31) $$g_p(n) = g_p(\gamma - L + r)$$

Comme $\gamma - L + r < R'L$, on a, d'après le lemme 6 :

(32) $$g_p(\gamma - L + r) = g_{p'}(\gamma - L + r)$$

D'autre part, si $1 \leqslant j \leqslant a-2$, on a :

$$RL \leqslant \gamma - L + r < j\gamma + \text{reste}(n, \gamma) < a^pL - RL$$

Comme :

$$\gamma - L + r \equiv j\gamma + \text{reste}(n, \gamma) \mod.(L)$$

on a, d'après le lemme 5 :

(33) $$g_{p'}(\gamma - L + r) = g_{p'}\big(j\gamma + \text{reste}(n, \gamma)\big) \qquad (1 \leqslant j \leqslant a-2)$$

D'après les égalités (31), (32) et (33), on a bien :

$$g_p(n) = g_{p'}\big(j\gamma + \text{reste}(n, \gamma)\big) \qquad (1 \leqslant j \leqslant a-2)$$ ▨

ANNEXE 2

DEMONSTRATION DE LA PROPRIETE 8.1

Cette propriété donne des indications sur les stratégies optimales quand le temps d'accès est constant (voir chapitre 8).

propriété 8.1. : Soit un arbre optimal pour une fonction de coût du type

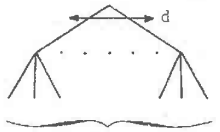
(1) $$\varphi(d) = \alpha\left[\lambda + (1+\sqrt{d})^2\right]$$

$$\alpha \in R^+ \quad \lambda \in R^+ \cup \{0\}$$

Si tous les fils d'un même noeud de degré d ont même degré d', alors la différence $|d-d'|$ n'excède pas un.

preuve : Par l'absurde : supposons que $|d-d'|$ soit au moins égal à deux. Les sous-arbres et contractions d'un arbre optimal étant optimaux (proposition 1.4), l'arbre T_1 suivant est optimal :

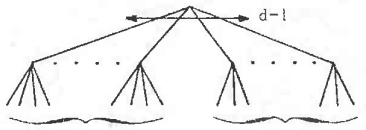
T_1 =



Appelons n le poids de T_1 et S sa suite de poids. Dans la suite nous supposons d > d' (si ce n'est pas le cas, il suffit de filtrer le sommet de T_1 pour obtenir un arbre optimal équivalent où d > d'). On a donc :

$$d > d'+1 \geqslant 3$$

Nous allons construire un arbre T_2 de coût strictement inférieur à T_1. Considérons l'arborescence suivante qui a le même nombre de feuilles que T_1 :



Sur cette arborescence, nous plaçons les poids de la suite S de la manière suivante : les plus petits poids de S sont attribués aux fils des noeuds de degré d'+1 (soit n_1 la somme de ces poids) ; les plus grands poids de S sont attribués aux feuilles restantes (soit n_2 la somme de ces poids). L'arbre aini obtenu a même suite de poids que T_1 et on a :

$$(2) \qquad n_2 \geqslant n \ \frac{d-d'-1}{d}$$

Comparons les coûts de T_1 et T_2 :

$$\mathcal{C}(T_1) = n\left[\varphi(d') + \varphi(d)\right]$$

$$\mathcal{C}(T_2) = n_1\left[\varphi(d'+1) + \varphi(d-1)\right] + n_2\left[\varphi(d') + \varphi(d-1)\right]$$

En posant $n_1 = n-n_2$, la différence des coûts s'écrit :

$$\mathcal{C}(T_1) - \mathcal{C}(T_2) = n_2\left[\varphi(d'+1) - \varphi(d')\right] + n\left[\varphi(d') + \varphi(d) - \varphi(d'+1) - \varphi(d-1)\right]$$

D'après (2), on en déduit :

$$(3) \qquad \mathcal{C}(T_1) - \mathcal{C}(T_2) \geq \frac{n}{d}\left[d\left[\varphi(d) - \varphi(d-1)\right] - (d'+1)\left[\varphi(d'+1) - \varphi(d')\right]\right]$$

Posons .

$$f(x) = x\left[\varphi(x) - \varphi(x-1)\right]$$

L'inégalité (3) devient :

$$(4) \qquad \mathcal{C}(T_1) - \mathcal{C}(T_2) \geqslant \frac{n}{d}\left[f(d) - f(d'+1)\right]$$

Un calcul simple montre que f(x) est strictement croissante quand $x \geqslant 3$. Comme par hypothèse $d > d'+1 \geqslant 3$, on en déduit que $f(d) > f(d'+1)$. D'après (4), le coût de T_1 est donc strictement supérieur au coût de T_2, ce qui contredit l'hypothèse d'optimalité de T_1.

ANNEXE 3

DEGRE-LIMITE EN FONCTION DE λ

Le degré-limite a est défini par l'inégalité suivante :

$\Psi(a-1) < \lambda < \Psi(a)$ $\qquad \lambda = 2.\ Tt.M/Ta$

d	ψ(d)	d	ψ(d)	d	ψ(d)	d	ψ(d)
4	0.15	24	55.78	44	135.37	64	220.64
5	2.12	25	62.32	45	139.46	65	225.08
6	4.29	26	65.90	46	143.57	66	229.52
7	6.51	27	69.52	47	147.70	67	233.99
8	9.05	28	73.16	48	151.85	68	238.46
9	11.52	29	75.54	49	156.02	69	242.95
10	14.29	30	80.56	50	160.20	70	247.46
11	17.05	31	84.30	51	164.43	71	251.99
12	19.90	32	88.07	52	169.64	72	256.50
13	22.82	33	91.87	53	172.88	73	261.03
14	25.81	34	95.70	54	177.15	74	265.62
15	28.87	35	99.56	55	181.41	75	270.14
16	31.99	36	103.44	56	185.72	76	274.73
17	35.17	37	107.35	57	190.03	77	279.34
18	38.40	38	111.28	58	194.37	78	283.93
19	41.69	39	115.24	59	198.71	79	288.55
20	45.02	40	119.22	60	203.06	80	293.19
21	48.39	41	123.22	61	207.45	81	297.83
22	51.82	42	127.25	62	211.82	82	302.47
23	55.28	43	131.30	63	216.23	83	307.14

ANNEXE 4

DEGRE-MAXIMUM EN FONCTION DE λ

Le degré-maximum d_{max} est défini par l'inégalité suivante :

$$f(d_{max}-1) < \lambda < f(d_{max}) \qquad \lambda = 2.\ Tt.M/Ta$$

d	f(d)	d	f(d)	d	f(d)	d	f(d)
8	0.56	29	18.07	50	35.74	71	55.96
9	1.43	30	18.94	51	37.65	72	56.88
10	2.19	31	19.80	52	38.56	73	57.81
11	2.95	32	20.67	53	39.48	74	58.73
12	3.71	33	21.54	54	40.39	75	59.66
13	4.48	34	22.40	55	41.30	76	60.58
14	5.32	35	23.29	56	42.22	77	61.50
15	6.16	36	24.19	57	43.13	78	62.43
16	6.99	37	25.09	58	44.04	79	63.35
17	7.83	38	25.98	59	44.95	80	64.27
18	8.66	39	26.88	60	45.86	81	65.19
19	9.49	40	27.76	61	46.77	82	66.12
20	10.32	41	28.67	62	47.68	83	67.04
21	11.15	42	29.57	63	48.59	84	67.96
22	11.97	43	30.46	64	49.50	85	68.89
23	12.83	44	31.36	65	50.41	86	69.82
24	13.70	45	32.25	66	51.33	87	70.76
25	14.56	46	33.14	67	52.25	88	71.69
26	15.43	47	34.03	68	53.18	89	72.63
27	16.32	48	34.92	69	54.11	90	73.56
28	17.20	49	35.82	70	55.03	91	74.50

d	f(d)	d	f(d)	d	f(d)	d	f(d)
92	75.43	122	103.55	152	131.89	182	160.41
93	76.37	123	104.49	153	132.84	183	161.37
94	77.30	124	105.43	154	133.78	184	162.32
95	78.23	125	106.37	155	134.73	185	163.28
96	79.17	126	107.30	156	135.68	186	164.24
97	80.10	127	108.24	157	136.64	187	165.19
98	81.03	128	109.18	158	137.59	188	166.15
99	81.96	129	110.12	159	138.54	189	167.10
100	82.90	130	111.07	160	139.49	190	168.06
101	83.83	131	112.02	161	140.44	191	169.02
102	84.76	132	112.96	162	141.40	192	169.97
103	85.69	133	113.91	163	142.35	193	170.93
104	86.62	134	114.86	164	143.30	194	171.88
105	87.55	135	115.81	165	144.25	195	172.84
106	88.49	136	116.75	166	145.20	196	173.79
107	89.43	137	117.70	167	146.15	197	174.75
108	90.37	138	118.65	168	147.11	198	175.71
109	91.32	139	119.60	169	148.06	199	176.66
110	92.26	140	120.54	170	149.01	200	177.62
111	93.20	141	121.49	171	149.96	201	178.57
112	94.14	142	122.43	172	150.91	202	179.53
113	95.08	143	123.38	173	151.86	203	180.48
114	96.02	144	124.33	174	152.81	204	181.44
115	96.96	145	125.27	175	153.76	205	182.39
116	97.91	146	126.22	176	154.71	206	183.34
117	98.85	147	127.17	177	155.66	207	184.30
118	99.79	148	128.11	178	156.61	208	185.25
119	100.73	149	129.06	179	157.56	209	186.21
120	101.67	150	130.00	180	158.51	210	187.16
121	102.61	151	130.95	181	159.46	211	188.12

d	f(d)	d	f(d)	d	f(d)	d	f(d)
212	189.07	242	217.83	272	246.68	302	275.59
213	190.03	243	218.79	273	247.64	303	276.55
214	190.99	244	219.75	274	248.60	304	277.52
215	191.95	245	220.71	275	249.56	305	278.48
216	192.91	246	221.67	276	250.52	306	279.44
217	193.87	247	222.63	277	251.48	307	280.41
218	194.83	248	223.59	278	252.44	308	281.37
219	195.79	249	224.56	279	253.41	309	282.34
220	196.74	250	225.52	280	254.37	310	283.30
221	197.70	251	226.48	281	255.33	311	284.26
222	198.66	252	227.44	282	256.30	312	285.23
223	199.62	253	228.41	283	257.26	313	286.19
224	200.58	254	229.37	284	258.23	314	287.15
225	201.54	255	230.33	285	259.19	315	288.12
226	202.50	256	231.29	286	260.16	316	289.08
227	203.46	257	232.25	287	261.12	317	290.05
228	204.42	258	233.22	288	262.09	318	291.02
229	205.38	259	234.18	289	263.05	319	291.98
230	206.33	260	235.14	290	264.02	320	292.95
231	207.29	261	236.10	291	264.98	321	293.92
232	208.25	262	237.06	292	265.95	322	294.88
233	209.21	263	238.03	293	266.91	323	295.85
234	210.17	264	238.99	294	267.87	324	296.82
235	211.13	265	239.95	295	268.84	325	297.78
236	212.08	266	240.91	296	269.80	326	298.75
237	213.04	267	241.87	297	270.77	327	299.72
238	214.00	268	242.83	298	271.73	328	300.68
239	214.96	269	243.79	299	272.70	329	301.65
240	215.92	270	244.76	300	273.66	330	302.62
241	216.88	271	245.72	301	274.62	331	303.58

REFERENCES

[1] N.A. BLACK
"Optimum merging from mass storage"
CACM 13,12 Dec. 70,p : 745-749.

[2] B.T. BENNET - W.D. FRAZER
"Approximating optimal direct access merge performance"
IFIP 1971, p : 450-453.

[3] S. COOK
"The complexity of theorem proving procedures"
Conference Record of third ACM Symposium on Theory of Computing
1970, p : 151-158.

[4] D.E. FERGUSON
"Buffer allocation in merge sorting"
CACM 14,7,July 71,p : 476-478.

[5] R.G. GALLAGER
"Information theory and reliable communication"
John Wiley and sons, 68,p : 38-49.

[6] J.W. GRAHAM - P.S. KRITZINGER
"A survey of sorting activity at Canadian Computer installation",
Canadian data systems, January 73, p : 40-41.

[7] D.A. HUFFMAN
"A method for the construction of minimum redundancy codes"
Proc. IRE, 40, 1962, p : 1098-1101.

[8] L. HYAFIL - F. PRUSKER - J. VUILLEMIN
"Design of optimal merge on direct access devices"
Proc. IFIP Congres 1974, p : 979-982.

[9] L. HYAFIL - F. PRUSKER - J. VUILLEMIN
"An efficient algorithm for computing optimal disk merge patterns"
Proc. of sixth annual ACM Symposium on Theory of Computing
Avril 1974, p: 216-229.

[10] L. HYAFIL
"Sur la complexité de trois problèmes liés au tri, aux graphes, à la recherche de l'information"
Thèse de 3ème cycle - Université de PARIS VI - Janvier 1974.

[11] L. HYAFIL - F. PRUSKER - J. VUILLEMIN
"Allocation optimale et quasi optimale de la mémoire centrale dans le tri par fusion sur disque"
Juillet 74, Rapport LABORIA N° 85.

[12] R.M. KARP
"Reducibility among combinatorial problems"
Tech. Report 3, Univ. of Calif. - BERKELEY - April 1972.

[13] D.E. KNUTH
"The art of computer programming"
Vol. 3 - Addison Wesley, 1973,p : 361-378.

[14] D.E. KNUTH
"The art of computer programming"
Vol. 3 - Addison Wesley, 1973,p : 247-360.

[15] L.G. KRAFT
"A device for quantizing, grouping and coding amplitude modulated pulses"
M.S. Thesis, Dept of E.E, MIT, Cambridge, Mass., 1949.

[17] C.F. PICARD
"Graphes et questionnaires"
Tome 2 - Gauthier-Villaurs - 1972.

[18] N. PIPPENGER - L.G. VALIANT
"Shifting graphs and their applications"
IBM Research Report - RC 5693, 1975.

[19] C.E. SHANNON
"A mathematical theory of communication"
Reprinted in book, University of Illinois Press, Urbana, 1949.

[20] M. SCHLUMBERGER - J. VUILLEMIN
"Optimal disk merge patterns"
Acta informatica 3, 1973, p: 25-35.